

Think outside the Black Box

Model-Agnostic Deep Learning with Domain Knowledge

vorgelegt von

Konstantin Kobs

Würzburg, Oktober 2023



Monografie zur Erlangung des naturwissenschaftlichen Doktorgrades der Bayerischen
Julius-Maximilians-Universität Würzburg



This document is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License (CC BY-NC-SA 4.0):
<http://creativecommons.org/licenses/by-nc-sa/4.0> This CC license does not apply to third party material (attributed to another source) in this publication.

Abstract

Deep Learning (DL) models are trained on a downstream task by feeding (potentially *preprocessed*) *input data* through a *trainable Neural Network (NN)* and updating its parameters to minimize the *loss function* between the predicted and the *desired output*. While this general framework has mainly remained unchanged over the years, the architectures of the trainable models have greatly evolved. Even though it is undoubtedly important to choose the right architecture, we argue that it is also beneficial to develop methods that address other components of the training process. We hypothesize that utilizing domain knowledge can be helpful to improve DL models in terms of performance and/or efficiency. Such *model-agnostic* methods can be applied to any existing or future architecture. Furthermore, the black box nature of DL models motivates the development of techniques to understand their inner workings. Considering the rapid advancement of DL architectures, it is again crucial to develop model-agnostic methods.

In this thesis, we explore six *principles* that incorporate domain knowledge to **understand** or **improve** models. They are applied either on the **input** or **output** side of the trainable model. Each principle is applied to at least two DL tasks, leading to task-specific *implementations*. To *understand* DL models, we propose to use **Generated Input Data** coming from a controllable generation process requiring knowledge about the data properties. This way, we can understand the model’s behavior by analyzing how it changes when one specific high-level input feature changes in the generated data. On the output side, **Gradient-Based Attribution** methods create a gradient at the end of the NN and then propagate it back to the input, indicating which low-level input features have a large influence on the model’s prediction. The resulting input features can be interpreted by humans using domain knowledge.

To *improve* the trainable model in terms of downstream performance, data and compute efficiency, or robustness to unwanted features, we explore principles that each address one of the training components besides the trainable model. **Input Masking and Augmentation** directly modifies the training *input data*, integrating knowledge about the data and its impact on the model’s output. We also explore the use of **Feature Extraction using Pretrained Multimodal Models** which can be seen as a beneficial *preprocessing* step to extract useful features. When no training data is available for the downstream task, using such features and domain knowledge expressed in other modalities can result in a Zero-Shot Learning (ZSL) setting, completely eliminating the *trainable model*. The **Weak Label Generation** principle produces new *desired outputs* using knowledge about the labels, giving either a good pretraining or even exclusive training dataset to solve the downstream task. Finally, improving and choosing the right **Loss Function** is another principle we explore in this thesis. Here, we enrich existing loss functions with knowledge about label interactions or utilize and combine multiple task-specific loss functions in a multitask setting.

We apply the principles to classification, regression, and representation tasks as well as to image and text modalities. We propose, apply, and evaluate existing and novel methods to understand and improve the model. Overall, this thesis introduces and evaluates methods that complement the development and choice of DL model architectures.

Zusammenfassung

Deep-Learning-Modelle (DL-Modelle) werden trainiert, indem potenziell *vorverarbeitete Eingangsdaten* durch ein *trainierbares Neuronales Netz (NN)* geleitet und dessen Parameter aktualisiert werden, um die *Verlustfunktion* zwischen der Vorhersage und der *gewünschten Ausgabe* zu minimieren. Während sich dieser allgemeine Ablauf kaum geändert hat, haben sich die verwendeten NN-Architekturen erheblich weiterentwickelt. Auch wenn die Wahl der Architektur für die Aufgabe zweifellos wichtig ist, schlagen wir in dieser Arbeit vor, Methoden für andere Komponenten des Trainingsprozesses zu entwickeln. Wir vermuten, dass die Verwendung von Domänenwissen hilfreich bei der Verbesserung von DL-Modellen bezüglich ihrer Leistung und/oder Effizienz sein kann. Solche *modellagnostischen* Methoden sind dann bei jeder bestehenden oder zukünftigen NN-Architektur anwendbar. Die Black-Box-Natur von DL-Modellen motiviert zudem die Entwicklung von Methoden, die zum Verständnis der Funktionsweise dieser Modelle beitragen. Angesichts der schnellen Architektur-Entwicklung ist es wichtig, modellagnostische Methoden zu entwickeln.

In dieser Arbeit untersuchen wir sechs *Prinzipien*, die Domänenwissen verwenden, um Modelle zu **verstehen** oder zu **verbessern**. Sie werden auf Trainingskomponenten im **Eingang** oder **Ausgang** des Modells angewendet. Jedes Prinzip wird dann auf mindestens zwei DL-Aufgaben angewandt, was zu aufgabenspezifischen *Implementierungen* führt. Um DL-Modelle zu *verstehen*, verwenden wir kontrolliert **generierte Eingangsdaten**, was Wissen über die Dateneigenschaften benötigt. So können wir das Verhalten des Modells verstehen, indem wir die Ausgabeänderung bei der Änderung von abstrahierten Eingabefeatures beobachten. Wir untersuchen zudem **gradienten-basierte Attribution**-Methoden, die am Ausgang des NN einen Gradienten anlegen und zur Eingabe zurückführen. Eingabefeatures mit großem Einfluss auf die Modellvorhersage können so identifiziert und von Menschen mit Domänenwissen interpretiert werden.

Um Modelle zu verbessern (in Bezug auf die Ergebnishüte, Daten- und Recheneffizienz oder Robustheit gegenüber ungewollten Eingaben), untersuchen wir Prinzipien, die jeweils eine Trainingskomponente neben dem trainierbaren Modell betreffen. Das **Maskieren und Augmentieren von Eingangsdaten** modifiziert direkt die Trainingsdaten und integriert dabei Wissen über ihren Einfluss auf die Modellausgabe. Die Verwendung von **vortrainierten multimodalen Modellen zur Featureextraktion** kann als ein Vorverarbeitungsschritt angesehen werden. Bei fehlenden Trainingsdaten können die Features und Domänenwissen in anderen Modalitäten als Zero-Shot Setting das *trainierbare Modell* gänzlich eliminieren. Das **Weak-Label-Generierungs**-Prinzip erzeugt neue *gewünschte Ausgaben* anhand von Wissen über die Labels, was zu einem Pretrainings- oder exklusiven Trainingsdatensatz führt. Schließlich ist die Verbesserung und Auswahl der **Verlustfunktion** ein weiteres untersuchtes Prinzip. Hier reichern wir bestehende Verlustfunktionen mit Wissen über Label-Interaktionen an oder kombinieren mehrere aufgabenspezifische Verlustfunktionen als Multi-Task-Ansatz.

Wir wenden die Prinzipien auf Klassifikations-, Regressions- und Repräsentationsaufgaben sowie Bild- und Textmodalitäten an. Wir stellen bestehende und neue Methoden vor, wenden sie an und evaluieren sie für das Verstehen und Verbessern von DL-Modellen, was die Entwicklung und Auswahl von DL-Modellarchitekturen ergänzt.

Acknowledgements

Working on the dissertation was a long and challenging process with many ups and downs. For every rejection and setback, there was a moment of joy and success. In the end, the result of the process is not this thesis but me, and I am thankful for the journey.

I would like to express my sincere gratitude to all the people who supported me during the long and challenging process of writing this dissertation. First and foremost, I want to thank my advisor, Andreas Hotho, for helping me to shape my research area, for his encouragement, and for invaluable feedback throughout my research journey. I am also grateful for his support in allowing me to explore the topics I am passionate about, even if they were not directly related to our projects.

I would like to thank my current and former colleagues at the Chair of Data Science for providing me with a fantastic working environment and supporting my work on the topics that interest me. Experiencing several conferences (before and after Covid-19; even being part of organizing one) and retreats with you spark some memorable experiences. My gratitude goes towards all my colleagues, but especially Michael Steininger (my dearest discussion and research partner), Jan Pfister (the guy who cheers me up with his memes), Albin Zehe (the Lord of the Harry Potter books), as well as Tobias Koopmann and Padraig Davidson (the sports guys), who supported me and became my friends during this shared journey. With your help, this experience became so much cooler.

I would also like to thank my wife Rebecca, whom I met during my time as a PhD student. I will never regret moving to Würzburg because of you, and your emotional support during countless deadlines and rejections was invaluable. I would also like to acknowledge the support of my family, including my parents and siblings, and my friends, who were always there for me and helped me get my mind off stressful work. Thank you all for being a part of my life and making this journey a memorable one.

Finally, I would like to thank all people who help to develop the software that I used in and for this dissertation. Libraries such as PyTorch, NumPy, and Pandas are indispensable for my work and tools like GitHub Copilot and ChatGPT supported me in expressing my thoughts and logical thinking. The development of these tools allows for faster research and its communication to the public by democratizing the access to resources and leveling the playing field. I hope that this mindset of accessible research and open collaboration will continue to spread.

Contents

1. Introduction	1
1.1. Research Topics and Contributions	3
1.1.1. Model-Agnostic Understanding of Deep Learning (DL) models . . .	4
1.1.2. Model-Agnostic Improvement of Deep Learning (DL) Models . . .	7
1.2. Thesis Outline	10
I. On Deep Learning and our Use-Cases	13
2. Deep Learning (DL) Foundations	15
2.1. Notation	15
2.2. Neural Network Architectures and Components	15
2.2.1. Multilayer Perceptrons (MLPs)	16
2.2.2. Activation Functions	16
2.2.3. Convolutional Neural Networks (CNNs)	17
2.2.4. Transformer	20
2.3. Pretrained Models	23
2.3.1. Word2Vec	24
2.3.2. MobileNetV2	25
2.3.3. BatchNorm Inception Network	26
2.3.4. Contrastive Language-Image Pre-Training (CLIP)	26
2.4. Loss Functions	27
2.4.1. Classification: Categorical Cross Entropy	27
2.4.2. Regression: Mean Squared Error	28
2.4.3. Representation	28
2.5. Evaluation Metrics	30
2.5.1. Classification	30
2.5.2. Regression	32
2.5.3. Representation	33
2.5.4. Ranking	35
3. Applications	37
3.1. Land Use Regression (LUR) [image; regression]	37
3.1.1. Application-Specific Related Work	37
3.1.2. Data	38
3.2. Image Aesthetics Assessment (IAA) [image; classification, regression] . . .	39
3.2.1. Application-Specific Related Work	39

3.2.2. Data	40
3.3. Age Estimation using Class Similarities [image; classification, regression]	40
3.3.1. Application-Specific Related Work	41
3.3.2. Data	41
3.4. Image Classification with Class Similarities [image; classification]	41
3.4.1. Application-Specific Related Work	42
3.4.2. Data	42
3.5. Deep Metric Learning (DML) [image; representation]	42
3.5.1. Application-Specific Related Work	43
3.5.2. Data	44
3.6. Scientific Venue Recommendation [text; classification]	45
3.6.1. Application-Specific Related Work	45
3.6.2. Data	46
3.7. Sentiment Analysis [text; classification]	47
3.7.1. Application-Specific Related Work	48
3.7.2. Data	48
4. Related Work for the Explored Principles	49
4.1. Understanding Deep Learning (DL) Models	49
4.1.1. Generated Input Data	49
4.1.2. Gradient-Based Attribution	50
4.2. Improving Deep Learning (DL) Models	51
4.2.1. Input Masking and Augmentation	51
4.2.2. Feature Extraction using Pretrained Multimodal Models	52
4.2.3. Weak Label Generation	53
4.2.4. Loss Function	53
4.2.5. Other	55
II. Understanding Deep Learning (DL) Models	59
5. Principles for Understanding Deep Learning (DL) Models	61
5.1. Input Principle: Generated Input Data	61
5.2. Output Principle: Gradient-Based Attribution	64
6. Analyzing a Land Use Regression Model using Generated Map Images	67
6.1. Methodology	69
6.1.1. Data Generation	69
6.1.2. Output Interpretation	70
6.2. Experiments	70
6.3. Results	71
6.4. Conclusion	72

7. Analyzing Deep Metric Learning Models using Generated Car Images	75
7.1. Methodology	77
7.1.1. Notation	77
7.1.2. Normalized R-Precision	78
7.1.3. Data Generation	80
7.2. Experiments	82
7.3. Results	82
7.4. Conclusion	83
8. Analyzing a Scientific Venue Recommender using Integrated Gradients	85
8.1. Methodology	87
8.2. Experiments	88
8.3. Results	89
8.4. Conclusion	91
9. Comparing Deep Metric Learning (DML) Models with a new Attribution Map Generation Method	95
9.1. Methodology	95
9.2. Experiments	97
9.3. Results	99
9.4. Conclusion	101
III. Improving Deep Learning (DL) Models	103
10. Principles for Improving Deep Learning (DL) Models	105
10.1. Input Principle: Input Masking and Augmentation	106
10.2. Input Principle: Feature Extraction using Pretrained Multimodal Models	108
10.3. Output Principle: Weak Label Generation	110
10.4. Output Principle: Loss Function	111
10.5. Hybrid: Combining Input and Output Principles	112
11. Making Deep Metric Learning (DML) Models More Robust to Background Bias using Background Augmentation	115
11.1. Methodology	117
11.1.1. Test Setting	117
11.1.2. BGAugment: Background Replacement During Training	118
11.2. Experiments	119
11.2.1. Training and Evaluation Setup	119
11.3. Results	119
11.4. Analysis	120
11.5. Conclusion	122

12. Using Contrastive Language-Image Pre-Training (CLIP) for Image Aesthetics	
Assessment (IAA)	125
12.1. Methodology	127
12.1.1. Prompting	127
12.1.2. Linear Probing	129
12.2. Experiments	130
12.3. Results	131
12.4. Analysis	134
12.5. Conclusion	135
13. Zero-shot Deep Metric Learning (DML) with Contrastive Language-Image Pre-Training (CLIP)	137
13.1. Methodology	139
13.1.1. Training	139
13.1.2. Inference	141
13.2. Experiments	141
13.2.1. Datasets and Similarity Notions	141
13.2.2. Baselines	142
13.3. Results	145
13.4. Analysis	147
13.4.1. What does InDiReCT attend to in the input?	147
13.4.2. Do other embedding sizes perform differently?	147
13.4.3. Do larger Contrastive Language-Image Pre-Training (CLIP) models improve performance?	149
13.4.4. Do more text prompts improve performance?	150
13.5. Conclusion	150
14. Improving Classification Models using Class Similarities in the Loss Function	153
14.1. Methodology	154
14.1.1. Relation between Similarity and Probability-based Matrices in Similarity Based Loss (SimLoss)	154
14.1.2. Matrix Generation	155
14.2. Experiments	156
14.2.1. Generating the Similarity Matrix	157
14.2.2. Experimental Setup	157
14.3. Results	157
14.4. Analysis	159
14.5. Conclusion	160
15. Training a Sentiment Analysis Model with Weak Labels and Input Masking	161
15.1. Methodology	162
15.1.1. Weak Label Generation using Sentiment Lexica	163
15.1.2. Input Masking of the Neural Network (NN)	166

15.2. Experiments	166
15.2.1. Baselines	167
15.3. Results	168
15.4. Analysis	169
15.4.1. Ablation Study: Emotes Matter	169
15.4.2. Comparison of Approaches: Complexity Matters	170
15.5. Conclusion	171
16. Improving Image Aesthetics Assessment (IAA) Models using Self-Supervised Pretraining with Image Augmentations, Weak Labels, and Multitask Learning	173
16.1. Methodology	175
16.1.1. Aesthetic-Aware Image Distortions	176
16.1.2. Highly Aesthetic Dataset	176
16.1.3. Self-Supervised Aesthetic-Aware Pretext Tasks	177
16.1.4. Multitask Pretraining and Finetuning	179
16.2. Experiments	180
16.2.1. Aesthetic-Aware Image Distortions	180
16.2.2. Highly Aesthetic Dataset	180
16.2.3. Self-Supervised Aesthetic-Aware Pretraining	183
16.2.4. Baselines	183
16.2.5. Finetuning Pretrained Models	184
16.2.6. Evaluation Setup	185
16.3. Results	185
16.4. Analysis	186
16.5. Conclusion	189
17. Conclusion	191
17.1. Principle Discussion	192
17.1.1. Generated Input Data	192
17.1.2. Gradient-Based Attribution	193
17.1.3. Input Masking and Augmentation	193
17.1.4. Feature Extraction using Pretrained Multimodal Models	194
17.1.5. Weak Label Generation	195
17.1.6. Loss Function	195
17.2. Outlook	196
Bibliography	199
A. Jensen-Shannon Divergence (JSD) Tables for Deep Metric Learning (DML) Attribution Method	227
B. InDiReCT Full Results	229
C. Contribution Statement	231

Acronyms

AE	Autoencoder
AI	Artificial Intelligence
AVA	Aesthetic Visual Analysis
BERT	Bidirectional Encoder Representations from Transformers
CAM	Class Activation Map
CCE	Categorical Cross Entropy
CE	Cross Entropy
CLIP	Contrastive Language-Image Pre-Training
CNN	Convolutional Neural Network
CV	Computer Vision
DL	Deep Learning
DML	Deep Metric Learning
DNN	Deep Neural Network
EMD	Earth Mover's Distance
FSA	Failed Superclass Accuracy
GAN	Generative Adversarial Network
GPT	Generative Pre-Training
IAA	Image Aesthetics Assessment
IQA	Image Quality Assessment
JSD	Jensen-Shannon Divergence
LAE	Linear Autoencoder
LAEI	London Atmospheric Emissions Inventory
LLM	Large Language Model

LR Linear Regression

LSTM Long Short-Term Memory

LUR Land Use Regression

MAE Mean Absolute Error

MAP@R Mean Average Precision at R

MaskedAE Masked Autoencoder

ML Machine Learning

MLP Multilayer Perceptron

MRR Mean Reciprocal Rank

MSE Mean Squared Error

NIMA Neural Image Assessment

NLP Natural Language Processing

NN Neural Network

NR-IQA No Reference Image Quality Assessment

OSM OpenStreetMap

PA_ IAA Personality-Assisted Image Aesthetics Assessment

PCA Principal Component Analysis

Prec@1 Precision at 1

Prec@i Precision at i

ReLU Rectified Linear Unit

RF Random Forest

RMSE Root Mean Squared Error

RNN Recurrent Neural Network

SA Superclass Accuracy

SAM Sharpness-Aware Minimization

SelfSL Self-Supervised Learning

SemiSL Semi-Supervised Learning

SGD Stochastic Gradient Descent
SimLoss Similarity Based Loss
SL Supervised Learning
SOP Stanford Online Products
SSIM Structural Similarity Index Measure
UL Unsupervised Learning
ViT Vision Transformer
WTS Where to Submit
ZSL Zero-Shot Learning

1. Introduction

Deep Learning (DL) is a very powerful paradigm that has become the defacto standard to solve many Machine Learning (ML) tasks in recent years. DL is based on Neural Networks (NNs) that are trained to learn a mapping from input data to output data.

The general DL framework is shown in Figure 1.1: On the one side, there is an *input* dataset, which can be *preprocessed*. On the other side is the corresponding *output* data. In between, there is a *trainable model* that takes the input, transforms it into more useful representations, extracts useful features to solve the task, and maps these features to the output, which is compared to the *desired output* using a *loss function* [73]. Trainable means that the parameters of the model can be optimized to better map the input to the output. The computed loss value is then backpropagated to the model's parameters to minimize the loss function.

From this basic framework, multiple DL training regimes can be derived [246]. *Supervised Learning (SL)* is the most common training regime, where the input and output data are paired, i.e., for each input example there is one desired output [145]. When no explicit desired output data is available, *Unsupervised Learning (UL)* can be used in DL, where the model learns to detect useful patterns in the input data and creates vector representations that can be clustered, for example. One possible trainable model used in this case is the Autoencoder (AE), where the desired output is the input itself, i.e., the model learns to reconstruct the input. A subregime of UL is *Self-Supervised Learning (SelfSL)*, where the output is based on contrasting the features of different views, i.e., variations, of the input data [110]. When there is a mix of labeled and unlabeled data, *Semi-Supervised Learning (SemiSL)* can be used, where the unlabeled data is used as base data to let the model learn useful features and the labeled data is then used to refine the model.

In recent years, *Zero-Shot Learning (ZSL)* has become a popular research topic, where for a downstream task, no training data is available (*zero* training examples for the

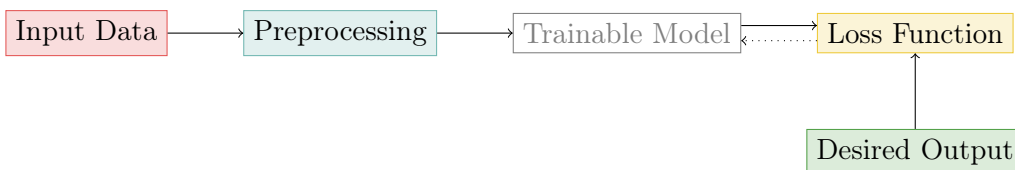


Figure 1.1.: The general training framework of DL models. The input data is preprocessed and fed through the trainable model. Its output is then compared to a desired output using a loss function. The loss is then used to update the trainable model's parameter in order to minimize the loss function (dotted arrow).

1. Introduction

downstream task) [283]. Instead, the model is trained on a different task, where the input and output data is available. A popular example are autoregressive *Large Language Models (LLMs)* such as GPT-4 [202] (based on Generative Pre-Training [29]), which learn to predict the next word/token in a sentence. Then, after trained on large text corpora, the ability to continue sentences can be used to solve a task by cleverly embedding the input text in a template such that the continuation, that the model produces, resembles the desired output. Also, the output activations of the model can be used for further processing. The training of the ZSL model fits the provided basic framework of DL, since it is a trainable model trained on inputs and outputs. The downstream task, however, keeps the weight parameters of the formerly trainable model fixed and only uses its outputs. Since no model optimization takes place, we can interpret the trained model as a preprocessing step, which, in turn, directly maps the input to the output or produces new features that are further processed.

The basic framework for training NNs has not changed much since their invention [226]. During training, input data is fed through the NN and the output is compared to the desired output using a loss function. The error, as measured by the loss function, is then propagated back through the NN by differentiating the loss function with respect to the weights of the NN. Given the computed gradients, the weights are updated using an optimization algorithm, such as Stochastic Gradient Descent (SGD). The process is repeated until performance metrics are no longer improving, indicating that the NN converges to an optimum.

NNs can represent very complicated functions, which can make it challenging to interpret the process of how they arrive at their prediction. Due to this occurring non-interpretability, they are often called “black boxes”. Given the training examples, a NN finds one function that explains the data well. Often, multiple or even infinitely many functions can explain the data equally well. This can lead to undesired behavior. For example, in order to predict that a ship is shown in an image, NNs often pay attention to water, as most ships are pictured in water [142]. When showing ships in the desert, the NN will not be able to predict that a ship is shown, since there is no water present. Since using water as a proxy to detect ships seems to work for the dataset it has been trained and evaluated on, catching such errors is difficult. Thus, it is desirable to understand what kind of input features the NN has learned to focus on in order to generate its output. Understanding this can help to catch errors and debug a model.

While the basic training framework of NNs — gradient descent based optimization of the model’s weights — has not really changed, the used model architectures are constantly evolving. Each model architecture and its variants use different assumptions to model the data, aiming to introduce so-called inductive biases into the model to achieve better performance. Inductive bias aims to narrow down the space of possible functions that can explain the data, based on domain knowledge that lead to useful assumptions about the data and the task. This can lead to the improvement of different aspects of the model: better predictive performance, higher robustness regarding certain influencing factors, fewer parameters to train, lower data requirements in terms of training dataset size, shorter training times due to faster convergence, etc. The term “improvement” is thus highly dependent on the context but means in general an outcome that is beneficial for the

task at hand. For example, Convolutional Neural Networks (CNNs) use the assumption that the input data is given in a grid-like structure that can be processed with the same model at every location [115]. This assumption is then encoded in the model architecture by using convolutional layers, which leads to fewer parameters to train, fewer training examples needed, and better performance than using fully connected models. In other situations, better performance is not always the highest priority. MobileNet [236] is a CNN architecture that is optimized for mobile devices. Thus, the highest priority is to have a small model size and faster inference times, while the performance of the model is allowed to degrade a little. In other situations, the amount of training data is constrained, which motivates other model architectures or training methods to reduce the amount of training data needed.

Given the fast development of new architectures, we argue that it is necessary to develop methods to understand the output generation process of DL models in a model-agnostic way such that they can be applied to any model architecture that is or will be used in the future. We hypothesize that domain knowledge, i.e. the insights, concepts, rules, and heuristics that experts have acquired through their experience and study of a particular domain, can be beneficial to understanding the model. In this thesis, we will focus on methods to understand what low-level and high-level input features are important for a model to form a prediction: so-called feature attribution. Here, domain knowledge can be used to compute such attributions, but also to interpret them. It is consequently desirable to also develop methods that improve NNs without dictating a specific model architecture to allow for future developments.

The **goal of this thesis** is to provide an exploration, investigation, and practical application of different methods and techniques to understand and improve NNs in a model-agnostic way by using domain knowledge. We cover two principles to *understand* trained DL models (namely Generated Input Data and Gradient-Based Attribution) and four principles to *improve* them for their downstream task during training (namely Input Masking and Augmentation, Feature Extraction using Pretrained Multimodal Models, Weak Label Generation, and Loss Function). The four principles to improve the model each touch on a different aspect of the training process besides the trainable model from Figure 1.1. We discuss the principles in more detail in the following section.

1.1. Research Topics and Contributions

In this section we give an overview over the contributions in this thesis, which are also depicted in Figure 1.2. We propose and introduce *principles* that incorporate domain knowledge to **understand** and **improve** NNs without dictating a specific trainable model. These are split into principles that take advantage of prior knowledge about the **model input** and the **model output**. From these principles, we then derive task-specific *implementations* that are applied in at least two contexts per principle. This distinction between *principles* and *implementations* can be made clear using an analogy from the software development domain. Here, a *principle* is similar to a *design pattern*, which is a general solution to a recurring problem. It features the general idea and goal of

1. Introduction

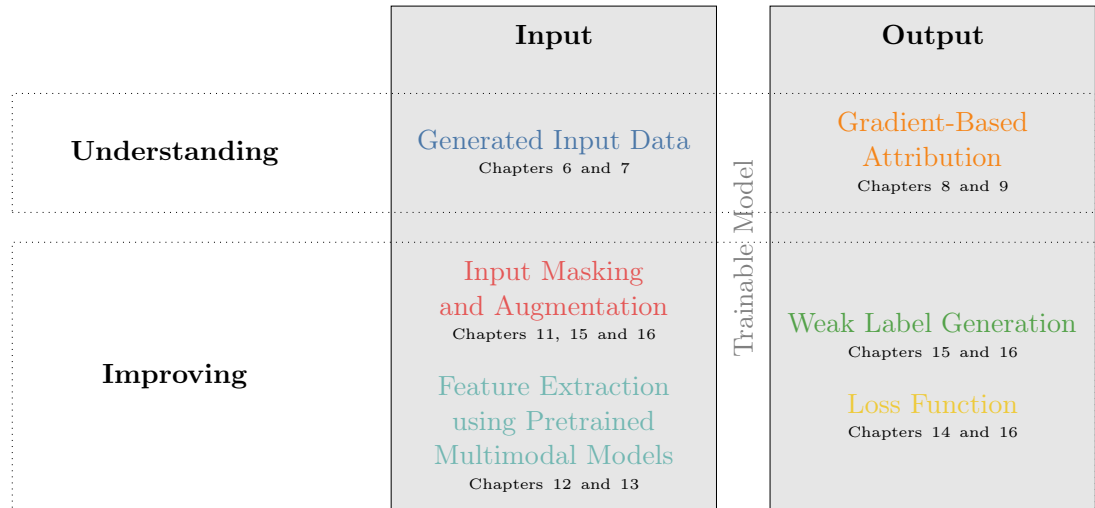


Figure 1.2.: Overview of this thesis. Shown are the six principles we explore in this thesis. Each can be categorized into principles to understand or improve DL models as well as principles that work with the input or the output of the model. All principles incorporate domain knowledge in some way. Also, the chapters in this thesis where each principle is implemented is stated.

the solution, but does not specify how to implement it. An *implementation*, in software development as in this thesis, is then a concrete application of the design pattern to a specific problem. The task at hand can be different and so is the implementation of the principle, but the general idea — the principle — remains the same. Given this distinction, we can discuss applicability of the principles and their implementations in different contexts.

The contexts in which we apply the principles are very different from each other. We cover several regression, classification, and representation tasks using image and text modalities. We do this to demonstrate the general applicability of the principles and with exemplary implementations for them in different scenarios.

Overall, we can structure our specific contributions along the understanding/improvement and input/output axes, as shown in Figure 1.2. We now briefly describe each of the investigated principles and what the contributions are. Structurally, we go from top to bottom and left to right in Figure 1.2, beginning with the understanding principles. We color-code the principles to make it easier to understand which principles are used in combination for the proposed implementations in later sections.

1.1.1. Model-Agnostic Understanding of DL models

Understanding why and how a DL model generates its outputs can be very useful to debug a model’s decisions and to prevent harmful behavior of a model, such as accidentally discriminating among certain characteristics. Such analysis processes can be used to assure quality of the model before considering deploying it. Explaining a model’s prediction can uncover failure cases, which in turn help with identifying bugs and improving the

model [217]. Also, understanding why a model outputs a prediction can lead to novel domain knowledge, since the model might uncover new patterns in the data. This can lead to new insights into how to solve tasks, which can then help humans to solve the task more efficiently. This is referred to as “microscope” Artificial Intelligence (AI) [103]. We later demonstrate such a case in Chapter 6, where we identify certain patterns in how different features are used by the model to generate its output in a Land Use Regression (LUR) task.

Research on interpreting DL models has gained traction in recent years, which is differentiated by Molnar using four criteria [185]:

1. **Intrinsic vs. post-hoc:** Intrinsic methods try to design DL models that are inherently interpretable, while post-hoc methods try to interpret a model after it has been trained.
2. **Result of the interpretation method:** Among others, some methods return feature importances, that can be displayed as values or visualized (feature attribution). Others return information about specific weights of the model or data points that are representative of or similar to the original input but lead to a different prediction (counterfactual explanations).
3. **Model-specific vs. model-agnostic:** Model-specific methods are tailored to a specific model or model class, while model-agnostic methods can be applied to any model. Model-agnostic methods are by definition post-hoc.
4. **Local vs. global:** Local methods try to explain a single prediction, while global methods try to explain the model as a whole.

Local methods are further split by Liang et al. into data-driven and model-driven methods [153]. Data-driven methods are applied at the data-level, for example by perturbing the input data and analyzing the output of the model (e.g., LIME [221] or SHAP [165]). Model-driven methods rely on model properties, such as the fact that DL models are differentiable. Analyzing single neurons or weights would also be model-driven.

Motivated by the rapid development in DL model architectures, we focus on model-agnostic (and thus post-hoc) methods that can be applied to any model architecture. Similarly, with no knowledge about the internals of the NN, we do not consider methods that understand specific parameters or layers. Instead, we focus on methods that perform feature attribution. Here, we distinguish between high-level and low-level input features. Low-level input features are the raw input data, such as pixels or tokens. High-level features are more abstract and often influence the low-level features, such as the image background or the writing style of a text. With the two principles we investigate, we mainly provide local methods, that can, however, be generalized to global methods, which we also show in the corresponding sections. Here, **Generated Input Data** is a data-driven principle that we use for high-level feature attribution and **Gradient-Based Attribution** is a model-driven principle that we mostly use for low-level feature attribution.

1. Introduction

Input

As shown in Figure 1.2, we evaluate two principles to understand the model. The first one relies on knowledge about the input data of the model to better interpret what input features are important for the model’s output.

Generated Input Data The goal of a model analysis is to understand to what kind of input features the model is sensitive to. This can be investigated by changing certain features of the input and observe the model’s output changes. In order to be able to control certain input features in isolation from others, we propose to use generated input data to analyze the model’s behavior. Here, it is necessary to be able to generate data that is similar to the real data, requiring domain knowledge. Then, the change in model output given changes in the desired input property can be analyzed. This gives insights into how certain features are processed by the model and how large the effect of the input property is on the model output.

We implement this principle for two concrete applications. First, we use this approach for a LUR model, which is trained to predict air pollution using map images from the OpenStreetMap (OSM) service. Since map images are computer-generated, it is quite simple to imitate the map generation process and produce realistic map images. The influence of map entities and their positions, such as streets, buildings, and parks, on the model output can then be analyzed. Second, we analyze and compare different Deep Metric Learning (DML) models that were trained on car images. We use a 3D rendering software to generate different car models from different angles, in different lighting conditions, etc. The change in model outputs then indicates the influence of those features. To compare the model sensitivities to different input properties, we introduce a new evaluation metric.

Output

While the Generated Input Data principle uses knowledge about the input data to understand the model, the second principle applies to the output of the model. Here, domain knowledge is then used to interpret the resulting low-level feature attributions.

Gradient-Based Attribution This principle uses gradient-based attribution maps to understand what low-level input features are important to generate the given output. For this, based on the output of the model, a gradient is created and propagated from the output to the input of the model. Large absolute input gradients for specific input features indicate that altering these features changes the output of the model considerably, thus having a large effect on the model’s prediction.

Most proposed methods for gradient-based attribution are assuming that the task is classification [248, 61, 320, 319, 305, 250]. Very often, the investigated model is an image classifier, which is trained to predict a class label for an image. As a first implementation for this principle, we use a previously proposed approach (Integrated Gradients [260]) in a text classification task. While usually, pixels are highlighted in the attribution maps,

we highlight words in the text that are important for the model’s prediction. More specifically, we use this approach on a model that predicts scientific venues based on a given title, abstract, and keywords of a publication. Using domain knowledge, the highlighted words can be interpreted to understand why a certain venue might be a fitting option for the provided publication or whether the model might have paid attention to the wrong tokens.

As a second implementation, we propose a novel gradient-based attribution method that highlights the important features for a DML model. Since DML is a representation learning task, it is not trivial to apply previously proposed methods from the classification domain to the DML domain. We use our proposed method to qualitatively and quantitatively compare different DML models trained with different loss functions.

1.1.2. Model-Agnostic Improvement of DL Models

DL models learn a function that maps from inputs to desired outputs. Since the amount of training data is finite, there are infinitely many functions that can fit the data. Thus, it is not directly known what the model is learning and if the learned function is useful. After *understanding* what input features have an influence on the model output, we can *improve* the model in task-specific ways using additional domain knowledge. Task-specific means that the performance in terms of evaluation metrics is not always the only goal. Other properties such as robustness to certain input changes, the need for fewer parameters or less training data, faster training times, or reusability of the intermediate representations can also be considered.

We propose multiple methods to improve the performance of NNs by incorporating domain knowledge and thus introducing assumptions about the input and output of the NN. Again, this is done without any architecture changes of the trainable model. We mainly explore four principles and ideas that can be mixed and matched. These principles correspond to the training setup components displayed in Figure 1.1.

Input

The following principles consider the input side of the model, i.e., the input data and preprocessing components in Figure 1.1.

Input Masking and Augmentation The first principle we explore to improve a DL model is masking and augmentation of the input. Masking specific input features, e.g., pixels or words, directly removes information for the model to use. This way, the model has no chance of using these input features for its learning process. Masking features that the model should not learn to use can make the model more robust and let it generalize better. Similarly, augmenting (parts of) input examples while using the same (or different) desired output helps to guide the model to ignore the augmentation process, making it more robust against the augmentation operation. Well-chosen masking and augmentation operations can then guide the model’s focus on the important input features, leading

1. Introduction

to better performance. Choosing these operations requires extensive domain knowledge regarding the task, data, and desired outputs.

We use input augmentation to make a DML model more robust to changing image backgrounds in order to improve item retrieval performance. For the task of Image Aesthetics Assessment (IAA), we augment aesthetically pleasing images with image transforms in order to make them less appealing. The relation between a beautiful and unaesthetic version of an image is then used to guide the model to understand what aesthetically pleasing images look like. Finally, we use input masking in a sentiment analysis setting, aiming to classify Twitch chat messages into “positive”, “neutral”, or “negative” sentiment classes. Here, we first use a weak label generator that estimates the sentiment based on sentiment words. These words are then masked with a certain probability in the input text such that a DL based model does not necessarily have access to these words. It is thus forced to not exclusively rely on the known sentiment words but to identify other words and phrases that help with estimating the sentiment.

Feature Extraction using Pretrained Multimodal Models Whenever only little or no training data is available for a downstream task, using transfer learning is a common approach. Using a pretrained model that was trained on another but related task for which enough training data exists, the model can be finetuned for the downstream task. Using such models to extract features is a form of preprocessing, since the inputs are processed to get more abstract features used as input for the trainable model. In this thesis, we primarily explore the use of pretrained multimodal models for feature extraction. More specifically, we focus on the image-text based Contrastive Language-Image Pre-Training (CLIP) model, which is trained to map images and their respective text-based descriptions to similar representations [214]. We explore how the additional modality of text can be used to improve the performance of vision models, which should require less trainable parameters and achieves faster training, while yielding comparable performance to other unimodal models. When no image training data is available, we can utilize domain knowledge expressed as text to solve the task in a so-called ZSL setting, which removes the trainable model from the usual training pipeline altogether.

In this thesis, we explore this principle by using CLIP to estimate the aesthetics of images. We show that CLIP has learned aesthetic-relevant features, which are better suited for the IAA task than commonly used ImageNet features. We use a simple Linear Regression (LR) model to predict the aesthetic of an image based on the representation of the image, which already performs similarly compared to other fully supervised methods while having only very few parameters to train. Additionally, we show that we can apply CLIP for IAA in a ZSL setting by estimating the image aesthetics just using text prompts and no explicit training data. This method does not reach the performance of fully supervised models, but requires training data or parameters, is easy to apply, and possible to use along other use cases of CLIP such as zero-shot search, since the model is frozen.

On a related note, we also introduce the advantages of pretrained multimodal models in the DML domain. Usually, DML datasets define one notion of similarity, e.g., “two car images are similar if they show the same car model”. We introduce a method that

allows for fast adaptation of this similarity notion by using CLIP. Here, CLIP is used to generate representations that contain general information about the image. We then use domain knowledge encoded as text prompts to alter these representations to focus on the desired similarity notion. This way, no training images and labels are needed, and the method can be applied very easily and with fast results.

Output

Given the training setup shown in Figure 1.1, the following principles utilize domain knowledge at the output side of the model, i.e., the desired output and the loss function components.

Weak Label Generation This principle allows for generating larger training datasets, which may be able to guide the model to learn more useful functions. Given a large set of unlabeled input data, weak labeling is the process of finding small patterns within the data that allow for assigning labels to the input data. These patterns come from domain knowledge that is expressed by simple heuristics, rules, or other models. Weakly labeled data is then easy to obtain in large amounts, but is also more noisy than hand-labeled data (which, in turn, is expensive to obtain). As a consequence, such weakly labeled datasets are usually used for pretraining DL models when not enough labeled data is available. The pretrained model then usually has learned more useful features, which can further be finetuned with the labeled data. The main goal is thus to allow for good performance while using as few hand-labeled data points as possible. We explore this principle in two settings, one using few and one using no labeled data points at all.

First, we apply this principle to sentiment analysis of Twitch.tv chat comments. Here, we express our domain knowledge by a lexicon-based approach to weakly label large amounts of chat comments. Since there is no labeled data available, we use the weakly labeled data for training the DL model exclusively. We combine this principle with the input masking principle described above. Second, we use weak labels in the estimation of image aesthetics. For this, we use a corpus of aesthetically pleasing images, scraped from a stock photo website. We then deteriorate these images' aesthetics using image style transforms such as increasing or decreasing the image's contrast or brightness (applying the Input Masking and Augmentation principle). We then use the observation that images with no style destruction should usually be more aesthetically pleasing than their transformed counterparts. This relation between image pairs is then used as weak labels to guide a pretraining process of a NN before finetuning it on a labeled dataset. The resulting model shows better performance than the NN that was exclusively trained on the labeled dataset.

Loss Function The last principle we explore in this thesis is the design of loss functions that incorporate additional task-specific knowledge. The loss function intuitively gives a measure of how well the model performs on the task. The higher the loss, the worse the model's performance. Optimizing the NN aims to minimize the loss. This is done by differentiating the loss function w.r.t. the model's parameters and adjusting the parameters

1. Introduction

in the direction of the negative gradient. A well-crafted loss function can thus guide the model to learn the desired function. Using additional knowledge about the task, the desired function can be learned easier or faster by the model. Letting the model solve multiple tasks simultaneously can also be beneficial, since the model can learn more useful features given multiple gradient signals. Then, the loss function needs to combine multiple single losses in a multitask setting. In this thesis, we investigate this principle in two settings.

In our first implementation, we propose an adaptation of the Categorical Cross Entropy (CCE) loss function, which is usually applied in classification settings. This new loss function incorporates additional knowledge about the class similarities to guide the model to predict more similar classes when not sure about the correct class. This leads to the model making less severe mistakes, which can be an improvement in applications where severe mistakes by the model are fatal.

We also combine task-specific loss functions in a multitask setting to pretrain an IAA model. Here, we generate distorted images from a set of aesthetically pleasing images. The relation between corrupted and original images is then the foundation for a loss function that lets the model learn this relationship by ranking the images. Also, we use the parameters of the image augmentation functions for a regression task and the corresponding image augmentation filters for a classification task in order to teach the model style-based aesthetic features. The combination of loss functions into a multitask loss is then done with a technique from the literature [152]. We show that using these pretraining tasks improves the performance of the IAA model on the aesthetic estimation task. We also show that not every task is beneficial for the IAA task, but that the tasks need to be carefully chosen.

1.2. Thesis Outline

This thesis is mainly split into two parts. The first part is dedicated to the analysis of NNs — Model Understanding — while the second part aims to improve NNs — Model Improvement — as displayed in Figure 1.2. In each of these parts, we describe principles to understand/improve models in a model-agnostic way, i.e., without any assumptions about the model architecture. These principles in turn are categorized into input and output principles, as shown in Figure 1.1. Concrete implementations that apply the principles in task-specific settings are given afterwards.

To make it easier to use this thesis as a reference for future exploration of these principles, we provide a summary box at the beginning of each principle section. This info box contains the main idea of the principle, its goals, the domain knowledge utilized, as well as the task and data requirements. This should give practitioners a fast and easy to understand overview of the principle, its applications, and limitations.

The rest of the thesis is structured as follows. Chapter 2 gives an overview on the mathematical notation used in this thesis and introduces the basic concepts of NNs. While the methods described in this thesis are model-agnostic, it is certainly helpful to have a basic understanding of NNs and what kind of models are used in the task-specific

implementations. We then discuss the applications we implement our principles for in Chapter 3. Chapter 4 discusses related work for each of the explored principles and puts the contributions of this thesis in context. In Part II, we first describe the principles for model understanding: Generated Input Data and Gradient-Based Attribution (Chapter 5). For Generated Input Data, we demonstrate two implementations in Chapter 6 and Chapter 7. In Chapter 8 and Chapter 9, we provide implementations for Gradient-Based Attribution.

In Part III, we describe the principles for model improvement: Input Masking and Augmentation, Feature Extraction using Pretrained Multimodal Models, Weak Label Generation, and Loss Function (Chapter 10). After the introduction of the principles, we provide concrete implementations regarding different tasks and modalities (Chapters 11 to 16). These include implementations where we combine multiple principles. Based on the results that we obtain by exploring these six principles, we finally conclude this thesis and provide an outlook on possible future work in this area in Chapter 17.

Part I.

On Deep Learning and our Use-Cases

2. Deep Learning (DL) Foundations

This chapter introduces the basic concepts and architectures of Neural Networks (NNs) that are used throughout this thesis. Even though this thesis focuses on principles that analyze and improve NNs without dictating their architecture, the concrete implementations still use well-suited NN architectures as a baseline for the improvement. We introduce the notation, different NNs architectures, pretrained models, loss functions, as well as the evaluation metrics that are used in our implementations. Please note that this chapter does not fully cover DL, but rather should be seen as an introduction with the goal of better understanding the work in this thesis. For better readability, we introduce only those concepts that are used at several points in this thesis. Infrequently used concepts are introduced when needed. For a more complete introduction to DL, see Goodfellow et al. [73].

2.1. Notation

To facilitate the understanding of mathematical expressions, we unify the notation used throughout this thesis. We mainly follow the notation of Goodfellow et al. [73]:

- **Scalars** are denoted by lowercase letters, e.g., a .
- **Vectors** are denoted by bold lowercase letters, e.g., \mathbf{x} .
- **Matrices** are denoted by bold uppercase letters, e.g., \mathbf{W} .
- **Sets** are denoted by calligraphic uppercase letters, e.g., \mathcal{A} . Natural and real numbers are denoted by \mathbb{N} and \mathbb{R} , respectively.
- **Functions** are denoted by lowercase Roman or Greek letters, e.g., f or σ . One exception is the notation of loss functions, which are denoted by the uppercase letter L and often further specified by a subscript. Also, some metric functions are denoted by uppercase letters, letter combinations, or multiple words for better readability.

2.2. Neural Network Architectures and Components

Even though this thesis tries to analyze and improve NN models without changing their architecture, it is necessary to define the basic concepts and architectures of NNs to understand the concrete implementations and applications. We thus introduce important architectures and components that are used in the rest of the thesis.

2. DL Foundations

2.2.1. Multilayer Perceptrons (MLPs)

MLPs are very basic NNs that consist of multiple fully connected (also called linear) layers [73]. Each layer is a linear transformation followed by a non-linear function. Given an input vector $\mathbf{x} \in \mathbb{R}^{n_1}$, a k -layer deep MLP is defined by

$$\mathbf{h}_1 = \sigma_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) \quad // \text{ Input layer} \quad (2.1)$$

$$\mathbf{h}_l = \sigma_l(\mathbf{W}_l\mathbf{h}_{l-1} + \mathbf{b}_l) \quad // \text{ Hidden layers} \quad (2.2)$$

$$\mathbf{o} = \sigma_k(\mathbf{W}_k\mathbf{h}_{k-1} + \mathbf{b}_k) \quad // \text{ Output layer,} \quad (2.3)$$

where all $\mathbf{W}_l \in \mathbb{R}^{n_l \times n_{l-1}}$ and $\mathbf{b}_l \in \mathbb{R}^{n_l}$ are the weights and biases of the network, respectively, and \mathbf{o} is the output of the network. The activation function σ_l ($l \in \{1, \dots, k\}$) is applied to its input vector. It is a non-linear function that is applied to the output of each linear transformation. Depending on the task, however, the final activation function σ_k can be linear. For example, in regression tasks, the output of the network is a scalar and the final activation function is the identity function $\sigma(\mathbf{x}) = \mathbf{x}$. Note: The simplest trainable regression model, the Linear Regression (LR), has only one linear transformation with no activation function, i.e., only consists of the \mathbf{h}_1 output, where σ_1 is the identity. In representation tasks, the output of the network is a vector and the final activation function is usually the identity or a normalization function. For classification, the output of the network is a vector of probabilities and the final activation function is the softmax function, which we will introduce in the next section.

Given the non-linear activation functions, MLPs are able to learn complex, non-linear functions. With linear activation functions only, the network would be a linear model. Given this simple architecture with at least one hidden layer, MLPs are theoretically universal approximators. In simple terms, this means that they can approximate any continuous function with a finite number of hidden neurons [93]. In practice, however, it is not known how many hidden neurons are needed, and it is not guaranteed that the target function is learned using optimization methods like Stochastic Gradient Descent (SGD). Thus, empirical studies have shown that more hidden layers usually help to improve the performance of MLPs [73]. The more hidden layers, the deeper the NN.

2.2.2. Activation Functions

Now that the term *activation function* is defined, we can discuss the activation functions used in this thesis. Since most activation functions are applied element-wise to their input vectors, we will define them mathematically with input scalars x , except when mentioned otherwise.

Linear The linear activation function is defined as $\sigma(x) = \alpha x$ with its derivative $\sigma'(x) = \alpha$, where α is a constant. With $\alpha = 1$, this function is the identity and thus equivalent to not using any activation function at all. Using linear activations in the hidden layers of a neural network is not recommended, as the model thus cannot learn

non-linear functions. They are thus typically used in regression tasks, where the output of the network is a scalar and thus the output should not be transformed.

Rectified Linear Unit (ReLU) The ReLU activation function is a piece-wise linear function and is defined as $\sigma(x) = \max(0, x)$. Even though it has the same values as the identify function for $x \geq 0$ and is also linear for values below zero, the change in slope at zero makes it a non-linear function. Its derivative is $\sigma'(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$, leading to easy

gradient calculations. The output of ReLU for negative input values is zero, resulting also in a derivative of zero. This can be problematic during network optimization with backpropagation. Given the chain rule $\frac{\partial f(g(x))}{\partial x} = \frac{\partial f(g(x))}{\partial g(x)} \cdot \frac{\partial g(x)}{\partial x}$, gradients for the chain of functions is computed by multiplication. If the derivative of the ReLU function is zero, the gradient of all weights before it are also zero. This is called the dying ReLU problem, which hinders parameter updates in NNs.

Leaky ReLU The leaky ReLU activation function solves the dying ReLU problem by always having a non-zero derivative, even for negative values [169]. It is defined as $\sigma(x) = \max(\alpha x, x)$, where α is a small positive constant. Thus, its derivative is $\sigma'(x) = \begin{cases} \alpha & x < 0 \\ 1 & x \geq 0 \end{cases}$, giving a small gradient to negative values. The gradient that flows through the network is thus not completely cut off.

Sigmoid The sigmoid function is defined as $\sigma(x) = \frac{1}{1+\exp(-x)}$, thus squashing the input value between zero and one. It is often used in the output layer of binary classification networks, since its output can be interpreted as a probability. Sigmoid is applied element-wise to the input vector, so multiple outputs are independent of each other. Its derivative is $\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$, which makes it easy to compute in practice, as the value for $\sigma(x)$ has already been computed in the forward pass of the network and can thus be reused.

Softmax When performing classification tasks where it is needed to output a probability distribution across the output dimensions of the network, the softmax function is used. It is defined as $\sigma(\mathbf{x}) = \frac{\exp(\mathbf{x}_i)}{\sum_{i=1}^n \exp(\mathbf{x}_i)}$, where n is the number of output dimensions. The softmax function is applied to the entire output vector, not element-wise (thus using vector notation).

While there are many more activation functions, all of them aim to solve the same problem: to introduce non-linearity to the network while being differentiable. We mainly use the described activation functions in this thesis.

2.2.3. Convolutional Neural Networks (CNNs)

The weight matrices in MLPs are multiplied by the input vectors and thus need to be at least as large as the input vector in one dimension. For large inputs, this can lead to a

2. DL Foundations

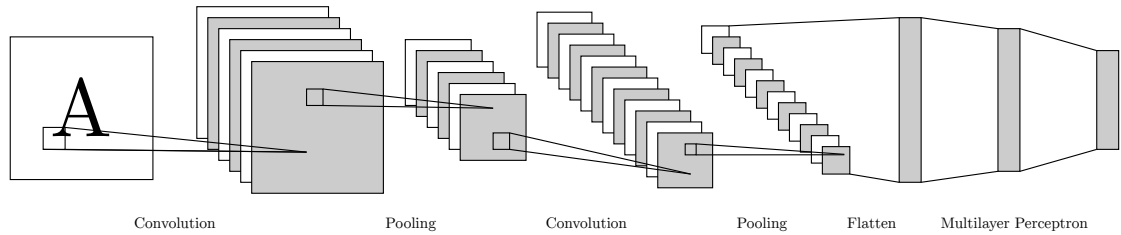


Figure 2.1.: Illustration of a CNN. The figure is inspired by a figure from LeCun et al. [144].

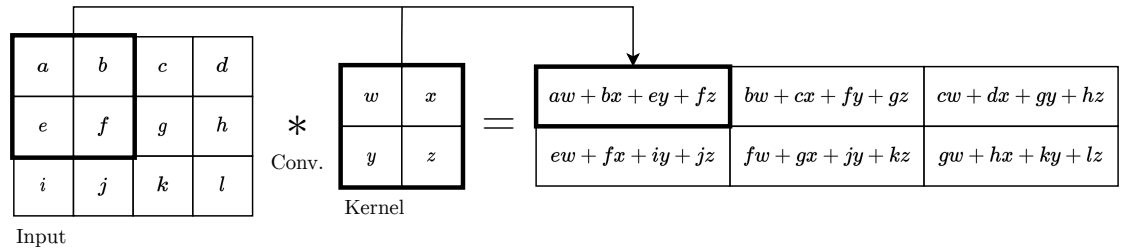


Figure 2.2.: Example 2D convolution of a matrix of size 3×4 with a kernel of size 2×2 . The values of the kernel are multiplied by the values of the input matrix while sliding the kernel over the input matrix. Figure inspired by Goodfellow et al. [73].

large number of parameters. Especially for images, the number of input pixels is very large. However, images show a lot of spatial structure, where it is usually not important if a certain pattern is present in the top left corner or the bottom right corner. Thus, it is possible to reduce the number of parameters by using the same NN weights for all regions in the image. This is the basic idea of CNNs [144, 204]. They are designed to work on spatial grids of inputs, such as images. Due to their use of convolutions, they are equivariant by design, i.e., each position in the input is treated the same way. This reduces the parameter count of the model and allows for more efficient training, since parameters are shared. In the following, we focus on the application of CNNs to image data.

Figure 2.1 shows a simple CNN architecture. The input to a CNN is a tensor of shape $w \times h \times d$, where w is the width of the image, h is the height of the image, and d is the number of channels (the *depth* of the input). In a grayscale image, $d = 1$, while in a color image encoded using RGB, $d = 3$. This image is then fed through the CNN, which consists of multiple convolutional layers, that are often accompanied by pooling layers, as well as a MLP, depending on the task at hand. In the following, we explain convolutional layers and pooling layers.

Convolutional Layer

The convolutional layer learns a set of convolutional kernels, which are applied to the input image. The convolutional kernels are also called filters and have a shape of $k_1 \times k_2 \times d$,

where k_1 and k_2 are the kernel sizes and d is the depth of the kernel. Since mostly $k_1 = k_2$, we will use k to denote the kernel size. The depth of the kernel is the same as the depth of the input image or subsequent representations. Figure 2.2 shows an example of a 2D convolution, i.e., the depth of the input and kernel is one. The kernel is slid over the input image with a certain step size (called the stride). For each position, the kernel is multiplied element-wise with the input image region and the resulting values are summed up. The resulting value is then stored in the output tensor at the current position. Since the image gets smaller with the application of a kernel, it is possible to pad the image with zeros or other appropriate values to keep the image size constant. For each kernel in a convolutional layer, a bias value is added to the output. Thus, we can interpret the application of a kernel to the input image as a linear transformation of each image region, which is similar to applying a fully connected layer to the image region. Each convolutional layer has one or multiple kernels, which are applied to the input, resulting in two dimensional matrices. These matrices are then concatenated in a third dimension to form the output of the convolutional layer. The output of a convolutional layer is thus again a three dimensional tensor of shape $w' \times h' \times d'$, where w' and h' are the width and height of the output, and d' is the number of kernels in the convolutional layer. Since the convolutional operation is a linear operation, a non-linear activation function is then applied to the output.

Pooling Layer

The pooling layer is used to reduce the spatial size of the input. Especially in image classification tasks, it is not important to keep the spatial dimension of the image. Here, it is irrelevant if the object of interest is in the top left or the bottom right corner of the image. We can thus reduce the spatial dimension of the intermediate representations by applying a pooling operation. Also, it reduces the computational complexity in the network, since the input to the next convolutional layer is smaller. Pooling combines multiple spatial pixels in the activation map into one pixel. The most common pooling operation is max pooling, which takes the maximum value of the input pixels in a region and stores it in the output. For example, using a 2×2 pooling kernel with a stride of two, the input image is divided into non-overlapping 2×2 regions and the maximum value of each region is stored in the output. This effectively results in a four times smaller representation, since four pixels are combined into one. Other possible pooling operations include average pooling, which takes the average of the input pixels in a region, and L2 pooling, which takes the L2 norm of the input pixels in a region.

After each convolutional layer, a pooling layer is typically applied. The resulting smaller representation is then fed into the next convolutional layer. After the last convolutional or pooling layer, the resulting representation can be flattened to a one-dimensional vector that is fed into a MLP to generate an output vector for the CNN.

2. DL Foundations

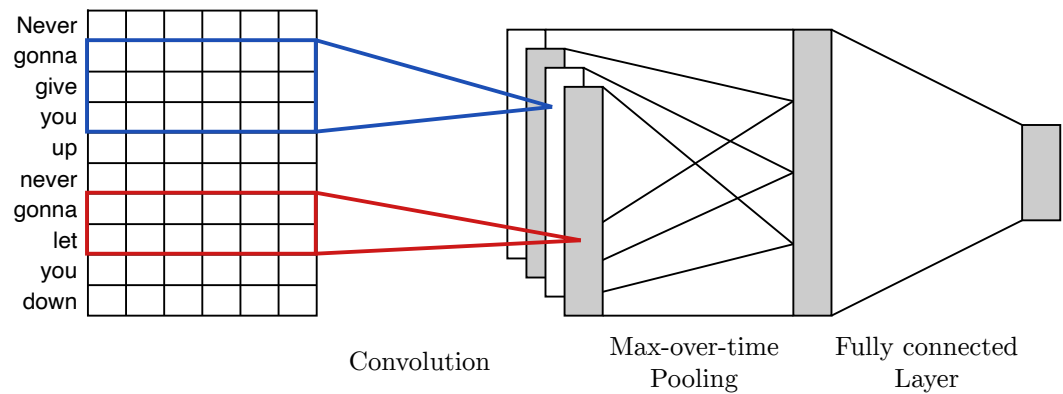


Figure 2.3.: TextCNN architecture. Figure inspired by Kim [126].

CNNs for Text: TextCNN

While the previous section described CNNs for image data as an example, CNNs can also be used for text data. Kim introduced a CNN architecture for text classification, called TextCNN [126] (shown in Figure 2.3). The input to the model is a sequence of words, which are represented as word embeddings. These embeddings are concatenated to form a matrix with a width of the embedding size and a height of the sequence length. This matrix is then fed into one convolutional layer with a set of differently sized kernels. While all kernels have the same width to cover the complete embedding size, their height differs. In the original paper, the heights are 3, 4, and 5 with 100 kernels per size in the convolutional layer.

For each kernel, the output of the convolutional layer is a vector that is pooled with a max-over-time pooling operation, which takes the maximum over all vector values along the sentence length. The pooled outputs for all kernels are concatenated into one vector and fed into a fully connected layer that generates the output of the model using a softmax activation function. Even though this model is not the state-of-the-art anymore, we use this architecture for text classification in some of our implementations, since it is fast, efficient, and performs quite well.

2.2.4. Transformer

Proposed as an alternative to CNNs for texts, the Transformer is a NN architecture that works on sequences of inputs [271]. Instead of using kernels as in CNNs, the Transformer uses self-attention, which is a mechanism to compute a representation of a sequence based on the representations of the individual elements in the sequence. In this explanation, we will focus on the main building blocks of the Transformer, which are self-attention and the multi-head attention mechanism (a schematic depiction of the Transformer architecture is shown in Figure 2.4).

The input to a Transformer is a sequence of inputs. Here, we will focus on text as the

2.2. Neural Network Architectures and Components

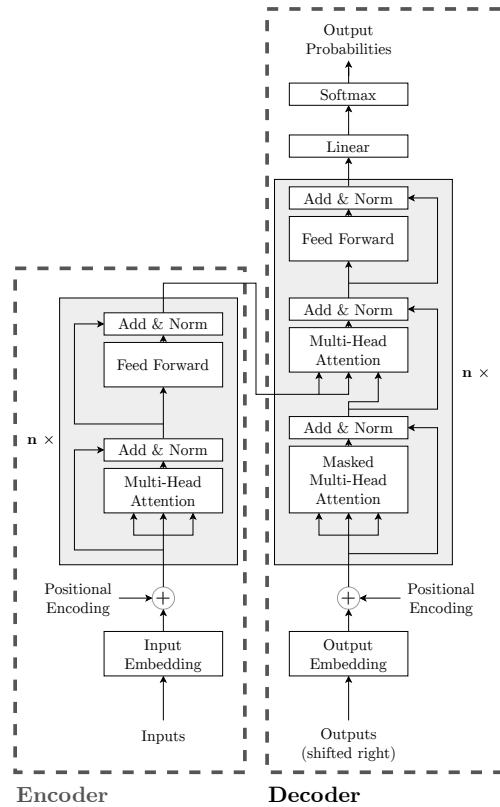


Figure 2.4.: Transformer architecture. Figure inspired by Vaswani et al. [271].

2. DL Foundations

input data. While originally, the Transformer is a sequence-to-sequence model in order to perform machine translation, using only its encoder was shown to be very effective for other text related tasks [48]. In this thesis, only the encoder is used, so we focus on this component of the model.

Given the input data, which is a sequence of words, the Transformer first embeds each word (also called *token*, as sometimes not whole words but subwords are used) into a vector. Since all operations in a Transformer are position-agnostic, the order of the words needs to be encoded explicitly into the sequence. For this, positional embeddings, i.e., embeddings encoding the position of the word in the sequence, are added to the word embeddings. In the original paper, the positional embeddings are defined using the following formula:

$$\text{PE}(\textit{pos}, 2i) = \sin(\textit{pos}/10000^{2i/d_{\text{model}}}) \quad (2.4)$$

$$\text{PE}(\textit{pos}, 2i + 1) = \cos(\textit{pos}/10000^{2i/d_{\text{model}}}), \quad (2.5)$$

where \textit{pos} is the position of the word in the sequence, i is the index of the embedding dimension, and d_{model} is the embedding size. The position-enriched input sequence is then fed into the Transformer.

The Transformer consists of n Transformer layers, each of them consisting of two sublayers. The first sublayer is a multi-head attention layer, which is surrounded by a residual connection and a layer normalization. The second sublayer is a MLP that is applied to each sequence position independently. It is also surrounded by a residual connection and a layer normalization. Residual connections add the initial input to the sublayer output, letting the layer only learn to output the necessary change from the input, which stabilizes training [86]. Layer Normalization normalizes the mean and standard deviation of each input sample to a learned mean and standard deviation, which stabilizes training and results in faster training speeds [12]. We now give a brief introduction to the multi-head attention layer. For this, it is necessary to understand the self-attention mechanism, which will be covered first.

Self-Attention

Self-attention is a mechanism that allows the model to focus on certain parts of its own output. Given the concatenated sequence of token embeddings \mathbf{X} that has a width of the embedding size and a height of the sequence length, each token embedding is linearly transformed into a query, key, and value vector, resulting in three matrices \mathbf{Q} , \mathbf{K} , and \mathbf{V} , respectively. The (scaled dot-product) attention mechanism as introduced in the original paper is then defined as

$$\mathbf{A} = \text{softmax} \left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} \right) \mathbf{V}, \quad (2.6)$$

where \mathbf{A} is the attention matrix, d_k is the dimensionality of the key vectors, and softmax is the softmax function, which is applied row-wise to the matrix. The dot products are scaled by the square root of the dimensionality of the key vectors in order to prevent large

dot products that can come from higher vector dimensionalities. The large dot products can lead to small vanishing gradients in the softmax function.

The attention matrix \mathbf{A} is the new output of this self-attention layer and has the same height as the input (the sequence length). Its width depends on the dimensionality choice of the value matrix \mathbf{V} . This is the intermediate output of a Transformer layer.

Multi-Head Attention

Multi-head attention combines multiple self-attention layers. For this, the self-attention is applied multiple times independently to the input sequence. The resulting new outputs are concatenated along the embedding dimension and linearly transformed to a new output. This mechanism allows the model to learn multiple different attention mechanisms, each focusing on different aspects of the input.

The new output has the same height as the input, i.e., both share the same sequence length. This output is a new representation of the input, which can be used as the input for downstream tasks. For some tasks, special input tokens can be defined to capture task-specific information. For example, for classification, the input is prepended with a special token (called the [CLS], i.e., class token). The representation of this token is then used as the input for a classification layer. Other strategies include using the representation of the last token in the sequence or the mean of all tokens in the sequence, which can be useful for representing the input [218].

Transformers for Images: Vision Transformers (ViTs)

While the Transformer was originally introduced for text, it can also be used for images [50]. Here, instead of token embeddings, the input is a sequence of non-overlapping image patches, which are flattened to a vector and linearly transformed to the desired input dimensionality. The resulting sequence is then fed into the Transformer encoder. A schematic overview can be found in Figure 2.5.

The sizes of ViTs are given by the number of Transformer encoder layers, the number of attention heads, the hidden size of the used MLPs, and the dimensionality of the token embeddings. In the original paper, three different sizes are introduced, which are denoted as ViT-B/16, ViT-L/16, and ViT-H/14. The letter indicates Base, Large, and Huge, respectively. The number indicates the size of each image patch. While ViT-B/16 and ViT-L/16 use image patches of size 16 by 16 pixels, ViT-H/14 uses image patches of size 14 by 14 pixels, resulting in more patches and thus a longer input sequence. Hence, the ViT-H/14 model is computationally more expensive than the other two models.

2.3. Pretrained Models

In our experiments, we use several pretrained models that are publicly available. In this section, we give a brief overview of these models.

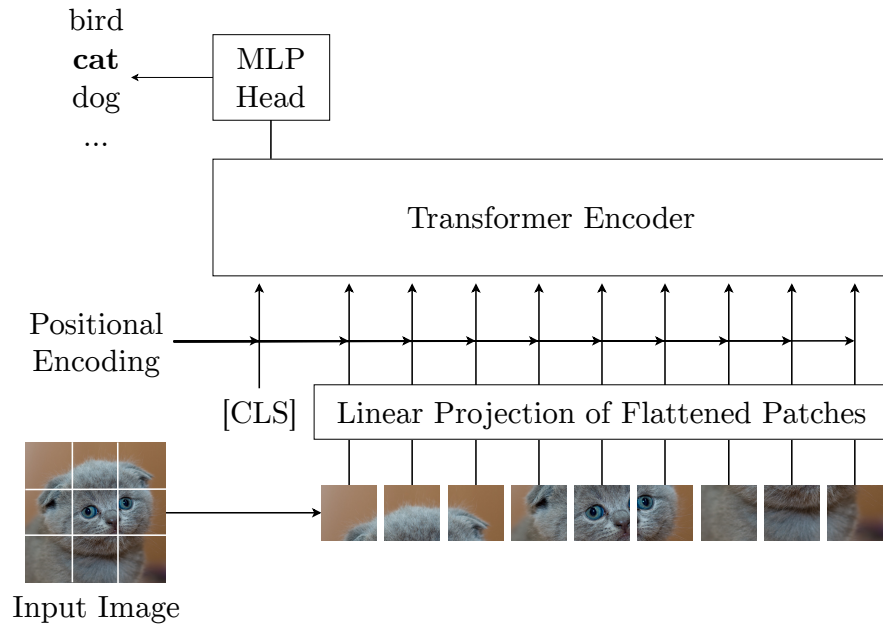


Figure 2.5.: The ViT architecture. Figure inspired by Dosovitskiy et al. [50].

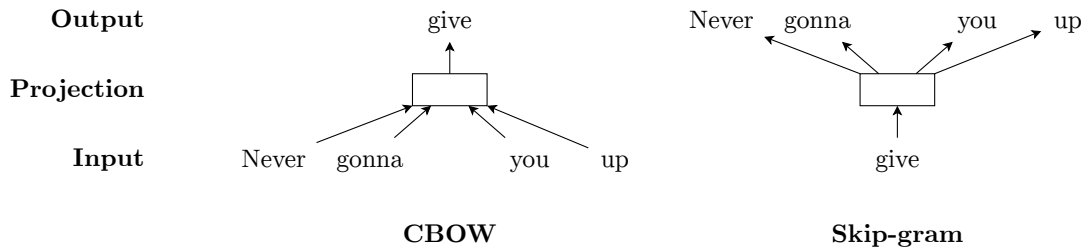


Figure 2.6.: The Word2Vec architecture. Figure inspired by Mikolov et al. [178].

2.3.1. Word2Vec

Word2Vec uses a two-layer NN to learn word embeddings from text corpora, such that two words that share the same context, i.e., the same words surrounding them, are likely to have similar meanings and should thus have similar embeddings [178, 179, 180].

There are two basic models, the continuous bag-of-words (CBOW) and the skip-gram model, which are shown in Figure 2.6. The CBOW model predicts the current word from the surrounding context words, while the skip-gram model predicts the surrounding context words from the current word. While the diagram suggests simultaneous prediction of all context words for the skip-gram model, one context word is sampled at random for each training step. As input to the network, the center word of the chosen text window is one-hot encoded and fed into the first layer. The first layer acts basically as a lookup table, which maps the one-hot encoded word to its embedding, which has much fewer dimensions than the one-hot encoded vector. This embedding is then fed during training

into the second layer, which is a linear layer followed by a softmax function. It predicts the probability of each word in the vocabulary to be the context word. The model is trained using the Categorical Cross Entropy (CCE) loss function until convergence. Afterwards, the weights of the first layer are used as the word embeddings' lookup table.

Word embeddings have been shown to be very useful, since they can be trained on large amount of unlabeled texts and capture the semantic meaning of words to some extent. As a result, it is possible to perform “semantic calculations” with the word vectors. For example, a famous example is that the word embedding of the word “king” minus the word embedding of the word “man” plus the word embedding of the word “woman” is very similar to the word embedding of the word “queen”. This behavior usually emerges when trained on large text corpora. The original authors, Mikolov et al., provide pretrained Word2Vec models [178]. These 300-dimensional embeddings for three million words and phrases were trained on news texts of approximately 100 billion words. We use this pretrained model to estimate the semantic similarity between dataset classes in Chapter 14. In other chapters, we utilize the Word2Vec technique to train our own word embeddings.

2.3.2. MobileNetV2

As the name suggests, MobileNetV2 (which is the successor to MobileNet [96]) is a CNN architecture designed for image applications on mobile devices [236]. This means that the model needs to have a low number of parameters to fit on mobile processors (3.4 million parameters in the standard setting), needs to require as minimal memory as possible during inference (maximal 400 kilobytes of memory), and should be computationally efficient by using as few operations as possible (300 million multiply-adds). It was mainly developed for image classification on ImageNet [46] but can also be used as a backbone for object detection and semantic segmentation.

Since the underlying architectural details of MobileNetV2 are not relevant for this thesis, we will only give a brief explanation. The main idea is to replace the convolutional layers with operations that need fewer parameters and are faster to compute. In the first version of MobileNet [96], the convolutional layers were replaced by depthwise separable convolutions. This means that the convolutional layer was split into two layers, a depthwise convolution and a pointwise convolution. The depthwise convolution is a convolutional layer that is applied to each channel of the input independently (in contrast to the full convolution, which takes all channels into account). The output of this layer has again the same number of channels, since they are not interacting. The pointwise convolution is a convolutional layer with a kernel size of 1 by 1, mixing the newly created feature maps from the depthwise convolution. MobileNetV2 introduces a bottleneck by letting the pointwise convolution have a smaller number of output channels than the input channels, followed by another pointwise convolution to inflate the number of channels again. While residual connections are often applied to the tensors with a large channel count, MobileNetV2 applies them to the bottleneck tensors, given the intuition that the information that is fed through the network is already fully captured in the bottleneck tensor. This saves add operations and requires less memory, since smaller intermediate

2. DL Foundations

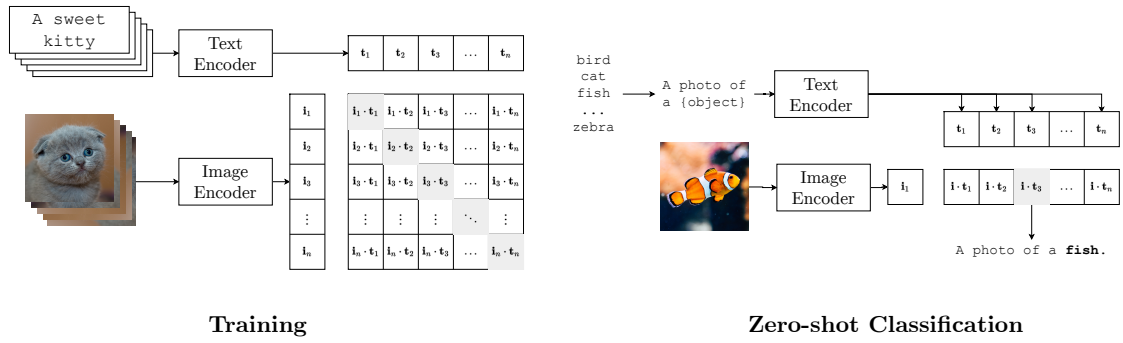


Figure 2.7.: Overview of the training process of CLIP and how to classify in a zero-shot fashion. Each batch of images and their corresponding text descriptions are embedded using the corresponding encoders and the cosine similarity between correct pairs is maximized. When classifying, the possible class names are put into a text template and embedded. The most similar prompt to the image embedding is the prediction. Figure inspired by Radford et al. [214].

result tensors must be saved for the residual connection.

In our Image Aesthetics Assessment (IAA) experiments in Chapter 16, we use the MobileNetV2 model pretrained on ImageNet [46] as a base model for further training.

2.3.3. BatchNorm Inception Network

The Inception Network [262] is a CNN architecture that was developed to improve the performance of the ImageNet Challenge [46]. It employs the so-called Inception block, which is a set of parallelly executed convolutional layers with different kernel sizes (1, 3, and 5) as well as a 3 by 3 max pooling operation to capture different spatial information of the same input. The input is padded to ensure that the outputs have the same spatial dimensions as the input. This way, the outputs of all convolutional layers in one block can be concatenated. The authors also propose to put convolutions with kernel size 1 in front of the other convolutions to reduce the number of channels beforehand and can thus reduce the number of required multiplications. The BatchNorm Inception Network [263] is a variant of the Inception Network that uses batch normalization [108], which normalizes the mean and standard deviation of the features for each input batch, leading to better performance on ImageNet than the model without batch normalization. We use this model that was pretrained on ImageNet as a base for some of our Deep Metric Learning (DML) experiments, since it is a popular architecture in the literature.

2.3.4. Contrastive Language-Image Pre-Training (CLIP)

CLIP is a multimodal model, embedding images and texts into the same vector space [214]. CLIP consists of an image encoder and a text encoder. In the best performing setting from the original paper, both models are Transformers (i.e., vision and text, respectively). CLIP is trained on a corpus of around 400 million web images and their corresponding text descriptions. Images and texts are mapped to a 512-dimensional vector space using

the image and text encoder, respectively. Both encoders are trained to maximize cosine similarity between the embeddings of corresponding images and texts while minimizing similarity between mismatching image-text pairs. An overview of CLIP is depicted in Figure 2.7.

Cosine similarity between two vectors \mathbf{i} and \mathbf{t} is the cosine of the angle between both vectors and is defined as

$$sim_{cos}(\mathbf{i}, \mathbf{t}) = \frac{\mathbf{i} \cdot \mathbf{t}}{\|\mathbf{i}\| \|\mathbf{t}\|}, \quad (2.7)$$

where $\|\cdot\|$ denotes the Euclidean norm, i.e., the length of the vector. Thus, the cosine similarity is the dot product of both vectors normalized to a length of one.

Given a batch of images and their corresponding texts, the cosine similarity is computed between all image-text combinations in this batch. A softmax function is applied to the similarities comparing one text to all images and one image to all texts. On these distributions, the CCE is applied to boost the correct pair of image and text and make the cosine similarity smaller for all other pairs.

Due to this training objective, CLIP is able to classify images based on natural language prompts: Given an image and a set of possible text descriptions, the image and all texts are encoded using CLIP’s image and text encoder, respectively. Then, the cosine similarities between each text embedding and the image are computed. The text prompt with the highest similarity is chosen as the predicted class. This procedure is also shown in Figure 2.7. CLIP shows very good performance on many datasets in a kind of “zero-shot” setting; no training example from the target dataset is used and only suitable natural language prompts are engineered to identify the correct class. We use CLIP for one of our IAA experiments in Chapter 12 as well as in one DML implementation in Chapter 13.

2.4. Loss Functions

Given a model architecture, the input of the model is transformed to an output. This output is compared to the target label using a loss function. Depending on the task, there are different loss functions that can be used. In this thesis, there are one loss function for classification, one for regression, and five loss functions for representation tasks that are used multiple times and thus are introduced in this section.

2.4.1. Classification: Categorical Cross Entropy

The CCE is the most common classification loss function. Its more general version, Cross Entropy (CE), is used to compare two probability distributions. Given a batch of examples \mathcal{B} for a classification task with possible classes \mathcal{C} , the model outputs probability distributions $\hat{\mathbf{p}}_1, \dots, \hat{\mathbf{p}}_{|\mathcal{B}|}$ estimating the true distributions $\mathbf{p}_1, \dots, \mathbf{p}_{|\mathcal{B}|}$. The CE loss is then defined as

$$L_{CE} = -\frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} \sum_{c \in \mathcal{C}} \mathbf{p}_c \log(\hat{\mathbf{p}}_c). \quad (2.8)$$

2. DL Foundations

In multi-class classification settings, the target distribution is a one-hot encoded vector, since only one class is correct. In this case, the estimated probability of the correct class should be maximized. Given the target classes for the batch $y_1, \dots, y_{|\mathcal{B}|}$, the loss is then called CCE and simplifies to

$$L_{\text{CCE}} = -\frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} \log(\hat{\mathbf{p}}_i[y_i]), \quad (2.9)$$

where $\hat{\mathbf{p}}_i[y_i]$ is the estimated probability for the target class c_i . The closer the model output matches the target one-hot encoded vector, the closer the loss function gets to zero, its minimum.

2.4.2. Regression: Mean Squared Error

MSE is usually used in regression tasks. For a given batch of examples \mathcal{B} , the model produces number estimates $\hat{y}_1, \dots, \hat{y}_{|\mathcal{B}|}$. The true labels are $y_1, \dots, y_{|\mathcal{B}|}$. Then, the MSE loss function is defined as

$$\text{MSE} = \frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} (y_i - \hat{y}_i)^2. \quad (2.10)$$

Larger deviations from the true label are punished more due to the squared term. The minimum of this function is zero, i.e., when the predictions are the same as the target values.

2.4.3. Representation

We now turn to a set of common DML loss functions, where inputs are transformed to vectors by the model.

Contrastive Loss In the case of the Contrastive Loss [80], an example in the batch \mathcal{B} consists of a pair of inputs along with a binary label y , whether these two inputs are deemed similar ($y = 0$) or not ($y = 1$). The model converts both inputs to vectors \mathbf{x}_1 and \mathbf{x}_2 . Also given is the distance function d that computes the distance between two input vectors. The Contrastive Loss function is then defined as

$$L_{\text{Contrastive}} = \frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} (1 - y_i) \frac{1}{2} d(\mathbf{x}_{1_i}, \mathbf{x}_{2_i})^2 + y_i \frac{1}{2} \max(0, m - d(\mathbf{x}_{1_i}, \mathbf{x}_{2_i}))^2, \quad (2.11)$$

where m is the margin, a hyperparameter that ensures that dissimilar vectors are not pushed away indefinitely. Consequentially, dissimilar vectors need to have a distance of at least m in order to count as zero into the loss function.

Triplet Loss Similar to the Contrastive Loss is the Triplet Loss, which — instead of pairs — uses triplets of inputs [288]. The input are converted to the vectors \mathbf{a} (the anchor),

\mathbf{p} (the positive), and \mathbf{n} (the negative). The inputs are chosen such that \mathbf{a} and \mathbf{p} are considered to be similar, while \mathbf{a} and \mathbf{n} are dissimilar. The idea of the loss is that \mathbf{n} should be farther away from \mathbf{a} than \mathbf{p} . Specifically, their distance should be larger by m than the similar items, where m is a chosen hyperparameter. With distance function d and the batch \mathcal{B} , the loss function is defined as

$$L_{\text{Triplet}} = \frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} \max(0, d(\mathbf{a}, \mathbf{p}) - d(\mathbf{a}, \mathbf{n}) + m). \quad (2.12)$$

Multi Similarity Loss While the Contrastive and Triplet Losses are using only one or two other items to compute their loss for each example, Multi Similarity Loss uses a set of positive (\mathcal{P}_i) and a set of negative (\mathcal{N}_i) examples for each example vector \mathbf{x}_i in the batch \mathcal{B} [284]. These sets are found by selecting the positives (examples that are deemed similar) with the lowest cosine similarity sim_{cos} to the example and the negatives (examples that are deemed dissimilar) with the highest cosine similarity to the example from the batch, respectively. Then, the loss is defined as

$$L_{\text{MultiSimilarity}} = \frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} \left[\frac{1}{\alpha} \log \left(1 + \sum_{k \in \mathcal{P}_i} \exp(-\alpha(sim_{cos}(\mathbf{x}_i, \mathbf{x}_k) - \lambda)) \right) \right. \quad (2.13)$$

$$\left. + \frac{1}{\beta} \log \left(1 + \sum_{k \in \mathcal{N}_i} \exp(\beta(sim_{cos}(\mathbf{x}_i, \mathbf{x}_k) - \lambda)) \right) \right], \quad (2.14)$$

where α and β are hyperparameters to weight the influence of the positive and negative sets, respectively, and λ is a hyperparameter that offsets the exponent in the loss.

Normalized Softmax Loss The idea for the Normalized Softmax Loss is based on the CCE loss function (see Section 2.4.1), which is usually used for classification tasks [159, 276, 312]. Thus, the Normalized Softmax Loss is a so-called classification based loss. The CCE loss receives a probability distribution over all classes as input, that is usually obtained by feeding the model’s output vector through a softmax function. In the case of vector representations, each item belongs to a class $c_1, \dots, c_{|\mathcal{C}|} \in \mathcal{C}$, so items with the same class should be deemed similar and thus more similar. In the Normalized Softmax Loss, each class has a representation vector $\mathbf{p}_1, \dots, \mathbf{p}_{|\mathcal{C}|}$ — a so-called proxy — that is trained alongside the model parameters. For a batch \mathcal{B} , the model outputs item vector representations $\mathbf{x}_1, \dots, \mathbf{x}_{|\mathcal{B}|}$. We also know the class index $y_1, \dots, y_{|\mathcal{B}|}$ for each example in the batch.

Normalized Softmax Loss then computes the Cosine Similarities sim_{cos} between each vector representations and the class proxies by normalizing all vectors to unit-length and computing the dot product. Then, these similarities are fed through a softmax function to obtain a probability distribution (with an optional scaling factor s called the temperature).

2. DL Foundations

Given the correct class index and this probability distribution, the CCE can be computed to maximize the similarity between the vector and its corresponding class proxy while minimizing the similarity to all other class proxies. All together, the loss function is defined as

$$L_{\text{NormalizedSoftmax}} = -\frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} \log \left(\frac{\exp(\text{sim}_{\text{cos}}(x_i, p_{y_i})/s)}{\sum_{c \in \mathcal{C}} \exp(\text{sim}_{\text{cos}}(x_i, p_c)/s)} \right). \quad (2.15)$$

ArcFace Loss Finally, ArcFace Loss is also a classification based loss function [47]. It is similar to Normalized Softmax Loss, but adds a margin penalty to the loss function for the correct class. More specifically, the margin is added to the cosine similarity calculation: Usually, cosine similarity is the cosine of the angle between the two vectors. An angle of zero results in a similarity of one. ArcFace adds the margin m to the angle between the example’s vector and its class proxy before computing the cosine, resulting usually in lower similarities for this pair. Since the loss then optimizes for this pair’s similarity to be higher than the similarity of the example to other class proxies, the similarities to other proxies needs to be lower. This results in a higher similarity between examples from the same class, while lowering the similarity between examples of other classes.

Mathematically, we define the margin-enhanced cosine similarity to be $\text{sim}_{\text{cos}}^m(\mathbf{x}_i, \mathbf{p}_j) = \cos(\theta_{ij} + m)$, where θ_{ij} is the angle between example vector \mathbf{x}_i and proxy \mathbf{p}_j . Then, with the same variables defined for the Normalized Softmax Loss, the ArcFace Loss is defined as

$$L_{\text{ArcFace}} = -\frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} \log \left(\frac{\exp(\sigma \cdot \text{sim}_{\text{cos}}^m(x_i, p_{y_i}))}{\exp(\sigma \cdot \text{sim}_{\text{cos}}^m(x_i, p_{y_i})) + \sum_{c \in \mathcal{C} \setminus y_i} \exp(s \cdot \text{sim}_{\text{cos}}(x_i, p_c))} \right), \quad (2.16)$$

where s is an exponent scaling factor, which is a loss hyperparameter.

2.5. Evaluation Metrics

In our implementations, we work on different tasks and modalities and try to understand and improve DL models regarding different aspects. One of these aspects is the model performance. To measure the overall model performance and potential performance improvements, we make use of different evaluation metrics. These are described in the following sections. We structure them into classification, regression, representation, and ranking metrics to show the broad task range of our experiments.

2.5.1. Classification

For classification tasks, there is a set of possible classes \mathcal{C} . E.g. in a binary setting, there are two classes, while ImageNet images have 1000 possible classes [46]. The goal is to predict the correct class for a given input. Given the test dataset \mathcal{T} with the target classes

$y_1, \dots, y_{|\mathcal{T}|}$ for each data point and the predicted classes $\hat{y}_1, \dots, \hat{y}_{|\mathcal{T}|}$, we can compute the following numbers for each class $c \in \mathcal{C}$:

- True Positives (TP_c): $\sum_{i=1}^{|\mathcal{T}|} \mathbb{1}(y_i = c, \hat{y}_i = c)$
- False Positives (FP_c): $\sum_{i=1}^{|\mathcal{T}|} \mathbb{1}(y_i \neq c, \hat{y}_i = c)$
- False Negatives (FN_c): $\sum_{i=1}^{|\mathcal{T}|} \mathbb{1}(y_i = c, \hat{y}_i \neq c)$
- True Negatives (TN_c): $\sum_{i=1}^{|\mathcal{T}|} \mathbb{1}(y_i \neq c, \hat{y}_i \neq c)$

From this, we can compute the following metrics:

Accuracy is the fraction of correctly predicted examples:

$$\text{Accuracy} = \frac{1}{|\mathcal{T}|} \sum_{c \in \mathcal{C}} \text{TP}_c. \quad (2.17)$$

Recall for a class c is defined as:

$$\text{Recall}_c = \frac{\text{TP}_c}{\text{TP}_c + \text{FN}_c}. \quad (2.18)$$

We can then combine the recall values for each class using different strategies. In our experiments, we use the Macro Recall, which is the unweighted average over all classes:

$$\text{Macro Recall} = \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} \text{Recall}_c. \quad (2.19)$$

Hence, it does not take class imbalance into account.

F1-Score for a class label c is defined as:

$$\text{F1}_c = \frac{\text{TP}_c}{\text{TP}_c + \frac{1}{2} \cdot (\text{FP}_c + \text{FN}_c)}. \quad (2.20)$$

Again, we use the Macro F1-Score in our experiments, which is the unweighted average of the class F1-Scores:

$$\text{Macro F1} = \frac{1}{|\mathcal{C}|} \cdot \sum_{c \in \mathcal{C}} \text{F1}_c. \quad (2.21)$$

Superclass Accuracy (SA) In addition to the standard metrics, we define and use task-specific evaluation metrics. When measuring the similarity between class predictions and their target classes in Chapter 14, we introduce the SA metric. Given the predictions of the model, we introduce a function $\text{sup} : \mathcal{C} \rightarrow \mathcal{S}$ that maps a class label to its superclass (which captures similar classes, so $|\mathcal{S}| < |\mathcal{C}|$; e.g., the classes “rose” and “violet” both have

2. DL Foundations

the superclass “flower”). We can then compute the classification numbers TP_s , FP_s , FN_s , and TN_s for each superclass $s \in \mathcal{S}$. SA is then defined as:

$$SA = \frac{1}{|\mathcal{T}|} \sum_{s \in \mathcal{S}} TP_s. \quad (2.22)$$

This value is always at least as high as Accuracy, as a correctly assigned class implies the correct superclass.

Failed Superclass Accuracy (FSA) is another custom metric for the class similarity setting. FSA only observes misclassified examples, thus measuring the similarity of misclassifications compared to the target class. A high FSA means that if the model predicts the wrong class, the predicted class is at least similar to the correct class:

$$FSA = \frac{\sum_{s \in \mathcal{S}} TP_s - \sum_{c \in \mathcal{C}} TP_c}{|\mathcal{T}| - \sum_{c \in \mathcal{C}} TP_c}. \quad (2.23)$$

2.5.2. Regression

For the test dataset \mathcal{T} , we also get a set of predictions $\hat{y}_1, \dots, \hat{y}_{|\mathcal{T}|}$ and a set of ground truth labels $y_1, \dots, y_{|\mathcal{T}|}$. However, in regression tasks, these values are continuous floating point numbers and not class labels. The goal is thus to measure the difference between the predictions and the ground truth labels or their correlations. We do this in our experiments with the following metrics:

Mean Absolute Error (MAE) is defined as the mean of the absolute differences between the predictions and the ground truth labels:

$$MAE = \frac{1}{|\mathcal{T}|} \sum_{i=1}^{|\mathcal{T}|} |y_i - \hat{y}_i|. \quad (2.24)$$

MSE is defined as the mean of the squared differences between the predictions and the ground truth labels, thus giving more weight to large errors:

$$MSE = \frac{1}{|\mathcal{T}|} \sum_{i=1}^{|\mathcal{T}|} (y_i - \hat{y}_i)^2. \quad (2.25)$$

Root Mean Squared Error (RMSE) is the square root of the MSE and measures the average error in the same units as the ground truth labels:

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}. \quad (2.26)$$

Coefficient of Determination/ R^2 measures how well the variance of the output variable is explained by the model. It is defined as

$$R^2 = 1 - \frac{\sum_{i=1}^{|\mathcal{T}|} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{|\mathcal{T}|} (y_i - \bar{y})^2}, \quad (2.27)$$

where $\bar{y} = \frac{1}{|\mathcal{T}|} \sum_{i=1}^{|\mathcal{T}|} y_i$ is the mean of the ground truth values.

Pearson Correlation Sometimes, it is not important to predict the exact value of the output variable, but rather to predict the correct ranking of the values. For this, we can use different correlation metrics. While Pearson indicates the linear correlation between the values, Spearman measures the correctness of the ranking. Formally, Pearson correlation is defined as

$$\text{Pearson} = \frac{\sum_{i=1}^{|\mathcal{T}|} (\hat{y}_i - \bar{\hat{y}})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{|\mathcal{T}|} (\hat{y}_i - \bar{\hat{y}})^2} \sqrt{\sum_{i=1}^{|\mathcal{T}|} (y_i - \bar{y})^2}}, \quad (2.28)$$

where $\bar{\hat{y}}$ and \bar{y} are the mean of the predicted and ground truth mean scores, respectively.

Spearman Correlation Spearman correlation computes the ranks of the predicted and ground truth mean scores and then computes the Pearson correlation between the ranks:

$$\text{Spearman} = \frac{\sum_{i=1}^{|\mathcal{T}|} (\hat{r}_i - \bar{\hat{r}})(r_i - \bar{r})}{\sqrt{\sum_{i=1}^{|\mathcal{T}|} (\hat{r}_i - \bar{\hat{r}})^2} \sqrt{\sum_{i=1}^{|\mathcal{T}|} (r_i - \bar{r})^2}}, \quad (2.29)$$

where \hat{r}_i and r_i are the ranks of the predicted and ground truth mean scores, respectively.

Jensen-Shannon Divergence (JSD) is a measure of the similarity between two probability distributions. For this, the predictions and ground truth values have to be vectors instead of scalars, so we use $\hat{\mathbf{y}}$ and \mathbf{y} , respectively. JSD is defined as the symmetric Kullback-Leibler divergence between the two distributions $\hat{\mathbf{y}}$ and \mathbf{y} :

$$\text{JSD}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{2} \left(\text{KL} \left(\hat{\mathbf{y}} \parallel \frac{\hat{\mathbf{y}} + \mathbf{y}}{2} \right) + \text{KL} \left(\mathbf{y} \parallel \frac{\hat{\mathbf{y}} + \mathbf{y}}{2} \right) \right), \quad (2.30)$$

where $\text{KL}(\cdot \parallel \cdot)$ is the Kullback-Leibler divergence. For two probability distributions \mathbf{p} and \mathbf{q} , it is defined as

$$\text{KL}(\mathbf{p} \parallel \mathbf{q}) = \sum_i \mathbf{p}_i \log \left(\frac{\mathbf{p}_i}{\mathbf{q}_i} \right). \quad (2.31)$$

2.5.3. Representation

For representation tasks, the input to evaluation metrics is a list of different items that are represented by the model in an embedding space, where closer items mean higher

2. DL Foundations

similarity. The goal of the metrics is to measure how well similar items are clustered together. This is usually done by computing the distance between the embeddings of the items and then sorting the items by their distance to a query item.

Formally, the model outputs vector \mathbf{q} for a query item from the test set \mathcal{T} and all other test set items (called the reference set \mathcal{X}). We then compute the i nearest neighbors $\mathcal{N}_{\mathbf{q}}^i \subset \mathcal{X}$ of \mathbf{q} in the reference set \mathcal{X} as an ordered set with $|\mathcal{N}_{\mathbf{q}}^i| = i$ and $\forall \mathbf{x} \in \mathcal{X} \setminus \mathcal{N}_{\mathbf{q}}^i$ it holds:

$$d(\mathbf{x}, \mathbf{q}) \geq \max_{\mathbf{z} \in \mathcal{N}_{\mathbf{q}}^i} d(\mathbf{z}, \mathbf{q}), \quad (2.32)$$

where $d(\cdot, \cdot)$ is the distance between two embeddings. We then compute any of the following metrics for each possible query item and average the results over all query items. For simplicity, we omit the average over all query items in the following equations.

Precision at i (Prec@ i) Given the first i nearest neighbors $\mathcal{N}_{\mathbf{q}}^i$ of a query embedding \mathbf{q} , the Prec@ i is the percentage of this set that have the same class as the query embedding:

$$\text{Prec@}i = \frac{\sum_{\mathbf{x} \in \mathcal{N}_{\mathbf{q}}^i} \mathbb{1}(y_{\mathbf{x}} = y_{\mathbf{q}})}{i}, \quad (2.33)$$

where $y_{\mathbf{x}}$ is the class of embedding \mathbf{x} .

Precision at 1 (Prec@1) Consequently, Prec@1 is a special case of Prec@ i , where $i = 1$. It measures the percentage of query embeddings where the nearest neighbor from the reference set has the same class as the query.

R-Precision R-Precision is similar to Prec@ i , however, i depends on the query embedding. For one query embedding \mathbf{q} , the $R_{\mathbf{q}}$ closest embeddings are retrieved, where $R_{\mathbf{q}}$ is the number of examples with the same class label y_i as the query embedding:

$$R_{\mathbf{q}} = |\{\mathbf{x} \in \mathcal{X} \mid y_{\mathbf{x}} = y_{\mathbf{q}}\}|. \quad (2.34)$$

The R-Precision then simply follows the definition of Precision@ $R_{\mathbf{q}}$ for this query item \mathbf{q} , i.e., the average fraction of items having the same class label. Note that $R_{\mathbf{q}}$ can be different for different query embeddings \mathbf{q} , so the number of nearest neighbors is not fixed as in Prec@ i . This metric measures how well the model puts items with the same property value closer together. The higher the R-Precision, the better the embedding clusters w.r.t. the class. Note that we use R as the name for this variable even though it contradicts our notation. We use the uppercase letter to be consistent with its naming in the literature [193].

Mean Average Precision at R (MAP@R) A weakness of R-Precision is that it does not take the ranking of the retrieved items into account. Given an R-Precision of 0.5, this means that half of the $R_{\mathbf{q}}$ nearest neighbors belong to the same class as the query embedding \mathbf{q} . However, it is not possible to tell whether the items with the same class are

closer or farther apart from the query. Musgrave et al. [193] thus propose the MAP@R metric, which takes the ranking into account. It is defined as

$$\text{MAP@R} = \frac{1}{R_{\mathbf{q}}} \sum_{i=1}^{R_{\mathbf{q}}} p(i), \quad (2.35)$$

where $p(i)$ is

$$p(i) = \begin{cases} \text{Prec@}i & \text{if } y_i = y_{\mathbf{q}} \\ 0 & \text{otherwise,} \end{cases} \quad (2.36)$$

and y_i is the class of the i -th nearest neighbor of \mathbf{q} . This gives lower scores to lower ranked correct items, which is desirable. Since this metric combines multiple desirable properties of Precision and R-Precision, we use it as our main metric for representation tasks.

2.5.4. Ranking

Ranking tasks are similar to classification tasks, but the output of the model is not a single class, but a list of classes, where the first class is the most likely class. The goal of the metrics is to measure how well the model ranks the ground truth class of the input to the top of the list. Formally, the model outputs a list of classes $\mathcal{C}_{\mathbf{x}}$ for an input item \mathbf{x} from the test set \mathcal{T} .

Accuracy@5 Accuracy@5 measures the fraction of test examples where the method correctly puts the target output in the top five ranks. We thus define $\mathcal{C}_{\mathbf{x}}^5$ as the top five most likely classes in $\mathcal{C}_{\mathbf{x}}$:

$$\text{Accuracy@5} = \frac{1}{|\mathcal{T}|} \sum_{\mathbf{x} \in \mathcal{T}} \mathbb{1}(y_{\mathbf{x}} \in \mathcal{C}_{\mathbf{x}}^5), \quad (2.37)$$

where $y_{\mathbf{x}}$ is the ground truth class of the test item \mathbf{x} .

Mean Reciprocal Rank (MRR) takes the rank of the ground truth class into account. The higher the correct class is ranked by the model in the class list, the better. The MRR is defined as

$$\text{MRR} = \frac{1}{|\mathcal{T}|} \sum_{\mathbf{x} \in \mathcal{T}} \frac{1}{\text{rank}(\mathbf{x})}, \quad (2.38)$$

where $\text{rank}(\mathbf{x})$ is the position at which the target item is ranked by the model in the output list $\mathcal{C}_{\mathbf{x}}$. Always predicting the correct item at first position leads to a MRR of one, while bad models achieve a MRR closer to zero.

3. Applications

In this thesis, we aim to understand and improve Deep Learning (DL) models in a model-agnostic way. For this, we explore the effectiveness of six principles by utilizing them in a broad set of applications. We now give an overview on these applications that span the three tasks *classification*, *regression*, and *representation*, as well as the two input modalities *image* and *text*. We state the modalities and tasks for each application in square brackets in the section title as a comma-separated list each, separated by a semicolon. For each application, we give application-specific related work w.r.t. the facets that we investigate in our implementations as well as a short overview of the datasets we use.

3.1. Land Use Regression (LUR) [image; regression]

LUR refers to the task of estimating air pollution concentrations based on land-use information [92, 22, 300]. Based on the application needs, the target variable can vary, e.g., NO_2 , $\text{PM}_{2.5}$, PM_{10} , or surface dust concentrations. Different approaches use features such as map entities, population, or road information. Such models are useful to estimate concentrations in areas without monitoring stations, which are usually sparsely distributed. In our application, we use openly available map images from OpenStreetMap (OSM) to estimate NO_2 (nitrogen dioxide) concentrations for locations without measurements using a DL model called MapLUR. We then generate artificial map images to better understand the behavior of this model.

3.1.1. Application-Specific Related Work

Current LUR models establish Linear Regression (LR) techniques as the de facto standard model [22, 300, 194]. Especially noteworthy is the Escape project [52, 22] which has built models for 36 European areas. The model building procedure of this project has become a standard approach [194, 297, 176, 186, 280] with a tool automating the process of variable generation, modeling, and prediction with a model based on LR [189].

However, more advanced Machine Learning (ML) methods are starting to become more common. For example, Random Forests (RFs) [27] have been used successfully in LUR scenarios [28, 34]. Neural Networks (NNs) have also been applied to a range of pollutants [2, 158, 34, 58, 304, 15, 14, 4, 158, 304, 31]. Such NN-based models are typically simple Multilayer Perceptrons (MLPs). However, there are DL models which use Recurrent Neural Networks (RNNs) or Deep Belief Regression networks to forecast pollution concentrations from earlier measurements or fill missing values for locations

3. Applications

where measurements already exist [58, 304, 15, 14], which is not directly related to the task of LUR. Both RFs and NNs have been shown to outperform LR in LUR [34, 28].

All aforementioned LUR models rely on manually engineered features, which are typically gathered from various locally available data sources that might not be available elsewhere. The combination of using hand-engineered features with a simple model like LR helps with the interpretability of the model.

In contrast to these methods, we propose a DL model based on Convolutional Neural Networks (CNNs), which is able to automatically learn relevant features from openly available maps. This makes the model more powerful but less interpretable. With our approach of using generated map images, we try to better understand what kind of high-level features the model is extracting from the image.

Image-based approaches have been used before in the context of air quality estimation and pollution detection. Singh interpreted modeled air pollution data as images and used non-ML image classification techniques in order to detect significant periods of elevated air pollution [249]. Furthermore, CNNs have been used before in the context of air quality estimation by Zhang et al. [313] and Li et al. [150], who propose models to estimate air haze level using photos from, for example, mobile phones or webcams. In contrast, our work uses map and satellite imagery depicting land-use as model input, making our model more closely related to LUR models. Additionally, our model estimates pollution concentrations instead of haze levels.

3.1.2. Data

In our LUR experiments, we use pollutant concentrations from the London Atmospheric Emissions Inventory (LAEI) [3]. It contains modeled annual mean concentrations of NO_2 and PM_{10} , among other pollutants, at a 20 m grid level for the complete Greater London area in 2013. Mainly, we use the NO_2 concentrations of the dataset since it is a very frequently used pollutant for LUR models. The data is the result of a dispersion model which incorporates a vast number of input factors like for example road and rail networks, traffic data, aviation, pollution from individual industrial premises, domestic and commercial fuel consumption, as well as fires. Through this approach, 5 856 428 data points were generated where each data point represents a 20 m by 20 m cell [3].

From the dataset, we sample a training set consisting of 3000 data points and a test set consisting of 1500 data points from the Central London part of the dataset in order to have a reasonable number of urban data points for our experiments. The map images, which we use to depict the areas around each air pollution data point, cover an area of 80 m by 80 m. The 20 m by 20 m cells are in the center of these images, allowing our model MapLUR to see more of the surroundings and incorporate information about distant emission sources. In order to avoid a potential evaluation issue, we sample data points in such a way that no images can overlap. Please refer to our paper for more details on the dataset and the sampling procedure [256].

As input data for our CNN based MapLUR model in [256], we use map images from OSM [203], which is an open database for map data that is built and maintained by volunteers. Data from OSM can be used to render maps in various ways through different

stylesheets. We render map images based on OSM data without any text like street or station names, avoiding the obstruction of map features. See our work in [256] for details.

3.2. Image Aesthetics Assessment (IAA) [image; classification, regression]

IAA is the task of automatically quantifying the aesthetic appeal of an image. In the literature, mainly two subtasks can be found [265, 311, 122, 94, 149]: The *binary task* is a binary classification task where each image is classified as either aesthetic or unaesthetic. A classification model is then trained to assign each image to one of the two classes. The more realistic task is the *continuous task*, since it allows for applications such as image ranking or generally a more fine-grained comparison of different images [60]. Here, the task is to output a continuous value, where higher values indicate more aesthetic images. In our experiments, we try to improve IAA models with better pretraining as well as explore the use of Contrastive Language-Image Pre-Training (CLIP) for this task.

3.2.1. Application-Specific Related Work

Several DL architectures and loss functions have been developed to improve the performance on the binary and continuous IAA tasks. The vast majority of these models is based on CNN architectures pretrained on ImageNet, which are modified and finetuned on the Aesthetic Visual Analysis (AVA) dataset. One of the most popular DL models for IAA is Neural Image Assessment (NIMA) [265]. NIMA replaces the classification head of a pretrained CNN with a fully connected layer and is finetuned to predict the image’s rating distribution (as given by the AVA dataset) using the Earth Mover’s Distance (EMD) loss. The EMD loss explicitly guides the network to include the order of scores in the training process, resulting in competitive performance to more complex methods on both tasks. Due to the simplicity and elegance of this approach, we use the training procedure of NIMA in our experiments. Instead of using an ImageNet pretrained model, however, we use CLIP model in Chapter 12 to show that CLIP is a more suitable feature extractor than the ImageNet model.

Another DL model for IAA is Personality-Assisted Image Aesthetics Assessment (PA_IAA) [149], which uses multitask learning to predict both a general and personalized aesthetics score based on individual personality traits. The authors train a Siamese network based on pretrained classification models using additional personality training data and the EMD loss. This way, the network learns features that model personality and subjectivity. Note that for this approach, additional personalized data is necessary, which is not present in the AVA dataset.

Most methods resize and crop images to fit them into the required dimensions of the underlying architecture, which can lose details and destroy image compositions. MLSP [94] allows the model to extract features from the whole image by using activations from multiple convolution blocks of a pretrained Inception network. On these features, the authors train a custom CNN architecture.

3. Applications

A related task to IAA is No Reference Image Quality Assessment (NR-IQA), which assesses the *technical* quality of images. Many existing NR-IQA approaches make heavy use of human-sourced quality scores [25, 116, 265]. Under the assumption that distorting images degrades their technical quality, RankIQA [160] distorts technical aspects of high-quality images and learns to rank the resulting images against each other using an efficient ranking loss. Some NR-IQA approaches embed this task into a multitask setting [117, 315] by adding a classification task to the ranking objective [117, 166, 301, 72]. In our multitask experiments in Chapter 16, we expand this methodology to the IAA task and also include a regression task [315] in the multitask setting, estimating the distortion intensity.

The use of distortions to generate weak labels is similar to other self-supervised methods that aim to learn useful features. Many self-supervised pretext tasks for CNNs [112, 68, 77, 136] teach universally applicable image features using tasks like rotation classification [68], image part shuffling, or super-resolution [112]. Sheng et al. [243] transfer distortion-based self-supervised NR-IQA pretext tasks to the IAA task. They are able to improve the prediction performance for the downstream IAA task on AVA over the de-facto default ImageNet [46] pretext task [242, 265, 167, 129] by pretraining on mainly *technical* quality focused distortions in a self-supervised multitask pretext setup. In Chapter 16, we expand on this approach by explicitly introducing style and composition aware distortions and evaluating different pretext task combinations against each other.

3.2.2. Data

In our implementations that work on the IAA task, we use the common benchmark dataset AVA [191]. It consists of 255 522 (229 971 training and 25 551 test) images scraped from a photography website¹ along with the distribution of user ratings in $\{1, 2, \dots, 10\}$, where higher scores mean more aesthetically pleasing images. The mean score is then used as the label of the continuous task to indicate how aesthetically pleasing the image is. To create labels for the binary task, images with a mean score of ≥ 5 are considered aesthetic.

3.3. Age Estimation using Class Similarities [image; classification, regression]

Age Estimation is an ordinal classification task with the goal of predicting the age of a person (in years) given an image of their face. It can be used in a variety of areas such as advertisement, surveillance, or biometrics [54, 10]. It has been shown that reliable Age Estimation is hard for humans and that algorithms can outperform humans in this task [82]. We use this application in our experiments to demonstrate how to incorporate additional information about class similarities into the loss function.

¹www.dpchallenge.com (last accessed: 2023-08-15)

3.3.1. Application-Specific Related Work

Age Estimation can be interpreted as either a regression or a classification task. Since the target values have an inherent order and are numbers, regression is the more natural choice. However, since the possible age values are discrete, classification is also a valid approach. Thus, there are many approaches that use regression, classification, or both for Age Estimation [54, 10].

According to ELKarazle et al., approaches for Age Estimation can be categorized into handcrafted, built-from-scratch, and pretrained models [54]. Handcrafted methods rely on manually extracted features and can be deployed on devices with low computational power, but are not very accurate. Built-from-scratch models give control over the design of the model and build upon a learning process, which lets them perform better than handcrafted methods. However, they require more computational power and take more time to build, as they need a lot of training data and time. Pretrained models use transfer learning to build upon the features extracted from a pretrained model. Given that the pretrained model was trained on a suitable pretraining task, these models do not need as much training data and time as built-from-scratch models. However, they are still more computationally expensive than handcrafted methods. Also, the use of predefined models does not give much choice regarding the design of the model. If the model is pretrained on an unsuitable task, the model might not be able to learn the features needed for the Age Estimation task and thus can perform badly.

In our implementation in Chapter 14, we interpret Age Estimation as a classification task, but include similarities between the age classes as domain knowledge in the loss function. We then measure the performance improvement that can be achieved using our loss compared to the standard Categorical Cross Entropy (CCE) loss function.

3.3.2. Data

For our experiments, we use two common datasets for Age Estimation [54]: The UTK-Face [318] dataset is a dataset for Age Estimation that contains over 20 000 images of human faces annotated with their age ranging from 0 to 116 years. The AFAD [200] dataset is also an Age Estimation dataset, containing over 160 000 images of Asian people, ranging from 15 to 40 years. For both datasets, we randomly sample training/validation/test sets using 60/20/20 splits.

3.4. Image Classification with Class Similarities [image; classification]

In Image Classification, the goal is to recognize an object shown in an image, i.e., to classify an image into one of several classes. In our experiments, we will work with a variation of this task, where we want to classify an image but also take into account the semantic similarity between the classes. In this task, classifying an image of a rose as “violet” is less harmful than classifying it as “truck”.

3. Applications

3.4.1. Application-Specific Related Work

Previous work on including task-specific knowledge in classification is mostly designed for specific use cases, requires modifications to the model architecture or training procedure, or implicitly learns the information while training. Sukhbaatar et al. *implicitly* learn a probability matrix that indicates the chance of a falsely assigned class label in order to compensate for noise [258]. This, however, requires changes in the network architecture and a special training procedure. Related to tasks with similar classes are tasks where classes have a taxonomic structure, which is called hierarchical classification. Specifically designed loss functions and/or model architectures use the fact that classes that belong to the same category are more similar than others [33, 298]. Izbicki et al. exploit the geospatial relation between areas on earth to automatically geotag input photos [109]. Their model learns to predict a mixture of densities that spread across multiple areas instead of specific classes/areas. A number of task-specific methods try to use the inherent class order of so-called ordinal classification tasks [65, 78]. For example, Niu et al. use multiple binary classifications each indicating whether the value is greater than the class value [200]. Model architectures incorporating semantic similarities using word embeddings were shown to usually predict more similar classes if they fail compared to models without similarity information [64, 201]. In contrast to the related work, the loss function that we introduce in Chapter 14, Similarity Based Loss (SimLoss), incorporates the domain knowledge as class similarities in the loss function, which does not require special model architectures and works on any common NN classifier. This makes it easy to explicitly support the training procedure with background knowledge. After publication of our approach, the TreeLoss [286] was proposed, which also is a drop-in replacement for the CCE loss. However, this loss function requires that the classes are organized in a tree structure, which is not always the case.

3.4.2. Data

For our experiments in this application, we use the CIFAR-100 dataset [141], which is an image classification dataset that consists of 100 classes, each containing 600 images of size 32×32 pixels. Each class also belongs to one of 20 superclasses, which are more general classes, e.g., classes “rose” and “orchid” have the superclass “flower”. We use these superclasses in the evaluation of our image classification with class similarities task.

3.5. Deep Metric Learning (DML) [image; representation]

In DML, a Deep Neural Network (DNN) is trained to map input images to m -dimensional embedding vectors that should be close to each other in embedding space (as measured by a distance like Euclidean distance or a similarity like cosine similarity) if the corresponding inputs share a given class. Thus, the network has to learn to extract discriminating input features to embed an item.

DML has been applied to many scenarios such as image clustering, retrieval, person reidentification, face verification, 3D shape retrieval, semantic textual similarity, and

speaker verification [119]. In our implementations, we focus on images, since this is the most common application of DML.

There, we mainly work on two facets of DML: The influence of certain high-level features (particularly the background of an image) on the embedding and making DML models more data-efficient by eliminating the need for training images.

In order to measure the generalization capabilities of DML models, one usually does split the dataset into a training and a test set with disjoint classes. Hence, the model has never seen any of the test classes during training. Embeddings thus have to capture relevant features so that examples from these unseen classes are still correctly clustered.

3.5.1. Application-Specific Related Work

In the following, we discuss related work on the two topics we cover in this thesis.

Background Bias in DML

Background bias is the phenomenon that a DNN uses information from the background of an image to compute its output, which can be undesirable and lead to poor generalization. We investigate background bias in DML models, which can be harmful in settings like item retrieval, where the background should not have an influence on the embedding to find similar items. However, researchers have mainly investigated background bias in **classification** NNs. While during test time, only new images from a fixed set of classes are given for classification, in a typical DML setting, the test classes are disjoint from the training classes [193]. Also, DML networks map images to an m -dimensional embedding space and do not classify them. Thus, findings of background bias in classification models do not directly transfer to the DML setting.

In addition, methods developed to combat background bias are often specialized to classification networks and cannot be directly applied to DML networks. Such methods can be divided into two categories, which we term Background Augmentation and Attribution Regularization. Background Augmentation methods exchange the background of images during training or inference with random images [303, 267, 121]. This way, the model cannot find correlations between background features and class labels. Another work proposes to crop the image near the main object to prevent background from being visible in the image [289]. In our experiments in Chapter 11, we use a Background Augmentation technique. Attribution Regularization computes the attribution map of an input sample during training to identify the image regions the model focuses on. The loss function then guides the model to produce attribution maps that resemble the image’s foreground/background segmentation map [230, 241, 155, 36]. Attribution Regularization has not been applied yet to DML, even though attribution map generation methods for DML models exist [134]. In Chapter 9, we also introduce a new attribution map generation method for DML.

Related fields of background bias are also investigated in the literature. NN classifiers often suffer from simplicity bias [240], using the simplest clues to classify an image. Training an additional network that complements a biased model [198] or ensembles

3. Applications

that learn diverse feature sets alleviate the problem that the model only learns a few potentially irrelevant features [205]. To prevent models from using spurious correlations between the image and the class label [307, 264, 234], the network’s last layer can be finetuned on data that does not show such correlations [128].

Zero-Shot DML

As already stated, DML models are usually trained on images that are organized into classes, so binary similarity annotations are readily available for each pair of data points [119]. Testing then uses a disjoint set of image classes to measure the model’s generalization ability, but the data is semantically similar to the training data, e.g., Cars196 [139] only shows cars and face recognition datasets [97] contain faces.

Studying DML generalization for images outside of the training domain has recently become popular [182, 231, 181, 99, 98, 306]. However, all proposed methods to improve the generalization performance to new datasets still use training images. We explore the possibility of zero-shot DML by only using text prompts to create an image embedding space specifically tailored to a desired similarity notion. For this, we use a fixed CLIP [214] model to extract general purpose features.

The ability to rank possible text labels for an image using the cosine similarity of CLIP embeddings has been used in the original paper to perform zero-shot image classification [214]. For classification, the class names need to be known during inference, while in our zero-shot DML approach, we create image embedding spaces reflecting the desired similarity notion. Hence, the model needs to be able to handle images of unknown objects and characteristics, e.g., new car models.

Baldrati et al. use CLIP to alter fashion image embeddings using text prompts [16], e.g., the image of a black dress is combined with the text “is red” to find images of red dresses. While exploiting similar properties of CLIP, we only use text prompts for training a transformation to focus on the desired similarity notion. Image retrieval also uses joint text-image embeddings search for image contents using text [177, 39]. We use text exclusively during training, not during inference. Roth et al. [232] improve DML models by feeding the class names through the CLIP text encoder and guide the embedding process using these representations. While they also use text supervision in their setup, they still use training images for their method and do not explore the suitability for different similarity notions. To the best of our knowledge, no other work has a comparable task setting or method as our method we explore in Chapter 13.

3.5.2. Data

We use a large set of datasets for the evaluation of DML models, depending on the implementation. The most common benchmark datasets for DML are Cars196 [139], CUB200 [275], and Stanford Online Products (SOP) [253] and we use them in all DML related implementations. Here, one half of the classes is used for training and the other half for testing. In addition, we use other datasets for our work in Chapter 13. We now describe each dataset: **Cars196** [139] features 16 185 real world car images. It contains

3.6. Scientific Venue Recommendation [text; classification]

196 classes, each one representing images of one car model. **CUB200** [275] consists of 11 788 images showing birds that are grouped by bird species. Overall, it has 200 classes. **SOP** [253] contains 120 053 images of 22 634 different products. These images are scraped from *eBay* product pages. **Synthetic Cars** [134] is a self-created dataset that contains 100 000 3D-rendered car images with different car models, car colors, background colors, car orientations, sun positions, and camera angles, all sampled independently at random. We provide more information about this dataset in Chapter 7, where we create this dataset for DML analysis purposes. **DeepFashion** [162] contains over 800 000 images of persons wearing different clothes. With each image, there are different attributes such as “Clothing Category”, “Texture”, “Fabric”, and “Fit” that can be used to group the images into several classes. We use this dataset for DML evaluation. **Movie Posters** [41] is a dataset of movie posters and corresponding metadata about the movie, such as genre or production country. Due to non-standard encoding of the files, we overall are able to read 8052 different movie posters and use them in our experiments to evaluate DML models.

3.6. Scientific Venue Recommendation [text; classification]

Scientific Venue Recommendation is the task of predicting the conference or journal (venue in the following) a paper should be submitted to, based on the contents of the paper (i.e., its title, abstract, and/or keywords). We interpret this as a classification task that only outputs one venue, but rank the outputs of the model to provide a list of fitting venues.

3.6.1. Application-Specific Related Work

There exist several online services that recommend venues based on the contents of a publication, but all of them are lacking in some ways:

1. Most of them only recommend journals, not conferences, e.g., Elsevier Journal Finder² [118], Journal Guide³, Springer Journal Suggester⁴, Wiley Journal Finder⁵, Enago Open Access Journal Finder⁶, Edanz Journal Selector⁷, Manuscript Matcher⁸, Journal/Author Name Estimator⁹, and SJFinder¹⁰. Especially in the fields of Computer Science and Artificial Intelligence (AI), most work is published on conferences [274], making this a severe drawback for AI researchers.

²<https://journalfinder.elsevier.com> (last accessed: 2020-09-21)

³<https://www.journalguide.com> (last accessed: 2020-09-21)

⁴<https://journalsuggester.springer.com> (last accessed: 2020-09-21)

⁵<https://journalfinder.wiley.com/search?type=match> (last accessed: 2020-09-21)

⁶<https://www.enago.com/academy/journal-finder/> (last accessed: 2020-09-21)

⁷<https://en-author-services.edanzgroup.com/journal-selector> (last accessed: 2020-09-21)

⁸<https://endnote.com/product-details/manuscript-matcher/> (last accessed: 2020-09-21)

⁹<https://jane.biosemantics.org> (last accessed: 2023-08-15)

¹⁰<http://www.sjfinder.com/journals/recommend> (last accessed: 2020-09-21)

3. Applications

2. Most of the services are commercially motivated, such as Elsevier Journal Finder, IEEE Publication Recommender¹¹, Springer Journal Suggester, Wiley Journal Finder, Enago Open Access Journal Finder, Edanz Journal Selector, or Manuscript Matcher. Publishers and companies provide them to promote their own portfolio or other services. Thus, they diminish the variety of the recommendations by only considering their own journals.
3. Many of the services are black boxes without any information on how they perform their recommendations. There are a few exceptions to this: Journal/Author Name Estimator uses the open source search engine software Lucene to find the 50 most similar papers according to the Lucene index and recommends the journals that occur most often in this set [237]. Kang et al. extract noun phrases from the paper and match these with a database using the Okapi BM25 algorithm [118, 224].
4. None of the provided services explain why a specific venue was chosen.

Only very recently, recommending *conferences* based on authors, abstracts, and keywords became a new research area [105]. However, the authors approach a more general setting that includes conferences from a wide variety of fields. They also incorporate author information into the recommendation and do not provide an explanation to the user why a given conference was recommended. With our work in Chapter 8, Where to Submit (WTS), we introduce an open and explainable system that recommends both journals and conferences and that is automatically analyzed to showcase the relevant words and phrases for the recommendations using the Integrated Gradients method [260].

3.6.2. Data

In our implementation, we base our dataset on the Semantic Scholar [8] dataset¹² by extracting all publications that were published in the research fields *Artificial Intelligence* and *medicine*.

A publication is considered to be an AI paper if it was published in one of the scientific venues given by Kersting et al. [124]. We manually match them as closely as possible to the Semantic Scholar venues. This procedure leads to 77 distinct venues. We also add a class called “non-AI” consisting of 20 000 publications from other fields, to let the model learn the difference between AI and non-AI venues, resulting in 78 classes for this dataset. Overall, we have 245 573 publications in the AI dataset.

In the field of medicine, we only use publications from Semantic Scholar that originate from Medline, a medical publication database. Due to a high number of venues with few publications, we only consider the top 78 venues (the same number as for the AI dataset, making the performance metrics comparable), which account for about 10 % of the whole dataset. This leads to overall 2 924 609 publications in the medicine dataset. In general, we only keep publications where no input information is missing. For more information about the dataset, see our work in [131]. Table 3.1 gives an overview of both datasets.

¹¹<http://publication-recommender.ieee.org/home> (last accessed: 2020-09-21)

¹²Release from 2019-01-31.

3.7. Sentiment Analysis [text; classification]

Table 3.1.: Comparison of the subsets for AI and medicine venues from Semantic Scholar.

Metric	AI	Medicine
Publications	245 573	2 924 609
Venues	78	78
Avg Title length	9.24	13.29
Avg Abstract length	153.65	185.18
Avg used Keywords	8.73	12.33
# total Keywords	32 139	209 525
Min. publication count	63	20 959
Mean publication count	3148.37	37 494.99
Standard Deviation	4102.96	28 170.86
Median publication count	1597.5	27 734.5
Max. publications count	21 122	201 469

From both datasets we randomly sample 80 % as training, 10 % as validation, and 10 % as test sets in a stratified manner. While our approach might favor larger conferences with this sampling strategy, we argue that this procedure better reflects the venue landscape. Larger conferences usually cover a larger thematic scope and accept more manuscripts.

3.7. Sentiment Analysis [text; classification]

Sentiment analysis refers to the task of predicting the sentiment of a text, i.e., classifying each text into a sentiment category: *negative*, *neutral*, or *positive*.

In our experiments, we perform sentiment analysis on Twitch.tv comments. Twitch.tv is a live streaming platform that enables individuals and companies to broadcast live content, including gaming, cooking, and other live events. The platform allows users to follow channels, access a list of currently streaming channels, and engage in live chat with the channel’s community. Channel owners can also appoint moderators to help manage the chat and can set chat rooms to be accessible only to subscribers or moderators. Twitch has a rich history, beginning as a spin-off to the Justin.tv platform and later becoming an independent entity that was acquired by Amazon. Twitch has become a source of income for individual streamers, who earn money from subscriptions, advertising deals, and branded content. One of the defining features of the Twitch platform is its use of emotes, which are graphics used to express emotions and ideas in the chat. Emotes play a crucial role in the Twitch community, helping to create a unique language and culture on the platform. Twitch and streamers may want to analyze the sentiment of users regarding certain products or events, and the information gathered from this analysis can help to increase user engagement and income.

3. Applications

3.7.1. Application-Specific Related Work

Sentiment analysis is a widely researched application area of AI. Besides popular usage areas containing datasets of Amazon product reviews [174] and IMDB movie reviews [168], lots of studies have also challenged more difficult domains such as the short, orthographically inconsistent messages found on Twitter [196, 227, 228, 197, 229]. Research in this area also entails the use of emojis for gaining insights into the sentiment of a message [138], while there are also openly available resources for sentiment classification specifically geared towards social media texts, such as the Valence Aware Dictionary for sEntiment Reasoning (VADER) lexicon [69]. Additionally, there exist labeled datasets such as the Sentiment140 Twitter dataset [71] utilizing text emoticons at the end of messages to generate a large amount of so-called weakly labeled messages for use in supervised training environments. We will use a similar approach to generate our dataset in Chapter 15.

The streaming platform Twitch itself has also gathered some research interest over the years. For a general overview of Twitch and its user communities, we refer readers to Smith et al. [251]. While Kaytoue et al. [120] analyze viewer numbers and prove aspects such as the impact of tournaments and video game releases, Nascimento et al. [199] conduct a more indepth research on behavioral patterns of audiences, such as channel switching and channel surfing. There also exist studies in the area of viewer sentiment, with Löffler et al. [163] investigating the impact of background color on the perceived sentiment of chat messages. Barbieri et al. [18] research the process of removing ending Twitch emotes from comments and predicting the removed emotes with Bidirectional Long Short-Term Memory (LSTM) NNs. Predicting the overall sentiment of individual comments on Twitch, however, is a novel contribution of our work in Chapter 15.

3.7.2. Data

Since we are the first to investigate this setting, we create two datasets for our experiments: a large unlabeled dataset of Twitch.tv comments and a small labeled dataset, which is exclusively used for evaluation. For more information about the dataset creation process and a detailed analysis of the dataset, see our work in [133].

We collect a large dataset of publicly accessible comments from Twitch.tv by crawling all comments from the site using the available API for April, May, and June of 2018. From these 3 069 046 977 unlabeled Twitch messages, we sample 2000 comments and label them by three crowd workers each, resulting in 1922 comments with a clear sentiment label given by majority voting. From the labeled comments, 404 (21.02 %) are classified as negative, 748 (38.92 %) as neutral, and 770 (40.06 %) as positive.

4. Related Work for the Explored Principles

We now discuss related work of the principles we explore in this thesis. Since the principles are oftentimes not task- or domain-specific, we are not able to cover all related work for each principle. We instead aim to structure the methodological implementations and discuss representative and highly influential works such that we can classify our implementations we propose in this thesis.

4.1. Understanding Deep Learning (DL) Models

Given the taxonomy for interpretable Artificial Intelligence (AI) by Molnar from Section 1.1.1, we focus on discussing related work about model-agnostic/post-hoc interpretation methods that can provide local and global explanations for DL models. Most research in this taxonomy area focuses on explaining classification models by highlighting important input elements such as pixels or tokens or by exploring the influence of certain input concepts [171, 153]. Our concrete implementations use similar techniques but are designed for the explanation of regression and representation models. Thus, they fill an important gap in the literature.

4.1.1. Generated Input Data

We explore the use of synthetic data to analyze the influence of properties on the output of a Neural Network (NN) in a post-hoc and model-agnostic way, as categorized by Molnar [185]. While prior work mostly uses generated datasets for training NNs to improve performance in real-world scenarios [269, 137, 88], only few works explore the use of synthetic data to analyze DL models trained on real-world data.

Many works investigate the robustness of models on different data variations, which is possibly closest to our explicit analysis objective. One can argue that robustness and high-level feature influence are complementary: If a model is robust to a certain property change, it is not influenced by this property. However, robustness research mostly tries to find properties that should not, but do influence the model. The general idea of feature influences is to find influencing properties, even if they are not necessarily bad.

For robustness research, we can identify mostly two methodological directions: Creating augmentations from real-world data or generating synthetic data. Even though performing data augmentation creates new data by controlling specific aspects of the original data, we would not consider it generated data as we use it in our principle implementations. However, works in the image domain find that even simple spatial transformations such

4. Related Work for the Explored Principles

as rotation, scaling, or translations of real images significantly impact the performance of vision models [57, 11]. Here, the properties of interest are possible to vary using spatial transformations. In Natural Language Processing (NLP), transformations can be applied to the text itself, such as replacing words with synonyms [285]. Approaches such as adversarial examples slightly modify existing data inputs such that the model output changes [261, 75, 6, 285]. Since they are based on existing data, we also do not consider them to be generated data as we use it in our implementations.

On the other hand, synthetic data allows for fine control over the generation process, allowing to create different compositions and more abstract inputs. As with data augmentations, most works from the literature investigate the robustness aspect of NNs. Alcorn et al. [5], Abbas and Deny [1] generate synthetic images in a 3D rendering software to analyze the influence of poses (position and orientation) of objects in images on the output of image classification models. Unsurprisingly, they find that unusual poses lead to a decrease in classification accuracy. Madan et al. [170] shows the same for changes in 3D viewing angles and lighting. Ibrahim et al. extend the variations to background and size properties and to more vision models, but mainly reach the same conclusion as the work before [106]. These results are supported by von Kügelgen et al., who specifically investigate Self-Supervised Learning (SelfSL) models [273]. While the investigation of robustness is certainly an important topic for DL, we use the Generated Input Data principle (see Section 5.1) to analyze the influence of specific properties, i.e., higher-level features, on the output of a model. For this, we analyze the changes in the output of the model, not only their drop in evaluation metrics. This gives us more insight into how high-level features influence the model’s predictions. Also, we can compare different models based on these results.

Possibly closest to our work is research on bias detection in NLP [171]. For example, Vig et al. [272] investigate gender bias in Large Language Models (LLMs) by feeding the beginning of sentences like “The nurse said that” to a language model and analyze the probabilities for the next token to be “he” or “she”. They then swap the word “nurse” with “man” or “woman” and observe the change in the probabilities, computing the so-called natural indirect effect. Here, the higher-level feature is the gender of the person in the sentence and it is investigated how this affects the model’s behavior.

4.1.2. Gradient-Based Attribution

Using the gradient of the model w.r.t. the input data to analyze the influence of specific features on the output of a model (post-hoc feature attribution, as categorized by Molnar [185]) is a common approach in the literature. Multiple methods have emerged from this principle, which have been applied to different tasks and different modalities. The resulting feature attributions can be visualized and show what the model attends to when making a prediction. Most methods are by default only applicable to classification methods [248, 61, 320, 319, 305, 250]. These methods highlight input features that have encouraged the model’s decision for *one* class.

For regression tasks, there are attempts to extend these methods to regression models [256]. In Deep Metric Learning (DML), most work on feature importance focuses on

the image retrieval task and aims to highlight areas that were the reason for the respective similarity score [257, 76, 323, 238]. Stylianou et al. [257] base their work on the Class Activation Maps (CAMs) [319] method, which calculates the dot product of each position in the last convolutional activation map with the other image’s representation, resulting in a low resolution image showing what regions were deemed similar. Their approach is only applicable to NN architectures that obtain the image representation by applying global average pooling to the activation map of the final convolution layer and compute the image pair’s similarity using the dot product. Its extension Grad-CAM [238] takes the gradient of additional fully connected layers into account. Adapted to the DML setting by Zhu et al. [323], the activation map’s pixels of two images are compared to each other, thus matching similar regions between image pairs. The method proposed by Chen et al. [37] uses triplets of training images and saves Grad-CAM’s attribution maps to a database along with the corresponding embeddings. For test images, attribution maps of similar images from the database are interpolated.

While explaining similarities between image pairs is useful for tasks like image retrieval, in our proposed method in Chapter 9, we want to assess what image features are used by the model to embed one image. We then use this information to compare different DML methods regarding these regions. Our proposed method works with any NN architecture and computes importances on pixel level.

In general, using raw gradients often leads to noisy attribution maps, which motivated the development of methods that smooth the gradients. One method that does that and that we apply in our experiments is Integrated Gradients [260]. We explain the idea and methodology of it in Chapter 8. Another method is SmoothGrad [250], which we apply in our attribution method that we propose for DML models.

4.2. Improving DL Models

As with the understanding of DL models, there are many works that aim to improve the performance of DL models in a model-agnostic and non-invasive way.

4.2.1. Input Masking and Augmentation

Data augmentation and masking can be seen as regularization techniques that allow DL practitioners to include biases about the data and tasks into the model. Thus, they are not data- and task-agnostic, since they require deep knowledge about data properties and task specifics. For example, flipping the input might work for images but is not applicable to text, leading to reversing sentences. Also, horizontally flipping an image might work for identifying natural objects, but not for detecting digits. Thus, a large corpus of data augmentation methods has been developed by different strings of research. We focus on the most common methods in Computer Vision (CV) [244] and NLP [245, 59], but there are a myriad of methods for other modalities, such as audio [287] and time series [290]. Data augmentations have been better explored in CV than NLP, since it is more intuitive to estimate the effects of certain augmentations on images and their desired outputs. In general, analytically investigating the effects and properties of data augmentations have

4. Related Work for the Explored Principles

gained research interest in recent years [17, 66]. For example, Geiping et al. find that well-crafted data augmentations can increase generalization more than additional training data [66].

Both, image and text data augmentation methods, can be split into basic manipulations and DL approaches. Basic manipulations include color space transformations, geometric transformations, and mixing different inputs for images. For text, basic manipulations include word and character insertions/deletions/replacements using rules, or augmenting intermediate representations in the NN. DL approaches for data augmentation are based on methods that generate new data by utilizing DL models. For images, these include adversarial training, neural style transfer, and Generative Adversarial Network (GAN) based methods. Text can be translated back and forth between languages to receive different input texts that should convey the same meaning [244]. Similar to CV, style transfer and generative data augmentations are possible with text. For example, given some examples for one class as prompts, LLMs can generate new data points with the same class [245].

While data augmentation can be used in both domains, the application of those methods is less common in NLP. Only recently, with the rise of generative AI and LLM, augmentation is easy to obtain for text [291]. Before, input masking has been a more common technique in NLP, mainly in combination with a pretraining task. The Bidirectional Encoder Representations from Transformers (BERT) [48] by Devlin et al. and derivatives such as [161] are perhaps the most popular examples for this. They mask 15% of the input tokens and train the model to predict the masked tokens, which makes it easy to collect a large corpus of training data. The model has to learn the connections between words to be able to solve this task. Inspired by this simple pretraining task that has become a very solid base for downstream task training, CV researchers have adopted this idea for Vision Transformers (ViTs), leading to Masked Autoencoders (MaskedAEs) [87]. In the sense of using Transformer-based models for both, text and images, we can see that NLP and CV are converging in their approaches.

Our concrete implementations of this principle work with images and texts. We replace the background of images, which simultaneously masks and augments the image (Chapter 11). We also mask/delete words from text input (Chapter 15) and apply filters to images (Chapter 16).

4.2.2. Feature Extraction using Pretrained Multimodal Models

Multimodal models use multiple modalities of data to learn a task. Whereas there are many ways to manage multiple modalities as given by the structure by Liang et al. [151], we concentrate on approaches that learn *representations*, since we want to use these as features in downstream tasks. Here, we can divide the literature into fusion, coordination, and fission approaches. Fusion approaches map the m modalities to $n < m$ representations. Coordination approaches represent the m modalities into $n = m$ representations by exchanging cross-modal information. Fission approaches represent the m modalities into $n > m$ representations.

Our implementations focus on the Contrastive Language-Image Pre-Training (CLIP),

which is one of the most popular multimodal models from recent years [214] and has been described in Section 2.3.4. It is a representation coordination approach, since it maps the two input modalities to two representations in the same representation space. Due to its very large training dataset and quite simple training objective, it shows very general and powerful feature extraction capabilities. The authors have shown that CLIP is able to perform Zero-Shot Learning (ZSL) on different datasets, which means that no training data for the target classes is required. While CLIP has been trained using two modalities, each encoder can be used separately. We do this in our experiments in Chapter 12 and Chapter 13 by using CLIP frozen image encoder and text encoder to extract image and text features that are in the same representation space.

4.2.3. Weak Label Generation

Weak labels are target outputs for the training of DL models that are less than perfect, e.g., because they are not manually annotated and checked. They can be collected relatively easily, but are usually not as accurate as manually annotated labels. However, due to the low cost of obtaining, they can be obtained in large quantities to compensate their less than ideal quality. Such labels can come from user-written heuristics, knowledge bases, pretrained models, or external tools [314, 225]. This way of obtaining labels is also called programmatic weak labeling.

In the programmatic weak supervision framework, a set of labeling functions is defined that provide weak labels for a subset of the data [216, 215]. Due to the noise in the weak labels, there can be conflicts between the labeling functions regarding the same data point. Thus, so-called label models are used to combine the weak labels into a single label, e.g., majority voting, which is then used for training the Machine Learning (ML) model [216]. Also, other approaches directly incorporate the label models into the trainable model, denoising the weak labels during training [219].

In our principle implementations, we use models with stronger assumptions and hypotheses that generate weak labels. In all implementations, we use only one labeling function, so we do not need to combine the weak labels. While in one of our implementations we exclusively use weakly labeled data to train a model, in the other we use weak labels for pretraining and then finetune the model on ground truth labels. This intertwines this principle with Semi-Supervised Learning (SemiSL).

4.2.4. Loss Function

Choosing an appropriate loss function for the training of the NN is a crucial step in the training process. The loss function should be able to measure the distance between the desired output and the actual output of the model. Its choice thus has a large impact on what the model learns and how well and fast it learns [281]. For example, for regression tasks, both the Mean Absolute Error (MAE) and the Mean Squared Error (MSE) (see Section 2.4.2) share the same fundamental idea, but the MSE is more sensitive to outliers due to its squared distance. However, while the MAE loss gets lower for smaller distances between predicted and desired outputs, its gradient is independent of this distance (-1

4. Related Work for the Explored Principles

or 1). This can make the model overshoot model parameter optima. To combine the strengths of both loss functions, the Huber loss performs a piece-wise combination of the MSE and MAE [102].

For classification tasks, the Categorical Cross Entropy (CCE) is the dominant loss function in the literature. However, losses such as MSE or MAE can also be used for classification tasks, since they can calculate a difference between the prediction of the model (fed through a softmax function) and the desired output (a one-hot encoded vector giving the correct class) [104]. The PolyLoss approximates different loss functions using a combination of polynomials and tunes the combination parameters to the task at hand [147]. Barz and Denzler propose to use the Cosine loss instead of CCE, which is defined as one minus the cosine similarity between the predicted and desired output. They show that this loss is suitable for datasets with only few samples per class. If an additional hierarchy of classes is given, the desired output vectors can easily be adapted from one-hot encoded vectors to class embedding vectors that reflect the semantic similarity between classes [20]. We introduce additional knowledge about the similarities between classes into the CCE in Chapter 14. Wang and Izbicki [286] explore a similar approach but assume that classes are arranged as a tree. Intuitively, whenever the weights for one class is updated, the weights for all classes that are similar to this class should also be updated.

In DML, there are a multitude of different loss functions that aim to improve the performance of the model, make it more robust, or make it faster to train [193]. Mainly, we can split the list of loss functions into three categories: ranking-based losses, classification-based losses, and hybrid losses. Ranking-based losses, e.g., Contrastive loss [80] or Triplet loss [288], work on tuples of samples and guide the model to output representations such that similar input items are mapped to representations with smaller distance than dissimilar input items. Classification-based losses, e.g., Normalized Softmax loss [159, 276, 312] or ArcFace loss [47], optimize vectors that represent classes, so-called proxies. Items should then be mapped to representations such that they are close to their respective class proxy and farther apart from other proxies. While ranking-based losses usually lead to better performance since they can contrast two or more examples directly to each other, classification-based losses often converge faster, because they do not need to compare all items to each other and have one proxy per class that is more stable during training than the representations of the items. Thus, hybrid methods try to combine the strengths of both approaches [101].

Independent from the chosen task and domain, different loss functions can be combined when training a NN in a multitask setting. Here, the choice of how the different task losses are combined is important, since some tasks could have more influence on the model than others. There are different approaches for loss combination [152]. The simplest and most widely used one is the direct sum approach, where the sum of all training losses is calculated and optimized. Sometimes, the mean loss is computed, but this is equivalent to a scaled version of the direct sum approach. The weighted sum approach assigns a weight to each task loss and then sums them up. The weights can be manually set, learned during training by taking the types of loss functions into account [42], based on the gradient norms of each loss [40], or by adapting the loss weights during training based on the loss changes in the last few update steps [157]. In [175], the maximum of

the losses is optimized, weighting the other losses by a factor of zero. This optimizes for the worst case, but gives no smooth loss landscape and it thus hard to optimize. Inspired by curriculum learning [23], where the difficulty of training examples the model is presented with increases over time, multitask learning adopted this concept by increasing the task difficulty during training, letting the model first learn easy tasks and then more difficult ones [192]. Here, a weighting of the task losses is used, where the weights are updated during training based on the loss values of the previous update steps. If a loss was already low, the weight for this task is increased, prioritizing the training of already well-performing tasks. The multi-objective optimization approach interprets the different task losses as objectives and tries to optimize for all of them at the same time using a quadratic programming solver [239]. Liang and Zhang propose a balanced multitask learning framework that applies a monotonically increasing function to each loss value. The higher the loss, the higher the weight for this task, which is the opposite of the curriculum learning approach. We explain this approach in more detail in Chapter 16 and use it to improve on the Image Aesthetics Assessment (IAA) task.

4.2.5. Other

While we implement the principles introduced in Section 1.1, there also exist other techniques to improve DL methods without dictating the model architecture. We briefly discuss a non-exhaustive list in this section.

Scaling up Improving the performance of DL models can be done by scaling up most of the components of the training process. Especially larger training datasets usually result in better performance, since the model generalizes better, which initially led to the rise of DL [145]. Also, training the model for more epochs can lead to better performance, even when the validation performance has plateaued [211].

Optimizer There are a lot of different optimizers that can be used for training DL models which feature different strengths and weaknesses, such as better generalization or faster convergence [81]. Stochastic Gradient Descent (SGD) — and its version with momentum — is one of the simplest possible choices for gradient based methods. It has the learning rate as one parameter, which is applied to the gradients before updating the model parameters. This uniform scaling of the gradients in all directions can lead to bad performance on some ill-scaled problems and requires a lot of learning rate tuning [125]. Thus, optimizers have been explored that introduce an adaptable learning rate for each parameter. Examples for such optimizers are Adam [127], AdaGrad [51], and RMSProp [268]. Some works have found that SGD generalizes better than adaptive optimizers [295]. Thus, researchers try to combine the strengths of both optimizer types, e.g., by switching from Adam to SGD during training [125]. Other works found that there is a connection between generalization and the loss landscape, leading to a new suite of optimizers. Such optimizers as Sharpness-Aware Minimization (SAM) take the curvature of the loss function into account by using the second-order derivative of the loss [62].

4. Related Work for the Explored Principles

They are usually computationally more expensive, since the second-order derivative has to be computed, but generalize better than other optimizers.

Initialization A careful choice of the parameter initialization method for a NN is crucial for the training process. Sampling the weights from a gaussian or uniform distribution can lead to exploding or vanishing gradients. To combat this, Glorot and Bengio introduce an initialization, which samples the weights from a uniform distribution with a variance that is inversely proportional to the number of input and output units, leading to more stable training and improved performance [70]. However, their method is specifically designed for feed-forward networks with sigmoid or tanh activation functions. He et al. propose a variant for Rectified Linear Unit (ReLU) activation functions [85]. We argue that these methods are not model-agnostic, since they are specifically designed for certain model architecture components.

Only recently, Zhu et al. developed a method that is model-agnostic and can be used for any NN architecture [322]. Their method introduces scaling factors for the weight matrices that are optimized at the beginning of the training process, such that the loss of the main task decreases maximally.

Pretraining Pretraining NNs on datasets other than the downstream task and then finetuning on the downstream dataset — commonly referred to as transfer learning — can be beneficial for smaller downstream datasets. It is claimed that the pretrained model then already has learned some general representations and extracts knowledge from the external dataset, which can be specialized to the needs of the downstream task [173]. LLMs, for example, are trained on large text datasets, which teaches the model to extract general language related features and correlations. Finetuning such models (or parts of them) on a small downstream dataset can then lead to better performance than training a model from scratch on the small dataset [259]. In CV, usually pretrained ImageNet models are used for finetuning on smaller datasets [223].

Recently, so-called self-pretraining has been explored in CV [87, 53] and NLP [140], challenging the claims and justification of above findings. For self-pretraining, no external dataset is used for pretraining, but only the data that is available for the downstream task. Also, eliminating pretraining and combining labeled downstream data with a small subset of the unlabeled dataset to enrich the dataset has been explored in the literature [308].

Prompting Prompting refers to finding a suitable input to a pretrained model to solve a specific task, which leads to a ZSL setting. In NLP, this is often done with LLMs. Since they learn to predict the next (or a masked) word given the beginning of a text, choosing a suitable text that accompanies the downstream task’s input can lead to good performance [156]. These *prompt templates* are filled with the downstream task’s input and then fed into the model. The resulting answer can then be mapped to the downstream task’s output space. For example, for sentiment analysis of movie reviews, the prompt template “[X] Overall, it was a [Z] movie.” could be used, where [X] is the movie review (the downstream task’s input) and [Z] is the token in the template that the model should

predict [156]. Given the output probabilities for tokens in [Z], the sentiment can be derived. In this example, [Z] was in the middle of the prompt template, which makes it applicable to models that were trained with masked language modelling. Autoregressive models, such as Generative Pre-Training (GPT), can also be prompted, but the prompt template has to be at the beginning of the input [156]. While the prompt templates can consist of real text, their embedding vectors can also be optimized to maximize the performance of the downstream task [156, 321].

While most prompting methods are proposed in NLP, Bahng et al. explore prompting large CV models [13]. Here, they optimize an image perturbation that is applied to all images of the downstream task such that the frozen pretrained model predicts the desired output. They utilize the multimodal CLIP model and other image-only pretrained vision models.

Part II.

Understanding DL Models

5. Principles for Understanding Deep Learning (DL) Models

We now introduce the principles we explore to understand DL models in detail. Namely, these are the **Generated Input Data** and **Gradient-Based Attribution** principles. We split this section into input and output based methods, following the structure displayed in Figure 1.2. Generated Input Data uses the forward pass of a DL model to understand the model. Here, domain knowledge about the data properties is used to change the input data. Then, the model's output is observed. This principle thus belongs to the input category. Gradient-Based Attribution uses the backward pass of a DL model: It applies a custom gradient to the output of the model. It thus proceeds from the network's output to its input, belonging to the output category. Domain knowledge is then used to interpret the results.

5.1. Input Principle: Generated Input Data

Generated Input Data

Idea Using generated/synthetic data to analyze the influence of certain (high-level) input features onto the output of the trained model.

Goal Identifying the high-level properties that influence the output of the model and quantifying their influence.

Domain Knowledge used How to generate data that imitates the input data and can be changed w.r.t. the desired features?

Task Requirements None, however, the analyses conducted on the observed outputs are task-specific (e.g., change in accuracy in classification tasks or magnitude of change in regression tasks).

Data Requirements Every modality might be used as long as the input data can be generated in a controllable way w.r.t. the higher-level features that are of interest. Also, depending on the desired method to analyze the output (e.g., correlations of input change to output, change in evaluation metric), the parameters of the generation process might need to have an order.

When trying to understand the influence of different factors on people, researchers usually conduct a controlled study, where all factors are kept constant except for the one

5. Principles for Understanding DL Models

they are interested in. This is a standard procedure in many fields, such as psychology, sociology, and medicine [19]. In computer science, this method is often referred to as A/B testing. With only one changing factor, the change in outcome (e.g., medication treatment results, survey answers, or click statistics on websites) can then be directly attributed to the manipulated factor. The amount of change in the outcome is then a measure of the importance of the manipulated factor. We transfer this idea to DL models by altering their inputs in one property and observing the change in the output. In contrast to humans, DL models are deterministic and produce their outputs independently, which allows us to use the same model with different inputs.

In order to have full control over the input, we propose to use synthetic data, created using a controllable generation process, which only changes one property of interest. The property of interest can be categorical (e.g., type of object that is shown in an image), ordinal (e.g., comparative forms of adjectives), or numerical (e.g., position of an object in an image). The generated data is then fed through the model and the change in the output is observed. From this, one can obtain insight into how sensible the network is to the property of interest. Similar to this method being used for human studies, the implementation of the **Generated Input Data** principle is highly dependent on the input modality and task at hand.

Both, text and images, can either be very structured or natural in terms of their content. Structured text can be well-defined and follow a specific grammar such as programming languages, while natural text can be anything from a tweet to a novel. In the same way, images can be either very structured, such as a generated OpenStreetMap (OSM) image, or natural, such as a photo of an object in possibly arbitrary lighting, position, and orientation. Structured data can often be generated more easily, since there are less degrees of freedom that need to be taken into account. Natural data, on the other hand, is often more difficult to generate. For texts, this can be done by manually writing texts or operating on existing texts by e.g., deleting or repeating words or exchanging words and phrases of interest with synonyms [187]. For structured images, the oftentimes used generation process can be used to generate new images. Natural images can be manipulated using image editing software such as Photoshop. For the generation of natural images, more advanced software such as 3D modeling software can be used to generate images of objects in arbitrary positions and orientations. Also, Generative Adversarial Networks (GANs) can be used to generate images [74]. The controllability of the generated output is here not directly assured, but some works show that there are units in GANs that encode specific image properties that can be used to alter the image in that property [282]. Recently, also diffusion models became popular for image synthesis [49]. We concentrate on images in our implementations and create structured data (OSM map images) and natural images (car images) using image editing and 3D modelling software, respectively.

The Generated Input Data principle is also very task-dependent, since the analyses that can be done on the output depend on the task. For example, for a classification task, one can observe the change in the classification result when changing specific properties of the input, but also the difference in output probabilities for all classes [106]. For regression, the difference between the prediction on different variations can indicate how

much the model relies on the property of interest and how much it changes with larger variation. Given multiple characteristics of an ordinal or numerical property, one can also observe the change in the output when changing the property in a certain direction. In cases of numerical properties, correlations can be calculated or the output can be plotted against the property of interest, which can give insights into how much the property of interest matters. We do this in our analysis of the MapLUR model in Chapter 6, where we investigate the influence of the position of streets in OSM map images on the output of a Convolutional Neural Network (CNN). For representation tasks such as found in Deep Metric Learning (DML), the change in a property might induce a change in the representation of the input, which might be measured using the distance between the representations of the original and the manipulated input. Also, since the representations are often used for item retrieval, the change in retrieval quality when changing the property of interest can be observed. If the retrieval quality changes significantly, this can indicate that the property of interest is important for the model to represent the input. In Chapter 7, we propose to use a metric that measures the clustering capabilities of the embeddings given the different properties of the input. If a property is important for the model to represent the input, changing the property should induce a change in the clustering of the embeddings. The generation of independent images given the investigated properties allows us to make this observation.

We present two concrete implementations of the Generated Input Data principle. First, in Chapter 6, we investigate the influence of certain entities and properties of OSM map images on the output of a Land Use Regression (LUR) model that estimates the air quality of a location. For this, we synthetically generate OSM map images with different properties and observe and visualize the change in the output of the model. Second, in Chapter 7, we investigate the influence of different properties (e.g., model, rotation, or color) of car images on a DML model using 3D renders. Both implementations aim to better understand how certain properties influence the model output, giving insights into the model's behavior and how it might be improved.

5.2. Output Principle: Gradient-Based Attribution

Gradient-Based Attribution

Idea Using the gradient of the output w.r.t. the input to analyze the influence of certain input features onto the output of the trained model.

Goal Identifying (low-level) input features that influence the output of the model.

Domain Knowledge used How can low-level features with high influence be interpreted in the current task and data setting?

Task Requirements The gradient that is propagated back to the input needs to be chosen carefully for each task. This gradient can be based on a certain output of the model or designed by hand.

Data Requirements The input feature positions need to be interpretable, e.g., pixels in an image or words in a text.

In this section, we introduce the **Gradient-Based Attribution** principle, which is mainly applied to the output of the model. The **Gradient-Based Attribution** principle is based on the idea that if an input feature is important for the output of the model, then a small change in this feature should have a large impact on the output. It makes use of the fact that DL models are differentiable. We make the idea clearer by comparing the methodology to the usual training of such models. When training DL models, we feed an input through the network and compute a loss term, which measures the divergence from a desired target value. Then, the gradients of the loss function are computed w.r.t. the weight parameters of the model. Large gradients for a weight means that it needs to be changed more than a weight with small gradients in order to change the loss value. For attribution methods, the idea is that instead of computing gradients to the weights, we compute them to the input features. The loss can also be chosen differently, depending on what the gradients should present. It is even possible to directly create gradients that are fed to the output of the network. For example, for classification tasks, the gradient of the logit of the correct class w.r.t. the input is used. Large absolute gradients for an input feature then means that changing them leads to a large change in the loss value. In the classification setting, a large absolute gradient for a feature mean that the model's output changes its output probability for the correct class. This, in turn, indicates that this feature and its value are important for the model. We can thus use the input gradients as a measure of importance for the model's output, which can be visualized by coloring the input features according to their gradient value. Such visualizations are called attribution maps. Domain knowledge can then be used to interpret them.

The principle is universal for all DL models, but needs task- and data-specific implementations. The most methods for this principle are implemented for classification, which we also use in our implementation in Chapter 8. Regression tasks are not that common, but there are works that use this principle to understand what input features let the

5.2. Output Principle: Gradient-Based Attribution

prediction go up or down [256]. For representation tasks, there are some methods based on this principle, but they often dictate a specific Neural Network (NN) architecture or highlight the pixels in a pair of images that let them be close together (see Section 4.1.2). We thus propose in Chapter 9 a new attribution map approach for DML models, which only requires one input image and no specific architecture.

The general principle also needs to be adapted to the input modality. For images, the gradient is given for all pixels and their channels. Therefore, usually, an aggregation is done across the channel dimension in order to obtain a two-dimensional attribution map, which only highlights locality importances. For text, the gradients are calculated for all dimensions of all token embeddings. Thus, they are aggregated across the embedding dimension to obtain a token-based attribution sequence. We use this principle in our implementations for images and text and show common postprocessing steps of the gradients in order to obtain attribution maps. In our two implementations, we first apply the already established method “Integrated Gradients” in a text classification scenario and show why the model predicts certain outputs based on words present in the input in Chapter 8. Then, in Chapter 9, we adapt the principle to DML and propose a novel attribution map generation method together with some comparison metrics in order to investigate the learned features of DML models on a pixel level.

6. Analyzing a Land Use Regression Model using Generated Map Images

Our first implementation uses **Generated Input Data** to better understand a Deep Learning (DL) model that estimates air pollution given map images and mainly follows the analysis section of our work in [256]. The estimation of air pollution concentrations is important, since it is known to have adverse effects on human health and the environment [30, 55] and monitoring stations are only sparsely deployed in many cities. The estimation of such concentrations is thus necessary to assess whether the pollution levels are still within acceptable and legal limits. LUR describes the task to assess the air quality by using land-use, population, or road features to predict pollution concentrations with models such as Linear Regression (LR) or Random Forests (RFs) [92, 22, 300]. In [256], we propose a DL model — more specifically a Convolutional Neural Network (CNN) — called MapLUR that estimates pollution concentrations in areas without monitoring stations. MapLUR implements the “Data-driven, Open, and Global” (DOG) paradigm for LUR models that we also propose in [256]. It automatically extracts features from map images (data-driven), which are openly available (open) almost anywhere on earth (global), and estimates air pollution based on these features.

Setting MapLUR is a CNN (see Section 2.2.3) whose structure is depicted in Figure 6.1. The input to the model is an OpenStreetMap (OSM) image with the location of interest in the center. The images cover approximately 80 m by 80 m around the location of interest and are scaled to size 224 px by 224 px, following the input size to popular CNN architectures for the ImageNet competition [46]. Given this image, the model estimates the pollution concentration at the location of interest, taken from the London Atmospheric Emissions Inventory (LAEI) dataset, which was split such that no images from the training, validation, and test sets are overlapping.

The MapLUR model architecture contains 15 convolutional layers with batch normalization [108] and the Rectified Linear Units (ReLUs) [195] activation function. The last convolutional layer is followed by a Multilayer Perceptron (MLP) with two layers, 128 neurons in the hidden layer, and ReLU activation. The output is a single neuron with linear activation that produces the estimated pollution concentration using the Mean Squared Error (MSE) loss function (see Section 2.4.2). Each convolutional layer has 16 filters, a kernel size of 3, a padding of 1, and a stride of 1. The output size of these layers is the same as their input size. Maximum pooling layers with a kernel size of 2 and stride of 2 are applied after the ReLUs of the first, third, fifth, seventh, tenth, and thirteenth convolutional layer in order to reduce the number of activations. We found this architecture and the corresponding hyperparameters by evaluating different variations of

6. Analyzing a Land Use Regression Model using Generated Map Images

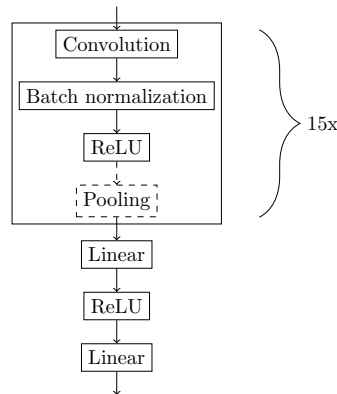


Figure 6.1.: Structure of MapLUR. The model consists of 15 feature-learning building blocks which contain a convolution layer, batch normalization, rectified linear units (ReLU), and sometimes a pooling layer. These building blocks are concatenated and only the first, third, fifth, seventh, tenth, and thirteenth block contain a pooling layer. These blocks are followed by a simple fully connected layer with ReLU activation and finally a single fully connected neuron with linear activation which returns the estimation of the pollution at the given location.

the model using tenfold cross-validations on the training set.

MapLUR is trained using the Adam optimizer [127] on batches of size 400 for at most 2000 epochs with a learning rate of 0.0001. We augment the training data by flipping or transposing the images. Additionally, we employ early stopping, interrupting training when the validation performance has not increased for 20 consecutive epochs.

In order to evaluate MapLUR, we compare our model to baseline models which are commonly used in LUR: always predicting the mean training label, training a LR, RF, and a MLP. For the baseline models, we use a set of standard land-use and road-related features, which have been shown to be important influencing factors for air pollution [52]. These are computed in different radii around each datapoint. All of these features can be calculated from OSM data, since we want to provide similar information to all models for a fair comparison. The features include the areas of commercial, industrial, and residential land-use, the lengths of big and local streets, and the distances to the next traffic signal, motorway, primary road, and industrial premise. For all but the mean baselines, we use a feature selection procedure to select the most relevant subset of features [52].

All models are evaluated using standard metrics for the evaluation of LUR models, namely R^2 and Root Mean Squared Error (RMSE) (described in Section 2.5.2). The average performance over 40 initializations is reported to counteract unfortunate initialization results. Additionally, the sample of 40 evaluation runs can be used as the input to statistical significance tests to formally confirm differences in evaluation results. For more information on the baseline models and the experimental setup, see [256].

The results in Table 6.1 show that MapLUR, which is trained on OSM images, performs better than all baselines regardless of metric. Based on this observation, we are now interested in the features the model has learned to extract from the input map images.

Table 6.1.: Results of baseline methods and experiments. MapLUR, which is trained on OSM images, is providing the best performance overall, beating all baselines. The RF model is outperforming all other baseline models on this dataset which makes it the best baseline. All results are significantly different to each other.

Model	R ² ↑	RMSE [$\mu\text{g m}^{-3}$] ↓
Mean baseline	0.000	13.971
LR	0.487	10.004
MLP	0.499	9.887
RF	0.662	8.119
MapLUR	0.673	8.002

6.1. Methodology

We now describe the task- and modality-specific implementation of the **Generated Input Data** principle. In this case, we want to generate OSM images using a controllable generation process, such that we can observe the changes they induce on the estimated air pollution concentration.

The leading question in developing LUR methods in previous work is: *What entity of what area in what distance to the center is contributing to the pollution?* From this, we infer three categories of features that we want to investigate: entity features, area features, and distance features.

6.1.1. Data Generation

To generate OSM images, we take advantage of the well-defined structure of map images with different color-coded entities. Map images therefore can easily be recreated using graphic editing software, which makes it possible to create artificial OSM images for which we can control the features separately while keeping all other features fixed.

Entity Features

Entity features describe *what* is seen on the image. Entity features are often used for the estimation of air pollution, as, for example, industrial areas are usually contributing more to air pollution values than parks. For these features, we investigate how certain entities are influencing the model estimate. We build two kinds of images: On the one hand, we create images that are each completely covered by one specific type of entity, resulting in uniformly colored square images. On the other hand, the same images are then overlaid by the depiction of a motorway and a trunk road. We expect that different underlying entities provide different estimates according to the usual presence of sources for NO₂ pollution. We also expect an increase in the air pollution estimate whenever a road is added to the underlying entity. Depending on the type of road this increase might fluctuate.

Area Features

Area features describe *how large* a given entity is in the image. The area that an OSM entity has on an image should contribute to the estimated pollution value. For these features, we use the trunk road entity to analyze the influence of the area. We build multiple images that contain a straight road that goes top to bottom or left to right through the center of the image. As the background we always use the same neutral background that depicts general land-use in OSM. We then vary the width of that street either horizontally or vertically, depending on its orientation. A linear increase in the street's width is equivalent to a linear increase in the street's area.

Distance Features

Distance features describe *how far away* a given entity is from the image center. For this analysis, we create images that contain only one straight trunk road that is then moved vertically or horizontally, depending on the direction of the street. With this setup, we can control the distance of the motorway to the center of the image while fixing the area and entity features. We expect that the model produces higher estimates for images where the street is closer to the center, since the desired value from LAEI is coming from a 20m subframe of the image which is in the image center. The model therefore should have learned a tendency to weight features from the center of the image more than from the borders.

6.1.2. Output Interpretation

Since the model solves a regression task, the output of the model is a continuous value that inherently has an order. This value is directly interpretable as the estimated air pollution concentration, so we are able to directly interpret the effects on the output values when changing the input features.

Since for area and distance features, the high-level properties of the input images (the area and the distance of certain entities), has an inherent order to them, we can plot the resulting pollution estimates as a function of the input features. This allows us to qualitatively observe the influence of the input features on the output values. Estimating smooth approximating functions that fit the data can help with understanding the proportionality of the input features to the output values. Assuming a linear dependency between the input features and the output values, we can also compute the Pearson correlation coefficient to quantify the proportionality of the input features to the output values.

6.2. Experiments

We conduct three analysis experiments: One for each of the three categories of features. Given the data generation processes for each high-level feature, we feed the generated images through the model and observe the resulting pollution estimates. We then plot the

Table 6.2.: Model estimate for a given OSM entity. The entities span across the whole image and they are overlaid with different types of roads. Overlaying a road with another road does not make sense so these values are omitted. The “neutral” entity is a background that is used by OSM for indicating land with no particular land-use. All estimates are in $\mu\text{g m}^{-3}$.

Road type	Entity Name								
	industrial area	residential area	commercial area	park	forest	water	neutral	motorway	trunk
no road	37.71	38.29	38.72	38.87	39.27	41.66	42.06	47.23	80.63
trunk	61.70	50.94	59.73	64.14	57.94	59.62	58.62	—	—
motorway	60.04	48.48	64.18	46.05	54.21	53.45	55.00	—	—

resulting pollution estimates as a function of the input features and analyze the resulting plots in a qualitative manner.

6.3. Results

Table 6.2 shows the resulting pollution estimates by MapLUR for the entity features. All estimates exceed the World Health Organization’s recommended limit¹ for NO_2 long-term exposure of $10 \mu\text{g m}^{-3}$, which can lead to severe health conditions [30]. It also indicates that the general air pollution in the training data seems to exceed this limit. In general, different underlying entities do not lead to large differences in pollution estimates if there is no road. Completely covering the image by a motorway or trunk road result in the largest estimates. Additionally, trunk roads seem to have a much higher impact on the air pollution estimate than motorways. Adding a trunk road or motorway to any entity increases the air pollution estimate as expected. The amount of increase depends on the underlying entity of the map and what kind of road is present. This shows that the relationship of the entities that are visible in the map image are also important. There seem to be complex correlations between different entity features, which cannot be modeled easily in simpler models like LR. Surprisingly, adding a trunk road to a park entity leads to a high estimation, while adding a motorway leads to a relatively low estimation, compared to the other entities. We suspect that in the training data, trunk roads or motorways going through parks rarely occurs, which results in relatively unintuitive estimates by the model when such combination is presented.

For the area features, Figure 6.2 shows some of the artificial OSM images as well as a plot of MapLUR’s output given the street width in pixels. As expected, an increasing width — and therefore an increasing area — of the street tends to increase the pollution estimate. Both horizontal and vertical growth have very similar curves that are not linear but instead seem to be more logarithmic. The similarity was expected, as during training, the images are augmented by rotation and flipping such that the direction of streets should not have any impact on the overall output.

Concerning distance features, Figure 6.3 shows image samples and the resulting estima-

¹see <https://www.who.int/publications/i/item/9789240034228> for 2021 (last accessed: 2023-08-17)

6. Analyzing a Land Use Regression Model using Generated Map Images

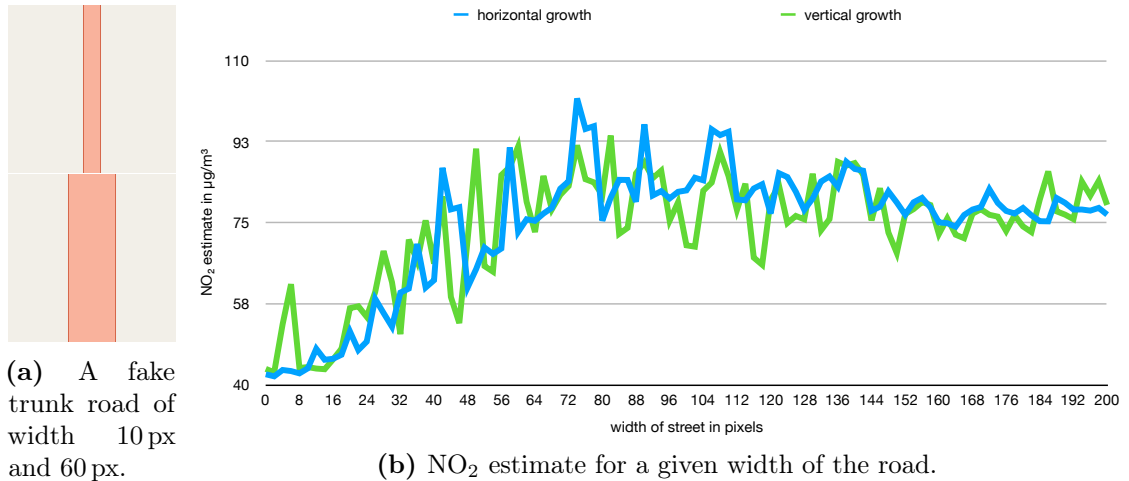


Figure 6.2.: Varying the width/area of the street while keeping other features such as distance to the center and type of street fixed.

tion curves when moving the trunk road farther away from the center of the image. As expected, the proximity of a street to the image’s center contributes to the overall NO₂ estimate positively. Pearson correlations of the distance with the estimated values are always lower than -0.6 , indicating a relatively strong negative correlation. The curves that are shown are also not linear and can be better fitted by polynomials with a squared feature term than by a line. To capture this non-linearity, more sophisticated methods than LR need to be used, which justifies the use of RFs or Neural Networks (NNs).

6.4. Conclusion

We have analyzed MapLUR, a CNN based LUR model for air pollution estimation, which automatically extracts useful features from OSM images, which are openly and globally available. Using a concrete implementation of the **Generated Input Data** principle, we have analyzed the factors that influence the prediction of this model, finding that the automatically extracted features strongly relate to typical manually engineered features for LUR models. However, instead of linear dependencies between the features and the target variable, MapLUR uses non-linear functions to model their relation, which might be a way to improve LR models: Train MapLUR on a large dataset of data such as the LAEI dataset. Then, analyze the model’s behavior using our proposed analysis for the features that are used in the LR model. Since the features of interest are altered in isolation, finding non-linear dependencies between the features and the target variable can be used to “linearize” the features for the LR model. For example, since the area features for a street seem to influence the model output logarithmically, taking their exponential could make them more linearly influencing. Given such transformations and that linear models with hand-engineered features need much less data than MapLUR, this would make it possible to use LUR models in areas with little data.

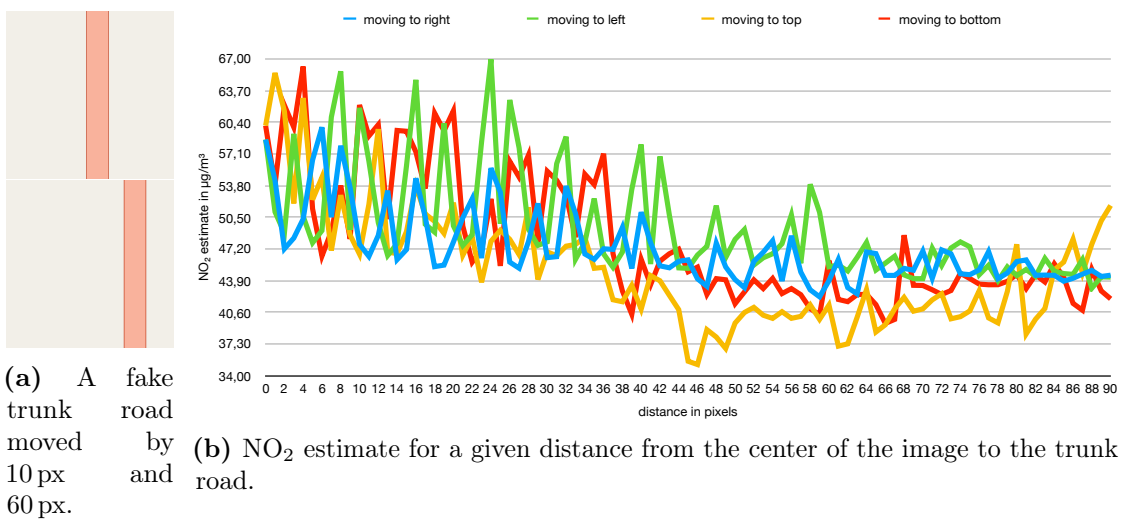


Figure 6.3.: Varying the distance of a trunk road to the image center pixel by pixel while keeping other features fixed.

7. Analyzing Deep Metric Learning Models using Generated Car Images

In this section, we use the proposed principle of using **Generated Input Data** to compare what different DML models have learned to extract from the input data in order to produce an embedding. Since each of the models was trained on a dataset of car images, we utilize domain knowledge about how car images can look and generate synthetic car images using a 3D modelling software, which are fed through the models. Here, we can control different aspects of the generated images, such as the camera angle, the background, the lighting, and the car model. To quantitatively compare the different models, we introduce a new metric, which measures the influence of a feature on the clustering capabilities of the model. An overview of this idea is shown in Figure 7.1.

The following sections mainly follow the second part of our work in [134]. First, we motivate and explain the setting as well as describe the different models we compare. We then describe the data generation process for the synthetic data. After that, we introduce the metric we use to compare the models. Finally, we present the results of our experiments and discuss them.

Recall that in DML, a Neural Network (NN) is trained to map input items to m -dimensional embedding vectors, that should be close to each other if the corresponding inputs share a given class. Thus, the network has to learn to extract discriminating input features to embed an image. To achieve that, ranking based, classification based, and hybrid loss functions have been introduced. An overview is given in Section 4.2.4.

In recent studies, different DML loss functions were shown to lead to similar test performances if compared fairly [193, 231]. Musgrave et al. identify flaws in the evaluation setups of many DML papers and conduct a fair comparison between DML methods by testing several common loss functions with the same benchmark datasets, architecture, and test metrics. Their study finds very similar performances for all the tested losses. In general, research shows that even with similar performance, NNs might learn to focus on different [45] and sometimes even undesired input features [142] for their output.

In this implementation of the **Generated Input Data** principle, we analyze and compare what high-level features are paid attention to by NNs trained with common DML loss functions, concretely the 14 pretrained models provided by Musgrave et al. (shown in Table 7.1). We measure the influence of image properties, e.g., the rotation or color of an object, on the embeddings. Usually in DML, networks learn to differentiate one specific property, e.g., the car model for the Cars196 dataset [139], such that images of the same car model have similar embeddings and different car models are farther apart in embedding space. For testing, the network’s ability to cluster new car images regarding their model is measured. Due to its training objective, other properties such as a car’s

7. Analyzing Deep Metric Learning Models using Generated Car Images

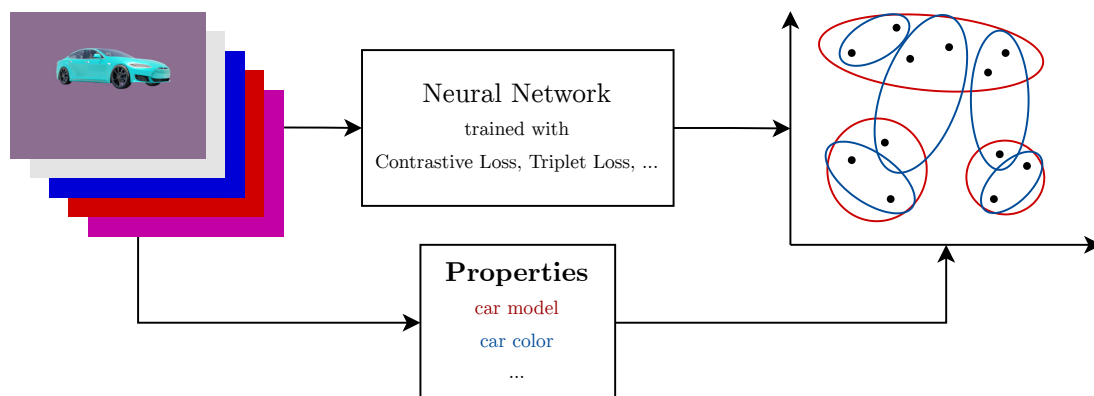


Figure 7.1.: Given a trained DML model that maps images to an embedding space. Our idea is to investigate the influence of image properties on the clustering behavior in the embedding space and compare them between loss functions.

color or environmental illumination should have minimal influence on the embedding, since the dataset contains images of the same car model in different colors and in different lighting conditions. If the network makes use of a property to output an embedding, images of the same property are likely to be clustered as well, potentially less pronounced. To ensure the properties are not correlated, we generate a large image dataset consisting of photo-realistic car renders. As we will argue in Section 7.1.2, measuring the clustering behavior with the common metric R-Precision depends on the number of possible property values. We thus propose a property-independent extension, *Normalized R-Precision*, that enables the comparison of multiple properties.

Setting For our analysis, we use trained models provided by Musgrave et al., who train a BatchNorm Inception network [108] with 14 DML loss functions and compare their performance [193]. Table 7.1 lists all used losses with their loss type and distance/similarity measure used. We have introduced a representative subset of those loss functions in Section 2.4.3. Each network outputs an 128 dimensional embedding per image and is trained and evaluated on Cars196 [139], CUB200 [275], and Stanford Online Products (SOP) [253] datasets. For the image property analysis, we only analyze models trained on the Cars196 dataset, since we loosely imitate this dataset using generated car images. In Chapter 9, we will perform a pixel level analysis, where we use all three datasets.

For each loss, four trained models are provided, one for each fold of a four-fold cross validation performed to optimize hyperparameters using a Bayesian optimizer. All models are trained under the same conditions and the test results for all folds are averaged. We also report average results, since for all folds, the results are very similar. More information about the training setup and best hyperparameters of the used models can be found in [193]. In addition, we also add an untrained model “None”, which is initialized using weights from an ImageNet [46] classifier, while the last layer is initialized with random weights [193].

Table 7.1.: Analyzed losses taken from [193]. Method “None” has the same architecture but without any training, thus only using weights from the feature extractor pretrained on ImageNet.

Method	Year	Loss type	Distance/Similarity
Contrastive [80]	2006	Ranking	Euclidean Distance
Triplet [288]	2006	Ranking	Euclidean Distance
NTXent [252]	2016	Ranking	Cosine Similarity
ProxyNCA [190]	2017	Classification	Squared Euclidean
Margin [299]	2017	Ranking	Euclidean Distance
Margin / class [299]	2017	Ranking	Euclidean Distance
Normalized Softmax [159, 276, 312]	2017	Classification	Dot Product Similarity
CosFace [277, 278]	2018	Classification	Cosine Similarity
ArcFace [47]	2019	Classification	Cosine Similarity
FastAP [32]	2019	Ranking	Squared Euclidean
SNR Contrastive [310]	2019	Ranking	SNR Distance
Multi Similarity [284]	2019	Ranking	Cosine Similarity
Multi Similarity + Miner [284]	2019	Ranking	Cosine Similarity
SoftTriple [212]	2019	Classification	Cosine Similarity
None	—	None	—

7.1. Methodology

We now describe the methodology used to analyze the trained models. Based on the **Generated Input Data** principle, we need to generate input data using an interpretable and controllable data generation process. The data is then fed through the trained model and the output changes are observed and interpreted. The main idea is to measure the clusterability of the embeddings given a fixed property. If the model uses a certain property when embedding the input, changing said property should change the embedding. Also, changing properties that are not picked up by the model should not change the embedding that much. This leads to clusters of item embeddings with the same property. We now aim to measure this clusterability that can be used to compare different models and properties.

7.1.1. Notation

For a given loss function $L \in \{L_{\text{contrastive}}, L_{\text{triplet}}, \dots\}$, a NN $f_L : \mathcal{I} \rightarrow \mathbb{R}^m$ maps an input image \mathbf{I} from the dataset $\mathcal{I} = \{\mathbf{I}_1, \dots, \mathbf{I}_n\}$ to the m -dimensional embedding space. This results in the embeddings $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ with $\mathbf{x}_i = f_L(\mathbf{I}_i)$ for $i \in \{1, \dots, n\}$. Each image has properties; property k has the possible values \mathcal{A}_k . The value of image \mathbf{I}_i ’s property k is $a_k(\mathbf{I}_i) \in \mathcal{A}_k$. One property $a_{\text{class}}(\mathbf{I}_i) \in \mathcal{A}_{\text{class}}$ is the class of the image \mathbf{I}_i that is used to define if two images are similar to each other while training (if $a_{\text{class}}(\mathbf{I}_i) = a_{\text{class}}(\mathbf{I}_j)$). A loss-specific distance function $d_L : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}_0^+$ calculates the distance for two embeddings, e.g., the Euclidean distance. While d_L can also be a

7. Analyzing Deep Metric Learning Models using Generated Car Images

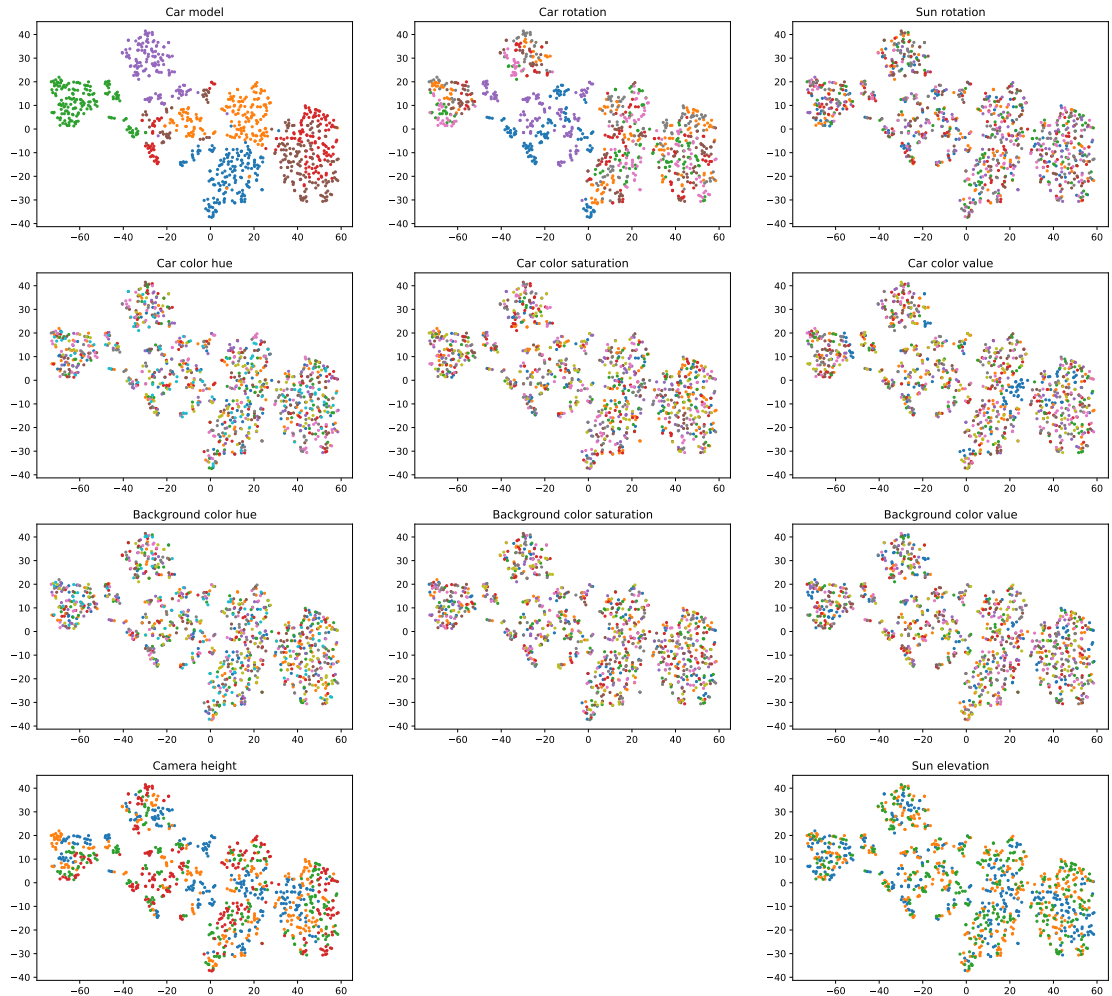


Figure 7.2.: 1000 embeddings from a Cars196 model with Contrastive Loss visualized with t-SNE [270]. Color denotes different property values.

similarity measurement, e.g., cosine similarity, that should be maximized between similar embeddings, we assume it to be a distance metric for brevity.

7.1.2. Normalized R-Precision

We now want to measure the clusterability of the embeddings that are generated from a set of generated images. Generated images vary in certain properties, describing concepts like object form, color, or orientation. We investigate the question “What image properties influence the model output?”. Each input image has a set of properties and their values. A property with high influence on the embedding fulfills the two clustering objectives: First, fixing this property and changing all other property values should result in small deviations in embedding space. Second, changing the property while keeping everything

else fixed should result in large deviations in embedding space. This idea is used in common evaluation metrics in DML, but are only applied to the image’s “class” property. A DML NN is considered to work well, if it maps test images with the same class to similar locations in embedding space, while embedding images with different classes to different locations. For Cars196, a test class is a certain *car model*, for which many different images from different angles, car colors, etc. exist in the dataset. Instead of the car model “class”, we use other image properties such as the car’s color or the car’s orientation. A list of all properties that we investigate is shown in Table 7.2. Even though the NN has not been trained on these types of data splits, we can still measure the closeness of the resulting embeddings regarding the defined property. If, for example, grouping embeddings by car orientation shows well-defined clusters, we can conclude that changing the orientation has significant effect on the network’s output. If the network is invariant to the car’s orientation, changing it does not significantly alter the embedding vector, thus showing no clustering behavior. Clustering examples for different properties are shown in Figure 7.2. Here, *car model* is clustered well which shows that the model uses this property as a discriminating feature for its embedding output. *Car rotation* shows local clusters, thus still having an influence on the embedding. Other properties such as *sun rotation* have no influence and are not clustered at all. Our goal is to quantify this clusterability of the embeddings.

In order to measure the clustering behavior of properties, we propose to use the common DML metric R-Precision as the base. We have introduced R-Precision in Section 2.5.3, but generalize it now to work with multiple properties. Intuitively, for one query embedding \mathbf{x}_q and a property k , the set of the closest $R_{k,q}$ embeddings from the dataset are retrieved. $R_{k,q}$ is the number of images with the same property value $a_k(\mathbf{x}_i)$ in the dataset. This set is called $\mathcal{F}_{k,\mathbf{x}_q}^R$. Formally, it is defined as $\mathcal{F}_{k,\mathbf{x}_q}^R \subset \mathcal{X} \setminus \{\mathbf{x}_q\}$ subject to $|\mathcal{F}_{k,\mathbf{x}_q}^R| = R_{k,q}$ and $\forall \mathbf{x}_i \in \mathcal{X} \setminus \mathcal{F}_{k,\mathbf{x}_q}^R$ it holds:

$$d_L(\mathbf{x}_i, \mathbf{x}_q) \geq \max_{\mathbf{x}_j \in \mathcal{F}_{k,\mathbf{x}_q}^R} d_L(\mathbf{x}_j, \mathbf{x}_q). \quad (7.1)$$

The R-Precision for property k (R-Prec $_k$) is then defined as

$$\text{R-Prec}_k = \frac{1}{n} \sum_{q=1}^n \frac{|\{\mathbf{x}_i \in \mathcal{F}_{k,\mathbf{x}_q}^R \mid a_k(\mathbf{I}_i) = a_k(\mathbf{I}_q)\}|}{R_{k,q}}, \quad (7.2)$$

i.e., the average fraction of items having the same property value. This metric measures how well item embeddings with the same property value are spatially separated from items with different property values. To illustrate this, imagine there are 100 items in the dataset \mathcal{X} . From this dataset, we choose one query item embedding \mathbf{x}_q . $R_{k,q} = 10$ other items in the dataset have the same property value as \mathbf{x}_q . We thus retrieve the ten nearest neighbors for \mathbf{x}_q . Pretend that five out of these ten are items with the same property value as \mathbf{x}_q . The fraction in Equation (7.2) is then $\frac{5}{10} = 0.5$, since only half of the closest embeddings have the same property value as the query, even though ten could be possible.

7. Analyzing Deep Metric Learning Models using Generated Car Images

Equation (7.2) averages these results for all possible query embeddings, i.e., the whole dataset.

The higher the R-Precision for a certain property, the better the embedding clusters w.r.t. to this property. Altering the property thus significantly changes the embedding vectors, while the network is less influenced by other properties. However, R-Precision (for any property k) depends on the number of property values: Given a random embedding and only two possible property values with the same number of items, the expected R-Precision is 0.5. For a property with ten possible values, the expected R-Precision score is 0.1. Thus, an absolute comparison between properties is not possible, as random embeddings would score differently. Therefore, we propose to apply a normalization step to the R-Precision calculation. With randomly generated embeddings for all n images, the number of images with the same property value of the property k as the query embedding \mathbf{x}_q is binomially distributed. For the metric calculation, we take $R_{k,q}$ samples. There is a probability of $p_{k,q} = \frac{|\{\mathbf{x}_i \in X | a_k(\mathbf{I}_i) = a_k(\mathbf{I}_q)\}|}{n}$ that a close embedding has the same property value. We use the mean $\mu_{k,q} = R_{k,q} \cdot p_{k,q}$ and standard deviation $\sigma_{k,q} = R_{k,q} \cdot p_{k,q} \cdot (1 - p_{k,q})$ of this binomial distribution to normalize the R-Precision calculation per query embedding. We obtain the *Normalized R-Precision (NR-Prec)*:

$$\text{NR-Prec}_k = \frac{1}{n} \sum_{q=1}^n \frac{|\{\mathbf{x}_i \in \mathcal{F}_{k, \mathbf{x}_q}^R | a_k(\mathbf{I}_i) = a_k(\mathbf{I}_q)\}| - \mu_{k,q}}{\sigma_{k,q}}. \quad (7.3)$$

Normalized R-Precision is zero if the clustering is as good as for random embeddings. The larger the deviation from zero, the lower the probability of this clustering being due to randomness. Due to the normalization, we gain two advantages over R-Precision: On the one hand, we can now compare properties with different numbers of possible values, allowing us to sort different properties by how much the model is influenced by them. Normalized R-Precision results and their ranking can also be compared between models in order to check if different loss functions pick up similar high-level features. On the other hand, it is possible to measure statistical significance. Given a sufficiently large dataset, the normalized binomial distribution approximates a normal distribution, so if Normalized R-Precision exceeds 2.576, the embeddings locations are significantly different from random embeddings with a 1% significance level.

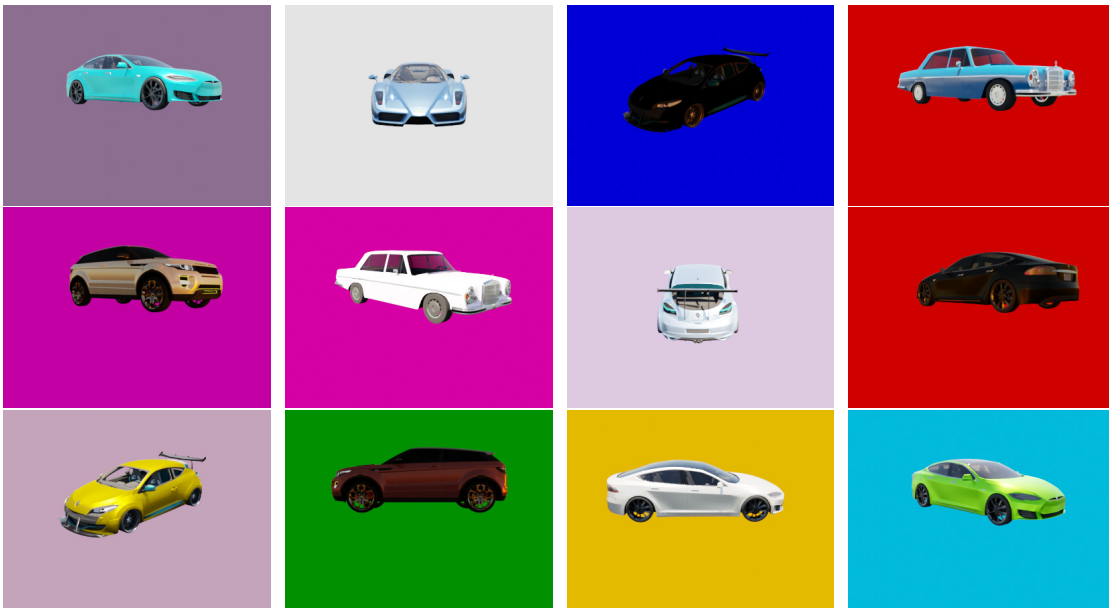
It is important that properties in the dataset used to calculate Normalized R-Precision are chosen independently for each image to correctly measure the property importance. Imagine that the dataset consists of car images where each car model has its own specific color. If the network is trained to cluster the car model, the high correlation between car model and color leads to well-clustered embeddings for the color, even though the network might only attend to car model features.

7.1.3. Data Generation

In order to make sure that properties are independent of each other for each image in our test dataset, we create photo-realistic 3D renders of cars, loosely imitating the Cars196

Table 7.2.: All properties and corresponding possible values in the car renderings. Combinations were chosen uniform at random.

Property	Possible Values
Car model	Ferrari Enzo, Mercedes Benz 300sel, Megane RS, Mercedes AMG Coupe, Range Rover Evoque, Tesla Model S
Car rotation	0°, 45°, ..., 315°
Car color	
Hue	0.0, 0.1, ..., 0.9
Saturation	0.0, 0.25, ..., 2.0
Value	0.0, 0.25, ..., 2.0
Background color	
Hue	0.0, 0.1, ..., 0.9
Saturation	0.0, 0.25, ..., 2.0
Value	0.0, 0.25, ..., 2.0
Camera height	0.5, 1.5, 2.5, 3.5
Sun elevation	0°, 45°, 90°
Sun rotation	0°, 45°, ..., 315°

**Figure 7.3.:** Sample images of our generated car dataset. We vary eleven properties, such as the model, lighting, and colors.

7. Analyzing Deep Metric Learning Models using Generated Car Images

dataset. Besides the *car model*, we alter properties such as the *car’s color*, *rotation*, and *illumination*. Table 7.2 shows all altered properties with their possible values. We sample 100 000 from all possible combinations (overall 3 023 308 800) uniformly at random to ensure all splits have similar sizes and independent property value choices. Figure 7.3 shows samples of the dataset that we generate using Blender [44]. While many other properties can be modelled (e.g., camera parameters, more complex backgrounds, etc.), we restrict ourselves to these ten properties in addition to the model itself, which the networks are trained on.

7.2. Experiments

The images of our generated dataset are fed through all fourteen tested models and the Normalized R-Precision is computed for each of the eleven properties. The resulting values are comparable, with higher values meaning that the property is more important for the network.

7.3. Results

	Car		Car Color			Background Color			Sun			
	Model	Rotation	Hue	Saturation	Value	Hue	Saturation	Value	Camera Height	Elevation	Rotation	
Ranking	Contrastive	<u>58.32</u>	<u>39.60</u>	<u>3.02</u>	<u>3.28</u>	<u>4.72</u>	1.52	1.30	2.09	<u>20.92</u>	<u>8.23</u>	0.85
	Triplet	<u>57.36</u>	<u>37.37</u>	<u>3.25</u>	<u>3.26</u>	<u>4.69</u>	1.63	1.43	2.50	<u>19.50</u>	<u>8.22</u>	0.74
	NTXent	<u>57.87</u>	<u>38.22</u>	<u>3.44</u>	<u>3.30</u>	<u>4.54</u>	1.57	1.46	2.18	<u>20.32</u>	<u>8.55</u>	0.78
	Margin	<u>57.91</u>	<u>38.58</u>	<u>3.50</u>	<u>3.18</u>	<u>4.76</u>	1.59	1.29	2.04	<u>21.21</u>	<u>8.30</u>	0.78
	Margin / class	<u>58.92</u>	<u>39.65</u>	<u>3.41</u>	<u>3.36</u>	<u>4.83</u>	1.89	1.72	2.25	<u>21.55</u>	<u>8.82</u>	0.76
	FastAP	<u>55.84</u>	<u>38.44</u>	<u>2.81</u>	<u>3.10</u>	<u>4.87</u>	1.05	1.24	1.81	<u>20.49</u>	<u>7.45</u>	0.71
	SNR Contrastive	<u>57.38</u>	<u>39.88</u>	<u>3.41</u>	<u>3.37</u>	<u>5.05</u>	1.69	1.56	2.22	<u>21.40</u>	<u>8.66</u>	0.82
	Multi Similarity	<u>59.81</u>	<u>41.03</u>	<u>3.18</u>	<u>3.37</u>	<u>4.95</u>	1.83	1.74	2.59	<u>21.14</u>	<u>8.73</u>	0.87
	Multi Similarity + Miner	<u>57.82</u>	<u>38.86</u>	<u>3.24</u>	<u>3.13</u>	<u>4.52</u>	1.84	1.52	2.03	<u>20.03</u>	<u>7.66</u>	0.77
	ProxyNCA	<u>57.68</u>	<u>37.64</u>	<u>4.73</u>	<u>3.93</u>	<u>5.72</u>	2.33	2.09	2.52	<u>19.82</u>	<u>9.77</u>	0.84
	Normalized Softmax	<u>57.26</u>	<u>37.76</u>	<u>3.85</u>	<u>3.97</u>	<u>5.44</u>	1.68	1.80	2.31	<u>19.43</u>	<u>8.81</u>	0.80
	CosFace	<u>56.50</u>	<u>38.51</u>	<u>4.00</u>	<u>3.65</u>	<u>5.32</u>	2.40	2.39	2.64	<u>19.10</u>	<u>8.72</u>	0.83
	ArcFace	<u>55.62</u>	<u>37.20</u>	<u>4.15</u>	<u>3.72</u>	<u>4.91</u>	<u>2.90</u>	<u>2.84</u>	<u>2.93</u>	<u>18.70</u>	<u>8.98</u>	0.78
	SoftTriple	<u>57.36</u>	<u>38.08</u>	<u>3.63</u>	<u>3.81</u>	<u>5.37</u>	1.81	2.00	2.25	<u>19.72</u>	<u>8.43</u>	0.77
None	<u>50.61</u>	<u>36.25</u>	<u>4.32</u>	<u>3.92</u>	<u>4.48</u>	<u>4.81</u>	<u>4.09</u>	<u>6.80</u>	<u>20.75</u>	<u>8.64</u>	0.75	
Ranking Mean	<u>57.91</u>	39.07	<u>3.25</u>	<u>3.26</u>	<u>4.77</u>	1.62	1.48	2.19	20.73	<u>8.29</u>	0.79	
Classification Mean	<u>56.89</u>	<u>37.84</u>	4.07	3.82	5.35	2.23	2.22	2.53	<u>19.35</u>	8.94	0.80	

Table 7.3.: NR-Precisions for the rendered car images. The higher the value (darker the cell shade), the less likely that the performance stems from randomly sampling neighbors. Significantly different values are underlined. We also give means for ranking and classification losses. There, bold text indicates that, on average, one loss type pays significantly more attention to this property than the other loss type.

Our experiments give the results in Table 7.3. All losses attend to the properties in the same order. The car’s *model* yields the most notable embedding clusters, which is not surprising, since all networks are trained to differentiate between car models. The *rotation*’s clusters likely stem from discriminating features being visible to the camera. For presumably similar reasons, the *camera height* shows good clustering as well. This might also be because only few Cars196 training images show the car from low perspectives. The *sun rotation* shows expectedly bad clustering behavior: Cars illuminated from many possible directions are seen during training. Surprisingly, the *sun’s elevation* has high

influence on the embedding. This might come from the training dataset consisting mostly of photos taken in daylight. A low *sun elevation* makes the light warmer and casts longer shadows. These influences on the image might be picked up by the DML models, since there are few training examples in this situation. The *background color* has negligible effect on the clustering, which is expected, since cars are pictured in many different environments. In contrast, the *car color* leads to embeddings significantly different from random embeddings, which is somewhat surprising, as each training *car model* is shown in multiple colors. We suspect that different colors make it more difficult to identify certain features. However, there might be small tendencies in the training dataset that the networks pick up on. For example, Ferrari cars are usually red, while Lamborghini cars are usually yellow.

We observe that the “None” baseline has the same property order as trained models. Compared to trained embeddings, the *car model* shows weaker clusters and the *background color* properties yield embeddings significantly different from random assignments. The network’s weights, except for the last layer, are initialized with trained weights from the ImageNet classification task. The network’s embedding therefore represents features that were important for image classification. For this task, the learned features are usually invariant to lighting conditions, but the environment can be a discriminating feature, e.g., the presence of water helps to identify ships [142]. Thus, the “None” network attends to the *background* properties more to generate embeddings. During finetuning on the Cars196 dataset, all loss functions guide the network to learn that the *background* is less important for embedding the *car model*.

When grouped by their loss type, we identify differences in Normalized R-Precision scores between ranking vs classification losses. We apply a Mann-Whitney U test [172] with a significance level of 1%, showing significant differences between classification and ranking based loss functions for all image properties except the *car model* and the *sun rotation*. While ranking loss functions show significantly larger influence of *car rotation* and *camera height*, classification based loss functions attend to the *car and background color* properties as well as the *sun elevation* significantly more than ranking losses.

7.4. Conclusion

Based on the DML objective to cluster images from the same class together, we would expect models to be invariant to unimportant features for the class, e.g., the car’s color, its orientation, or environmental illumination when trained to embed the car model. However, we have shown that the properties *car color*, *rotation*, *sun’s elevation*, and *camera height* have significant influence on the embeddings. Also, classification losses usually pay more attention to the background of images than ranking losses. Our proposed implementation of **Generated Input Data** serves as a tool to analyze what high-level features are learned by DML neural networks and to evaluate if they are invariant to unimportant properties. Our tools can be used to develop and evaluate methods that encourage invariance for undesired properties, e.g., [26]. Image augmentations like hue shifts, grayscaling, or skewing, could remove dependencies on the car’s color or camera heights/angles. While

7. Analyzing Deep Metric Learning Models using Generated Car Images

we have not investigated the influence of other camera parameters such as focal length or image properties such as contrast or brightness, these might also have undesired effects on the resulting embeddings. Correcting for such parameters methodologically is desirable. In Chapter 11, we will introduce a method to control for the image backgrounds and their influence on the embeddings, which uses the **Input Masking and Augmentation** principle.

Since we find differences between classification and ranking based methods, future work might analyze hybrid loss functions and find reasons for found differences. Besides losses, our method is able to examine differences between other methodological choices, e.g., tuple mining or regularization methods. Since Cars196 is a common DML benchmark, most researchers already train models on this dataset. Our image property analysis can thus be conducted without additional training. Adapting this method to other datasets, such as the birds dataset CUB200 [275], would be interesting but is not trivial, since rendering convincing images of birds is difficult. Given car images, there are many different 3D car models available, which are mostly static and do not have moving parts. Birds, on the other hand, show larger flexibility in positions and posture. The number of properties that need to be considered is thus much higher.

8. Analyzing a Scientific Venue Recommender using Integrated Gradients

The following section mainly follows our work in [131]. When choosing a scientific conference or journal (in the following also called *venue*) to submit a manuscript, researchers consider several factors. While factors such as the venue’s impact, time, or location are important, the main factor is the manuscript’s thematic fit to the conference. This can be ensured by inspecting the Call for Papers or by analyzing previously published papers at the given conference. Given the growing number of conferences (e.g., the exponential growth of computer science publications indicates more and/or larger conferences¹), the second approach has become harder than ever, especially for novice researchers, or even senior researchers wanting to publish in a new domain. Finding a thematically fitting venue for a manuscript therefore is a time consuming task.

We aim to simplify this process by introducing Where to Submit (WTS), a Natural Language Processing (NLP) system based on a Convolutional Neural Network (CNN) that recommends academic venues given the title, abstract, and/or keywords of a planned publication. The system is trained on previously published manuscripts. To understand the system’s choice of recommending a specific venue, WTS analyzes the words and phrases that had the highest impact on a recommendation using the **Gradient-Based Attribution** principle. More specifically, we use the Integrated Gradient method [260] to calculate the importance of each word in the input. A researcher can then use the list of recommended venues and their importance highlights as a starting point to find the best fitting venue based on other factors such as the Call for Papers, rank, or deadline.

Setting Given a title, abstract, and keywords of a publication, we aim to predict the venue where the paper was published. We interpret the classification task as a ranking task by ordering the potential venues according to their score in the model output and use metrics that assess the ranking performance.

To solve this task, we use a TextCNN model that is depicted in Figure 8.1. We lowercase and embed each word in the title, abstract, and keywords using Word2Vec [179], trained on the abstracts and titles of the respective dataset. This creates three two-dimensional inputs for the model. Each input is then processed through a convolution layer with potentially multiple filter sizes and max-over-time pooling, which maps the processed

¹As visualized on <https://dblp.uni-trier.de/statistics/recordsindb1p> (last accessed: 2023-02-10).

8. Analyzing a Scientific Venue Recommender using Integrated Gradients

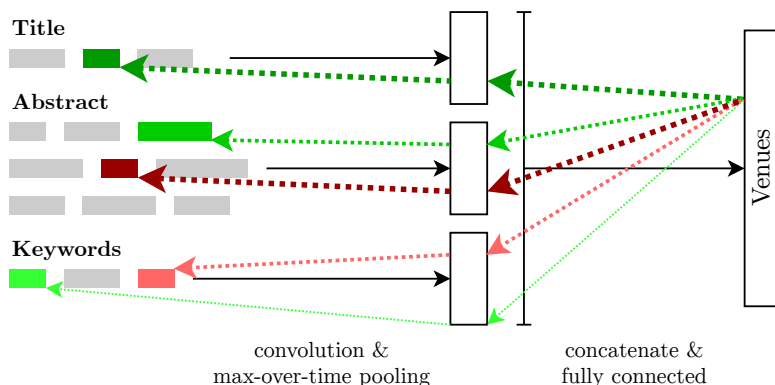


Figure 8.1.: Overview of WTS. Title, abstract, and keywords are processed separately by a convolution layer and max-over-time pooling. The output vectors are concatenated and fed through two fully connected layers that predict fitting venues. The green and red, dotted arrows indicate how the gradient is used to visualize the most important words and phrases using the Integrated Gradient method [260]. Green arrows indicate positive impact of the word on the output, red arrows indicates a negative influence. Thicker arrows with stronger colors mean larger gradients.

inputs to a fixed size. The resulting vectors are concatenated and fed through two feed-forward layers that map to a vector representing the venues. Training with Categorical Cross Entropy (CCE) leads to higher outputs for more likely venues. Dropout [255] and batch normalization [108] are used for regularization.

In order to assess the performance of our model, we compare it to several baselines: The *random baseline* always predicts venues in a uniformly random order. We report the expected value for each metric. The *majority baseline* orders venues by the number of publications in the training set in descending order. For the *Logistic Regression baseline*, we tokenize the title, abstract, and keywords. From all tokens, we create a term frequency vector and train a multi-class Logistic Regression. The venues are then sorted in decreasing order based on the model output. We also compare our method to one of the approaches outlined by Iana et al. in [105]. For better comparison, we use their best performing approach (according to Recall@10) that does not incorporate any third-party information (called “Ensemble TF-IDF & word2vec plus CNN (10)”). Here, a logistic regression combines the outputs of two classifiers: (1) Concatenating all corresponding abstracts of a venue, creating one TF-IDF representation and ranking venues using their distance to the provided abstract representation and (2) classifying abstracts using TextCNN [126], which has been described in Section 2.2.3. In contrast to our model, their approach does not provide the network with title and keyword information.

We train and evaluate all models on the subsets of the Semantic Scholar dataset that we have introduced in Section 3.6.2 as well as perform a hyperparameter random search [24] on the validation data. We then report the three metrics Accuracy, Accuracy@5, and Mean Reciprocal Rank (MRR) that we have described in Section 2.5 in Table 8.1.

On both datasets, Artificial Intelligence (AI) and medicine, WTS significantly outper-

Table 8.1.: Results of the baselines and our methods. Best values are displayed in bold.

	Method	Accuracy \uparrow	Acc@5 \uparrow	MRR \uparrow
AI	Random	0.013	0.064	0.063
	Majority	0.086	0.319	0.212
	Log. Reg.	0.487	0.811	0.628
	Iana et al.	0.065	0.198	0.154
	WTS (Ours)	0.503	0.831	0.645
Medicine	Random	0.013	0.064	0.063
	Majority	0.069	0.213	0.157
	Log. Reg.	0.587	0.911	0.724
	Iana et al.	0.440	0.808	0.599
	WTS (Ours)	0.659	0.948	0.782

forms all baseline methods in all metrics according to a Wilcoxon signed-rank test [293, 294] at 1% confidence level. In approximately 83% (AI) and 95% (Medicine) of all cases, the correct venue is in the top five.

Interestingly, compared to the Medicine dataset, the method by Iana et al. [105] performs poorly on AI publications. We suspect this is due to the smaller size of the AI dataset and a higher skew in publication counts per venue (cf. Table 3.1).

8.1. Methodology

We now turn to the analysis of WTS’s predictions using a concrete implementation of the **Gradient-Based Attribution** principle, more specifically the Integrated Gradients algorithm [260]. This method allows us to not only recommend venues to the user but also explain which words had the highest influence on a recommendation.

Integrated Gradients [260] works by defining a so-called baseline input. It aims at providing counterfactual intuition: If a feature is important for the output, the absence of that feature should change the output. Comparing the gradients of the model w.r.t. the input with and without a feature allows us to quantify the importance of that feature. Thus, a completely featureless input is a good baseline [9]. For images, this can be a black image, for texts, all embedding vectors could be zero. Integrated Gradients then integrates the gradients along the path from the baseline to the input. Since the integration is computationally not feasible, the result is approximated by interpolating the input and baseline in discrete steps and computing the gradients w.r.t. the model input for each of the interpolated inputs. These gradients are then summed up. Given the input matrix \mathbf{X} containing all token embedding vectors for an input text, the baseline matrix \mathbf{X}_{base} , and the Deep Learning (DL) model as function f that maps the input embedding matrix to a class distribution, the approximated Integrated Gradients for the class index c are defined as

8. Analyzing a Scientific Venue Recommender using Integrated Gradients

$$\text{IntegratedGradients}(\mathbf{X}, c) = (\mathbf{X} - \mathbf{X}_{\text{base}}) \cdot \sum_{k=1}^m \frac{\partial f(\mathbf{X}_{\text{base}} + \frac{k}{m} \cdot (\mathbf{X} - \mathbf{X}_{\text{base}}))_c}{\partial \mathbf{X}} \cdot \frac{1}{m}, \quad (8.1)$$

where m is the number of interpolation steps between the input and baseline and $f(\cdot)_c$ is the output of the model for class c . The output of this equation is a matrix of the same size as the input embedding matrix \mathbf{X} . The method satisfies two axioms that are desirable for attribution methods: First, the sensitivity axiom states that if the input and baseline differ in a feature and the resulting prediction of the model, the attribution of that feature should be non-zero. Complementarily, if the output of a model does not depend on an input feature, the attribution of that feature should be zero (“insensitivity”). Second, the implementation invariance axiom states that if there are two functionally equivalent models (the same inputs result in the same prediction for all inputs), the attribution of a feature should be the same for both models. More information about these axioms can be found in the original paper [260].

We use an implementation of Integrated Gradients from the PyTorch Captum library² to find the most influential words and phrases for the top five recommendations of the network. The number of interpolation steps between the embedding vectors and zero embeddings is set to $m = 50$, which is the Captum default. The resulting attribution scores are per embedding and per embedding vector dimension, so we need to average the attribution scores over the embedding dimensions to obtain a single attribution score per token. Positive attribution scores contribute positively to the prediction, negative scores negatively. While each token gets assigned a score, oftentimes, the absolute scores are comparably small except for a few tokens, because only those are enough to produce the output. The resulting attribution scores can then be interpreted using domain knowledge about the venues.

8.2. Experiments

We perform qualitative experiments with the Integrated Gradients method on our proposed model. For this, we visualize the positive and negative highlights given by the explanation method for specific papers. Since Integrated Gradients is a local method, we need to select a specific paper to explain. Given the highlights, we can interpret them in terms of usefulness for the recommendation. We expect to receive highlighted words and phrases that are relevant for the recommended venue. We test this by looking at three papers from different areas of AI: the well-known BERT paper (published and awarded with the best paper award at NAACL, an NLP conference) [48] as well as the ResNet paper (published at CVPR, a Computer Vision (CV) conference) [86] and the BatchNorm paper (published at ICML, a core Machine Learning (ML) conference) [108]. We would expect to receive different highlights for each paper, since the papers are from different areas of AI. The BERT paper highlights should focus on language aspects, while the

²<https://captum.ai> (last accessed: 2023-02-10)

ResNet and the BatchNorm paper highlights should focus on the image and the learning aspects, respectively. Now that we formulated our expectations, we can compare them to the actual highlights. In general, evaluating formulated expectations could be done by surveying domain experts about how well they match the highlights for a sufficiently large set of papers and making statistical statements about the usefulness of the highlights.

If, according to the surveyed experts, the expectations match the actual highlights, we can conclude that the WTS model seems to extract the correct information from the papers. It also allows us to find dissonances between the expectations and the actual highlights, which can be used as a starting point for further investigations into the model behavior. Given a non-obvious recommendation for a paper, we can also use the highlights to find out why the model recommends a specific venue. Since obtaining feedback from a group of domain experts for a sufficiently large set of papers is time-consuming and expensive, it is out of scope for this thesis. We thus restrict ourselves to the three mentioned papers and our opinion on the fit of highlights and expectations.

8.3. Results

Figure 8.2 visualizes the highlights given by Integrated Gradients for the three provided papers. WTS correctly recommends the corresponding venues to all publications at the highest rank. The follow-up recommendations by WTS are also thematically similar to the top recommendation. For the NAACL paper, INTERSPEECH and EMNLP are also recommended, which are both conferences for research in language and text processing. ECCV and ICCV are recommended for the CVPR paper, which are also CV conferences. Due to the relatively broad and theoretical impact of BatchNorm that reflects in the ICML recommendation, the recommendations also include the AAAI. Here, however, the RSS, which is a conference for robotics, is also recommended. We will explore why this conference is recommended later.

The Integrated Gradients method highlights the words that are important for the top recommendation in the title and abstract. Green text contributes positively to the prediction, red text negatively. We see that the words “Transformers” and “Language Understanding” are plausibly identified by the gradient attribution method as words that qualify this publication as an NLP paper.

For the ResNet paper, the words “Recognition” and “visual” are most important for the CVPR recommendation, while “Learning” seems to have a negative effect on this venue recommendation. This negative impact might be surprising at first, but only approx. 26 % of CVPR training dataset papers contain the word “learning”, while core ML conferences or some robotic conferences show much higher shares (e.g., ICML: approx. 56 %, ECML: approx. 59 %, or “Conference on Robot Learning” (CoRL): approx. 90 %). The presence of the word “learning” is thus usually less of an indicator for the CVPR conference.

For the BatchNorm paper, “Training” and “method” are highlighted, indicating a method paper, leading to core ML conference recommendations. To better understand why WTS recommends the “Robotics: Science and Systems” (RSS) conference for the BatchNorm paper [108], we visualize the attribution scores regarding this conference in Figure 8.3.

8. Analyzing a Scientific Venue Recommender using Integrated Gradients

<p>BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding We introduce a new language representation model called BERT, which stands for Bidirectional Encoder Representations from Transformers. Unlike recent language representation models, BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications. BERT is conceptually simple and empirically powerful. It obtains new state-of-the-art results on eleven natural language processing tasks, including pushing the GLUE score to 80.5% (7.7% point absolute improvement), MultiNLI accuracy to (86.7% (4.6% absolute improvement), SQuAD v1.1 question answering Test F1 to 93.2 (1.5 point absolute improvement) and SQuAD v2.0 Test F1 to 83.1 (5.1 point absolute improvement).</p> <p>Venue recommendations: <i>NAAACL</i>, <i>INTERSPEECH</i>, <i>EMNLP</i></p>
<p>Deep Residual Learning for Image Recognition Deeper neural networks are more difficult to train. We present a residual learning framework to ease the training of networks that are substantially deeper than those used previously. We explicitly reformulate the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. We provide comprehensive empirical evidence showing that these residual networks are easier to optimize, and can gain accuracy from considerably increased depth. On the ImageNet dataset we evaluate residual nets with a depth of up to 152 layers—8x deeper than VGG nets but still having lower complexity. An ensemble of these residual nets achieves 3.57% error on the ImageNet test set. This result won the 1st place on the ILSVRC 2015 classification task. We also present analysis on CIFAR-10 with 100 and 1000 layers. The depth of representations is of central importance for many visual recognition tasks. Solely due to our extremely deep representations, we obtain a 28% relative improvement on the COCO object detection dataset. Deep residual nets are foundations of our submissions to ILSVRC & COCO 2015 competitions, where we also won the 1st places on the tasks of ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation.</p> <p>Venue recommendations: <i>CVPR</i>, <i>ECCV</i>, <i>ICCV</i></p>
<p>Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift Training Deep Neural Networks is complicated by the fact that the distribution of each layer’s inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. We refer to this phenomenon as internal covariate shift, and address the problem by normalizing layer inputs. Our method draws its strength from making normalization a part of the model architecture and performing the normalization for each training mini-batch. Batch normalization allows us to use much higher learning rates and be less careful about initialization, and in some cases eliminates the need for Dropout. Applied to a state-of-the-art image classification model, Batch Normalization achieves the same accuracy with 14 times fewer training steps, and beats the original model by a significant margin. Using an ensemble of batch-normalized networks, we improve upon the best published result on ImageNet classification: reaching 4.82% top-5 test error, exceeding the accuracy of human raters.</p> <p>Venue recommendations: <i>ICML</i>, <i>AAAI</i>, <i>RSS</i></p>

Figure 8.2.: Titles and abstracts for three publications [48, 86, 108] along their top venue recommendations by WTS. It also highlights the words leading to the highest ranked recommendation: green text contributes positively to the prediction, red text negatively. Black text does not contribute. The more saturated the color, the more important the word.

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Training Deep Neural Networks is complicated by the fact that the distribution of each layer’s inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. We refer to this phenomenon as internal covariate shift, and address the problem by normalizing layer inputs. Our method draws its strength from making normalization a part of the model architecture and performing the normalization for each training mini-batch. Batch Normalization allows us to use much higher learning rates and be less careful about initialization, and in some cases eliminates the need for Dropout. Applied to a state-of-the-art image classification model, Batch Normalization achieves the same accuracy with 14 times fewer training steps, and beats the original model by a significant margin. Using an ensemble of batch-normalized networks, we improve upon the best published result on ImageNet classification: reaching 4.82% top-5 test error, exceeding the accuracy of human raters.

Figure 8.3.: Integrated Gradients attributions for the BatchNorm paper [108] w.r.t. the RSS venue recommendation, which was ranked third by WTS.

Here, the method highlights “normalization” and “image”, but also “accelerating” shows some positive influence for this recommendation. Due to the use of motors in robots, “accelerating” seems to be a reasonable keyword for this conference. Also, many robotic papers work on or with machine vision techniques, which is why “image” is also reasonable. The word “normalization” seems to be relatively arbitrary, but the method assigns a high influence on the model’s prediction. Such findings can motivate further investigations into the model’s outputs and the underlying data. Does the model always correlate “normalization” with robotics venues? Is there a thematically sound explanation for this influence? The use of gradient-based attribution methods gives a first indication of where to look. Without it, there would be no direct way to find out what word influences the model to predict the RSS as a venue for the BatchNorm paper. Integrated Gradients thus is a tool to debug the model and to reason about its predictions using domain knowledge.

8.4. Conclusion

We have analyzed our DL based model WTS that recommends scientific venues based on the title, abstract, and/or keywords of a publication. WTS is designed to provide an explanation as to why a certain venue was recommended by using the Integrated Gradients method, which implements the **Gradient-Based Attribution** principle. This integration makes WTS the first interpretable and open recommendation service for both, conferences and journals. We have shown that WTS provides strong recommendations on publications from the areas of AI and medicine.

In order to make our system available to the public, we release WTS as a web service³ where researchers can input their AI paper’s information and receive recommendations for

³<https://wts.professor-x.de> (last accessed: 2023-02-10)

8. Analyzing a Scientific Venue Recommender using Integrated Gradients

The screenshot displays the WTS website interface. At the top, the logo 'WTS?' is prominently shown in red, with the tagline 'WHERE TO SUBMIT YOUR PAPER' below it. The main content area is divided into several sections:

- Introduction:** A paragraph explaining that WTS recommends Artificial Intelligence conferences and journals based on user input (title, abstract, keywords). It mentions that users get a list of the five most fitting out of 77 conferences and journals, with words and phrases highlighted in color to indicate their impact.
- How does it work?:** A section describing the method, referencing a paper from EMNLP 2020 titled 'Where to Submit? Helping Researchers to Choose the Right Venue'. It notes that the code and data for local runs are available on GitHub.
- Disclaimer:** A note stating that the website is not responsible if a paper is rejected at the proposed conference.
- Input Fields:** Fields for 'Title' (containing 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift') and 'Abstract' (containing a detailed paragraph about training Deep Neural Networks and the use of Batch Normalization). A 'Keywords' field is also present.
- Action Button:** A blue button labeled 'Where to Submit?'.
- Fitting Conferences and Journals:** A section listing the top five fitting venues. Two are visible:
 - 1. ICML:** International Conference on Machine Learning. Description: 'The International Conference on Machine Learning (ICML) is the premier gathering of professionals dedicated to the advancement of the branch of artificial intelligence known as machine learning. ICML is globally renowned for presenting and publishing cutting-edge research on all aspects of machine learning used in closely related areas like artificial intelligence, statistics and data science, as well as important application areas such as machine vision, computational biology, speech recognition, and robotics. (source)'. Search links for Google, conferenceranks.com, and WikiCFP.com are provided.
 - 2. AAAI:** AAAI Conference on Artificial Intelligence. Description: 'The purpose of the AAAI conference is to promote research in artificial intelligence (AI) and scientific exchange among AI researchers, practitioners, scientists, and engineers in affiliated disciplines. (source)'.

Figure 8.4.: A screenshot of the WTS website, which is openly available at <https://wts.professor-x.de> (last accessed: 2023-02-10).

venues. The web service applies the trained CNN and explainability method and shows the top five predicted venues for the given paper along with a color-coded explanation and venue-related information (see Figure 8.4). Given the Accuracy@5 results described in Table 8.1 as well as the useful backup recommendations shown in Figure 8.2, a fitting venue is usually displayed to the user. The explanations then help users to understand the recommendation of the model and to build trust in the model's performance.

9. Comparing Deep Metric Learning (DML) Models with a new Attribution Map Generation Method

For this implementation of the **Gradient-Based Attribution** principle, we revisit the DML model comparison setting from Chapter 7. We thus skip the motivation, setting, and notation and refer to Chapter 7 and our work in [134] for details, from which this section is mostly taken. While our implementation of the Generated Input Data principle focused on high-level features, we now want to analyze the low-level features that are used by the network to embed an image. Our general goal is still to find differences in the feature importances between different DML loss functions. For this, we develop a novel method to generate attribution maps for DML models that can be compared and interpreted using domain knowledge. Such techniques are usually focused on classification tasks. In the case of DML, we try to generate maps for a representation task. We then qualitatively and quantitatively compare such maps between all models.

We first introduce our method for generating attribution maps for DML models and then compare the 14 loss functions from Chapter 7 in terms of their low-level, i.e., pixel-level, attribution across datasets. A qualitative analysis of the maps on a few samples is also provided.

9.1. Methodology

Our proposed method aims to identify features on pixel-level that are important for the network’s decision to output a certain embedding. For this, we adapt a gradient based attribution map generation method to the DML setting [248]. While attribution maps are usually used to qualitatively analyze one network on a single image, however, we propose quantitative measures to compare models.

Our attribution maps seek to answer the question “What are the main image regions that guided the network to output the specific embedding?”. Intuitively, we obtain the final embedding \mathbf{x}_i for image $iminst_i$ by altering the pixels of an image that shows no features (a baseline image as described in Section 8.1), i.e., a completely black image \mathbf{I}_{base} , towards \mathbf{I}_i . The larger the change towards the final embedding, the more important a pixel. A visual depiction of this intuition is shown in Figure 9.1. Thus, we want to identify the most influential pixels regarding the distance $d_L(\mathbf{x}_i, \mathbf{x}_{\text{base}})$ between the image’s embedding \mathbf{x}_i and the embedding of the black image $\mathbf{x}_{\text{base}} = f_L(\mathbf{I}_{\text{base}})$. We do

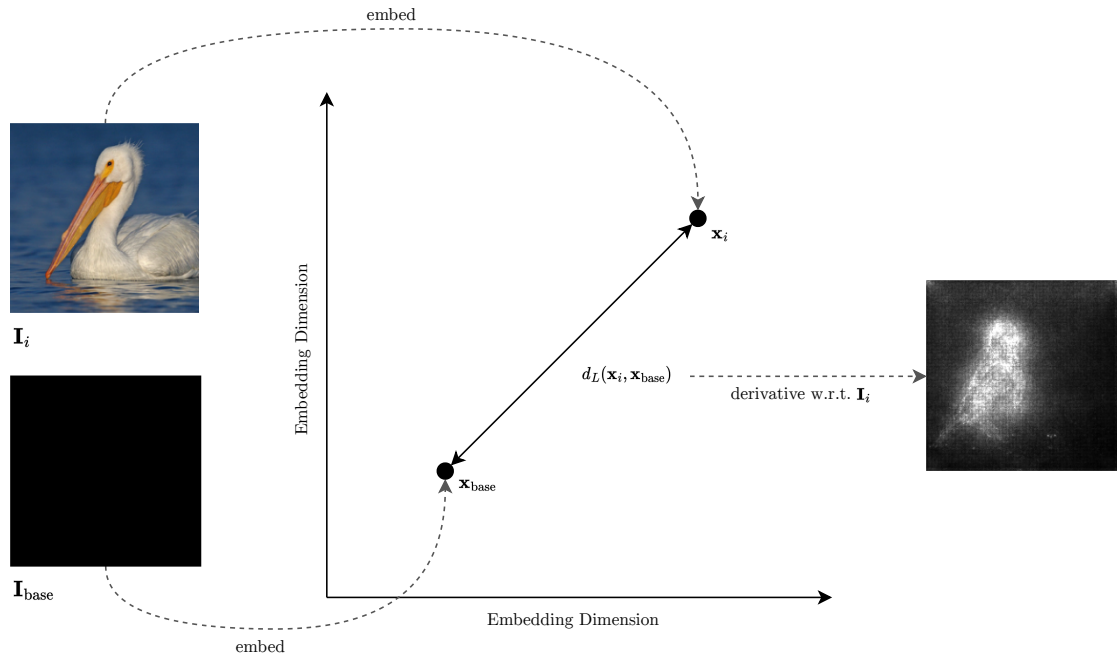


Figure 9.1.: Intuition for our gradient based attribution method. The distance between the embeddings of the image and a completely black image (here shown in a two-dimensional space) can be differentiated w.r.t. the image's pixels. The larger the gradient for a pixel, the more important the pixel is for the network to change the embedding from the black image's embedding (with no features) to the image's embedding.

this by computing the gradients of the loss-specific distance w.r.t. the input \mathbf{I}_i :

$$\mathbf{s}_L(\mathbf{I}_i) = \partial d_L(\mathbf{x}_i, \mathbf{x}_{\text{base}}) / \partial \mathbf{I}_i. \quad (9.1)$$

Since these gradients can be noisy, we apply the SmoothGrad method [250] by creating l image variants by adding gaussian noise $\mathcal{N}(0, \sigma^2)$ to the input image and averaging the resulting gradients:

$$\hat{\mathbf{s}}_L(\mathbf{I}_i) = \frac{1}{l} \sum_1^l \mathbf{s}_L(\mathbf{I}_i + \mathcal{N}(0, \sigma^2)). \quad (9.2)$$

High absolute gradients indicate that changing the corresponding input value has large influence on the measured distance, thus identifying pixels responsible for the deviation of the base embedding. We post-process the gradients using common techniques, namely (in this order) taking the absolute value, averaging across the color channel dimension, clipping values higher than the 99th percentile, and scaling the values to a range from zero to one. These steps make the raw gradients more semantic, yielding an interpretable attribution map $\tilde{\mathbf{s}}_L(\mathbf{I}_i)$ [250].

9.2. Experiments

Overall, our proposed method is a qualitative technique to highlight important image areas for the network. While this can be used to visualize differences between DML loss functions on single images, we propose to quantify differences using this technique: Given two models f_{L_1} and f_{L_2} trained with different losses L_1 and L_2 , we apply both models on the same test images $\mathbf{I}_1, \dots, \mathbf{I}_n$ and compute the attribution maps $\tilde{\mathbf{s}}_{L_1}(\mathbf{I}_1), \dots, \tilde{\mathbf{s}}_{L_1}(\mathbf{I}_n)$ for L_1 and L_2 . Inspired by the literature for the visual saliency task [143, 222], i.e., estimating a heatmap of a human’s eye fixations on an image, we compare attribution maps by calculating the average Pearson product-moment correlation coefficient and Jensen-Shannon Divergence (JSD) [56] between the same image’s attribution maps of two networks. We transform correlations to Fisher-Z space before averaging [247] and divide each attribution map by its sum to obtain probability distributions for JSD. Mean correlations close to one show that both attribution maps usually have a linear dependency, meaning that both networks attend to the same image regions. Lower values indicate that both models learned different features in order to represent images. JSD, on the other hand, measures a divergence between attribution maps and thus grows with larger differences between the attribution maps, which is the opposite to correlation. A mean JSD of zero means that both methods produce the same attribution maps, while higher values (bounded by 1, due to base 2 logarithm) show larger differences.

9. Comparing DML Models with a new Attribution Map Generation Method

	Ranking									Classification					None
	Contrastive	Triplet	NTXent	Margin	Margin/class	FastAP	SNR Con.	MS	MS+Miner	ProxyNCA	N. Softmax	CosFace	ArcFace	SoftTriple	
Ranking	Contrastive	84±6	86±5	86±5	85±5	85±5	86±5	86±5	87±5	85±5	86±5	85±5	85±5	86±5	74±11
	Triplet	84±6	85±6	85±6	84±6	85±6	84±6	84±6	84±6	83±6	84±6	82±7	82±7	84±6	75±10
	NTXent	86±5	85±6	85±6	85±6	85±6	85±5	85±5	86±5	85±5	86±5	84±6	84±6	85±5	74±10
	Margin	86±5	84±6	85±6	85±6	85±6	85±6	85±5	85±6	84±6	85±6	84±6	83±6	84±6	74±10
	Margin/class	85±5	85±6	85±5	85±6	85±6	84±6	84±6	85±5	84±5	85±5	84±5	83±6	84±6	75±10
	FastAP	85±5	84±6	85±6	85±6	84±6	85±6	85±6	85±5	85±5	85±5	85±5	84±6	84±6	73±11
	SNR Con.	86±5	84±6	85±5	85±6	84±6	85±6	85±5	86±5	84±5	86±5	86±5	84±6	84±5	74±10
	MS	86±5	84±6	86±5	85±5	85±5	85±5	86±5	86±5	85±5	86±5	85±5	85±6	85±5	74±10
	MS+Miner	87±5	84±6	86±5	85±6	85±5	85±5	85±5	86±5	85±5	85±5	86±5	85±6	85±5	73±11
	ProxyNCA	85±5	83±6	85±5	84±6	84±5	85±5	84±5	85±5	85±5	87±4	87±4	85±5	85±5	86±5
Classif.	N. Softmax	86±5	84±6	86±5	85±6	85±5	85±5	86±5	86±5	87±4	87±4	86±5	86±5	87±4	74±10
	CosFace	85±5	82±7	84±6	84±6	84±6	84±6	84±6	85±6	85±5	86±5	85±5	85±5	87±4	73±11
	ArcFace	85±5	82±7	84±6	83±6	83±6	84±6	84±5	85±5	85±5	86±5	85±5	85±5	86±4	73±11
	SoftTriple	86±5	84±6	85±5	84±6	84±6	85±5	86±5	86±5	86±5	87±4	87±4	86±4	86±4	74±11
	None	74±11	75±10	74±10	74±10	75±10	73±11	74±10	74±10	73±11	74±10	74±11	73±11	73±11	74±11

Table 9.1.: Average Pearson correlations between all loss functions on the Cars196 dataset. All values are in 10^{-2} .

	Ranking									Classification					None
	Contrastive	Triplet	NTXent	Margin	Margin/class	FastAP	SNR Con.	MS	MS+Miner	ProxyNCA	N. Softmax	CosFace	ArcFace	SoftTriple	
Ranking	Contrastive	88±7	89±5	89±6	89±5	89±5	89±5	89±5	89±5	88±6	89±6	89±6	89±5	89±5	86±6
	Triplet	88±7	89±5	89±6	89±5	89±5	89±5	89±5	89±5	88±6	89±5	88±7	88±6	88±6	86±6
	NTXent	89±6	89±5	89±5	89±5	90±5	90±5	90±5	90±4	89±5	90±5	88±6	89±5	89±5	86±6
	Margin	89±6	89±6	89±5	90±5	90±5	90±5	90±5	90±5	89±5	89±5	88±6	89±5	89±6	86±6
	Margin/class	89±5	89±6	90±5	90±5	90±5	90±5	90±5	90±5	89±6	89±5	88±6	89±5	89±5	86±6
	FastAP	89±6	89±6	90±5	90±5	90±5	90±5	90±5	90±5	89±5	89±5	89±6	90±5	90±5	86±6
	SNR Con.	89±5	89±5	89±5	90±5	90±5	90±5	90±4	90±5	89±5	90±5	89±5	89±5	89±5	87±6
	MS	89±5	89±5	90±5	90±5	90±5	90±5	90±4	90±4	89±5	90±4	89±5	89±5	89±5	87±5
	MS+Miner	89±5	89±5	90±4	90±5	90±5	90±5	90±4	90±4	89±5	90±5	89±5	89±5	89±5	86±6
	ProxyNCA	88±6	88±6	89±5	89±5	89±6	89±5	89±5	89±5	89±5	89±5	88±6	88±6	89±5	87±5
Classif.	N. Softmax	89±6	89±5	90±5	89±5	89±5	90±5	90±4	90±5	89±5	89±5	89±6	90±5	90±5	86±6
	CosFace	89±6	88±6	89±5	89±6	88±6	89±6	89±5	89±5	88±6	89±6	89±6	90±5	90±5	86±6
	ArcFace	89±6	88±6	89±5	89±6	89±5	89±5	89±5	89±5	89±5	90±5	90±5	90±5	90±5	86±6
	SoftTriple	89±5	88±6	89±5	89±6	89±5	90±5	89±5	89±5	89±5	90±5	90±5	90±5	90±5	86±6
	None	86±6	86±6	86±6	86±6	86±6	86±6	87±6	87±5	86±6	87±5	86±6	86±6	86±6	86±6

Table 9.2.: Average Pearson correlations between all loss functions on the CUB200 dataset. All values are in 10^{-2} .

	Ranking									Classification					None
	Contrastive	Triplet	NTXent	Margin	Margin/class	FastAP	SNR Con.	MS	MS+Miner	ProxyNCA	N. Softmax	CosFace	ArcFace	SoftTriple	
Ranking	Contrastive	62±12	63±12	63±12	66±11	66±12	63±13	61±13	63±12	46±20	49±17	45±20	45±20	46±20	51±21
	Triplet	62±12	63±12	63±12	66±11	66±12	63±13	61±13	63±12	50±19	52±18	50±20	49±20	50±20	59±20
	NTXent	59±12	63±12	63±12	66±11	66±12	63±13	61±13	63±12	52±18	54±16	52±18	52±18	53±17	56±19
	Margin	61±12	66±11	63±12	66±11	66±12	63±13	61±13	63±12	50±19	51±17	50±19	49±19	50±19	58±19
	Margin/class	61±12	66±11	63±12	65±12	65±12	63±13	61±13	63±12	50±19	52±17	50±20	49±20	50±20	58±20
	FastAP	59±12	63±13	63±13	61±13	61±13	61±13	61±13	60±13	49±20	52±17	50±20	49±20	50±19	55±21
	SNR Con.	60±12	61±13	57±14	59±13	59±13	61±13	61±13	59±14	44±22	47±19	43±22	43±22	44±21	48±23
	MS	59±13	63±12	62±12	62±12	62±12	61±13	61±13	59±14	53±18	54±16	53±18	53±18	53±18	58±18
	MS+Miner	60±13	64±12	62±12	62±12	63±13	60±13	60±13	57±14	54±17	55±16	54±18	53±18	54±17	58±18
	ProxyNCA	46±20	50±19	52±18	50±19	50±19	49±20	44±22	53±18	54±17	62±14	62±14	67±12	66±12	64±13
Classif.	N. Softmax	49±17	52±18	54±16	51±17	52±17	52±17	47±19	54±16	62±14	62±14	64±13	63±14	63±13	59±16
	CosFace	45±20	50±20	52±18	50±19	50±20	50±20	43±22	53±18	67±12	64±13	69±11	69±11	67±11	63±14
	ArcFace	45±20	49±20	52±18	49±19	49±20	49±20	43±22	53±18	66±12	63±14	69±11	69±11	65±12	61±15
	SoftTriple	46±20	50±20	53±17	50±19	50±20	50±19	44±21	53±18	64±13	63±13	67±11	65±12	65±12	62±15
	None	51±21	59±20	56±19	58±19	58±20	55±21	48±23	58±18	60±16	59±16	63±14	61±15	62±15	62±15

Table 9.3.: Average Pearson correlations between all loss functions on the Stanford Online Products (SOP) dataset. All values are given in percent. Larger values have darker cells.

9.3. Results

Quantitative Comparison

Tables 9.1 to 9.3 show the correlations’ means and standard deviations for the test datasets of Cars196, CUB200, and Stanford Online Products (SOP), respectively. We omit the Jensen-Shannon Divergence and their standard deviations for legibility, as all values are around 0.02 ± 0.01 and show similar tendencies as the correlation. The tables for the JSD metric can be found in Appendix A.

Compared to the other datasets, correlations for SOP are generally weaker with larger standard deviations, showing that losses are not consistent across images in terms of feature extraction. Surprisingly, loss pairs of different loss types (ranking vs classification) show lower correlations than pairs of the same loss type, suggesting that different loss types lead to different attribution maps. Grouping loss pairs by their distance/similarity metric does not show such clear differences. The “None” model has stronger correlation with classification than with ranking losses, which is expected due to its training on a classification task.

For the Cars196 and CUB200 datasets, we find overall stronger correlations, which shows that models tend to focus on the same pixels to embed an image. Also, the standard deviation is around 0.06 for both datasets, indicating consistent behavior across all images. We cannot identify the same large correlation drops when comparing ranking and classification losses. A noticeable drop in correlation can only be observed with the untrained “None” model, which is expected, but surprisingly not that steep. This shows a high similarity in extracted features between them. We conclude that ImageNet based initialization of the untrained models already leads to features that are picked up by the analyzed DML networks.

Qualitative Comparison

Given the findings of the quantitative results, we also visually inspect the learned features. Figures 9.2 to 9.4 show attribution maps for all investigated networks for a sample image of each of the three tested datasets. In the following, we focus on Figure 9.4, which shows a chair from the SOP dataset, as this dataset shows clear tendencies in the quantitative results. We observe that most methods highlight parts of the chair, but focus on different areas. While e.g., Contrastive Loss attends to the chair’s legs and back, CosFace pays more attention to the seat. Given the quantitative difference between ranking and classification losses, we observe that ranking based methods usually show more pronounced local highlights, while classification based methods highlight broader areas. It also seems that the background area is more important for classification approaches. This matches our observations from Chapter 7, where we have found that classification based losses tend to focus more on the background than ranking based losses.

9. Comparing DML Models with a new Attribution Map Generation Method

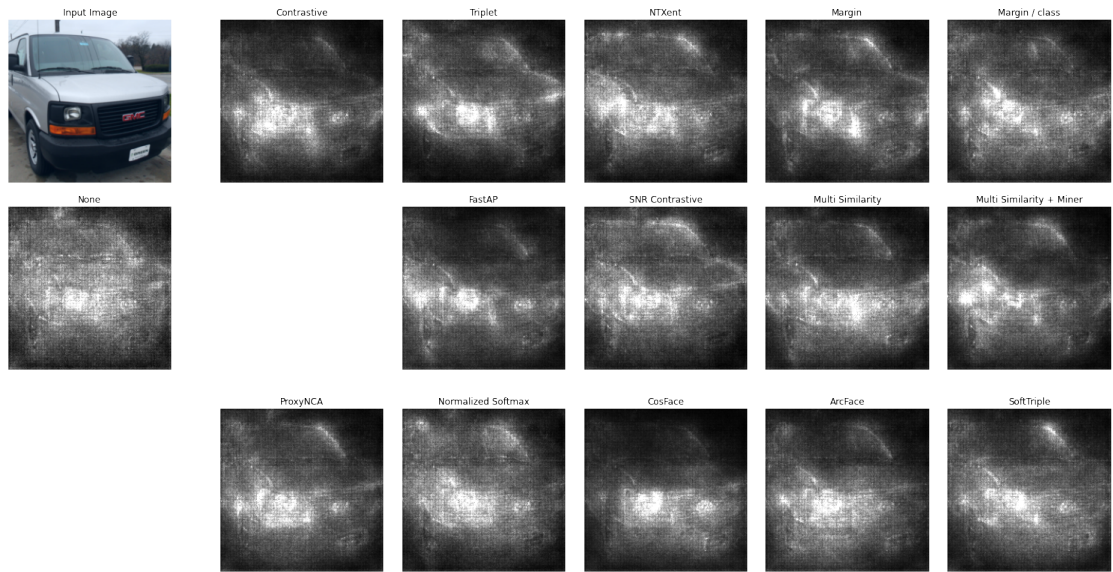


Figure 9.2.: Attribution maps of a sample image from Cars196. The original image and the “None” baseline (pretrained ImageNet weights) are in the first column. The first two rows show *embedding* losses, the third row shows *classification* losses.

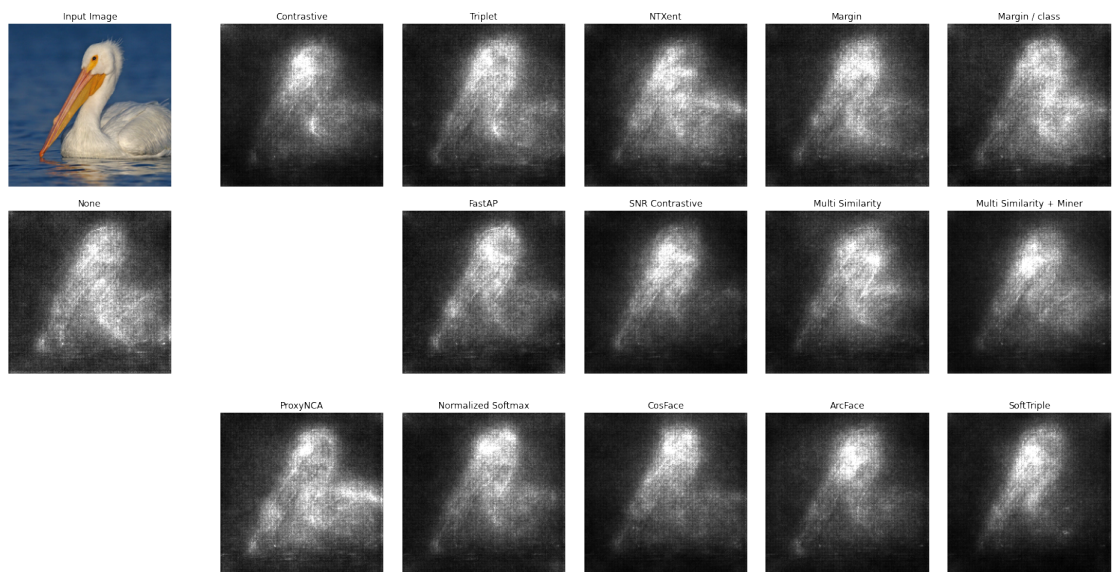


Figure 9.3.: Attribution maps of a sample image from CUB200. The original image and the “None” baseline (pretrained ImageNet weights) are in the first column. The first two rows show *embedding* losses, the third row shows *classification* losses.

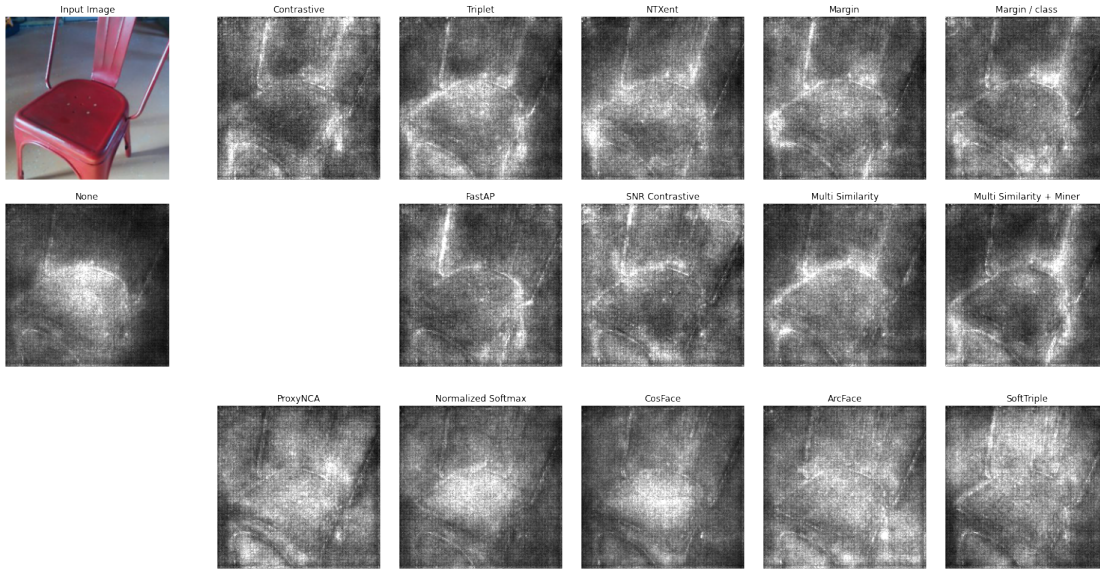


Figure 9.4.: Attribution maps of a sample image from SOP. The original image and the “None” baseline (pretrained ImageNet weights) are in the first column. The first two rows show *embedding* losses, the third row shows *classification* losses.

9.4. Conclusion

We have introduced a new attribution method for DML models. With this, we have analyzed and compared different DML loss functions. Our analysis has shown that on the SOP dataset, classification based losses attend to fairly different regions than ranking based losses. This matches our other findings that classification based losses tend to focus more on the background than ranking based losses. However, in this case, we have used another principle and a novel implementation of this principle to make these observations. Overall, these two independent investigations show similar results, which also confirms the validity of our approach.

Part III.

Improving Deep Learning (DL) Models

10. Principles for Improving Deep Learning (DL) Models

Now that we can obtain insights into how the Neural Network (NN) makes its decisions and what low-level and high-level input features are important to form an output, we turn to the improvement of DL models using domain knowledge by exploring four principles and multiple of their implementations. Similar to the structure of Part II, we split this part into principles that are mostly applied to the input and output. We then describe specific implementations of these principles and evaluate their effectiveness in different task settings. In addition, we also present implementations that use multiple principles from both, the input and output.

First, we explore two principles that are based on the input side of DL models, namely affecting the input data using the **Input Masking and Augmentation** principle and altering the preprocessing step with the **Feature Extraction using Pretrained Multimodal Models** principle. Afterwards, two principles relating to the output components of the training process of the DL model are discussed. Here, we explore the use of **Weak Label Generation** to enlarge the training dataset by altering the desired output of the model. We also enrich the used **Loss Function** with additional prior knowledge or better prerequisites for the model to learn or combine multiple loss functions in a multitask setting.

10.1. Input Principle: Input Masking and Augmentation

Input Masking and Augmentation

Idea Masking certain input features or augmenting the input data using transformations during training.

Goal Decorrelate/correlate new input features with the output to make the model more robust/sensitive to unimportant/important changes.

Domain Knowledge used What are relevant input features, and how can we alter them in a beneficial and meaningful way?

Type of Improvement Better generalizability and robustness

Task Requirements None

Data Requirements For input masking, the positions of the input features must be interpretable, e.g., words in a sentence or image regions/pixel positions. For augmentations, the input data must be augmentable, i.e., there must be clearly defined transformations that can be applied to the input data that do or do not change the initial output label for the input. For example, flipping images horizontally when detecting objects is usually not a problem, but flipping images of text is.

Humans and DL models learn from examples. For instance, to learn to recognize a dog, a human needs to see a dog and a DL model needs to be trained on images of dogs. Due to the nature of the human's vision system, we are able to separate the dog from the background and focus on it. Also, due to the knowledge about the environment, we are able to understand the interaction of light with the object and can thus interpret cast shadows and different appearances based on lighting. DL models do not have these properties. They find correlations between the input and the output. If the image backgrounds or the lighting conditions correlate with the output label, the model will learn to focus on these features. This is undesirable, since it makes the model less robust to changes in the input. If the background or the lighting conditions change, the model will not be able to make a correct prediction anymore. We already have shown such undesired correlations for Deep Metric Learning (DML) models in Chapters 7 and 9.

One way to prevent this is to collect larger training datasets that show a larger variety of input features, such as dogs in different environments, poses, and lighting conditions. While this has shown to be effective in improving the performance of DL models, it is not always possible or sustainable to collect more data. Thus, principles are needed to decorrelate certain aspects of the input from the output. One of them is the **Input Masking and Augmentation** principle, which hides certain input features or exchanges them during training. This way, the model cannot pick up correlations between the hidden or transformed input features and the output label. We hypothesize that as a

result, the model has to focus on generalizing better to new situations and is more robust to changes in the input.

The **Input Masking and Augmentation** principle can be applied to basically all tasks, but the choice of which input features to hide or transform is highly task- and modality-dependent as well as requires domain knowledge. There are mainly two approaches when applying this principle: keeping the output label intact by altering “unimportant” input features or deliberately changing the output label by altering “important” input features. For example, in image content classification, the brightness and contrast of the image are not crucial to detect the content of the image. We can say that they are not important for solving the task. Thus, augmentations that apply brightness and contrast filters to the input images can be used during training, since they should keep the output label unchanged [146]. However, in the Image Aesthetics Assessment (IAA) task, for example, the brightness and contrast of the image are important, since they severely influence the aesthetic quality of the image.

Usually, Input Masking and Augmentation do not aim to change the desired output but to create new training examples from existing ones. This is the approach we use in Chapter 11 for DML and later in Chapter 15 for sentiment analysis. However, combined with other knowledge about the input properties, it is also possible to augment the input such that the desired output also changes. If it is known how an input transformation affects the input, the new training examples can be used to guide the model to learn new differentiating features. For example, consider a dataset of images of arrows, showing different directions. When augmenting this dataset, we could integrate the knowledge that horizontally flipping an arrow that points right will result in an arrow that points left. This way, the desired output label changes, which is unusual for image augmentations, but is possible due to the knowledge about the input. Similarly, we apply augmentations and change the desired output of images in the IAA task in Chapter 16 where we apply image filters to aesthetic images in order to deteriorate them. Doing this, we know that the aesthetic value of the image decreases, which we can use to give the model comparisons between an aesthetic and the new unaesthetic image.

Modality-wise, input masking can be applied to all modalities, as long as the positions in the input have a certain meaning, so it can be decided whether this position can be masked to achieve the desired goal. In images, for example, the pixels semantically belong to different objects, so masking groups of pixels aims to remove certain objects or parts of objects from the training input. For text, masking or removing words from the input aims to remove certain semantic information from the input. We do this in Chapter 15 where we mask indicator words in texts that are important for sentiment analysis, aiming to let the model learn other correlations between the text and the output label. For input augmentations, transformations that can be applied to the input data that help to decorrelate certain input features from the output (see Chapter 15) or create new correlations that can be exploited for the training process (see Chapter 16) are essential.

10.2. Input Principle: Feature Extraction using Pretrained Multimodal Models

Feature Extraction using Pretrained Multimodal Models

Idea Using a pretrained multimodal model as a fixed feature extractor. These features can then be used to either train a new model on top of them or to use different modalities to solve the task without any training.

Goal Creating a more suitable input representation that can be used in the trainable model or that can even eliminate it altogether.

Domain Knowledge used Has the pretrained model potentially learned relevant features for the downstream task? Can we express knowledge about the task in other modalities?

Type of Improvement Fewer or no trainable parameters, less or no training data

Task Requirements Task objective can be modelled in combination with another modality.

Data Requirements A pretrained model must exist for the used modalities in the data. Additional modalities that can be handled by the pretrained model can be used to solve the task.

As already mentioned, one way to let a DL model generalize better is to use more data. However, collecting more labels for a specific downstream task often becomes infeasible or even impossible. This motivates the development of methods to reduce the required training dataset size to achieve good performance. To this end, using models that are pretrained on larger datasets for a different but related task has become a common practice. These models can then be used as feature extractors to create a more suitable input representation for the task at hand. As a result, the trainable model does not have to learn the features from scratch, but can focus on the task-specific correlations, even with few training examples. When taking an additional modality into account during the training of these models, using knowledge about one modality can be used to solve the task in another modality, e.g., knowledge about text descriptions can be used to solve an image classification task [214]. We thus focus on the potential of pretrained multimodal models for the **Feature Extraction using Pretrained Multimodal Models** principle, more specifically Contrastive Language-Image Pre-Training (CLIP) [214] (see Section 2.3.4). This model can be used to extract features from images and texts with aligned representation spaces. This way, multiple modalities can be used to solve the task, either by training the trainable model on these features or by using zero-shot capabilities of the pretrained multimodal model.

For this principle, we focus on CLIP, which works with images and texts. It aligns the representations of images with their corresponding text descriptions. Given this training objective, the model has to learn connections between the visual and textual

10.2. Input Principle: Feature Extraction using Pretrained Multimodal Models

representations of the same object, scenes, or actions. Given the large variety of language and descriptive capabilities, the image encoder extracts features from the image that can be (often) also described by text. On the other hand, the text encoder needs to extract features that are visually grounded in the image. For example, to align the representations of an image of “a painting of a man sitting on a chair” with its text, it needs to extract features that represent the objects “man”, “chair”, and “painting” as well as the action “sitting”, that connects two objects, from both the image and the text. Due to this, the variety of concepts the model can represent is larger than that of commonly used pretrained models that are based on content classification, which only has a strictly defined set of objects and usually no actions [46].

Given such a pretrained multimodal model, the features extracted from one of the encoders can be used as inputs to the trainable model when task-specific training data is available. In the same way, features from intermediate layers from the encoders can be used. We argue that due to the broad set of learned concepts from the multimodal data, the extracted features are better suited for many tasks than features from classification models. Overall, there are basically no task limitations when using pretrained multimodal models for feature extraction.

When no task-specific training data is available, those models can still potentially be used in a zero-shot setting. The original CLIP paper shows that with clever prompting, image classification tasks can be solved by the model without any training. In our implementations, we show that CLIP can also be used for zero-shot regression and retrieval tasks. Other work has shown that tasks such as image segmentation can also be solved in a zero-shot setting with CLIP [164]. Since CLIP is relatively new, there are still many possibilities for future work to explore other tasks that can be solved using this model.

Data-wise, the multimodal model needs to support the modalities that are used in the task. CLIP focuses on images and text, but there are other models that combine other modalities in the same way as CLIP does. For example, there is a line of work that incorporates audio into the same contrastive framework as CLIP [79, 302]. This way, the model can be used for zero-shot tasks on audio. While our implementations focus on images, they can potentially be easily adapted for the audio modality.

We also focus on models that have text as one of the multiple modalities, since it is usually used to describe labels for classification, regression, and different similarity notions for the retrieval task. It is widely available on the internet together with other accompanying media such as images, giving paired training data without much manual collection work.

When pairing modalities that do not contain text, the methods need to be adapted to the new modalities. For example, suppose we have a model that is pretrained on images and audio. Then, to classify an image, we need to define audio clips of the classes we want to detect. To classify images into the classes “cats” and “dogs”, we would need one or multiple audio clips of meowing and barking.

In our implementations, we use the fact that CLIP is pretrained on a large dataset and thus extracts broad features from images and texts that somehow correlate in the embedding space. In Chapter 12, we use image and text features to assess the aesthetics

of images using prompting and thus in a zero-shot setting, completely removing the need for a trainable model and training data. As an addition, we optimize a simple trainable model on top of the image features, using CLIP as a capable fixed feature extractor. In Chapter 13, we use CLIP’s image and text features to solve the DML task without training images. Here, we use text prompts to guide the optimization of a simple trainable model that learns to transform the image features into the desired embedding space, encoding user-defined similarity notions.

10.3. Output Principle: Weak Label Generation

Weak Label Generation

Idea Generating or collecting many weak labels for unlabeled data, which can be used for (pre-)training the NN, optionally along with hand-labeled data.

Goal Enlarging the training dataset.

Domain Knowledge used What heuristics and rules can be used to infer labels for the task, that might not be perfect but better than random?

Type of Improvement Better generalizability, less explicitly labeled training data required

Task Requirements None

Data Requirements There must be an unlabeled dataset which can be weakly labeled, i.e., it possesses certain regularities that can be exploited by a weak label process.

Larger training corpora usually improve the performance of DL models, since they better reflect the distribution of the data and include more examples of rare cases [145]. Often, the collection of new training examples is costly, as described in previous sections. While the previously described methods, such as image augmentations, enlarge the already existing training examples by altering the input data but keeping or deterministically changing the output label, there are often unlabeled examples available that could be used to generate new training examples. For those examples, no desired output is known. The **Weak Label Generation** principle proposes to use simple heuristics or models with stronger assumptions to generate imperfect labels for unlabeled data in order to enlarge the training dataset. Even though the new data is arguably noisy, the DL model can extract useful features from it.

When applying this principle to a task, usually only a small number of labeled examples is available, which is not enough to train the DL model from scratch. However, a large corpus of unlabeled data is present that has a similar input data distribution. Then, some knowledge about how the output is connected to the input can be expressed as a heuristic to label the unlabeled data. For example, for image classification, a large set of

images from the internet could be scraped along with their textual descriptions. Then, one heuristic could be to use images that contain a certain word in their description to label them with the corresponding class. This procedure is certainly noisy, but allows for a large weakly labeled dataset, where the label has a better chance of being correct than a random label. Jiang et al. [111] conduct a study to investigate how well an image classification model can be trained using a dataset that includes noise, i.e., wrong labels, based on this weak labeling technique. They show that a dataset which has 80% wrong labels because of the used heuristic still shows only a drop in performance of maximally 20% compared to a dataset with only correct labels. On the other hand, when randomly swapping labels, which imitates a random labeling and thus a weaker heuristic, the performance drops by up to 75%. The choice of heuristic is thus crucial for a successful application of this principle. As long as there is a heuristic that somehow captures some ideas about the labeling process, the **Weak Label Generation** principle can be applied to every task. We will explore this principle for Natural Language Processing (NLP) and Computer Vision (CV) tasks. In Chapter 15, we perform sentiment analysis using weak labels generated using sentiment lexica that label each word as positive, negative, or neutral. In Chapter 16, we generate weak labels for the IAA task by applying image filters to beautiful images. Here, the heuristic is that applying image filters such as brightness or cropping to a well-lit and composed image, it becomes less aesthetically pleasing.

10.4. Output Principle: Loss Function

Loss Function

Idea Choosing or modifying the loss function or combining multiple loss functions to better reflect the task's goal and to incorporate additional knowledge about the task.

Goal Guiding the model to a better solution faster.

Domain Knowledge used What does the task want to achieve? What kind of task-dependent assumptions can be used to achieve this goal?

Type of Improvement Better generalizability, faster convergence, using different task data effectively

Task Requirements None

Data Requirements None for the input data. Additional knowledge about the task is incorporated into the loss, which comes from other data sources than the training data. In case of multitask learning, additional output labels or even additional input data may be necessary.

The loss function is a standard component of the DL model training process and choosing a suitable one is crucial for the performance of the model and the convergence

speed of the training. This differentiable function measures how accurate the model's predictions are and guides the model to improve them given its gradients. Different loss functions emphasize different aspects that the model should pay attention to. The knowledge about the needed emphasis is usually goal-driven: If, for example, a goal is that outliers should be avoided, a loss function that penalizes large errors more than small errors is needed. If the absolute output of the model is not as important as the relation between different outputs, a ranking based loss might be more suitable than a regression loss. Sometimes, the goal is to let the model extract useful features to be used in later tasks. Then, the loss does not necessarily need to be related to the downstream task. Also, multiple tasks could be solved simultaneously to let the model extract a broader range of features [48]. Here, the loss function is a combination of the losses of the individual tasks. The loss combination strategy needs to be chosen carefully in order to ensure that all losses contribute accordingly to the overall loss. We explore a multitask pretraining strategy in Chapter 16, where we define different tasks on a pretraining dataset that guide the model to learn IAA related features. Modifying existing loss functions to incorporate additional knowledge that is otherwise not captured by the loss function is also possible. We explore this in Chapter 14 by enriching the Categorical Cross Entropy (CCE) loss function with class similarities, which leads to lower losses when the NN predicts the wrong, but closely related class.

As in the general DL training setting, the choice of loss function and its possible adaptations is task specific. Classification, regression, and representation tasks usually have different loss functions, as they want to achieve different goals. Loss functions are independent of the input data modality, but depend on the task target. For example, the CCE loss is used for multi-class classification tasks, independent of whether it is image classification, text classification, or any other kind of classification. The desired output for all multi-class classification tasks is a one-hot encoded vector over the possible classes. Hence, the type of output is the same and thus the same loss function can be used. This makes it possible to apply our proposed loss function modification in Chapter 14 to any classification setup, whenever the additional knowledge about the class similarities are known. Given other output structures or modalities, the loss function needs to be adapted as well. For example, the Structural Similarity Index Measure (SSIM) Loss is used for image regression tasks, where the goal is to predict an image. This loss function is specific to images as outputs and cannot be applied in text tasks.

10.5. Hybrid: Combining Input and Output Principles

The discussed principles can be applied in isolation to a certain task. In some cases, however, it can be beneficial to combine multiple principles. We call implementations based on multiple principles *hybrid* methods. Sometimes, mixing multiple principles might be necessary to avoid deterioration of the model's performance. For example, in Chapter 15, we explain a setting where using the **Weak Label Generation** principle for text classification alone might not add any value, since the model would just learn to imitate the weak label generation process. We thus explore the use of the **Input**

10.5. Hybrid: Combining Input and Output Principles

Masking and Augmentation principle to let the model improve on the weak labels.

Another implementation we provide uses three principles at once: For the IAA task in Chapter 16, we make use of the **Input Masking and Augmentation** principle, using augmentations for **Weak Label Generation**. Given these labels, we introduce multiple tasks with a multitask **Loss Function** to guide the model to learn useful features for the IAA task. Altogether, this results in better performance metrics for the finetuned model and substantially faster training times for the downstream task.

11. Making Deep Metric Learning (DML) Models More Robust to Background Bias using Background Augmentation

After we have found evidence that DML models seem to pay attention to unnecessary input properties when trained on images (see Chapters 7 and 9), we now explore the use of image masking and augmentation to improve such models. In this section, we first investigate the effect of the image background on DML models in more detail and then propose a method based on the **Input Masking and Augmentation** principle to make them more robust to it. The content of this section mainly follows our work in [130].

To recap, DML is the task of training a Neural Network (NN) to embed input items (in this case, images) such that embeddings of similar items are closer together than embeddings of dissimilar items [193]. This technique is often used in applications such as face recognition, person reidentification, and item retrieval [119]. For instance in item retrieval, a query image of an item is used to find semantically similar images by identifying the closest images in embedding space. Two images are deemed similar if they show the same item. Given this definition, the background of the images should not affect the embedding process, since objects can be photographed in different environments and thus appear in front of different backgrounds. Similar desired properties can be defined for other DML applications such as person reidentification.

Previous analytical work for the different task of **content classification** shows that NNs suffer from so-called *background bias*, i.e., they use information from the image background to identify the image category. For example, image classifiers trained to identify ships often focus on the water and not on the ship itself. This way, the classifier is not able to identify ships on land [142].

Since DML does not classify images but embeds them as continuous vectors, the findings on background bias from the literature are not directly transferable to these models. However, our visual analysis in Chapter 9 indicates that sometimes, the background is important for the DML model. If background bias was present in DML models, image backgrounds would influence the embedding process. Then, taking a picture of an object on the street or in a studio setup could lead to different search results when searched for in item retrieval methods, resulting in performance degradations of the item retrieval system. Figure 11.1 shows such a situation: Placing the bike in front of a brick wall or a studio backdrop gives completely different nearest neighbor search results, given a model that was trained on the Stanford Online Products (SOP) dataset. This is not desirable, since the retrieval system should only take the main object — in this case the bicycle — into account.

11. Making DML Models More Robust to Background Bias

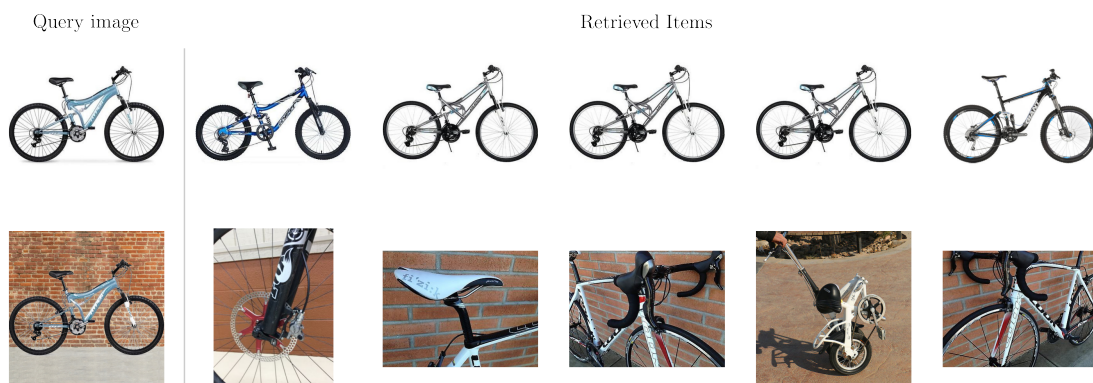


Figure 11.1.: Retrieval results for two query images (first column) based on the distance of embeddings of a DML model trained on the Stanford Online Products (SOP) [253] dataset with the Contrastive Loss [80]. The second query image shows the exact same object as the first one, but we exchange the background using an image editing software. Ideally, the embeddings for the two images should be similar since they show the same object, leading to similar retrieval results. However, both queries result in very different retrieval results mostly based on background similarity. While the first row only shows images that have a white background, the second one only shows images with patterns resembling the brick wall background in the query image. This behavior is not desirable in item retrieval systems.

We first investigate background bias in DML by conducting multiple experiments on the three standard DML datasets (Cars196 [139], CUB200 [275], Stanford Online Products [253]) and five different DML loss functions (see Section 2.4.3). We design a test setting where we replace image backgrounds with other images and measure the retrieval’s performance drop compared to the unmodified images; larger drops in performance indicate that the model relies more on the background. We show that, depending on the dataset, models can suffer from severe background bias. To combat this behavior, we apply a simple but effective training strategy based on the **Input Masking and Augmentation** principle that does not require any additional manual labeling work or model changes and keeps the same inference times. Here, we extract the main object from the images during training using a *salient object detection method* [213] and put them onto randomly selected background images. We show that this technique, which we call BGAugment, indeed improves performance in our test setting, even though no foreground/background segmentation is available during testing, indicating that the model learns to focus less on the background. To verify this, we qualitatively and quantitatively analyze the resulting models and show that the model trained with BGAugment attends more to the main object instead of the background, leading to better performance when backgrounds change. For this, we introduce a metric that quantifies the focus of the model on the foreground and background.

11.1. Methodology

We first propose a test setting to quantify how prone the DML model is to background bias. Given this measure, we can then propose our method to make the models more robust to it. Given the test setting, we can measure the improvement of our method on the DML setting.

11.1.1. Test Setting

In this section, we introduce our new test setting that quantifies the dependence of trained DML models on the image background. Intuitively, the more a DML model attends to the background of images to generate an embedding, the larger the change of the embeddings when changing the image’s background. In turn, when randomly changing the background of test images in the DML setting, the retrieval performance is expected to drop substantially if the model pays much attention to the background.

We can assume that in item retrieval, the most salient object in an image is the object that was intended to be photographed. Thus, for each image $\mathbf{I} \in \mathbb{R}^{h \times w \times 3}$ (RGB image with width w and height h) in the test dataset \mathcal{D} , we identify the main object and create a binary mask $\mathbf{M} \in [0, 1]^{h \times w}$ separating the most salient object from the background (1/white denotes main object, 0/black denotes background). In order to obtain such masks, we use the salient object detection NN U²-Net [213]. It is designed to detect the most salient regions — in our case the main object — in the image and outputs a binary mask that separates the object from the background. We verify the segmentation quality of the network by computing the average overlap of generated and hand-annotated masks on a randomly sampled subset of images for each tested dataset. Overlap is defined as the percentage of the ground truth foreground area that is also covered by the generated mask. We use overlap as a metric since it is more important to cover the relevant parts of the image than removing the background. On average, the automatic mask generator has an overlap of more than 90% with the manual annotated binary masks for all datasets, so the generated masks mostly cover the relevant parts of the image.

In addition to the image \mathbf{I} and mask \mathbf{M} , we collect a dataset of background images scraped from the popular stock photo website Unsplash¹. We filter the dataset such that no obvious foreground objects are present in the images. The resulting background image dataset \mathcal{B} contains one hundred images and shows solid colors, color gradients, background objects, and abstract patterns.

During testing, for each image \mathbf{I} , we sample a background photo $\mathbf{B} \in \mathcal{B}$ to create a new test example \mathbf{I}' with

$$\mathbf{I}'_{::,c} = \mathbf{M} \cdot \mathbf{I}_{::,c} + (\mathbf{1} - \mathbf{M}) \cdot \mathbf{B}_{::,c} \quad (11.1)$$

for each color channel $c \in \{\text{red, green, blue}\}$. Here, \cdot is the element-wise multiplication and $\mathbf{1} = \mathbf{1}^{h \times w}$, i.e., a matrix with the same size as \mathbf{M} consisting of ones. We call the newly created test dataset \mathcal{D}' the “corrupted” test set, while the original test dataset \mathcal{D} is called the “clean” test set. Both datasets \mathcal{D} and \mathcal{D}' are fed separately through the

¹Scraped from <https://unsplash.com/s/photos/background> on 2022-05-13

11. Making DML Models More Robust to Background Bias

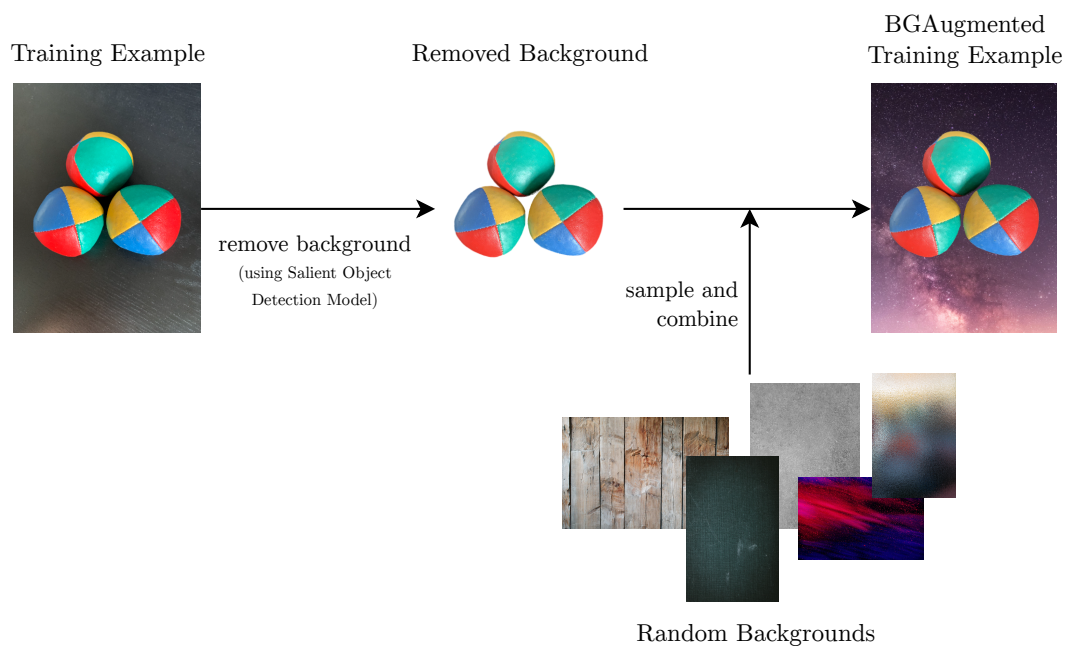


Figure 11.2.: Illustration of the BGAugment method. The background of each training image is replaced with a randomly sampled background image. This decorrelates the background of images with their embeddings.

trained model and a common evaluation metric is computed for the resulting embeddings. Here we use Mean Average Precision at R (MAP@R) [193]. We can then observe the drop in performance from the clean to the corrupted dataset. In order to average out the influence of background samples, we run our test setting five times and report means and standard deviations.

11.1.2. BGAugment: Background Replacement During Training

In order to combat background bias in DML, we apply a simple but effective strategy during training inspired by the literature about background bias in classification networks. We call this method BGAugment and show it in Figure 11.2. Similar to the test setting description, we replace the background of input images. However, we do this during training and validation. To not interfere with the background images used in the test set, we collect another one hundred background images from Unsplash.

During each training iteration, we sample a random background image for each training image and use the automatically generated binary mask from the salient object detection model to replace the image background. Since we use a pretrained salient object detection model [213], there is no need for additional manual data labeling. Also, since we do not apply the background replacement during inference, there is *no computational overhead when applying the model in production*. Additionally, BGAugment only touches the data

loading process, leaving all other training components such as the model or loss function intact, allowing for fast adoption of this technique.

11.2. Experiments

In our experiments, we compare five different loss functions. Given our insights from Chapters 7 and 9 that there are differences between classification and ranking based losses, we evaluate overall five loss functions: the three ranking losses Contrastive Loss [80], Triplet Loss [288], and Multi Similarity Loss [284], as well as the two classification losses ArcFace Loss [47] and Normalized Softmax Loss [159, 276, 312]. This should allow us to identify differences in their performances. We perform experiments on the three standard benchmark datasets for DML: Cars196, CUB200, and SOP.

11.2.1. Training and Evaluation Setup

We mostly follow the training procedure described by Musgrave et al. who design a fair setting to compare different DML loss functions [193]. As a model base, we use a BatchNorm Inception [108] network pretrained on ImageNet [46] with frozen Batch Normalization layers. The last fully connected layer is replaced to output 128-dimensional vectors. The outputs are normalized to unit length to stabilize training. The model is trained with a learning rate of 10^{-6} on the first 80% of training classes and validated on the remaining 20%. Musgrave et al. report the best hyperparameters for each loss function on each dataset by conducting a cross validation. We adopt these hyperparameters for our experiments and use them for all tested methods, i.e., there is no difference in hyperparameters for the BGAugment runs. While a dedicated hyperparameter search might improve the performance, we want to investigate how well BGAugment performs when just applied to an existing model setup. More information on the training process can be found in the original paper by Musgrave et al. [193].

11.3. Results

Table 11.1 shows the means and standard deviations of the MAP@R [193] for our experiments. Depending on the dataset, the drop in performance from the clean to the corrupted test set can be severe. While for Cars196, the performance drops by only around 2 to 3 percentage points for all loss functions, CUB200 and SOP show much larger differences (approx. 8 and 34 percentage points, respectively). That gives a relative performance drop of around 20%, 40%, and 85% for Cars196, CUB200, and SOP, respectively. Even though the performance of all models on the corrupted dataset is still better than randomly sampling embeddings, the drop in performance is substantial. We hypothesize that this is due to the training datasets' properties. The images in Cars196 show cars in different environments, so the background does not often correlate with the similarity between images. Since the birds in the CUB200 dataset are shown in their natural habitat, the environment gives clues about the similarity between images. For

11. Making DML Models More Robust to Background Bias

Table 11.1.: Mean and standard deviations of MAP@R for our experiments. All values are given in percent. The best results between the vanilla and BGAugment models are written in bold.

	Cars196		CUB200		SOP	
	clean \uparrow	corrupted \uparrow	clean \uparrow	corrupted \uparrow	clean \uparrow	corrupted \uparrow
Contrastive	15.22	13.31 \pm 0.15	20.27	12.47 \pm 0.08	37.98	4.17 \pm 0.05
+ BGAugment	16.43	16.31 \pm 0.03	19.39	15.46 \pm 0.08	31.52	24.09 \pm 0.04
Triplet	15.33	13.40 \pm 0.04	18.29	10.94 \pm 0.05	36.71	6.05 \pm 0.06
+ BGAugment	15.28	15.19 \pm 0.05	18.56	15.33 \pm 0.16	29.63	21.39 \pm 0.07
Multi Similarity	18.80	16.25 \pm 0.06	19.19	12.21 \pm 0.16	39.65	6.51 \pm 0.06
+ BGAugment	15.61	15.48 \pm 0.05	18.93	15.65 \pm 0.16	32.67	24.00 \pm 0.07
ArcFace	16.25	13.63 \pm 0.08	20.66	12.87 \pm 0.16	40.50	6.98 \pm 0.05
+ BGAugment	16.19	16.23 \pm 0.04	20.91	17.92 \pm 0.08	22.25	16.53 \pm 0.05
Normalized Softmax	18.06	15.36 \pm 0.03	20.18	12.41 \pm 0.10	41.52	7.67 \pm 0.04
+ BGAugment	18.15	18.12 \pm 0.05	20.60	17.39 \pm 0.10	32.16	24.60 \pm 0.03

example, there are waterbirds and landbirds in the dataset, thus the background features can be used to differentiate between them. The background influence is the strongest for the eBay product images in the SOP dataset. Images of one product are often taken in the same environmental conditions. The DML model then picks up these features to embed the image, since they are similar across images of one class and thus can be used to find similarities between the images.

Between loss functions, we observe no large difference in drops and overall performance on all datasets, indicating similar vulnerability to background bias of ranking and classification based losses. The use of BGAugment improves the models' performance on the corrupted dataset, except for the Multi Similarity Loss on the Cars196 dataset. On Cars196 and CUB200, BGAugmented models perform similarly or even outperform their base model on the clean dataset. This means that the backgrounds of images in Cars196 and CUB200 are not necessary to achieve good performance.

On the other hand, applying BGAugment to models trained on SOP improves the performance on the corrupted dataset significantly but shows large performance drops on the clean dataset. We hypothesize that the high performance without BGAugment is only achievable by exploiting the background. In other words, the good performance of models on the clean SOP dataset is misleading in terms of item retrieval, since the models are not able to keep up the performance when backgrounds are exchanged during training. In realistic item retrieval settings, query images most often show other backgrounds than the images in the database.

11.4. Analysis

To better understand the improved test performance of BGAugment, we visualize the input pixels the models are most sensitive to, using the DML attribution map generation method introduced in Chapter 9. Figure 11.3 shows attribution maps of the Normalized Softmax Loss model trained with and without BGAugment. BGAugment has a darker

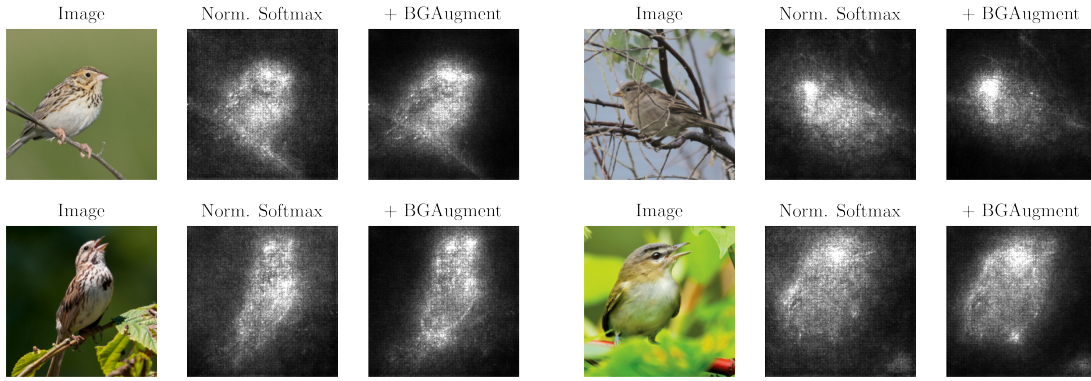


Figure 11.3.: Four images from the CUB200 test set and their corresponding attribution maps (Normalized Softmax Loss model with and without BGAugment). The model focuses more on brighter areas. The base model shows some attention on the background, while the BGAugment model has sharper focus on the main object.

background area and thus focuses more on the main object in the image.

To quantify this, we introduce a new metric that measures how much of the model’s focus is on the main object. Given a ground truth mask $\mathbf{M} \in [0, 1]^{h \times w}$ and an attribution map $\mathbf{A} \in \mathbb{R}_+^{h \times w}$ of a model for an image $\mathbf{I} \in \mathbb{R}^{h \times w \times c}$. Our ideal metric has the following desired properties: (1) The best possible value is one, i.e., the model attends to only the foreground area. (2) If the model does not focus on anything, but rather distributes its attribution uniformly, the metric’s value should be zero. This makes it possible to compare two images where the foreground areas are of different size. If, for example, the main object has double the pixel count, then a naive metric that measures the percentage of attribution that is on the foreground is also doubled when the attribution is in fact uniformly distributed. We thus need to account for the size of the foreground object in the metric. Overall, we propose the following score:

$$f(\mathbf{M}) = \sum_{i,j} \frac{\mathbf{M}_{i,j}}{w \cdot h} = \frac{1}{w \cdot h} \sum_{i,j} \mathbf{M}_{i,j} \quad // \text{ percentage of foreground} \quad (11.2)$$

$$a(\mathbf{M}, \mathbf{A}) = \frac{\sum_{i,j} \mathbf{M}_{i,j} \cdot \mathbf{A}_{i,j}}{\sum_{i,j} \mathbf{A}_{i,j}} \quad // \text{ percentage of attribution on the foreground} \quad (11.3)$$

$$\text{score}(\mathbf{M}, \mathbf{A}) = \frac{a(\mathbf{M}, \mathbf{A}) - f(\mathbf{M})}{1.0 - f(\mathbf{M})} \quad // \text{ final score normalized by foreground size} \quad (11.4)$$

Our previous experiments have shown the dependence of models on the background only indirectly, by showing that their performance drops substantially when replacing backgrounds. With this metric, we can directly quantify the attribution that the model

11. Making DML Models More Robust to Background Bias

Table 11.2.: Means and standard deviations for our analysis using our proposed metric (Equation (11.4)) to quantify the attribution of the model on the foreground. Lower values indicate larger dependence on the background.

	Cars196 \uparrow	CUB200 \uparrow	SOP \uparrow
Contrastive	0.50 ± 0.09	0.25 ± 0.08	0.07 ± 0.16
+ BGAugment	0.53 ± 0.10	0.32 ± 0.10	0.31 ± 0.20
Triplet	0.50 ± 0.09	0.24 ± 0.08	0.08 ± 0.17
+ BGAugment	0.54 ± 0.09	0.33 ± 0.09	0.26 ± 0.22
Multi Similarity	0.52 ± 0.09	0.26 ± 0.08	0.11 ± 0.17
+ BGAugment	0.50 ± 0.09	0.30 ± 0.09	0.28 ± 0.19
ArcFace	0.49 ± 0.10	0.25 ± 0.08	0.13 ± 0.14
+ BGAugment	0.56 ± 0.10	0.33 ± 0.09	0.26 ± 0.17
Normalized Softmax	0.51 ± 0.09	0.24 ± 0.08	0.12 ± 0.14
+ BGAugment	0.56 ± 0.09	0.30 ± 0.08	0.28 ± 0.20

assigns to the foreground. Lower values thus signal more dependence on the background. If the metric is negative, the model attends more to the background than the foreground. Overall, the metric can achieve values in the interval $(-\infty, 1]$. We apply it to all trained models and test datasets and show means and standard deviations in Table 11.2. It shows that models trained with BGAugment achieve higher values than their basic training counterparts, i.e., are less dependent on the background. Overall, however, none of the mean scores is negative, meaning that models trained without BGAugment are also focusing on the foreground for the most part.

11.5. Conclusion

We have shown that DML suffers from background bias. Our experiments show that performance can drop substantially when backgrounds are exchanged. Exchanging image backgrounds during training using a salient object detection network improves performance while having neither model changes, additional parameters, nor increased inference time. Our qualitative and quantitative analyses confirm that models trained this way focus more on the foreground, leading to better robustness regarding background changes. While automatically generated masks from the state-of-the-art salient object detection network mostly isolate the relevant object, masking accuracy is not perfect. Our hand-annotated samples show that, on average, up to 10% of the ground truth foreground is not present in the generated mask. While this might partially explain the performance drop in our experiments, we can observe the dependence of standard trained models on the background in Figure 11.3 without needing to trust the mask generation process.

Investigating and combating background bias in DML is beneficial to the development of retrieval settings such as item retrieval or person reidentification systems. Our work

suggests that such systems need to be trained carefully in order to find relevant images without simply relying on unimportant background information. Implementing the **Input Masking and Augmentation** principle can help substantially.

12. Using Contrastive Language-Image Pre-Training (CLIP) for Image Aesthetics Assessment (IAA)

We now introduce a first implementation of the **Feature Extraction using Pretrained Multimodal Models** principle that mainly follows our work in [90]. Here, we use the large multimodal model CLIP [214] to solve the IAA task. CLIP is pretrained on a large dataset of images and texts. We freeze its weights during its usage for the downstream task. Technically, due to not optimizing its weights, we argue that it is not a trainable model as in our definition of the Deep Learning (DL) training setup. We instead use CLIP as a fixed general feature extractor for images and texts, i.e., a preprocessing step in the input pipeline, using our domain knowledge that it should be able to extract aesthetic features. These features can then be used to train a model on the target dataset or can be combined with knowledge expressed in another modality to solve the task in a zero-shot setting. In this section, we describe both of these methods and show that they can be used to solve the IAA task, making it easy to reduce the number of trainable parameters of the model.

Usually, IAA methods based on Deep Neural Networks (DNNs) are built on top of models that were trained on the ImageNet classification task [94, 122, 149, 167, 242, 265, 311]. We argue that content classification is not well-suited as a pretraining task for IAA methods, since the model is optimized to be invariant to important factors of image aesthetics, e.g., contrast, lighting, or composition, as these are not important to identify the content of an image.

In contrast to content classes, natural language can provide much richer descriptions. Among others, it can describe styles (“a high-contrast image of a dog”, “a black-and-white portrait”), compositions (“a man standing next to a chair”, “an image of a house with the sun in the upper right corner”), or can even directly express the subjective feeling of aesthetics (“a beautiful sunset”, “an ugly sweater”). We hypothesize that models trained using natural language supervision are better suited for the IAA task, since they extract broader and more useful features, since they can be related to a larger set of concepts. In this section, we utilize the CLIP [214] model for our experiments. We have given an introduction to CLIP in Section 2.3.4. In the original paper, CLIP’s capabilities only have been shown on *content classification* datasets [214]. For the IAA task, however, the extraction of both, *content and style* features is important, so it is not obvious that CLIP is able to perform well on this task. We hypothesize that, due to the language guided training objective, CLIP’s image encoder also extracts useful features for the IAA task, such as lighting, composition, and properties of beauty ideals, such that it serves as a

12. Using CLIP for IAA

better base model for IAA networks than commonly used ImageNet classification models.

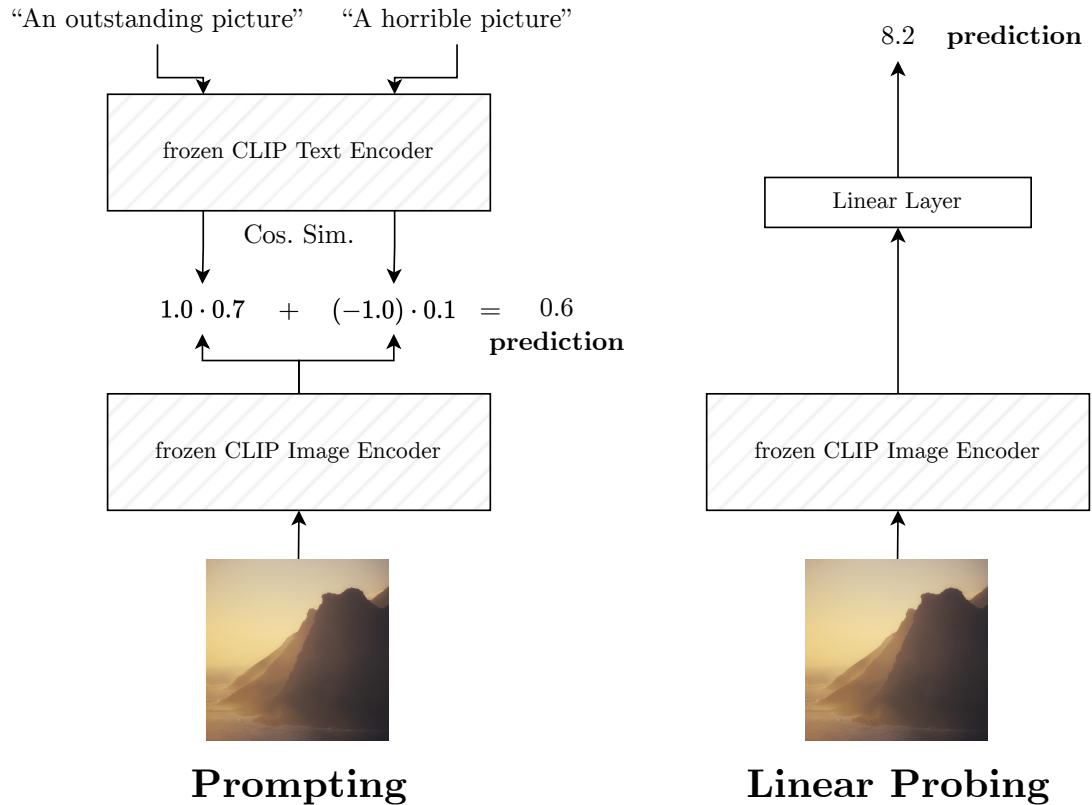


Figure 12.1.: A schematic overview of our investigation on the suitability of CLIP as a base model for the Image Aesthetic Assessment task. Hatched elements are components with frozen weights. Prompting does not modify any parameters, leading to a Zero-Shot Learning (ZSL) scenario by eliminating the trainable model from the standard DL training setting. Linear Probing, on the other hand, trains a Linear Regression (LR) on top of the last hidden activations of CLIP’s image encoder.

To test our hypothesis, we conduct a study to investigate the suitability of CLIP for the IAA task, which in turn shows the effectiveness of the Feature Extraction using Pretrained Multimodal Models principle. As a ground truth for this very subjective task, we utilize the Aesthetic Visual Analysis (AVA) dataset and estimate CLIP’s performance on the IAA binary classification task (“aesthetic”/“not aesthetic”) and ranking of images (predicting a mean aesthetic score over user ratings). A schematic overview of our two approaches is shown in Figure 12.1: We use prompting, which uses no trainable parameters (i.e., Zero-Shot Learning (ZSL) for this task), and Linear Probing, which uses CLIP as a fixed feature extractor for a linear layer. We compare the results to state of the art IAA models and an ImageNet pretrained Vision Transformer (ViT) with a comparable model size as the CLIP image encoder. With this, we hope to show that CLIP’s features are more suitable for the IAA task than ImageNet features, which is commonly used as a base

model in the literature. Also, following most IAA methods [94, 167, 242, 265, 311], we finetune both the ImageNet ViT and CLIP’s image encoder on the AVA training dataset to have an upper limit of how well CLIP might perform when using it as the trainable model.

As a first experiment, inspired by CLIP’s ability to classify images without explicit training using only natural language prompts [214], we test multiple ways of prompting CLIP to estimate the aesthetic appeal of images. We use fixed prompts, add context to better reflect the content of images, and create ensembles of prompts for binary predictions. To overcome the issue that only classification tasks can be solved by finding the most similar text prompt, we introduce a method to convert multiple positive and negative prompts to a continuous score by calculating a weighted sum over the prompt similarities. Our results show that plausible, carefully chosen prompts can beat simple baselines without any model training, which shows that CLIP extracts features correlated with subjective adjectives such as “outstanding” or “horrible”. This shows that we are able to solve this task using only a pretrained model without any additional training, i.e., excluding the trainable model from the DL training framework.

Second, we use CLIP’s image encoder as a static feature extractor and train a Linear Regression (LR) on top of it, called Linear Probing [214]. We compare the results to the same regression optimized on features extracted from a comparable Vision Transformer (ViT) [50], which has been pretrained on the ImageNet classification task. We show that CLIP’s features substantially outperform ImageNet features, indicating that they extract more useful information for the IAA task from the image. In fact, Linear Probing CLIP achieves a performance competitive to fully finetuned ImageNet models, while only optimizing 768 model parameters. This shows that feature extraction using a pretrained multimodal model can give competitive results, minimizing the size of the trainable model in the DL training setting. Overall, our investigations show that CLIP is well-suited as a base model for the IAA task, since it extracts more useful features from the image, even with frozen weights.

12.1. Methodology

We explore multiple methods that make use of the fixed CLIP model. We start with different variants of prompting, which is a method that has been used to show that CLIP can solve classification tasks without any training [214]. Here, we explore different styles of prompting to assess the best possible performance without training any model. We also perform Linear Probing, where we train a linear classifier or regressor on top of the fixed CLIP image encoder. This introduces a small trainable model. We later evaluate the methods on the binary as well as continuous tasks for IAA, as described in Section 3.2.

12.1.1. Prompting

CLIP has been shown to have good “zero-shot” performance on classification tasks by prompting the network with text inputs based on the desired labels and measuring the similarity to the image’s feature vector. More formally, CLIP’s image encoder represents

12. Using CLIP for IAA

an input image (resized and center-cropped to 224×224 pixels) as a 512-dimensional vector \mathbf{i} . CLIP’s text encoder encodes all possible class labels \mathcal{C} , getting 512-dimensional vectors \mathbf{t}_c for $c \in \mathcal{C}$. Often, string templates, i.e., text strings that allow the insertion of variable values into specific designated locations within them, are used to embed the class labels into a coherent natural language text prompt before feeding it into CLIP’s text encoder [214]. The label whose vector has the highest cosine similarity to the image vector $\mathit{argmax}_{c \in \mathcal{C}} \cos(\mathbf{i}, \mathbf{t}_c)$ is chosen as the prediction.

It is important to note that there is no training involved in this method. Since CLIP was trained on natural language, any text label can be used for the classification task. In our setting, we use adjectives used to describe aesthetic or unaesthetic images as labels, e.g., “beautiful” or “ugly”. To collect possible labels, we search synonyms for “ugly” and “great” in WordNet [183] and augment the list of positive adjectives with “beautiful” and “pretty”, since these were not part of the list. Overall, we get 27 positive and 12 negative words.

Given descriptive words for aesthetic or unaesthetic images, we can construct prompts by embedding them into a string template that are then used to measure the similarity to the image vector. In the following, we describe and experiment with different prompting methods that we term as fixed prompts, prompts with context, and ensembling. All the prompting methods are designed for classification tasks, thus are naturally applicable to the binary classification task of IAA. Predictions are made by finding the most similar prompt to the image. For the continuous task, however, it is necessary to predict a single scalar score that encodes both, positive and negative image aspects. We thus propose a simple but effective strategy to compute scores from prompts.

Intuitively, prompts that suit an image better should be more similar to the image’s embedding, while non-fitting prompts are less similar. Thus, high similarity between a positive prompt and the image should lead to a high image aesthetic score. Correspondingly, if the negative prompt has high similarity, the score should get lower. We build on this intuitive idea and calculate the predicted score for an image and all prompts we use by weighting the similarity of each prompt to the image with one or minus one, depending on the prompt’s incentive:

$$\mathit{score}(I) = \sum_{c \in \mathcal{C}} \mathit{sim}_{\cos}(\mathbf{i}, \mathbf{t}_c) \cdot w_c \quad (12.1)$$

$$w_c = \begin{cases} 1, & \text{if } c \text{ is a positive label} \\ -1, & \text{if } c \text{ is a negative label} \end{cases} \quad (12.2)$$

Since the cosine similarity sim_{\cos} is in range $[-1, 1]$, very dissimilar negative (positive) prompts make the score prediction higher (lower). In the following, we describe the prompting methods we explore.

Fixed Prompt

Our fixed prompt approach utilizes exactly two prompts, one for aesthetic images, one for unaesthetic images. Both prompts are formed using the string template “a [label] picture”, where [label] is either a positive or a negative word from our list of adjectives.¹ Given these two prompts, we find the one more similar to the image using CLIP and predict its label for the binary classification task. The weighted score prediction described above is used for the continuous task.

Context-Aware Prompts

Fixed prompts do not account for the content of the image. Instead of the generic prompt “a beautiful picture”, we hypothesize that it is better to include the content of the image, e.g., “a beautiful picture of a dog”. This specification of the text prompt moves the encoded prompt vectors closer towards the image vector, thus reducing noise in similarity measurements and maybe helping with the improvement on the IAA task. Since it is not known what is in the photo, we use the text descriptions of all 1000 ImageNet [46] classes.² We then construct a prompt using the string template “a [label] picture, of a #[content class]_i”, where [content class]_i is the *i*th class name of ImageNet. Including the content as a hashtag showed performance improvements in preliminary experiments and accounts for the internet-based origins of the dataset used to train CLIP.

Instead of two prompts as with the fixed prompt approach, we now have 2000, i.e., 1000 for each positive/negative label. The label of the closest text prompt to the image vector is used as the prediction in the binary task, while the weighed score is calculated across all prompts in the continuous setting.

Ensembling

Our ensembling approach is structurally similar to the context-aware prompts. However, we condense the 2000 prompts down to two vectors by averaging all prompt vectors of each aesthetic label.

12.1.2. Linear Probing

Most IAA models use a classification Neural Network (NN) pretrained on ImageNet [46]. We also want to investigate the question whether CLIP’s features (as taken from the pretrained model) are more suitable for the IAA task than features from an ImageNet classification model. Intuitively, CLIP extracts broader features than an ImageNet model, since for classification, only the content of the image is important. Features describing certain aspects of the image are not needed for classification, especially features like

¹In preliminary experiments, we found that only giving the adjectives as prompts did not work well and “picture” in the prompt string template usually performed better than other words like “photo” or “image”.

²Class names from <https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a> (last accessed: 2023-02-10), using only the name up to the first comma for each class.

12. Using CLIP for IAA

lighting, contrast, and other properties of the image that are important for IAA. On the other hand, CLIP’s language-based training learns broader features, since language is more descriptive and thus more nuanced than class labels.

For the experiments in this section, we train a LR on the features of CLIP’s image encoder to predict AVA’s training images’ mean score. We take the activations from the last layer before the output, which provides 768-dimensional feature vectors. The output of the LR model is a scalar value, which is thresholded at five to predict binary classes (as are the labels) while staying untouched for the continuous task.

12.2. Experiments

In this section, we present the experiments we conduct to investigate the usefulness of CLIP’s features for the IAA task. We apply each of the prompting as well as the Linear Probing methods to the binary and continuous IAA tasks. We evaluate the results using the AVA dataset and the common metrics for the IAA task, i.e., Accuracy for the binary task and Spearman and Pearson correlation coefficients for the continuous task. For our experiments, we always use CLIP’s “ViT-B/32” variant for the image encoder.

For the fixed prompt method, we evaluate all combinations of positive and negative prompts from our collection of adjectives. The labels performing best on AVA’s training dataset for the binary task are “outstanding” and “atrocious” for positive and negative prompts, respectively. For the continuous task, “outstanding” and “horrible” perform best on the training dataset based on the Spearman correlation. We use these combinations for our evaluation setting.

When adding context to the prompts, we find that “smashing”/“horrible” are the best labels to use for the binary task on the training dataset. In the continuous setting, the labels “outstanding”/“horrible” perform best. In the ensemble setting, “outstanding”/“horrible” are the best labels in the binary as well as the continuous task. Generally, “outstanding” and “horrible” can be seen as sensible defaults that always perform well and almost always best.

Baselines

We compare our results to results reported in the literature. We train and test our methods on the same datasplit as MP_{ada} [242], A-LAMP [167], [311], and MUSIQ [122] and compare our results to these methods. Note that PA_IAA [149], NIMA [265], and MLSP [94] certainly train and validate their models on a different dataset split as the other models, thus their stated test results are only mentioned for reference. To allow a fair comparison, we display the results of a NIMA reimplementation [148] that has been trained on the same datasplit as the other models. In addition, we train and evaluate MLSP on our datasplit using the code for the original implementation of MLSP provided by Hosu et al. [94].

We also introduce a simple Majority Voting baseline that always predicts the positive class in the binary classification task. Since the distribution of labels is skewed towards aesthetic images, always predicting the positive class already gives 70.3% accuracy. Due

to no variation in the predictions, correlations cannot be computed, thus this baseline is not available for the continuous task.

In order to better estimate the performance gain of using CLIP instead of the commonly used ImageNet pretrained models, we compare our results for the Linear Probing method with results obtained by using features from a ViT³ [50, 292] that is very similar in size to CLIP’s image encoder (also of type “ViT-B/32”), but was trained on ImageNet21k [223]. According to Radford et al. [214], the architectural difference between ViT and CLIP’s image encoder are additional layer normalizations and a slightly different initialization scheme. We thus argue that it is fair to compare both models.

Finally, as an upper-bound for what can be achieved with a CLIP based model, we also finetune CLIP and the Vision Transformer on the AVA training dataset. We replace the last layer of both models with a linear layer with ten outputs, each representing one possible score. A softmax activation function converts the outputs into distributions that can be compared to the target score distribution. We employ the Earth Mover’s Distance (EMD) loss [233] as utilized by Neural Image Assessment (NIMA) [265] and finetune the models using a batch size of 128, the Stochastic Gradient Descent (SGD) optimizer with momentum of 0.5 and a learning rate of 0.01 for the classifier head and 0.0001 for the rest of the model. Besides resizing and center-cropping the images to 224×224 pixels, no data augmentation is applied. Early stopping trains the model until the loss on a 10% subset of the training data does not improve for ten consecutive epochs.

12.3. Results

Table 12.1 shows the results of all our models. Using fixed prompts (row “Fixed Prompt CLIP”) already shows better performance than always predicting the majority class. On the continuous scale, the predictions have moderate correlations with the ground truth. This indicates that CLIP extracts features from images that correlate with descriptive adjectives such as “outstanding” or “horrible”.

Adding context to the prompts (row “Context-Aware Prompt CLIP”) improves the results in all evaluation metrics, especially in the continuous task. To better understand the effects of adding content classes to the prompts, we visualize some test images with their binary prediction, their actual AVA mean score, as well as the content class that was used in the most similar prompt in Figure 12.2. We can observe that the closest content class tends to describe the content quite well or approximates its visual appearance. The predicted binary aesthetic label is mostly correct. Overall, the use of content classes improves the results by moving the text prompt vector closer to the image vector. Then, the choice between an aesthetic and an unaesthetic image is made depending on the content of the image.

Ensembling the context-aware prompts into two vectors (row “Ensembling Prompt CLIP” in Table 12.1) further improves the binary task performance while being computationally less expensive, since only two instead of 2000 comparisons have to be done for each image.

³The pretrained model can be downloaded with the timm Python library under the model name ‘vit_base_patch32_224_in21k’.

12. Using CLIP for IAA

Table 12.1.: Comparison between the results of all our models and the results of existing models, which were published in their respective papers. “n/a” refers to metrics not stated in the original work. Methods with “—” for Spearman and Pearson correlation are only designed to solve the binary task. Methods in grey rows do not state the used training/test split or do not use the same split as ours, thus we only reference them for context. In the case of NIMA, we refer to the results of a reimplementation trained on the correct datasplit [148]. For MLSP, we use the original code provided by Hosu et al. and train it on our datasplit. The other models use the same training/test split as our work.

Method	Accuracy \uparrow	Spearman \uparrow	Pearson \uparrow
PA_IAA (Inception-V3) [149]	0.837	0.677	n/a
NIMA (Inception-V2; paper results) [265]	0.815	0.612	0.636
MLSP (InceptionResNet-V2; paper results) [94]	0.817	0.756	0.757
Majority Voting Baseline	0.703	—	—
MP _{ada} [242]	0.830	—	—
A-LAMP [167]	0.825	—	—
Zeng et al. (ResNet101) (2020)	0.808	0.719	0.720
MUSIQ [122]	0.815	0.726	0.738
NIMA (MobileNet; reimplementation) [148]	n/a	0.626	0.609
MLSP (InceptionResNet-V2; original code trained and evaluated on our datasplit) [94]	0.808	0.714	0.728
Finetuned ViT (ImageNet21k)	0.793	0.660	0.675
Finetuned CLIP	0.816	0.731	0.741
Fixed Prompt CLIP	0.725	0.435	0.453
Context-Aware Prompt CLIP	0.737	0.539	0.554
Ensembling Prompt CLIP	0.756	0.539	0.554
Linear Probing ViT (ImageNet21k)	0.767	0.574	0.587
Linear Probing CLIP	0.800	0.683	0.694



Figure 12.2.: Example images with the binary prediction by the Context-Aware Prompts for CLIP, the actual score by AVA (in parenthesis), and the content of the prompt with the highest similarity to the image. The predictions and actual scores are colored to easily find the predictions that are correct or wrong. Image credit is given in gray (name or username of photographer on dpchallenge.com).

Overall, our prompting results show that CLIP extracts features that can be used for the IAA task. These features correlate with text prompts describing the corresponding aesthetic value of the image. Overall, the labels “outstanding” and “horrible” are well suited for this task. In an application, it might be possible to get acceptable results if only a pretrained CLIP model is available.

In our Linear Probing method, we train a LR on top of the features of CLIP’s penultimate layer. Due to the training, the results exceed the prompting approach by approx. 4% points in Accuracy and approx. 0.14 in correlations (see row “Linear Probing CLIP”). Compared to the ImageNet pretrained ViT baseline, using CLIP as a static feature extractor is much more effective for the IAA task. It achieves approx. 3% points higher accuracy and improves both correlations by approx. 0.1. In fact, our results approach competitive performance to IAA models that optimize all network weights, such as the method introduced by Zeng et al. [311]. Their method finetunes a ResNet101 pretrained on ImageNet using the Cross Entropy (CE) loss, while also changing the target score distributions based on manual examination of AVA’s target labels. Our results are obtained by simply minimizing the Mean Squared Error (MSE) of the predicted and target mean score. Using more sophisticated loss functions and target labels might

12. Using CLIP for IAA

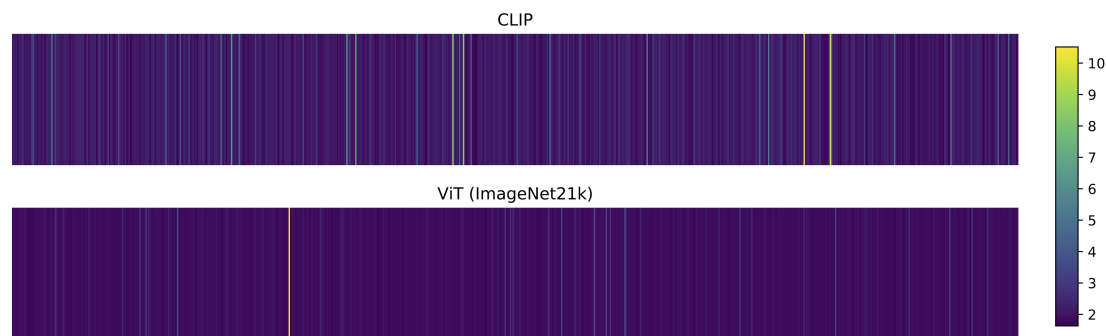


Figure 12.3.: Visualized weights of the LR trained on AVA. While the ImageNet-pretrained Vision Transformer (ViT) uses mainly one feature in the estimation of the target score, CLIP’s features are more broadly used. Together with the fact that the LR on CLIP features outperforms ImageNet features, this indicates that CLIP’s image features encode more useful and broader information for the IAA task.

improve the results while still only requiring few parameters to optimize.

While the prompting and Linear Probing approaches show good performance without any training or with only few trainable parameters, they are not as effective as finetuning the whole model. The results in the row “Finetuned CLIP” show that the finetuned CLIP model performs better than our other methods. One thing to notice is that it performs better than the ImageNet pretrained ViT baseline, which gives evidence that CLIP is a better base model for the IAA task than an ImageNet pretrained model. Most notably, the CLIP model achieves the highest Pearson and Spearman correlations for the models trained and evaluated on the same dataset split as ours. It also achieves the best Accuracy when only compared to methods that are designed to output continuous scores. Compared to the previous experiment, both models improve on their Linear Probing counterpart. However, Linear Probing CLIP achieves better performance on all metrics than the finetuned ImageNet model while optimizing a fraction of the parameters.

In conclusion, while our methods implementing the **Feature Extraction using Pretrained Multimodal Models** principle do not exceed full finetuning in terms of performance, they are much more efficient in terms of computational resources. They require no to only few trainable parameters and can be applied to the pretrained CLIP model, that might be available in applications for other tasks, such as content classification.

12.4. Analysis

To better understand the difference between CLIP’s and ViT’s features, we visualize the 768 weights from the trained Linear Regression in Figure 12.3. For this, we scale the LR weights by the standard deviation of the input features to alleviate the impact of differences in the features’ value ranges. The visualization shows that the LR mainly focuses on only one dimension of the ViT features but on a broader set of features from CLIP. Thus, the features from CLIP are a better starting point to predict the aesthetic

Table 12.2.: Linear Probing results for similar ResNet-50 [86] models. CLIP models trained on similarly sized datasets as ImageNet21k still outperform the baseline model. The performance of CLIP on AVA improves with larger dataset sizes.

Method	Num. Training Images	Accuracy \uparrow	Spearman \uparrow	Pearson \uparrow
ResNet-50 (ImageNet21k)	approx. 14 million	0.758	0.554	0.567
ResNet-50 OpenCLIP (Conceptual 12M)	approx. 12 million	0.777	0.623	0.631
ResNet-50 OpenCLIP (YFCC-15M)	approx. 15 million	0.793	0.662	0.673
ResNet-50 CLIP (OpenAI)	approx. 400 million	0.793	0.674	0.682

score of images, since the LR model can combine more useful features. Overall, we can summarize that CLIP as a static feature extractor is better suited for the IAA task than a classification based feature extractor.

An alternative hypothesis would be that the improvements stem from the amount of data CLIP has been trained on (400 million image-text pairs) compared to the ImageNet-ViT (approximately 14 million images). To rule this out, we compare a ResNet-50 [86] trained on ImageNet21k to ResNet-50 image encoders from OpenCLIP [107] trained on the Conceptual 12M [35] and the YFCC-15M (a subset of the YFCC-100M dataset [266]) datasets. These datasets comprise of approx. 12 and 15 million image-text pairs, respectively, so are comparable in size to the ImageNet21k dataset. The results for the Linear Probing of these models in Table 12.2 show that both CLIP models show better performance than the ImageNet21k model. This provides more evidence that the dataset size is not the reason for CLIP’s performance improvement, but the quality of the training data and its training task.

12.5. Conclusion

In our prompting experiments, context-aware ensemble prompts work best. Since we do not know the correct content of the image, we have tried all class names of the thousand ImageNet classes. While this seems to work quite well, it is not clear if the chosen class names are the best choice for this task. Since each ensemble is represented by only one feature vector, more content descriptions could be added without larger computational requirements during inference. Future work might evaluate more and different content descriptions, e.g., all class names of the ImageNet21k dataset or using Knowledge Graphs such as ConceptNet [254]. Automatically generating image captions for the image content and using these for more targeted prompting is also an interesting, though more computationally expensive, research direction [184].

Our experiments have shown that the performance improves with the number of trained parameters, but using prompt ensembles alone already shows acceptable performance for many real-world tasks. Given that CLIP shows good performance on many different datasets without explicit training [214], we see high potential for CLIP in image databases, such as personal photo collections. It may be possible to store CLIP features for each image and use them to perform tasks like searching for contents using text [16], finding similar images, or creating image descriptions [184]. Our work shows that finding aesthetically

12. Using CLIP for IAA

pleasing pictures is another task that can be done with a pretrained CLIP model.

We have investigated the suitability of CLIP, a model jointly trained with image-text pairs, as a static feature extractor for the IAA task. We have tested multiple methods with different levels of computational complexity, namely prompting in different variations and Linear Probing. All experiments have led to the conclusion that CLIP extracts features from images that are related to human’s image aesthetic perception due to its training on images and their corresponding human-generated descriptions. In our experiments, CLIP features always outperform features extracted from an ImageNet classification model, which is the base model in most IAA papers. We can thus use this pretrained multimodal model as a preprocessing and feature extraction step for a variety of tasks, improving the range of tasks that can be done without specifically trained NNs.

13. Zero-shot Deep Metric Learning (DML) with CLIP

In this section, we again use CLIP as a static feature extractor for images and texts. Here, we implement the [Feature Extraction using Pretrained Multimodal Models](#) principle for the DML task, which is a representation task. Two inputs, here images, should be mapped to similar vectors when deemed similar by a given similarity notion. We introduce a way to simply describe this similarity notion with text and use it to train a model that maps images to vectors that reflect the similarity notion. This way, we can incorporate additional domain knowledge about the similarity notion expressed as text into the model. We mainly follow our work in [135].

As already discussed, DML is the task of training Neural Networks (NNs) that map input items to a low-dimensional manifold such that similar items are represented by vectors close to each other [119, 193]. In the usual DML setting, training examples are needed that let the model learn which image properties make an image pair (dis)similar. For example, in the Cars196 dataset’s setting [139], two images are deemed similar if they show the same car model. Factors such as the car color, its orientation, or the image’s environment should be suppressed by the embedding process. Other datasets have different image properties to define when two images are similar. We call this high-level interpretation of when two inputs are deemed similar a **similarity notion**. For Cars196, the similarity notion is “Two car images are similar if they show the same car model”. During testing, the ability of the NN to generalize this learned similarity notion to new unseen classes (e.g., new car models) is measured.

Often, people have different similarity notions depending on the task at hand or personal preference. It is thus desirable to be able to quickly adapt to changing similarity notions. However, large labeled training datasets are needed to train a model for a new similarity notion, which is time-consuming and tedious for users to create. We thus aim for a zero-shot setting, where no training images and labels are needed. We argue that users often can express the desired similarity notion using words, e.g., “Two car images are similar if both cars have the same color”. This is especially the case when there are categorical aspects with names that sort the images into disjoint classes. More specifically, users can list a set of distinct aspects describing the similarity notion, e.g., “a red car”, “a green car”, ... The use of language simplifies the process of expressing custom similarity notions, which alleviates the problem of collecting new labeled datasets.

First, we introduce a new task we work on that we call **Language-Guided Zero-Shot Deep Metric Learning (LanZ-DML)**: Given a set of images \mathcal{I} and a desired similarity notion \mathcal{S} that is described using text $\mathcal{T}_{\mathcal{S}}$. Train a DML model using only the text input $\mathcal{T}_{\mathcal{S}}$ such that the resulting model can embed images $\mathcal{I} \rightarrow \mathbb{R}^r$ to r -dimensional embedding

13. Zero-shot DML with CLIP

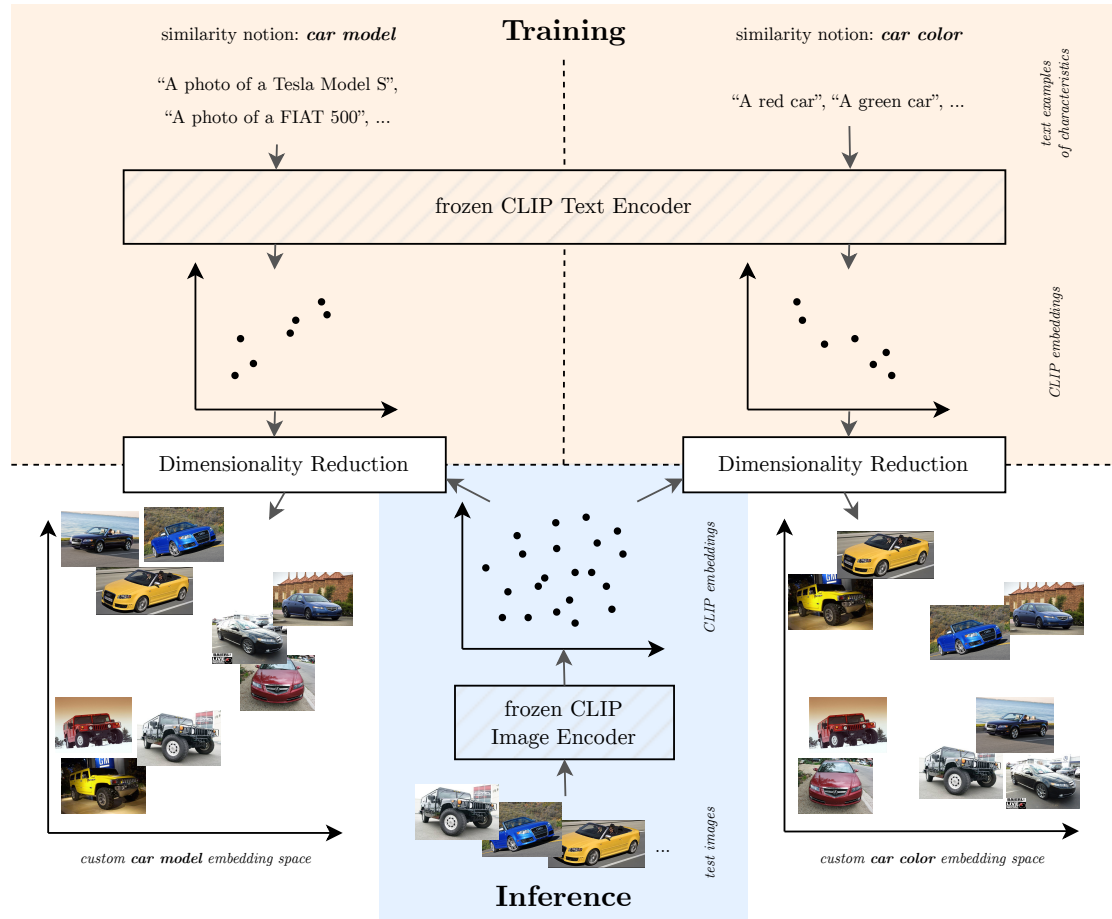


Figure 13.1.: During InDiReCT’s training (top half), different aspects of the desired similarity notion (e.g., car model or color) are collected in form of text prompts. CLIP’s frozen text encoder embeds them and a dimensionality reduction method is learned to extract the dimensions that encode the similarity notion’s aspects. During inference (bottom half), the trained dimensionality reduction is applied to CLIP encoded images to obtain custom image embeddings representing the desired similarity notion. Hatched components depict frozen elements of the model.

vectors, making image embeddings more similar if they are deemed similar regarding the similarity notion. For optimization, no training images or labels are allowed (thus zero-shot).

Second, we propose InDiReCT (**I**mage representations using **D**imensionality **R**eduction on **CL**IP embedded **T**exts), a model for LanZ-DML that uses a list of text prompts as input and learns a transformation that maps images to a vector space that reflects the desired similarity notion. It utilizes the CLIP [214] model as a static general purpose feature extractor for images and texts. We assume that CLIP embeddings for images and texts encode similar concepts in similar directions of the embedding space and that image descriptions can focus on certain properties. For example, the text description “a photo of a red car” focuses on the car color and not on other features, such as the car’s position, orientation, or environmental factors.

Figure 13.1 gives an overview of InDiReCT. During training, CLIP’s *fixed* text encoder represents different characteristics of a desired similarity notion \mathcal{S} as 512-dimensional vectors, e.g., “a red car”, “a white car”, and so on to encode the car color. We then extract the largest variations of these vectors in the embedding space by applying a dimensionality reduction method to these text representations, focusing on the changing aspects and abstracting away other non-related dimensions. Learning the dimensionality reduction is fast and often, only a few dozen text prompts are needed. Also, no training images or labels are used, only text prompts.

During inference, images are fed through CLIP’s *fixed* image encoder and the trained dimensionality reduction. Assuming that CLIP’s embeddings encode similar concepts in similar embedding space directions for both modalities, the resulting image representations are focused on the same dimensions as described by the text prompts. Finally, lower-dimensional vectors can be used to find images similar to an image w.r.t. the desired similarity notion.

13.1. Methodology

We now introduce InDiReCT, our method for Language-Guided Zero-Shot Deep Metric Learning on images. It makes use of a fixed CLIP [214] as a general-purpose feature extractor for images and texts. See Section 2.3.4 for an introduction to CLIP. Due to its training task, CLIP learns to extract broad image features that can be correlated with/expressed by language. Intuitively, we aim to learn a transformation that focuses on the most important features extracted by CLIP regarding the desired similarity notion. Figure 13.1 shows InDiReCT’s training and inference.

13.1.1. Training

In the training phase, n different text prompts are created that describe certain characteristics of the desired similarity notion. For example, if the target images show cars and we want to differentiate them by their color, we create a list of texts $\mathcal{T}_{\mathcal{S}}$ with $|\mathcal{T}_{\mathcal{S}}| = n$ such as “a red car”, “a blue car”, “a white car”, and so on. The text prompts should only vary in the notion that we want to differentiate (here, the color descriptions). Note that

13. Zero-shot DML with CLIP

the aspects in the training text prompts are chosen independently from inference data, since inference labels are not known during training and we want to generalize to new aspects of the similarity notion as well.

When feeding all texts through CLIP’s text encoder, the resulting r -dimensional row vectors¹ $\mathbf{t}_i \in \mathbb{R}^{1 \times r}$ for $i \in \{1, \dots, n\}$ vary in certain directions. This is introduced by the change in aspects of the desired similarity notion in the text prompts. Here, the variation of the vectors is only explained by the change of color names in the texts.

Due to CLIP encoding similar concepts to similar embedding dimensions, varying the same aspects in images and texts should result in embeddings that vary in similar directions. Our goal is to find these directions using the text embeddings and suppress all other directions in the image embedding space, which are influenced by undesired factors. Given the n text representations $\mathbf{t}_i \in \mathbb{R}^{1 \times r}$ for $i \in 1, \dots, n$, we thus aim to identify the dimensions that vary the most in order to learn a transformation that retains these directions while reducing the embedding to $r' < r$ dimensions (similar to dimensionality reduction techniques such as Principal Component Analysis (PCA) [296]).

For this, we transform the text representations \mathbf{t}_i using a matrix $\mathbf{U} \in \mathbb{R}^{r \times r'}$ and reconstruct them using \mathbf{U}^\top . We optimize \mathbf{U} with gradient descent to minimize the reconstruction loss L :

$$\mathbf{t}_i^{\text{norm}} = \frac{\mathbf{t}_i}{\|\mathbf{t}_i\|} \quad // \text{ normalize text embedding} \quad (13.1)$$

$$\mathbf{t}'_i = \frac{\mathbf{t}_i^{\text{norm}} \mathbf{U}}{\|\mathbf{t}_i^{\text{norm}} \mathbf{U}\|} \quad // \text{ transform and normalize} \quad (13.2)$$

$$\mathbf{t}_i^{\text{recon}} = \frac{\mathbf{t}'_i \mathbf{U}^\top}{\|\mathbf{t}'_i \mathbf{U}^\top\|} \quad // \text{ backtransform and normalize} \quad (13.3)$$

$$L = \frac{1}{n} \sum_{i=1}^n \arccos(\mathbf{t}_i^{\text{norm}} \mathbf{t}_i^{\text{recon} \top}) \quad // \text{ minimize distance.} \quad (13.4)$$

CLIP’s use of cosine similarity as a similarity measure for embeddings disregards the length of all vectors, so we map input vectors \mathbf{t}_i and their reconstructions to a unit hypersphere (Equations (13.1) and (13.3)). Then we minimize the mean spherical distance (Equation (13.4)) between the input and reconstructed vectors [123]. It is the geodesic distance between the vectors along the surface of the hypersphere, scaling linearly with the vector angle. The training objective effectively minimizes the angles between the inputs and reconstructions.

In addition, the lower-dimensional embedding projections \mathbf{t}'_i are also mapped to a unit hypersphere (Equation (13.2)). This ensures that the reconstruction only uses the angles between the r' -dimensional vectors, keeping cosine similarity as a similarity measure in the lower-dimensional space, while preserving the varying directions of the text embeddings.

Since only up to a few hundred text prompts are used and only the matrix \mathbf{U} must be optimized, L typically converges really fast. The whole optimization process usually

¹ $r = 512$ for CLIP’s base model, but it is not limited to that number

finishes in less than a minute on a common laptop’s CPU, allowing InDiReCT to adapt to new similarity notions fast.

13.1.2. Inference

Given query and reference images, we feed them through CLIP’s fixed image encoder and apply the learned transformation to map these embeddings $\mathbf{v}_i \in \mathbb{R}^{1 \times r}$ ($i \in \{1, \dots, m\}$) to r' dimensions on a unit hypersphere:

$$\mathbf{v}_i^{\text{norm}} = \frac{\mathbf{v}_i}{\|\mathbf{v}_i\|} \quad // \text{ normalize image embedding} \quad (13.5)$$

$$\mathbf{v}'_i = \frac{\mathbf{v}_i^{\text{norm}} \mathbf{U}}{\|\mathbf{v}_i^{\text{norm}} \mathbf{U}\|} \quad // \text{ transform and normalize.} \quad (13.6)$$

These vectors can be compared using the cosine/dot product similarity to find similar images w.r.t. the desired similarity notion. Since the transformation learns to suppress dimensions that do not vary in the text prompts, these dimensions are also suppressed for images, e.g., a car *model* dimension in the CLIP embedding space is suppressed when training with the similarity notion “car color”.

13.2. Experiments

We now perform multiple experiments using InDiReCT and other baselines. Since we are in a zero-shot learning setting, we have no access to labeled training images. Hyperparameters cannot be tuned on a validation dataset, since labeled data is not allowed. We thus define prompts and set hyperparameters based on commonly used values or educated guesses. This resembles the real world scenario, where users do not have any training data at hand to verify and optimize their input to the system.

We implement InDiReCT using PyTorch [206] and sample the initial values of \mathbf{U} from $\mathcal{N}(0, 0.1)$. We then optimize \mathbf{U} using Adam [127] with a learning rate of 0.01 until it does not improve the loss L (Equation (13.4)) for 100 consecutive iterations. We reduce CLIP’s vectors to 128 dimensions, which is a common embedding size for DML models [193].

After training models, we compute image embeddings for each dataset and similarity notion, following the standard evaluation setting of DML by measuring the retrieval performance for these embeddings using the Mean Average Precision at R (MAP@R) and Precision at 1 (Prec@1) (introduced in Section 2.5.3). Results for other evaluation metrics can be found in Appendix B, but they show the same tendencies.

13.2.1. Datasets and Similarity Notions

We experiment with five datasets and overall thirteen similarity notions, which are listed in Table 13.1. For each dataset, we define one to four similarity notions, e.g., the “Car Model” similarity notion of the Synthetic Cars [134] and Cars196 [139] datasets can be expressed as “Two car images are similar if they show the same car model”. Other notions

Table 13.1.: Details on the datasets and similarity notions used for our experiments.

Dataset	Similarity Notion	Class Count	Prompt Template	Aspects (Count)
Synthetic Cars [134]	Car Model	6	“a photo of a [car model]”	Volvo S60, BMW X5 M, ... (569)
	Car Color	18	“a [color name] car”	orange, black, ... (18)
	Background Color	18	“a car in front of a [color] background”	orange, black, ... (18)
Cars196 [139]	Car Model	98	“a photo of a [car model]”	Volvo S60, BMW X5 M, ... (569)
	Manufacturer	35	“a photo of a car produced by [manufacturer]”	Tesla, BMW, ... (46)
	Car Type	7	“a photo of a [car type]”	convertible, SUV, ... (7)
CUB200 [275]	Bird Species	100	“a photo of a [bird species]”	Black footed Albatross, Rusty Blackbird, ... (100)
DeepFashion [162]	Clothing Category	50	“a photo of a person wearing a [clothing category]”	anorak, turtleneck, ... (50)
	Texture	7	“a photo of a person wearing clothes with a [texture type] texture”	floral, striped, ... (7)
	Fabric	6	“a photo of a person wearing clothes made out of [fabric type]”	cotton, leather, ... (6)
	Fit	3	“a photo of a person wearing clothes with a [fit type] fit”	tight, loose, conventional (3)
Movie Posters [41]	Genre	25	“a poster of a [genre] movie”	Comedy, Action, ... (25)
	Production Country	69	“a poster of a movie produced in [country]”	USA, India, ... (69)

can be formulated accordingly. Given a similarity notion, the datasets are split into different numbers of test classes (shown in the “Class Count” column), e.g., we use the 98 car models from Cars196’s test dataset. We create multiple text prompts for each similarity notion by collecting possible aspects and inserting them into a prompt template (listed in the corresponding columns). The varying aspects are collected from different sources, such as an online car dealer website (“Car Model” and “Manufacturer”), the CSS2.1 color names (“Car Color” and “Background Color”), or the training dataset labels (e.g., “Bird Species”). This promotes text prompts being possibly different from the test class labels, ensuring a realistic DML scenario, where train and test classes are commonly disjoint.

13.2.2. Baselines

InDiReCT is the first method for Language-Guided Zero-Shot Deep Metric Learning, i.e., it can efficiently generate specialized embedding spaces for images based on the desired similarity notions. We visualize the embeddings produced by InDiReCT for multiple similarity notions of the Cars196 dataset using TriMap [7] in Figure 13.2. In addition to the “Car Model”, “Car Manufacturer”, and “Car Type” similarity notions, we also add the “Car Color” similarity notion, which is not present in the original dataset’s metadata. For visualization purposes only, we rudimentary label each image with one of eight colors (‘black’, ‘blue’, ‘white’, ‘yellow’, ‘silver’, ‘red’, ‘mixed’, ‘other’). Note that since InDiReCT does not need labeled images, this process was only necessary for this visualization of the embedding space. The visualizations still show that cars with the same properties are clustered relatively well, even though InDiReCT does not use any training images but only text prompts. Given this, it is not fair to compare InDiReCT to fully supervised

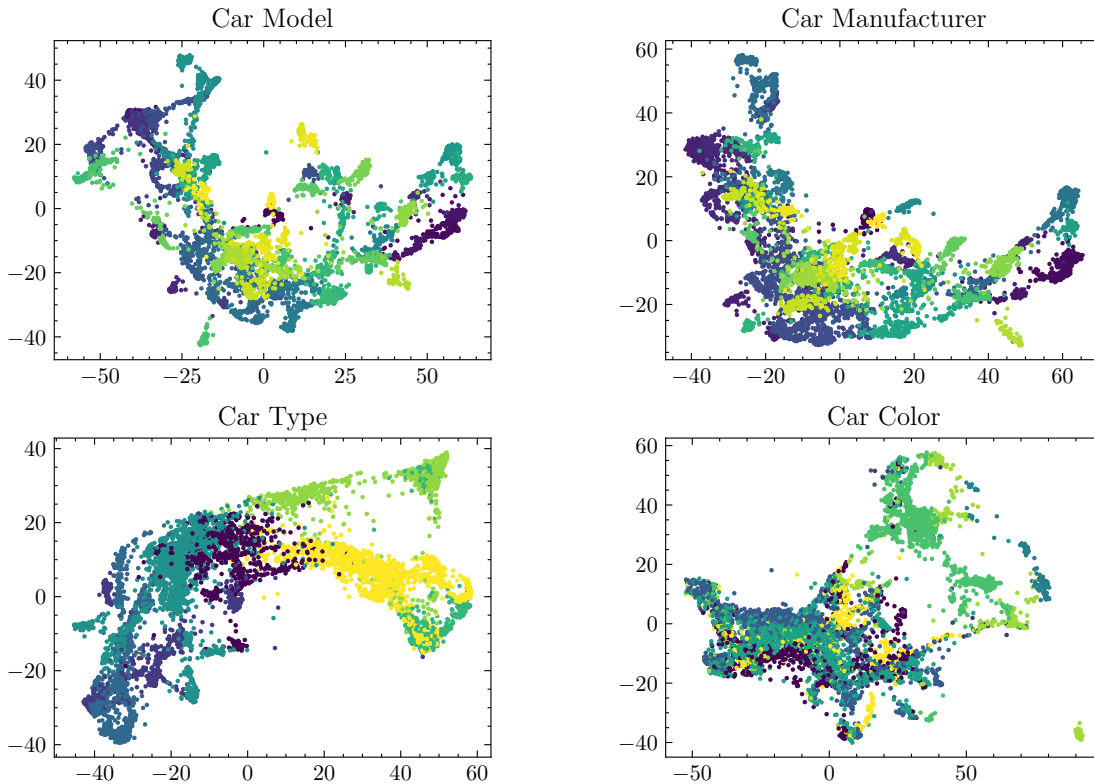


Figure 13.2.: TriMap visualizations of InDiReCT embedding spaces for multiple similarity notions of the Cars196 dataset.

baselines. However, we still contrast some of InDiReCT’s results with fully supervised models and an Oracle baseline. We use the following methods in our experiments to get a sense of how well InDiReCT performs.

Random Baseline For this baseline, we sample r' -dimensional embedding vectors for each image uniformly from the unit hypersphere [67]. To get such vectors, we sample all dimensions independently from a normal distribution $\mathcal{N}(0, 1)$ with mean 0 and standard deviation 1. Note that the standard deviation can be any number as long as it is the same for all dimensions. Afterwards, we normalize the resulting vector to a length of 1. This baseline indicates the performance lower bound for all methods.

CLIP [214] This baseline feeds all images through CLIP’s image encoder and uses the unmodified r -dimensional vectors as embeddings ($r = 512$). Due to the broad set of features CLIP extracts, its performance should already be quite good. However, since it does not focus on specific dimensions, InDiReCT is assumed to perform better while having fewer dimensions. Even more so, CLIP cannot adapt its embeddings based on the desired similarity notion, i.e., it always yields the same embeddings for an image. This

13. Zero-shot DML with CLIP

limitation holds for all embedding methods that do not use additional data regarding the desired similarity notion.

Random Transformation InDiReCT optimizes a transformation that is applied to CLIP’s image embeddings to achieve an embedding specialized towards a similarity notion expressed by text. We evaluate how well the learning procedure of InDiReCT improves the performance by leaving \mathbf{U} as initialized for testing, i.e., sampled from $\mathcal{N}(0, 0.1)$. We hypothesize that this baseline should, on average, be worse than both InDiReCT and the CLIP baseline.

Principal Component Analysis [296] PCA is a popular dimensionality reduction technique which finds orthogonal directions that explain the largest variation in the data. We test it as a possible alternative to our proposed method. In contrast to our method, PCA solves for principal components analytically, requiring r' to be strictly smaller than the number of input data points [207]. This is not satisfied for almost all scenarios in our experiments, since we only use a few text prompts while wanting to reduce CLIP’s embeddings to a target size of 128 dimensions. We thus can apply PCA only on the datasets that we collect more than 128 text prompts for, i.e., the “Car Model” similarity notion for the Synthetic Cars and Cars196 datasets.

Linear Autoencoder The LAE is an alternative to PCA that provably spans the same subspace while being able to be trained using gradient descent [209]. Formally, we optimize the weight matrices $\mathbf{W}_1 \in \mathbb{R}^{r \times r'}$, $\mathbf{W}_2 \in \mathbb{R}^{r' \times r}$ and bias vectors $\mathbf{b}_1 \in \mathbb{R}^{1 \times r'}$, $\mathbf{b}_2 \in \mathbb{R}^{1 \times r}$ with Adam (learning rate 0.01 and early stopping after 100 iterations) to minimize the loss function

$$L_{LAE} = \sum_{i=1}^n \sum_j ((\mathbf{t}_i^{\text{norm}})_j - (\mathbf{W}_2(\mathbf{W}_1 \mathbf{t}_i^{\text{norm}} + \mathbf{b}_1) + \mathbf{b}_2)_j)^2. \quad (13.7)$$

Image vectors are then transformed with

$$\mathbf{v}'_i = \mathbf{W}_1 \mathbf{v}_i^{\text{norm}} + \mathbf{b}_1. \quad (13.8)$$

Nonlinear Autoencoder (AE) While PCA and LAE are linear models, we also test a more powerful nonlinear Autoencoder, which consists of a two-layer encoder and decoder with 512 hidden units and leaky ReLU activation functions [169]. We use the same loss function and hyperparameters as for LAE, but add a weight decay of 10^{-2} to alleviate overfitting on the few text prompts.

Oracle InDiReCT uses only text prompts to optimize the transformation matrix \mathbf{U} that maps CLIP embeddings to a more specialized, lower-dimensional unit hypersphere. To estimate how well InDiReCT could theoretically perform, we employ an Oracle that optimizes \mathbf{U} directly on *test images and their labels*. For this, we use the common

DML loss function Normalized Softmax Loss [312]. We first compute unit-length image embeddings $\mathbf{v}'_i = \frac{\mathbf{v}_i^{\text{norm}} \mathbf{U}}{\|\mathbf{v}_i^{\text{norm}} \mathbf{U}\|}$ with $\mathbf{v}_i^{\text{norm}} = \frac{\mathbf{v}_i}{\|\mathbf{v}_i\|}$ as in Equations (13.5) and (13.6) and then optimize the transformation matrix \mathbf{U} to minimize the loss function

$$L_{Oracle} = -\frac{1}{m} \sum_{i=1}^m \log \left(\frac{\exp(\mathbf{v}'_i \mathbf{c}_{l_i}^\top)}{\sum_j \exp(\mathbf{v}'_i \mathbf{c}_j^\top)} \right), \quad (13.9)$$

where m is the number of test images and $\mathbf{c}_{l_i} \in \mathbb{R}^{1 \times r'}$ with $\|\mathbf{c}_{l_i}\| = 1$ is the prototype vector of the class for the label of the i th image l_i , which is optimized jointly with \mathbf{U} using Adam (learning rate 0.01, early stopping with patience 100).

Note that in our proposed Language-Guided Zero-Shot Deep Metric Learning, neither images nor their labels are available for training. We use this baseline method in order to provide a *very optimistic* estimate of what performance InDiReCT could achieve given perfect information. The Normalized Softmax Loss is a classification-based training objective, so image embeddings are processed independently. Thus, the loss does not optimize for the best nearest neighbor performance, i.e., Precision@1. To compare the Oracle baseline to other models, we thus primarily use MAP@R.

Low (high) Oracle performance can be used to identify similarity notions that cannot (can) be reliably represented using InDiReCT since they are not captured (are captured) in the CLIP embeddings. If InDiReCT performs substantially worse than the Oracle, it means that the text prompts were not capable of capturing the desired similarity notion.

13.3. Results

We report the mean and standard deviation of the evaluation metrics over five runs in Table 13.2. The CLIP baseline typically achieves substantially better results than the random baseline. Since the embeddings stay the same in each run, its performance does have a standard deviation of zero and is omitted for brevity. Despite the fact that the CLIP baseline uses four times larger embedding vectors, InDiReCT almost always performs better than CLIP and achieves the best performance in most datasets and similarity notions. Depending on the dataset and similarity notion, InDiReCT can improve CLIP’s MAP@R score by up to 14 percentage points (see Table 13.2). Switching the learned matrix to a random transformation matrix in InDiReCT usually performs worse than CLIP. As described in Section 13.2.2, PCA is only applicable to two datasets and similarity notions. There, InDiReCT and PCA perform similarly. Training a LAE on the text embeddings usually improves the CLIP baseline, but does not achieve better performance than InDiReCT. Applying a more complex nonlinear Autoencoder performs oftentimes worse than the CLIP baseline and also shows substantially larger standard deviations, which might be due to the model not handling the few datapoints well. These results show that choosing a suitable dimensionality reduction technique can improve performance and opens up new research directions. In general, InDiReCT learns a useful embedding function by using text prompts that describe different aspects of the desired similarity notion.

13. Zero-shot DML with CLIP

Table 13.2.: Results for our experiments. All values are given in percent, best in bold. The Oracle baseline is given for reference.

		Random	CLIP (512-dim.)	InDiReCT	Rand. trans.	PCA	LAE	AE	Oracle	
Synthetic Cars	Car Model	MAP@R ↑	3.3 ± 0.1	43.5	57.4 ± 0.2	39.1 ± 1.6	56.2 ± 0.1	52.5 ± 0.5	39.5 ± 4.4	100 ± 0.0
		Prec@1 ↑	17.5 ± 0.9	95.4	96.4 ± 0.0	93.4 ± 0.5	96.6 ± 0.1	95.9 ± 0.5	88.7 ± 3.6	100 ± 0.0
	Car Color	MAP@R ↑	5.0 ± 0.1	6.2	9.1 ± 0.1	6.1 ± 0.1	—	7.3 ± 0.2	8.6 ± 0.4	57.9 ± 0.9
		Prec@1 ↑	17.5 ± 0.8	27.6	31.4 ± 0.5	26.3 ± 1.3	—	29.4 ± 0.9	30.2 ± 1.3	79.3 ± 0.8
	Background Color	MAP@R ↑	5.4 ± 0.0	6.2	7.1 ± 0.0	6.1 ± 0.2	—	6.3 ± 0.2	6.1 ± 0.2	74.0 ± 0.9
		Prec@1 ↑	19.4 ± 1.1	27.0	28.3 ± 0.3	26.6 ± 1.1	—	28.3 ± 0.7	21.6 ± 1.3	88.0 ± 0.4
Cars196	Car Model	MAP@R ↑	0.1 ± 0.0	23.5	37.4 ± 0.0	19.2 ± 0.3	37.5 ± 0.1	33.2 ± 0.2	20.0 ± 5.8	41.8 ± 0.0
		Prec@1 ↑	1.1 ± 0.1	78.0	84.4 ± 0.1	72.9 ± 0.5	84.2 ± 0.1	82.4 ± 0.2	63.8 ± 8.1	76.6 ± 0.1
	Manufacturer	MAP@R ↑	0.5 ± 0.0	24.4	33.6 ± 0.1	21.2 ± 0.4	—	24.2 ± 0.4	18.0 ± 2.2	51.4 ± 0.0
		Prec@1 ↑	5.4 ± 0.3	89.0	90.5 ± 0.1	84.7 ± 0.8	—	85.5 ± 0.3	63.1 ± 3.9	84.0 ± 0.1
	Car Type	MAP@R ↑	3.5 ± 0.0	25.1	36.1 ± 0.3	22.1 ± 0.8	—	27.7 ± 0.6	24.4 ± 1.6	73.8 ± 0.0
		Prec@1 ↑	17.3 ± 0.4	91.1	90.7 ± 0.2	88.3 ± 0.5	—	89.1 ± 0.4	63.2 ± 3.1	89.1 ± 0.0
CUB200	Bird Species	MAP@R ↑	0.1 ± 0.0	18.0	26.5 ± 0.0	15.2 ± 0.3	—	18.8 ± 0.2	15.1 ± 1.9	34.1 ± 0.0
		Prec@1 ↑	1.2 ± 0.1	58.2	65.3 ± 0.1	52.6 ± 0.3	—	58.1 ± 0.5	44.4 ± 3.6	65.3 ± 0.2
DeepFashion	Clothing Category	MAP@R ↑	2.3 ± 0.0	12.5	18.7 ± 0.1	11.3 ± 0.4	—	13.3 ± 0.3	16.9 ± 1.8	32.2 ± 0.1
		Prec@1 ↑	11.1 ± 0.4	45.2	50.9 ± 0.2	43.0 ± 0.6	—	45.5 ± 0.5	44.5 ± 2.4	55.8 ± 0.6
	Texture	MAP@R ↑	11.8 ± 0.0	18.7	33.0 ± 0.4	11.2 ± 0.4	—	22.2 ± 0.5	16.3 ± 0.7	66.1 ± 0.1
		Prec@1 ↑	29.6 ± 0.7	60.2	66.8 ± 0.3	43.3 ± 0.5	—	61.2 ± 0.7	43.8 ± 1.7	80.6 ± 0.3
	Fabric	MAP@R ↑	32.4 ± 0.0	34.0	37.7 ± 0.2	10.8 ± 0.3	—	35.6 ± 0.3	17.2 ± 0.6	64.2 ± 0.3
		Prec@1 ↑	49.4 ± 0.6	64.5	66.1 ± 0.6	42.6 ± 0.7	—	65.1 ± 0.6	44.7 ± 1.9	77.8 ± 0.4
Fit	MAP@R ↑	51.8 ± 0.0	53.3	53.9 ± 0.4	11.1 ± 1.0	—	53.4 ± 0.3	16.1 ± 1.8	82.0 ± 0.1	
	Prec@1 ↑	66.6 ± 0.6	77.1	76.5 ± 0.4	43.1 ± 0.5	—	76.7 ± 0.7	42.9 ± 1.9	87.8 ± 0.6	
Movie Posters	Genre	MAP@R ↑	4.1 ± 0.0	11.4	14.9 ± 0.0	9.1 ± 0.3	—	8.4 ± 0.1	9.8 ± 2.4	19.6 ± 0.1
		Prec@1 ↑	17.5 ± 0.4	41.8	44.0 ± 0.2	38.1 ± 0.7	—	36.6 ± 0.4	33.3 ± 3.0	43.2 ± 0.7
	Production Country	MAP@R ↑	44.6 ± 0.0	49.3	51.3 ± 0.1	48.9 ± 0.4	—	47.7 ± 0.2	49.4 ± 0.7	58.1 ± 0.0
		Prec@1 ↑	59.2 ± 0.5	69.3	69.8 ± 0.3	67.9 ± 0.7	—	68.1 ± 0.3	64.9 ± 0.7	71.8 ± 0.3

The Oracle baseline is optimized directly on the image dataset and their labels. Despite all this, InDiReCT matches or exceeds the Prec@1 performance of the Oracle baseline for Cars196, CUB200, and the “Genre” similarity notion for the Movie Posters dataset. As discussed in Section 13.2.2, this might be due to the classification-based nature of the Normalized Softmax Loss. For MAP@R, the Oracle is the best model for all datasets and similarity notions.

Even though the comparison is not fair, we contrast InDiReCT’s performance with state of the art models from the literature that train on a large labeled training dataset regarding the desired similarity notion. Note that only Cars196’s “Car Model” and CUB200’s “Bird Species” similarity notions have been used in the literature in a DML setting, so we only compare to them. Jun et al. [114] achieve Prec@1 of 94.8 and 79.2 for Cars196 and CUB200 in a supervised setting, respectively [220], which outperform InDiReCT by ten to fourteen percentage points. However, the trained models output 1536-dimensional vectors, more than ten times the embedding dimensions we use in our experiments. For embeddings of dimensions 128, Jun et al. achieve 90.1 (Cars196) and 67.6 (CUB200) Prec@1, which is only approximately six and two percentage points better than InDiReCT. These results show that despite not using any training images, InDiReCT can show strong performance even compared to fully supervised methods.

13.4. Analysis

To better understand InDiReCT, we visualize the image regions focused by InDiReCT and investigate the influence of different hyperparameters.

13.4.1. What does InDiReCT attend to in the input?

We want to visualize the image regions that are used by InDiReCT to output a certain embedding. Due to the positive experimental results, we assume that, for a given similarity notion, InDiReCT attends to subjectively more useful regions than CLIP. We thus compute attribution maps using the method we described in Chapter 9 and subtract InDiReCT’s attribution maps from CLIP’s attribution maps to qualitatively showcase the difference between both methods.

We choose Cars196 and its similarity notions and hypothesize that InDiReCT pays more attention to regions that represent the desired similarity notion than CLIP. In order to increase the chance of obtaining visible differences in the attribution maps, we reduce the number of embedding dimensions for InDiReCT to two, thus only extracting the most important features to embed the given images. Figure 13.3 shows five randomly selected example images. Yellow areas indicate image regions InDiReCT pays more attention to than CLIP, while CLIP focuses more on blue regions. Grey areas show similarly strong saliency.

Compared to CLIP, InDiReCT focuses more on the area of the car when using the “Car Model” similarity notion, which is useful for the task. Interestingly, for “Manufacturer”, InDiReCT mostly uses the front of the car, where the manufacturer’s logo is usually found. Additionally, the design of the radiator grill and headlights is often relatively unique to manufacturers. For the “Car Type” similarity notion, InDiReCT focuses more on the back of the car, as car types such as “convertible”, “van”, or “sedan” differ mainly in terms of trunk and roof design.

13.4.2. Do other embedding sizes perform differently?

While our experiments set the embedding size arbitrarily to 128, we now measure the performance on the Cars196 dataset with varying target embedding dimensions $r' \in \{2, 4, 8, \dots, 256, 512\}$. We plot the MAP@R mean and standard deviation over five runs for all methods and all similarity notions in Figure 13.4. CLIP with its fixed 512 dimensions is plotted as a reference line.

InDiReCT matches or exceeds CLIP’s performance when using at least 16 embedding dimensions and peaks at 64 dimensions for all three similarity notions. The learned transformation presumably selects, combines, and weights CLIP’s embedding dimensions such that InDiReCT even outperforms CLIP when no dimensionality reduction is performed at 512 dimensions.

13. Zero-shot DML with CLIP

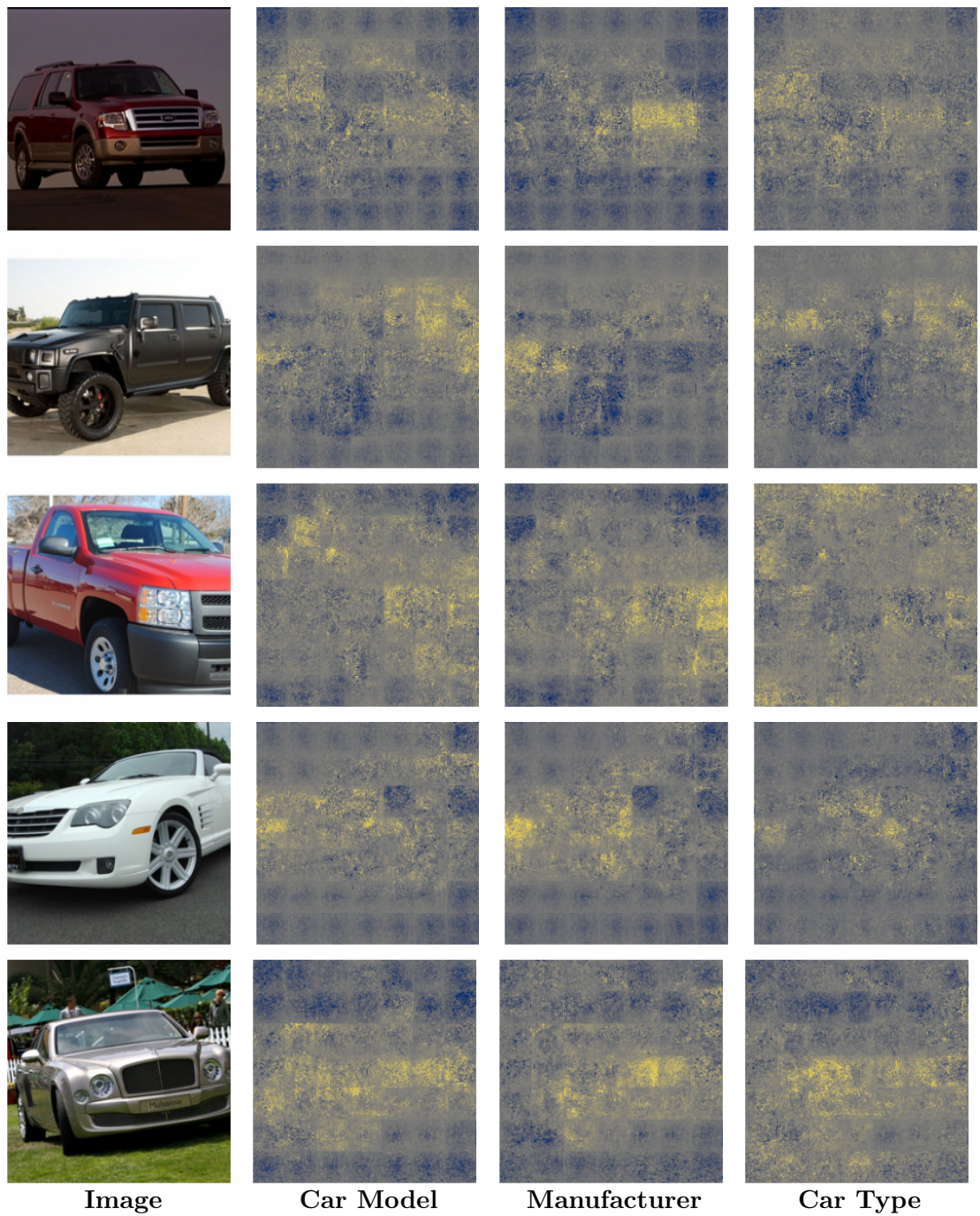


Figure 13.3.: Example images from the Cars196 dataset and the attribution map differences between each similarity notion and CLIP. InDiReCT focuses more on yellow regions, CLIP more on blue regions. The patch patterns in the images are due to the patch creation of CLIP’s Vision Transformer (ViT) [50].

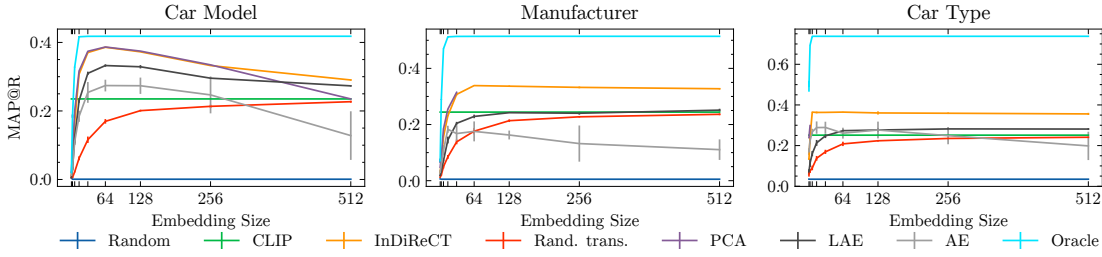


Figure 13.4.: On Cars196, InDiReCT outperforms other zero-shot models for embedding sizes 16 and up, while it peaks at 64 dimensions.

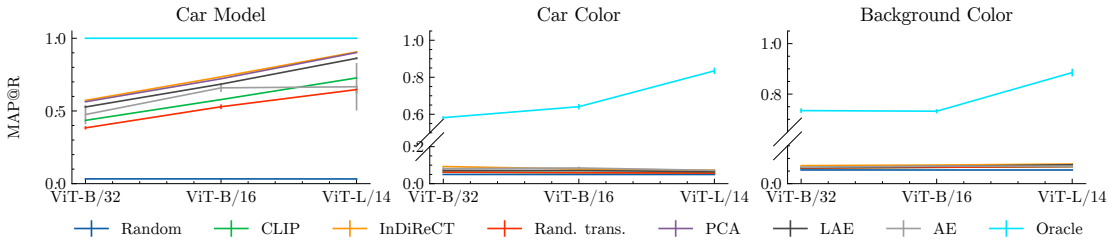


Figure 13.5.: Larger CLIP models improve performance for the “Car Model” but not for color similarity notions on Synthetic Cars.

13.4.3. Do larger CLIP models improve performance?

For our experiments, we use the CLIP model “ViT-B/32” [214], i.e., a Vision Transformer (ViT) [50] with 12 layers and input patches of size 32×32 pixels. We now test larger CLIP models as feature extractors in InDiReCT with CLIP’s “ViT-B/16” and “ViT-L/14” versions, which change the input patches to 16×16 and 14×14 pixels, respectively, while “ViT-L/14” also doubles the Transformer layers. Besides other parameters, “ViT-L/14” also increases CLIP’s outputs from 512- to 768-dimensional vectors.

We test all three ViT sizes to see if larger CLIP versions lead to better performance [214]. The “Synthetic Cars” dataset with its similarity notions is used, since the performance of InDiReCT is quite good for “Car Model”, but bad for “Car Color” and “Background Color”, compared to the Oracle baseline. With this analysis, we can investigate whether larger models can improve performance for these similarity notions. We use 128 embedding dimensions.

Figure 13.5 shows that the performance of the Oracle baseline increases with larger models, which means that the model extracts more useful features that could potentially be picked up by InDiReCT. For the “Car Model” similarity notion, this also translates to better performance of InDiReCT and CLIP in general. On the other two similarity notions, however, we cannot find any performance improvements. Since the Oracle baseline improves, we can conclude that the text prompts used to train InDiReCT lead to a focus on suboptimal features for these similarity notions. Other text prompt templates might increase performance.

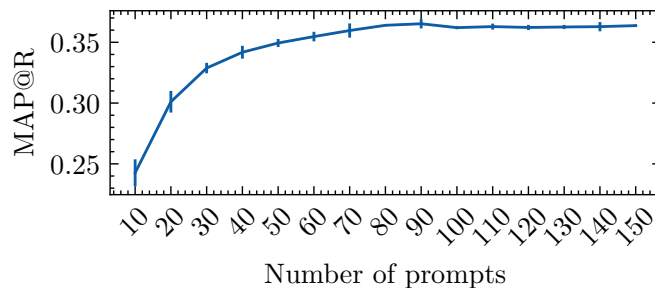


Figure 13.6.: Performance of InDiReCT for different number of training prompts. We sample different car model names for each run.

13.4.4. Do more text prompts improve performance?

Our final analysis takes a closer look at how the performance of InDiReCT changes if we use different numbers of prompts for our experiments. We use the Cars196 dataset and focus on the “Car Model” similarity notion. Originally, we use 569 different car model names from an online car dealer as a basis for the text prompts (“a photo of a [car model name]”). We now sample differently sized sets from these car model names and run our experiment five times with different samples. Figure 13.6 shows the means and standard deviations for sizes $\{10, 20, \dots, 150\}$. The performance increases with larger sample sizes and converges at around 90 prompts to the performance we observe in our main experiments. This behavior is expected, since the learned transformation is able to better capture the important dimensions in the text embeddings when more prompts are used. For fewer prompts, InDiReCT can almost perfectly reconstruct the text embeddings, thus is not forced to select the important dimensions. Figure 13.6 also shows that with larger prompt sets, the standard deviation of performance tends to decrease. Overall, we can observe that more text prompts should stabilize and improve performance for InDiReCT.

13.5. Conclusion

Using natural language, the proposed LanZ-DML setting offers a simple interface for adapting item retrieval systems to the desired similarity notion. This adaption is not achievable using raw CLIP embeddings or other self-/unsupervised methods. For InDiReCT, it is not necessary to collect and annotate example images, which is time-consuming and tedious. Expressing the desired similarity notion using text prompts is certainly simpler, but limits its application to similarity notions with categorical aspects. However, this is a limitation that also holds for popular proxy-based DML loss functions such as Normalized Softmax Loss [312] or ProxyNCA [190], i.e., loss functions that use class prototype vectors. It should also be noted that the quality of text prompts might vary significantly. In our experiments, we comply with the zero-shot setting by choosing plausible prompt templates without validating them on the data. Overall, we achieve good performance across datasets and similarity notions. However, as already shown for

prompt engineering [214], there might be prompts that work substantially better. Often, exploiting the peculiarities of the dataset CLIP has been trained on helps. For example, instead of using single words as text prompts, short sentences usually work better [214]. Therefore, it is recommended to test different text prompts when applied in real-world scenarios. Also, tuning the number of embedding dimensions is not straightforward without validation data, leading to suboptimal performance when using 128- instead of 64-dimensional vectors for the Cars196 dataset, as shown in our analysis.

Since we use CLIP as a fixed feature extractor, we need to rely on the usefulness of its embeddings. If CLIP does not extract properties from images and texts related to a desired similarity notion, InDiReCT cannot show its full potential. We have shown that InDiReCT mostly outperforms CLIP, so the text prompts help to focus on the desired similarity notion. Given the Oracle results, however, some datasets and similarity notions (e.g., Synthetic Cars’ color notions) could potentially work better. In some cases, larger CLIP models can improve the performance as shown in our analysis.

Since we use pretrained CLIP embeddings and only a handful of text prompts, training the dimensionality reduction is fast. It also allows us to precompute CLIP embeddings for a whole image database and adaptively transform them with a trained dimensionality reduction. The disadvantage of this is that, for each search, the transformation matrix must be applied to all vectors in the image collection. Potentially, existing vector search databases [279, 113] can efficiently incorporate the transformation to retrieve relevant images.

We have introduced Language-Guided Zero-Shot Deep Metric Learning (LanZ-DML), a setting where no training data and labels, but only texts are allowed to guide a Deep Metric Learning model for a given similarity notion. Our proposed model InDiReCT is based on fixed CLIP embeddings of text prompts describing the varying aspects of a given similarity notion. We have shown that InDiReCT outperforms strong baselines and approaches fully supervised methods. Our analyses show that InDiReCT focuses on image regions that are subjectively important for the desired similarity notion. We have also investigated the influence of different hyperparameters on the model performance.

Due to its simple design and fast training, InDiReCT can be useful for users to customize the similarity notion of item retrieval systems. The need to define multiple prompts based on the changing aspects of a similarity notion could be facilitated, e.g., by directly learning the transformation from sentences such as “Two car images are similar if both cars are the same model”. Automatic selection of hyperparameters and developing methods for LanZ-DML on other modalities, e.g., audio or texts, are also interesting research directions.

14. Improving Classification Models using Class Similarities in the Loss Function

This section mainly follows our work in [132] which proposes to include domain knowledge into the commonly used classification **Loss Function** Categorical Cross Entropy (CCE). CCE tries to maximize the assigned target class probability and punishes every misclassification in the same way, independent of other information about the predicted class. Often, however, classes have a special order or are similar to each other, such as different flowers in image classification. Including class similarities using the inherent class structure (e.g., class order), class properties (e.g., class names) or external information about the classes (e.g., knowledge graphs) in the training procedure would allow the classifier to make less severe mistakes as it learns to predict similar classes.

In this implementation, we modify CCE and propose Similarity Based Loss (SimLoss) as a way to explicitly introduce domain knowledge into the training process, as visualized in Figure 14.1. For this, we augment CCE with a matrix containing class similarities and propose two techniques in order to prepare such matrices that exploit certain class relations: class order and general class similarities. We show on two tasks, Age Estimation (exploiting class order) and Image Classification (exploiting semantic similarities using word embeddings), that SimLoss can significantly outperform CCE. We also show that tuning the hyperparameters of both generation techniques influences the model’s performance on metrics measuring either more or less specific predictions.

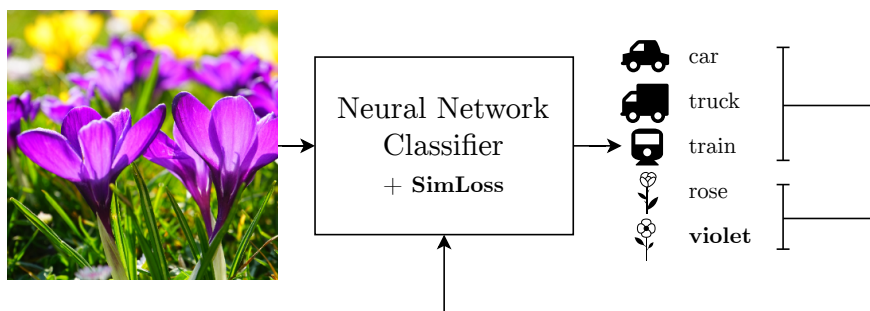


Figure 14.1.: Similarity Based Loss (SimLoss) includes knowledge about class relations in the loss function. In this example, the classes “car”, “truck”, and “train”, as well as “rose” and “violet” are closer together, respectively.

14.1. Methodology

Our proposed SimLoss is based on the CCE, which is introduced in Section 2.4.1. To reiterate, CCE assumes that only one class is correct and is defined as

$$L_{\text{CCE}} = -\frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} \log(\mathbf{p}_i[y_i]), \quad (14.1)$$

where \mathcal{B} is the current batch and $\mathbf{p}_i[y_i]$ is the probability vector output of the network at the target index y_i for the i th example. To model additional knowledge, SimLoss adds a matrix \mathbf{S} , which gives

$$L_{\text{SimLoss}} = -\frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} \log \left(\sum_{c=1}^{|\mathcal{C}|} \mathbf{S}_{y_i, c} \cdot \mathbf{p}_i[c] \right), \quad (14.2)$$

where $\mathbf{S} \in [0, 1]^{|\mathcal{C}| \times |\mathcal{C}|}$ encodes class relations between all possible classes $c \in \mathcal{C}$. $\mathbf{S}_{i, j}$ is the similarity between classes i and j . $\mathbf{S}_{i, j} = 1$ if and only if classes i and j are identical or interchangeable.

SimLoss is equal to CCE if $\mathbf{S} = \mathbf{I}_{|\mathcal{C}|}$ is the identity matrix. Non zero values lead to smaller losses when the network gives a high score to classes similar to the correct one. For misclassifications, this leads the network to predict similar classes.

14.1.1. Relation between Similarity and Probability-based Matrices in SimLoss

Some works from the literature use a loss function similar to our proposed SimLoss. Instead of similarities, the matrix \mathbf{S} consists of probabilities, such that each row sums to one [258, 109]. We will call this loss L_{prob} .

We can show that both similarities and probabilities in the matrix lead to the same gradients: We can transform our loss L_{SimLoss} into L_{prob} by dividing each similarity matrix entry by the row's sum. This loss depends on the network's output — the probability distribution \mathbf{p} — as well as the corresponding target class indices y . It can be written as:

$$L_{\text{prob}} = -\frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} \log \left(\sum_{c=1}^{|\mathcal{C}|} \frac{1}{\sum_{c'=1}^{|\mathcal{C}|} \mathbf{S}_{y_i, c'}} \mathbf{S}_{y_i, c} \cdot \mathbf{p}_i[c] \right) \quad (14.3)$$

$$= -\frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} \log \left(\frac{1}{\sum_{c'=1}^{|\mathcal{C}|} \mathbf{S}_{y_i, c'}} \sum_{c=1}^{|\mathcal{C}|} \mathbf{S}_{y_i, c} \cdot \mathbf{p}_i[c] \right) \quad (14.4)$$

$$= -\frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} \left[\log \left(\sum_{c=1}^{|\mathcal{C}|} \mathbf{S}_{y_i, c} \cdot \mathbf{p}_i[c] \right) - \log \left(\sum_{c'=1}^{|\mathcal{C}|} \mathbf{S}_{y_i, c'} \right) \right] \quad (14.5)$$

$$= -\frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} \left[\log \left(\sum_{c=1}^{|\mathcal{C}|} \mathbf{S}_{y_i,c} \cdot \mathbf{p}_i[c] \right) \right] + \frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} \left[\log \left(\sum_{c'=1}^{|\mathcal{C}|} \mathbf{S}_{y_i,c'} \right) \right] \quad (14.6)$$

$$= L_{\text{SimLoss}} + \frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} \left[\log \left(\sum_{c'=1}^{|\mathcal{C}|} \mathbf{S}_{y_i,c'} \right) \right]. \quad (14.7)$$

$$(14.8)$$

The second summand leads to different loss function values for both matrices, but does not depend on the probability output \mathbf{p} . Therefore, when calculating the gradients with respect to \mathbf{p} , this term becomes zero:

$$\frac{\partial}{\partial \mathbf{p}} L_{\text{prob}} = \frac{\partial}{\partial \mathbf{p}} L_{\text{SimLoss}}. \quad (14.9)$$

As a result, both L_{SimLoss} and L_{prob} yield the same gradients when optimizing the model. If the largest values in the matrices are on the diagonal, both matrix variants will have the same parameters when reaching the global optimum [317]. Even though both methods theoretically lead to the same results, our method is less restrictive since it does not require a probability distribution per row. For example, while similarities can be calculated for each class pair independently, a probability distribution needs to be normalized over all values in the row prior to use. For tasks with a large number of classes, the similarity matrix might not need to be stored but could be calculated on the fly, while probabilities would cause larger computational costs. Especially on edge devices with very limited memory, this is an advantage of our method. Also, a large number of classes would result in very small non-zero probability values in a probability matrix, which might lead to numerical instabilities.

Another advantage of similarities compared to probabilities is that, because the diagonal of the similarity matrix consists of ones, a value of zero can be reached by the loss function, making the loss value more interpretable. A loss value of zero always means that the probability mass of the Neural Network (NN) output vector is put into the correct class, even if there are similar classes. Normalizing such a matrix to ensure probability distributions per row would always yield larger loss values, even if the correct class is predicted with 100% probability. Therefore, L_{SimLoss} always has a lower bound of zero, which gives an interpretable impression of the training status. In a probability matrix based loss, such an interpretation is not given as the lower bound of the loss depends on the probability distributions in the matrix rows. A loss value of zero can only be achieved when using the identity matrix. This, however, would be equivalent to CCE and would not allow for including domain knowledge into the model.

14.1.2. Matrix Generation

We now propose two techniques to generate the matrix \mathbf{S} , which explicitly captures domain knowledge about class relations. Our techniques allow the modeling of class order

14. Improving Classification Models using Class Similarities in the Loss Function

and general class similarities.

Class Order If classes have an inherent order, we can calculate class similarities based on the distance between the class indices. As classes lying next to each other are more similar, we construct the similarity matrix \mathbf{S} as follows: Assuming the same distance between neighboring classes, we define the reduction factor $r \in [0, 1)$ to be the rate at which the similarity will get smaller given the distance to the correct class. The similarity matrix is then

$$\mathbf{S}_{i,j} = r^{|i-j|} \quad \forall i, j \in \{1, \dots, |\mathcal{C}|\}. \quad (14.10)$$

The smaller the reduction factor, the faster the entries converge to 0 with increasing distance to the target class. If the reduction factor is set to 0, the matrix becomes the identity, resulting in the CCE loss. The reduction factor is a hyperparameter of this technique, which can be tuned using a validation dataset to optimize the model for different metrics, as we show in Section 14.2. As SimLoss is equivalent to CCE when $r = 0$ (assuming $0^0 = 1$), an optimized r will always perform at least as good as CCE unless we overfit.

General Class Similarity For some classification tasks, a similarity between classes, such as class names, is available or can be defined. Then, we can use an appropriate similarity measure $sim : |\mathcal{C}| \times |\mathcal{C}| \rightarrow [0, 1]$ that returns the similarity for two classes $i, j \in \{1, \dots, |\mathcal{C}|\}$ and calculate all entries of the similarity matrix \mathbf{S} . Such similarity measures can be manual, semi- or fully-automatic. Additionally, we define a lower bound $l \in [0, 1)$ as a hyperparameter that controls the minimal class similarity that should have an impact on the network punishment. We cut all similarities below l and then scale them such that l becomes 0:

$$\mathbf{S}_{i,j} = \frac{\max(0, sim(i, j) - l)}{1 - l} \quad \forall i, j \in \{1, \dots, |\mathcal{C}|\}. \quad (14.11)$$

Assuming only the diagonal of \mathbf{S} are ones, setting l to the largest similarity value below one ($l = \max(\{sim(i, j) | i, j \in \{1, \dots, |\mathcal{C}|\}, i \neq j\})$) leads to the CCE loss, as only the ones in the diagonal are preserved by the lower bound cut-off.

14.2. Experiments

In the following, we compare SimLoss to CCE by applying them to the same NN model with the same hyperparameters for Age Estimation and Image Classification. For *Age Estimation*, we train NNs on the UTKFace [318] and AFAD [200] datasets. We randomly sample training/validation/test sets using 60/20/20 splits. For *Image Classification*, we use the CIFAR-100 dataset [141]. We also use pretrained Word2Vec embeddings [178] to calculate the semantic similarity between class names. Four class names do not yield a word embedding and are therefore eliminated. Each remaining class has 450 training, 50 validation, and 100 test examples.

We evaluate the *Age Estimation* models using Accuracy (Acc), Mean Absolute Error (MAE), and Mean Squared Error (MSE) (introduced in Sections 2.5.1 and 2.5.2). Accuracy

captures exact predictions, while MAE and MSE capture the distance to the target class, thus considering class order. For *Image Classification*, we use Accuracy, Superclass Accuracy (SA), and Failed Superclass Accuracy (FSA) (introduced in Section 2.5.1). Recall that every example in the CIFAR-100 dataset has a main class and a superclass (e.g., classes “rose” and “orchid” have the superclass “flower”). SA (FSA) determines, how many of the (incorrectly classified) examples are put into the correct superclass. Accuracy only counts exact predictions, while SA and FSA focus on the semantic similarity of the prediction to the target class.

14.2.1. Generating the Similarity Matrix

Since *Age Estimation* has equidistant classes, the similarity matrix can be built using Equation (14.10) without any modifications. In *Image Classification*, we define the similarity matrix as the cosine similarity $sim_{cos} : \mathbf{w} \rightarrow [-1, 1]$ between class name embeddings, where $sim_{cos}(\mathbf{w}, \mathbf{w}) = 1$. To ensure compatibility with the definition in Section 14.1.2, we set $sim(i, j) = \max(0, sim_{cos}(\mathbf{w}_i, \mathbf{w}_j))$ in Equation (14.11).

14.2.2. Experimental Setup

Since SimLoss is a drop-in replacement for CCE, we investigate the effects of changing the loss function on our example tasks. Recall that we do not focus on task specific models, but rather on the evaluation of SimLoss as a general loss function which can be used on various tasks. Both classification tasks are typical examples for using CCE. For *Age Estimation*, we take the Convolutional Neural Network (CNN) from Niu et al. [200] and change the output size to be the dataset’s number of classes. The input images are resized to 60 px by 60 px and the values of all color channels are standardized. We use the softmax function and apply the SimLoss function using the similarity matrix introduced above. We study the effect of the reduction factor r by performing grid search for $r \in \{0.0, 0.1, \dots, 0.9\}$ on the validation set. Optimizing the network using Adam [127] with a learning rate of 0.001 and a batch size of 1024, we employ early stopping [188] with a patience of 10 epochs on the validation MAE. We smooth random differences in the results (e.g., by weight initialization) by averaging over 10 runs. For *Image Classification*, the LeNet CNN [144] is used. Global standardization is applied to the color channels of the input images. We stop early if the Accuracy on the validation set plateaus for 20 epochs of the Adam optimizer with a learning rate of 0.001, and a batch size of 1024. We optimize the matrix generation technique’s lower bound $l \in \{0.0, 0.1, \dots, 0.8, 0.9, 0.99\}$ with grid search and average 10 runs per configuration. $l = 0.99$ makes the loss equivalent to CCE, cutting all similarities except the diagonal.

14.3. Results

Table 14.1 shows the resulting mean metrics for the validation and test sets given a reduction factor r for both *Age Estimation* datasets. The best performing reduction factors on the validation and test set are always higher than 0.0, meaning that SimLoss

14. Improving Classification Models using Class Similarities in the Loss Function

Table 14.1.: Validation and test results averaged over 10 runs on UTKFace and AFAD. Accuracy (Acc) is given in percent. Best validation values are written in bold. Statistically significantly different test values are marked by + or -, if they are on average better or worse than CCE (i.e., $r = 0.0$).

r	UTKFace						AFAD					
	Validation			Test			Validation			Test		
	Acc ↑	MAE ↓	MSE ↓	Acc ↑	MAE ↓	MSE ↓	Acc ↑	MAE ↓	MSE ↓	Acc ↑	MAE ↓	MSE ↓
0.0	15.23	7.09	122.12	14.47	7.39	131.65	11.17	4.05	32.61	11.22	4.10	33.64
0.1	15.43	7.06	119.87	14.48	7.29	127.18	11.21	4.06	32.75	11.30	4.10	33.73
0.2	15.94	7.06	121.28	14.57	7.27	127.13	11.40	4.09	33.52	11.37	4.15 ⁻	34.60 ⁻
0.3	16.25	6.95	117.67	15.17 ⁺	7.19 ⁺	125.70	11.34	4.10	33.53	11.38 ⁺	4.16 ⁻	34.53 ⁻
0.4	16.13	6.95	117.52	15.46 ⁺	7.18 ⁺	125.74	11.33	4.10	33.44	11.45 ⁺	4.16 ⁻	34.56 ⁻
0.5	16.10	6.89	115.59	15.09	7.18 ⁺	123.94	11.44	4.06	33.02	11.49 ⁺	4.13	34.21
0.6	15.62	6.83	112.85	14.34	7.09 ⁺	120.34 ⁺	11.26	4.01	31.99	11.31	4.05 ⁺	32.84 ⁺
0.7	14.39	6.79	110.12	13.07	7.08 ⁺	121.19 ⁺	11.22	3.95	31.17	11.11	4.02 ⁺	32.36 ⁺
0.8	13.50	6.74	108.80	12.57 ⁻	7.01 ⁺	117.99 ⁺	8.58	4.58	38.69	8.55 ⁻	4.64	39.78
0.9	9.69	6.90	106.23	9.16 ⁻	7.18 ⁺	117.62 ⁺	6.55	5.09	44.87	6.47 ⁻	5.15 ⁻	45.82 ⁻

outperforms CCE. Choosing the reduction factor then depends on the metric to optimize for. For UTKFace, a reduction factor of 0.3 leads to the best validation Accuracy, while 0.8 or 0.9 optimize MAE and MSE, respectively. For AFAD, $r = 0.5$ yields the best validation result on Accuracy, while $r = 0.7$ results in the best MAE and MSE. Overall, choosing a smaller reduction factor $r \approx 0.4$ optimizes the Accuracy, while larger $r \approx 0.8$ optimizes MAE and MSE. This is because large r lead to higher matrix values and thus smaller punishments for estimating a class near the correct age. A model optimized for that is favored by metrics that accept approximate matches, such as MAE or MSE.

A Wilcoxon signed-rank test [293, 294] with a confidence level of 5% shows that optimizing the reduction factor always leads to significant improvements over CCE. Sometimes, however, choosing the reduction factor based on a specific metric also results in a trade-off between the chosen and other metrics.

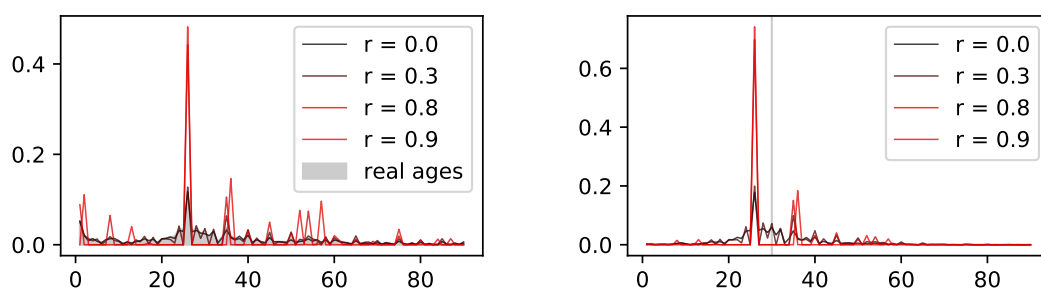
For the *Image Classification* task, Table 14.2 shows the results for the validation and test set of the CIFAR-100 dataset given a lower bound l . On average, the best performing model always has a lower bound of less than 0.99, again showing that SimLoss outperforms CCE. Also, a statistical test reveals that $l = 0.9$ gives significantly better results on the test set in terms of Accuracy and SA. Smaller lower bounds tend to reduce the Accuracy as the loss function hardly punishes any misclassification. For $l \approx 1$, the loss is equivalent to CCE, forcing the network to predict the correct class, thus increasing Accuracy. In between, the network is guided to predict the correct class but is also not punished severely for misclassifications of similar classes. This improves SA, which pays attention to more similar classes.

Table 14.2.: Validation and test results over 10 runs with early stopping on the modified CIFAR-100 dataset. Best validation values are written in bold. Statistically significantly different test values are marked by + or -, if they are on average better or worse than CCE (i.e., $l = 0.99$).

l	Validation			Test		
	Accuracy \uparrow	SA \uparrow	FSA \uparrow	Accuracy \uparrow	SA \uparrow	FSA \uparrow
0.99	46.89 %	55.78 %	16.73 %	39.51 %	49.22 %	16.05 %
0.90	47.42 %	56.32 %	16.95 %	40.15 % ⁺	49.93 % ⁺	16.36 %
0.80	46.37 %	55.38 %	16.80 %	39.49 %	49.32 %	16.22 %
0.70	46.95 %	55.92 %	16.90 %	39.86 %	49.63 %	16.25 %
0.60	47.28 %	56.44 %	17.39 %	40.00 %	50.00 %	16.67 % ⁺
0.50	46.36 %	56.18 %	18.28 %	39.26 %	49.40 %	16.70 % ⁺
0.40	38.03 %	50.58 %	20.28 %	32.18 % ⁻	44.58 % ⁻	18.30 % ⁺
0.30	28.65 %	43.76 %	21.18 %	24.43 % ⁻	38.90 % ⁻	19.13 % ⁺
0.20	21.66 %	37.97 %	20.80 %	18.54 % ⁻	33.68 % ⁻	18.58 % ⁺
0.10	16.40 %	31.68 %	18.31 %	14.25 % ⁻	28.70 % ⁻	16.85 %
0.00	2.80 %	8.37 %	5.77 %	2.53 % ⁻	8.06 % ⁻	5.71 % ⁻

14.4. Analysis

To understand the effect of SimLoss, we focus on Age Estimation whose one dimensional classes are easy to visualize. We compare the best performing models for UTKFace trained using SimLoss and CCE, i.e., models with $r \in \{0.0, 0.3, 0.8, 0.9\}$ (see Table 14.1). For each r , we plot the mean output distribution for all examples in the dataset as well as the real age distribution, which is shown in Figure 14.2a. CCE ($r = 0.0$) resembles the real age distribution the best, while higher reduction factors tend to aggregate groups of multiple age classes. With a higher reduction factor, the number of spikes decreases and the distances between them increase: The model chooses representative classes to which it mainly distributes the output probability mass. This becomes apparent in Figure 14.2b, where we plot the mean output distribution for all examples of age 30. The network with $r = 0.9$ focuses its probability output to the two nearest representative classes, in this case “26” and “35”. The Accuracy of the network decreases, as the output probability mass is not on the correct class, but the distance of the prediction to the correct class is smaller than for CCE. Representative classes are apparently chosen such that frequent items receive more probability mass from the model. A higher reduction factor therefore leads to a coarser class selection. This can be explained by the optimization objective of the loss function. The loss should be smaller for misclassifications of similar classes than for dissimilar classes. Representing multiple similar classes as one class and predicting it more often for similar classes does not lead to the smallest possible loss value. However, the loss gets smaller compared to predicting dissimilar classes, as the punishment should be smaller for classifying a similar class. In the case of Age Estimation, predicting an age that lies close to the correct age will decrease the Accuracy, but perform better than



(a) All examples. CCE fits the real data distribution the best. (b) All examples of class "30". The grey line indicates the target age.

Figure 14.2.: Mean probability distribution output for different r . High reduction factors lead the network to choose only few representative classes.

CCE on MAE and MSE. In Image Classification, selecting one or multiple representative classes leads to smaller Accuracy but to higher SA and FSA than CCE. Higher similarities in the matrix thus guide the network to make coarser predictions, improving metrics that accept predictions of similar classes. The results from Section 14.2 also show that keeping the loss near CCE by choosing the similarity matrix conservatively can improve on specific prediction metrics such as Accuracy as well.

14.5. Conclusion

In this section, we have presented SimLoss, a modified CCE loss function that incorporates domain knowledge about class relations in form of class similarities, thus implementing the **Loss Function** principle. We have introduced two techniques to prepare similarity matrices to exploit class order and general class similarity that can be used to significantly improve the performance of NN classifiers on different metrics. Also, SimLoss helped with predicting more similar classes if the model misclassified an example. In our analysis, we found that SimLoss forced the model to focus on choosing representative classes. The number of representative classes can be implicitly tuned by a hyperparameter. While finding the best hyperparameter and similarity metric can be computationally expensive and non-trivial, SimLoss can incorporate arbitrary similarity metrics into a classifier.

15. Training a Sentiment Analysis Model with Weak Labels and Input Masking

In this section, we mainly follow our work in [133] where we explore the combination of the **Input Masking and Augmentation** and **Weak Label Generation** principles to train a sentiment analysis model on Twitch.tv¹ chat messages without ground truth labels. Since live streaming gaming sessions has evolved into a very lucrative business, streamers are interested in knowing how their audience reacts to certain events in their streams. The comments in the fast moving chat section of a stream — where users can interact with each other and comment on the stream in a timely fashion — can give valuable feedback to the streamer. Automatically estimating the sentiment trends of the comments enables streamers to adapt their behavior or presentation in real-time, or learn for future streams in order to achieve the desired emotions from the audience. Due to popular streams getting many comments per second, automatically estimating the comments’ emotions in real-time would facilitate this implicit way of gathering feedback. In this implementation, we automatically assess the emotion of comments by applying sentiment analysis methods. This way it is possible to check whether an event is positively or negatively perceived by the audience, which helps streamers understand the preferences of their target audience.

The biggest challenge in performing sentiment analysis on Twitch comments is the non-standard language. An impression of Twitch language usage can be seen in Figure 15.1. The language consists of many abbreviations, intentional and unintentional grammatical and orthographic mistakes, duplicated phrases, and short sentences. Pictographical images and animations called emotes are also very popular² due to their ability to express emotions in a way that is easily interpretable by the human eye. The way emotes are used makes them an own form of language that is captured in the comment by the emote’s, sometimes cryptic, text representation. For this reason, lexicon based sentiment analysis methods designed for common English typically fail to correctly classify Twitch comments.

To overcome the issue of having no direct training data, we explore the suitability of emotes as emotion indicators to perform sentiment analysis on Twitch comments, and introduce multiple methods that rely on emote-, emoji-, and word-sentiment lexica. We show that emotes are a good complement to other lexica. Additionally, we compare two types of lexica: *average based* sentiment lexica that provide one sentiment score per word and *distribution based* sentiment lexica that contain a distribution over all classes based on the annotator’s votes. We show that distribution based sentiment lexica improve our test scores as they provide more information regarding “controversial” emotion indicators, i.e., words that can have positive as well as negative connotations.

¹<https://www.twitch.tv> (last accessed: 2023-02-10)

²<https://stats.streamelements.com/c/global> (last accessed: 2023-02-10)

15. Training a Sentiment Analysis Model with Weak Labels and Input Masking

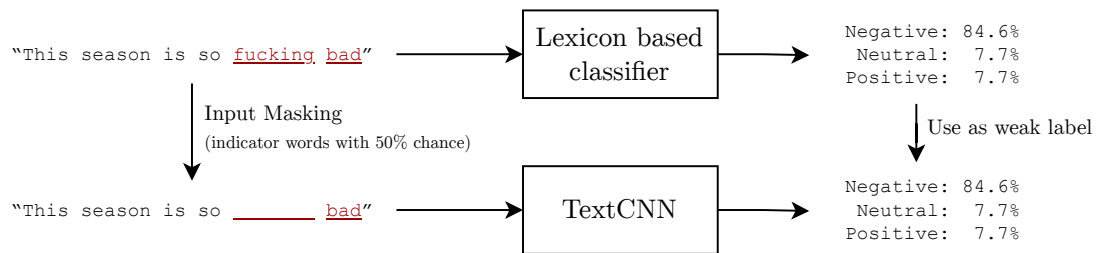


Figure 15.2.: Overview of our training strategy of the Deep Learning (DL) model for sentiment analysis of Twitch comments. We generate weak labels using a lexicon based approach and use input masking to let the Neural Network (NN) not directly imitate the weak labeling approach.

We then use the **Weak Label Generation** principle and generate weak labels for a very large unlabeled dataset of Twitch comments using the distribution based sentiment lexicon approach. Introducing additional domain knowledge to the model in this way, we train a Convolutional Neural Network (CNN) model on these weak labels (see Figure 15.2). Arguing that the model would just learn to take the signal words from the lexica into account and thus effectively mimicking it, we explore the use of the **Input Masking and Augmentation** principle and randomly mask words that were used for the weak labeling process. This way, we aim to guide the model to take other words from the input text into account and find other correlations to classify the comments. We thus investigate whether this combination of principles leads to better generalization of the classifier on texts that do not have signal words in them.

15.1. Methodology

The basic task in sentiment analysis is to classify a text as one of the classes “positive”, “negative”, and “neutral”. Due to Twitch comments being fairly short and often not containing punctuation, we follow the structurally similar setting employed by the sentiment classification tasks on Twitter messages of Rosenthal et al. [229], which is predicting the sentiment of entire comments. Given enough comments over time in a chat, the overall sentiment of the audience can be estimated by aggregating the sentiment of all individual comments. We thus aim to train a Neural Network (NN) that can predict the sentiment of a comment given its text.

Since there are no Twitch.tv related labeled sentiment analysis datasets, it is not

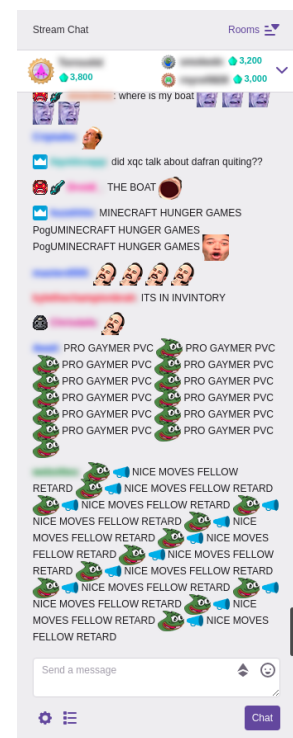


Figure 15.1.: Screenshot of a Twitch comment section. The language used in the comments is fairly different from common English. In the bottom right, an emote picker helps with selecting an emote. User names are anonymized due to privacy reasons.

possible to perform supervised learning. We thus explore the **Weak Label Generation** principle by training the NN in a weakly supervised manner. To weakly label Twitch.tv messages, we propose to make use of the emotes that are used in Twitch comments.

15.1.1. Weak Label Generation using Sentiment Lexica












To generate weak labels for our NN based approach, we propose to use a sentiment lexicon based approach. Sentiment lexica are a commonly used resource in sentiment analysis. Generally, they are lists of words associating each word with a polarity, providing valuable hints for the sentiment conveyed by sentences including these words. We use three lexica for weakly labeling our data, namely a word-, emoji-, and emote-based lexicon:

- **The VADER lexicon** is a word based sentiment lexicon [69], providing a list of 7517 English words, phrases and ASCII text emoticons together with ratings by ten subjects on an integer scale from -4 (very negative) to 4 (very positive).
- **The Emoji lexicon** by Kralj Novak et al. [138] contains 969 unicode emojis and their respective sentiment distribution based on the sentiment of tweets these emojis appear in. To ensure reliable labels, we only considered the emojis that appear in 50 or more tweets, which yields annotations for 300 unicode emojis.
- **The Emote lexicon** is of special importance for Twitch comments. Since there is no labeled sentiment lexicon for emotes, we create our own by labeling the top 100 emotes measured by the usage frequency in the unlabeled dataset we introduced in Section 3.7.2. To label these emotes, a survey was conducted using Google forms and published on two gaming-related Twitter accounts and on various gaming and Twitch related subreddits on Reddit³ to ensure that mainly users of Twitch and therefore people with background knowledge about emotes and emote usages were participating. Participants were then shown images and text representations of the emotes and were instructed to “rate [the emotes] as either negative, neutral or positive, according to the sentiment of the situation in which you would or already have used this emote.” Overall, the survey received answers from 108 participants, which was sufficient to show clear tendencies for the sentiment of most emotes. More information on the survey, its participants, and the results can be found in [133], but an excerpt of the resulting lexicon can be found in Table 15.1.

We construct three distribution based lexica $\mathcal{L}_{\text{dist}}$ from the provided sentiment scores by saving the distribution over the three possible classes “positive”, “neutral”, and “negative” for each word, emoji, and emote. While the emoji and emote lexica already offer only three classes, we group the VADER scores -4 to -2 as negative, -1 to 1 as neutral and 2 to 4 as positive and construct the distribution over these labels. Hence, the distribution based lexica $\mathcal{L}_{\text{dist}}$ are a partial function that maps each token from the vocabulary for which lexica entries are available $\mathcal{V}_{\text{dist}}$ to a three-dimensional vector whose sum of values is

³<http://reddit.com> (last accessed: 2023-02-10)

Table 15.1.: Some answers to our Twitch emote sentiment survey.

Sentiment		Negative	Neutral	Positive	Unknown/NA
Emote					
	FeelsBadMan	71	17	19	1
	FeelsGoodMan	1	7	98	2
	LUL	11	23	72	2
	OMEGALUL	17	26	62	3
	PogChamp	1	3	101	3
⋮		⋮	⋮	⋮	⋮
	Jebaited	25	27	37	19
⋮		⋮	⋮	⋮	⋮
	mcaT	10	34	12	52
	forsenPls	13	26	17	52
	PepoDance	9	26	20	53
	RedCoat	5	46	4	53
	jinnytHype	7	31	17	53

one: $\mathcal{L}_{\text{dist}} : \mathcal{V}_{\text{dist}} \rightarrow [0, 1]^3$. Each entry $\mathcal{L}_{\text{dist}}(t)$ for a token t is the probability distribution for the three possible classes given this token.

As preprocessing for our weak label generation, we lowercase and tokenize each Twitch.tv comment into words and punctuation while preserving emoticons, unicode emojis and capitalization of Twitch emotes. To standardize occurring words for our learning procedures, we replaced occurrences of urls with the tag “URL” and reduced characters occurring more than twice in succession in a word to two occurrences (e.g., “loooove” is standardized to “loove”). We then look up each token in the three lexica. In case of entries that are present in multiple lexica, the emote lexicon takes precedence over the emoji lexicon, which in turn supersedes VADER, representing how specialized the lexica are to the domain of Twitch comments.

Given the tokenized comment $T = (t_1, t_2, \dots, t_m)$, we construct the sequence $T^* = (t_i \mid t_i \in \mathcal{V}_{\text{dist}})$ of tokens that are present in the lexica $\mathcal{L}_{\text{dist}}$. We now want to predict the correct class c for this list $T^* = (t_1, t_2, \dots, t_n)$ using $p(c \mid t_i)$ where $i \in 1, \dots, n$ and $c \in \{\text{negative, neutral, positive}\} = \mathcal{C}$. This can be done by assigning the most likely class c^* to T^* given t_1, \dots, t_n , that is $c^* = \arg \max_{c \in \mathcal{C}} p(c \mid t_1, \dots, t_n)$. The standard naive Bayes formula

$$c^* = \arg \max_{c \in \mathcal{C}} p(c \mid t_1, \dots, t_n) = \arg \max_{c \in \mathcal{C}} p(c) \prod_{i=1}^n p(t_i \mid c) \quad (15.1)$$

cannot be applied here as we want to classify in an unsupervised manner and do not have examples to infer $p(t_i | c)$ from. However, using Bayes' theorem and assuming conditional independence $p(t_1, \dots, t_n | c) = \prod_{i=1}^n p(t_i | c)$, we can show that

$$c^* = \arg \max_{c \in \mathcal{C}} p(c | t_1, \dots, t_n) \quad (15.2)$$

$$\stackrel{\text{Bayes}}{=} \arg \max_{c \in \mathcal{C}} \frac{p(t_1, \dots, t_n | c)p(c)}{p(t_1, \dots, t_n)} \quad (15.3)$$

$$= \arg \max_{c \in \mathcal{C}} p(t_1, \dots, t_n | c)p(c) \quad (15.4)$$

$$\stackrel{\text{Indep.}}{=} \arg \max_{c \in \mathcal{C}} \prod_{i=1}^n p(t_i | c)p(c) \quad (15.5)$$

$$\stackrel{\text{Bayes}}{=} \arg \max_{c \in \mathcal{C}} \prod_{i=1}^n \frac{p(c | t_i)p(t_i)}{p(c)} p(c) \quad (15.6)$$

$$= \arg \max_{c \in \mathcal{C}} \prod_{i=1}^n p(c | t_i)p(t_i) \quad (15.7)$$

$$= \arg \max_{c \in \mathcal{C}} \prod_{i=1}^n p(c | t_i) \prod_{i=1}^n p(t_i) \quad (15.8)$$

$$= \arg \max_{c \in \mathcal{C}} \prod_{i=1}^n p(c | t_i) \quad (15.9)$$

$$= \arg \max_{c \in \mathcal{C}} \prod_{i=1}^n \mathcal{L}_{\text{dist}}(t_i), \quad (15.10)$$

which only uses the sentiment distributions $p(c | t_i)$ given by our lexica $\mathcal{L}_{\text{dist}}$. We use this fact to build a probabilistic classifier that computes c^* as its prediction, if any token in the comment is present in one of the lexica:

$$\text{sentiment}(T^*) := \begin{cases} c^* & \text{if } |T^*| > 0 \\ \text{neutral} & \text{otherwise.} \end{cases} \quad (15.11)$$

Given this classifier, we can now generate weak labels for the unlabeled dataset (see Section 3.7.2). The labels are weak, since they are only based on the presence of tokens in the lexica. Our distribution based approach labels messages as “neutral” if they contain no signal tokens found in any of the three lexica. On the one hand, they may actually be neutral and therefore not contain any signals. On the other hand, many of the comments in Twitch contain orthographic mistakes or Twitch specific words which are not covered by any of the lexica but might still provide valuable information about the sentiment in a comment. The resulting labels are therefore not perfect. However, we can easily generate a large amount of weak labels for the unlabeled dataset, which can be used to train a more robust classifier.

15.1.2. Input Masking of the NN

Given a large dataset of weakly labeled comments, we can train a NN to classify comments into the three possible classes. Our intuition is that the network will be able to find the relation between a comment’s sentiment and words that are not covered by the lexica, as well as being more robust to orthographical errors. This stems from converting the input tokens to embedding vectors to allow the model to work with them. Here, frequent typos are given a similar embedding, and can therefore be evaluated correctly by the NN, while a typo cannot be found in a lexicon.

As our NN model, we use the TextCNN for sentiment analysis introduced by Kim [126] and already explained in Section 2.2.3. As inputs to the model, we use Word2Vec embeddings [178] trained on the unlabeled corpus as input representation (see Section 2.3.1). To train the embeddings, we use the preprocessing described above on our unlabeled dataset and filter all words that occur fewer than 100 times. This means that some words are not available in the training phase of the network. Tokens that yield no embedding are replaced by zero vectors. We use the “cnn-nonstatic” variant of the TextCNN, which means that the embedding layer of the model is initialized with the pretrained embeddings. This layer is then finetuned along with the other network weights during training.

We train the network by feeding the labels produced by our distribution based lexicon classifier as targets. This approach predicts a neutral label for every comment that has no token present in one of the used sentiment lexica. Since this is only a default assumption and not a label actually provided by the classifier, we model these predictions as uncertain. Therefore, for any comment that does not contain any signal tokens from our lexica, we use a target distribution of 25 % negative, 50 % neutral, and 25 % positive as target.

As this might lead to the network simply overfitting to this target distribution, we adapt a method proposed by Go et al. [71]: masking signal tokens from the network’s input, which implements the **Input Masking and Augmentation** principle. This forces the network to look for other words, phrases, and structures in the comment that correlate with its sentiment. In order to enable the network to rely on both signal tokens from the lexica and possible new clues, we replace any signal token with a zero vector with a 50 % probability.

15.2. Experiments

With the weak labels obtained by the distribution based lexicon approach on the large unlabeled dataset, we train a TextCNN on the resulting dataset and evaluate its performance on the labeled test set, which is described in Section 3.7.2. The metrics for evaluating our results are Accuracy, Macro Recall and Macro F1 score (described in Section 2.5.1). Since Accuracy (i.e., the percentage of correctly classified samples) tends to overestimate the predictive power of a classifier when classes are unbalanced, results are here reported using the Macro F1 score as well as the Macro Recall.

To find the best hyperparameters for the CNN, we use approx. 20 % of the unlabeled Twitch chat message training dataset (introduced in Section 3.7.2) as a validation set and perform a random search over the CNN filter count and dropout probability [255],

following Zhang and Wallace [316]. After training about 30 different configurations, we select the model with the lowest validation loss. This results in a TextCNN consisting of 182 filters and a dropout probability of 27% on the CNN layers during training. During training, we use early stopping on the validation loss, which is calculated after every 500 batches consisting of 2816 training examples. If the validation loss does not improve during 5 consecutive validation iterations, we stop and use the training state that produces the lowest loss.

15.2.1. Baselines

We compare our proposed weakly supervised approach with the following baselines:

Random Baseline This very simple baseline consists of two possible strategies: (i) Sampling uniformly from the three possible sentiment labels for each comment and (ii) exploiting the knowledge about the distribution of the labeled dataset and then sampling randomly from this distribution.

Majority Baseline The most common class in the evaluation dataset is “positive” with 40.06%. This baseline always predicts the “positive” class.

NLTK VADER Baseline As a more sophisticated baseline, we choose the sentiment analysis system that was proposed along with the VADER lexicon [69] and is implemented in the Python NLTK module⁴. This module uses the VADER lexicon, as well as some rules to combine the word labels for predicting the overall sentiment of a text. Rules include intensification of all-caps words, dampening a word’s sentiment if preceded by “kind of”, or negating the sentiment when a negation word is found. VADER serves as a relatively strong baseline, as it was designed specifically for social media texts.

Distribution Based Lexicon Approach We can use the weak label generation process as the baseline for our approach. Here, the output of the model is directly used as its prediction. We can use this baseline to estimate the performance gain of our approach over the distribution based lexicon approach.

Average Based Lexicon Approach The distribution based lexicon approach requires the lexica to provide distributions over the sentiment labels. We thus want to test another method that simplifies this requirement by using the average sentiment score of a token instead of a distribution. We call these the average based lexica \mathcal{L}_{avg} , a partial function that maps a token from the available vocabulary $t \in \mathcal{V}_{\text{avg}}$ to a number $\mathcal{L}_{\text{avg}}(t) \in [-1, 1]$, where higher numbers indicate a more positive sentiment: $\mathcal{L}_{\text{avg}} : \mathcal{V}_{\text{avg}} \rightarrow \mathbb{R}$. After applying the same preprocessing and tokenization as for the distribution based lexicon approach,

⁴<https://www.nltk.org/api/nltk.sentiment.vader.html#nltk.sentiment.vader.SentimentIntensityAnalyzer> (last accessed: 2023-02-10)

15. Training a Sentiment Analysis Model with Weak Labels and Input Masking

Table 15.2.: Results for sentiment classification achieved by all methods. Best values are given in bold.

Method	Accuracy \uparrow	Macro Recall \uparrow	Macro F1 score \uparrow
Random Baseline	33.3%	33.3%	32.7%
Random Baseline (sampling from target distribution)	35.6%	33.3%	33.3%
Majority Baseline	40.1%	33.3%	19.1%
NLTK Baseline	43.0%	39.3%	34.0%
Average Based Lexicon Approach	61.8%	58.9%	60.5%
Distribution Based Lexicon Approach	62.8%	60.5%	61.7%
TextCNN	63.8%	61.4%	62.6%

the token sequence $T^* = (t_i \mid t_i \in \mathcal{V}_{\text{avg}})$ that consists only of the tokens that are present in at least one of the lexica is generated. T^* is then scored as follows:

$$\text{score}(T^*) := \begin{cases} \text{average}(\mathcal{L}_{\text{avg}}(t) \mid t \in T^*) & \text{if } |T^*| > 0 \\ 0 & \text{otherwise.} \end{cases}$$

Thus, the score of an entire comment is the average over all scores of tokens the lexica provide an entry for. This results in a continuous score between -1 and 1. To receive the final, discrete sentiment labels of *negative*, *neutral* or *positive*, thresholds were introduced as follows:

$$\text{sentiment}(T^*) := \begin{cases} \text{negative} & \text{if } \text{score}(T^*) < -0.33 \\ \text{neutral} & \text{if } -0.33 \leq \text{score}(T^*) \leq 0.33 \\ \text{positive} & \text{if } 0.33 < \text{score}(T^*). \end{cases}$$

15.3. Results

Table 15.2 shows Accuracy, Macro Recall, and Macro F1 score for all methods on the labeled dataset. We report the expected metrics for the random baselines. Sampling uniformly from the three possible sentiment labels for each comment produces the worst performance for our tested methods. Exploiting knowledge about the distribution of the labeled dataset and sampling randomly from this distribution, we can increase the expected Accuracy to 35.6% and Macro F1 score to 33.3%, while the expected Macro Recall stays the same at 33.3%. The Majority Baseline, i.e., always predicting the “positive” class, as it is the most frequent one in the dataset, leads to an improved Accuracy but worse performance for the Macro F1 score.

Even though VADER is specifically designed for dealing with social media texts, the Macro F1 score obtained by this method is 34.0%, which is only a small increase in contrast to randomly selecting labels. This is due to the fact that the language used

on Twitch is very different from common English and even from common social media language. The Macro Recall and Accuracy, however, increase to 39.3 % and 43.0 %, respectively.

Our simplest method based on multiple sentiment lexica, the average based lexicon approach, yields an Accuracy of 61.8%, Macro Recall of 58.9 % and a Macro F1 score of 60.5 %. 65.2 % of comments in our evaluation dataset contained tokens found in our lexica and were therefore labeled by the classifier. The other 34.8 % were assigned the “neutral” label by default. The large improvement over the other baselines shows that incorporating sentiment lexica for emoji and emotes can provide reasonable accuracy even with a rather simple classifier.

Our weak label generation method, the distribution based lexicon classifier, achieves an Accuracy of 62.8%, a Macro Recall of 60.5%, and a Macro F1 score of 61.7 %, which is an improvement to the previous approach. As above, 65.2 % of the comments in the dataset had tokens found in the lexica, the remaining comments were labeled as neutral by default.

The best method in our experiments is our weakly supervised TextCNN, which achieves an Accuracy of 63.8 %, a Macro Recall of 61.4 %, and a Macro F1 score of 62.6 %. This result improves the distribution based lexicon approach, even though the weak labels were produced by the lexicon based method, which provides evidence that the combination of weak labels and input masking can lead to improved performance.

15.4. Analysis

We have shown that our weakly supervised model outperforms the baselines as well as the weak label generation method. In the following, we provide some analyses of the model to better understand if the better performance is due to the applied principles. For more analyses and case studies that do not directly relate to these specific principle implementations, see our work in [133]. There, we analyze the capabilities of the Word2Vec embeddings trained on Twitch messages and provide case studies where we apply our sentiment analysis model to Twitch streams and show that certain stream events correlate with the sentiment of the chat.

15.4.1. Ablation Study: Emotes Matter

In order to validate our assumption that emotes have major influence on the sentiment of Twitch chat messages, we conduct an ablation study for our two lexicon based classifiers, investigating the influence of different lexica. We find that both approaches profit strongly from the inclusion of emotes. Table 15.3 shows the results for all combinations of the three lexica. Along with the measures Accuracy, Macro Recall, and Macro F1 score, the table shows the percentage of messages with at least one token found in the lexicon. The emoji lexicon does not improve the classification performance by much, but increases the amount of messages that are not simply assigned a default “neutral” label by two percentage points. It can be seen that all lexica are relevant to the classification, while the emote lexicon has the single largest influence. Also note that the emote lexicon covers

15. Training a Sentiment Analysis Model with Weak Labels and Input Masking

more messages than any other lexicon. These findings are in line with our expectation that emotes are crucial for the understanding of comments on Twitch.

Table 15.3.: Results of both lexical approaches using different combinations of lexica. Best results are written in bold.

	Accuracy \uparrow Average/Distribution	Macro Recall \uparrow Average/Distribution	Macro F1 score \uparrow Average/Distribution	% of comments containing signal tokens
Emoji	39.5 % / 39.8 %	34.0 % / 34.3 %	21.1 % / 21.4 %	3.8 %
Vader	48.3 % / 45.9 %	45.1 % / 43.1 %	42.1 % / 38.1 %	26.6 %
Emoji + Vader	48.5 % / 46.4 %	45.3 % / 43.6 %	42.7 % / 39.2 %	29.1 %
Emote	58.9 % / 59.7 %	54.3 % / 55.2 %	55.1 % / 56.0 %	48.0 %
Emote + Emoji	58.9 % / 60.3 %	54.4 % / 55.8 %	55.3 % / 56.7 %	50.6 %
Emote + Vader	61.8 % / 62.4 %	58.8 % / 60.2 %	60.4 % / 61.3 %	63.4 %
Emote + Emoji + Vader	61.8 % / 62.8 %	58.9 % / 60.5 %	60.5 % / 61.7 %	65.2 %

15.4.2. Comparison of Approaches: Complexity Matters

In addition to the ablation study presented above, we analyze the differences between the predictions our classifiers make in order to enable a better understanding of their relative performance. Despite similar numeric results in the average and distribution based approach and a Spearman correlation coefficient of 0.88, there are a number of cases where the approaches classify messages differently. In fact, both approaches are significantly different from each other with a significance level of 1%, based on the Randomized Matched-Pair Test by Yeh [309] (p-value for F1 score: 1.9×10^{-6}). Using the same test, comparing the CNN and distribution based approach also shows significant difference with a p-value for F1 score of 2.9×10^{-6} .

As previously mentioned, approx. 35% of messages in the evaluation dataset did not contain tokens present in the lexica. These messages were assigned the “neutral” label by default by the lexicon based approaches. When comparing the results of all three classifiers, it is noticeable that in contrast to our expectations, the CNN did not improve the classification of these messages. Almost all of these 669 messages (i.e., 35% of the evaluation dataset) were also classified as neutral by the CNN. However, as seen in Table 15.4, the TextCNN generally classifies fewer messages as neutral. This means that messages that have signal tokens in them but were classified as neutral by the lexicon (e.g., because of not reaching the threshold) based approaches are more likely to be classified as positive or negative by the CNN. This seems to be the largest influence for the improved performance over the average based and distribution based approaches. The fact that the CNN does not improve the classification of messages without signal tokens suggests that the input masking is not the main reason for the improved performance of the CNN. Inspecting some messages and their predicted probability distributions across the three labels “negative”, “neutral”, and “positive”, we find that the distribution based approach often has the same probabilities for at least two labels. This is due to the lexica entries often having the same probability for multiple labels. Choosing the label with the largest

Table 15.4.: Distribution of classified messages of all three approaches and the original evaluation dataset.

Classifier	Negative	Neutral	Positive
True Sentiment	404	748	770
Average Based Classifier	237	1027	658
Distribution Based Classifier	290	971	661
TextCNN	281	962	679

Table 15.5.: Amount of messages labeled as negative/neutral/positive by the classifiers in comparison to the true sentiment. Excluding messages with default neutral sentiment due to undetected tokens.

True Sentiment \ Estimated Sentiment	Average Based Lexicon Approach			Distribution Based Lexicon Approach			TextCNN		
	neg.	neu.	pos.	neg.	neu.	pos.	neg.	neu.	pos.
negative (312)	174	112	26	195	80	37	193	83	36
neutral (320)	49	112	159	71	104	145	63	111	146
positive (621)	14	134	473	24	118	479	25	100	496

probability then can lead to possibly undesirable labels, since the maximum function from the used Python library NumPy [83] usually takes the first occurrence of the largest value, making the prediction more negative. The TextCNN, due to its NN approach, does not give the same probability for multiple labels. We suspect that this leads to a more nuanced and thus accurate prediction of the sentiment.

Table 15.5 compares the sentiment predicted by our methods in comparison to the true sentiment. The table shows that, except for true neutral messages, the classifiers show a clear tendency to correctly classify negative and positive messages. Only very few messages have completely contrary sentiment where the classifiers predict negative sentiment for a message labeled as positive by the crowd workers or vice versa.

15.5. Conclusion

In this implementation of the **Input Masking and Augmentation** and **Weak Label Generation** principles, we have presented a NN based method that is able to reliably estimate the sentiment of Twitch comments. This allows streamers to visualize trends in the audience’s sentiment to get feedback for their product or stream and perhaps adapt their presentation accordingly.

The basic assumption was that, due to the very specific language of Twitch comments, generic sentiment analysis approaches would fail to provide satisfactory classifications

on Twitch data. We also hypothesized that using emotes could be a way to overcome the challenge posed by this language, as they make up a large part of Twitch comments. Our experiments show that both of these assumptions are correct: The VADER baseline, even though it is designed for social media texts, cannot capture the sentiment expressed in Twitch comments. Our methods however, which include sentiment information about emotes in addition to words and emojis, are able to detect sentiment with reasonable accuracy. Our ablation study has shown that this is indeed mostly due to the emote lexicon.

A common shortcoming of lexicon based classifiers is their inability to deal with spelling errors or, more, generally, words not contained in the lexica they are based on. We proposed to use a CNN based on word embeddings to enable generalization to unknown words. Our analysis shows that, while the CNN does indeed perform better than the lexicon based classifiers, this improvement is not due to the generalization we had hoped for. This could be due to the network overfitting to the target distribution given by our distribution based lexicon classifier. We had hoped that marking the default neutral classification for comments that do not contain words in our lexica as uncertain by representing it as 25 % positive, 50 % neutral, and 25 % negative would be enough to prevent this, however this does not seem to be the case. Exploring other methods to enforce better generalization is an interesting topic for future work. Possible approaches include providing a target distribution closer to the uniform distribution, deleting uncertain training examples with a given chance, or modifying the learning rate of the NN to be lower when the label is uncertain.

Despite the better results, the CNN requires time-consuming training and hyperparameter optimization as well as rather large amounts of storage space for the embeddings and weights compared to the lexicon based approaches. While this does not pose a significant problem for most applications, it could be relevant for real-time use. Our lexicon based classifiers could easily be integrated into a browser plugin to provide streamers with real-time information about their audience’s sentiment and enable them to adjust their stream accordingly. On the other hand, the slightly higher accuracy of the CNN could be useful for offline analysis of comments after events.

16. Improving Image Aesthetics Assessment (IAA) Models using Self-Supervised Pretraining with Image Augmentations, Weak Labels, and Multitask Learning

We now revisit the task of Image Aesthetics Assessment which is described in Section 3.2 and combine several principles in this implementation in order to improve the performance of IAA models. For this, we mainly follow our work in [208].

As already discussed, assessing the aesthetics of an image automatically can be used in many applications. Modern IAA methods are based on Convolutional Neural Networks (CNNs) that receive an image as input and output a score that is higher for more aesthetic images. Such models are usually initialized with weights trained on the ImageNet classification task [46] to build on the already learned features by finetuning the network on a labeled dataset such as Aesthetic Visual Analysis (AVA) [191]. As we already have discussed in Chapter 12, we argue that the ImageNet classification task is not well-suited for IAA models, since it is not designed to teach the network aesthetically relevant features. Due to the classification objective, it even discourages features important for IAA. For example, a classification network should be invariant to the image’s lighting conditions and thus discourages features taking the image’s brightness into account. We therefore propose to pretrain the model on pretext tasks that are specifically designed to let the network learn relevant features to assess the aesthetics of an image.

While in Chapter 12, we proposed to handle this by using the Contrastive Language-Image Pre-Training (CLIP) model as a fixed general feature extractor, we now use Self-Supervised Learning (SelfSL) to pretrain the IAA model such that it learns to extract more useful features for the IAA task. In the related task of *technical* Image Quality Assessment (IQA), the method RankIQA [160] pretrains a CNN to rank images based on the intensity of an applied *technical* distortion. Images of high technical quality are distorted by applying artificial technical distortions such as noise addition, blurring, or JPEG compression. Since technical distortions clearly degrade an image, the network can be trained to assess images with higher distortion intensity to be of lower quality in a self-supervised fashion. We propose to adapt and extend this self-supervised pretraining for the task of IAA. Instead of just technical distortions, we apply image filters that usually change an image’s *technical quality* (e.g., noise), *style* (e.g., contrast), or *composition* (e.g., cropping). However, applying style or composition filters to ordinary images can

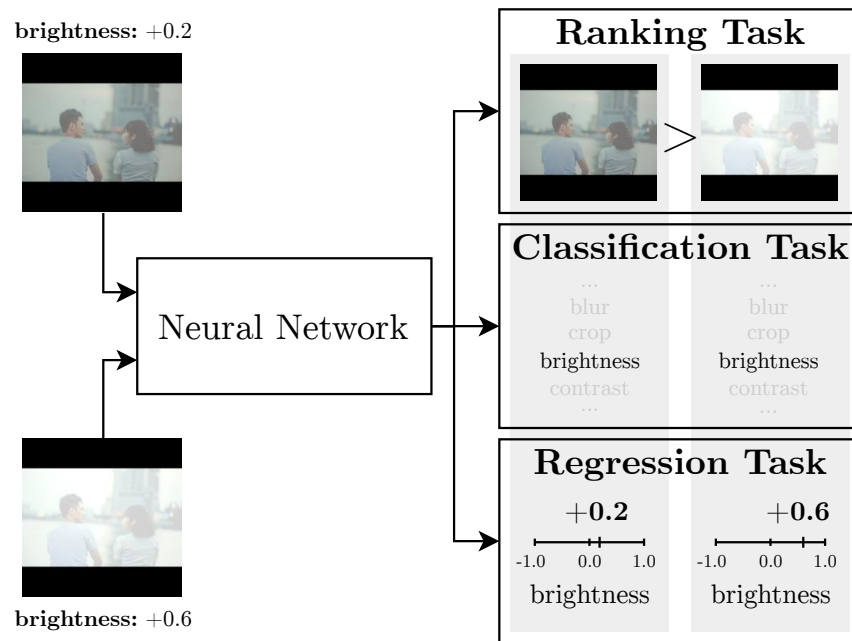


Figure 16.1.: A schematic overview of our novel pretext tasks. Images are singly distorted using image filters in different intensities. The Neural Network (NN) then learns to output higher scores for less distorted images. In a multitask setting, the network additionally classifies the distortion or estimates the distortion’s intensity.

lead to undesired improvements in the image aesthetics, violating the assumption of the self-supervised task. Thus, we introduce a large dataset of highly *aesthetic* images, consisting of the most popular images from the stock photo website `pexels.com`. Using the domain knowledge and showing that applying filters to those images usually results in less appealing images, we can use this ranking as an application of the **Weak Label Generation** principle and adopt the same ranking task for IAA. Degrading images through filters is implementing the **Input Masking and Augmentation** principle.

According to related IQA literature [117, 166, 301, 72, 315], embedding similar approaches into a multitask training setting can improve the prediction performance of the resulting image assessor by making use of additionally available information. Therefore we propose to combine this ranking task with a classification or regression task adapted for IAA: While the ranking task itself focuses on predicting relative changes in image aesthetics, the classification learns to predict the applied distortion and the regression task estimates the intensity of the currently applied distortion. Since we define a task-specific loss function to each task and an overall combination of these losses into one final loss, this method is an implementation of the **Loss Function** principle. Figure 16.1 shows a high-level overview of our proposed pretext tasks.

In our experiments, we use the IAA method Neural Image Assessment (NIMA) [265] as a reference and replace its pretrained weights with the weights from our other self-supervised pretext tasks before finetuning on the labeled AVA dataset [191]. We find that we can improve the performance over the baseline method while reducing the number of epochs until convergence by up to 47% over the ImageNet-initialized NIMA. In an analysis, we find that we match the fully supervised model’s correlation and Mean Absolute Error (MAE) metrics while requiring approximately 20% less labeled training data.

16.1. Methodology

Our method is based on the observation that singly distorting a high-quality image results in a lower-quality image, which is exploited in the *technical IQA* method RankIQA [160]. The authors apply image filters to high-quality photos, e.g., JPEG compression and noise, thus effectively degrading the technical quality of the image and then train a CNN to output higher scores for less distorted pictures. We transfer their approach to the Image Aesthetic Assessment setting by using image filters that degrade not only technical quality, but different aspects of image aesthetics. For example, changing the brightness of a well-lit photo will usually result in a less appealing image or cropping a well-composed image will certainly result in a less than optimal photo crop. To overcome the challenge that applying image filters such as contrast adjustments might enhance images, we define requirements for the dataset used in the pretext tasks. By training a Neural Network (NN) to output higher scores for less distorted versions of an image, the network has to learn what filter intensity is more appropriate between two images. We propose to also optionally add one of two other pretext tasks in a multitask setting, making use of other objectives besides the relative ranking:

1. Distortion identification guides the network to differentiate between distortions.

16. Improving IAA Models using Self-Supervised Pretraining

2. Distortion intensity estimation lets the model learn about the absolute intensity.

Our three proposed pretext tasks are depicted in Figure 16.1.

Following the common transfer learning setting, the resulting model acts as initialization for finetuning on a labeled image aesthetic dataset in a supervised fashion. We later show that our pretrained models learn more suitable features for the IAA task than the common ImageNet [46] initialization [242, 265, 167, 129].

16.1.1. Aesthetic-Aware Image Distortions

From related works [265, 243] we identify five essential aspects of general image aesthetics: technical quality, style, composition, content, and semantics. For our pretext tasks we aim to select image filters that distort aesthetic base images regarding these aspects. Since it is non-trivial to find image filters that distort image content and semantics in a self-supervised way [243], we focus on the remaining aspects *technical quality*, *style*, and *composition*, even though it is possible to include content or semantic based distortions if such image filters are found. In the following, we discuss these aspects in general, while in Section 16.2.1, we select specific image distortions.

Technical Quality Technical distortions can appear on an image under real world use cases, when taking, saving or transmitting an image, e.g., compression, blurring, or the addition of noise. It is well documented that applying such technical distortions to an image lowers its perceived quality [117, 166, 301, 315, 72, 210]. For our method, a set of distortions \mathcal{D}_{tech} influencing this aspect has to be chosen.

Image Style Image style is mostly described through image properties such as contrast, brightness, or saturation. Thus, style based distortions consist of image filters that are often used for color correction and color grading. Applying a filter from the chosen filter set \mathcal{D}_{style} to an aesthetic image results in a degradation in quality, especially when using high filter intensities.

Image Composition Image composition concerns the location of subjects and objects in the image, thus it is highly correlated with the chosen image crop. We assume that for highly aesthetic images, the original framing is selected in an aesthetically pleasing way, e.g., by following common guides such as the rule of thirds. Operations like crops or rotations then destroy such alignments. Applying these distortions \mathcal{D}_{comp} in pretraining has the additional benefit of making the network learn to recognize structures in images in general, which has been shown to be useful in similar image pretext tasks [112, 68, 77, 136].

16.1.2. Highly Aesthetic Dataset

The dataset chosen for pretraining is important for our proposed method, as the purpose of applying any distortion described in Section 16.1.1 to an image from the dataset is to degrade its aesthetics regarding the corresponding aspect. In RankIQA, high-quality

images with regard to the technical aspect are required to make sure that applying a technical distortion does in fact degrade an image’s quality [160]. The application of style and composition filters thus has to consistently degrade the associated aspect of aesthetic quality of an image from the dataset. We therefore derive two requirements for the pretraining dataset used in our method:

Highly Aesthetic The dataset needs to contain only highly aesthetic images with regard to their *technical quality*, *style*, and *composition*. This minimizes the risk that applying a filter accidentally improves the image’s aesthetics, since the undistorted image presumably already has optimal filter parameters.

Diverse in Style and Content To prevent the network from overfitting on a specific editing style or image content, the dataset needs to contain a wide variety of different images. High content diversity ensures that the model learns to generally correlate content with style features, e.g., sunsets with orange tints or portraits with natural skin tones. Consequently the dataset needs to be of sufficient size to meet the requirement regarding its diversity.

16.1.3. Self-Supervised Aesthetic-Aware Pretext Tasks

Applying the selected aesthetic-aware image distortions to the high-quality images results in a dataset containing the original, unedited images and some automatically generated lower quality image variants (regarding technical quality, style, and composition). While there is no absolute aesthetic score for neither the original image nor any of the generated images, we can access the intensity of the applied distortion and the fact that a higher intensity of an applied distortion makes the image look less aesthetically pleasing. In the following we introduce our main ranking-based self-supervised pretext task as well as two additional tasks based on classification and regression. We combine the ranking task with each of the other tasks in a multitask setting to guide the network to learn features related to image aesthetics. In our experiments, we then assess the effects of the pretext tasks on the downstream IAA performance.

Given an image \mathbf{I} and a set of distortions that unites all image filters defined for technical quality, style, and composition $\mathcal{D}_{all} = \mathcal{D}_{tech} \cup \mathcal{D}_{style} \cup \mathcal{D}_{comp}$. Each distortion $d \in \mathcal{D}_{all}$ has a set of possible intensity values $\mathcal{V}^{(d)} = \{v | v \in \mathbb{R}\}$ that can be positive or negative. Applying the filter with these distortion intensities to the image \mathbf{I} , we obtain a set of distorted images $\mathcal{I}^{(d)} = \{\mathbf{I}_v^{(d)} | v \in \mathcal{V}^{(d)}\}$. An intensity value of zero equals the original image $\mathbf{I}_0^{(d)} = \mathbf{I}$.

In general, a NN pretrained on ImageNet is used as the base model. For each task, the last layer is replaced in order to conform to the desired output.

Ranking Aesthetic Value

Our main pretext task is a ranking task based on the RankIQA method for technical image assessment [160]. Given all distorted versions of an image, the original image

$\mathbf{I}_0^{(d)}$ is assumed to be the most aesthetically pleasing and should therefore be rated with the highest score. Images $\mathbf{I}_i^{(d)}$ with a higher absolute distortion intensity value $|i| > 0$ should decrease the image aesthetics, e.g., increasing the brightness results in a less appealing image and a lower score compared to an image with a weaker increase in brightness. Given these assumptions, we utilize the ranking-based pretext task from Liu et al. [160] that ranks all images with respect to their intensity value. With our dataset and aesthetic-aware image distortions, we are able to apply this method to IAA. The NN is supposed to output scalars that are larger for higher-quality images and predict smaller values for images with larger distortion intensities. We let the last layer of the base NN output one scalar and apply a sigmoid activation function.

For one image, the ranking loss is defined as

$$L_{\text{rank}} = \sum_{\substack{\mathbf{I}_i^{(d)}, \mathbf{I}_j^{(d)} \in \mathcal{I}^{(d)}, \\ d \in \mathcal{D}_{\text{all}}, \\ |i| < |j|, \\ \text{sign}(i) = \text{sign}(j)}} \max \left(0, (f_{\text{rank}}(\mathbf{I}_i^{(d)}) - f_{\text{rank}}(\mathbf{I}_j^{(d)})) - m \right), \quad (16.1)$$

where m is the margin denoting the desired output difference between two differently distorted images when fed through the ranking network f_{rank} which returns a score between 0 and 1 for low- and high-quality images, respectively. sign is the distortion intensity’s sign. Since we can not estimate how positive or negative filter parameters relate to the image distortion, $\text{sign}(i) = \text{sign}(j)$ ensures that only image pairs with the same distortion “direction” (e.g., increasing the contrast) are compared.

Classifying Applied Distortions

In addition to the ranking task, we optionally train on a distortion identification task in a self-supervised manner as done in some No Reference Image Quality Assessment (NR-IQA) methods [117, 166, 301, 72]. We replace the network’s last layer with three separate layers, one for each quality aspect $a \in \{\text{tech}, \text{style}, \text{comp}\}$. Each layer has $|\mathcal{D}_a|$ outputs followed by a softmax activation function to output probabilities $f_{\text{class}}^a(\mathbf{I})$ for a given input image \mathbf{I} . We consider each quality aspect separately to allow cross synergies, e.g., adding noise to an image often results in lower saturation. A single probability distribution across all distortions could not consider such synergies and would increase the loss since multiple changes were correct.

The loss for this pretext task is thus defined as

$$L_{\text{class}} = \sum_{\substack{\mathbf{I}_i^{(d)} \in \mathcal{I}^{(d)}, \\ d \in \mathcal{D}_a, \\ a \in \{\text{tech}, \text{style}, \text{comp}\}}} L_{\text{CCE}} \left(f_{\text{class}}^a \left(\mathbf{I}_i^{(d)} \right), d \right), \quad (16.2)$$

where L_{CCE} is the Categorical Cross Entropy (CCE) loss function taking the output probability distribution and the index of the applied distortion.

Estimating Intensity of Applied Distortions

While the distortion identification task only classifies the distortion, another task we can add to our multitask setting is to explicitly predict the distortion intensity. The last network layer is replaced to output $|\mathcal{D}_{all}|$ values, one for each distortion. We calculate the loss by applying the squared error to the output for the applied distortion. Only calculating the error at the output index of the applied distortion makes it possible to model cross relations between distortions.

The loss function for the regression task is

$$L_{\text{regr}} = \sum_{\substack{\mathbf{I}_i^{(d)} \in \mathcal{I}^{(d)}, \\ d \in \mathcal{D}_{all}}} \left(f_{\text{regr}} \left(\mathbf{I}_i^{(d)} \right)_d - \text{norm}(i) \right)^2, \quad (16.3)$$

where $f_{\text{regr}} \left(\mathbf{I}_i^{(d)} \right)_d$ is the estimated intensity value by the regression network f_{regr} for the given distortion d and $\text{norm}(i)$ is the normalized intensity value to be predicted. The intensity value is normalized such that all non-negative intensities are normalized to the range $[0, 1]$ and all non-positive intensity values are normalized to the range $[-1, 0]$. This scales all intensity values to similar ranges, lowering the influence of single distortions on the loss.

16.1.4. Multitask Pretraining and Finetuning

Related IQA approaches [117, 166, 301, 315] have shown that a multitask training setup improves the resulting image assessor due to the additional information being available during training. For our proposed multitask setting that implements the **Loss Function** principle, we combine the ranking task with either the classification or regression task by adding the losses for the given pretext tasks using the balanced multitask learning framework by Liang and Zhang [152]. For the final loss that is used for training, it defines a function that is applied to each individual loss value. This function h needs to have certain properties to ensure good mathematical behavior: First, h should be a strictly monotonically increasing function. This ensures that large loss terms contribute more to the overall loss than small ones. Second, the derivative of h should also be strictly monotonically increasing. Given that the chain rule is used to differentiate the loss, the derivative of h can be interpreted as a weight of the individual loss term's derivative. Intuitively, this weight should also be larger for larger loss terms. Finally, h and its derivative should yield non-negative values when the input is non-negative. Given that the loss values are larger than or equal to zero, the transformed loss values as well as the derivative, i.e., the weight for the update step, should also be non-negative.

The basic approach of summing up all loss terms corresponds to using the identity function $h(x) = x$ as h . Given the requirements above we see that the derivative of the function is not strictly monotonically increasing, as it is always 1. Thus, the weight of the derivative of the loss term is constant, which does not give optimal mathematical properties. Thus, the authors propose to use $h(x) = \exp(\frac{x}{t})$, where t is a positive

hyperparameter. The exponential function is strictly monotonically increasing and its derivative $\frac{1}{t} \exp(\frac{x}{t})$ is also strictly monotonically increasing. Also, the function and its derivative are non-negative for non-negative input values (actually, for all real numbers).

We use this loss balancing scheme, since it is easy to implement. Overall, this results in the loss

$$L = h(L_{\text{rank}}) + h(L_{\text{class}}) \quad (16.4)$$

when adding classification to the ranking task and

$$L = h(L_{\text{rank}}) + h(L_{\text{regr}}) \quad (16.5)$$

when adding the regression task. We set the hyperparameter t in $h(x) = \exp(\frac{x}{t})$ to 50, according to the authors’ recommendation [152]. Each loss component is non-negative, so the final loss is also non-negative. Additionally, due to the exponential in h , even negative values would become positive.

After pretraining the model with the proposed self-supervised tasks, the network should have learned important visual features to identify and judge image aesthetics. We can finetune the pretrained model on a labeled image aesthetics dataset, which should achieve better performance on the labeled dataset.

16.2. Experiments

Now, we describe implementation details and our experimental setup in order to evaluate our methodology. This includes the acquisition of a suitable high-quality dataset, image distortions and NN architecture, training, baselines, and evaluation measures.

16.2.1. Aesthetic-Aware Image Distortions

During our self-supervised pretraining we apply *technical*, *style*, and *composition* filters to the high-quality images. Table 16.1 shows our selected distortions with corresponding intensity values.

To change the technical quality of an image, we use the library “imagenet-c” [89], introduced to benchmark the robustness of image classification models against distortions on the ImageNet [46] dataset. For changes in style we use the graphics suite darktable¹ that provides common color correction and color grading filters. All compositional distortions such as cropping, rotation, or adjusting image ratios are implemented in Python using Pillow [43]. We resize and pad all images to 224×224 pixels. This ensures that we do not accidentally destroy the image’s composition.

16.2.2. Highly Aesthetic Dataset

For our dataset, we download the 130 000 most popular images of all time² from the free stock photo website [pexels.com](https://www.pexels.com). As detailed in Section 16.1.2, the dataset needs

¹<https://www.darktable.org> (last accessed: 2023-09-26)

²[pexels.com/popular-photos/all-time/](https://www.pexels.com/popular-photos/all-time/) (scraped on 2020-02-06; now offline)

Table 16.1.: Actual parameters and implementation specifics for each distortion and intensity level. Technical parameters are passed to imagenet-c [89] and style parameters to darktable while compositional distortions are implemented by us.

	Distortion	Intensity	Actual Parameters
Technical	JPEG compression	[0, ... +4, +5]	The editing library accepts the technical intensities as is.
	Defocus blur	[0, ... +4, +5]	
	Motion blur	[0, ... +4, +5]	
	Pixelate	[0, ... +4, +5]	
	Gaussian noise	[0, ... +4, +5]	
	Impulse noise	[0, ... +4, +5]	
Style	Brightness	[-5, -4, ..., +4, +5]	[-1.0, -0.8, ..., 0.8, 1.0]
	Contrast	[-5, -4, ..., +4, +5]	[-1.0, -0.8, ..., 0.8, 1.0]
	Saturation	[-5, -4, ..., +4, +5]	[-1.0, -0.8, ..., 0.8, 1.0]
	Exposure	[-5, -4, ..., +4, +5]	[-3.0, -2.4, ..., 2.4, 3.0]
	Shadows	[-5, -4, ..., +2, +3]	[-100, -60, -20, 20, 40, 50, 60, 80,100]
	Highlights	[-3, -2, ..., +4, +5]	[-100, -80, -60, -50, -40, -20, 20, 60,100]
	Temperature	[-4, -3, ..., +4, +5]	[2000, 3000, 5000, 6000, 6500, 7000, 8000, 10 000, 14 000, 18 000]
	Tint	[-5, -4, ..., +4, +5]	[0.75, 0.8, ..., 1.2,1.25]
	Vibrance	[-2, -1, ..., +3, +4]	[0, 20, 25, 40, 60, 80, 100]
Composition	Rotation	[-5, -4, ..., +4, +5]	[10° ◯, 8° ◯, ..., 8° ◯, 10° ◯]
	Horizontal crop	[-5, -4, ..., +4, +5]	We resize the image to 336px and then crop patches of size
	Vertical crop	[-5, -4, ..., +4, +5]	224px from the resulting image. Intensity 0 is a centercrop,
	Left Diagonal crop	[-5, -4, ..., +4, +5]	while a $ intensity == 5$ results in a crop from the images'
	Right Diagonal crop	[-5, -4, ..., +4, +5]	border.
	Image Ratio	[-5, -4, ..., +4, +5]	[stretch along y-axis 100%, y80%, ..., x80%, stretch along x-axis 100%]

to contain *highly aesthetic* images of *diverse style and content* to be usable in our self-supervised pretext tasks.

Highly Aesthetic

Stock photos are inherently made to be aesthetically pleasing, which makes a stock photo website an ideal source for our dataset. It can safely be assumed that stock photos in general are quite well lit and color coordinated. By taking only the most popular photos, we are certain that these images are liked by the general public. Figure 16.2 shows randomly selected images from the dataset. To spot-check the aesthetics, we conduct a survey: A random image from the dataset and the same image with an image filter applied are shown. The subject clicks on the image that they find to be more aesthetically pleasing. 73 annotators have rated 6182 image pairs. For 74% of these pairs, the unedited image is preferred, when compared to *any* style filter in *any* intensity. Besides the survey, we apply a trained version of NIMA³ [265] to the original images, which classifies approximately 88% of them as high-quality.

Diverse in Style and Content

According to pexels.com, the stock photos are from a wide variety of art styles and topics. Available categories range from dominant colors over common photo tags like

³<https://github.com/kentsyx/Neural-Image-Assessment> (last accessed: 2023-02-10)

16. Improving IAA Models using Self-Supervised Pretraining

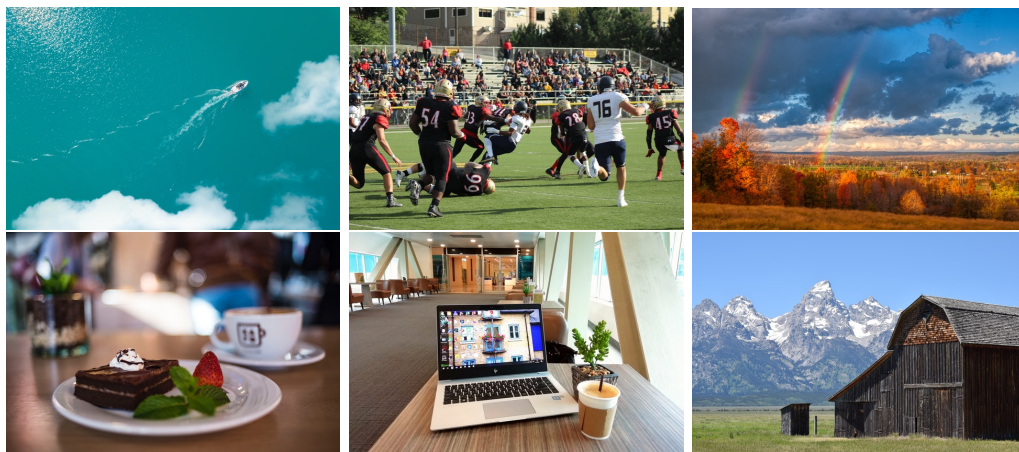


Figure 16.2.: Random examples from our unlabeled dataset from pexels.com used for our self-supervised pretraining.

food, fashion, or people to emotional aspects like *moody, wellbeing, or happy*. This induces a high diversity in image contents and styles. Assuming that the most popular images are a somewhat representative sample of all images, our dataset covers a wide range of different styles while each individual image remains well edited. Additionally, we apply a pretrained DenseNet121 [100] for image classification and RetinaNet [154] for object detection to our dataset to make sure that the images are diverse in content. We find that the images of our dataset spread across many different classes and contain a wide variety of objects and subjects, as can be seen in Table 16.2.

Table 16.2.: Most and least commonly detected classes and objects in the images of our dataset.

most common classes		most common objects	
class	count	object	count
seashore	3554	person	44301
alp	2568	car	3186
lakeside	2446	cup	2880
fountain	2265	bird	2788
valley	2011	cell phone	1749
miniskirt	1455	boat	1618
gown	1430	dog	1581
bikini	1176	potted plant	1580
...
sloth_bear	2	refrigerator	31
affenpinscher	2	snowboard	25
patas	1	skis	23
Sealyham_terrier	1	hair dryer	7
Japanese_spaniel	1	toaster	7

16.2.3. Self-Supervised Aesthetic-Aware Pretraining

As the main pretext task, we train the network to rank an image with differently intense filter settings for one filter. In addition, we add one of two tasks, as described in Section 16.1.4, resulting in three pretext task combinations: *ranking*, *ranking+classification*, and *ranking+regression*. We set the hyperparameter m , which describes the desired minimal margin between two images with different distortion intensities, to 0.2 in our experiments.

Prior to training, we randomly split our dataset into 100 000 training images, 15 000 validation, and 15 000 test images. Every image is then distorted using each of the filters described above with all respective intensity values, resulting in 173 different variants per image, including the original image. Overall, this makes $100000 \times 173 = 17300000$ training images and $15000 \times 172 = 2595000$ images for validation and testing.

As the NN architecture, we use MobileNetV2 [236], since it speeds up training times due to fewer parameters while still performing well [265]. This allows us to train comparatively quickly even on the approximately 17 million training images.

For each pretext single-task or multitask setting, we train the model for 20 epochs and then select the epoch with the lowest validation loss as the initialization for finetuning. We select an initial learning rate of 10^{-4} for all model configurations based on a hyperparameter search. All other training settings are taken directly from the original MobileNetV2 paper [236], i.e., optimizing using RMSProp [91] with decay and momentum set to 0.9, weight decay regularization of 4×10^{-5} , and an exponential learning rate decay of 0.98 per epoch. As batch size, we take eight images and all their distorted variants as a batch, thus resulting in 1384 images in each batch.

16.2.4. Baselines

We compare our method to three other models. On the one hand, we train a model that is initialized with weights trained on the ImageNet classification task [46], a common initialization for IAA methods [242, 265, 167, 129]. On the other hand, we employ two common self-supervised pretext tasks. **RotNet** [68] classifies the rotation $\in \{0^\circ, 90^\circ, 180^\circ, 270^\circ\}$ applied to an input image. **SimCLR** [38] aims to output similar representations for different augmentations of the same image. To allow for a fair comparison, we use the same MobileNetV2 [236] architecture for all methods. Each model is initialized with ImageNet weights. Additionally, all algorithms are applied to our collected highly aesthetic dataset to make sure that all methods have access to the same images while pretraining.

RotNet For RotNet [68], we use the authors' implementation⁴. We follow the same procedure as in the original paper and only change the given network architecture and dataset.

⁴<https://github.com/gidariss/FeatureLearningRotNet> (last accessed: 2023-02-10)

Table 16.3.: Performance results: Shown are the Accuracy on the binary task, Pearson and Spearman correlations, as well as the Mean Absolute Error between the ground truth and the scores returned by our model. The best value for each metric is printed in bold. For our pretext tasks, † indicates significantly better results compared to the ImageNet pretext task.

Pretext Task	Epochs	AVA				TID2013		
		Pearson ↑	Spearman ↑	Accuracy ↑	MAE ↓	Pearson ↑	Spearman ↑	MAE ↓
ImageNet [46]	147	0.5491	0.5372	0.7548	0.4716	0.4760	0.3897	0.9747
RotNet [68]	79	0.3272	0.3139	0.7112	0.5376	0.1282	0.1075	1.1334
SimCLR [38]	65	0.5483	0.5360	0.7540	0.4718	0.4824	0.3944	0.9688
ranking	96	0.5536	0.5414	0.7560	0.4698†	0.4842	0.3946	0.9679†
ranking + classification	89	0.5535	0.5409	0.7550	0.4702	0.5009	0.4111	0.9620†
ranking + regression	77	0.5541	0.5420	0.7582	0.4697†	0.4855	0.3971	0.9672†

SimCLR For this baseline, we use an unofficial reimplementation⁵ that is able to reproduce the results from the paper [38]. The code provides the choice between the Adam optimizer and the LARS optimizer [38]. For our experiments we select the latter with a cosine annealing learning rate schedule as in the original paper. We are therefore positive that this is comparable to the original implementation.

16.2.5. Finetuning Pretrained Models

After pretraining, each model is finetuned on the AVA dataset [191]. Note that our pretrained models are drop-in replacements for all IAA methods that initialize their weights with a pretrained CNN. For our experiments, we follow the training method from NIMA [265], since it is simple and elegant: We replace the last layer of the network with a fully connected layer with ten outputs, indicating the possible AVA scores one to ten. Outputs are normalized to a distribution using the softmax function. The training procedure then uses the Earth Mover’s Distance Loss [95] on the rating distribution of the annotator’s votes for each image.

Training parameters are kept close to the original values proposed by Talebi and Milanfar [265], only incorporating small practical improvements by Lennan et al. [148] and changes to adapt for the different pretext tasks: We train using the Adam [127] optimizer with separate learning rates for the pretrained layers (10^{-4}) and for the new fully connected layer (10^{-3}) and a weight decay regularization of 4×10^{-5} . Additionally, we use a learning rate schedule that monitors the validation loss and halves the learning rate if the loss does not significantly improve for five consecutive epochs. This schedule is a compromise between the originally proposed schedule in NIMA [265] and the more aggressive decline in learning rate used by Lennan et al. [148]. As suggested by Lennan et al. we keep the weights of all layers but the very last frozen until the validation loss plateaus for the first time. We train on the AVA dataset and employ an early stopping strategy that stops training once the validation loss does no longer improve for ten consecutive epochs.

⁵<https://github.com/Spijkervet/SimCLR> (last accessed: 2023-02-10)

16.2.6. Evaluation Setup

For evaluating the trained models, we mostly follow the setup by Talebi and Milanfar [265]. Given the ten-dimensional output denoting a distribution across the possible rating scores, a mean score can be computed by calculating the expected score given the output distribution. We let all models predict the mean aesthetic scores for the test split of the AVA dataset and label images with scores above five as aesthetic and below five as not aesthetic. As in our other IAA works in this thesis, we calculate the Accuracy [235] and the Spearman [235] and Pearson [63] correlation coefficients. Additionally, we compute the MAE [235] between the predicted and ground truth mean scores.

In addition to AVA, we, like Talebi and Milanfar [265], also test our models on the TID2013 [210] dataset that is designed to measure the performance of technical IQA models. Since our pretext tasks incorporate distortions leading to technically degraded images, we suspect that the performance of pretrained models improves on this dataset as well. We calculate the Spearman correlation, Pearson correlation, and MAE between the predicted and ground truth scores from the dataset.

16.3. Results

In Table 16.3, the finetuned models are identified on the basis of their respective pretext task they were initialized with. In addition to the evaluation metrics specified in Section 16.2.6, we also provide the number of training epochs before halted by early stopping.

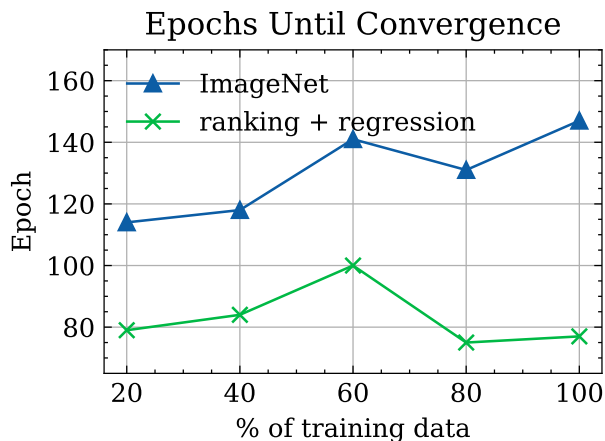
Models pretrained on any of our proposed pretext tasks show better evaluation metrics than all baseline models. Compared to the ImageNet baseline, any additional pretraining reduces the number of epochs during finetuning by 29% to 55%, while only our pretext tasks consistently improve the performance of the resulting model at the same time.

Adding classification to the ranking task in a multitask setting does not improve the performance of the ranking task on AVA, however, results in the best model evaluated on the TID2013 dataset. This supports the findings of previous work that used classification pretext tasks for technical image quality assessment [117, 166, 301, 72]. We suspect that this is due to the difficulty of distinguishing some categories of distortions. In the classification task, we categorize all image filters into the categories *technical*, *style*, and *composition* and train the network to identify the distortion in the corresponding category. We observe that the classification accuracy per category of the pretrained model on the test set of our collected stock photo dataset is higher for technical distortions than for style or composition changes (see Table 16.4). The pretrained model has learned to extract features that are especially useful for technical image distortions, making the model more suitable to be used as an initialization for IQA.

In contrast to classification, adding the regression task yields the best performance metrics on AVA in our experiments. Our intuition is that this is due to two effects. First, the loss only takes exactly one distortion into account, thus making all distortions independent of each other. For the classification task, we have to combine multiple distortions based on their type in order to be able to calculate the CCE loss, making their

Table 16.4.: Prediction accuracy for the correct distortion by the classification layer. Included is a random baseline guessing a random distortion.

Accuracy	technical	style	composition
ranking+classification	0.445	0.127	0.257
random	0.167	0.111	0.167

**Figure 16.3.:** Epoch in which early stopping occurred by dataset size.

outputs dependent on each other. Second, and more importantly, letting the network predict an intensity per distortion encourages the extraction of features that already have a notion of order. These features are then effectively used during finetuning and result in better performance. In fact, Bonferroni-corrected [84] one-sided Wilcoxon signed rank tests [293, 294] on the MAE show that using our pretext tasks over ImageNet produces *significantly better* results at an α level of 1%. This means, on average, the returned score is significantly closer to the mean human annotated score than when using the ImageNet initialization.

The self-supervised baselines RotNet [68] and SimCLR [38] are explicitly designed to learn content based features in order to improve image classification performance. Thus, similar to the ImageNet pretraining, learned features are influenced more by the image’s content than its style, which is a disadvantage in the IAA task. When finetuning the pretrained models, these style features need to be relearned in order to solve the task. Since our method explicitly embraces these features during pretraining, this leads to more useful features to start with and better results in the downstream task.

16.4. Analysis

Now that we know that our pretraining improves performance, we conduct an analysis of the capabilities of the pretrained models.

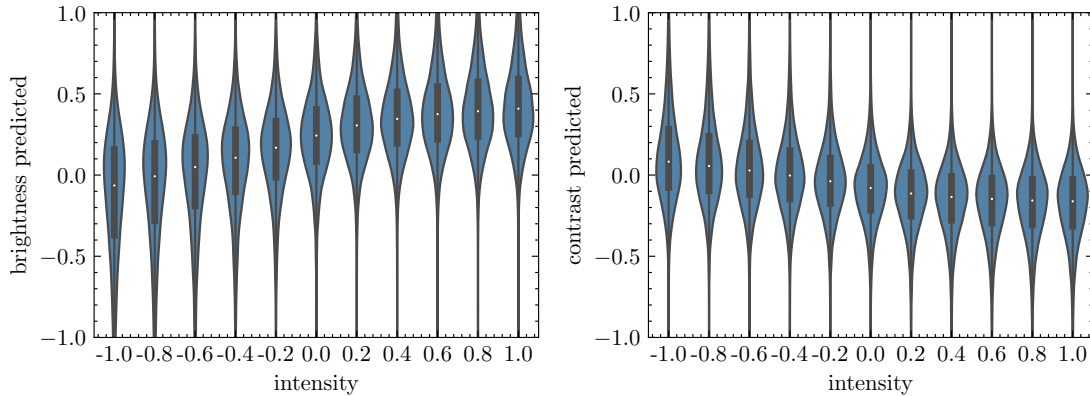


Figure 16.4.: Outputs of the brightness (left) and contrast (right) neuron for different intensities (x-axis) of the brightness distortion.

Aesthetic Pretraining Learns Cross-Distortion Relations In the following we analyze the outputs of the pretext regression network f_{regr} . Note that we study the pretrained model that was *not* yet finetuned on the AVA dataset. During pretraining, the regression loss is only computed on the corresponding output dimension denoting the currently applied distortion, as described in Section 16.1.3. All other outputs are not used for loss calculation, thus can independently predict intensities for their respective distortions. This allows the network to learn synergies between different distortions, e.g., an increase in brightness usually reduces the contrast (see Figure 16.1) and vice versa.

To check if the model has learned these relations, we feed all test images of our collected dataset with all brightness changes through the network. Then, we plot the model’s predicted brightness and contrast changes as a violin plot in Figure 16.4. Given higher or lower brightness intensities, the predicted outputs are also increasing or decreasing, respectively. An increase in brightness also results in a decrease in contrast, which shows the model learned cross-distortion relations, which it was never explicitly trained on.

Aesthetic-Aware Pretrained Models Need Less Labeled Data According to our results, models pretrained on our proposed tasks were able to learn useful image features for IAA, resulting in significantly improved prediction performance and training time. We hypothesize that this also reduces the need for labeled training images in order to achieve comparable performance to the ImageNet baseline. To verify this, we finetune the *ranking+regression* pretrained model and the ImageNet baseline on subsets of the AVA training data: 20%, 40%, 60%, 80%, 100%. Figure 16.5 shows the Accuracy, Pearson and Spearman Correlation Coefficients, as well as the MAE on the AVA test set for both models. While our model seems to match the baseline’s accuracy on the binary task, it consistently outperforms the baseline on both correlation coefficients and the MAE. Due to the higher correlations, we can assume that our model matches the relative aesthetic order of images better than the baseline. Furthermore, we find that we match the ImageNet model in terms of correlation and MAE while needing approximately 20%

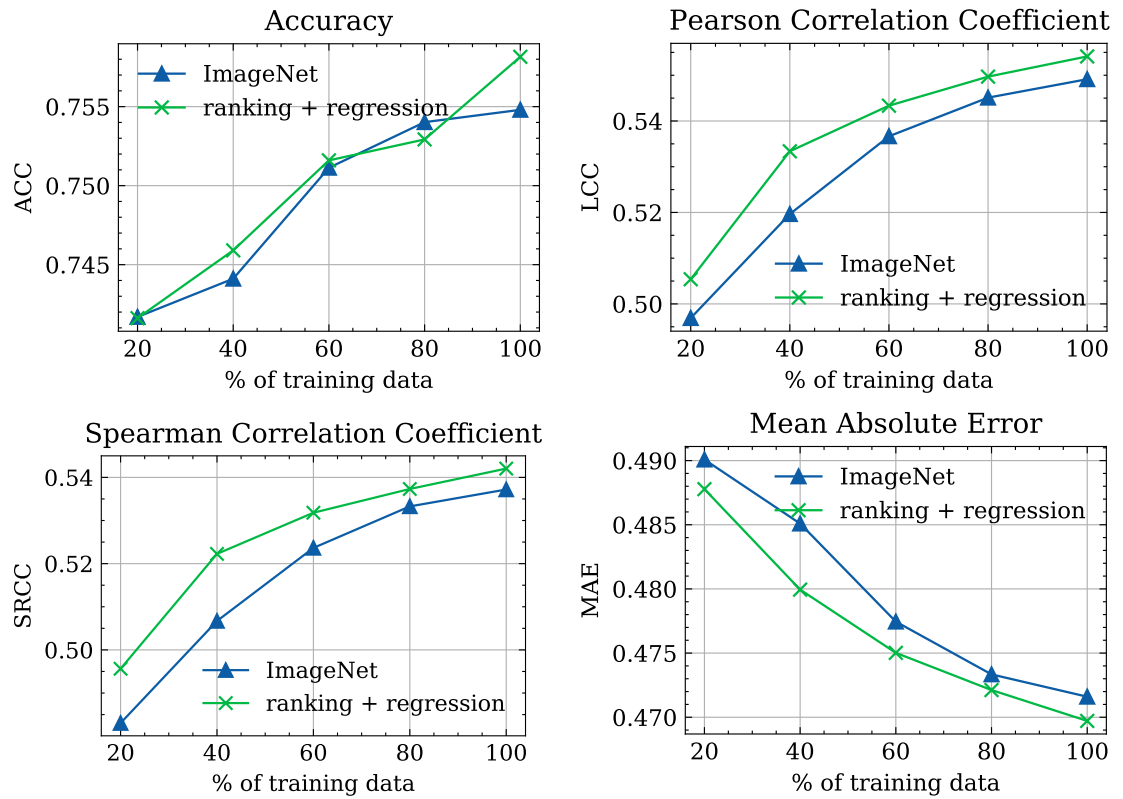


Figure 16.5.: Accuracy (ACC), Pearson (LCC)/Spearman (SRCC) correlation coefficient and MAE for different sizes of the labeled dataset.

less training data. Our pretrained model also converges faster than the ImageNet baseline across all training data sizes, as shown in Figure 16.3. Overall, pretraining the model on our proposed tasks thus provides a better initialization for finetuning than the ImageNet classification task.

16.5. Conclusion

In our self-supervised pretraining, our models do not learn to rank different images against each other but only to differentiate between distortions and intensities applied to one image. Our models are therefore not able to learn some potentially relevant aesthetic features and relations between different images during pretraining. To furthermore enable our models to e.g., rank different base images during the pretext task, we suggest exploring additional losses: A loss minimizing the distance between scores of two images of the same distortion and same intensity could be introduced. Additionally, we can extend the ranking loss and not only rank distortions of the same image against each other, but against other base images as well. Under the assumption that all images in the dataset are roughly equally aesthetic, this allows our network to learn relationships between aesthetic features of different base images.

We have proposed self-supervised aesthetic-aware pretext tasks optimized for finetuning IAA models. For this, we have introduced a large dataset of highly aesthetic images that were systematically degraded in quality using distortions in three aspects of image aesthetics: technical quality, image style, and composition. We have applied the pretext tasks in a multitask setting and have shown that ranking as well as estimating distortion intensities improves performance over the employed baselines and converges faster than starting with ImageNet initialization. An analysis has shown that our pretext tasks are able to teach the NN meaningful and relevant features about image aesthetics, without access to an explicit human opinion as reference. This strengthens the evidence for our hypothesis from Chapter 12 that pretrained models that extract more suitable features are needed for the task of IAA.

17. Conclusion

Based on the results presented in this thesis, we can now discuss the effectiveness of the proposed methods and principles. The goal of this thesis was to explore, investigate, and apply different methods to understand and improve Deep Learning (DL) models in a model-agnostic way by using domain knowledge, as discussed in Chapter 1. We have introduced six principles that formulate potential ways to incorporate domain knowledge into the analysis and training routines of Neural Networks (NNs) in different components of the common DL training framework. We have tested these principles by implementing them for different applications. Our numerous experiments have shown that the application of the discussed principles indeed help with understanding and improving DL models. We can thus conclude in this thesis that considering the provided principles can be a beneficial route to take when developing DL systems. Implementing the understanding based principles gives us insights into how the model reaches a conclusion. Implementing the improving principles alone or in combination can increase the model’s performance, its robustness, reduce training data requirements, or let it converge faster than before.

While all principles are independent of each other, principles aiming for understanding the model can help us to identify the biggest flaws in a model. This can facilitate the search for how we can improve the model or which improving principle is most likely to achieve the desired results. For example, the **Generated Input Data** principle might be applied to an image model. The results indicate that the model is prone to background bias (as shown in Chapters 7, 9 and 11 for Deep Metric Learning (DML) models). We can then use this insight to make a fitting choice for improving the model, for example by applying the **Input Masking and Augmentation** principle (done in Chapter 11). This thesis can thus serve as inspiration and as a systematic reference for DL practitioners who want to understand and improve their models in a model-agnostic way. The key for each of the considered principles to work is the domain knowledge that is necessary to better understand and improve the model. While NNs are universal function approximators [93], the use of additional domain knowledge allows for better control of the training routine. For this, inductive biases are defined, which steer the model training to a certain direction based on the domain knowledge. These inductive biases can be incorporated at every step of the typical DL training framework and lead to the desired improvements of the model, but need to be chosen carefully. We thus now discuss the effectiveness and limitations of each of our principles. Afterwards, we give an outlook on what future research directions might be for the topic of model-agnostic DL with domain knowledge.

17.1. Principle Discussion

For each of the six principles we explore, we have provided at least two different implementations that make use of this principle. Given the results of our exploration, we can now discuss the possibilities and limitations of the principles.

17.1.1. Generated Input Data

Knowing the influence of different properties on a DL model's output and how it changes with different variations of the property can give insights into the model's behavior and how it might be improved. For example, our analysis of the Land Use Regression (LUR) model provides estimates of how an increase in street width might influence the air quality. Such insights can be incorporated into simpler models such as Linear Regression (LR). Our investigation of DML models using car images has shown that these models can be sensitive to properties that should not be important for the embedding process. This can be used to improve the model by for example designing property-specific data augmentations that we have presented in Chapter 11.

One thing to consider is that generated data compromises on the generality of the analysis results. DL models are highly non-linear models that combine low-level features in sometimes non-obvious ways. Generating fake data then always abstracts away from the real data, which could discard other influencing factors that interfere with the property of interest. The sheer number of properties that can be changed in the data can also be overwhelming, which means that it might not be possible to analyze all properties and how they interact with each other. For instance, while we have investigated the effect of the street's position and its width on the air quality estimates separately, we have not investigated how the two properties interact with each other. Due to the number of possible property characteristics, a certain abstraction level has to be reached when generating fake data. For example, the fake car images for the DML model only have solid background colors, which is not a realistic property of car images.

On the other hand, using realistic backgrounds would result in a huge number of possible backgrounds, which would make the generation of fake data infeasible. Also, the influence of certain background patterns might influence the model in non-obvious ways. This may need to be investigated in a different setting.

On a related note, the change of high-level properties can influence low-level and other high-level features. For example, the change in the background color can show in reflections of the car, which indirectly influences the car color. This can lead to unintended effects, which is why it is important to carefully craft the generation process and find the right abstraction level. In the case of the background color, for example, we generate the image such that the background does not interact with the car, leading to no car color changes when changing the background.

Despite all these limitations and considerations to take into account, the Generated Input Data principle is very powerful, since it can be applied to all tasks and most modalities, as long as they are fakable in a controllable way. This requires a deep domain knowledge but makes it a very useful tool for understanding the influence of certain

properties on DL models, which in turn can be used to improve them.

17.1.2. Gradient-Based Attribution

The Gradient-Based Attribution principle is a model-agnostic principle that can be applied to all DL models that are differentiable. In comparison to the Generated Input Data principle, it highlights low-level features such as pixels or words. This can make it sometimes more difficult to interpret, since a human observer has to look at the generated attribution maps, the input, and use domain knowledge to find a connection between them. Also, methods implementing this principle can by default only work on single inputs, leading to only anecdotal evidence and qualitative results. This makes it difficult to make more broad statements about the model, which would require looking at and interpreting a large amount of examples. In Chapter 8, we provide anecdotal evidence that our proposed venue recommendation model Where to Submit (WTS) indeed relies on useful words and phrases present in the publication’s title and abstract. This might be a useful tool to understand why the model chose a certain venue for the given input, but does not directly provide high-level features the model learned. Thus, in Chapter 9, we compare different models to each other, which includes the idea that different models should usually learn similar features for the same images. This way, the anecdotal evidence can be turned into more general statements about the differences between models, without using any other information about the images.

Incorporating additional knowledge about the input data allows for more general statements about the model by the attribution method. For example, computing statistics for the attribution given to specific words compared to other words in a text can help to understand what general words and phrases seem to be more important for the model’s output. We do something similar for DML models in Chapter 11, where we use separation masks of images to compute the influence of foreground and background elements on the model’s output. Overall, the Gradient-Based Attribution principle can be used to debug DL models, which helps with finding ideas to improve them.

17.1.3. Input Masking and Augmentation

The Input Masking and Augmentation principle aims to guide the model to ignore unwanted input features when producing an output. Our concrete implementation in Chapter 11 shows that it is possible to decorrelate the background from the foreground of an image by applying background augmentations. This results in better robustness of models regarding background changes, which, in the case of item retrieval, is a desirable property. We have shown that the higher robustness regarding background changes due to the BGAugment method leads to a drop in performance on the clean test dataset for the Stanford Online Products (SOP) dataset. However, the performance on the test dataset with background changes increases significantly. We suspect that this comes from the fact that the good performance on the clean dataset can only be achieved by learning correlations between the background and the foreground of the image. Then, this is a deceptive result, since this is not the goal of the task. As with the provided results, it is

17. Conclusion

also possible to use augmentations and masking to better understand the performance of DL models. We have created a new test setting that shows that DL models are prone to background bias, which is based on augmenting the background of the images. This is similar to the Generated Input Data principle, where one aspect of the input is changed. However, in this case, real data is used and transformed to make statements about the influence of the “high-level feature” *background* on the model’s output.

While decorrelating certain input features from the model’s output, we also designed a scenario where a new correlation is introduced using this principle. In Chapter 16, we augmented aesthetic images in order to generate less aesthetic ones. This introduced the correlation of certain style changes to the desired output of the model: The more we destroy an aesthetic image with image filters, the less aesthetic the image becomes.

In general, the creation of transformations and masking procedures needs to be done carefully in order to achieve the desired goal. It is certainly task- and modality-specific, which draws parallels to the feature engineering process for other Machine Learning (ML) models that do not directly work on raw data. Here, features are engineered from the data in order to remove unimportant correlations that might influence the model’s output. With data augmentations and masking, the same goal is reached but for DL models. This way, the model can still work on low-level data, but task- and modality-specific semantic knowledge can be captured in the data preparation step.

17.1.4. Feature Extraction using Pretrained Multimodal Models

Our experiments have shown that using pretrained multimodal models can be a good way to solve tasks that do not have enough training data. As we have seen, zero-shot methods are capable of achieving good results on different tasks. However, they can often not achieve the same performance as methods that are trained on the task-specific data. Using the features from the pretrained model to train a new model on top of them shows better performance. However, it does not reach the performance of a fully finetuned model.

There certainly is a trade-off that has to be made. On the one side, there is computational complexity and data need, on the other side performance. Zero-shot methods do not need any training data, and they eliminate the need for a trainable model in the DL training pipeline. Thus, no training data needs to be collected, and the model can usually be run on less powerful hardware, since only the inference is performed, which can be an advantage in compute- and data-constrained environments. However, zero-shot methods in this setting often require an input that guides the model to solve the task, e.g., a text prompt in language models. The choice of these prompts can be difficult, since models are often very sensitive to certain word changes.

Training a small trainable model on top of the multimodal model’s features, such as a LR, can improve the performance over zero-shot approaches, but needs task-specific training data and requires the optimization of a model. This is computationally more expensive than Zero-Shot Learning (ZSL) with pretrained models, but usually has still fewer hardware requirements than full finetuning, since the extraction of features for a training dataset needs to be done only once. Due to the small number of parameters, the

model can be trained relatively quickly and with relatively few datapoints.

The full finetuning of the multimodal model usually achieves the best performance, but is also the most expensive approach. The model cannot be reused for other tasks anymore since it is finetuned specifically for the task at hand. When we aim to improve DL models using feature extraction from pretrained multimodal models, we mean to reduce the computational complexity and the data needs of the trainable model. We also achieve a certain reusability of the multimodal model: When using Contrastive Language-Image Pre-Training (CLIP) for Image Aesthetics Assessment (IAA) by using prompting, linear probing, or for DML using text prompts. The original parameters of CLIP stay untouched, which allows us to use it for other tasks such as content search or item retrieval. This can reduce memory requirements in practical applications, since not multiple model instances need to be initialized with different weights. Also, oftentimes, the feature vectors can be precomputed and then used for different tasks without much computational overhead. Given all these improvements, we can still achieve good performance on the task at hand, even though it might not be as good as the performance of a model that is trained on the task-specific data.

17.1.5. Weak Label Generation

Weak labels are a simple way to enlarge the training data. However, they require additional knowledge about the task and data which is then captured by the heuristic or the model with stronger assumptions. The improvement that is achieved by this principle is a better performance of the DL model. This comes with larger training data and thus larger computational complexity, since the model needs to be trained on more data. Because DL models are usually more powerful than the weak labeling model, the final trained DL model usually has better performance than the weak labeling model. However, this depends on the weak labeling strategy and the complexity of the task. If the weak label generation is too simple, the model will not learn anything useful from it. If the weak label generation already gives very good results, the computational overhead to train a NN on the weak labels might not be worth it. In the sentiment analysis task, the lexicon based approach already shows good results, so the NN does not add much performance gain. The DL model, however, is able to find other correlations than the weak label generation process between the input and the output, leading to a better generalization. The use of weak labels thus is a trade-off between the performance gain and the computational overhead.

17.1.6. Loss Function

The loss function is one of the most important choices to make in a DL training pipeline. Since it is only used during training and does not need to be evaluated during inference, the exchange of a loss function for another one does not affect the inference times of the model. Based on the goal that the model should achieve, the loss function can often be chosen from a wide variety of options. The advantage of the loss function is that it provides a simple interface that receives the predictions of the model and the corresponding ground

17. Conclusion

truths and returns a single value. Apart from required differentiability, there are basically no restrictions on what happens inside the loss function. The use of multiple loss functions in a multitask setting for the pretraining of an IAA model in Chapter 16 has shown that with the right choice of loss function, the model can be steered towards learning beneficial features for a downstream task. However, the side effects of the loss function on the learned features or model outputs are not directly visible. As we have shown for our proposed Similarity Based Loss (SimLoss) in Chapter 14, the trained model predicts more similar classes when making mistakes, which is certainly desired. On the other hand, the model seems to reduce its possible output classes to representatives that have, on average, good similarity scores to the ground truth class. This behavior might not be desired in some circumstances. The introduction of a hyperparameter in SimLoss that balances the effect of the similarity matrix and the usual Categorical Cross Entropy (CCE) loss function can alleviate this problem, which shows the trade-off between the performance on certain metrics and the desired behavior of the model.

17.2. Outlook

We have seen that the explored principles can, when implemented for specific tasks and modalities, lead to a better understanding and improvement of DL models. All methods we have demonstrated are model-agnostic and can be applied to any DL model. The results shown in this thesis indicate that modifying components of the DL training setup other than the trainable model can lead to multiple model improvements and is thus a promising starting point for better DL. The selection of the right principle heavily depends on the type of improvement one wants to achieve. Overall, we promote the use of such principles as a complement to the exploration of a better model architecture. The goal should be to model as much domain knowledge into the training process and model architecture as possible. This starts with the choosing a suitable model architecture. Then, evaluating which understanding principles can be applied to the task and data at hand can lead to a better understanding of and new insights about the model behavior. Finally, the obtained knowledge can be used by one or multiple of our explored principles to improve the model.

In this thesis, we have shown that the model can be better understood and improved by using the explored principles. We have successfully done this for different tasks and different modalities, which shows the broad applicability of these principles. Combining multiple principles and specific techniques can lead to even better results, as we have shown. In the future, the combination of multiple independent principles without necessity and their joint effects on the model are worth exploring. As different principles apply to different components of the training setting, multiple principles can be implemented independently. To simplify the adoption of these principles for practitioners, a software library with well-tested implementations of the discussed principles is another more practical direction for future work. With a growing set of supported applications and tasks, the broader impact of certain principles can be assessed, leading to a better understanding of the requirements for tasks to benefit from a principle. We hope that

this thesis with its initial application of principles for different tasks and modalities will inspire others to explore the use of these principles and techniques in their own work. Also, we hope that this thesis will encourage others to explore and invent new principles and implementations in order to make DL models better in a model-agnostic way using domain knowledge.

Bibliography

- [1] A. Abbas and S. Deny. Progress and Limitations of Deep Networks to Recognize Objects in Unusual Poses. *arXiv:2207.08034 [cs]*, July 2022.
- [2] M. Adams. *Advancing the Use of Mobile Monitoring Data for Air Pollution Modelling*. PhD thesis, McMaster University, Hamilton, 2015.
- [3] Air Quality Team (Greater London Authority). London Atmospheric Emissions Inventory (LAEI), 2013. URL <https://data.london.gov.uk/dataset/london-atmospheric-emissions-inventory-2013>. Last accessed: 2023-09-29.
- [4] M. S. Alam and A. McNabola. Exploring the Modeling of Spatiotemporal Variations in Ambient Air Pollution within the Land Use Regression Framework: Estimation of PM10 Concentrations on a Daily Basis. *Journal of the Air & Waste Management Association*, 65(5):628–640, 2015.
- [5] M. A. Alcorn, Q. Li, Z. Gong, C. Wang, L. Mai, W.-S. Ku, and A. Nguyen. Strike (With) a Pose: Neural Networks Are Easily Fooled by Strange Poses of Familiar Objects. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 4840–4849, Long Beach, CA, USA, June 2019. IEEE. ISBN 978-1-72813-293-8. doi: 10.1109/CVPR.2019.00498.
- [6] B. Alshemali and J. Kalita. Improving the Reliability of Deep Neural Networks in NLP: A Review. *Knowledge-Based Systems*, 191:105210, Mar. 2020. ISSN 09507051. doi: 10.1016/j.knosys.2019.105210.
- [7] E. Amid and M. K. Warmuth. TriMap: Large-scale Dimensionality Reduction Using Triplets. *arXiv:1910.00204 [cs, stat]*, Mar. 2022. doi: 10.48550/arXiv.1910.00204.
- [8] W. Ammar, D. Groeneveld, C. Bhagavatula, I. Beltagy, M. Crawford, D. Downey, J. Dunkelberger, A. Elgohary, S. Feldman, V. Ha, R. Kinney, S. Kohlmeier, K. Lo, T. Murray, H.-H. Ooi, M. E. Peters, J. Power, S. Skjonsberg, L. L. Wang, C. Wilhelm, Z. Yuan, M. van Zuylen, and O. Etzioni. Construction of the Literature Graph in Semantic Scholar. *arXiv:1805.02262 [cs]*, 2018.
- [9] M. Ancona, E. Ceolini, C. Öztireli, and M. Gross. Gradient-Based Attribution Methods. In W. Samek, G. Montavon, A. Vedaldi, L. K. Hansen, and K.-R. Müller, editors, *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, volume 11700, pages 169–191. Springer International Publishing, Cham, 2019. ISBN 978-3-030-28953-9 978-3-030-28954-6. doi: 10.1007/978-3-030-28954-6_9.

Bibliography

- [10] R. Angulu, J. R. Tapamo, and A. O. Adewumi. Age Estimation via Face Images: A Survey. *EURASIP Journal on Image and Video Processing*, 2018(1):42, June 2018. ISSN 1687-5281. doi: 10.1186/s13640-018-0278-6.
- [11] A. Azulay and Y. Weiss. Why do Deep Convolutional Networks Generalize so Poorly to Small Image Transformations? *Journal of Machine Learning Research*, 20(184):1–25, 2019. ISSN 1533-7928.
- [12] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer Normalization. *arXiv:1607.06450 [cs, stat]*, July 2016.
- [13] H. Bahng, A. Jahanian, S. Sankaranarayanan, and P. Isola. Exploring Visual Prompts for Adapting Large-Scale Models. *arXiv:2203.17274 [cs]*, June 2022.
- [14] Y. Bai, Y. Li, B. Zeng, C. Li, and J. Zhang. Hourly PM2.5 Concentration Forecast using Stacked Autoencoder Model with Emphasis on Seasonality. *Journal of Cleaner Production*, 224:739–750, 2019.
- [15] Y. Bai, B. Zeng, C. Li, and J. Zhang. An Ensemble Long Short-Term Memory Neural Network for Hourly PM2.5 Concentration Forecasting. *Chemosphere*, 222: 286–294, 2019.
- [16] A. Baldrati, M. Bertini, T. Uricchio, and A. Del Bimbo. Conditioned Image Retrieval for Fashion using Contrastive Learning and CLIP-based Features. In *ACM Multimedia Asia*, pages 1–5, 2021.
- [17] R. Balestrieri, I. Misra, and Y. LeCun. A Data-Augmentation Is Worth A Thousand Samples: Analytical Moments And Sampling-Free Training. *Advances in Neural Information Processing Systems*, 35:19631–19644, 2022.
- [18] F. Barbieri, L. Espinosa Anke, M. Ballesteros, J. Soler, and H. Saggion. Towards the Understanding of Gaming Audiences by Modeling Twitch Emotes. In *Workshop on Noisy User-Generated Text*, pages 11–20, Copenhagen, Denmark, 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-4402.
- [19] M. J. Barone, K. S. Coulter, and X. Li. The Upside of Down: Presenting a Price in a Low or High Location Influences How Consumers Evaluate It. *Journal of Retailing*, 96(3):397–410, Sept. 2020. ISSN 00224359. doi: 10.1016/j.jretai.2020.02.003.
- [20] B. Barz and J. Denzler. Hierarchy-Based Image Embeddings for Semantic Image Retrieval. In *IEEE Winter Conference on Applications of Computer Vision*, page 10, 2019.
- [21] B. Barz and J. Denzler. Deep Learning on Small Datasets without Pre-Training using Cosine Loss. In *IEEE Winter Conference on Applications of Computer Vision*, pages 1360–1369, Snowmass Village, CO, USA, Mar. 2020. IEEE. ISBN 978-1-72816-553-0. doi: 10.1109/WACV45572.2020.9093286.

- [22] R. Beelen, G. Hoek, D. Vienneau, M. Eeftens, K. Dimakopoulou, X. Pedeli, M.-Y. Tsai, N. Künzli, T. Schikowski, A. Marcon, et al. Development of NO₂ and NO_x Land Use Regression Models for Estimating Air Pollution Exposure in 36 Study Areas in Europe – the ESCAPE Project. *Atmospheric Environment*, 72:10–23, 2013.
- [23] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum Learning. In *International Conference on Machine Learning*, pages 41–48, Montreal Quebec Canada, June 2009. ACM. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553380.
- [24] J. Bergstra and Y. Bengio. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.
- [25] S. Bosse, D. Maniry, K.-R. Müller, T. Wiegand, and W. Samek. Deep Neural Networks for No-Reference and Full-Reference Image Quality Assessment. *IEEE Transactions on Image Processing*, 27(1):206–219, Jan. 2018. doi: 10.1109/tip.2017.2760518.
- [26] B. Brattoli, K. Roth, and B. Ommer. MIC: Mining Interclass Characteristics for Improved Metric Learning. In *IEEE International Conference on Computer Vision*, pages 7999–8008, Seoul, Korea (South), Oct. 2019. IEEE. ISBN 978-1-72814-803-8. doi: 10.1109/ICCV.2019.00809.
- [27] L. Breiman. Random Forests. *Springer Machine Learning*, 45(1):5–32, 2001.
- [28] C. Brokamp, R. Jandarov, MB. Rao, G. LeMasters, and P. Ryan. Exposure Assessment Models for Elemental Components of Particulate Matter in an Urban Environment: A Comparison of Regression and Random Forest Approaches. *Atmospheric Environment*, 151:1–11, 2017.
- [29] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901, June 2020.
- [30] B. Brunekreef and S. T. Holgate. Air Pollution and Health. *The Lancet*, 360(9341):1233–1242, 2002.
- [31] A. G. Buevich, A. N. Medvedev, A. P. Sergeev, D. A. Tarasov, A. V. Shichkin, M. V. Sergeeva, and TB. Atanasova. Modeling of Surface Dust Concentrations using Neural Networks and Kriging. In *International Conference on Applications of Mathematics in Engineering and Economics*, volume 1789, page 020004. AIP Publishing, 2016.

Bibliography

- [32] F. Cakir, K. He, X. Xia, B. Kulis, and S. Sclaroff. Deep Metric Learning to Rank. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1861–1870, Long Beach, CA, USA, June 2019. IEEE. ISBN 978-1-72813-293-8. doi: 10.1109/CVPR.2019.00196.
- [33] N. Cesa-Bianchi, C. Gentile, and L. Zaniboni. Incremental Algorithms for Hierarchical Classification. *Journal of Machine Learning Research*, 7(Jan):31–54, 2006.
- [34] A. Champendal, M. Kanevski, and P.-E. Huguenot. Air Pollution Mapping using Nonlinear Land Use Regression Models. In *International Conference on Computational Science and Its Applications*, pages 682–690. Springer, 2014.
- [35] S. Changpinyo, P. Sharma, N. Ding, and R. Soricut. Conceptual 12M: Pushing Web-Scale Image-Text Pre-Training To Recognize Long-Tail Visual Concepts. In *IEEE Computer Vision and Pattern Recognition*, 2021.
- [36] H. Chefer, I. Schwartz, and L. Wolf. Optimizing Relevance Maps of Vision Transformers Improves Robustness. *arXiv:2206.01161 [cs]*, 2022.
- [37] L. Chen, J. Chen, H. Hajimirsadeghi, and G. Mori. Adapting Grad-CAM for Embedding Networks. *arXiv:2001.06538 [cs]*, Jan. 2020.
- [38] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A Simple Framework for Contrastive Learning of Visual Representations. *arXiv:2002.05709 [cs, stat]*, June 2020.
- [39] Y.-C. Chen, L. Li, L. Yu, A. El Kholy, F. Ahmed, Z. Gan, Y. Cheng, and J. Liu. Uniter: Universal Image-Text Representation Learning. In *European Conference on Computer Vision*, pages 104–120. Springer, 2020.
- [40] Z. Chen, V. Badrinarayanan, C.-Y. Lee, and A. Rabinovich. GradNorm: Gradient Normalization for Adaptive Loss Balancing in Deep Multitask Networks. In *International Conference on Machine Learning*, pages 794–803. PMLR, July 2018.
- [41] W.-T. Chu and H.-J. Guo. Movie Genre Classification based on Poster Images with Deep Neural Networks. In *Workshop on Multimodal Understanding of Social, Affective and Subjective Attributes*, pages 39–45, 2017.
- [42] R. Cipolla, Y. Gal, and A. Kendall. Multi-task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics. In *IEEE Computer Vision and Pattern Recognition*, pages 7482–7491, Salt Lake City, UT, USA, June 2018. IEEE. ISBN 978-1-5386-6420-9. doi: 10.1109/CVPR.2018.00781.
- [43] A. Clark. Pillow (PIL Fork) Documentation, 2015. URL <https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf>.
- [44] B. O. Community. Blender — a 3D modelling and rendering package. Blender Foundation, 2018. URL <http://www.blender.org>.

- [45] A. D’Amour, K. Heller, D. Moldovan, B. Adlam, B. Alipanahi, A. Beutel, C. Chen, J. Deaton, J. Eisenstein, M. D. Hoffman, F. Hormozdiari, N. Houlsby, S. Hou, G. Jerfel, A. Karthikesalingam, M. Lucic, Y. Ma, C. McLean, D. Mincu, A. Mitani, A. Montanari, Z. Nado, V. Natarajan, C. Nielson, T. F. Osborne, R. Raman, K. Ramasamy, R. Sayres, J. Schrouff, M. Seneviratne, S. Sequeira, H. Suresh, V. Veitch, M. Vladymyrov, X. Wang, K. Webster, S. Yadlowsky, T. Yun, X. Zhai, and D. Sculley. Underspecification Presents Challenges for Credibility in Modern Machine Learning. *arXiv:2011.03395 [cs, stat]*, Nov. 2020.
- [46] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE, June 2009. doi: 10.1109/cvpr.2009.5206848.
- [47] J. Deng, J. Guo, N. Xue, and S. Zafeiriou. ArcFace: Additive Angular Margin Loss for Deep Face Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 4685–4694, Long Beach, CA, USA, June 2019. IEEE. ISBN 978-1-72813-293-8. doi: 10.1109/CVPR.2019.00482.
- [48] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423.
- [49] P. Dhariwal and A. Nichol. Diffusion Models Beat GANs on Image Synthesis. In *Advances in Neural Information Processing Systems*, volume 34, pages 8780–8794. Curran Associates, Inc., 2021.
- [50] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *arXiv:2010.11929 [cs]*, June 2021.
- [51] J. Duchi, E. Hazan, and Y. Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12(7):39, 2011.
- [52] M. Eeftens, R. Beelen, K. de Hoogh, T. Bellander, G. Cesaroni, M. Cirach, C. Declercq, A. Dedele, E. Dons, A. de Nazelle, et al. Development of Land Use Regression Models for PM_{2.5}, PM_{2.5} Absorbance, PM₁₀ and PM_{coarse} in 20 European Study Areas; Results of the ESCAPE Project. *Environmental Science & Technology*, 46(20):11195–11205, 2012.
- [53] A. El-Nouby, G. Izacard, H. Touvron, I. Laptev, H. Jegou, and E. Grave. Are Large-scale Datasets Necessary for Self-Supervised Pre-training? *arXiv:2112.10740 [cs]*, Dec. 2021.

Bibliography

- [54] K. ELKarazle, V. Raman, and P. Then. Facial Age Estimation Using Machine Learning Techniques: An Overview. *Big Data and Cognitive Computing*, 6(4):128, Oct. 2022. ISSN 2504-2289. doi: 10.3390/bdcc6040128.
- [55] D. M. Elsom. *Atmospheric Pollution: A Global Problem*. Blackwell Oxford, 1992.
- [56] D. Endres and J. Schindelin. A new Metric for Probability Distributions. *IEEE Transactions on Information Theory*, 49(7):1858–1860, July 2003. ISSN 1557-9654. doi: 10.1109/TIT.2003.813506.
- [57] L. Engstrom, B. Tran, D. Tsipras, L. Schmidt, and A. Madry. Exploring the Landscape of Spatial Robustness. In *International Conference on Machine Learning*, pages 1802–1811. PMLR, May 2019.
- [58] J. Fan, Q. Li, J. Hou, X. Feng, H. Karimian, and S. Lin. A Spatiotemporal Prediction Framework for Air Pollution based on Deep RNN. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 4:15, 2017.
- [59] S. Y. Feng, V. Gangal, J. Wei, S. Chandar, S. Vosoughi, T. Mitamura, and E. Hovy. A Survey of Data Augmentation Approaches for NLP. *arXiv:2105.03075 [cs]*, Dec. 2021.
- [60] M. Fischer, K. Kobs, and A. Hotho. NICER: Aesthetic Image Enhancement with Humans in the Loop. *arXiv:2012.01778 [cs]*, 2020.
- [61] R. Fong and A. Vedaldi. Interpretable Explanations of Black Boxes by Meaningful Perturbation. *IEEE International Conference on Computer Vision*, pages 3449–3457, Oct. 2017. doi: 10.1109/ICCV.2017.371.
- [62] P. Foret, A. Kleiner, H. Mobahi, and B. Neyshabur. Sharpness-Aware Minimization for Efficiently Improving Generalization. *arXiv:2010.01412 [cs, stat]*, Apr. 2021. doi: 10.48550/arXiv.2010.01412.
- [63] D. Freedman, R. Pisani, and R. Purves. *Statistics (International Student Edition)*. Norton & Company, 2007.
- [64] A. Frome, G. S. Corrado, J. Shlens, S. Bengio, J. Dean, T. Mikolov, et al. Devise: A Deep Visual-Semantic Embedding Model. In *Advances in Neural Information Processing Systems*, 2013.
- [65] Y. Fu and T. S. Huang. Human Age Estimation with Regression on Discriminative Aging Manifold. *IEEE Transactions on Multimedia*, 10(4):578–584, 2008.
- [66] J. Geiping, M. Goldblum, G. Somepalli, R. Shwartz-Ziv, T. Goldstein, and A. G. Wilson. How Much Data Are Augmentations Worth? An Investigation into Scaling Laws, Invariance, and Implicit Regularization. *arXiv:2210.06441 [cs]*, Oct. 2022. doi: 10.48550/arXiv.2210.06441.

- [67] M. George. Choosing a Point from the Surface of a Sphere. *The Annals of Mathematical Statistics*, 43:645–646, 1972.
- [68] S. Gidaris, P. Singh, and N. Komodakis. Unsupervised Representation Learning by Predicting Image Rotations. *arXiv:1803.07728 [cs]*, Mar. 2018.
- [69] C. H. E. Gilbert. Vader: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. In *International Conference on Weblogs and Social Media*, 2014.
- [70] X. Glorot and Y. Bengio. Understanding the Difficulty of Training Deep Feedforward Neural Networks. In *International Conference on Artificial Intelligence and Statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, Mar. 2010.
- [71] A. Go, R. Bhayani, and L. Huang. Twitter Sentiment Classification using Distant Supervision. *CS224N Project Report, Stanford*, 1(12):1–6, 2009.
- [72] S. A. Golestaneh and K. Kitani. No-Reference Image Quality Assessment via Feature Fusion and Multi-Task Learning. *arXiv:2006.03783 [cs]*, 2020.
- [73] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.
- [74] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Networks. *Communications of the ACM*, 63(11):139–144, Oct. 2020. ISSN 0001-0782. doi: 10.1145/3422622.
- [75] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and Harnessing Adversarial Examples. *arXiv:1412.6572 [cs, stat]*, Mar. 2015.
- [76] A. Gordo and D. Larlus. Beyond Instance-Level Image Retrieval: Leveraging Captions to Learn a Global Visual Representation for Semantic Retrieval. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 5272–5281, Honolulu, HI, July 2017. IEEE. ISBN 978-1-5386-0457-1. doi: 10.1109/CVPR.2017.560.
- [77] P. Goyal, D. Mahajan, A. Gupta, and I. Misra. Scaling and Benchmarking Self-Supervised Visual Representation Learning. In *IEEE International Conference on Computer Vision*. IEEE, Oct. 2019. doi: 10.1109/iccv.2019.00649.
- [78] G. Guo, G. Mu, Y. Fu, and T. S. Huang. Human Age Estimation using Bio-inspired Features. In *IEEE Computer Vision and Pattern Recognition*. IEEE, 2009.
- [79] A. Guzhov, F. Raue, J. Hees, and A. Dengel. AudioCLIP: Extending CLIP to Image, Text and Audio. *arXiv:2106.13043 [cs, eess]*, June 2021. doi: 10.48550/arXiv.2106.13043.
- [80] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality Reduction by Learning an Invariant Mapping. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 1735–1742, June 2006. doi: 10.1109/CVPR.2006.100.

Bibliography

- [81] S. H. Haji and A. M. Abdulazeez. Comparison of Optimization Techniques based on Gradient Descent Algorithm: A Review. *PalArch's Journal of Archaeology of Egypt / Egyptology*, 18(4):2715–2743, Feb. 2021. ISSN 1567-214X.
- [82] H. Han, C. Otto, and A. K. Jain. Age Estimation from Face Images: Human vs. Machine Performance. In *International Conference on Biometrics*, pages 1–8, Madrid, Spain, June 2013. IEEE. ISBN 978-1-4799-0310-8. doi: 10.1109/ICB.2013.6613022.
- [83] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array Programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020. ISSN 1476-4687. doi: 10.1038/s41586-020-2649-2.
- [84] W. Haynes. Bonferroni Correction. In W. Dubitzky, O. Wolkenhauer, K.-H. Cho, and H. Yokota, editors, *Encyclopedia of Systems Biology*, pages 154–154. Springer New York, New York, NY, 2013. ISBN 978-1-4419-9863-7.
- [85] K. He, X. Zhang, S. Ren, and J. Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *arXiv:1502.01852 [cs]*, Feb. 2015.
- [86] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, June 2016. doi: 10.1109/CVPR.2016.90.
- [87] K. He, X. Chen, S. Xie, Y. Li, P. Dollar, and R. Girshick. Masked Autoencoders Are Scalable Vision Learners. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 15979–15988, New Orleans, LA, USA, June 2022. IEEE. ISBN 978-1-66546-946-3. doi: 10.1109/CVPR52688.2022.01553.
- [88] X. He, I. Nassar, J. Kiros, G. Haffari, and M. Norouzi. Generate, Annotate, and Learn: NLP with Synthetic Text. *Transactions of the Association for Computational Linguistics*, 10:826–842, Aug. 2022. ISSN 2307-387X. doi: 10.1162/tacl_a_00492.
- [89] D. Hendrycks and T. Dietterich. Benchmarking Neural Network Robustness to Common Corruptions and Perturbations. In *International Conference on Learning Representations*, volume 4, Mar. 2019.
- [90] S. Hentschel, K. Kobs, and A. Hotho. CLIP knows Image Aesthetics. *Frontiers in Artificial Intelligence*, 5, 2022. ISSN 2624-8212.
- [91] G. Hinton, N. Srivastava, and K. Swersky. Overview of Mini-Batch Gradient Descent. *Neural Networks for Machine Learning*, 575(8), 2012.

- [92] G. Hoek, R. Beelen, K. de Hoogh, D. Vienneau, J. Gulliver, P. Fischer, and D. Briggs. A Review of Land-Use Regression Models to Assess Spatial Variation of Outdoor Air Pollution. *Atmospheric environment*, 42(33):7561–7578, 2008.
- [93] K. Hornik, M. Stinchcombe, and H. White. Multilayer Feedforward Networks are Universal Approximators. *Neural Networks*, 2(5):359–366, Jan. 1989. ISSN 0893-6080. doi: 10.1016/0893-6080(89)90020-8.
- [94] V. Hosu, B. Goldlucke, and D. Saupe. Effective Aesthetics Prediction With Multi-Level Spatially Pooled Features. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 9367–9375, Long Beach, CA, USA, June 2019. IEEE. ISBN 978-1-72813-293-8. doi: 10.1109/CVPR.2019.00960.
- [95] L. Hou, C.-P. Yu, and D. Samaras. Squared Earth Mover’s Distance-based Loss for Training Deep Neural Networks. *arXiv:1611.05916 [cs]*, Nov. 2016.
- [96] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv:1704.04861 [cs]*, Apr. 2017.
- [97] J. Hu, J. Lu, and Y.-P. Tan. Discriminative Deep Metric Learning for Face Verification in the Wild. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1875–1882, 2014.
- [98] Z. Hu, D. Wu, F. Nie, and R. Wang. Generalization Bottleneck in Deep Metric Learning. *Information Sciences*, 581:249–261, 2021.
- [99] M. Huai, H. Xue, C. Miao, L. Yao, L. Su, C. Chen, and A. Zhang. Deep Metric Learning: The Generalization Analysis and an Adaptive Algorithm. In *International Joint Conference on Artificial Intelligence*, pages 2535–2541, Macao, China, Aug. 2019. International Joint Conferences on Artificial Intelligence Organization. ISBN 978-0-9992411-4-1. doi: 10.24963/ijcai.2019/352.
- [100] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely Connected Convolutional Networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2261–2269. IEEE, July 2017. doi: 10.1109/CVPR.2017.243.
- [101] H. Huang and Y. Liang. Learning Robust Embedding Representation With Hybrid Loss for Classification and Verification. *IEEE Access*, 7:13643–13652, 2019. ISSN 2169-3536. doi: 10.1109/ACCESS.2019.2894652.
- [102] P. J. Huber. Robust Estimation of a Location Parameter. *The Annals of Mathematical Statistics*, 35(1):73–101, Mar. 1964. ISSN 0003-4851, 2168-8990. doi: 10.1214/aoms/1177703732.
- [103] E. Hubinger. An Overview of 11 Proposals for Building Safe Advanced AI. *arXiv:2012.07532 [cs]*, Dec. 2020.

Bibliography

- [104] L. Hui and M. Belkin. Evaluation of Neural Architectures Trained with Square Loss vs Cross-Entropy in Classification Tasks. *arXiv:2006.07322 [cs, stat]*, Oct. 2021. doi: 10.48550/arXiv.2006.07322.
- [105] A. Iana, S. Jung, P. Naeser, A. Birukou, S. Hertling, and H. Paulheim. Building a Conference Recommender System based on SciGraph and WikiCFP. In *International Conference on Semantic Systems*, pages 117–123. Springer, 2019.
- [106] M. Ibrahim, Q. Garrido, A. Morcos, and D. Bouchacourt. The Robustness Limits of SoTA Vision Models to Natural Variation. *arXiv:2210.13604 [cs]*, Oct. 2022.
- [107] G. Ilharco, M. Wortsman, R. Wightman, C. Gordon, N. Carlini, R. Taori, A. Dave, V. Shankar, H. Namkoong, J. Miller, H. Hajishirzi, A. Farhadi, and L. Schmidt. OpenCLIP, July 2021.
- [108] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*, pages 448–456. PMLR, June 2015.
- [109] M. Izbicki, E. E. Papalexakis, and V. J. Tsotras. Exploiting the Earth’s Spherical Geometry to Geolocate Images. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, page 16, 2019.
- [110] A. Jaiswal, A. R. Babu, M. Z. Zadeh, D. Banerjee, and F. Makedon. A Survey on Contrastive Self-Supervised Learning. *MDPI Technologies*, 9(1):2, Mar. 2021. ISSN 2227-7080. doi: 10.3390/technologies9010002.
- [111] L. Jiang, D. Huang, M. Liu, and W. Yang. Beyond Synthetic Noise: Deep Learning on Controlled Noisy Labels. In *International Conference on Machine Learning*, pages 4804–4815. PMLR, Nov. 2020.
- [112] L. Jing and Y. Tian. Self-supervised Visual Feature Learning with Deep Neural Networks: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2020. doi: 10.1109/tpami.2020.2992393.
- [113] J. Johnson, M. Douze, and H. Jégou. Billion-Scale Similarity Search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.
- [114] H. Jun, B. Ko, Y. Kim, I. Kim, and J. Kim. Combination of Multiple Global Descriptors for Image Retrieval. *arXiv:1903.10663 [cs]*, 2019.
- [115] U. Kamath, J. Liu, and J. Whitaker. Convolutional Neural Networks. In U. Kamath, J. Liu, and J. Whitaker, editors, *Deep Learning for NLP and Speech Recognition*, pages 263–314. Springer International Publishing, Cham, 2019. ISBN 978-3-030-14596-5.
- [116] L. Kang, P. Ye, Y. Li, and D. Doermann. Convolutional Neural Networks for No-Reference Image Quality Assessment. In *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, June 2014. doi: 10.1109/cvpr.2014.224.

- [117] L. Kang, P. Ye, Y. Li, and D. Doermann. Simultaneous Estimation of Image Quality and Distortion via Multi-task Convolutional Neural Networks. In *IEEE International Conference on Image Processing*, pages 2791–2795, Quebec City, QC, Canada, Sept. 2015. IEEE. ISBN 978-1-4799-8339-1. doi: 10.1109/ICIP.2015.7351311.
- [118] N. Kang, M. A. Doornenbal, and R. J. Schijvenaars. Elsevier Journal Finder: Recommending Journals for your Paper. In *ACM Conference on Recommender Systems*, pages 261–264. ACM, 2015.
- [119] M. Kaya and H. Ş. Bilge. Deep Metric Learning: A Survey. *Symmetry*, 11(9):1066, Sept. 2019. doi: 10.3390/sym11091066.
- [120] M. Kaytoue, A. Silva, L. Cerf, W. M. Jr., and C. Raissi. Watch me Playing, I am a Professional: A First Study on Video Game Live Streaming. In *International Conference on World Wide Web*, pages 1181–1188. ACM, 2012. ISBN 978-1-4503-1230-1.
- [121] K. Kc, Z. Yin, D. Li, and Z. Wu. Impacts of Background Removal on Convolutional Neural Networks for Plant Disease Classification In-Situ. *MDPI Agriculture*, 11(9): 827, 2021.
- [122] J. Ke, Q. Wang, Y. Wang, P. Milanfar, and F. Yang. MUSIQ: Multi-scale Image Quality Transformer. In *IEEE International Conference on Computer Vision*, pages 5148–5157, 2021.
- [123] L. M. Kells, W. F. Kern, and J. R. Bland. *Plane and Spherical Trigonometry with Tables*. US Armed Forces Institute, 1940.
- [124] K. Kersting, J. Peters, and C. Rothkopf. Was ist eine Professur fuer Kuenstliche Intelligenz? *arXiv:1903.09516 [cs]*, 2019.
- [125] N. S. Keskar and R. Socher. Improving Generalization Performance by Switching from Adam to SGD. *arXiv:1712.07628 [cs, math]*, Dec. 2017.
- [126] Y. Kim. Convolutional Neural Networks for Sentence Classification. In *Conference on Empirical Methods in Natural Language Processing*, pages 1746–1751, Doha, Qatar, Oct. 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1181.
- [127] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, 2014.
- [128] P. Kirichenko, P. Izmailov, and A. G. Wilson. Last Layer Re-Training is Sufficient for Robustness to Spurious Correlations. *arXiv:2204.02937 [cs, stat]*, Apr. 2022.
- [129] K. Ko, J.-T. Lee, and C.-S. Kim. PAC-Net: Pairwise Aesthetic Comparison Network for Image Aesthetic Assessment. In *International Conference on Image Processing*. IEEE, Oct. 2018. doi: 10.1109/icip.2018.8451621.

Bibliography

- [130] K. Kobs and A. Hotho. On Background Bias in Deep Metric Learning. In *International Conference on Machine Vision*, volume 12701, pages 331–338. SPIE, June 2023. doi: 10.1117/12.2679270.
- [131] K. Kobs, T. Koopmann, A. Zehe, D. Fernes, P. Krop, and A. Hotho. Where to Submit? Helping Researchers to Choose the Right Venue. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 878–883, Online, Nov. 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.78.
- [132] K. Kobs, M. Steininger, A. Zehe, F. Lautenschlager, and A. Hotho. SimLoss: Class Similarities in Cross Entropy. In D. Helic, G. Leitner, M. Stettinger, A. Felfernig, and Z. W. Raś, editors, *Foundations of Intelligent Systems*, Lecture Notes in Computer Science, pages 431–439, Cham, 2020. Springer International Publishing. ISBN 978-3-030-59491-6. doi: 10.1007/978-3-030-59491-6_41.
- [133] K. Kobs, A. Zehe, A. Bernstetter, J. Chibane, J. Pfister, J. Tritscher, and A. Hotho. Emote-Controlled: Obtaining Implicit Viewer Feedback Through Emote-Based Sentiment Analysis on Comments of Popular Twitch.tv Channels. *ACM Transactions on Social Computing*, 3(2), 2020.
- [134] K. Kobs, M. Steininger, A. Dulny, and A. Hotho. Do Different Deep Metric Learning Losses Lead to Similar Learned Features? In *IEEE International Conference on Computer Vision*, pages 10644–10654, 2021.
- [135] K. Kobs, M. Steininger, and A. Hotho. InDiReCT: Language-Guided Zero-Shot Deep Metric Learning for Images. In *IEEE Winter Conference on Applications of Computer Vision*, pages 1063–1072, 2023.
- [136] A. Kolesnikov, X. Zhai, and L. Beyer. Revisiting Self-Supervised Visual Representation Learning. In *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, June 2019. doi: 10.1109/cvpr.2019.00202.
- [137] A. Kortylewski, B. Egger, A. Schneider, T. Gerig, A. Morel-Forster, and T. Vetter. Analyzing and Reducing the Damage of Dataset Bias to Face Recognition With Synthetic Data. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 2261–2268, Long Beach, CA, USA, June 2019. IEEE. ISBN 978-1-72812-506-0. doi: 10.1109/CVPRW.2019.00279.
- [138] P. Kralj Novak, J. Smailović, B. Sluban, and I. Mozetič. Sentiment of Emojis. *PLoS ONE*, 10(12):e0144296, 2015.
- [139] J. Krause, M. Stark, J. Deng, and L. Fei-Fei. 3D Object Representations for Fine-Grained Categorization. In *IEEE International Conference on Computer Vision Workshops*, pages 554–561, Sydney, Australia, Dec. 2013. IEEE. ISBN 978-1-4799-3022-7. doi: 10.1109/ICCVW.2013.77.

- [140] K. Krishna, S. Garg, J. P. Bigham, and Z. C. Lipton. Downstream Datasets Make Surprisingly Good Pretraining Corpora. *arXiv:2209.14389 [cs]*, Sept. 2022.
- [141] A. Krizhevsky and G. Hinton. Learning Multiple Layers of Features from Tiny Images. Technical report, Citeseer, 2009.
- [142] S. Lapuschkin, S. Waldchen, A. Binder, G. Montavon, W. Samek, and K.-R. Muller. Unmasking Clever Hans Predictors and Assessing What Machines Really Learn. *Nature Communications*, 10(1):1096, Mar. 2019. ISSN 2041-1723. doi: 10.1038/s41467-019-08987-4.
- [143] O. Le Meur and T. Baccino. Methods for Comparing Scanpaths and Saliency Maps: Strengths and Weaknesses. *Behavior Research Methods*, 45(1):251–266, Mar. 2013. ISSN 1554-3528. doi: 10.3758/s13428-012-0226-9.
- [144] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based Learning Applied to Document Recognition. *IEEE*, 86(11):2278–2324, 1998.
- [145] Y. LeCun, Y. Bengio, and G. Hinton. Deep Learning. *Nature*, 521(7553):436–444, May 2015. ISSN 1476-4687. doi: 10.1038/nature14539.
- [146] C. Lei, B. Hu, D. Wang, S. Zhang, and Z. Chen. A Preliminary Study on Data Augmentation of Deep Learning for Image Classification. In *Asia-Pacific Symposium on Internetware*, pages 1–6, Fukuoka Japan, Oct. 2019. ACM. ISBN 978-1-4503-7701-0. doi: 10.1145/3361242.3361259.
- [147] Z. Leng, M. Tan, C. Liu, E. D. Cubuk, X. Shi, S. Cheng, and D. Anguelov. PolyLoss: A Polynomial Expansion Perspective of Classification Loss Functions. *arXiv:2204.12511 [cs]*, May 2022. doi: 10.48550/arXiv.2204.12511.
- [148] C. Lennan, H. Nguyen, and D. Tran. Image Quality Assessment, 2018. URL <https://github.com/idealo/image-quality-assessment>. Last accessed: 2023-09-29.
- [149] L. Li, H. Zhu, S. Zhao, G. Ding, and W. Lin. Personality-Assisted Multi-Task Learning for Generic and Personalized Image Aesthetics Assessment. *IEEE Transactions on Image Processing*, 29:3898–3910, 2020. ISSN 1941-0042. doi: 10.1109/TIP.2020.2968285.
- [150] Y. Li, J. Huang, and J. Luo. Using User Generated Online Photos to Estimate and Monitor Air Pollution in Major Cities. In *International Conference on Internet Multimedia Computing and Service*, page 79. ACM, 2015.
- [151] P. P. Liang, A. Zadeh, and L.-P. Morency. Foundations and Recent Trends in Multimodal Machine Learning: Principles, Challenges, and Open Questions. *arXiv:2209.03430 [cs]*, Sept. 2022.
- [152] S. Liang and Y. Zhang. A Simple General Approach to Balance Task Difficulty in Multi-Task Learning. *arXiv:2002.04792 [cs]*, Feb. 2020.

Bibliography

- [153] Y. Liang, S. Li, C. Yan, M. Li, and C. Jiang. Explaining the Black-Box Model: A Survey of Local Interpretation Methods for Deep Neural Networks. *Neurocomputing*, 419:168–182, Jan. 2021. ISSN 09252312. doi: 10.1016/j.neucom.2020.08.011.
- [154] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar. Focal Loss for Dense Object Detection. In *IEEE International Conference on Computer Vision*. IEEE, Oct. 2017. doi: 10.1109/iccv.2017.324.
- [155] K. Liu and H. Ma. Exploring Background-Bias for Anomaly Detection in Surveillance Videos. In *ACM Multimedia*, pages 1490–1499, 2019.
- [156] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. *arXiv:2107.13586 [cs]*, July 2021.
- [157] S. Liu, E. Johns, and A. J. Davison. End-To-End Multi-Task Learning With Attention. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1871–1880, Long Beach, CA, USA, June 2019. IEEE. ISBN 978-1-72813-293-8. doi: 10.1109/CVPR.2019.00197.
- [158] W. Liu, X. Li, Z. Chen, G. Zeng, T. León, J. Liang, G. Huang, Z. Gao, S. Jiao, X. He, et al. Land Use Regression Models Coupled with Meteorology to Model Spatial and Temporal Variability of NO₂ and PM₁₀ in Changsha, China. *Atmospheric Environment*, 116:272–280, 2015.
- [159] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song. SphereFace: Deep Hypersphere Embedding for Face Recognition. *arXiv:1704.08063 [cs]*, Jan. 2018.
- [160] X. Liu, J. V. D. Weijer, and A. D. Bagdanov. RankIQA: Learning from Rankings for No-Reference Image Quality Assessment. In *IEEE International Conference on Computer Vision*. IEEE, Oct. 2017. doi: 10.1109/iccv.2017.118.
- [161] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv:1907.11692 [cs]*, July 2019.
- [162] Z. Liu, P. Luo, S. Qiu, X. Wang, and X. Tang. DeepFashion: Powering Robust Clothes Recognition and Retrieval with Rich Annotations. In *IEEE Computer Vision and Pattern Recognition*, June 2016.
- [163] D. Löffler, L. Giron, and J. Hurtienne. Night Mode, Dark Thoughts: Background Color Influences the Perceived Sentiment of Chat Messages. In *Human-Computer Interaction - INTERACT*, volume 10514 of *Lecture Notes in Computer Science*, pages 184–201. Springer, 2017. ISBN 978-3-319-67684-5.
- [164] T. Luddecke and A. Ecker. Image Segmentation Using Text and Image Prompts. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 7076–7086, New Orleans, LA, USA, June 2022. IEEE. ISBN 978-1-66546-946-3. doi: 10.1109/CVPR52688.2022.00695.

- [165] S. M. Lundberg and S.-I. Lee. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.
- [166] K. Ma, W. Liu, K. Zhang, Z. Duanmu, Z. Wang, and W. Zuo. End-to-End Blind Image Quality Assessment Using Deep Neural Networks. *IEEE Transactions on Image Processing*, 27(3):1202–1213, Mar. 2018. doi: 10.1109/tip.2017.2774045.
- [167] S. Ma, J. Liu, and C. W. Chen. A-Lamp: Adaptive Layout-Aware Multi-patch Deep Convolutional Neural Network for Photo Aesthetic Assessment. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 722–731, Honolulu, HI, July 2017. IEEE. ISBN 978-1-5386-0457-1. doi: 10.1109/CVPR.2017.84.
- [168] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning Word Vectors for Sentiment Analysis. In *Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, HLT '11*, pages 142–150, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics. ISBN 978-1-932432-87-9.
- [169] A. L. Maas, A. Y. Hannun, A. Y. Ng, et al. Rectifier Nonlinearities Improve Neural Network Acoustic Models. In *International Conference on Machine Learning*, volume 30, page 3. Citeseer, 2013.
- [170] S. Madan, T. Sasaki, T.-M. Li, X. Boix, and H. Pfister. Small in-distribution Changes in 3D Perspective and Lighting fool both CNNs and Transformers. *arXiv:2106.16198 [cs]*, June 2021.
- [171] A. Madsen, S. Reddy, and S. Chandar. Post-hoc Interpretability for Neural NLP: A Survey. *arXiv:2108.04840 [cs]*, Apr. 2022.
- [172] H. B. Mann and D. R. Whitney. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics*, 18(1):50–60, 1947. ISSN 0003-4851.
- [173] C. D. Manning, K. Clark, J. Hewitt, U. Khandelwal, and O. Levy. Emergent Linguistic Structure in Artificial Neural Networks trained by Self-Supervision. *National Academy of Sciences*, 117(48):30046–30054, Dec. 2020. doi: 10.1073/pnas.1907367117.
- [174] J. J. McAuley, R. Pandey, and J. Leskovec. Inferring Networks of Substitutable and Complementary Products. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, Aug. 2015. doi: 10.1145/2783258.2783381.
- [175] N. Mehta, D. Lee, and A. Gray. Minimax Multi-Task Learning and a Generalized Loss-Compositional Paradigm for MTL. In *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

Bibliography

- [176] X. Meng, L. Chen, J. Cai, B. Zou, C.-F. Wu, Q. Fu, Y. Zhang, Y. Liu, and H. Kan. A Land Use Regression Model for Estimating the NO₂ Concentration in Shanghai, China. *Environmental Research*, 137:308–315, 2015.
- [177] D. Merckx, S. L. Frank, and M. Ernestus. Language Learning using Speech to Image Retrieval. *arXiv:1909.03795 [cs]*, 2019.
- [178] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv:1301.3781 [cs]*, 2013.
- [179] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed Representations of Words and Phrases and their Compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, pages 3111–3119. Curran Associates, Inc., 2013.
- [180] T. Mikolov, W.-t. Yih, and G. Zweig. Linguistic Regularities in Continuous Space Word Representations. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, 2013.
- [181] T. Milbich, K. Roth, B. Brattoli, and B. Ommer. Sharing Matters for Generalization in Deep Metric Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(1):416–427, 2020.
- [182] T. Milbich, K. Roth, S. Sinha, L. Schmidt, M. Ghassemi, and B. Ommer. Characterizing Generalization under Out-Of-Distribution Shifts in Deep Metric Learning. *Advances in Neural Information Processing Systems*, 34, 2021.
- [183] G. A. Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41, Nov. 1995. ISSN 0001-0782. doi: 10.1145/219717.219748.
- [184] R. Mokady, A. Hertz, and A. H. Bermano. ClipCap: CLIP Prefix for Image Captioning. *arXiv:2111.09734 [cs]*, Nov. 2021.
- [185] C. Molnar. *Interpretable Machine Learning*. Lulu.com, 2020. ISBN 978-0-244-76852-2.
- [186] D. R. Montagne, G. Hoek, J. O. Klomp maker, M. Wang, K. Meliefste, and B. Brunekreef. Land Use Regression Models for Ultrafine Particles and Black Carbon based on Short-Term Monitoring Predict Past Spatial Variation. *Environmental Science & Technology*, 49(14):8712–8720, 2015.
- [187] M. Moradi and M. Samwald. Evaluating the Robustness of Neural Language Models to Input Perturbations. In *Conference on Empirical Methods in Natural Language Processing*, pages 1558–1570, Online and Punta Cana, Dominican Republic, Nov. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.117.

- [188] N. Morgan and H. Bourlard. Generalization and parameter estimation in feedforward nets: Some experiments. In *Advances in Neural Information Processing Systems*, 1990.
- [189] D. W. Morley and J. Gulliver. A Land Use Regression Variable Generation, Modelling and Prediction Tool for Air Pollution Exposure Assessment. *Environmental Modelling & Software*, 105:17–23, 2018.
- [190] Y. Movshovitz-Attias, A. Toshev, T. K. Leung, S. Ioffe, and S. Singh. No Fuss Distance Metric Learning using Proxies. *arXiv:1703.07464 [cs]*, pages 360–368, Aug. 2017.
- [191] N. Murray, L. Marchesotti, and F. Perronnin. AVA: A Large-Scale Database for Aesthetic Visual Analysis. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2408–2415, Providence, RI, June 2012. ISBN 978-1-4673-1228-8 978-1-4673-1226-4 978-1-4673-1227-1. doi: 10.1109/CVPR.2012.6247954.
- [192] K. Murugesan and J. Carbonell. Self-Paced Multitask Learning with Shared Knowledge. In *International Joint Conference on Artificial Intelligence*, pages 2522–2528, Melbourne, Australia, Aug. 2017. International Joint Conferences on Artificial Intelligence Organization. ISBN 978-0-9992411-0-3. doi: 10.24963/ijcai.2017/351.
- [193] K. Musgrave, S. Belongie, and S.-N. Lim. A Metric Learning Reality Check. In *European Conference on Computer Vision*, pages 681–699. Springer, Mar. 2020.
- [194] S. Muttoo, L. Ramsay, B. Brunekreef, R. Beelen, K. Meliefste, and R. N. Naidoo. Land Use Regression Modelling Estimating Nitrogen Oxides Exposure in Industrial South Durban, South Africa. *Science of the Total Environment*, 610:1439–1447, 2018.
- [195] V. Nair and G. E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *International Conference on Machine Learning*, pages 807–814, 2010.
- [196] P. Nakov, S. Rosenthal, Z. Kozareva, V. Stoyanov, A. Ritter, and T. Wilson. SemEval-2013 Task 2: Sentiment Analysis in Twitter. In *International Workshop on Semantic Evaluation*, volume 2, pages 312–320, 2013.
- [197] P. Nakov, A. Ritter, S. Rosenthal, F. Sebastiani, and V. Stoyanov. SemEval-2016 Task 4: Sentiment Analysis in Twitter. In *International Workshop on Semantic Evaluation*, pages 1–18, 2016.
- [198] J. Nam, H. Cha, S. Ahn, J. Lee, and J. Shin. Learning from Failure: De-biasing Classifier from Biased Classifier. *Advances in Neural Information Processing Systems*, 33:20673–20684, 2020.
- [199] G. Nascimento, M. Ribeiro, L. Cerf, N. Cesário, M. Kaytoue, C. Raïssi, T. Vasconcelos, and W. Meira. Modeling and Analyzing the Video Game Live-Streaming Community. In *Latin American Web Congress*, pages 1–9. IEEE, 2014.

Bibliography

- [200] Z. Niu, M. Zhou, L. Wang, X. Gao, and G. Hua. Ordinal Regression with Multiple Output CNN for Age Estimation. In *IEEE Computer Vision and Pattern Recognition*, 2016.
- [201] M. Norouzi, T. Mikolov, S. Bengio, Y. Singer, J. Shlens, A. Frome, G. S. Corrado, and J. Dean. Zero-shot Learning by Convex Combination of Semantic Embeddings. *arXiv:1312.5650 [cs]*, 2013.
- [202] OpenAI. GPT-4 Technical Report. *arXiv:2303.08774 [cs]*, Mar. 2023. doi: 10.48550/arXiv.2303.08774.
- [203] OpenStreetMap contributors. Planet Dump Retrieved from <https://planet.osm.org>, 2018. URL <https://www.openstreetmap.org>. Last accessed: 2023-09-29.
- [204] K. O’Shea and R. Nash. An Introduction to Convolutional Neural Networks. *arXiv:1511.08458 [cs]*, Dec. 2015.
- [205] M. Pagliardini, M. Jaggi, F. Fleuret, and S. P. Karimireddy. Agree to Disagree: Diversity through Disagreement for Better Transferability. *arXiv:2202.04414 [cs]*, 2022.
- [206] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, pages 8024–8035. Curran Associates, Inc., 2019.
- [207] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011. ISSN 1533-7928.
- [208] J. Pfister, K. Kobs, and A. Hotho. Self-Supervised Multi-Task Pretraining Improves Image Aesthetic Assessment. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 816–825, June 2021. doi: 10.1109/CVPRW53098.2021.00091.
- [209] E. Plaut. From Principal Subspaces to Principal Components with Linear Autoencoders. *arXiv:1804.10253 [stat]*, 2018.
- [210] N. Ponomarenko, O. Ieremeiev, V. Lukin, K. Egiazarian, L. Jin, J. Astola, B. Vozel, K. Chehdi, M. Carli, F. Battisti, and C.-C. J. Kuo. Color Image Database TID2013: Peculiarities and Preliminary Results. In *European Workshop on Visual Information Processing*, volume 30, pages 106–111. IEEE, 2013.

- [211] A. Power, Y. Burda, H. Edwards, I. Babuschkin, and V. Misra. Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets. *arXiv:2201.02177 [cs]*, Jan. 2022.
- [212] Q. Qian, L. Shang, B. Sun, J. Hu, T. Tacoma, H. Li, and R. Jin. SoftTriple Loss: Deep Metric Learning Without Triplet Sampling. In *IEEE International Conference on Computer Vision*, pages 6449–6457, Seoul, Korea (South), Oct. 2019. IEEE. ISBN 978-1-72814-803-8. doi: 10.1109/ICCV.2019.00655.
- [213] X. Qin, Z. Zhang, C. Huang, M. Dehghan, O. Zaiane, and M. Jagersand. U2-Net: Going Deeper with Nested U-Structure for Salient Object Detection. In *Pattern Recognition*, volume 106, page 107404, 2020.
- [214] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning Transferable Visual Models From Natural Language Supervision. In *International Conference on Machine Learning*, pages 8748–8763, Feb. 2021.
- [215] A. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré. Snorkel: Rapid Training Data Creation with Weak Supervision. *The VLDB Journal*, 29(2):709–730, May 2020. ISSN 0949-877X. doi: 10.1007/s00778-019-00552-1.
- [216] A. J. Ratner, C. M. De Sa, S. Wu, D. Selsam, and C. Ré. Data Programming: Creating Large Training Sets, Quickly. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.
- [217] T. Räuker, A. Ho, S. Casper, and D. Hadfield-Menell. Toward Transparent AI: A Survey on Interpreting the Inner Structures of Deep Neural Networks. *arXiv:2207.13243 [cs]*, Sept. 2022. doi: 10.48550/arXiv.2207.13243.
- [218] N. Reimers and I. Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *arXiv:1908.10084 [cs]*, Aug. 2019.
- [219] W. Ren, Y. Li, H. Su, D. Kartchner, C. Mitchell, and C. Zhang. Denoising Multi-Source Weak Supervision for Neural Text Classification. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3739–3754, Online, Nov. 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.334.
- [220] S. Rendle. Evaluation metrics for Item Recommendation under Sampling. *arXiv:1912.02263 [cs]*, 2019.
- [221] M. Ribeiro, S. Singh, and C. Guestrin. “Why Should I Trust You?”: Explaining the Predictions of Any Classifier. In *Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 97–101, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-3020.

Bibliography

- [222] N. Riche, M. Duvinage, M. Mancas, B. Gosselin, and T. Dutoit. Saliency and Human Fixations: State-of-the-Art and Study of Comparison Metrics. In *IEEE International Conference on Computer Vision*, pages 1153–1160, Sydney, Australia, Dec. 2013. IEEE. ISBN 978-1-4799-2840-8. doi: 10.1109/ICCV.2013.147.
- [223] T. Ridnik, E. Ben-Baruch, A. Noy, and L. Zelnik-Manor. ImageNet-21K Pretraining for the Masses. *arXiv:2104.10972 [cs]*, Aug. 2021.
- [224] S. E. Robertson. On Term Selection for Query Expansion. *Journal of Documentation*, 46(4):359–364, 1990.
- [225] Y. Roh, G. Heo, and S. E. Whang. A Survey on Data Collection for Machine Learning: A Big Data - AI Integration Perspective. *IEEE Transactions on Knowledge and Data Engineering*, 33(4):1328–1347, Apr. 2021. ISSN 1558-2191. doi: 10.1109/TKDE.2019.2946162.
- [226] F. Rosenblatt. The Perceptron: A probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, 65(6):386–408, 1958. ISSN 1939-1471, 0033-295X. doi: 10.1037/h0042519.
- [227] S. Rosenthal, A. Ritter, P. Nakov, and V. Stoyanov. SemEval-2014 Task 9: Sentiment Analysis in Twitter. In *International Workshop on Semantic Evaluation*, pages 73–80, Dublin, Ireland, Aug. 2014. Association for Computational Linguistics and Dublin City University.
- [228] S. Rosenthal, P. Nakov, S. Kiritchenko, S. Mohammad, A. Ritter, and V. Stoyanov. SemEval-2015 Task 10: Sentiment Analysis in twitter. In *International Workshop on Semantic Evaluation*, pages 451–463, 2015.
- [229] S. Rosenthal, N. Farra, and P. Nakov. SemEval-2017 Task 4: Sentiment Analysis in Twitter. In *International Workshop on Semantic Evaluation*, pages 502–518, 2017.
- [230] A. S. Ross, M. C. Hughes, and F. Doshi-Velez. Right for the Right Reasons: Training Differentiable Models by Constraining their Explanations. *arXiv:1703.03717 [cs, stat]*, May 2017.
- [231] K. Roth, T. Milbich, S. Sinha, P. Gupta, B. Ommer, and J. P. Cohen. Revisiting Training Strategies and Generalization Performance in Deep Metric Learning. *arXiv:2002.08473 [cs]*, Aug. 2020.
- [232] K. Roth, O. Vinyals, and Z. Akata. Integrating Language Guidance into Vision-based Deep Metric Learning. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 16156–16168, New Orleans, LA, USA, June 2022. IEEE. ISBN 978-1-66546-946-3. doi: 10.1109/CVPR52688.2022.01570.
- [233] Y. Rubner, C. Tomasi, and L. J. Guibas. The Earth Mover’s Distance as a Metric for Image Retrieval. *International Journal of Computer Vision*, 40:99–121, 2000.

- [234] S. Sagawa, A. Raghunathan, P. W. Koh, and P. Liang. An Investigation of why Overparameterization Exacerbates Spurious Correlations. In *Proceedings of the International Conference on Machine Learning*, pages 8346–8356. PMLR, 2020.
- [235] C. Sammut and G. I. Webb. Encyclopedia of Machine Learning. In *Encyclopedia of Machine Learning*. Springer US, Boston, MA, 2010. ISBN 978-0-387-30164-8.
- [236] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, June 2018. doi: 10.1109/cvpr.2018.00474.
- [237] M. J. Schuemie and J. A. Kors. Jane: Suggesting Journals, Finding Experts. *Bioinformatics (Oxford, England)*, 24(5):727–728, 2008.
- [238] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. *International Journal of Computer Vision*, 128(2):336–359, Feb. 2020. ISSN 0920-5691, 1573-1405. doi: 10.1007/s11263-019-01228-7.
- [239] O. Sener and V. Koltun. Multi-Task Learning as Multi-Objective Optimization. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.
- [240] H. Shah, K. Tamuly, A. Raghunathan, P. Jain, and P. Netrapalli. The Pitfalls of Simplicity Bias in Neural Networks. *Advances in Neural Information Processing Systems*, 33:9573–9585, 2020.
- [241] X. Shao, A. Skryagin, W. Stammer, P. Schramowski, and K. Kersting. Right for Better Reasons: Training Differentiable Models by Constraining their Influence Functions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9533–9540, May 2021.
- [242] K. Sheng, W. Dong, C. Ma, X. Mei, F. Huang, and B.-G. Hu. Attention-based Multi-Patch Aggregation for Image Aesthetic Assessment. In *ACM International Conference on Multimedia*, MM '18, pages 879–886, New York, NY, USA, Oct. 2018. Association for Computing Machinery. ISBN 978-1-4503-5665-7. doi: 10.1145/3240508.3240554.
- [243] K. Sheng, W. Dong, M. Chai, G. Wang, P. Zhou, F. Huang, B.-G. Hu, R. Ji, and C. Ma. Revisiting Image Aesthetic Assessment via Self-Supervised Feature Learning. *AAAI Conference on Artificial Intelligence*, 34(04):5709–5716, Apr. 2020. ISSN 2374-3468. doi: 10.1609/aaai.v34i04.6026.
- [244] C. Shorten and T. M. Khoshgoftaar. A Survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, 6(1):60, July 2019. ISSN 2196-1115. doi: 10.1186/s40537-019-0197-0.

Bibliography

- [245] C. Shorten, T. M. Khoshgoftaar, and B. Furht. Text Data Augmentation for Deep Learning. *Journal of Big Data*, 8(1):101, July 2021. ISSN 2196-1115. doi: 10.1186/s40537-021-00492-0.
- [246] R. Shyam and R. Singh. A Taxonomy of Machine Learning Techniques. *Journal of Advancements in Robotics*, 8(3):18–25, Dec. 2021.
- [247] N. C. Silver and W. P. Dunlap. Averaging Correlation Coefficients: Should Fisher’s z Transformation be used? *Journal of Applied Psychology*, 72(1):146, 1987.
- [248] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *arXiv:1312.6034 [cs]*, Apr. 2014.
- [249] V. Singh. Higher Pollution Episode Detection using Image Classification Techniques. *Environmental Modeling & Assessment*, 21(5):591–601, Oct. 2016. ISSN 1573-2967. doi: 10.1007/s10666-015-9497-8.
- [250] D. Smilkov, N. Thorat, B. Kim, F. Viégas, and M. Wattenberg. SmoothGrad: Removing Noise by Adding Noise. *arXiv:1706.03825 [cs, stat]*, June 2017.
- [251] T. Smith, M. Obrist, and P. C. Wright. Live-Streaming Changes the (Video) Game. In P. Paolini, P. Cremonesi, and G. Lekakos, editors, *European Conference on Interactive TV and Video*, pages 131–138. ACM, 2013. ISBN 978-1-4503-1951-5.
- [252] K. Sohn. Improved Deep Metric Learning with Multi-class N-pair Loss Objective. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, pages 1857–1865. Curran Associates, Inc., 2016.
- [253] H. O. Song, Y. Xiang, S. Jegelka, and S. Savarese. Deep Metric Learning via Lifted Structured Feature Embedding. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 4004–4012, Las Vegas, NV, USA, June 2016. IEEE. ISBN 978-1-4673-8851-1. doi: 10.1109/CVPR.2016.434.
- [254] R. Speer, J. Chin, and C. Havasi. ConceptNet 5.5: An Open Multilingual Graph of General Knowledge. *arXiv:1612.03975 [cs]*, Dec. 2018.
- [255] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, Jan. 2014. ISSN 1532-4435.
- [256] M. Steininger, K. Kobs, A. Zehe, F. Lautenschlager, M. Becker, and A. Hotho. MapLUR: Exploring a New Paradigm for Estimating Air Pollution Using Deep Learning on Map Images. *ACM Transactions on Spatial Algorithms and Systems*, 6(3):1–24, May 2020. ISSN 2374-0353, 2374-0361. doi: 10.1145/3380973.
- [257] A. Stylianou, R. Souvenir, and R. Pless. Visualizing Deep Similarity Networks. *arXiv:1901.00536 [cs]*, Jan. 2019.

- [258] S. Sukhbaatar, J. Bruna, M. Paluri, L. Bourdev, and R. Fergus. Training Convolutional Networks with Noisy Labels. *arXiv:1406.2080 [cs]*, June 2014.
- [259] C. Sun, X. Qiu, Y. Xu, and X. Huang. How to Fine-Tune BERT for Text Classification? *arXiv:1905.05583 [cs]*, Feb. 2020.
- [260] M. Sundararajan, A. Taly, and Q. Yan. Axiomatic Attribution for Deep Networks. In *International Conference on Machine Learning*, pages 3319–3328. JMLR. org, 2017.
- [261] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing Properties of Neural Networks. *arXiv:1312.6199 [cs]*, Feb. 2014.
- [262] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going Deeper with Convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.
- [263] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the Inception Architecture for Computer Vision. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, Las Vegas, NV, USA, June 2016. IEEE. ISBN 978-1-4673-8851-1. doi: 10.1109/CVPR.2016.308.
- [264] K. Szyc, T. Walkowiak, and H. Maciejewski. Checking Robustness of Representations Learned by Deep Neural Networks. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, pages 399–414. Springer, 2021.
- [265] H. Talebi and P. Milanfar. NIMA: Neural Image Assessment. *IEEE Transactions on Image Processing*, 27(8):3998–4011, Aug. 2018. ISSN 1057-7149, 1941-0042. doi: 10.1109/TIP.2018.2831899.
- [266] B. Thomee, D. A. Shamma, G. Friedland, B. Elizalde, K. Ni, D. Poland, D. Borth, and L.-J. Li. YFCC100M: The New Data in Multimedia Research. *Communications of the ACM*, 59(2):64–73, 2016.
- [267] M. Tian, S. Yi, H. Li, S. Li, X. Zhang, J. Shi, J. Yan, and X. Wang. Eliminating Background-bias for Robust Person Re-identification. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 5794–5803, Salt Lake City, UT, June 2018. IEEE. ISBN 978-1-5386-6420-9. doi: 10.1109/CVPR.2018.00607.
- [268] T. Tieleman and G. Hinton. Lecture 6.5-RMSProp: Divide the Gradient by a Running Average of its recent Magnitude. *COURSERA: Neural Networks for Machine Learning*, 4(2):26–31, 2012.
- [269] J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Boochoon, and S. Birchfield. Training Deep Networks with Synthetic Data: Bridging the Reality Gap by Domain Randomization. In *IEEE*

Bibliography

- Conference on Computer Vision and Pattern Recognition Workshops*, pages 1082–10828, Salt Lake City, UT, June 2018. IEEE. ISBN 978-1-5386-6100-0. doi: 10.1109/CVPRW.2018.00143.
- [270] L. van der Maaten and G. Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.
- [271] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is All You Need. In *Advances in Neural Information Processing Systems*, NIPS’17, pages 6000–6010, Red Hook, NY, USA, Dec. 2017. Curran Associates Inc. ISBN 978-1-5108-6096-4.
- [272] J. Vig, S. Gehrmann, Y. Belinkov, S. Qian, D. Nevo, Y. Singer, and S. Shieber. Investigating Gender Bias in Language Models Using Causal Mediation Analysis. In *Advances in Neural Information Processing Systems*, volume 33, pages 12388–12401. Curran Associates, Inc., 2020.
- [273] J. von Kügelgen, Y. Sharma, L. Gresele, W. Brendel, B. Schölkopf, M. Besserve, and F. Locatello. Self-Supervised Learning with Data Augmentations Provably Isolates Content from Style. *arXiv:2106.04619 [cs, stat]*, Jan. 2022.
- [274] G. Vrettas and M. Sanderson. Conferences versus Journals in Computer Science. *Journal of the Association for Information Science and Technology*, 66(12):2674–2684, 2015.
- [275] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.
- [276] F. Wang, X. Xiang, J. Cheng, and A. L. Yuille. NormFace: L2 Hypersphere Embedding for Face Verification. *ACM International Conference on Multimedia*, pages 1041–1049, Oct. 2017. doi: 10.1145/3123266.3123359.
- [277] F. Wang, W. Liu, H. Liu, and J. Cheng. Additive Margin Softmax for Face Verification. *IEEE Signal Processing Letters*, 25(7):926–930, July 2018. ISSN 1070-9908, 1558-2361. doi: 10.1109/LSP.2018.2822810.
- [278] H. Wang, Y. Wang, Z. Zhou, X. Ji, D. Gong, J. Zhou, Z. Li, and W. Liu. CosFace: Large Margin Cosine Loss for Deep Face Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 5265–5274, Salt Lake City, UT, June 2018. IEEE. ISBN 978-1-5386-6420-9. doi: 10.1109/CVPR.2018.00552.
- [279] J. Wang, X. Yi, R. Guo, H. Jin, P. Xu, S. Li, X. Wang, X. Guo, C. Li, X. Xu, et al. Milvus: A Purpose-Built Vector Data Management System. In *International Conference on Management of Data*, pages 2614–2627, 2021.
- [280] M. Wang, R. Beelen, T. Bellander, M. Birk, G. Cesaroni, M. Cirach, J. Cyrus, K. de Hoogh, C. Declercq, K. Dimakopoulou, et al. Performance of Multi-City Land Use

- Regression Models for Nitrogen Dioxide and Fine Particles. *Environmental Health Perspectives*, 122(8):843, 2014.
- [281] Q. Wang, Y. Ma, K. Zhao, and Y. Tian. A Comprehensive Survey of Loss Functions in Machine Learning. *The Annals of Data Science*, 9(2):187–212, Apr. 2022. ISSN 2198-5804, 2198-5812. doi: 10.1007/s40745-020-00253-5.
- [282] R. Wang, J. Chen, G. Yu, L. Sun, C. Yu, C. Gao, and N. Sang. Attribute-specific Control Units in StyleGAN for Fine-grained Image Manipulation. In *ACM International Conference on Multimedia*, MM '21, pages 926–934, New York, NY, USA, Oct. 2021. Association for Computing Machinery. ISBN 978-1-4503-8651-7. doi: 10.1145/3474085.3475274.
- [283] W. Wang, V. W. Zheng, H. Yu, and C. Miao. A Survey of Zero-Shot Learning: Settings, Methods, and Applications. *ACM Transactions on Intelligent Systems and Technology*, 10(2):13:1–13:37, Jan. 2019. ISSN 2157-6904. doi: 10.1145/3293318.
- [284] X. Wang, X. Han, W. Huang, D. Dong, and M. R. Scott. Multi-Similarity Loss With General Pair Weighting for Deep Metric Learning. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 5017–5025, Long Beach, CA, USA, June 2019. IEEE. ISBN 978-1-72813-293-8. doi: 10.1109/CVPR.2019.00516.
- [285] X. Wang, Q. Liu, T. Gui, Q. Zhang, Y. Zou, X. Zhou, J. Ye, Y. Zhang, R. Zheng, Z. Pang, Q. Wu, Z. Li, C. Zhang, R. Ma, Z. Fei, R. Cai, J. Zhao, X. Hu, Z. Yan, Y. Tan, Y. Hu, Q. Bian, Z. Liu, S. Qin, B. Zhu, X. Xing, J. Fu, Y. Zhang, M. Peng, X. Zheng, Y. Zhou, Z. Wei, X. Qiu, and X. Huang. TextFlint: Unified Multilingual Robustness Evaluation Toolkit for Natural Language Processing. In *Annual Meeting of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing: System Demonstrations*, pages 347–355, Online, Aug. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-demo.41.
- [286] Y. Wang and M. Izbicki. The Tree Loss: Improving Generalization with Many Classes. In *International Conference on Artificial Intelligence and Statistics*, pages 6121–6133. PMLR, May 2022.
- [287] S. Wei, S. Zou, F. Liao, and w. lang. A Comparison on Data Augmentation Methods Based on Deep Learning for Audio Classification. *Journal of Physics: Conference Series*, 1453(1):012085, Jan. 2020. ISSN 1742-6588, 1742-6596. doi: 10.1088/1742-6596/1453/1/012085.
- [288] K. Q. Weinberger, J. Blitzer, and L. K. Saul. Distance Metric Learning for Large Margin Nearest Neighbor Classification. In Y. Weiss, B. Schölkopf, and J. C. Platt, editors, *Advances in Neural Information Processing Systems*, pages 1473–1480. MIT Press, 2006.

Bibliography

- [289] C. Wen, J. Qian, J. Lin, J. Teng, D. Jayaraman, and Y. Gao. Fighting Fire with Fire: Avoiding DNN Shortcuts through Priming. In *International Conference on Machine Learning*, pages 23723–23750. PMLR, 2022.
- [290] Q. Wen, L. Sun, F. Yang, X. Song, J. Gao, X. Wang, and H. Xu. Time Series Data Augmentation for Deep Learning: A Survey. In *International Joint Conference on Artificial Intelligence*, pages 4653–4660, Aug. 2021. doi: 10.24963/ijcai.2021/631.
- [291] C. Whitehouse, M. Choudhury, and A. F. Aji. LLM-powered Data Augmentation for Enhanced Crosslingual Performance. *arXiv:2305.14288 [cs]*, May 2023. doi: 10.48550/arXiv.2305.14288.
- [292] R. Wightman. PyTorch Image Models, 2019. URL <https://github.com/rwightman/pytorch-image-models>. Last accessed: 2023-09-29.
- [293] F. Wilcoxon. Individual Comparisons by Ranking Methods. *Biometrics Bulletin*, 1(6):80, Dec. 1945. doi: 10.2307/3001968.
- [294] F. Wilcoxon. Individual Comparisons by Ranking Methods. In *Breakthroughs in Statistics*, pages 196–202. Springer, 1992.
- [295] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht. The Marginal Value of Adaptive Gradient Methods in Machine Learning. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.
- [296] S. Wold, K. Esbensen, and P. Geladi. Principal Component Analysis. *Chemometrics and Intelligent Laboratory Systems*, 2(1-3):37–52, 1987.
- [297] K. Wolf, J. Cyrus, T. Hrciníková, J. Gu, T. Kusch, R. Hampel, A. Schneider, and A. Peters. Land Use Regression Modeling of Ultrafine Particles, Ozone, Nitrogen Oxides and Markers of Particulate Matter Pollution in Augsburg, Germany. *Science of the Total Environment*, 579:1531–1540, 2017.
- [298] C. Wu, M. Tygert, and Y. LeCun. Hierarchical Loss for Classification. *arXiv:1709.01062 [cs, stat]*, Sept. 2017.
- [299] C.-Y. Wu, R. Manmatha, A. J. Smola, and P. Krahenbuhl. Sampling Matters in Deep Embedding Learning. In *IEEE International Conference on Computer Vision*, pages 2859–2867, Venice, Oct. 2017. IEEE. ISBN 978-1-5386-1032-9. doi: 10.1109/ICCV.2017.309.
- [300] J. Wu, J. Li, J. Peng, W. Li, G. Xu, and C. Dong. Applying Land Use Regression Model to Estimate Spatial Variation of PM_{2.5} in Beijing, China. *Environmental Science and Pollution Research*, 22(9):7045–7061, 2015.
- [301] Q. Wu, H. Li, K. N. Ngan, B. Zeng, and M. Gabbouj. No Reference Image Quality Metric via Distortion Identification and Multi-Channel Label Transfer. In *IEEE International Symposium on Circuits and Systems*. IEEE, June 2014. doi: 10.1109/iscas.2014.6865189.

- [302] Y. Wu, K. Chen, T. Zhang, Y. Hui, T. Berg-Kirkpatrick, and S. Dubnov. Large-scale Contrastive Language-Audio Pretraining with Feature Fusion and Keyword-to-Caption Augmentation. *arXiv:2211.06687 [cs, eess]*, Nov. 2022. doi: 10.48550/arXiv.2211.06687.
- [303] K. Xiao, L. Engstrom, A. Ilyas, and A. Madry. Noise or Signal: The Role of Image Backgrounds in Object Recognition. *arXiv:2006.09994 [cs]*, 2020.
- [304] J. Xie, X. Wang, Y. Liu, and Y. Bai. Autoencoder-based Deep Belief Regression Network for Air Particulate Matter Concentration Forecasting. *Journal of Intelligent & Fuzzy Systems*, 34(6):3475–3486, 2018.
- [305] S. Xu, S. Venugopalan, and M. Sundararajan. Attribution in Scale and Space. *arXiv:2004.03383 [cs]*, Apr. 2020.
- [306] X. Xu, H. Cao, Y. Yang, E. Yang, and C. Deng. Zero-shot Metric Learning. In *International Joint Conference on Artificial Intelligence*, pages 3996–4002, 2019.
- [307] Y.-Y. Yang and K. Chaudhuri. Understanding Rare Spurious Correlations in Neural Networks. *arXiv:2202.05189 [cs]*, 2022.
- [308] X. Yao, Y. Zheng, X. Yang, and Z. Yang. NLP From Scratch Without Large-Scale Pretraining: A Simple and Efficient Framework. *arXiv:2111.04130 [cs]*, July 2022.
- [309] A. Yeh. More Accurate Tests for the Statistical Significance of Result Differences. In *Conference on Computational Linguistics*, pages 947–953. Association for Computational Linguistics, 2000.
- [310] T. Yuan, W. Deng, J. Tang, Y. Tang, and B. Chen. Signal-To-Noise Ratio: A Robust Distance Metric for Deep Metric Learning. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 4810–4819, Long Beach, CA, USA, June 2019. IEEE. ISBN 978-1-72813-293-8. doi: 10.1109/CVPR.2019.00495.
- [311] H. Zeng, Z. Cao, L. Zhang, and A. C. Bovik. A Unified Probabilistic Formulation of Image Aesthetic Assessment. *IEEE Transactions on Image Processing*, 29:1548–1561, 2020. ISSN 1057-7149, 1941-0042. doi: 10.1109/TIP.2019.2941778.
- [312] A. Zhai and H.-Y. Wu. Classification is a Strong Baseline for Deep Metric Learning. *arXiv:1811.12649 [cs]*, Aug. 2019.
- [313] C. Zhang, J. Yan, C. Li, H. Wu, and R. Bie. End-to-End Learning for Image-based Air Quality Level Estimation. *Machine Vision and Applications*, 29(4):601–615, 2018.
- [314] J. Zhang, C.-Y. Hsieh, Y. Yu, C. Zhang, and A. Ratner. A Survey on Programmatic Weak Supervision. *arXiv:2202.05433 [cs, stat]*, Feb. 2022.

Bibliography

- [315] Y. Zhang and D. M. Chandler. Opinion-Unaware Blind Quality Assessment of Multiply and Singly Distorted Images via Distortion Parameter Estimation. *IEEE Transactions on Image Processing*, 27(11):5433–5448, Nov. 2018. doi: 10.1109/tip.2018.2857413.
- [316] Y. Zhang and B. Wallace. A Sensitivity Analysis of (and Practitioners’ Guide to) Convolutional Neural Networks for Sentence Classification. In *International Joint Conference on Natural Language Processing*, volume 1, pages 253–263, 2017.
- [317] Z. Zhang and M. Sabuncu. Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels. In *Advances in Neural Information Processing Systems*, 2018.
- [318] Z. Zhang, Y. Song, and H. Qi. Age Progression/Regression by Conditional Adversarial Autoencoder. In *IEEE Computer Vision and Pattern Recognition*, 2017.
- [319] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Learning Deep Features for Discriminative Localization. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2921–2929, June 2016. doi: 10.1109/CVPR.2016.319.
- [320] B. Zhou, D. Bau, A. Oliva, and A. Torralba. Interpreting Deep Visual Representations via Network Dissection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(9):2131–2145, Sept. 2019. ISSN 1939-3539. doi: 10.1109/TPAMI.2018.2858759.
- [321] K. Zhou, J. Yang, C. C. Loy, and Z. Liu. Learning to Prompt for Vision-Language Models. *International Journal of Computer Vision*, 130(9):2337–2348, Sept. 2022. ISSN 1573-1405. doi: 10.1007/s11263-022-01653-1.
- [322] C. Zhu, R. Ni, Z. Xu, K. Kong, W. R. Huang, and T. Goldstein. GradInit: Learning to Initialize Neural Networks for Stable and Efficient Training. In *Advances in Neural Information Processing Systems*, volume 34, pages 16410–16422. Curran Associates, Inc., 2021.
- [323] S. Zhu, T. Yang, and C. Chen. Visual Explanation for Deep Metric Learning. *arXiv:1909.12977 [cs]*, July 2020.

A. Jensen-Shannon Divergence (JSD) Tables for Deep Metric Learning (DML) Attribution Method

		Ranking								Classification						
		Contrastive	Triplet	NTXent	Margin	Margin/class	FastAP	SNR Con.	MS	MS+Miner	ProxyNCA	N. Softmax	CosFace	ArcFace	SoftTriple	None
Ranking	Contrastive		2±0	1±0	2±0	2±0	2±0	2±0	1±0	1±0	2±0	2±0	2±0	2±0	2±0	3±1
	Triplet	2±0		2±0	2±0	2±0	2±0	2±0	2±0	2±0	2±0	2±0	2±0	2±0	2±0	3±1
	NTXent	1±0	2±0		2±0	1±0	2±0	1±0	2±0	2±0	1±0	2±0	2±0	2±0	3±1	
	Margin	2±0	2±0	2±0		2±0	2±0	2±0	2±0	2±0	2±0	2±0	2±0	2±0	3±1	
	Margin/class	2±0	2±0	1±0	2±0		2±0	2±0	2±0	2±0	2±0	2±0	2±0	2±0	3±1	
	FastAP	2±0	2±0	2±0	2±0	2±0		2±0	2±0	2±0	2±0	2±0	2±0	2±0	2±0	4±1
	SNR Con.	2±0	2±0	1±0	2±0	2±0	2±0		1±0	2±0	2±0	1±0	2±0	2±0	2±0	3±1
	MS	1±0	2±0	1±0	2±0	2±0	2±0	1±0		1±0	2±0	1±0	2±0	2±0	2±0	3±1
	MS+Miner	1±0	2±0	2±0	2±0	2±0	2±0	2±0	1±0		2±0	1±0	2±0	2±0	1±0	3±1
	ProxyNCA	2±0	2±0	2±0	2±0	2±0	2±0	2±0	1±0	2±0		1±0	1±0	2±0	1±0	3±1
Classif.	N. Softmax	2±0	2±0	1±0	2±0	2±0	2±0	1±0	1±0	1±0		1±0	1±0	2±0	1±0	3±1
	CosFace	2±0	2±0	2±0	2±0	2±0	2±0	2±0	2±0	1±0	1±0		2±0	1±0	3±1	
	ArcFace	2±0	2±0	2±0	2±0	2±0	2±0	2±0	2±0	2±0	2±0	2±0		1±0	3±1	
	SoftTriple	2±0	2±0	2±0	2±0	2±0	2±0	2±0	1±0	1±0	1±0	1±0		1±0	3±1	
	None	3±1	3±1	3±1	3±1	3±1	4±1	3±1	3±1	3±1	3±1	3±1	3±1	3±1		3±1

Table A.1.: JSD between all loss functions on the Cars196 dataset. All values are in percent.

		Ranking								Classification							
		Contrastive	Triplet	NTXent	Margin	Margin/class	FastAP	SNR Con.	MS	MS+Miner	ProxyNCA	N. Softmax	CosFace	ArcFace	SoftTriple	None	
Ranking	Contrastive		2±0	1±0	2±0	2±0	2±0	2±0	1±0	2±0	2±0	2±0	2±0	2±0	2±0	2±0	
	Triplet	2±0		1±0	1±0	1±0	2±0	2±0	1±0	1±0	1±0	1±0	2±0	2±0	2±0	2±0	
	NTXent	2±0	1±0		1±0	1±0	1±0	1±0	1±0	1±0	1±0	1±0	2±0	1±0	2±0	2±0	
	Margin	2±0	1±0	1±0		1±0	2±0	1±0	1±0	1±0	1±0	2±0	2±0	2±0	2±0	2±0	
	Margin/class	2±0	1±0	1±0	1±0		1±0	1±0	1±0	1±0	1±0	1±0	2±0	1±0	2±0	2±0	
	FastAP	2±0	2±0	1±0	2±0	1±0		1±0	1±0	1±0	2±0	1±0	2±0	2±0	2±0	2±1	
	SNR Con.	2±0	2±0	1±0	1±0	1±0	1±0		1±0	1±0	1±0	1±0	2±0	2±0	1±0	2±0	
	MS	1±0	1±0	1±0	1±0	1±0	1±0	1±0		1±0	1±0	1±0	1±0	1±0	1±0	2±0	
	MS+Miner	2±0	1±0	1±0	1±0	1±0	1±0	1±0	1±0		1±0	1±0	2±0	1±0	1±0	2±0	
	ProxyNCA	2±0	1±0	1±0	1±0	1±0	2±0	1±0	1±0	1±0	1±0		1±0	1±0	1±0	2±0	
Classif.	N. Softmax	2±0	1±0	1±0	1±0	1±0	1±0	1±0	1±0	1±0	1±0		2±0	1±0	1±0	2±0	
	CosFace	2±0	2±0	2±0	2±0	2±0	2±0	2±0	1±0	2±0	2±0	2±0		1±0	1±0	2±0	
	ArcFace	2±0	2±0	1±0	2±0	1±0	2±0	2±0	1±0	1±0	1±0	1±0	1±0		1±0	2±0	
	SoftTriple	2±0	2±0	2±0	2±0	2±0	2±0	2±0	1±0	1±0	1±0	1±0	1±0		1±0	2±0	
	None	2±0	2±0	2±0	2±0	2±0	2±1	2±0	2±0	2±0	2±0	2±0	2±0	2±0	2±0		2±0

Table A.2.: JSD between all loss functions on the CUB200 dataset. All values are in percent.

In Chapter 9, we compared the attribution maps for all images in a dataset between two loss functions. We mainly compared them using the mean correlation. We also calculate the JSD between the attribution maps of the two loss functions, after normalizing them to have a sum of 1, i.e., to resemble a probability distribution. Tables A.1 to A.3 show the mean and standard deviation for the three datasets we investigate. All values are around 0.02 ± 0.01 and show similar tendencies as the correlation, i.e., the Stanford Online Products (SOP) dataset shows differences between loss functions of different types (ranking or classification).

A. JSD Tables for DML Attribution Method

		Ranking									Classification					
		Contrastive	Triplet	NTXent	Margin	Margin/class	FastAP	SNR Con.	MS	MS+Miner	ProxyNCA	N. Softmax	CosFace	ArcFace	SoftTriple	None
Ranking	Contrastive		2±0	2±0	2±0	2±0	2±0	2±0	2±0	2±0	3±1	3±0	3±1	3±1	3±1	4±1
	Triplet	2±0		2±0	2±0	2±0	2±0	2±0	2±0	2±0	3±1	3±0	3±1	3±1	3±1	3±1
	NTXent	2±0	2±0		2±0	2±0	2±0	2±0	2±0	2±0	2±0	2±0	3±0	3±0	2±0	3±1
	Margin	2±0	2±0	2±0		2±0	2±0	2±0	2±0	2±0	3±0	3±0	3±1	3±1	3±1	3±1
	Margin/class	2±0	2±0	2±0	2±0		2±0	2±0	2±0	2±0	3±0	3±0	3±1	3±1	3±1	3±1
	FastAP	2±0	2±0	2±0	2±0	2±0		2±0	2±0	2±0	3±1	3±0	3±1	3±1	3±1	3±1
	SNR Con.	2±0	2±0	2±0	2±0	2±0	2±0		2±0	2±0	3±1	3±1	3±1	3±1	3±1	4±1
	MS	2±0	2±0	2±0	2±0	2±0	2±0	2±0		2±0	3±0	2±0	3±0	3±0	3±0	3±1
	MS+Miner	2±0	2±0	2±0	2±0	2±0	2±0	2±0	2±0		2±0	2±0	3±0	3±0	3±0	3±1
	Classif.	ProxyNCA	3±1	3±1	2±0	3±0	3±0	3±1	3±1	3±0	2±0		2±0	2±0	2±0	2±0
N. Softmax		3±0	3±0	2±0	3±0	3±0	3±0	3±1	2±0	2±0	2±0	2±0	2±0	2±0	2±0	3±1
CosFace		3±1	3±1	3±0	3±1	3±1	3±1	3±1	3±0	3±0	2±0	2±0	2±0	2±0	2±0	3±0
ArcFace		3±1	3±1	3±0	3±1	3±1	3±1	3±1	3±0	3±0	2±0	2±0	2±0	2±0	2±0	3±1
SoftTriple		3±1	3±1	2±0	3±1	3±1	3±1	3±1	3±0	3±0	2±0	2±0	2±0	2±0	2±0	3±0
None		4±1	3±1	3±1	3±1	3±1	3±1	4±1	3±1	3±1	3±1	3±1	3±0	3±1	3±0	

Table A.3.: JSD between all loss functions on the Stanford Online Products (SOP) dataset. All values are in percent.

B. InDiReCT Full Results

In Chapter 13, we evaluated our InDiReCT technique on five datasets and thirteen similarity definitions. There, we used Mean Average Precision at R (MAP@R) and Precision at 1 (Prec@1) as evaluation metrics. In Table B.1, we show more evaluation metric results for our experiments. Namely, these are MAP@R [193], Prec@1, R-Precision (R-Prec), Adjusted Mutual Information (AMI), Normalized Mutual Information (NMI), Mean Average Precision (MAP), and Mean Reciprocal Rank (MRR). For an overview of different evaluation metrics in the context of DML, we refer the reader to the appendix of the paper by Roth et al. [231]. For more technical details, see this documentation¹. The results are very similar as for the original two metrics.

¹https://kevinmusgrave.github.io/pytorch-metric-learning/accuracy_calculation/#explanations-of-the-default-accuracy-metrics (last accessed: 2023-02-10)

B. InDiReCT Full Results

Table B.1.: Full results for our experiments on five datasets and thirteen similarity notions in Chapter 13. The Oracle baseline is given for reference.

		Random	CLIP	InDiReCT	Rand. trans.	PCA	LAE	AE	Oracle
Car Model	MAP@R	0.033 ± 0.001	0.435	0.574 ± 0.002	0.391 ± 0.016	0.562 ± 0.001	0.526 ± 0.005	0.395 ± 0.044	1.000 ± 0.000
	Prec@1	0.175 ± 0.009	0.954	0.964 ± 0.000	0.934 ± 0.005	0.966 ± 0.001	0.959 ± 0.005	0.887 ± 0.036	1.000 ± 0.000
	R-Prec	0.167 ± 0.001	0.548	0.662 ± 0.001	0.510 ± 0.014	0.653 ± 0.001	0.624 ± 0.004	0.517 ± 0.036	1.000 ± 0.000
	AMI	-0.000 ± 0.002	0.623	0.737 ± 0.006	0.559 ± 0.086	0.730 ± 0.004	0.713 ± 0.010	0.539 ± 0.060	0.896 ± 0.000
	NMI	0.007 ± 0.002	0.626	0.738 ± 0.006	0.562 ± 0.085	0.732 ± 0.004	0.715 ± 0.010	0.542 ± 0.059	0.897 ± 0.000
	MAP	0.172 ± 0.001	0.591	0.716 ± 0.002	0.546 ± 0.016	0.706 ± 0.001	0.675 ± 0.005	0.551 ± 0.041	1.000 ± 0.000
	MRR	0.367 ± 0.009	0.974	0.980 ± 0.000	0.961 ± 0.003	0.981 ± 0.000	0.977 ± 0.003	0.928 ± 0.024	1.000 ± 0.000
	MAP@R	0.050 ± 0.001	0.062	0.091 ± 0.001	0.061 ± 0.001	—	0.073 ± 0.002	0.086 ± 0.004	0.579 ± 0.009
	Prec@1	0.175 ± 0.008	0.276	0.314 ± 0.005	0.263 ± 0.013	—	0.294 ± 0.009	0.302 ± 0.013	0.793 ± 0.008
Synthetic Cars	R-Prec	0.174 ± 0.001	0.198	0.250 ± 0.002	0.196 ± 0.002	—	0.218 ± 0.001	0.238 ± 0.004	0.676 ± 0.007
	AMI	-0.003 ± 0.004	0.029	0.170 ± 0.003	0.027 ± 0.008	—	0.073 ± 0.002	0.156 ± 0.016	0.629 ± 0.009
	NMI	0.058 ± 0.004	0.088	0.220 ± 0.003	0.086 ± 0.008	—	0.129 ± 0.002	0.207 ± 0.015	0.652 ± 0.008
	MAP	0.179 ± 0.001	0.197	0.247 ± 0.002	0.196 ± 0.002	—	0.213 ± 0.001	0.236 ± 0.005	0.712 ± 0.007
	MRR	0.336 ± 0.007	0.451	0.493 ± 0.003	0.439 ± 0.014	—	0.474 ± 0.008	0.477 ± 0.011	0.859 ± 0.006
	MAP@R	0.054 ± 0.000	0.062	0.071 ± 0.000	0.061 ± 0.002	—	0.063 ± 0.002	0.061 ± 0.002	0.740 ± 0.009
	Prec@1	0.194 ± 0.011	0.270	0.283 ± 0.003	0.266 ± 0.011	—	0.283 ± 0.007	0.216 ± 0.013	0.889 ± 0.004
	R-Prec	0.183 ± 0.001	0.200	0.218 ± 0.001	0.199 ± 0.003	—	0.204 ± 0.003	0.197 ± 0.004	0.805 ± 0.007
	AMI	0.004 ± 0.003	0.017	0.089 ± 0.003	0.025 ± 0.012	—	0.039 ± 0.006	0.048 ± 0.018	0.686 ± 0.008
Background Color	NMI	0.065 ± 0.002	0.076	0.144 ± 0.003	0.084 ± 0.011	—	0.097 ± 0.006	0.106 ± 0.017	0.705 ± 0.008
	MAP	0.188 ± 0.000	0.203	0.218 ± 0.000	0.202 ± 0.002	—	0.205 ± 0.002	0.200 ± 0.004	0.831 ± 0.007
	MRR	0.356 ± 0.007	0.444	0.462 ± 0.003	0.439 ± 0.008	—	0.453 ± 0.003	0.391 ± 0.013	0.920 ± 0.003
	MAP@R	0.001 ± 0.000	0.235	0.374 ± 0.000	0.192 ± 0.003	0.375 ± 0.001	0.332 ± 0.002	0.200 ± 0.058	0.418 ± 0.000
	Prec@1	0.011 ± 0.001	0.780	0.844 ± 0.001	0.729 ± 0.005	0.842 ± 0.001	0.824 ± 0.002	0.638 ± 0.081	0.766 ± 0.001
	R-Prec	0.010 ± 0.000	0.354	0.486 ± 0.000	0.309 ± 0.005	0.487 ± 0.001	0.450 ± 0.002	0.326 ± 0.058	0.545 ± 0.000
	AMI	-0.000 ± 0.001	0.634	0.766 ± 0.002	0.597 ± 0.010	0.771 ± 0.002	0.738 ± 0.008	0.606 ± 0.073	0.803 ± 0.000
	NMI	0.142 ± 0.001	0.685	0.738 ± 0.002	0.653 ± 0.008	0.803 ± 0.002	0.774 ± 0.007	0.661 ± 0.062	0.831 ± 0.000
	MAP	0.011 ± 0.000	0.335	0.501 ± 0.000	0.281 ± 0.005	0.504 ± 0.000	0.456 ± 0.003	0.305 ± 0.070	0.573 ± 0.000
Cars196	MRR	0.047 ± 0.002	0.853	0.898 ± 0.000	0.815 ± 0.004	0.897 ± 0.000	0.885 ± 0.001	0.745 ± 0.063	0.844 ± 0.000
	MAP@R	0.005 ± 0.000	0.244	0.336 ± 0.001	0.212 ± 0.004	—	0.242 ± 0.004	0.180 ± 0.022	0.514 ± 0.000
	Prec@1	0.054 ± 0.003	0.890	0.905 ± 0.001	0.847 ± 0.008	—	0.855 ± 0.003	0.631 ± 0.039	0.840 ± 0.001
	R-Prec	0.054 ± 0.000	0.363	0.445 ± 0.001	0.333 ± 0.004	—	0.362 ± 0.004	0.309 ± 0.021	0.622 ± 0.000
	AMI	0.001 ± 0.001	0.544	0.631 ± 0.002	0.509 ± 0.014	—	0.535 ± 0.008	0.436 ± 0.026	0.725 ± 0.001
	NMI	0.023 ± 0.001	0.555	0.640 ± 0.002	0.520 ± 0.013	—	0.546 ± 0.008	0.449 ± 0.026	0.732 ± 0.001
	MAP	0.055 ± 0.000	0.358	0.461 ± 0.001	0.321 ± 0.005	—	0.355 ± 0.005	0.293 ± 0.024	0.655 ± 0.000
	MRR	0.155 ± 0.002	0.928	0.938 ± 0.001	0.899 ± 0.005	—	0.904 ± 0.003	0.737 ± 0.030	0.891 ± 0.000
	MAP@R	0.035 ± 0.000	0.251	0.361 ± 0.003	0.221 ± 0.008	—	0.277 ± 0.006	0.244 ± 0.016	0.738 ± 0.000
Manufacturer	Prec@1	0.173 ± 0.004	0.911	0.907 ± 0.002	0.883 ± 0.005	—	0.891 ± 0.004	0.632 ± 0.031	0.891 ± 0.000
	R-Prec	0.171 ± 0.000	0.407	0.509 ± 0.003	0.381 ± 0.008	—	0.437 ± 0.006	0.420 ± 0.015	0.802 ± 0.000
	AMI	-0.000 ± 0.000	0.371	0.479 ± 0.012	0.317 ± 0.024	—	0.409 ± 0.011	0.390 ± 0.032	0.744 ± 0.001
	NMI	0.001 ± 0.000	0.372	0.480 ± 0.012	0.318 ± 0.024	—	0.410 ± 0.011	0.391 ± 0.032	0.744 ± 0.001
	MAP	0.172 ± 0.000	0.413	0.531 ± 0.003	0.383 ± 0.008	—	0.446 ± 0.006	0.421 ± 0.017	0.844 ± 0.000
	MRR	0.356 ± 0.003	0.946	0.942 ± 0.001	0.928 ± 0.003	—	0.933 ± 0.002	0.753 ± 0.022	0.929 ± 0.000
	MAP@R	0.001 ± 0.000	0.180	0.265 ± 0.000	0.152 ± 0.003	—	0.188 ± 0.002	0.151 ± 0.019	0.341 ± 0.000
	Prec@1	0.012 ± 0.001	0.582	0.653 ± 0.001	0.526 ± 0.003	—	0.581 ± 0.005	0.444 ± 0.036	0.659 ± 0.002
	R-Prec	0.013 ± 0.000	0.297	0.386 ± 0.000	0.265 ± 0.004	—	0.306 ± 0.002	0.261 ± 0.022	0.474 ± 0.000
CUB200	AMI	0.000 ± 0.002	0.562	0.659 ± 0.003	0.520 ± 0.009	—	0.578 ± 0.010	0.483 ± 0.024	0.736 ± 0.002
	NMI	0.160 ± 0.002	0.627	0.711 ± 0.002	0.593 ± 0.007	—	0.642 ± 0.008	0.564 ± 0.020	0.777 ± 0.002
	MAP	0.015 ± 0.000	0.268	0.379 ± 0.000	0.235 ± 0.004	—	0.282 ± 0.003	0.241 ± 0.023	0.488 ± 0.000
	MRR	0.055 ± 0.001	0.704	0.758 ± 0.001	0.656 ± 0.002	—	0.702 ± 0.003	0.579 ± 0.033	0.758 ± 0.001
	MAP@R	0.023 ± 0.000	0.125	0.187 ± 0.001	0.113 ± 0.004	—	0.133 ± 0.003	0.169 ± 0.018	0.322 ± 0.001
	Prec@1	0.111 ± 0.004	0.452	0.509 ± 0.002	0.430 ± 0.006	—	0.455 ± 0.005	0.445 ± 0.024	0.558 ± 0.006
	R-Prec	0.109 ± 0.000	0.247	0.322 ± 0.001	0.230 ± 0.003	—	0.256 ± 0.003	0.302 ± 0.020	0.449 ± 0.001
	AMI	-0.001 ± 0.002	0.239	0.350 ± 0.003	0.228 ± 0.010	—	0.266 ± 0.009	0.297 ± 0.027	0.439 ± 0.001
	NMI	0.049 ± 0.002	0.276	0.383 ± 0.003	0.266 ± 0.009	—	0.303 ± 0.009	0.333 ± 0.026	0.467 ± 0.001
Clothing Category	MAP	0.111 ± 0.000	0.226	0.307 ± 0.001	0.213 ± 0.003	—	0.238 ± 0.004	0.290 ± 0.020	0.449 ± 0.001
	MRR	0.242 ± 0.004	0.588	0.631 ± 0.002	0.565 ± 0.004	—	0.585 ± 0.004	0.577 ± 0.022	0.668 ± 0.003
	MAP@R	0.118 ± 0.000	0.187	0.330 ± 0.004	0.112 ± 0.004	—	0.222 ± 0.005	0.163 ± 0.007	0.661 ± 0.001
	Prec@1	0.296 ± 0.007	0.602	0.668 ± 0.003	0.433 ± 0.005	—	0.612 ± 0.007	0.438 ± 0.017	0.806 ± 0.003
	R-Prec	0.294 ± 0.000	0.358	0.480 ± 0.004	0.229 ± 0.004	—	0.388 ± 0.004	0.296 ± 0.008	0.743 ± 0.000
	AMI	0.000 ± 0.000	0.081	0.305 ± 0.014	0.224 ± 0.005	—	0.143 ± 0.012	0.295 ± 0.014	0.551 ± 0.001
	NMI	0.003 ± 0.000	0.083	0.307 ± 0.014	0.262 ± 0.004	—	0.145 ± 0.012	0.330 ± 0.013	0.553 ± 0.001
	MAP	0.295 ± 0.000	0.363	0.496 ± 0.004	0.211 ± 0.004	—	0.395 ± 0.005	0.282 ± 0.009	0.767 ± 0.000
	MRR	0.479 ± 0.006	0.723	0.768 ± 0.001	0.568 ± 0.003	—	0.728 ± 0.004	0.573 ± 0.016	0.865 ± 0.002
DeepFashion	MAP@R	0.324 ± 0.000	0.340	0.377 ± 0.002	0.108 ± 0.003	—	0.356 ± 0.003	0.172 ± 0.006	0.642 ± 0.003
	Prec@1	0.494 ± 0.006	0.645	0.661 ± 0.006	0.426 ± 0.007	—	0.650 ± 0.006	0.447 ± 0.019	0.778 ± 0.004
	R-Prec	0.498 ± 0.000	0.526	0.560 ± 0.002	0.224 ± 0.004	—	0.539 ± 0.002	0.307 ± 0.005	0.735 ± 0.002
	AMI	0.000 ± 0.000	0.049	0.119 ± 0.004	0.219 ± 0.012	—	0.079 ± 0.012	0.302 ± 0.010	0.403 ± 0.006
	NMI	0.003 ± 0.000	0.051	0.121 ± 0.004	0.257 ± 0.011	—	0.081 ± 0.012	0.337 ± 0.010	0.405 ± 0.006
	MAP	0.499 ± 0.000	0.524	0.556 ± 0.002	0.208 ± 0.004	—	0.536 ± 0.003	0.294 ± 0.004	0.746 ± 0.002
	MRR	0.636 ± 0.004	0.764	0.775 ± 0.003	0.564 ± 0.005	—	0.767 ± 0.004	0.580 ± 0.015	0.848 ± 0.003
	MAP@R	0.518 ± 0.000	0.533	0.539 ± 0.004	0.111 ± 0.010	—	0.534 ± 0.003	0.161 ± 0.018	0.820 ± 0.001
	Prec@1	0.666 ± 0.006	0.771	0.765 ± 0.004	0.431 ± 0.005	—	0.767 ± 0.007	0.429 ± 0.019	0.878 ± 0.006
Fit	R-Prec	0.666 ± 0.000	0.675	0.680 ± 0.001	0.227 ± 0.009	—	0.677 ± 0.001	0.294 ± 0.017	0.871 ± 0.001
	AMI	-0.000 ± 0.000	0.002	0.003 ± 0.001	0.217 ± 0.008	—	0.013 ± 0.004	0.284 ± 0.017	0.376 ± 0.002
	NMI	0.000 ± 0.000	0.002	0.004 ± 0.001	0.255 ± 0.008	—	0.013 ± 0.004	0.320 ± 0.016	0.377 ± 0.002
	MAP	0.667 ± 0.000	0.685	0.689 ± 0.002	0.211 ± 0.009	—	0.685 ± 0.002	0.320 ± 0.016	0.879 ± 0.001
	MRR	0.772 ± 0.005	0.850	0.846 ± 0.003	0.566 ± 0.003	—	0.845 ± 0.003	0.565 ± 0.017	0.919 ± 0.003
	MAP@R	0.041 ± 0.000	0.114	0.149 ± 0.000	0.091 ± 0.003	—	0.084 ± 0.001	0.098 ± 0.024	0.196 ± 0.001
	Prec@1	0.175 ± 0.004	0.418	0.440 ± 0.002	0.381 ± 0.007	—	0.366 ± 0.004	0.333 ± 0.030	0.432 ± 0.007
	R-Prec	0.174 ± 0.000	0.273	0.306 ± 0.000	0.246 ± 0.003	—	0.237 ± 0.001	0.248 ± 0.031	0.364 ± 0.001
	AMI	0.000 ± 0.000	0.186	0.196 ± 0.003	0.150 ± 0.004	—	0.101 ± 0.007	0.107 ± 0.044	0.254 ± 0.001
Movie Posters	NMI	0.013 ± 0.000	0.196	0.206 ± 0.003	0.160 ± 0.004	—	0.112 ± 0.007	0.118 ± 0.043	0.263 ± 0.001

C. Contribution Statement

Multiple publications have been written and published during the course of this thesis. The following list contains all publications that have been written by the author of this thesis and that contributed to the contents of this thesis. Note that not all contents of these publications are included in this thesis. We state what parts of the publications are contributing to this thesis. The list is sorted by the occurrence of the publications in the thesis. For each publication, we state the contributions of each coauthor, abbreviated by their initials.

M. Steininger, K. Kobs, A. Zehe, F. Lautenschlager, M. Becker, and A. Hotho. MapLUR: Exploring a New Paradigm for Estimating Air Pollution Using Deep Learning on Map Images. *ACM Transactions on Spatial Algorithms and Systems*, 6 (3):1–24, May 2020. ISSN 2374-0353, 2374-0361. doi: 10.1145/3380973 [256]

M.S., F.L., and M.B. conceived the methodology for MapLUR. M.S. designed and implemented the MapLUR model, conceived the experiments as well as the analysis using guided backpropagation and carried them out. K.K. contributed the idea and implementation of the analysis using artificial images and wrote the accompanying section in the publication. F.L. generated features for conventional ML models. M.S. designed visualizations with contributions from K.K. for the publication. M.S., K.K., A.Z., F.L., M.B., and A.H. analyzed and discussed the results. M.S. wrote the publication with substantial contributions from A.Z. and M.B. as well as contributions from K.K., F.L., and A.H.

The use of generated map images for analysis purposes, which was developed and executed by K.K., is the main contribution of this publication to this thesis and is described in Chapter 6. The main idea of MapLUR and its results are given in this thesis to put the analysis contribution in context.

K. Kobs, M. Steininger, A. Dulny, and A. Hotho. Do Different Deep Metric Learning Losses Lead to Similar Learned Features? In *IEEE International Conference on Computer Vision*, pages 10644–10654, 2021 [134]

K.K. developed the ideas for the methodology, designed the experiments, and implemented them. M.S., and A.D. contributed to the discussion for the development of the Normalized R-Precision metric based on K.K.’s idea. K.K., M.S., A.D., and A.H. analyzed and discussed the results. K.K. wrote the publication with substantial contributions from M.S. as well as contributions from A.D. and A.H.

The methodology and experiment results, which are mainly developed and executed by K.K., are the main contributions of this publication to this thesis and are described in

C. Contribution Statement

Chapter 7 and Chapter 9.

K. Kobs, T. Koopmann, A. Zehe, D. Fernes, P. Krop, and A. Hotho. Where to Submit? Helping Researchers to Choose the Right Venue. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 878–883, Online, Nov. 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.78 [131]

K.K., T.K., and A.Z. had the idea for the task and methodology and guided the implementation of the experiments. K.K. had the idea of using Integrated Gradients for the analysis and guided its implementation. T.K. created the dataset. D.F. and P.K. implemented the experiments. All authors analyzed and discussed the results. K.K. and T.K. implemented the accompanying website with code written by D.F. and P.K. K.K. wrote the paper with substantial contributions from T.K. and A.Z. as well as contributions from D.F., P.K., and A.H.

This publication is based on the results of a student project. For the publication, the experiments were extended by another dataset. The model analysis using Integrated Gradients was not part of the student project. Thus, the main contribution of this publication to this thesis is not part of any other examination work. Also, the analysis of the model was extended by K.K. for this thesis. The introduction of the task, model, and dataset is given in this thesis to put the analysis contribution in context. The contributions are mainly described in Chapter 8.

K. Kobs and A. Hotho. On Background Bias in Deep Metric Learning. In *International Conference on Machine Vision*, volume 12701, pages 331–338. SPIE, June 2023. doi: 10.1117/12.2679270 [130]

K.K. developed the methodology, designed the experiments, and implemented them. K.K. and A.H. analyzed and discussed the results. K.K. wrote the publication with contributions from A.H. The full publication contributes to the thesis and is mainly described in Chapter 11.

S. Hentschel, K. Kobs, and A. Hotho. CLIP knows Image Aesthetics. *Frontiers in Artificial Intelligence*, 5, 2022. ISSN 2624-8212 [90]

K.K. developed the idea for the methodology and guided the implementation of the experiments. K.K. and S.H. designed the experiments. S.H. implemented the experiments. K.K., S.H., and A.H. discussed the results. K.K. wrote the publication with contributions from S.H. and A.H.

This publication is based on the results of the bachelor thesis of S.H. For the publication, the experiments were extended with more analyses and discussions. The main contribution of this publication to this thesis is the methodology and the results of the experiments, which are mainly described in Chapter 12.

K. Kobs, M. Steininger, and A. Hotho. InDiReCT: Language-Guided Zero-Shot Deep Metric Learning for Images. In *IEEE Winter Conference on Applications of Computer Vision*, pages 1063–1072, 2023 [135]

K.K. had the idea for the methodology, designed the experiments, and implemented them. K.K., M.S., and A.H. refined the methodology during discussions and analyzed and discussed the results. K.K. wrote the publication with contributions from M.S. and A.H. The full publication contributes to the thesis, mainly in Chapter 13.

K. Kobs, M. Steininger, A. Zehe, F. Lautenschlager, and A. Hotho. SimLoss: Class Similarities in Cross Entropy. In D. Helic, G. Leitner, M. Stettinger, A. Felfernig, and Z. W. Raś, editors, *Foundations of Intelligent Systems, Lecture Notes in Computer Science*, pages 431–439, Cham, 2020. Springer International Publishing. ISBN 978-3-030-59491-6. doi: 10.1007/978-3-030-59491-6_41 [132]

K.K. developed the methodology, designed the experiments, and implemented most of them. M.S. implemented the similarity matrix generation for the CIFAR-100 dataset. K.K., M.S., A.Z., F.L., and A.H. analyzed and discussed the results. K.K. wrote the paper with substantial contributions from M.S., A.Z. and contributions from F.L. and A.H. The full publication contributes to the thesis, mainly in Chapter 14.

K. Kobs, A. Zehe, A. Bernstetter, J. Chibane, J. Pfister, J. Tritscher, and A. Hotho. Emote-Controlled: Obtaining Implicit Viewer Feedback Through Emote-Based Sentiment Analysis on Comments of Popular Twitch.tv Channels. *ACM Transactions on Social Computing*, 3(2), 2020 [133]

K.K. and A.Z. developed the idea for using a TextCNN, guided the implementation of the experiments, and had the idea of using weak labels and input masking for the TextCNN. A.B., J.C., J.P. implemented the experiments and analyses. A.B. contributed the creation of the emote lexicon and the average based lexicon approach. J.C. contributed the distribution based lexicon approach. J.P. implemented the TextCNN approach. All authors analyzed and discussed the results. K.K. wrote the paper with substantial contributions from A.Z. as well as contributions from A.B., J.C., J.P., J.T., and A.H.

This publication is based on the results of a student project, but the use of the TextCNN model and the weak label approach was not part of the original project. The main contribution of this publication to this thesis is the methodology of using weak labels and input masking to improve sentiment analysis as well as its results and analysis, mainly described in Chapter 15. Thus, the main contribution of this publication to this thesis is not part of any other examination work. The order of student authors (A.B., J.C., J.P.) is based on the alphabetical order of their last names and does not reflect the contribution of each author.

C. Contribution Statement

J. Pfister, K. Kobs, and A. Hotho. Self-Supervised Multi-Task Pretraining Improves Image Aesthetic Assessment. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 816–825, June 2021. doi: 10.1109/CVPRW53098.2021.00091 [208]

K.K. developed the idea for the methodology and guided the implementation of the experiments. K.K. and J.P. designed the experiments and discussed the results. J.P. implemented the experiments. K.K. and J.P. wrote the publication with contributions from A.H.

This publication is based on the results of the master thesis of J.P. For the publication, the experiments were extended, and the results were analyzed and discussed. The main contribution of this publication to this thesis is the methodology and the results of the experiments, mainly described in Chapter 16.