SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Robotics, Cognition, Intelligence

Utilizing Attribution Maps for Regularization of Deep Neural Networks

Ralf Christian Kinkel

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Robotics, Cognition, Intelligence

Utilizing Attribution Maps for Regularization of Deep Neural Networks

Nutzung von Attributionskarten zur Regularisierung tiefer neuronaler Netze

Author:	Ralf Christian Kinkel
Supervisor:	Prof. Dr. Daniel Cremers
Advisor:	Christian Tomani M.Sc.
Submission Date:	15.02.2023

I confirm that this master's thesis in robotics, cognition, intelligence is my own work and I have documented all sources and material used.

Munich, 15.02.2023

Ralf Christian Kinkel

Acknowledgments

I would like to express my deepest gratitude to my thesis advisor, Christian Tomani M.Sc., for his guidance, insightful feedback and support throughout this thesis. His expertise and encouragement have been a constant source of motivation for me.

I would like to thank my supervisor Prof. Dr. Daniel Cremers for the opportunity to write this thesis and the great courses he teaches, which I am grateful to have participated in.

I am grateful to the Chair for Computer Vision and Artificial Intelligence for providing me with the resources to carry out experiments and the knowledge I obtained from projects and courses offered by the chair.

I would like to thank my family and friends for their unwavering support and encouragement throughout my journey.

Finally, I want to acknowledge my fellow students and colleagues, who made this experience an enriching one through friendship, thoughtful discussions and camaraderie.

This thesis would not have been possible without the help and support of all of these individuals. I am deeply grateful for their contributions.

Abstract

Regularization to avoid overfitting is an important consideration in training deep learning models, especially when working with small datasets. Hand-engineered data augmentation is a very popular technique to achieve regularization in domains like vision, but finding appropriate augmentations is difficult in domains that humans are not naturally proficient in. In this thesis we analyze how attribution methods can be utilized for augmenting input data in a domain independent way.

Our contribution is a framework and terminology for attribution based augmentation, an analysis of applicability of different attribution methods in this context, the tension theory for adjustment sets, two new attribution based augmentation methods, MendABA and FastABA, as well as benchmarks for these and an already existing attribution based augmentation method. We also draw some parallels to other regularization techniques and discuss further possible research avenues.

For the theoretical and empirical sections we use the datasets CIFAR10, MNIST and Tiny ImageNet as vision datasets, 20NewsGroups as NLP dataset and PTB-XL as a time series dataset with a wide variety of models and settings. In these benchmarks attribution based augmentation looks particularly promising for NLP, increasing performance significantly. Improvements in generalization for vision are smaller and attribution based augmentation is outperformed by hand-engineered data augmentation. Models trained on the time series dataset PTB-XL did not show large improvements when using attribution based augmentation in the training process.

Contents

1. Introduction

Deep learning has experienced massive successes in the last decades, first in computer vision [43], then in domains like natural language processing [26], time series forecasting [48], and control and planning [54], [80]. The successes in research often translate to useful applications in industry and consumer products. Deep learning today is used in tasks like transcription of speech to text, recommending products based on user interest [46] and even to assist in writing code [51] and generating art [71].

1.1. Problem Setting

Deep learning in general faces many challenges, like data availability and overfitting [106], interpretability and calibration [27], compute limitations and efficient training and mismatching data distributions between training data and test data [41]. Data availability and overfitting are among the greatest challenges. This is the case for deep learning researchers even though they tend to concentrate on large and high quality datasets, but even more so for practitioners, who often have to work with much lower data availability or quality, at least for labeled data [58]. This can be the case because labeling and filtering big datasets for quality is time consuming and therefore expensive. Availability can also be limited because there is data that might not legally be used [64] or it is expensive to buy or generate the necessary data. There is already an arsenal of regularization techniques that aim at reducing overfitting, some by synthetically generating additional data, but they do not solve the problem of overfitting completely and many have limitations in the settings they can be used in. For example: a popular way to synthetically increase the size of the dataset is by using hand-crafted data augmentation. Here each sample might be changed in ways that a human practitioner knows should not change the correct label associated with the sample. It is especially common in computer vision (CV), where an original image might be cropped, flipped, zoomed, rotated, color shifted, while it still contains the same or very similar information, see [50] for example. The difficulty of using data augmentation gets much higher in domains where humans are not naturally talented. For weather forecasts for example, humans do not have a good intuition which augmentations should have no influence on the prediction, like one knows that a slightly more blue and flipped cat image still shows a cat. Other regularization

techniques are domain independent but computationally expensive like ensembling [17]. Some regularization techniques like weight decay [49] and early stopping are domain independent and computationally inexpensive but might be used in synergy with other regularization techniques, so that an incentive for new regularization techniques that do not necessarily outperform them in isolation but are stackable with them is still given. Another great challenge is interpretability [55], as it is difficult to discern how deep learning models come to their predictions, while it is advantageous to have this information. For one because of legal concerns and safety considerations, especially in industries like medicine, but also for giving cues of how model training progresses and as a sanity check for finished models. As a consequence, researchers have created a plethora of interpretation tools to shed light on the inner workings of these models, also called attribution methods. A potential of these interpretation tools that is only starting to be explored, is utilizing resulting information on input attribution for the purpose of generating new data and help avoid overfitting [93]. We use the term attribution based augmentation (ABA) for such methods.

1.2. Goal

ABA is a recent regularization technique, that promises to be domain independent and stackable with many other regularization techniques. Our goal is to expand the understanding of ABA methods and introduce improved methods from our findings to decrease overfitting, leading to more robust and performant models.

1.3. Approach

We will analyze properties of the regularization technique "attribution based augmentation" and create a theoretical framework for it. We will develop theories behind the interaction of components in this framework. We will create new ABA methods from the knowledge we gain in the process and benchmark them on a multitude of models and datasets. In this endeavor we concentrate on deep learning and the task classification. We also narrow the scope on attribution and augmentation of input features.

1.4. Structure

We will first explain the fundamentals and outline related work in Chapter 2, starting with the fundamentals of deep learning, challenges of the field and how deep learning

relates to science in general. We then specifically discuss the challenge of overfitting and how it is mitigated with regularization. We explain regularization and some of the most popular techniques and methods in greater detail. Finally, we discuss interpretability and attribution maps and show some of the attribution methods in use. Chapter 3 is about building an understanding of attribution based augmentation and creating new methods. We first define terms and develop a framework and pipeline for ABA. We then identify mechanisms of how attribution based augmentation works and underpin our theories with empirical evidence. Next we describe criteria of attribution methods that would make them good candidates for use in ABA and how well the attribution methods presented in the previous chapter fit the criteria. We draw parallels from attribution based augmentation to other regularization techniques and where they differ. Finally, the theoretical knowledge we gained is used to construct two new ABA methods, MendABA and FastABA. Chapter 4 is about evaluating and benchmarking, where we will compare performance of the ABA methods defined at the end of the last chapter. Here we use different datasets from the domains computer vision, natural language and time series and a multitude of models and settings. We will discuss our results and findings in Chapter 5. We outline some possible future work in Chapter 6 and conclude with key takeaways of the thesis in Chapter 7.

2. Fundamentals & Related Work

This chapter introduces the background of what artificial intelligence (AI) means and what defines deep learning as a sub field, along with some of the history and milestones. We explain the most important concepts and techniques in deep learning shortly. The concepts of overfitting, regularization, attribution methods and how attribution based augmentation fits in are explained in greater detail.

2.1. Artificial Intelligence

AI refers to the endeavor of simulating intelligence in machines. Sometimes encompassing aspects of human intelligence [53]. An AI might be able to do tasks like understanding language, problem-solving and navigating complex environments. All current AI systems are narrow, meaning that they are able to perform some specific tasks a human might perform and even surpass us in some respects [37]. General AI is the concept of an AI matching or surpassing human capabilities in most respects.

2.1.1. History of AI

The concept of simulated intelligence has been around for thousands of years in stories of magical artifacts capable of communication and independent thought. The concept of intelligent machines can be traced at least to the 1940s [28], where the science fiction author Isaac Asimov published his short story *Runaround* with the *Three Laws of Robotics* [3]. The term artificial intelligence is more recent and was coined in 1956 at the *Dartmouth Summer Research Project on Artificial Intelligence* [28], the first large scale scientific endeavor on the subject. At this point the computer offered a realistic avenue promising the rapid development of useful AI systems. The complexity of tasks that appear easy for humans was underestimated and the development of AI was slower than hoped. This led to a decline in interest in the field, which is known as *AI winter*. Since then the field of AI has experienced phases of higher interest, called *AI summer* and lower interest, *AI winter*. First with a resurgence of interest with the rise of expert systems [60], a decline in interest when limitations of expert systems became apparent and a smaller resurgence of interest when capabilities of machine learning increased. In recent years, the sub field of deep learning rose to prominence, leading to many

breakthroughs in AI and a period described as *Third AI Summer* [38]. With some of the most prominent breakthroughs getting widespread public attention. A small selection of those breakthroughs or advancements is listed here:

- **Winning ImageNet Challenge in 2012:** The ImageNet Challenge [14] is a competition in computer vision, where models have to learn to assign images to one of 1000 classes based on the object displayed in the image. In 2012 a many-layered artificial neural network AlexNet [43] entered the competition and won by a large margin. Some argue that this heralded the beginning of the current *AI Summer* [38]. Since that date deep learning models have consistently won the challenge and came to dominate the field of computer vision, the "deep" referring to the number of layers.

- **AlphaGo in 2016:** After the defeat of the chess world champion Kasparov by Deep Blue [11] in 1996, the Chinese board game Go was one of the last remaining classical games where humans still remained dominant over computers. The reason being that Go is much more complex and has a much greater space of possible positions [80] than Chess and most other board games, making brute-forcing solutions exceedingly difficult. In 2016 AlphaGo [80], an AI system incorporating deep learning, beat the world champion Lee Sedol in Go. This was a huge breakthrough and showed that narrow AI can surpass the greatest human experts even in such a complex domain.

- **AI Generated Art:** Recently AI generated art has captured the imagination of the public. There is not one single model or event that was a breakthrough here but a rapid succession of more and more capable models in the span of a few years. The first photo realistic human images where generated with generative adversarial networks (GANs) [24], but GANs were soon surpassed by the DALL-E [68] models at the end of 2020 and then again in 2022 with the open-sourced Stable Diffusion models [71].

- **Natural Language Processing:** Same as AI generated art, there has not been a single breakthrough here but a very rapid increase in capability. The GPT [66] model family might be the most popular one currently, especially with the most recent addition ChatGPT, a language model that users can interact with and ask questions in a natural conversation flow. These natural language processing (NLP) models promise to help as personal assistants [67], assist in writing code [104] and revolutionize how people access the huge knowledge base that the internet provides[1].

[1]This is the authors personal opinion based on interactions with ChatGPT and from anecdotal evidence.

- **AlphaFold in 2018:** AlphaFold [36] is an AI system that predicts the 3D protein structure from amino acid sequences and is much more capable than competing models. It helps in understanding the structure of proteins and with this, advance biological research.

2.1.2. Expert Systems

Expert systems [10] are a sub area of AI. They use databases of manually defined expert knowledge to offer advice or make decisions. They use rules to reason through problems with the help of the knowledge base. They offer to mimic capabilities of human experts to some degree, while being able to handle large amounts of data and might be used in fields like medicine and finance.

2.1.3. Machine Learning

Machine Learning (ML) is a sub area of AI where algorithms based on statistics and data are used to enable computers to learn [32]. This way ML models are able to learn to give useful predictions with only the architecture and the learning process being manually programmed. An important distinction in ML is between supervised and unsupervised learning. In supervised learning, the data is labeled and the model learns to predict the correct label. In unsupervised learning, the data is unlabeled and the model learns patterns between data points [62].

Deep Learning

Deep Learning is a subarea of machine learning with specific model architectures and training paradigms. We will explore deep learning in greater detail in Section 2.2.

2.2. Deep Learning in Detail

In this section, we will first explain deep learning in the context of models in science, then architectures of deep learning models. Afterwards we will describe how those models are trained and finally explain the domains deep learning are most popular in and typical tasks in those domains.

2.2.1. Science and Deep Learning

Science is the endeavor of building and organizing knowledge that allows one to make accurate predictions. Models are representations of systems and used to make such

predictions. While laws of nature can often be described by models with only a few parameters, many tasks humans face are much more abstract and complex. We can consider object classification as an example. On a physical level one is exposed to billions of photons with different wavelengths and has to assign that photon bundle to one category. While humans can mostly recognize objects intuitively, it is very difficult to describe all the implicit assumptions we do in this process and encode them in one model. Deep learning approaches those complex problems by creating models with a vast number of parameters connected in a way that allows them to theoretically model any mathematical relationship using enough parameters[2] [34][13]. These models are called *artificial neural networks* (ANN) and are inspired by the human brain [46]. ANN consist of many simple and interconnected processing units, learning tasks through training [20]. According to the universal approximation theorem, [34] any large enough ANN should be able to approximate any continuous function. The whole difficulty lies in the assignment of the correct weights via training, with models large enough to fit the specific function. ANN with the correct architecture and training can make highly accurate predictions and for this reason deep learning is expected to play a significant role in the future of technology [28].

2.2.2. Model Architectures

The are are a variety of deep learning architectures, which are often inspired by the human brain in some capacity [62]. They all fall under the term artificial neural network. The commonality between them is that they are arranged in connected layers[3], with an input layer an output layer and any number of layers in between, called hidden layers. The layers consist of neurons that are connected by weights to some neurons in previous and subsequent layers. The typical process of information flow is that input data gets passed from one layer to the next and is multiplied by weights assigned to each connection in the process. Then a value[4] specific to the neuron is added and finally some non-linear function is applied to the result. The outputs of the neurons of the layer can can then be passed to the next layers again. The process is repeated until the information reaches the final output layer. The non-linear functions are also called activation functions. In previous decades the sigmoid and tanh functions were used extensively, currently ReLU and slight variations of it are most popular [46]. We will

[2]Of course some implicit assumptions are built into the models and data pipelines to reduce the number of parameters needed. These assumptions are built into the model architecture on one hand but especially in the pre-processing. Vision models work on pixels for example, not on photon bundles.

[3]With graph neural networks being somewhat of an exception.

[4]This value is often called bias but might also be treated as another weight that always multiplies the input "1". We include it in the term "weights".

briefly mention and explain the key features of some popular classes of architectures. We will go into greater detail for convolutional neural networks, as it is the main architecture class used in this paper.

Feedforward Neural Network

The most basic architecture class is the feedforward neural network [72]. Here all information flows in one direction from input to output. All neurons of a layer are connected to all neurons of the previous and subsequent layers. These layers are called fully connected layers [62]. Another layer that is often employed here and specific to classification is the softmax layer [9]. In classification tasks it is useful to have an output that adds up to 1 so that the numbers can be interpreted as a probability allocation for the different classes. The layers of ANN do not typically add up to 1, the softmax function is a normalization that enforces outputs to this total.

Convolutional Neural Network

Convolutional neural networks (CNN) [62] are inspired by the visual cortex and were originally created for image-processing. The motivation is that fully connected layers struggle with very-high dimensional inputs like images, as the number of weights and computation needed are enormous. To address this in the 2D vision domain, the input image is arranged into height, width and channels and there are new types of layers making use of this in a more compute and parameter efficient manner:

- **Convolutional layer:** The convolutional layer is the defining feature of this architecture. The main difference to a fully connected layer is that a neuron is not connected to all neurons of connected layers but only with neurons that are close by in the height and width dimensions. This way models can make use of hierarchies in the structure of images with early layers often recognizing local geometric features like edges, corners, medium deep layers recognizing geometric patterns like meshes and circles and the later layers recognizing complete structures [107].
- **Pooling layer:** The pooling layer is used for reducing the spacial dimensionality, to reduce the number of parameters [62]. Pooling layers do not have adjustable parameters like weights, but operations like returning the average or the maximum of features passed in.

This type of model architecture was later adapted to be used with 1D data like sound or word sequences, too.

Recurrent Neural Networks

Recurrent neural networks (RNN) are derived from feedforward neural networks and specifically made for sequences like time series and language, they can take an input of variable length [74]. They have an internal state often called memory, this memory keeps track of information from previous steps and contributes to the output. This way the prediction for each time step depends on the current and previous elements of the input and output. This is important for example when translating texts, where there are multiple valid translations and information of the current translation path is needed to output a sensible next word.

Graph Neural Networks

Graph neural networks [73] are designed to process data represented as graphs. The network is created so that it can handle the variable size of the input and be invariant to permutation of the input data [103]. Each input is assigned one node connected to other nodes based on how the inputs relate to each other. The key feature is a message passing step, where graph nodes update their representation by information exchange with neighboring nodes. The number of exchange steps determines how far the information travels and determines how many operations are executed in sequence, like layers in other classes of ANN. Citation networks, molecular biology, social networks and multiple-object tracking [8] are some of the areas they are used in.

Transformers

Transformers are a recent class of architectures that use self-attention to compute outputs from sequences of inputs [95], self-attention allowing the model to weigh importance of the input features. The architecture was created for NLP but is usable in most fields. Transformers rapidly dominated the field of NLP [102], by being better able to parallelize computation than RNN. More recently transformers also gained success in the field of CV [39].

Generative Adversarial Networks

The idea behind generative adversarial networks [24], is that to generate samples for a certain distribution, the task of the generator model, it is helpful to train another model in conjunction that gives feedback on the generated samples, the task of the discriminator model. The generator model learns how to create realistic samples and the discriminator is passed either real samples or the generated samples and learns how to classify them as real or generated. The idea is that the models are getting better at

generating realistic samples and judging generated samples in a competetive approach. GANs are somewhere between a training approach and a model architecture. The generator and discriminator models can consist of different architectures.

2.2.3. Training ANN

The basic training process is conducted in a series of steps [77], we will first mention those and then explain parts of the pipeline:

1. **Pre-Processing:** The data might be pre-processed before being passed to the model.
2. **Forward Pass:** The pre-processed data is fed into the first layer of the ANN. The layer operations described in Section 2.2.2 are executed and the data is passed on. This is done for the successive layers until the output layers is reached, in a process called forward pass. At the end, a loss function is calculated that is lower the better the model prediction.
3. **Backward Pass:** The gradients of the weights in the model with respect to the loss are calculated in a process called backward pass applying the backpropagation algorithm.
4. **Weight Update:** The weights are updated based on these gradients, changing the weights into direction of lower loss.

The process is repeated several times for the different mini-batches but most times every sample is also viewed multiple times, each iteration of the dataset is called epoch.

Loss Function

The loss, sometimes called cost, represents an error of the network prediction. It depends heavily on the task. In the task of classification for example, the typical loss function is called cross entropy, which is higher the smaller the probability the model assigns to the correct class [99].

$$L = -\sum_{c=1}^{M} y_c \log(p_c) \tag{2.1}$$

For all M possible classes the network outputs its predicted probabilities p while y, the target, is 1 for the correct class and 0 for other classes. For regression there is the mean squared error (MSE) between prediction and target value for example and in generation tasks one might view the discriminator as a complex loss too [24].

Backpropagation

Backpropagation [46] is the concept of passing the derivative of this loss function through the network and all its weights via the chain rule, where derivatives for each layer are calculated in sequence, starting from the last layer. This way one can obtain the derivative of each weight, with regard to the loss, efficiently.

Weight Update and Optimization

By updating the weights in the direction of lower loss the network can gain suitable weights for making accurate predictions much faster than by random weight updates. The strength of this weight update is called learning rate α. This process occurs most often with mini-batch based gradient descent. Using the loss information from all samples in the dataset at once and updating the weights based on that is very slow, while updating by a single sample each time leads to erratic behavior in the optimization process [87]. A balance is best, where loss information of multiple samples is considered at once for the weight update, the mini-batch. The practice of updating with information from these mini-batches is at the basis of almost all optimization methods used in deep learning. Just updating weights based on gradients of these mini-batches is called mini-batch stochastic gradient descent [70] or just stochastic gradient descent (SGD) from now on. We will also refer to mini-batch just as batch moving onward[5]. One other very popular optimization method is Adam [40] where the updates are smoothed out by updating with exponentially moving averages of the gradients and making the update invariant to average gradient magnitude.

Overfitting

In the training process the problem of overfitting can occur [106], where the model learns to recognize every single sample by its composition, instead of learning general patterns. This leads to a poor ability to generalize to unseen data. Often this phenomenon can be observed in metrics for the training data improving continuously while test metrics decrease after some point in the training process. Overfitting is discussed in greater detail in Section 2.3.

[5]There are some complications with changing terminology, as in the original paper SGD could mean to use just one sample at a time and mini-batch stochastic gradient descent would be the more fitting terminology. Because updating on one sample at a time is very unpopular, SGD now refers to the mini-batch version in most literature and one would refer to the original SGD as "SGD with batch size = 1", for example. The term batch is also used instead of mini-batch in most literature, even though the term batch originally meant the whole dataset.

Dataset Splits

It is typical to split datasets in DL into a training, validation and test set [105]. The model uses batches of the training set to update parameters in the training process. The validation set is used to evaluate model performance and with this information tune hyperparameters[6] of the model and training process. As only the forward pass is performed on the validation set, the model can not learn to fit these samples. The test set is held back until the model is finished training to give a fair and unbiased evaluation of model performance. This is separate from the validation set as the act of tuning hyperparameters on the validation set also can lead to some overfitting on it, not through parameter updates directly but over a random or deliberate search for optimal hyperparameters.

2.2.4. Typical Deep Learning Domains and Tasks

Domains in AI are broad areas encompassing specific tasks and data types. We list the most important ones here[7].

Computer Vision

Computer vision is a domain about understanding visual information. It can be separated into 2D and 3D computer vision and both might also be analyzed in a video format with temporal information [96]. The possible tasks entail classification, semantic segmentation, object detection and motion tracking. Deep Learning came to dominate most of the field by being able to utilize the vast amounts of data accessible in this domain, it is also more easy to use in some respects than many older solutions, because no features have to be hand-crafted. Another advantage is that transfer-learning works very well here: One might train a computer vision model on a a large amount of data in a related task where the data is available and only adjust the weights slightly by retraining the model on the specific task, explained in more detail in 2.4.4.

Natural Language Processing

Natural language processing is about understanding human language [63]. The domain entails tasks like text classification, translation, text generation and summarization.

[6] While parameters are tuned in the training process, the term hyperparameters basically encompasses all settings for the model architecture and the training process, like batch size, learning rate and activation functions.

[7] Importance as current and historic interest. For computer vision, natural language processing and times series there are often separate conferences and journals for example.

Deep learning architectures like RNN have been used extensively in this field, but especially since the advent of transformers trained on large datasets, the field has advanced rapidly. The domain only encompasses processing text but is closely linked with speech recognition, where speech is converted into text.

Time Series

Time series are a sequence of data points that are in temporal relation to one another [22]. Examples where time series are often found are weather forecasting, economics and medicine. Tasks might be predicting future values, anomaly detection or feature extraction. Model architectures like RNN and CNN have been used a lot here, but recently transformers have also gained traction in the field [101], especially because of their ability to capture long-range dependencies.

2.2.5. Use in Robotics

Robotics is about improving robot behavior and performance, robots being agents interacting with the real world, often in difficult environments [89]. Robotics is at the intersection between many domains and deep learning might play a role in a variety of tasks here.

- Computer vision is often used in robotics for perceiving the environment and the robots position in it. Other sensors are used too and give a stream of data over time.
- Analyzing this time series can help the robot learn how to influence its surroundings by giving feedback on interactions with the environment and identify how to achieve goals, with planning and navigation.
- Planning and navigation are closely related. Planning is about strategic assessments of goals, priorities and actions [21] and navigation is about navigating a complex environment by choosing paths for the identified goals. The way the robot interacts with the world once a goal is analyzed is often referred to as control.
- Control is about the robot being able to move and interact with objects [57]. Here deep reinforcement learning might be applied, a combination of deep learning with reinforcement learning.
- Natural language processing is currently not that common in robotics [92], but might be used in the future to interact with robots verbally, in conjunction with speech recognition.

2.2.6. Interpretability of ANN

An important limitation of ANN is that that they are often difficult to interpret, which means that humans have difficulty in understanding what causes a model to make predictions [55]. The reason for this difficulty in interpreting ANN is, that they often have millions or billions of parameters and their predictions are based on highly non-linear relations between them. That is why ANN are often called *black boxes*, as one can understand the input and output, but not the processes in-between. To help with interpretability, a wide array of techniques have been proposed, some of which impose architecture constraints, so the model is intrinsically more interpretable, some concentrating on explaining the weights in the model and some on the way inputs influence predictions. For attribution based augmentation, the last category is most important, as attribution methods aim at identifying importance of input features. We discuss attribution methods explicitly in Section 2.5.

2.3. The Problem of Overfitting

Typically overfitting in science refers to parsimony: Following Occam's Razor one should use models that contain just the amount of parameters and procedures necessary for describing the relationship one wants to model [30]. That is relatively straightforward when describing simple regression problems or natural laws that can be modeled with a few parameters. For complex problems like understanding vision or language it is unknown exactly how many parameters are needed or what procedures to use to model them. That means that models will likely have too little parameters to accurately describe the relationships or too many, when dealing with these complex tasks. When deep learning models are used this is virtually unavoidable as they have the flexibility to fit any function given enough parameters [13], but at the cost of almost always using more parameters than necessary for the specific function.

2.3.1. Overfitting in Deep Learning

A related problem the term *overfitting* also refers to in the context of deep learning, is that a model memorizes the samples in the training set to make correct predictions. If there are too few samples, it may be easy for the model to fit the data perfectly, while not learning the underlying rules and patterns of the domain and task [106]. That leads to metrics for the dataset the model is trained with to improve continuously in the training process, while the ability of the model to generalize to unseen data decreases after some point. Even if a deep learning model had the perfect structure to describe a relationship, it is very likely that the model learns some rules specific

to describing the dataset and not the underlying relationships if the dataset is small enough. While Occam's Razor could be satisfied by that model in principle, it would still be much more likely to only fit that dataset and fail to generalize. Techniques to reduce overfitting in deep learning and thereby help the model to generalize better are often called regularization [44]. We will discuss regularization techniques in greater detail in the next section.

2.4. Regularization

Regularization in the context of deep learning describes techniques that aim at making trained models generalize better and thereby reducing test error [44], often by making it harder for the model to fit the training data. Regularization techniques can be divided into different classes along different criteria. We use the division into regularization via data, via network architecture, via regularization term and via optimization [44]. There are often multiple regularization techniques employed at the same time. In training a CNN on a small dataset for example, it is quite common to have regularization in the data, architecture and optimization at the same time.

2.4.1. Regularization via Data

Regularization via data includes augmentations on inputs, hidden features or targets. We put greater emphasis on augmentations on input data in this subsection, as attribution based augmentation belongs to that category. When using regularization via input data, transformations are applied to the training set D, resulting in a new augmented training set D_R. The general idea is that while training, the model is encouraged via the training process to find a configuration of weights that is more robust to these augmentations. This more robust configuration tends to also generalizes to unseen data better.

Noise Injection

Noise injection is a regularization method where random noise is added to data. There are different variants of this method aiming at inputs, hidden features or targets. Gaussian noise on input [7] for example, changes the input features based on a Gaussian function. Gaussian noise on hidden units [16] applies Gaussian noise on the hidden features. Annealed noise on targets [98] injects noise in the targets.

Adversarial Training

Adversarial training [25] is a regularization technique where a model is exposed to adversarial examples during training. These adversarial examples are constructed in a way where the model is confidently making wrong predictions. While there are different methods for generating an adversarial example, a common one is the Fast Gradient Sign Method (FGSM). Here the gradients for the inputs of a sample are calculated and then the inputs are changed slightly in direction of increasing loss:

$$x' = x + \epsilon * sign|\nabla_x J(\theta, x, y)| \tag{2.2}$$

with ϵ being a small value, x being the input and J being the loss depending on the weights, inputs and labels. Another advantage of adversarial training is that it makes models especially robust against adversarial attacks [6]. Adversarial attacks are used at test time to fool a model in a similar way adversarial examples are used at training time, but without the model being able to adjust.

Hand-Crafted Data Augmentation

Hand-crafted data augmentation is a regularization techniques used to artificially increase the size of a dataset by generating additional data samples from the existing data [76]. This is done by applying a set of transformations to the data. These transformations are defined manually and can therefore only be applied in domains the user understands well. It is especially popular and useful in computer vision, as there are many transformations that humans know should not change the class of the depicted object when applied on an image. Such transformations might be a slight color shift, rotations, zooming, or flipping the image [50]. Many of these augmentations can be combined, leading to a much larger augmented dataset. Due to the prominence of hand-crafted transformations in CV, researchers developed methods to select good combinations of them. RandAugment [12] is such an automated transformation selection method. Here there is a pool of 14 possible transformations, where a number N are chosen at random and applied on the image, with a defined magnitude. In natural language processing (NLP) transformations are also used but less popular [47]. For example, a text might be translated to a foreign language and back to the original or words can be replaced with synonyms. In other domains hand-crafted augmentations are often much less popular because finding sensible augmentations is hard in domains where human intuition is weak.

Attribution Based Augmentation

Attribution maps show which parts of an input are most relevant for model predictions. We discuss them in greater detail in Section 2.5. Utilizing attribution maps in a regularization technique is a relatively recent effort. The key idea is that augmenting regions of the input in different ways based on their importance might be advantageous. The first mention we found was [23] with KeepAugment. The proposed method is not quite a self-reliant method for regularization via data but rather an extension to hand-crafted data augmentation. KeepAugment identifies the most important regions and excludes them from the augmentation procedure, this way distribution shift is reduced and the trained model generalizes better. The opposite approach is taken by [93] with Challenger. The proposed Challenger method augments only the most relevant regions. This is the first self-reliant attribution based augmentation method we found[8]. The augmentation occurs in a 5-step approach.

1. Compute forward pass for the batch.
2. Compute attribution scores in respect to the $k - th$ class, based on Layer-wise Relevance Propagation (see 2.5.2).
3. Determine the most important input features by computing the highest and lowest percentiles of the attribution.
4. Randomly select either the highest or the lowest attributions for each sample.
5. Randomly either increase all selected features, decrease them or keep them unchanged, with a $\frac{1}{4}, \frac{1}{4}, \frac{1}{2}$ split, for each sample. Afterwards continue regular training with the augmented batch.

The main advantage of attribution based augmentation methods in comparison to hand-crafted data augmentation is that it is domain independent, meaning it is usable in domains where humans do not have as good intuitions about sensible augmentations as in fields like vision and language.

Label Smoothing

Label smoothing [90] is a regularization method that augments the target in supervised classification. Regularly the target for this task is either a 1 or 0. Label smoothing converts these hard labels into soft labels, by reassigning some of the probability of the correct class to all others. This counteracts overfitting and also reduces overconfidence in the model. Another problem deep learning models often face [61].

[8] Because of this, a subset of experiments conducted in Chapter 3 and Chapter 4 are inspired by the choices for datasets and models in this paper.

2.4.2. Regularization via Network Architecture

There are network architectures inherently well suited to generalize to unseen data and there are specific components that can be added to a model to increase regularization. Often these components behave differently in training and at test time. Architectures that tend to be inherently well suited are for example small networks that lack capacity to overfit and models with bottle-necked layers [44]. We also describe some components that might be added for regularization.

Batch Normalization

Batch normalization [35] is an architecture element in deep learning models with a regularizing effect. In training, batch normalization can be applied after a layer to normalize activations across a batch. This is done by first subtracting the mean and then dividing by the standard deviations of activations in the batch for each activation. The activations after this step have a mean of 0 and a standard deviation of 1. This normalization throughout the model prevents small changes in parameters to amplify to huge values over many layers, which stabilizes the optimization process. At test or inference time, the normalization step can not be applied as there is no batch and is instead replaced with an average over train time. The regularization is more of a side-effect from having randomness in the way the mini-batches are composed, which makes it harder for the model to overfit on the specific input features.

Ensembles

In ensembles [17], several models are combined to create a stronger predictor. For neural networks, multiple models are trained with the same or different architectures and at test time the results of each are put together and weighted to make predictions. The set of models is called ensemble. The technique is generally applicable in machine learning. In general as long as each model is better than random and the models make different errors, the ensemble is a better predictor than its constituents. The technique offers a trade off with a higher number of constituents leading to higher performance with diminishing returns and linearly more computation required to run the ensemble.

Dropout

Dropout [33] is a regularization method that can be interpreted as an architectural element or as regularization via data on the hidden features and data. In training and for each sample a percentage of weights in the model is omitted, so extreme dependencies of weights to each other get reduced. This way the model is encouraged

to form redundant sub-models to be robust enough to achieve accurate predictions despite the changes. The combination of possible sub-models can be interpreted as ensembles [29] with the same positive effect on generalization. If dropout is applied on input data it is also leading to a larger augmented dataset that is harder to overfit.

2.4.3. Regularization via Regularization Term

Regularization can be increased with adding a regularizer term to the loss function[9] [44]. This term is independent of the label and can be applied in unsupervised learning and tasks other than classification. Most notable here are $L2$-regularization and $L1$-regularization. They are additions to the loss function that penalize the weight magnitude of the model. This can be interpreted as more complex hypothesis being discouraged to better fulfill parsimony, leading to better generalization. $L1$ and $L2$ refers to the norm on the weights that is added to the loss function. The term added to to the loss function in these cases looks like:

$$R(w) = \lambda \frac{1}{2} ||w||_l^l \tag{2.3}$$

where λ determines how strong of an effect there should be, w being the weights and l being the norm chosen, with $l = 1$ and $l = 2$ being the most common. Weight decay (see 2.4.4) and $L2$-regularization are very closely related. With SGD, $L2$-regularization leads to weights being reduced after each weight update step by a certain fraction, which is equivalent to weight decay [49]. This relation does not hold for other optimization algorithms though.

Feature Diversity

Regularizing deep neural networks by enhancing diversity in feature extraction is another approach. As ANN tend to be highly overparametrized as discussed in Section 2.3, models tend to develop redundant features [4]. This can be reduced with a regularization term penalizing feature similarity.

2.4.4. Regularization via Optimization

The optimization process itself might also contribute to regularization [44]. SGD is often treated as a baseline for neural network optimization and is still widely used. Its use of mini-batches makes models escape saddle points in the loss landscape better, leading to faster optimization. Faster optimization implicitly also regularizes, as overfitting the

[9] In principle the loss function itself might also be constructed to generalize well, but this is currently much less common [44].

data takes a number of iterations to worsen. Good initialization can speed up training by selecting sensible starting weights, based either on heuristics or transfer-learning. Terminating the training early can also help prevent overfitting.

Early Stopping

Early stopping is a simple method where the validation accuracy or validation loss is used to identify the point at which overfitting takes place and stop training there. If this is not used there is often a U-shape in the validation loss curve, where the loss increases after some point, while the training loss continuously decreases.

Transfer Learning

Transfer learning [94] is a regularization technique for machine learning, where a model trained for one task and dataset is used as starting point for training a second model. Most often a base-model is used as the starting point, which was trained on a large and high quality dataset. The idea is that there are often generally useful low-level operations in one domain[10], which the base-model will likely have mastered. For example in CV any model has to master the recognition of edges, corners and other basic geometric shapes. The base-model can then be fine-tuned to the real task and dataset, potentially with freezing the lower layers, that mostly learn the basic low-level operations, or adding some layers on top.

Weight Decay

Weight decay means that in the weight update step, the weights are reduced by a fraction. In terms of parsimony, this can be interpreted as continuous reduction in the complexity of the hypothesis, improving generalization. Weight decay was first introduced as part of the optimization process and seperate from $L2$-regularization in [49].

2.5. Attribution Methods

Attribution methods broadly belong to the area of interpretable AI. Attribution methods are used to identify the importance of the different input features for the model prediction [93]. The output of these methods is often called attribution map or saliency

[10] We use domain as defined in Subsection 2.2.4, domain as used in transfer learning literature is often much more specific. There it is used to describe any space of possible inputs. Pictures of dogs in forests and pictures of dogs in gardens are from different domains when using the term in this way.

map [55]. These methods can be separated based on different criteria [55]. Their are occlusion based methods where a part of the input features are manipulated to generate the attribution map and gradient-based approaches, where gradients of the input features with respect to the loss or class scores are computed in variants of the standard backward pass used in training of DL models. One can also differentiate gradient-only methods and path-attribution methods. Gradient-only methods only consider the importance of a change of the input feature at the current configuration. Path-attribution methods on the other hand, compare the current input with a reference input, also called baseline, which has to be chosen sensibly as it impacts the attribution [86]. The model prediction with the current input is compared to the model prediction with the reference input and the difference is split among all input features. It is to note that attribution methods usually refer to the task of classification, which we focus on in this thesis. For the sake of attribution based augmentation some attribution methods might be altered to be used for other tasks, which is discussed in Chapter 6.

2.5.1. Properties of Attribution Methods

There are a number of properties an attribution method might have. We mention a selection from [88] here. They are mostly useful for distinguishing path-attribution methods and occlusion methods, as gradient-only methods can not really satisfy most of them, as described in Subsection 2.5.3. We will introduce some additional properties for evaluating attribution methods in the context of ABA in Chapter 3.

- **Sensitivity and completeness:** Sensitivity is satisfied if for two samples that differ in one feature and have different predictions, the attribution for this feature is non-zero . A related concept is completeness. Completeness means that the sum of attributions for all input features equals the difference in prediction between the input and a reference input. Completeness can be viewed as a stronger version of sensitivity and always implies sensitivity.
- **Implementation invariance:** If two networks have equal outputs for all inputs, it is desirable that all attributions of these two networks are equal for identical inputs.
- **Linearity:** If two models with similar architecture but different weights were linearly combined into a new model, then it would be desirable for the attributions of this third model to be the similarly weighted sum of the original two models, for all inputs.

2.5.2. Gradient-Based Path-Attribution Methods

Gradient-based path-attribution methods utilize feature baselines to determine input feature importance, which requires at least some understanding of the domain.

Layer-Wise Relevance Propagation

Layer-wise relevance propagation (LRP) [5] is a path-attribution method that is gradient based. The difference to vanilla gradient is that the way that the information is passed backwards through the model is different. Their are special operations in LRP for each layer depending on model architecture that aim at redistributing the relevance of one layer to the next in equal amount. One disadvantage is that some layers need special rules and workarounds to be handled with LRP [56], like batch normalization, which is widely used. This makes the method not immediately architecture-independent. LRP satisfies completeness, but not implementation invariance [88].

Integrated Gradients

In Integrated Gradients [88] a baseline input is chosen and gradients are computed at different steps on the path from the baseline input to the true input. Those gradients are averaged over the steps and multiplied with the difference of true input to baseline input to obtain the attribution map. Integrated gradients satisfies completeness, implementation invariance and linearity.

DeepLift

DeepLift [78] is similar to LRP in many regards. It is another method that redistributes the importance of each input feature to the prediction. It uses the chain rule similar to backpropagation to propagate the relevance down the layers but also suffers from special layers needing their own implementation like LRP. DeepLift satisfies completeness, but not implementation invariance [88].

2.5.3. Gradient-Only Methods

With these method the gradients of input features with respect to the class scores or loss are calculated in some variation of the regular backpropagation. These methods are *local* in the sense that the gradients determine importance of small changes of input features, but not how important the feature is in total or *globally*. It is this local nature that makes gradient-only methods unable to satisfy sensitivity, completeness and implementation invariance.

Vanilla Gradient

Vanilla gradient [81] is a gradient-only method utilizing the standard backpropagation approach of computing gradients of a target function and passing it backwards through the layers of the model. In regular DL training this target function is a loss but Vanilla Gradient calculates the attribution in relation to a class score by default. The gradient propagation through ReLU is an important consideration in this section, for Vanilla Gradient the standard is:

$$\frac{\delta L}{\delta z} = \frac{\delta L}{\delta a} \cdot g'(z) \tag{2.4}$$

with ReLU:

$$\frac{\delta L}{\delta z} = \frac{\delta L}{\delta a} \cdot I(z > 0) \tag{2.5}$$

Where L is the target function, either class score or loss, g is the activation function, z is the input of g and a is the output with $a = g(z)$. The derivative of ReLU is the indicator function I.

DeconvNet

DeconvNet [107] is an attribution method in the gradient-only category. Originally it was constructed for CNN with a reversal of pooling layers, convolutional layers and activation layers, but [81] demonstrated that it can be seen as a slight modification to Vanilla Gradient, specific to the ReLU activation function. The gradient propagation through ReLU is changed to:

$$\frac{\delta L}{\delta z} = \frac{\delta L}{\delta a} \cdot I(\frac{\delta L}{\delta a} > 0) \tag{2.6}$$

Guided Backpropagation

Guided Backpropagation [84] can be seen as a combination of Vanilla Gradient and DeconvNet, again specific to the ReLU activation function. The gradient propagation through ReLU is changed to:

$$\frac{\delta L}{\delta z} = \frac{\delta L}{\delta a} \cdot I(z > 0) \cdot I(\frac{\delta L}{\delta a} > 0) \tag{2.7}$$

Gradient*input

This method was proposed by [79] as equal to LRP when a few assumptions are made, like using 0 as reference, but with simpler implementation and architecture independence. They showed that by multiplying the attribution map obtained by

Vanilla Gradient with the input features themselves, an attribution map equivalent to LRP could be obtained.

CAM and variants

Class Activation Mapping (CAM) [108] is an attribution method that constructs a CNN without fully connected layers to help in explaining models. It is restricted to a subset of CNN architectures [75]. Gradient-weighted class activation map (Grad-CAM) [75] is a gradient-only generalization of CAM, but still specific to CNN. Here the gradients are not backpropagated to the input but the last convolutional layer. The target here is always one class score. In a visualization the features would be average pooled along the channel of the last convolutional layer and scaled to be between 0 and 1, the resulting attribution map is a "heatmap" for feature importance [55]. Guided Grad-CAM [75] is a variant that combines Grad-CAM with another attribution method, like Vanilla Gradient [55].

Smoothgrad

Smoothgrad [83] is an extension to all gradient-based methods. Instead of generating attributions for the current input only, the attributions are generated for multiple variants of the input with some noise and averaged afterwards. Smoothgrad is supposed to make attributions more meaningful by smoothing out small scale fluctuations [55]. The noise is typically a zero-centered Gaussian.

2.5.4. Occlusion based methods

In occlusion based methods parts of the input features are augmented to generate attribution maps, they are model-agnostic [55] and can be used even outside deep learning. All occlusion based methods we explored were also additive feature attribution methods. Additive feature attribution methods use simple explanation models that are interpretable approximations of the original model [52]. An explanation model g is linear and assigns importance scores ϕ to features that add up to the output of the model, with the assumption that each feature score is independent of other features:

$$g(z') = \phi_0 + \sum_{1}^{M} \phi_i z'_i, \tag{2.8}$$

where $\phi \in \mathbb{R}$, $z' \in \{0,1\}^M$ with M being the number of simplified input features and z representing a binary vector that represents if an input feature is the original of the sample or 0. g is the simplified model. There are a multitude of attribution methods in this category, we explain LIME and SHAP here, as particularly popular methods.

LIME

Local interpretable model-agnostic explanations (LIME) [69] is an attribution method that utilizes local explanation models. Local explanation models are simple and interpretable models that should approximate the behavior of the original model for a small region of inputs. To do this, first a sample is chosen x. Then a number of samples, each containing a fraction of non-zero elements of x, are created. Finally the local surrogate model is trained with.

$$\xi(x) = \underset{g \in G}{\arg\min}\, L(f, g, \pi_x) + \Omega(g) \tag{2.9}$$

Here G is a class of potentially interpretable models like decision trees or a linear model, L is a loss depending on the class prediction of a sample $f(z)$ based on the model g and is weighted by proximity to the original sample $\pi_x(z)$. The second term penalizes complexity of the considered surrogate model g. LIME is an occlusion-based path-attribution method [55]. We did not find information on completeness, implementation invariance and linearity for LIME.

SHAP

Shapley additive explanations (SHAP) [52] is an attribution method based on Shapley values. Shapley values are a concept from game theory and can be used to assign the contributions of each input feature to the final prediction, the mean marginal contribution for each feature. To generate the Shapley values a number of samples, each containing a fraction of non-zero elements of x is created. The exact process depends on the specific SHAP variant used, like Kernel SHAP or Tree SHAP. Kernel SHAP for example uses weighted coalitions of input features to train the linear model g quite similar to LIME [55]. SHAP satisfies completeness, implementation invariance and linearity [1]. SHAP is a path-attribution method and occlusion-based [55].

3. Analysis

In this chapter we develop a framework for attribution based augmentation (ABA), build theories around it and develop new methods. We first discuss properties of ABA and present a framework and terminology for it in Section 3.1. We describe the experiment settings for the chapter in Section 3.2. We then analyze adjustment behavior in Section 3.3. We develop two theories for finding good combinations of adjustments in adjustment sets for the augmentation step presented in the framework description in Section 3.4. We analyze attribution methods presented based on suitability for ABA in Section 3.5. Additional empirical findings are presented in Section 3.6. Relations of attribution based augmentation and specific methods for ABA to other regularization techniques and methods are discussed in Section 3.7. We finally use this framework, our theories and the additional findings to construct two new ABA methods in Section 3.8. The evaluation of those methods is shown in Chapter 4.

3.1. Framework

In this section we will develop the framework for attribution based augmentation and define relevant terminology. We make some mentions of apparently unsuitable combinations of components and settings but the majority of analyzing the different components is done in later sections. We show the finished pipeline in Figure 3.1.

3.1.1. Properties

ABA methods have two main components: The first component relates to the attribution map and how it is derived. For this one needs to choose a suitable attribution method together with a attribution target that the attribution method refers to, to obtain an attribution map. We discuss the selection of an attribution method in greater detail in Section 3.5. The attribution target might be a class score or a loss for example. We call this first part *attribution generation step*. The second component refers to the way the attribution map is utilized for augmenting the input. This component might further be divided into the task of selecting parts of the input data to change based on the attribution map and ways in which the selected parts are changed. Those we call *attribution selection step* and *augmentation step*. The attribution generation is covered

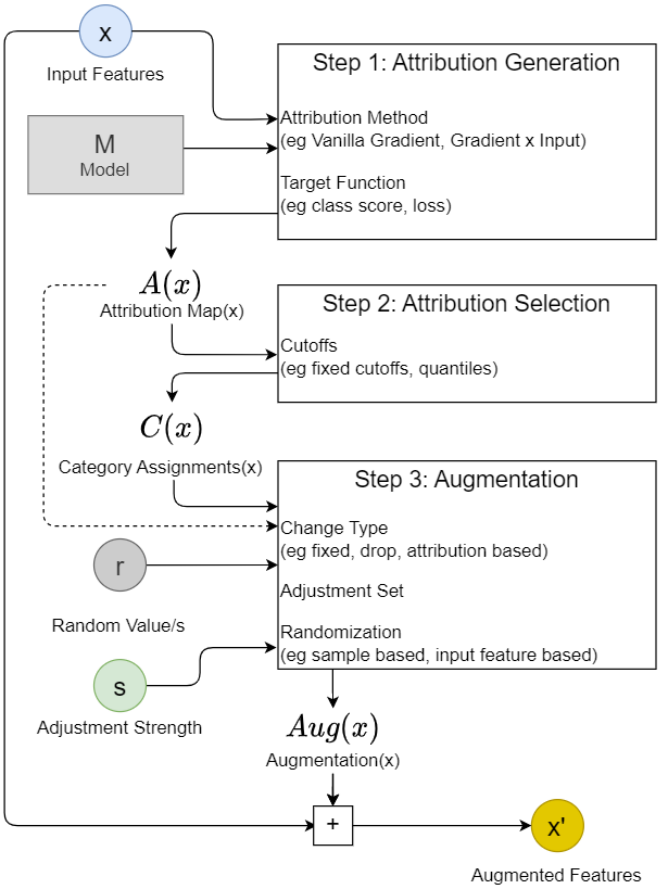


Figure 3.1.: **Pipeline for attribution based augmentation.** The input features of one sample and the model are passed to the *attribution generation step*, where the attribution map is generated based on attribution method and an attribution target. The attribution map is passed to the *attribution selection step*, where cutoffs are used to assign input features to categories. In the *augmentation step* one adjustment from the adjustment set is chosen randomly. The augmentation for input features is generated depending on this adjustment, the categories of the input features and optionally a randomization or attribution information. The results are multiplied with the adjustment strength s to obtain the augmentation that is added to the input features to obtain the augmented sample x'.

by selecting a attribution method and a attribution target potentially with some slight variations[1]. We describe the steps in detail and orient ourselves on Figure 3.1. For the sake of simplicity we will assume a flattened representation for x and the outputs of the steps, which is also the way it is implemented in our code.

3.1.2. Attribution Generation Step

We take one sample in a batch, x. This sample and our model M are passed to the attribution generation step. The attribution map of x, $A'(x)$ is then calculated based on the selected attribution target T and the attribution method E.

$$A'(x) = E(x, T) \tag{3.1}$$

$A'(x)$ is then flattened to $A(x)$ for the next steps. It has the shape of the flattened input dimension $(D,)$.

3.1.3. Attribution Selection Step

The attribution selection step can be used to divide the input features into *categories* based on attribution $A(x)$. We explain how splits might occur for 1, 2 and 3+ categories as they are treated differently. The output of this step is a category assignment for all input features $C(x)$. The category assignments have the shape of (c, D), with c being the number of categories.

- **One category:** Here all input features are selected and treated in the same way in the augmentation step.

$$C_i(x) = 1 \tag{3.2}$$

- **Two categories:** Here a split into a category of high attribution features and a category of low attribution features occurs. This split could be done by either using 0 or other fixed values as the cutoff or by using percentiles, for example the highest 50% being in one category and the lowest 50% being in another category. We show the fixed 0 split in equations here:

$$C_{ki}(x) = \begin{cases} 1, \text{if } k = 1 \ \& \ A_i(x) \geq 0 \\ 1, \text{if } k = 2 \ \& \ A_i(x) < 0 \\ 0, \text{otherwise} \end{cases} \tag{3.3}$$

[1] Many attribution methods specifically mention selecting the class score, but could also be applied on loss or regression score without much difficulty for example.

- **Three+ categories:** Three categories means a split into high, low and middle attribution features, with more than three categories allowing for more subdivision. This is likely most sensible with percentiles, as using fixed values for the cutoffs would require relatively precise knowledge of which attribution magnitudes might appear at what point in the training process based on model and data and the cutoff would have to be adjusted manually, which seems very impractical. We show the three category percentile case here. First the percentiles are calculated based on the flattened attribution map and the percentages v:

$$p = p(A(x), v) \tag{3.4}$$

  In the three category case we have two percentiles, p_1 being the lower cutoff and p_2 being the higher cutoff. These two percentiles are used to assign the categories.

$$C_{ki}(x) = \begin{cases} 1, \text{if } k = 1 \ \& \ A_i(x) < p_1 \\ 1, \text{if } k = 2 \ \& \ p_2 > A_i(x) \geq p_1 \\ 1, \text{if } k = 3 \ \& \ A_i(x) \geq p_2 \\ 0, \text{otherwise} \end{cases} \tag{3.5}$$

$C(x)$ is then passed to the augmentation step. We show a visualization of the attribution selection and the 3 resulting categories in Figure 3.2.

3.1.4. Augmentation Step

The augmentation of each input feature $Aug(x)$ is based on the category it is in and either some fixed value or on the attribution. Randomness might be added in both variants. Another option is to drop an input feature to 0 or another baseline. The category based component is called adjustment.

- **One category case:** If there is only one category, it is likely only sensible for the change to be based on attribution in some way. A change by fixed value would just mean that all inputs are higher or lower leading to little to no regularization. It would also disqualify the method as being attribution based augmentation, as the attribution would not contribute to the augmentation in any way. A change by randomness would also not utilize attribution and would be equivalent to "noise injection on input", discussed in 2.4.1. A change to a baseline for all inputs is also not sensible as there will be no information remaining in the input on which a good prediction could be based on. The change must then be based on attribution in some way. Here one might add the attribution directly, apply some normalization on the attributions to fix them to a certain range of values or apply any other function on them. Randomization might also be applied in conjunction.

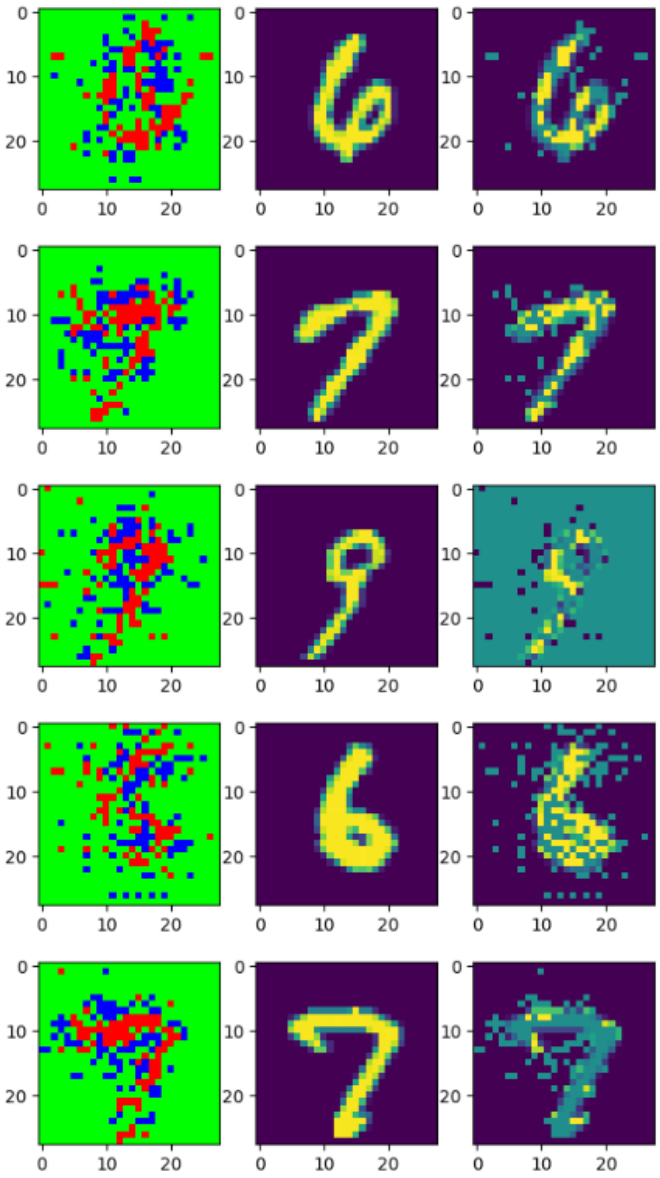


Figure 3.2.: **Visualization for categories and adjustments:** We use MNIST and a percentile split with $v = [10\%, 90\%]$ to show features with low, middle and high attributions (blue, green, red). We then show the original image and the augmented image with $s = 1$ and no randomization. Adjustments used from top to bottom are: [0,0,1],[0,0,1],[0,0,-1],[0,0,1],[1,0,0]

- **Two categories case:** If there are two categories one might still use the attribution information for the change as in the one category case, but might also only use fixed values with optional randomization for the changes, as there is already attribution information contained in the category split. For example by increasing higher attribution features and decreasing lower attribution features. It might also be feasible to set either higher or lower attribution features to a baseline in one of the two categories.
- **Three+ categories case:** In the three or more category case the same applies as in the two category case, just that there are more options for changes. Setting features to a baseline in one category might also be more feasible here, as enough information might be accessible in the other categories for the model to make accurate predictions.

Equations for this step are shown at the end of the next section, where adjustments are explained in detail.

3.1.5. Adjustments

We define a specific way in which changes are applied on input features based on category as adjustment. These adjustment based changes are the basis for the augmentation of x, components like randomization might then be added to that basis $AugBase(x)$. The number of possible adjustments is based on the categories and the possible changes h of each category with number of categories:

$$n_{adj} = n_c^h, \tag{3.6}$$

With 3 categories and possible changes being: increase by 1, decrease by 1 or keep the same, there would be $27 = 3^3$ possible adjustments. Those adjustments can be seen in Table 1. An adjustment $a = [1, 0, -1]$ would mean that the lowest attribution features are increased, the medium attribution features are kept the same and the highest attribution features are decreased.

$$AugBase_i(x) = \sum_{n=1}^{n_c} C_{ni} * a_n \tag{3.7}$$

The multiplication with adjustment strength s is applied afterwards, as well as possibly some randomization and attribution based change. For example, a randomization on a per sample basis without attribution based change would look like:

$$Aug'(x) = AugBase(x)| * s * |\mathcal{N}(0, 1) \tag{3.8}$$

The adjustment strength s is the only hyper-parameter in ABA that can not really be put to a standard value and has to be tuned based on model, dataset and other settings. In the end the flattening is reversed to obtain $Aug(x)$ from $Aug'(x)$ to be able to obtain the augmented sample with:

$$x' = x + Aug(x) \tag{3.9}$$

An illustration of augmentations with adjustments can be seen in 3.2.

3.1.6. Adjustment Sets

One can switch between adjustments used for different samples with defined probabilities to introduce more variation. One adjustment set is provided for training a model and one adjustment is used per sample in the training process. This helps with regularization as shown in Section 3.4. We define this bundle of possible adjustments as adjustment set. It can be presented either with probabilities for each adjustment or, when no probabilities are given, with equal probability for all adjustments in the set. See an example in the next subsection.

3.1.7. Framework Example: Challenger

We revisit the Challenger method presented in 2.4.1 and explain it in our new framework.

1. **Attribution generation step:** The attribution method used is LRP and the attribution target is the class score of a random class from the $top - k$ predictions.

2. **Attribution selection step:** The input features are split into three categories based on percentiles of attribution.

3. **Augmentation step:** The features are then adjusted with fixed values based on the category they are in, with either the highest or lowest input features being changed, either positively or negatively with adjustment strength s and without randomization[2]. Half the time no change occurs. We can show this as adjustment set as

   $A = \{[1,0,0], [-1,0,0], [0,0,1], [0,0,-1], [0,0,0], [0,0,0], [0,0,0], [0,0,0]\}$ or

   $A = \{[[1,0,0], \frac{1}{8}], [[-1,0,0], \frac{1}{8}], [[0,0,1], \frac{1}{8}], [[0,0,-1], \frac{1}{8}], [[0,0,0], \frac{4}{8}]\}$

   Using indexes introduced in Table 3.1, we can also write:

   $A = \{3,4,5,6,0,0,0,0\}$.

[2] In the challenger paper there are slightly different adjustment strengths for each category.

3.2. Experiment Settings

The experiments used to support conclusions and test hypotheses in this chapter are run on the datasets 20Newsgroups, MNIST and CIFAR10. The benchmarks in Chapter 4 use those and additional datasets. To better evaluate the regularizing effect of the experiments in the current chapter, we conduct the experiments on tiny versions of the datasets. An overview of the datasets can be seen in Table 4.1.

- **20Newsgroups:** 20Newsgroups [19] is an NLP dataset containing 20 classes of news articles. It consists of 20000 articles, with around 1000 per class. The tiny version contains 2000 articles with 100 per class.
- **MNIST:** MNIST [15] is a CV dataset containing 10 classes of gray-scale images of the handwritten digits 0 to 9. It consists of 70000 28x28 images, with around 7000 per class. The tiny version contains 100 images with 10 per class.
- **CIFAR10:** CIFAR10 [42] is a CV dataset containing 10 classes of RGB images containing various objects like dogs, ships and cars. It consists of 6000 $32x32$ images, with 600 per class. The tiny version contains 1000 images with 100 per class.

We use VGG11 [82] for MNIST. We slightly diverge from the paper by having an averagepool instead of a maxpool before the fully connected layers and reduced size in the fully connected layers, as the output of the VGG11 with a $28x28$ input is also much lower dimensional than with ImageNet the paper was designed for, we also only output 10 instead of 1000 class scores. We use VGG16 for CIFAR10, here we only changed the number of class score outputs to 10. For 20Newsgroups we use a CNN with 3 convolutional and 2 linear layers. The detailed models are included in the code base. For 20Newsgroups, we also use pre-trained GLOVE embeddings [65] with length of 100 in conjunction with the CNN. Throughout this chapter we use Adam as optimizer and no other type of regularization to better determine effects of experiments in isolation, learning rates are tuned to be close to ideal for each setting to ensure fair comparisons. We use Vanilla Gradient for the adjustment generation step, the choice is justified in Subsection 3.5.3.

3.3. Adjustments in Isolation

After establishing the attribution selection step in Subsection 3.1.3 and unifying the augmentation step with the concept of adjustment sets in Subsection 3.1.5 and Subsection 3.1.6, we need a theoretical basis for understanding which adjustment sets might

be most useful for attribution based augmentation. To do this, we want to build some intuition behind what different adjustments should do in isolation. We first group adjustments based on their expected influence on loss, accuracy and class score in inference and test our predictions as a sanity check. We then reason about influence of these adjustments on validation accuracy when applied while training.

3.3.1. Adjustment Groups

To group adjustments, we use the expected influence of these adjustments on the metrics loss, accuracy and class score of target class. Here "improving the metrics" means decrease of the loss, increase of target class score and increase of accuracy. Consequently "worsening the metrics", means increase of loss, decrease of target class score and decrease of accuracy. Given small adjustment strength s, we define adjustments that tend to improve the metrics at test time *Delta*-adjustments, we define adjustments, that tend to worsen the metrics at test time *Epsilon*-adjustments and we define adjustments that tend to neither increase nor decrease the metrics much at test time *Neutral*-adjustments. The reason these definitions are somewhat inexact is explained in 3.3.1.

On Inexactness of Definitions

As described in Section 3.5, there are different attribution targets and attribution methods the attribution map can be derived from. The most common are the cross entropy loss and class scores. If our attribution target is cross entropy and our attribution method is vanilla gradient, any negative attribution means that an infinitesimal increase in the corresponding input feature leads to lower loss and most likely also an increase in the class score for the correct class and higher accuracy. If our attribution target is the class score of the label class and our attribution method is vanilla gradient, any positive attribution means that an infinitesimal increase in the corresponding input feature, leads to an increase in class score for the correct class and most likely also lower loss and higher accuracy. If our attribution method is not vanilla gradient, there is no guarantee for the behavior of any of these metrics after augmentation, even with infinitesimal s. For practicality we do not define a separate category for each possible attribution target and attribution method, but rather the slightly inexact generalization that both adjustments described above are *Delta*-adjustments and tend to lower loss, increase class score and increase accuracy. *Neutral*-adjustments are also almost guaranteed to either improve or worsen some of the metrics, but the change should be much less pronounced.

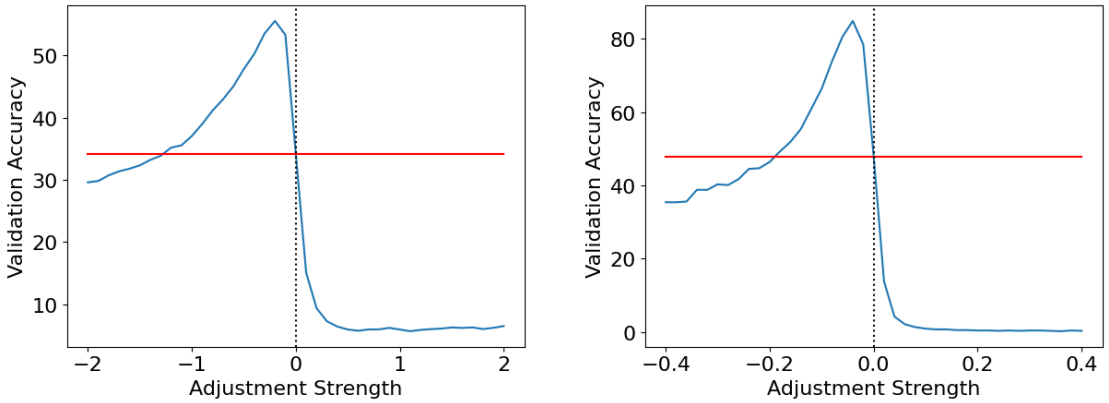
3.3.2. Adjustments at Test Time

As a sanity check we take trained models for the datasets MNIST, CIFAR10 and 20Newsgroups, freeze the weights and vary the strength s for a Delta-adjustment and an Epsilon-adjustment. The expectation being that Delta-adjustments increase accuracy[3] for increasing s up to a certain point and then decrease again when the adjustment strength gets too high. For Epsilon-adjustments the expectation would be an immediate decrease in accuracy, getting worse with highers s. For the experiment we use vanilla gradient as attribution method and cross entropy as the attribution target for the attribution generation step. We use the 3 category case with highest 10% and lowest 10% for percentile calculation in the attribution selection step. As Delta-adjustment we use $[1, 0, -1]$ and as Epsilon-adjustment, we use $[-1, 0, 1]$. The categories being low attribution features, middle attribution features, high attribution features. The results can be seen in Figure 3.3 for CIFAR10 and 20Newsgroups, a similar figure can be found for MNIST in Appendix A.1. The expectations described in this subsection were met. The only surprise being that for stronger Epsilon adjustments, the validation accuracy gets worse than random chance (10% for CIFAR10), before recovering for even stronger adjustments. If the chart was continued to even stronger Delta-adjustments, we expect validation accuracy would reach around 10% at some point too for CIFAR10 or 5% for 20Newsgroups.

3.3.3. Adjustments in Training

Here we divide all adjustments for the 3-category case into the adjustment groups Delta, Epsilon and Neutral and train the models instead of freezing them. Our assumption would be that when training models with the Delta-adjustments and Epsilon-adjustments, but validating on the unmodified validation set, the effects discussed in Subsection 3.3.2 and shown in Figure 3.3 would be reversed. The reason being that it should be easier to overfit for models when the training set is augmented in a way that makes it simpler for the model to correctly classify, which is the consequence of Delta-adjustments, and that overfitting should decrease when the training set is constantly augmented in a way that makes it harder for the model to fit the data, which is the consequence of Epsilon-adjustments. For the experiment we use vanilla gradient as attribution method and cross entropy as the attribution target for the attribution generation step. We use the 3 category case with highest 10% and lowest 10% for percentile calculation in the attribution selection step. The categories are low attribution features, middle attribution features, high attribution features. The strength s was

[3]We look at accuracy here as one of the 3 metrics we based the Delta, Epsilon and *Neutral* distinction on, but not directly targeted in either the cross entropy or class score setting.



(a) CIFAR10 (b) 20Newsgroups

Figure 3.3.: **Delta-adjustments and Epsilon-adjustments on frozen models for CIFAR10 and 20Newsgroups:** Both Delta-adjustments and Epsilon-adjustments are shown in each figure, with negative strength values showing Delta-adjustments and positive strength values Epsilon-adjustments. The red line displays the baseline accuracy with no adjustment.

tuned per adjustment to allow for fair comparison between the adjustments. The results can be seen in Table 3.1 for models trained on 20Newsgroups and CIFAR10, a similar table can be found for MNIST in Appendix A.1. The MNIST table also includes a description of a more efficient implementation that we use in our code. The index in the table is used to refer to adjustments more easily in later parts. The right side of the table shows the maximum and average validation accuracy after training, for 5 iterations of training. The table is colored depending on a rough grading of how generalization is improved. The results align exceptionally well with the predictions for 20Newsgroups. For CIFAR10 the predicted tendencies still hold but to a much lesser degree, with Epsilon-adjustments being helpful in training most of the time, Neutral-adjustments being almost as good and Delta-adjustments being worse than both, but with enough variance that sometimes the validation accuracy even improves for Delta-adjustments.

3.4. Tension and Variety in Adjustment Sets

After looking at adjustment sets in isolation, we explore the interaction of different adjustments in adjustment sets. The first assumption one could make is that an adjustment set results in the average regularization, expressed in validation accuracy

Low	Middle	High	Type	Index	MAX 5 runs	AVG 5 runs	Rating
0	0	0	Neutral	0	36.16	**34.73**	Baseline
0	1	0	Neutral	1	39.91	**38.73**	Top
0	-1	0	Neutral	2	37.09	**36.49**	Good
0	0	1	Epsilon	3	36.49	**35.01**	Medium
0	0	-1	Delta	4	36.13	**33.77**	Medium
1	0	0	Delta	5	36.00	**34.73**	Medium
-1	0	0	Epsilon	6	38.25	**34.53**	Medium
0	1	1	Epsilon	7	39.39	**38.62**	Top
0	-1	1	Epsilon	8	36.17	**35.03**	Medium
0	1	-1	Delta	9	36.21	**34.21**	Medium
0	-1	-1	Delta	10	34.73	**33.21**	Bad
1	1	0	Delta	11	37.84	**34.23**	Medium
1	-1	0	Delta	12	34.63	**33.89**	Medium
-1	1	0	Epsilon	13	39.19	**37.90**	Top
-1	-1	0	Epsilon	14	36.78	**36.03**	Good
1	0	1	Neutral	15	37.35	**36.76**	Good
1	0	-1	Delta	16	39.30	**35.96**	Good
-1	0	1	Epsilon	17	37.44	**36.11**	Good
-1	0	-1	Neutral	18	37.11	**35.86**	Good
1	1	1	Neutral	19	37.95	**36.14**	Good
1	-1	1	Neutral	20	36.93	**36.74**	Good
1	1	-1	Delta	21	37.20	**36.10**	Good
1	-1	-1	Delta	22	37.29	**34.38**	Medium
-1	1	1	Epsilon	23	39.30	**37.53**	Top
-1	-1	1	Epsilon	24	36.17	**35.33**	Medium
-1	1	-1	Neutral	25	38.65	**37.89**	Top
-1	-1	-1	Neutral	26	35.78	**33.78**	Bad

(a) CIFAR10

Low	Middle	High	Type	Index	MAX 5 runs	AVG 5 runs	Rating
0	0	0	Neutral	0	49.1	**47.2**	Baseline
0	1	0	Neutral	1	47.6	**46.44**	Medium
0	-1	0	Neutral	2	48.9	**47.02**	Medium
0	0	1	Epsilon	3	55.3	**53.12**	Good
0	0	-1	Delta	4	42.6	**41.88**	Bad
1	0	0	Delta	5	45.1	**41.28**	Bad
-1	0	0	Epsilon	6	57.5	**53.94**	Good
0	1	1	Epsilon	7	57.5	**55.86**	Top
0	-1	1	Epsilon	8	56.3	**53.82**	Good
0	1	-1	Delta	9	44.1	**42.06**	Bad
0	-1	-1	Delta	10	43.6	**42.38**	Bad
1	1	0	Delta	11	44.6	**41.46**	Bad
1	-1	0	Delta	12	46.3	**43.08**	Bad
-1	1	0	Epsilon	13	59.5	**57.98**	Top
-1	-1	0	Epsilon	14	57.6	**54.34**	Good
1	0	1	Neutral	15	49.4	**47**	Medium
1	0	-1	Delta	16	48.9	**44.68**	Bad
-1	0	1	Epsilon	17	57.6	**55.3**	Top
-1	0	-1	Neutral	18	47.9	**45.7**	Medium
1	1	1	Neutral	19	50.6	**48.14**	Medium
1	-1	1	Neutral	20	52.3	**46.72**	Medium
1	1	-1	Delta	21	47.7	**42.88**	Bad
1	-1	-1	Delta	22	45.6	**42.82**	Bad
-1	1	1	Epsilon	23	57.3	**53.08**	Good
-1	-1	1	Epsilon	24	58.3	**55.86**	Top
-1	1	-1	Neutral	25	51.4	**48.7**	Medium
-1	-1	-1	Neutral	26	49.2	**46.56**	Medium

(b) 20Newsgroups

Table 3.1.: **Comparison of adjustments in training for CIFAR10 and 20Newsgroups:** The 27 adjustments are sorted into equally large adjustment groups of Delta, Epsilon and Neutral adjustments. Validation accuracy after training aligns exceptionally well with adjustment group for 20Newsgroups, for CIFAR10 the tendency is less pronounced.

after training, of adjustments in the set. Instead we found that even when "diluting" the Epsilon-adjustments, that are much better in isolation, with Delta-adjustments performance improves. One theory we developed from this is that having opposing adjustment groups in an adjustment set helps the model generalize better. We call this theory *adjustment tension*. Another explanation of where the increase in generalization comes from would be that just having different adjustments in the set, no matter the adjustment group, is beneficial. We call this theory *adjustment variety*. Both theories might be true to some degree. We tried to refute these theories using our development datasets. We did this by creating subsets of the 27 adjustments displayed in Table 3.1, each containing the 9 adjustments of their corresponding adjustment group. This is done by comparisons of adjustment set sizes and composition:

- **Adjustment Variety Theory:** We looked at validation accuracy of a model trained with an adjustment set containing all 9 adjustments of one adjustment group and compared this to the average validation accuracy of 9 models trained with adjustment sets containing only one adjustment each. If the adjustment variety theory is incorrect, using all adjustments of a group in a set should lead to equal regularization, as the greater variety in the set would make no difference. The results are shown in Table 3.2. The data shows some support for adjustment variety theory. On MNIST there is good improvement, on CIFAR10 there is some decent improvement and on 20Newsgroups there does not seem to be much improvement.
- **Adjustment Tension Theory:** We now look at performance of using adjustments from different adjustment groups in one adjustment set. If the adjustment tension theory is incorrect, we would expect no difference in improvement to the single adjustment group setting, no matter if adjustment variety theory is true or not. The results of this experiment can also be seen in Table 3.2. The results from all datasets show significant improvements for combining the opposed Delta-adjustments and Epsilon-adjustments, much beyond the improvements seen from intra-group variety alone, giving great support to adjustment tension theory.

We postulate adjustment tension as main reason for Delta-adjustments and Epsilon-adjustments performing better in conjunction, with a smaller part of the improvements coming from adjustment variety. We could not explore the exact mechanism of action here and hope that future research can build a more complete understanding of this phenomenon.

3.5. Evaluation of Attribution Methods

To evaluate the attribution methods outlined in Section 2.5, we first describe what characteristics of an attribution method might be useful, necessary or detrimental in context of ABA. We also focus on attributions for input features. The attribution target for these attribution methods is usually the class score as most are developed for classification, but most should also be easy to modify for other tasks like regression.

3.5.1. Evaluation Criteria

We recognized different characteristics in the areas flexibility, cost in compute and implementation and quality of attribution maps, which we will describe in this section.

Adj. group	All of groups	AVG of groups	Difference
Neutral	37.43	36.35	1.08
Delta	34.98	34.50	0.48
Epsilon	39.90	36.23	3.66
Eps. & Neu.	40.40	36.29	4.11
Eps. & Del.	40.27	35.37	4.90
All	40.26	35.69	4.56

(a) CIFAR10

Adj. group	All of groups	AVG of groups	Difference
Neutral	46.62	47.05	-0.43
Delta	45.02	42.50	2.52
Epsilon	53.84	54.81	-0.97
Eps. & Neu.	56.50	50.93	5.57
Eps. & Del.	57.53	48.66	8.88
All	58.94	48.12	10.82

(b) 20Newsgroups

Adj. group	All of group	AVG of groups	Difference
Neutral	76.82	74.59	2.23
Delta	76.64	73.43	3.21
Epsilon	83.51	80.83	2.68
Eps. & Neu.	82.14	77.71	4.43
Eps. & Del.	84.55	77.13	7.42
All	83.72	76.28	7.44

(c) MNIST

Table 3.2.: **Experiments on adjustment set performance:** The "all of groups" column shows the validation accuracy for one model trained on one adjustment set consisting of all adjustments from the adjustment groups listed in the first column. The "AVG of adj." column shows the average validation accuracy for models trained on adjustment sets containing only a single adjustment each, one model per adjustment in the adjustment group. The "Difference" column shows improvements from using adjustment sets with multiple adjustments, from the adjustment group or groups, in the same set. While improvements using a single adjustment group compared to the constituent adjustments are only moderately high, generalization improves considerably when different groups are put in the same set. Especially for the Epsilon and Delta groups.

- **Domain Dependence**: Some attribution methods are only applicable in specific domains. This stands in direct opposition to the aspired advantage of domain independence for ABA, which would be the distinguishing feature compared to techniques like hand-crafted data augmentation. This is why we think attribution methods with this characteristic are not well suited for an ABA method.
- **Architecture Dependence**: There are also some methods that can not handle specific model architecture elements. Considering the goal of making ABA methods as general as possible, we also think that those methods are not well suited.
- **Baseline Dependence**: The need for manually selected baselines or features is also sometimes a characteristic of attribution methods. This is also best avoided because it is additional work for the practitioner and often necessitates some understanding of the domain or task, it is a less severe limitation than domain and architecture dependence though, as the amount of domain knowledge needed to set sensible baselines should not be very high and practitioners usually have some domain knowledge for tasks they perform.
- **Compute intensiveness**: It is also relevant how compute intensive a method is, as deep learning is resource intensive already and any additional computational burden should be limited. High compute intensiveness is a less severe limitation as a method not being general enough though. Especially as the need for regularization is often based on a lack in data, so that when regularization is most needed, compute is often not the greatest bottleneck but data.
- **Ease of implementation**: Methods differ a lot in how hard they are to implement, which is an important consideration if practitioners and researchers are to use the method.
- **Attribution map quality**: The last characteristic we identified as relevant for ABA methods is the quality of the attribution map. This metric is very hard to quantify. When reading different papers in this area, there are some methods that appear to be superior to others in how well the visualization corresponds with our human intuition of features that are relevant in the input. The problem here is that this quality assessment based on human intuition is only available in domains where we have good intuitions and also that it is not evident that attribution maps that better align with human intuition are better suited for ABA methods. We also found only one paper comparing attribution map quality by human assessment, which only included one attribution method in our selection and three in total, it is from 2021 and claims to be the first doing such an assessment [59]. Consequently

we can only use our own intuition based on examples in papers, as doing a human study for such an assessment would exceed the scope of this minor subarea of the thesis. We want to emphasize that our assessments based on a few examples in a small selection of papers are of limited usefulness though and should not be taken as an accurate assessment, only as a very rough evaluation. Another avenue of assessing quality is using the properties of completeness, implementation independence and linearity defined in Subsection 2.5.1, but it is uncertain how they relate to attribution map quality and again how that relates to suitability for ABA methods. In the end we still think that this very roughly approximated attribution map quality might be a beneficial characteristic, but do not put high priority on it for the scope of this thesis.

3.5.2. Evaluation

Here we evaluate the different attribution methods presented in Section 2.5. At the end of the section we select the most promising methods under different assumptions and discuss the selection for the experimental part of this thesis.

- **LRP and Gradient x Input:** As far as we could discern LRP is domain independent but not completely architecture independent, specifying own rules for layers and having difficulty with handling special layers like batch norm. LRP also necessitates a baseline and is relatively difficult to implement, especially for the special layers. It is not very compute intensive as a gradient-based method that only uses one backward pass, but might be a bit slower than some other gradient-based methods because of different and probably less optimized gradient calculation. The method satisfies completeness but breaks implementation invariance and the attribution maps looks moderately good to us so we estimate that the attribution map quality is medium high. As Gradient x Input seems to have all the strengths of LRP while being slightly faster, having easier implementation and architecture independence and no baseline, we recommend using Gradient x Input instead.

- **DeepLift:** DeepLift is domain independent but suffers from needing special implementation rules like LRP, which makes it not completely architecture independent. It is also relatively fast, being gradient based and generating the attribution map in one backward pass but probably slower than optimized methods that use a less modified backward pass. The implementation is relatively difficult, pretty similar to LRP. It does have baseline dependence. It satisfies completeness but breaks implementation invariance. Samples provided in the paper look very good to us, we also found another independent paper [2] where examples of different methods are shown and DeepLIFT looks best, the original DeepLift paper [78]

also argues for some additional characteristics of the method that could lead to higher attribution map quality.

- **Integrated Gradients:** Integrated Gradients is domain independent and architecture independent but needs a baseline. Is is highly compute intensive as a backward pass has to be calculated in numerous steps for each sample. The implementation is relatively easy as the standard backward pass can be used and only the intermediate steps have to be implemented. The attribution map quality looks pretty high in their paper and their method fulfills completeness, implementation invariance and linearity.
- **Vanilla Gradient:** Vanilla Gradient is very flexible and can be used in every area of DL as it uses the standard backward pass. It needs no baseline, is least compute intensive and easiest to implement. The only detriment is the low attribution map quality, as the examples we saw in papers were worse than most of the other attribution methods.
- **DeconvNet and Guided Backpropagation:** These methods are domain independent but not architecture independent, as they add special rules for ReLU. They have no baseline, need little compute and are relatively easy to implement. We did not find many comparisons with these methods in papers, but think that thy will likely be similar to Vanilla Gradient. Another small disadvantage is that Guided Backpropagation and DeconvNet can fail to highlight negatively contributing input features due to discarded negative gradients [78].
- **CAM and Variants:** CAM itself is domain independent but severely architecture limited only working on a subset of CNNs. The upgrades like Grad-CAM are still architecture dependent but work on all CNNs, there is no baseline dependence, but they are relatively compute intensive. The implementation is not that easy but attribution map quality seems to be relatively high from our assessment.
- **Smoothgrad:** Smoothgrad as an extension to other gradient based methods offers a trade-off of slightly better attribution maps for considerably more compute.
- **LIME:** LIME is one of the most flexible attribution methods, being domain, task and architecture independent, even extending to non-DL models. Training a local surrogate model for each sample is extreme compute intensive though and it is hard to implement too, which is why we do not think it is a good fit for ABA.
- **SHAP:** SHAP is similar to LIME in that it is highly flexible but very compute intensive and hard to implement, which is why we do not think that SHAP is a good fit for ABA.

3.5.3. Selection for Thesis

We think that the best methods for ABA methods will likely be Vanilla Gradient, Gradient x Input, Integrated Gradients and potentially the addition of Smoothgrad. They are all general and easy to implement. We think that for settings were compute is more limited, Vanilla Gradient or Gradient x Input are likely best. For settings with abundant compute Smoothgrad might be added to Vanilla Gradient or Gradient x Input and Integrated Gradients might also be a good choice in this setting. Because this thesis necessitates many experiments to be run and benchmarks to be created, we choose from the compute limited options. Gradient x Input seems to have slightly better attribution map quality, but for the scope of this thesis we concentrate on Vanilla Gradient, as the simplicity makes it easier to test hypotheses on it and we focus more on building an understanding of the attribution selection step and the augmentation step, than optimizing the attribution generation step.

3.6. Additional Findings

The findings in this section are not as well supported as the previous sections. We include it here to give some context for our decisions in the construction of new methods and also as a basis on which future work might build on. We orient ourselves on Figure 3.1.

3.6.1. Attribution Target

We tried changing the attribution target with different settings and consistently obtained better performance for choosing random class scores as target, compared with the cross entropy loss. The experiments for optimal k yielded pretty similar values, with $k = 5$ being best for MNIST and CIFAR10 and $k = 4$ being best for 20Newsgroups. The differences were relatively minor though.

3.6.2. Cutoffs

In the percentile setting with 3 categories, using 10% as highest and lowest cutoff each, performed consistently pretty well. In the 2 category case we did not compare percentiles vs the fixed cutoff 0. We do suspect that the differences between them will be minor though and would not justify investing the additional compute to calculate the percentiles, compared with using the 0 cutoff and just differentiating negative and positive attributions. One caveat is that if one chose to utilize an attribution method

without negative attributions, the percentile calculation would likely be necessary as reasoned in Subsection 3.1.4.

3.6.3. Change Type

The attributions are used in the attribution selection step to create the different categories of input features. Using the attribution again for determining how much the features should be changed yielded worse or at most equal results to using fixed values for all settings. Dropping all features of certain categories to 0 or another baseline also did not yield worthwhile results in a series of smaller experiments.

3.6.4. Adjustment Sets

There are no adjustments that are better than all others across the datasets. We did observe a trend that *opposite* adjustments in one adjustment set seem to work well as optimal tension is ensured. Opposite adjustments would be indexes $21 => [1, 1, -1]$ and $24 => [-1, -1, 1]$ or $3 => [0, 0, 1]$ and $4 => [0, 0, -1]$ for example. We also found that including neutral adjustments that change the medium features is often advantageous and better than the 0 adjustment.

3.6.5. Randomization

Randomization yielded benefits in many settings. The type we tested was calculating the absolute of one zero mean Gaussian per sample and multiplying it with all augmentation values for the sample, the equation can be seen in Equation 3.8.

3.7. Similarities to Different Regularization Techniques

We saw some similar concepts in our exploration of ABA that appear in other regularization techniques.

- **Noise Injection on Inputs:** The case of only having one category and Gaussian randomization on input features with fixed change values is the same as Noise Injection on Inputs as pointed out in Subsection 3.1.4. This case is not sensible as an attribution map is calculated without being utilized of course. The idea of applying randomization is kept in the augmentation step though.
- **Adversarial Training:** FGSM for adversarial training is entailed in our ABA framework. It is the special case of having Vanilla Gradient as attribution method, cross entropy as the attribution target, two categories with 0 as cutoff in the

input selection step and then using an adjustment set that contains only one Epsilon-adjustment:

$A = \{[-1, 1]\}]$

If tension theory is correct one should be able to construct an improved ABA based on FGSM by adding the opposite Delta-adjustment to this Epsilon-adjustment in one adjustment set. This should improve validation accuracy of the trained model but the model should still benefit from adversarial defense. A few experiments we ran looked promising but we do not have the resources to run enough experiments to really test this prediction. We included one experiment pointing towards greater adversarial defense in Appendix A.1.

- **Dropout:** The setting in the augmentation step, where certain categories are set to 0 or another baseline can be viewed as an attribution guided variant of dropout.

3.8. Construction of Methods

We construct two new methods, the first is based on our findings and we create another variant optimized for speed. We call them MendABA for an ABA method that utilizes randomization and medium attribution input features and FastABA for an ABA optimized for speed. In the next chapter we will create benchmarks for these two methods and a slightly altered and simplified variant of the Challenger method described in Subsection 3.1.7.

- **Attribution generation step:** We use Vanilla Gradient with a randomly selected class score as the attribution target for all 3 methods. This is a change for the original Challenger using LRP.
- **Attribution selection step:** For MendABA and Challenger we use a 3-category split based on percentiles, having the lowest 10%, highest 10% and medium 80% attribution features in one category each. For FastABA we only have a 2-category split with 0 as the cutoff, yielding the negative attribution features in one category and the positive attribution features in the other.
- **Augmentation step:** For Challenger we use no randomization like in the original paper. For MendABA and FastABA we use randomization with an absolute Gaussian $|\mathcal{N}(0, 1)|$ on a per sample basis The adjustment set for Challenger in indexes from Table 3.1 is:

  $A = \{0, 0, 0, 0, 3, 4, 5, 6\}$

  The adjustment set for MendABA is:

$A = \{1, 2, 3, 4, 5, 6\}$

The adjustment set for FastABA is:

$A = \{0, 0, 3, 4, 5, 6, 16, 17\}$

We fine-tuned the adjustment sets for the 2 new methods with pairs of opposite adjustments to ensure tension until we found good sets. It is interesting to see that the Challenger method already converged to a set of opposite adjustments with tension.

4. Benchmarks

This chapter is about measuring performance and creating benchmarks for the methods presented at the end of Chapter 3. We benchmark the ABA methods Challenger[1], MendABA and FastABA. We will first introduce the datasets and models and then explain the training and evaluation procedures. Finally we will show and describe results briefly. We discuss the results and their meaning in the context of ABA more deeply in Chapter 5.

4.1. Settings

We test the methods with a multitude of models including different model architecture elements, in conjunction with other regularization techniques and in different domains to verify that the methods lead to improved performance in a wide variety of settings.

4.1.1. Datasets

We use the datasets MNIST, CIFAR10, 20NewsGroups, PTB-XL [97] and Tiny ImageNet[2] [45]. The dataset descriptions for MNIST, CIFAR10 and 20NewsGroups and their tiny variants can be found in Section 3.2. For MNIST we add an additional subset that contains 1000 samples, with 100 per class. The additional datasets are:

- **Tiny ImageNet:** Tiny ImageNet is a CV dataset containing 200 classes of RGB images. It consists of 100000 $64x64$ images, with 500 per class. While it is called "tiny" it is still the biggest dataset we run experiments on.
- **PTB-XL:** PTB-XL [85] is a time-series dataset containing electrocardiography (ECG) data. ECG is a non-invasive tool to assess cardiac condition of patients to identify cardiovascular diseases. The dataset contains 21799 samples from 18885 patients with 10 seconds of ECG data each. It contains 71 statements about

[1] With the slight modifications discussed in Section 3.8.

[2] We started with the original ImageNet but the size of the dataset is so large, that models are either too small to overfit on the large dataset or so large that we can not train them enough times over all the different settings in a reasonable time frame with limited resources.

diagnostics, form and rhythm of the ECG data. We use a sampling frequency of 100Hz and follow [85] closely in implementation.

We use a train, validation and test set split for these experiments. We use the validation set for finding good hyperparameters and the test set only at the end of the process, for generating the results. An exception for this is Tiny ImageNet where we use the validation set for both, as the code[3] we built on uses the validation set in testing and we noticed too late to redo the experiments in time. An overview of the dataset settings is displayed in Table 4.1.

Dataset	Train-Full	T-Med	T-Tiny	Valid	Test	Input Dim	Classes
MNIST	50000	1000	100	10000	10000	28x28x1	10
CIFAR	40000	-	1000	10000	10000	32x32x3	10
20Newsgroups	14866	-	2000	1000	4000	1000x100	20
Tiny ImageNet	100000	-	-	10000	10000	64x64x3	200
PTB-XL	17441	-	-	2193	2203	1000x12	71

Table 4.1.: **Settings for datasets in benchmarking:** The numbers from column "Train-Full" to column "Test" show the number of samples in the subsets.

4.1.2. Models

We continue using the 3 models described in Section 3.2 for MNIST, CIFAR10 and 20Newsgroups. The experiments are repeated with the inclusion of batch normalization in the models, we also tune learning rate α and adjustment strength s again for this setting. As Tiny ImageNet is the largest and most difficult dataset in our benchmarks, we tested it most thoroughly to see the limitations of ABA. We trained 3 different models on it, the CNNs ResNet50 [31] and EfficientNet-B3 [91] and a Vision Transformer [18]. For PTB-XL we use a fully convolutional network (FCN) [100]. The Vision Transformer is based on ViT-B/16 from [18], with some small changes to adjust for the reduced input dimensions. We choose a patch size of 8 to get $8x8$ patches from the $64x64$ input of Tiny ImageNet. The exact implementations of all models are in the code base.

4.1.3. Other Settings

We train our models on the different datasets and subsets without ABA methods as a baseline or in conjunction with either Challenger, MendABA, FastABA. We first tune the learning rate for the baseline with the validation set and then use the test set for a

[3] https://github.com/pytorch/examples/tree/main/imagenet

fair evaluation of performance. For the ABA methods we reuse the learning rate α of the baseline, use the validation set to tune adjustment strength s and then run the test set. The metric we compare is validation or test accuracy. Throughout this chapter we use Adam as optimizer. For evaluation we train and test models 9 times for MNIST, CIFAR and 20Newsgroups, 5 times on the Tiny ImageNet CNNs and PTB-XL and 3 times on the vision transformer for Tiny ImageNet due to compute constraints. We compare medians of validation accuracy for these runs as an indicator of typical performance. We choose median instead of average accuracy as sometimes results were negatively impacted by outliers from subpar initialization. We also calculate the standard deviation between runs to indicate how much randomness is involved between the different training runs. For Tiny ImageNet we use other regularization sources described in Section 2.4 in conjunction with our ABA methods. We use weight decay in one setting and weight decay and hand-crafted data augmentation RandAugment in another for all 3 models. For PTB-XL we also use weight decay. Vanilla Gradient as attribution method is used throughout this Chapter. The attribution targets are usually the class scores, with the exception of PTB-XL where the attribution target is cross entropy loss.

4.2. Results

We first test our models with different settings on the basic datasets MNIST, CIFAR10 and 20Newsgroups for tiny and full datasets and either with or without batch normalization. We then look at the models trained on Tiny ImageNet and compare performance between them and the improvements from ABA methods. We finally look at PTB-XL results.

4.2.1. MNIST, CIFAR10 and 20Newsgroups

We include the distilled version of the results in Table 4.2 showing only the test set results and omitting standard deviation and augmentation strength, the full table can be seen in the Appendix A.2.

Result Description: Standard Models

When no batch normalization is used improvements from ABA are vast across the different datasets. For MNIST the test error reduces by half for the tiny and medium variant and improves slightly for the full dataset. For CIFAR10 the improvements are still considerable for the tiny dataset but much smaller for the full dataset. For 20Newsgroup the improvements are large for the tiny dataset and for the full dataset.

Dataset	Subset	Epochs	Vanilla	Challenger	MendABA	FastABA
MNIST	Tiny	1000	63.1	83.1	82.8	84.7
	Med	100	91.6	96.0	95.9	96.2
	Full	10	99.1	99.2	99.2	99.2
CIFAR	Tiny	500	33.5	38.4	41.0	38.5
	Full	30	73.8	74.5	74.8	74.7
NEWS	Tiny	100	46.2	57.2	60.5	58.1
	Full	30	66.7	72.8	72.9	73.0
MNIST	Tiny	500	83.3	83.4	83.6	85.4
	Med	50	96.7	96.6	96.9	96.9
	Full	10	99.2	99.3	99.3	99.3
CIFAR	Tiny	200	45.6	45.9	45.5	45.4
	Full	30	82.5	83.7	83.4	83.5
NEWS	Tiny	50	46.9	57.9	59.0	57.5
	Full	30	65.7	74.2	74.8	74.4

Table 4.2.: **Results for MNIST, CIFAR10 and 20Newsgroups:** Displayed are the test accuracies for the datasets and subsets with the different ABA methods and regular training. The upper half of the table shows results for the standard models for the datasets without batch normalization, the second half shows results for those models with batch normalization.

Between the different ABA methods it seems like MendABA is slightly superior to Challenger and FastABA, while FastABA and Challenger are relatively similar.

Result Description: Batch Normalization Models

For models using batch normalization, advantages of ABA are much smaller for the vision datasets, having almost no difference between regular training and training with ABA inclusion. The baseline values improve dramatically with the inclusion of batch normalization for MNIST but only very slightly for ABA methods. For CIFAR test accuracy improves for both the baseline and ABA methods but the gap between baseline and ABA almost closes completely. The 20Newsgroup model sees almost no change with the inclusion of batch normalization. Gaps between the ABA methods also close in this setting with no method being clearly superior.

4.2.2. Tiny ImageNet

We include the distilled version of the results in Table 4.3 omitting standard deviation and augmentation strength, the full table can be seen in the Appendix A.3.

Model	RandAug	Epochs	Vanilla	Challenger	MendABA	FastABA
ResNet50	No	100	31.31	32.74	32.86	33.13
ResNet50	Yes	100	46.93	45.75	47.98	46.71
EfficientB3	No	300	41.33	40.94	42.11	42.18
EfficientB3	Yes	300	53.59	53.29	53.66	53.63
VIT Base 16	No	50	28.20	29.95	31.77	33.01

Table 4.3.: **Results for Tiny ImageNet:** Displayed are the test accuracies for the different models, and the different ABA methods and regular training.

Result Description: Tiny ImageNet

For the CNNs, ResNet50 and EfficientNet-B3 without RandAugment the ABA methods improve results slightly. The regularization method RandAugment outperforms the ABA methods by a wide margin and ABA in conjunction with RandAugment does not seem to improve performance. Performance of ABA for the vision transformer looks better, but it seems likely that RandAugment is superior here too. Between the ABA methods MendABA and FastABA look slightly stronger than Challenger overall. Accuracy for the vision transformer is relatively low. One reason for that is that hyperparameters are likely sup-optimal, as compute was very limited in the weeks we ran the vision transformer experiments, so the search for good hyperparameters was less thorough than ideal and the results for the vision transformer must be seen with some caution. Some of the low performance could also be due to limitations of the architecture on small datasets, we discuss this in more detail in the Appendix A.2.

4.2.3. PTB-XL

The distilled results for PTB-XL can be seen in Table 4.4. The performance between the ABA methods and the baseline is very similar.

Dataset	Vanilla	Challenger	MendABA	FastABA
PTB-XL	91.40	91.50	91.50	91.40

Table 4.4.: **Results for PTB-XL:** Displayed are the median test accuracies for the different attribution based augmentation methods and regular training.

5. Discussion

We set out to expand the understanding of attribution based augmentation and introduce improved methods from our findings to decrease overfitting. In this Chapter we will reiterate major findings, give an interpretation of our results and discuss implications.

5.1. Major Findings

The framework incorporating the concepts of adjustments and adjustment sets lead to two major findings. The first is the concept of the adjustment groups Delta, Epsilon and Neutral and their influence on training shown in Table 3.1. Here one can see that in training, any adjustment that changes the input in a way that makes it easier for the model to make a correct prediction tends to decrease model performance and any adjustment that makes it harder for the model to make a correct prediction tends to increase model performance. The second major finding is that model performance seems to improve most when opposed adjustment groups are presented to the model in the training process, even when those adjustments are detrimental to the model when presented in isolation. This is supported well by Table 3.2. We termed this phenomenon "adjustment tension". The data from Chapter 4 points toward high performance of ABA for small datasets and simple models, which is supported by Table 4.2.

5.1.1. Limitations

The results from Chapter 4 also point towards limitations for ABA methods. The first is that ABA methods seem to be much less effective on bigger datasets, which can be observed across all experiments. Another limitation is that ABA methods seem to work less well with batch normalization than without as can be seen in Table 4.2. One other limitation that became apparent in the experiments shown in Table 4.3, is that well-tuned hand-crafted data augmentation seems to outperform ABA methods by a wide margin. Interpretations of the findings and limitations are discussed in the next section.

5.2. Interpretation

The first finding, the fact that Epsilon-adjustments that make it harder for the model to make accurate predictions in training increases performance of the trained model seems reasonable and is supported by adversarial training literature like [25]. That the reverse is true, that Delta-adjustments making prediction easier in training will decrease performance in testing seems like an obvious extension of this. The second finding of adjustment tension, that those detrimental Delta-adjustments improve performance when alternated with Epsilon-adjustments seems much more surprising. While we do not know the exact mechanism, we suspect that the improvement has to do with how the optimization process traverses the loss landscape, a definite explanation will only be available trough future work though.

The findings from Chapter 4 are mixed as some limitations became apparent. The positive finding from this chapter is that performance of ABA for small datasets seems high. Deep learning on small datasets suffers most from overfitting where the model memorizes the dataset. We think that ABA is especially well suited to counteract this, as the model itself gives information of how memorization instead of learning underlying patterns might lead to misclassification with small changes of the input. This memorization becomes a smaller problem the bigger the dataset. We suspect that the reason hand-crafted data augmentation outperforms ABA by so much in the much bigger Tiny ImageNet dataset is that it not only counteract memorization but also expands the data distribution. The hand-crafted augmentation "cropping" for example might expand the data distribution in a way the model can learn scale invariance and generalize better. It is a type of augmentation that ABA can not replicate. Some support for this interpretation can be found in [76], where cropping outperforms all other hand-crafted augmentations, that only permute input features of images or color-shift them, by a large degree.

5.3. Implications

Our framework can be used to analyze how components of attribution based augmentation interact and support the creation of new methods. We think that ABA might not be able to outperform hand-crafted data augmentation in vision, but be a useful addition in other domains like NLP and settings where hand-crafted data augmentation is not possible and datasets are small.

6. Future Work

As attribution based augmentation is a very recent addition to the collection of regularization techniques, there are a lot of avenues yet unexplored.

- **Attribution generation step:** We were unable to test other attribution methods than vanilla gradient in the scope of this thesis, focusing heavily on the augmentation step. Other attribution methods might be better suited and we hope that Section 3.5 can help with choosing good candidates. Using a method like integrated gradients would be especially interesting because of the "global" nature of the method, while gradient-only methods like Vanilla Gradient are only "local" in the sense that only the immediate surrounding in the loss landscape is considered for the attribution. For attribution methods it would also be really useful to have a human assessment study for evaluating attribution map quality as mentioned in Section 3.5.
- **Attribution selection step:** We think there might be high potential in extending the attribution selection step from input features to hidden features, as many other regularization techniques do.
- **Augmentation step:** We only explored attribution guided dropout very briefly. It may have some potential, as dropout itself is a relatively popular regularization method and adding attribution information could help improve it. For this it would likely be useful to apply it not only on the input features but also on hidden features. Determining exact mechanisms for the positive influence of adjustment tension in adjustment sets would be another interesting avenue.

We also focused exclusively on classification in this thesis. To extend the scope of where attribution based augmentation might be used, it is necessary to evaluate with a broader range of tasks. Testing the methods proposed in Section 3.8 on non-classification tasks might be a sensible next step. An easy addition would be to add the task of regression. In general, evaluating ABA on different datasets, models and in different domains could deepen understanding of ABA and help in constructing more performant methods. Analyzing the impact of ABA methods on adversarial defense is also a promising avenue, as we identified a parallel between some ABA adjustments and FGSM in adversarial training in Section 3.7.

7. Conclusion

Attribution based augmentation is a recent regularization technique with no common terminology or framework and little research on the mechanisms behind it as of yet. Our primary contribution to this field is a framework, pipeline and terminology for attribution based augmentation and the adjustment tension theory for adjustment sets. Our secondary contribution are the new attribution based augmentation methods created based on adjustment tension and other findings, MendABA and FastABA, and corresponding benchmarks on the datasets MNIST, CIFAR10, 20Newsgroups, Tiny ImageNet and PTB-XL. From these benchmark results it becomes apparent that hand-crafted data augmentation is the superior regularization technique for computer vision, but that ABA methods might well find their place in other domains like NLP, especially for small datasets. As this direction of research is recent, there are many open questions, which we hope Chapter 6 might help navigate.

A. Appendix

We add some supplementary information that might be useful but not necessary for understanding the content of the thesis.

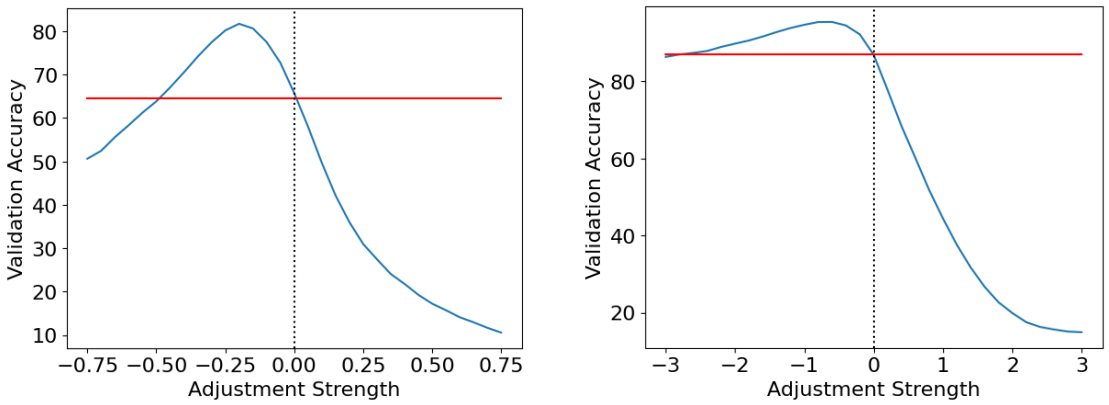
A.1. Additional Data

We show the sanity check for Delta-adjustments and Epsilon-adjustments on a frozen VGG11 on MNIST in Figure A.1. An experiment pointing towards higher adversarial defense for ABA trained models is included in the figure. The adjustments in isolation for MNIST are shown in Table A.1. It includes settings for a more efficient implementation of the attribution selection step. In this implementation we do not calculate the medium adjustment category, but use an offset changing all features and counteracting this change on high and low attribution features. We also show the full result tables here, results for MNIST, 20Newsgroups and CIFAR10 can be seen in Table A.2, full results for Tiny ImageNet can be seen in Table A.3 and full results for PTB-XL can be seen in Table A.4.

A.2. Limitations for Vision Transformer

There are several reasons why the validation accuracy scores for the vision transformer are so low across the different methods.

- One reason is that we could only tune hyperparameters for a few iterations, because we had especially limited resources when we conducted these training runs. Hyperparameters could not just be copied from existing implementations because our vision transformer is different from the paper it is derived from [18] in patch-size and image-size.
- This resource constraint is also the reason why training only lasts 50 epochs, which is before validation accuracy starts to stagnate or decrease.
- Another reason could be that transformers tend to need large datasets to outperform models like CNN with more inductive biases in the architecture [18].



(a) MNIST frozen

(b) MNIST frozen, ABA trained

Figure A.1.: **Delta-adjustments and Epsilon-adjustments on frozen models for MNIST:** Both Delta-adjustments and Epsilon-adjustments are shown in the figures, with negative strength values showing Delta-adjustments and positive strength values Epsilon-adjustments. The red line displays the baseline accuracy with no adjustment. The model trained with ABA is much more resilient to the Epsilon-adjustment that can be interpreted as adversarial examples, pointing toward higher adversarial defense with ABA training.

Low	Middle	High	Type	Offset	Low	High	Index	MAX 5 runs	AVG 5 runs	Rating
0	0	0	Neutral	0	0	0	0	68.39	**66.94**	Baseline
0	1	0	Neutral	1	-1	-1	1	82.09	**79.52**	Top
0	-1	0	Neutral	-1	1	1	2	80.59	**77.24**	Good
0	0	1	Epsilon	0	0	1	3	82.92	**78.04**	Good
0	0	-1	Delta	0	0	-1	4	77.87	**74.84**	Good
1	0	0	Delta	0	1	0	5	73.21	**67.29**	Medium
-1	0	0	Epsilon	0	-1	0	6	85.91	**79.50**	Top
0	1	1	Epsilon	1	-1	0	7	83.96	**81.19**	Top
0	-1	1	Epsilon	-1	1	2	8	83.22	**80.49**	Top
0	1	-1	Delta	1	-1	-2	9	78.65	**72.50**	Good
0	-1	-1	Delta	-1	1	0	10	76.57	**73.48**	Good
1	1	0	Delta	1	0	-1	11	79.42	**70.79**	Medium
1	-1	0	Delta	-1	2	1	12	79.45	**76.84**	Good
-1	1	0	Epsilon	1	-2	-1	13	83.25	**80.18**	Top
-1	-1	0	Epsilon	-1	0	1	14	82.00	**81.08**	Top
1	0	1	Neutral	0	1	1	15	77.31	**74.98**	Good
1	0	-1	Delta	0	1	-1	16	82.64	**77.38**	Good
-1	0	1	Epsilon	0	-1	1	17	85.08	**83.94**	Top
-1	0	-1	Neutral	0	-1	-1	18	81.01	**77.47**	Good
1	1	1	Neutral	1	0	0	19	76.34	**69.56**	Medium
1	-1	1	Neutral	-1	2	2	20	80.90	**77.10**	Good
1	1	-1	Delta	1	0	-2	21	77.31	**74.52**	Good
1	-1	-1	Delta	-1	2	0	22	79.47	**73.26**	Good
-1	1	1	Epsilon	1	-2	0	23	82.95	**80.99**	Top
-1	-1	1	Epsilon	-1	0	2	24	85.10	**82.03**	Top
-1	1	-1	Neutral	1	-2	-2	25	82.71	**78.20**	Good
-1	-1	-1	Neutral	-1	0	0	26	77.40	**70.28**	Medium

Table A.1.: **Comparison of adjustments in training for MNIST:** The 27 adjustments are sorted into equally large adjustment groups of Delta, Epsilon and Neutral adjustments. There is a tendency for worse validation accuracy for Delta adjustments and best validation accuracy for Epsilon adjustments. The tendency is stronger than for CIFAR10 but weaker than for 20Newsgroups. A difference to both other datasets is that most adjustments improve performance. An efficient implementation for adjustments is included in the three columns right of "Type".

Dataset			Vanilla		Challenger			MendABA			FastABA			FastABA-CE		
No BN, val	Subset	Epochs	median	std	median	std	strength	median	std	strength	median	std	strength	median	std	strength
MNIST	Tiny (100)	1000	63.19	9.27	85.00	1.02	2.00	84.75	1.33	1.00	84.54	2.32	1.00	85.04	1.71	1.00
k=4	Med (1000)	100	92.38	0.90	96.14	0.43	1.00	95.97	0.45	1.00	96.44	0.29	1.00	95.26	0.45	1.00
	Full	10	99.19	0.06	99.25	0.08	0.50	99.24	0.08	0.50	99.25	0.07	0.50	99.29	0.07	0.20
CIFAR	Tiny (1000)	500	34.73	1.57	38.76	2.72	0.20	41.22	1.22	0.50	39.60	0.58	1.00	39.91	0.74	0.50
k=4	Full	30	74.93	0.70	75.45	0.73	0.20	75.56	0.65	0.20	75.01	0.55	0.10	74.83	0.47	0.05
NEWS	Tiny (2000)	100	47.20	2.03	58.40	1.77	0.10	60.30	1.44	0.10	59.10	0.53	0.10	58.60	1.12	0.10
k=6	Full	30	67.90	1.88	74.00	0.81	0.05	74.00	1.09	0.05	73.70	1.01	0.05	73.70	1.00	0.02
No BN, test	Subset	Epochs	median	std	median	std	strength	median	std	strength	median	std	strength	median	std	strength
MNIST	Tiny (100)	1000	63.09	8.91	83.12	1.12	2.00	82.84	1.28	1.00	84.65	1.87	1.00	82.23	2.22	1.00
k=4	Med (1000)	100	91.62	1.08	96.03	0.41	1.00	95.88	0.58	1.00	96.20	0.21	1.00	94.94	0.37	1.00
	Full	10	99.07	0.07	99.24	0.06	0.50	99.23	0.06	0.50	99.21	0.06	0.50	99.32	0.08	0.20
CIFAR	Tiny (1000)	500	33.53	0.94	38.39	1.80	0.20	41.01	1.34	0.50	38.51	0.90	1.00	39.69	0.74	0.50
k=4	Full	30	73.83	0.82	74.53	0.62	0.20	74.77	0.50	0.20	74.71	0.73	0.10	74.22	0.36	0.05
NEWS	Tiny (2000)	100	46.23	1.46	57.25	2.02	0.10	60.54	1.40	0.10	58.09	0.86	0.10	57.98	0.97	0.10
k=6	Full	30	66.73	1.29	72.79	1.03	0.05	72.93	1.25	0.05	73.04	1.15	0.05	72.91	1.30	0.02
BN, val	Subset	Epochs	median	std	median	std	strength	median	std	strength	median	std	strength	median	std	strength
MNIST	Tiny (100)	500	84.63	1.52	85.48	1.55	0.10	85.74	0.95	0.10	86.28	0.81	0.50	86.20	0.79	0.50
k=4	Med (1000)	50	96.63	1.24	96.61	0.44	0.05	96.97	0.17	0.05	96.84	0.36	0.10	96.76	0.40	0.10
	Full	10	99.22	0.06	99.28	0.07	0.02	99.29	0.07	0.02	99.35	0.09	0.10	99.27	0.09	0.10
CIFAR	Tiny (1000)	200	46.18	0.89	46.80	0.64	0.01	46.32	0.88	0.01	46.50	0.90	0.01	46.69	0.70	0.01
k=4	Full	30	83.07	0.75	84.31	0.29	0.03	83.67	0.65	0.02	83.92	0.69	0.01	83.83	0.62	0.01
NEWS	Tiny (2000)	50	47.42	1.79	59.40	0.57	0.10	60.20	0.95	0.10	58.60	1.60	0.10	58.30	0.99	0.05
k=6	Full	30	67.00	1.45	75.30	1.09	0.05	76.00	0.38	0.05	75.70	0.61	0.05	75.20	0.69	0.02
BN, test	Subset	Epochs	median	std	median	std	strength	median	std	strength	median	std	strength	median	std	strength
MNIST	Tiny (100)	500	83.25	1.79	83.43	1.56	0.10	83.59	1.23	0.10	85.4	1.44	0.50	84.43	1.13	0.50
k=4	Med (1000)	50	96.72	1.55	96.63	0.38	0.05	96.91	0.19	0.05	96.93	0.41	0.10	96.68	0.34	0.10
	Full	10	99.22	0.05	99.30	0.08	0.02	99.33	0.07	0.02	99.31	0.12	0.10	99.29	0.07	0.10
CIFAR	Tiny (1000)	200	45.55	0.95	45.93	0.43	0.01	45.47	0.69	0.01	45.40	0.89	0.01	45.79	0.74	0.01
k=4	Full	30	82.49	0.66	83.73	0.49	0.03	83.35	0.38	0.02	83.46	0.50	0.01	83.32	0.41	0.01
NEWS	Tiny (2000)	50	46.89	2.60	57.86	0.89	0.10	58.97	0.90	0.10	57.45	1.20	0.10	57.53	1.55	0.05
k=6	Full	30	65.75	1.35	74.24	1.36	0.05	74.81	0.43	0.05	74.43	0.76	0.05	73.95	0.67	0.02

Table A.2.: **Full results for the benchmarking section for MNIST, CIFAR10 and 20Newsgroups:** Ideal adjustment strengths are given for the different settings. One additional column is added for FastABA trained with loss as the attribution target (cross-entropy) instead of class scores.

			Vanilla		Challenger			MendABA			FastABA		
Tiny ImageNet	RandAug	Epochs	median	std	median	std	strength	median	std	strength	median	std	strength
ResNet50	No	100	31.31	0.81	32.74	0.28	0.100	32.86	0.90	0.100	33.13	0.49	0.030
ResNet50	Yes	100	46.93	1.44	45.75	0.80	0.100	47.98	2.81	0.100	46.71	1.11	0.030
EfficientB3	No	300	41.33	0.60	40.94	0.92	0.030	42.11	0.83	0.030	42.18	0.64	0.030
EfficientB3	Yes	300	53.59	0.79	53.29	0.55	0.003	53.66	0.61	0.003	53.63	0.47	0.003
ViT Base 16	No	50	28.20	0.72	29.95	0.66	0.100	31.77	0.74	0.100	33.01	0.42	0.100

Table A.3.: **Full results for the benchmarking section for Tiny ImageNet:** Ideal adjustment strengths are given for the different settings.

	Epochs	median	std	median	std	strength	median	std	strength	median	std	strength
PTB-XL	50	91.40	0.26	91.50	0.36	0.100	91.50	0.40	0.100	91.40	0.28	0.030

Table A.4.: **Full results for the benchmarking section for PTB-XL:** Ideal adjustment strengths are given for the different settings.

List of Figures

List of Tables

Bibliography

[1] K. Aas, M. Jullum, and A. Løland. "Explaining individual predictions when features are dependent: More accurate approximations to Shapley values." In: *Artificial Intelligence* 298 (2021), p. 103502.

[2] M. Ancona, E. Ceolini, C. Öztireli, and M. Gross. "Towards better understanding of gradient-based attribution methods for deep neural networks." In: *arXiv preprint arXiv:1711.06104* (2017).

[3] I. Asimov. "Three laws of robotics." In: *Asimov, I. Runaround* (1941).

[4] B. O. Ayinde, T. Inanc, and J. M. Zurada. "Regularizing deep neural networks by enhancing diversity in feature extraction." In: *IEEE transactions on neural networks and learning systems* 30.9 (2019), pp. 2650–2661.

[5] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek. "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation." In: *PloS one* 10.7 (2015), e0130140.

[6] T. Bai, J. Luo, J. Zhao, B. Wen, and Q. Wang. "Recent advances in adversarial training for adversarial robustness." In: *arXiv preprint arXiv:2102.01356* (2021).

[7] C. M. Bishop et al. *Neural networks for pattern recognition*. Oxford university press, 1995.

[8] G. Brasó and L. Leal-Taixé. "Learning a neural solver for multiple object tracking." In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 6247–6257.

[9] J. Bridle. "Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters." In: *Advances in neural information processing systems* 2 (1989).

[10] B. G. Buchanan and R. G. Smith. "Fundamentals of expert systems." In: *Annual review of computer science* 3.1 (1988), pp. 23–58.

[11] M. Campbell, A. J. Hoane Jr, and F.-h. Hsu. "Deep blue." In: *Artificial intelligence* 134.1-2 (2002), pp. 57–83.

[12] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le. "Randaugment: Practical automated data augmentation with a reduced search space." In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*. 2020, pp. 702–703.

[13] G. Cybenko. "Approximation by superpositions of a sigmoidal function." In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. "Imagenet: A large-scale hierarchical image database." In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[15] L. Deng. "The mnist database of handwritten digit images for machine learning research." In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[16] T. DeVries and G. W. Taylor. "Improved regularization of convolutional neural networks with cutout." In: *arXiv preprint arXiv:1708.04552* (2017).

[17] T. G. Dietterich. "Ensemble methods in machine learning." In: *International workshop on multiple classifier systems*. Springer. 2000, pp. 1–15.

[18] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." In: *arXiv preprint arXiv:2010.11929* (2020).

[19] D. Dua and C. Graff. *UCI Machine Learning Repository*. 2017.

[20] D. Durstewitz, G. Koppe, and A. Meyer-Lindenberg. "Deep neural networks in psychiatry." In: *Molecular psychiatry* 24.11 (2019), pp. 1583–1598.

[21] T. Ersson and X. Hu. "Path planning and navigation of mobile robots in unknown environments." In: *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180)*. Vol. 2. IEEE. 2001, pp. 858–864.

[22] J. C. B. Gamboa. "Deep learning for time-series analysis." In: *arXiv preprint arXiv:1701.01887* (2017).

[23] C. Gong, D. Wang, M. Li, V. Chandra, and Q. Liu. "Keepaugment: A simple information-preserving data augmentation approach." In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 1055–1064.

[24] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. "Generative adversarial networks." In: *Communications of the ACM* 63.11 (2020), pp. 139–144.

[25] I. J. Goodfellow, J. Shlens, and C. Szegedy. "Explaining and harnessing adversarial examples." In: *arXiv preprint arXiv:1412.6572* (2014).

[26] A. Graves, A.-r. Mohamed, and G. Hinton. "Speech recognition with deep recurrent neural networks." In: *2013 IEEE international conference on acoustics, speech and signal processing*. Ieee. 2013, pp. 6645–6649.

[27] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. "On calibration of modern neural networks." In: *International conference on machine learning*. PMLR. 2017, pp. 1321–1330.

[28] M. Haenlein and A. Kaplan. "A brief history of artificial intelligence: On the past, present, and future of artificial intelligence." In: *California management review* 61.4 (2019), pp. 5–14.

[29] K. Hara, D. Saitoh, and H. Shouno. "Analysis of dropout learning regarded as ensemble learning." In: *International Conference on Artificial Neural Networks*. Springer. 2016, pp. 72–79.

[30] D. M. Hawkins. "The problem of overfitting." In: *Journal of chemical information and computer sciences* 44.1 (2004), pp. 1–12.

[31] K. He, X. Zhang, S. Ren, and J. Sun. "Deep residual learning for image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[32] J. M. Helm, A. M. Swiergosz, H. S. Haeberle, J. M. Karnuta, J. L. Schaffer, V. E. Krebs, A. I. Spitzer, and P. N. Ramkumar. "Machine learning and artificial intelligence: definitions, applications, and future directions." In: *Current reviews in musculoskeletal medicine* 13.1 (2020), pp. 69–76.

[33] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. "Improving neural networks by preventing co-adaptation of feature detectors." In: *arXiv preprint arXiv:1207.0580* (2012).

[34] K. Hornik, M. Stinchcombe, and H. White. "Multilayer feedforward networks are universal approximators." In: *Neural networks* 2.5 (1989), pp. 359–366.

[35] S. Ioffe and C. Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." In: *International conference on machine learning*. PMLR. 2015, pp. 448–456.

[36] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, et al. "Highly accurate protein structure prediction with AlphaFold." In: *Nature* 596.7873 (2021), pp. 583–589.

[37] A. Kaplan and M. Haenlein. "Siri, Siri, in my hand: Who's the fairest in the land? On the interpretations, illustrations, and implications of artificial intelligence." In: *Business Horizons* 62.1 (2019), pp. 15–25.

[38] H. Kautz. "The third AI summer: AAAI Robert S. Engelmore Memorial Lecture." In: *AI Magazine* 43.1 (2022), pp. 93–104.

[39] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah. "Transformers in vision: A survey." In: *ACM computing surveys (CSUR)* 54.10s (2022), pp. 1–41.

[40] D. P. Kingma and J. Ba. "Adam: A method for stochastic optimization." In: *arXiv preprint arXiv:1412.6980* (2014).

[41] W. M. Kouw and M. Loog. "An introduction to domain adaptation and transfer learning." In: *arXiv preprint arXiv:1812.11806* (2018).

[42] A. Krizhevsky, G. Hinton, et al. "Learning multiple layers of features from tiny images." In: (2009).

[43] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "Imagenet classification with deep convolutional neural networks." In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[44] J. Kukačka, V. Golkov, and D. Cremers. "Regularization for deep learning: A taxonomy." In: *arXiv preprint arXiv:1710.10686* (2017).

[45] Y. Le and X. Yang. "Tiny imagenet visual recognition challenge." In: *CS 231N* 7.7 (2015), p. 3.

[46] Y. LeCun, Y. Bengio, and G. Hinton. "Deep learning." In: *nature* 521.7553 (2015), pp. 436–444.

[47] B. Li, Y. Hou, and W. Che. "Data augmentation approaches in natural language processing: A survey." In: *AI Open* (2022).

[48] B. Lim and S. Zohren. "Time-series forecasting with deep learning: a survey." In: *Philosophical Transactions of the Royal Society A* 379.2194 (2021), p. 20200209.

[49] I. Loshchilov and F. Hutter. "Decoupled weight decay regularization." In: *arXiv preprint arXiv:1711.05101* (2017).

[50] D. G. Lowe. "Object recognition from local scale-invariant features." In: *Proceedings of the seventh IEEE international conference on computer vision*. Vol. 2. Ieee. 1999, pp. 1150–1157.

[51] S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy, A. Blanco, C. Clement, D. Drain, D. Jiang, D. Tang, et al. "Codexglue: A machine learning benchmark dataset for code understanding and generation." In: *arXiv preprint arXiv:2102.04664* (2021).

[52] S. M. Lundberg and S.-I. Lee. "A unified approach to interpreting model predictions." In: *Advances in neural information processing systems* 30 (2017).

[53] J. McCarthy. "What is artificial intelligence?" In: (2007).

[54] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. "Human-level control through deep reinforcement learning." In: *nature* 518.7540 (2015), pp. 529–533.

[55] C. Molnar. *Interpretable machine learning*. Lulu. com, 2020.

[56] G. Montavon, A. Binder, S. Lapuschkin, W. Samek, and K.-R. Müller. "Layer-wise relevance propagation: an overview." In: *Explainable AI: interpreting, explaining and visualizing deep learning* (2019), pp. 193–209.

[57] R. A. Mouha et al. "Deep learning for robotics." In: *Journal of Data Analysis and Information Processing* 9.02 (2021), p. 63.

[58] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic. "Deep learning applications and challenges in big data analytics." In: *Journal of big data* 2.1 (2015), pp. 1–21.

[59] G. Nguyen, D. Kim, and A. Nguyen. "The effectiveness of feature attribution methods and its correlation with automatic evaluation scores." In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 26422–26436.

[60] N. J. Nilsson. *The quest for artificial intelligence*. Cambridge University Press, 2009.

[61] J. Nixon, M. W. Dusenberry, L. Zhang, G. Jerfel, and D. Tran. "Measuring Calibration in Deep Learning." In: *CVPR Workshops*. Vol. 2. 7. 2019.

[62] K. O'Shea and R. Nash. "An introduction to convolutional neural networks." In: *arXiv preprint arXiv:1511.08458* (2015).

[63] D. W. Otter, J. R. Medina, and J. K. Kalita. "A survey of the usages of deep learning for natural language processing." In: *IEEE transactions on neural networks and learning systems* 32.2 (2020), pp. 604–624.

[64] A. Paullada, I. D. Raji, E. M. Bender, E. Denton, and A. Hanna. "Data and its (dis) contents: A survey of dataset development and use in machine learning research." In: *Patterns* 2.11 (2021), p. 100336.

[65] J. Pennington, R. Socher, and C. D. Manning. "Glove: Global vectors for word representation." In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[66] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al. "Improving language understanding by generative pre-training." In: (2018).

[67] V. Raina, S. Krishnamurthy, V. Raina, and S. Krishnamurthy. "Natural language processing." In: *Building an Effective Data Science Practice: A Framework to Bootstrap and Manage a Successful Data Science Practice* (2022), pp. 63–73.

[68] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever. "Zero-shot text-to-image generation." In: *International Conference on Machine Learning*. PMLR. 2021, pp. 8821–8831.

[69] M. T. Ribeiro, S. Singh, and C. Guestrin. ""Why should i trust you?" Explaining the predictions of any classifier." In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 1135–1144.

[70] H. Robbins and S. Monro. "A stochastic approximation method." In: *The annals of mathematical statistics* (1951), pp. 400–407.

[71] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. "High-resolution image synthesis with latent diffusion models." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 10684–10695.

[72] M. H. Sazli. "A brief review of feed-forward neural networks." In: *Communications Faculty of Sciences University of Ankara Series A2-A3 Physical Sciences and Engineering* 50.01 (2006).

[73] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. "The graph neural network model." In: *IEEE transactions on neural networks* 20.1 (2008), pp. 61–80.

[74] R. M. Schmidt. "Recurrent neural networks (rnns): A gentle introduction and overview." In: *arXiv preprint arXiv:1912.05911* (2019).

[75] 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." In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 618–626.

[76] C. Shorten and T. M. Khoshgoftaar. "A survey on image data augmentation for deep learning." In: *Journal of big data* 6.1 (2019), pp. 1–48.

[77] A. Shrestha and A. Mahmood. "Review of deep learning algorithms and architectures." In: *IEEE access* 7 (2019), pp. 53040–53065.

[78] A. Shrikumar, P. Greenside, and A. Kundaje. "Learning important features through propagating activation differences." In: *International conference on machine learning*. PMLR. 2017, pp. 3145–3153.

[79] A. Shrikumar, P. Greenside, A. Shcherbina, and A. Kundaje. "Not just a black box: Learning important features through propagating activation differences." In: *arXiv preprint arXiv:1605.01713* (2016).

[80] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. "Mastering the game of Go with deep neural networks and tree search." In: *nature* 529.7587 (2016), pp. 484–489.

[81] K. Simonyan, A. Vedaldi, and A. Zisserman. "Deep inside convolutional networks: Visualising image classification models and saliency maps." In: *arXiv preprint arXiv:1312.6034* (2013).

[82] K. Simonyan and A. Zisserman. "Very deep convolutional networks for large-scale image recognition." In: *arXiv preprint arXiv:1409.1556* (2014).

[83] D. Smilkov, N. Thorat, B. Kim, F. Viégas, and M. Wattenberg. "Smoothgrad: removing noise by adding noise." In: *arXiv preprint arXiv:1706.03825* (2017).

[84] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. "Striving for simplicity: The all convolutional net." In: *arXiv preprint arXiv:1412.6806* (2014).

[85] N. Strodthoff, P. Wagner, T. Schaeffter, and W. Samek. "Deep learning for ECG analysis: Benchmarks and insights from PTB-XL." In: *IEEE Journal of Biomedical and Health Informatics* 25.5 (2020), pp. 1519–1528.

[86] P. Sturmfels, S. Lundberg, and S.-I. Lee. "Visualizing the impact of feature attribution baselines." In: *Distill* 5.1 (2020), e22.

[87] S. Sun, Z. Cao, H. Zhu, and J. Zhao. "A survey of optimization methods from a machine learning perspective." In: *IEEE transactions on cybernetics* 50.8 (2019), pp. 3668–3681.

[88] M. Sundararajan, A. Taly, and Q. Yan. "Axiomatic attribution for deep networks." In: *International conference on machine learning*. PMLR. 2017, pp. 3319–3328.

[89] N. Sünderhauf, O. Brock, W. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford, et al. "The limits and potentials of deep learning for robotics." In: *The International journal of robotics research* 37.4-5 (2018), pp. 405–420.

[90] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. "Rethinking the inception architecture for computer vision." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[91] M. Tan and Q. Le. "Efficientnet: Rethinking model scaling for convolutional neural networks." In: *International conference on machine learning*. PMLR. 2019, pp. 6105–6114.

[92] S. Tellex, N. Gopalan, H. Kress-Gazit, and C. Matuszek. "Robots that use language." In: *Annual Review of Control, Robotics, and Autonomous Systems* 3 (2020), pp. 25–55.

[93] C. Tomani and D. Cremers. "CHALLENGER: Training with Attribution Maps." In: *arXiv preprint arXiv:2205.15094* (2022).

[94] L. Torrey and J. Shavlik. "Transfer learning." In: *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. IGI global, 2010, pp. 242–264.

[95] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. "Attention is all you need." In: *Advances in neural information processing systems* 30 (2017).

[96] A. Voulodimos, N. Doulamis, A. Doulamis, E. Protopapadakis, et al. "Deep learning for computer vision: A brief review." In: *Computational intelligence and neuroscience* 2018 (2018).

[97] P. Wagner, N. Strodthoff, R.-D. Bousseljot, D. Kreiseler, F. I. Lunze, W. Samek, and T. Schaeffter. "PTB-XL, a large publicly available electrocardiography dataset." In: *Scientific data* 7.1 (2020), p. 154.

[98] C. Wang and J. C. Principe. "Training neural networks with additive noise in the desired signal." In: *IEEE Transactions on Neural Networks* 10.6 (1999), pp. 1511–1517.

[99] Q. Wang, Y. Ma, K. Zhao, and Y. Tian. "A comprehensive survey of loss functions in machine learning." In: *Annals of Data Science* (2020), pp. 1–26.

[100] Z. Wang, W. Yan, and T. Oates. "Time series classification from scratch with deep neural networks: A strong baseline." In: *2017 International joint conference on neural networks (IJCNN)*. IEEE. 2017, pp. 1578–1585.

[101] Q. Wen, T. Zhou, C. Zhang, W. Chen, Z. Ma, J. Yan, and L. Sun. "Transformers in time series: A survey." In: *arXiv preprint arXiv:2202.07125* (2022).

[102] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, et al. "Transformers: State-of-the-art natural language processing." In: *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*. 2020, pp. 38–45.

[103] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. "A comprehensive survey on graph neural networks." In: *IEEE transactions on neural networks and learning systems* 32.1 (2020), pp. 4–24.

[104] F. F. Xu, B. Vasilescu, and G. Neubig. "In-ide code generation from natural language: Promise and challenges." In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31.2 (2022), pp. 1–47.

[105] Y. Xu and R. Goodacre. "On splitting training and validation set: a comparative study of cross-validation, bootstrap and systematic sampling for estimating the generalization performance of supervised learning." In: *Journal of analysis and testing* 2.3 (2018), pp. 249–262.

[106] X. Ying. "An overview of overfitting and its solutions." In: *Journal of physics: Conference series*. Vol. 1168. 2. IOP Publishing. 2019, p. 022022.

[107] M. D. Zeiler and R. Fergus. "Visualizing and understanding convolutional networks." In: *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*. Springer. 2014, pp. 818–833.

[108] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. "Learning deep features for discriminative localization." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2921–2929.