INSTITUT FÜR INFORMATIK
Datenbanken und Informationssysteme

Universitätsstr. 1 D–40225 Düsseldorf

Heinrich Heine Universität Düsseldorf

Evaluation of XAI-Algorithms Using Neural Networks

Julian Robert Ullrich

Masterarbeit

Date of issue:	19. Februar 2024
Date of submission:	19. August 2024
Reviewers:	Prof. Dr. Timo Dickscheid
	Dr.-Ing. Carsten Knoll

Erklärung

Hiermit versichere ich, dass ich diese Masterarbeit selbstständig verfasst habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Düsseldorf, den 19. August 2024

Julian Robert Ullrich

Abstract

The field of Explainable Artificial Intelligence (XAI) has the goal to aid humans to gain a deeper understanding into the decision-making process of Artificial Intelligence (AI) methods. In recent years many XAI methods have been proposed, but choosing the best one for the task at hand and the used AI model is not trivial.

This thesis explores ways to benchmark XAI methods in the context of image classification by occluding parts of an images deemed relevant or irrelevant by the XAI method while observing the impact the occlusion has on the accuracy of the AI model. To achieve this a task-specific dataset is introduced with the goal to establish a fair way to compare XAI methods with one another.

Additionally this thesis presents a novel XAI approach with the goal of being highly interpretable by humans. The method builds upon the established Integrated Gradients (Sundararajan et al., 2017) method, which by itself however is poorly interpretable. By combining the strengths of Integrated Gradients (IG) with concepts from the field of eye-tracking, "IG-Fixpoints" is introduced as a novel method which produces easily interpretable heatmaps.

Using the proposed benchmarking method, the novel method is evaluated alongside multiple popular existing XAI methods on a selection of convolutional neural networks.

Contents

1 Introduction and Motivation **1**

2 Fundamentals **2**

2.1 Convolutional Neural Networks 2

2.2 Interpretable and Explainable AI 12

2.3 Comparison and Evaluation of XAI Methods 14

3 Methods **15**

3.1 Dataset Selection and Creation 15

3.2 Convolutional Neural Networks 19

4 XAI Methods **29**

4.1 Grad-CAM 29

4.2 PRISM 31

4.3 LIME 31

4.4 Integrated Gradients 32

4.5 XRAI 33

4.6 IG-Fixpoints 35

4.7 Heatmap Processing 37

5 Evaluation of XAI-Methods using Neural Networks **39**

5.1 Occlusion Accuracy 39

5.2 Revealing Accuracy 40

5.3 Statistical Evaluation 40

6 Results **43**

6.1 Network Training Results 43

6.2 Occlusion and Revelation Accuracy 44

6.3 Statistical Evaluation of XAI Methods 53

6.4 Parameter Optimization for IGF 56

6.5 Grad-CAM on Changed ConvNeXt 59

7 Conclusion and Outlook **61**

A Shapes of Signs and Class Descriptions for the ATSDS **62**

B Python Code for the Simple CNN using the pytorch framework **63**

C Training Details **63**

D Implementation Details **64**

E Detailed Occlusion Accuracy Results **65**

F Detailed TPR Results **67**

References **69**

List of Figures **72**

List of Tables **75**

1 Introduction and Motivation

The field of Machine Learning (ML) and especially Artificial Intelligence (AI) is rapidly advancing, and more and more impressive feats are achieved.

Recent studies have for example shown, that AI might be capable of outperforming human medical experts, for example when it comes to cancer diagnosis (Salinas et al., 2024).

Nowadays AI is starting to become more and more important in our everyday lives as well. Computer Vision for example can be used for self driving cars.

The use of AI in the transportation sector and in many other areas could potentially increase safety, reduce workload or even save lives. At the same time bad decisions made by an AI could have the averse effect. Therefore it is important to have a good understanding for the reasoning behind the decision-making of an AI first. This process can also help to showcase flaws, and thus improve AI further. Establishing a good understanding of these methods is an important foundation to allow humans working with or being influenced by AI methods to consider them reliable and trustworthy.

Especially when it comes to Computer Vision (CV) tasks like image classification, the best results are often achieved by complex neural network architectures. Such system, often referred to as "black box" models, are not inherently interpretable by humans. The lack of intuitive understanding of these AI methods, has given rise to the scientific field of explainable and interpretable AI (XAI). The part of XAI relevant for this thesis aims to offer interpretable explanations for decisions made by complex neural networks, in form of an importance heatmap, highlighting which areas of an image played a big role in the decision-making of the network (Hassija et al., 2023).

While many XAI methods can offer us explanations, which from a human standpoint are convincing, it is hard to properly measure the quality of XAI algorithms and compare them against one another.

The goal throughout this thesis is to explore possible methods to compare XAI algorithms for Convolutional Neural Networks (CNN).

Additionally this thesis proposes a novel XAI method, with the specific goal of offering a highly interpretable heatmap to the user.

The thesis is organized as follows. In Section 2 concepts and terminology for AI and XAI relevant to the work done in the thesis are discussed. Section 3 introduces the task specific dataset and discusses the used CNNs. Section 4 covers the XAI methods to be evaluated against one another, including the newly introduced "IG-Fixpoints" method. In Section 5 the evaluation methods are presented. The results are discussed in Section 6

2 Fundamentals

2.1 Convolutional Neural Networks

2.1.1 Artificial Neuron

Artificial neurons (Figure 1) are the foundation of Artificial Neural Networks (ANN). The idea of an artificial neuron is to combine multiple inputs into an output feature through weighting of the inputs features and the use of an activation function. The weighted inputs are summed up together with a bias term before the activation function is applied to receive the output. The weights and the bias term are so called trainable parameters and are adjusted during the training process to fit the data (Sarker, 2021).

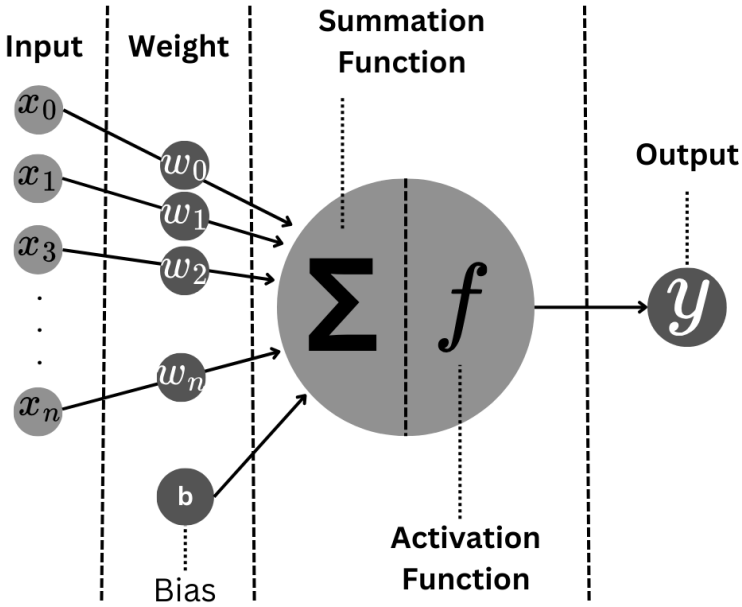

Figure 1: Artificial Neuron.

2.1.2 Artificial Neural Network

By using multiple Artificial Neurons with the same inputs in parallel, a so called inner or hidden layer is created. An Artificial Neural Network (ANN) consist of an input layer, an output layer and one or more hidden layers. When talking about neural networks, the depth of a network is defined by the amount of these hidden layers, while the width of the network is defined by the amount of parallel neurons per layer. Deep learning (DL) thus refers to the training of networks with a high depth. Networks or part of networks consisting of layers built with simple artificial neurons as depicted in Figure 2 are referred to as "Fully-Connected", "Feedforward Neural Networks" or "Multilayer Perceptrons" (MLP).

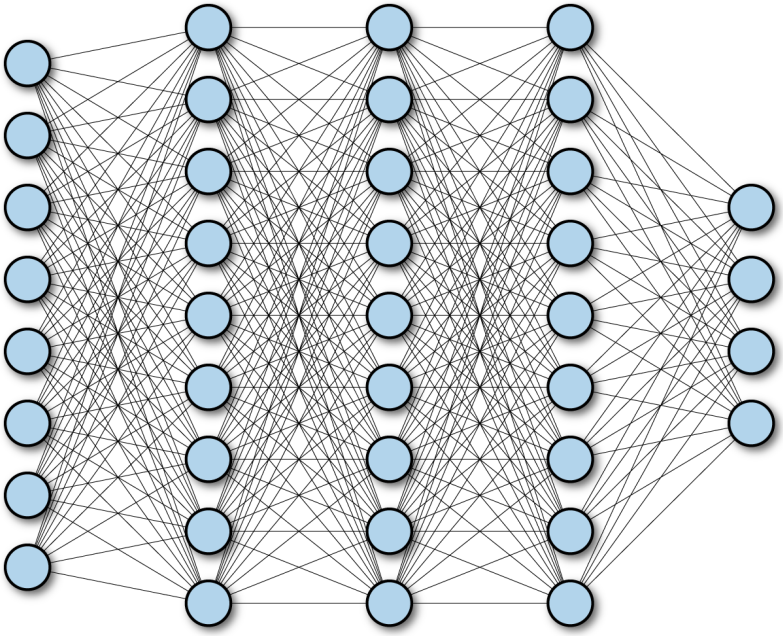

Figure 2: A fully connected ANN/FNN mapping eight input points (left) to four outputs (right) using three "hidden" layers in between as illustrated by Ramsundar and Zadeh (2018)

2.1.3 Activation Functions

Over the years, a multitude of activation functions have been proposed with different goals in mind. As a result activation functions can have different properties. The right choice for the activation function is important for the learning capability and the stability of the training process while balancing computational demand. Activation functions are essential to add non-linearity to the network, which enables it to learn more complex hierarchical representations and connect the input and output in a more sophisticated way compared to a simple linear transformation (Lee, 2023). Three methods relevant to this thesis will be covered below and an illustration for their behavior is given in Figure 3

Sigmoid

The (Logistic) Sigmoid activation function was very popular in the early days of DL and should be introduced as a reference point for other functions, even if it is not relevant for the network architectures used in this thesis. The Sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

has some desirable properties, like limiting the magnitude of the output space to the interval of $[0, 1]$ and thus preventing negative values while reducing the impact a single

neuron can have on the next layer (Dubey et al., 2021).

ReLU

The Sigmoid function comes with high computational demand. Additionally, as networks started to become deeper, reducing the magnitude of the output started to disrupted a smooth gradient flow through all layers, turning the once desirable property into an unwanted one. The approach taken by the Rectified Linear Unit (ReLU)

$$\mathrm{ReLU}(x) = \begin{cases} x & \text{if } x > 0, \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

is to prevent negative outputs by setting them to zero, while leaving the positive values unchanged. Intuitively one can consider this as ignoring unwanted or unimportant features, while allowing for relevant features to retain a higher impact. Thanks to its simplicity, low computational demand and good performance when deployed in networks, it has been one of the most popular activation functions in the last decade.

GELU

The ReLU has some shortcomings. Most notably it is not differentiable at zero, and by setting all negative inputs to zero, the risk of so called dead neurons arises, where neurons or part of a network might get stuck on outputting zeroes. The GELU addresses these issues by being smooth and differentiable for every given x and allowing for small negative values which only have their magnitude reduced, while behaving similar to the ReLU for positive values.

GELU has been widely used in transformer architectures and has found its way into recent CNN architectures like the ConvNeXt (Section 3.2.4) as well.

It is given by

$$\mathrm{GELU}(x) = x \cdot \Phi(x) \tag{3}$$

where $\Phi(x)$ is the standard Gaussian cumulative distribution function and is usually approximated by

$$\mathrm{GELU}(x) \approx \sigma(1.702x) \cdot x \tag{4}$$

where σ refers to the sigmoid function.

2.1.4 Convolutions

For Computer Vision tasks like image classification, typically CNNs are used instead of ANNs. CNNs utilize convolutional filters, which preserve the spatial structure of the images and are thus capable of capturing local patterns and features.

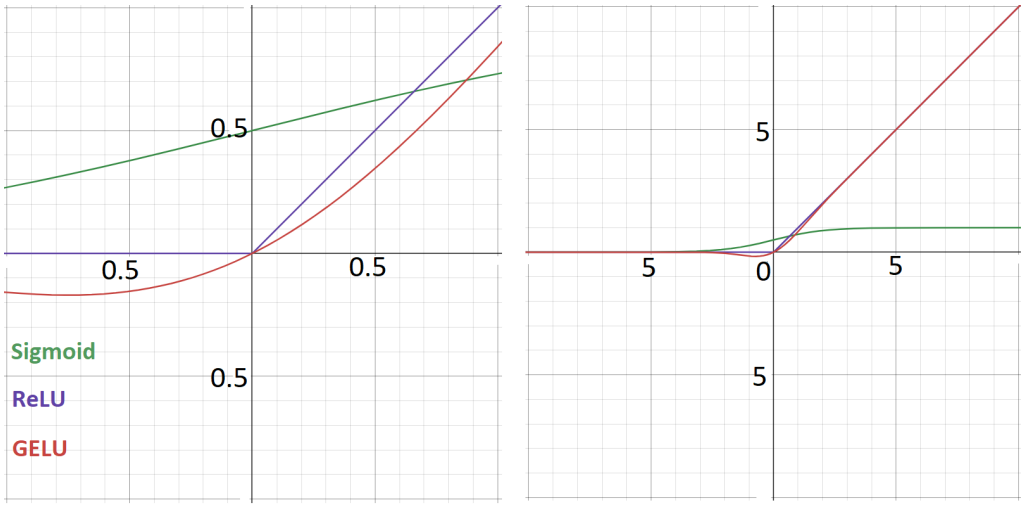

Figure 3: Behavior of the Sigmoid (green), ReLU (purple) and GELU (red) function close to zero (L) and when zoomed out (R).

A convolutional filter or kernel is a matrix consisting of weights and a bias term. The output of such a filter is obtained by sliding the kernel over the input data and summing up the element-wise products of the overlapping area. The trainable parameters (weights and bias term) are part of the kernel and shared for each application on the input. Sometimes it is desirable to not use every possible configuration, to for example decrease the spatial dimension of the output. The step size between the applications of the kernel is called the stride. The process is illustrated in Figure 4. The width of a convolutional layer is defined by the amount of convolutional filters/kernels applied on the input, which decides how many channels the output has. A high amount of kernels and thus channels allows the network to differentiate between more features and increases the capacity of the network but also the computational demand for training the network.

Given an input with dimensions $W_{\text{in}} \times H_{\text{in}}$, a kernel of size K and a stride S, the output size $W_{\text{out}} \times H_{\text{out}}$ is given by $W_{\text{out}} = \left\lfloor \frac{W_{\text{in}}-K}{S} \right\rfloor + 1$ and $H_{\text{out}} = \left\lfloor \frac{H_{\text{in}}-K}{S} \right\rfloor + 1$. Thus using a kernel size or stride greater than one results in a reduced spatial dimension for the output. This is well observable in the example in Figure 5 where a 3x3 convolution with stride 1 reduces the spatial dimension of the output by 2.

Such behavior is often unwanted and typically so called padding is used to counteract it. By surrounding the input data with an appropriate amount of new values before applying the kernel, the input dimension is increased which allows the output to retain the size of the unpadded input. The most common version of this is the zero-padding as depicted in Figure 5.

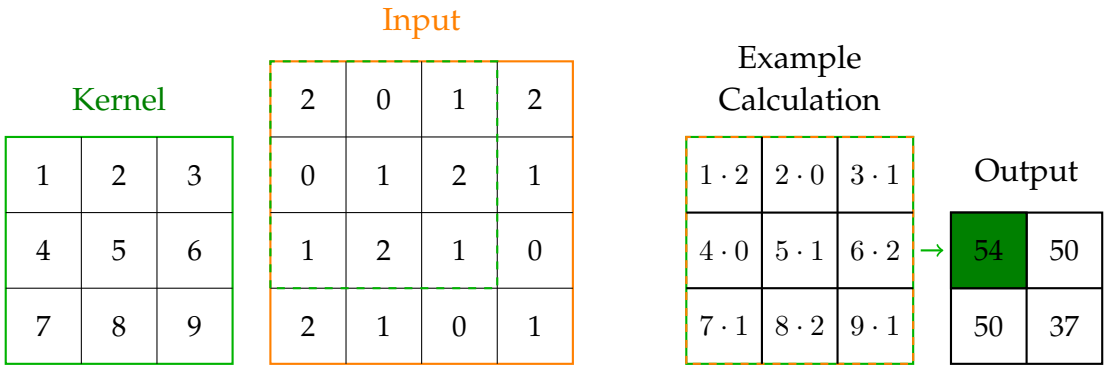

Figure 4: Example for the application of a 3x3 convolutional filter on a 4x4 input without padding. The example calculation is based on the area marked with the green dashed lines. Notably the spatial dimension of the output decreased to 2x2.

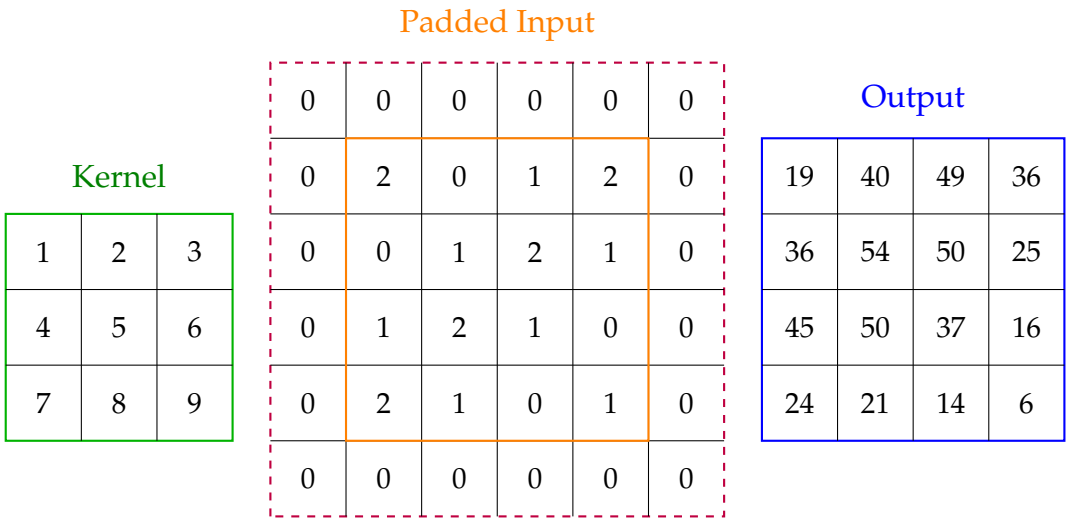

Figure 5: Usage of zero padding to prevent the reduction in the spatial dimension. The required padding depends on the size and stride of the kernel.

2.1.5 Pooling Layers

To be able to perform classification tasks on complex images, a large amount of channels is desirable, since they can be viewed as the amount of features the network can differentiate. As we increase the amount of channels, computational demand increases accordingly. To be able to train networks properly while increasing the number of channels and thus differentiable features, the spatial dimension of the feature space has to be reduced.

The typical way to achieve this is the use of so called Average Pooling or Max Pooling. Average pooling can be considered as a convolutional filter with fixed and equal weights which thus calculates the average for the overlapped area, while max pooling takes only the highest value into account instead. The process is illustrated in Figure 6.

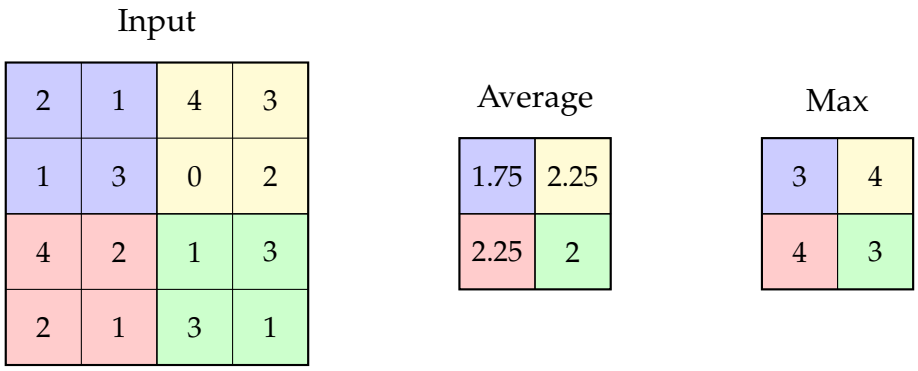

Figure 6: Illustration of pooling methods using a 2x2 kernel size and a stride of two. Average pooling takes the average of the four elements it overlaps with, while max pooling extracts the maximum. Notably the spatial dimension is reduced by half when using the typical kernel size of 2x2 and a stride of two.

A special form of pooling is the so called "global average pooling" (GAP) which is often used to reduce a feature map to a single value by matching the kernel size to the size of the feature map. Doing this prevents the downstream network from accessing individual pixels and enforces the idea of the feature maps prior to the GAP containing a single feature. This can help the network to learn meaningful and distinct features during training and is used in most of the CNNs in this thesis.

2.1.6 Depthwise Convolutions

When applying a convolutional filter to an input, for each desired output channel, a kernel is applied over all input channels at once and thus its depth matches the dimension of the input channels. A different approach is to reduce the depth of a kernel, and apply seperate lower dimensional kernels to selected channels. A so called depthwise convolution (Guo et al., 2019) is a convolutional layer, where the depth of a kernel is one. This means each input channel has its separate kernel.

As a result an output can only reflect the content of a single input channel. To restore

the so called channel mixing to the network, a 1x1 convolution is applied directly afterwards. Due to the kernel size, this is referred to as pointwise convolution. This process is depicted in Figure 7. Notationwise a "d" in front of the kernel size indicates the use of a depthwise convolution in this thesis.

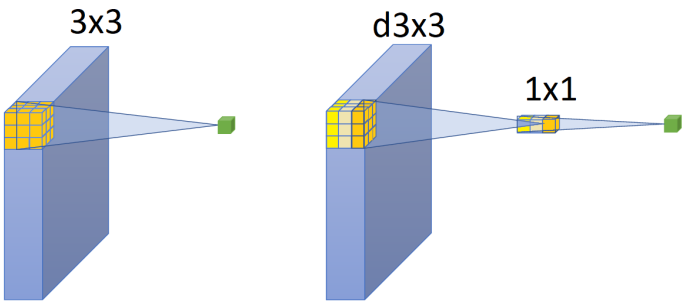

Figure 7: Illustration of a 3x3 kernel application on a three channel input using a standard and a depthwise convolution followed by a pointwise convolution (Guo et al., 2019). The standard convolution (L) is applied on all three channels at once, while depthwise convolution operates on a per channel basis instead and achieves the channel mixing through the pointwise convolution afterwards (R).

Compared to a standard convolution, a depthwise convolution followed by a pointwise convolution requires significantly less computations, but reduces the capacity of the network as well. However reducing the computational demand in the convolutions allows for higher computational demand in other areas. An example would be an increased width or depth of an otherwise similar network, without a major increase in the time required to train the network when using depthwise convolutions. While one method cannot be considered inherently superior to the other, the usage of depthwise convolutions can allow for more flexibility when designing a new model thanks to the lower computational demand.

2.1.7 Normalization

As weights change during the training process, the distribution of the intermediate output values and thus inputs for the downstream layers can change drastically. This so called internal covariate shift makes training neural networks harder. A way to address this issue is to apply normalization to the inputs of hidden layers. Using such normalization in neural networks improves the training stability and makes them less susceptible to overfitting (Ioffe and Szegedy, 2015). The CNNs used in this thesis use one of the following two different normalization methods.

Batch Normalization

A popular and intuitive normalization methods is to use Batch Normalization (BN). It takes the popular concept of normalizing input images for the training and applies it to intermediate inputs within the neural networks. The normalized inputs $\hat{x}_i$ are given by

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}, \tag{5}$$

where the inputs x_i are normalized based on the mean μ_B and variance σ_B^2 of the batch, with ϵ denoting a small value used to increase the stability of the process.

Using such normalization can however limit the range of possible values too much. The expressiveness and flexibility of the model is retained by changing the normalized values:

$$y_i = \gamma \hat{x}_i + \beta, \tag{6}$$

where β denotes the shift and γ denotes a scaling parameter applied to the normalized value. These parameters are trainable and adapted throughout the training process.

Layer Normalization

Since BN works using the whole batch, it naturally provides a better normalization effect with a greater batch size. Layer Normalization (LN) was proposed as a method to overcome the dependence of BN on using a large batch. While independent of the batch size, LN has the downside of having a weaker regularization effect (Ba et al., 2016). The process for normalization, shifting and scaling are identical, however the mean and variance are computed independently for each element across all feature maps, rather than per feature map across the whole batch.

2.1.8 Network Structures

The structure of a CNN is a key factor for the performance. Below concepts and terminology relevant throughout the thesis which are illustrated in Figure 8 are discussed

Block

A block in the context of CNN architectures refers to a sequence of layers, normalizations and activation functions which is usually repeated multiple times throughout the network. Examples for such blocks can be found in Figure 17. Inputs and outputs for blocks usually have the same spatial dimension and the same number of channels.

Stage

Most modern CNNs can be divided into multiple stages. A stage typically consists of multiple blocks and is defined and discriminated from other stages by the shape (spatial dimension and number of channels) of the (intermediate) inputs/outputs which usually do not change within the stage. In modern CNN architectures such as ConvNeXt (Section 3.2.4) a new stage usually starts by cutting the spatial dimension of the feature maps in half using a pooling method while doubling the amount of channels.

Stem

A stem layer or stem refers to the first layer(s) in a network. The purpose of the stem layer is the downsampling of the initial RGB input image. Typical characteristics of a stem layer are a relatively big kernel size (e.g. 7x7) to allow for a larger receptive field, and a stride of two or more to achieve the downsampling.

Body

The body of a network contains all the stages of the network. Usually the body ends after the last convolutional layers in the network and is referred to as feature extractor. The output of the body is typically prepared for the downstream task (in this case image classification) by flattening the feature maps into a vector, or applying GAP, which produces a singular feature vector as well (Section 2.1.5).

Head

The head of a CNN is located after the body and handles the downstream task the network is designed for. In case of this thesis a so called classification head is applied. Typically a classification head is a relatively simple MLP if the feature vectors were flattened. If GAP was applied on the feature maps, the head can be as simple as a direct mapping (fully-connected) of the feature vector to the output.

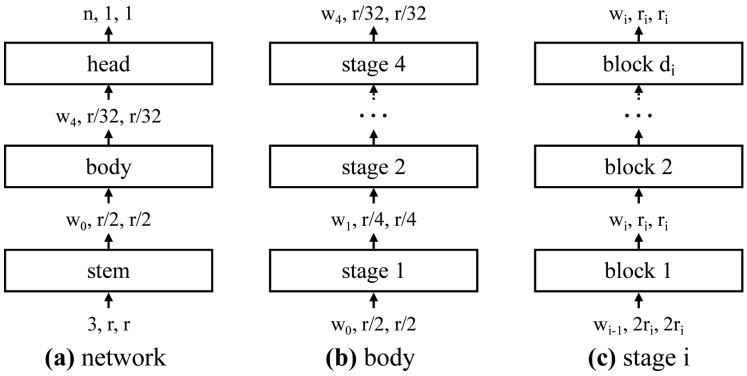

Figure 8: Example structure for a modern convolutional neural network. a) shows the basic structure consisting of a stem, the body and the head. b) shows a sample body, consisting of multiple stages. c) shows a stage consisting of multiple blocks. As explained in Section 2.1.8 the number of channels or width w_i is usually doubled between stages, while the spatial dimension r is halved. As depicted in c) this usually happens before the output of the prior stage is fed into the first block of the new stage, while number of channels and spatial dimension is retained throughout the stage.

2.1.9 AdamW Optimizer

A CNN is initialized with random weights and then trained by feeding images into the network and calculating scores for each class in a so called forward pass. In the so called backpropagation, the direction in which the network weights have to be changed in order to increase the score of the correct class and decrease the score of wrong classes is calculated using the gradients and the weights are changed slightly in this direction (Nielsen, 2015).

Achieving convergence to a optimal solution with a fast yet stable training process while retaining good generalization to unseen test images remains challenging and many different approaches like Stochastic Gradient Descent (SGD), RMSProp and Adam (Ruder, 2016) exist.

For the training process in this thesis the AdamW optimizer was used. AdamW uses so called momentum terms and a decoupled regularization term to achieve a stable training and a good generalization. The update for network parameters θ at timestep t for AdamW is given by

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} - \lambda \cdot \theta_{t-1} \tag{7}$$

where α denotes the learning rate (LR), $\hat{m}_t$ denotes a momentum term indicating the direction in which the weights are to be moved which is derived from the current and previous gradients and replaces the gradient term in classic backpropagation. A bias correction term $\sqrt{\hat{v}} + \epsilon$ is used to limit the variance of the gradients to further stabilize

the training.

A so called decoupled weight decay term $\lambda \cdot \theta_{t-1}$ is added, where λ is usually a small number denoting the desired strength of the weight decay. According to Loshchilov and Hutter (2017), the decoupled weight decay helps to further improve the already stable training process of the Adam optimizer and allows for better generalization to unseen data.

2.1.10 Cosine Annealing Learning Rate Scheduling

Adapting the learning rate α during training to fit the stage of the training is an essential method to properly train a CNN. For this thesis so called cosine annealing learning rate scheduling was used which adapts the learning rate η_t for epoch t as

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min}) \left(1 + \cos\left(\frac{T_{\text{cur}}}{T_{\max}}\pi\right)\right), \tag{8}$$

where $\eta_{\max}$ denotes the initial LR and $\eta_{\min}$ is the minimum LR (often zero). The current and total epoch count is denoted by T_{cur} and $T_{\max}$.

This scheduler can be viewed as having three phases, an initial phase, where the learning rate stays close to the initial LR for a relatively long time, which allows the network to make meaningful changes and learn broad representations of the data quickly. The second phase is when the learning rate is dropped to reduce the steps made by the optimizer and thus allow the network to converge. The final phase can be considered as a finetuning phase, where the learning rate is close to zero and only minor adjustments are made to the weights.

2.2 Interpretable and Explainable AI

Interpretable/Explainable Artificial Intelligence (XAI) is a relatively new research field which has gained popularity as AI methods became more and more complex and less intuitive to understand. The term is often used vaguely to describe multiple distinct approaches to allow humans a better understanding of the AI. According to Saranya and Subhashini (2023), XAI approaches belong to one of two main categories.

So called "intrinsic models" are self-interpretable applications of AI. Examples for this are a decision tree or linear regression. The second category are so called "post-hoc" approaches. The idea of these methods is to be applicable to an AI model which was already trained and wasn't designed to be interpretable. Such AI methods are often referred to as black-box models. Post-hoc approaches aim to offer an interpretation of these black box models in various ways. For this thesis, the chosen XAI methods produce heat maps to indicate important areas for a CNN during a classification task.

Post-hoc methods are further divided into model-agnostic and model-specific. The term model-agnostic is used to describe methods, which are applicable on a wide variety of

ML algorithms across different application domains like LIME (Section 4.3), while model-specific is used to describe methods that have a constrained application domain, for example if they are only applicable to a specific subset of CNNs and possibly impose restrictions or require modifications to a model in order for the method to work as it is the case with CAM (Section 4.1.1).

A clear distinction in model-specific or model-agnostic can be hard to make, as methods might for example require modifications to models or the XAI method might need to be adapted to the task in order to work, which would usually make it a model-specific method. However such applications might be relatively easy to implement on a wide range of models without significantly deteriorating the performance of the model, which would put the method more towards the model-agnostic domain.

The main goal of XAI methods is to offer the user a better understanding of the applied AI model. Arrieta et al. (2019) further splits the goal into different sections.

Trustworthiness

XAI methods are considered as a way to increase the trust of a user when interacting with AI.

Causality

With the use of XAI methods one can attempt to find causal connections in the data.

Informativeness

AI methods are supposed to give the user information either as decision support, or as a control mechanism for the decisions made by a fully automated AI based system. This is especially true for classification tasks where the output of a model is a score vector with no real way for humans to understand how it was produced.

Confidence

XAI-Methods can help identify the confidence of a network in certain choices made or how confident a network is that a certain part of the input is important. At the same time it can help identify lack of confidence or confidence for the wrong reasons, allowing improvements to the model, by for example adapting the dataset to address the issues.

Other goals mentioned are fairness, accessibility, interactivity, privacy awareness and transferability.

While these are all reasonable goals, evaluating XAI methods with these goals in mind is a hard task. It is for example not really possible to quantify which method is better suited to convey information or create trust and confidence with an user, as it strongly depends on the interacting person.

2.3 Comparison and Evaluation of XAI Methods

When a new CNN architecture or improvements to training procedures are proposed in an AI paper, typically a performance evaluation on the Imagenet Dataset (Deng et al., 2009) is part of it. This is for example the case for the three published CNNs used in this thesis (Section 3.2). Reinforcement Learning approaches are often benchmarked on the Atari Console Games (Brockman et al., 2016) and Large Language Models are usually compared using the Massive Multitask Language Understanding Benchmark (Hendrycks et al., 2020).

In XAI papers such benchmarks are often missing, making it hard to get a proper grasp how methods compare to one another. In many publications the focus is set more towards human interactions with the XAI method. Comparing XAI methods using experiments with human participants, as done for example in the Grad-CAM (Selvaraju et al., 2016) or XRAI (Kapishnikov et al., 2019) papers is a popular approach to argue that a novel method is useful. User-studies in general have become popular when it comes to evaluating the different aspects of XAI methods discussed in the previous section, but in many cases rely heavily on a single CNN or focus on a single method (Müller, 2024), rather than offering a broader comparison of multiple XAI methods on different network architectures.

While working with user-studies can be a good way to study how humans interact with XAI methods and if an XAI method is able to instill trust and confidence for the underlying AI method, they lack comparability even if multiple CNNs and XAI methods are compared, due to the subjective nature of the evaluation. They thus should not be considered a reliable method to rank XAI method against one another.

More quantifiable attempts to benchmark XAI methods include statistical evaluations based on the ground truth of images using datasets with given bounding boxes. However as discussed in more depth later in this thesis, the different ways how XAI methods produce their respective important map can distort the results of statistical evaluations.

Additionally even with an available ground truth, often given by a generously sized bounding box, drawing a direct conclusion to the quality of XAI methods might not be possible, as the network might either be able to learn the class from other related elements outside the bounding box, or only a small part of the area inside the bounding box is actually relevant for the network. In that case an XAI method would wrongfully be considered superior, if it was faithful to the ground truth rather than being faithful to the underlying decision-making process of the AI.

For example an AI method might be able to detect faces by only using the area around the eyes, while a bounding box in a corresponding face detection dataset might contain the whole face. XAI methods which would produce a more precise area around the eyes would thus be more truthful to the networks decision-making process, but would likely perform worse compared to a method that works less precise and thus covers a larger area of the bounding box.

According to Zhang et al. (2023) “standardized benchmarks for evaluating them [visual explanations] are seriously underdeveloped”. They further criticize the lack of scaling and the high variance when human assessment through user interfaces or questionnaires

is used. However their offered benchmarking, similar to most approaches to create such benchmarks is based on only one model trained on Imagenet.

While such attempts can offer certain value, the use of Imagenet and the limitation to using a single model have downsides, which will be discussed multiple times throughout this thesis.

This thesis pursues to offer an quantifiable, relevant and fair evaluation of multiple XAI methods, thanks to the use of a dataset designed specifically for the task at hand and an in depth comparison of XAI methods using multiple different CNNs.

3 Methods

3.1 Dataset Selection and Creation

Datasets play a vital role when it comes to successfully training CNNs. Large well designed datasets like Imagenet (Deng et al., 2009) have been very important for the progress made in the field of computer vision. While using a well designed high quality dataset that is commonly used in research papers would be ideal, these datasets are not well suited for the proposed approaches of evaluating XAI methods in this thesis. In order to properly compare the outputs of XAI methods this way, the important area in every image should be as unambiguous as possible.

To ensure a fair comparison using the approach taken in this thesis, the important areas in the image should have limited and similar size and for statistical comparisons as a sanity check, annotations for the exact location of the important object. Another requirement to the dataset is that the CNNs are capable of learning and generalizing the data close to perfection. In case of a lower accuracy, poor performance of an XAI method might be explainable by the poor performance of the network, thus not allowing for meaningful conclusions to be drawn for the XAI method itself. The dataset should thus be simple and allow for the used CNNs to achieve an accuracy close to 100%. Most popular and high-quality datasets can not meet all these requirements.

For this thesis the choice was made to use a dataset with traffic signs. Traffic signs usually have a limited and similar size for images taken from cars and it is usually unambiguous which part of the image is important. While large datasets like the German Traffic Sign Detection Benchmark (Houben et al., 2013) or the Mapillary Traffic Sign Dataset (Ertler et al., 2019) exist, they fail to meet the requirements stated above. A traffic sign indicating pedestrian crossing for example usually is accompanied by markings on the ground, or a network could learn that a sign for a high speed limit is connected to driving on a highway, making it unclear if elements other than the sign itself could by themselves be enough for the CNN to learn the related class. Other problems are, that in some cases the relevant traffic sign can occur multiple times in images, for example "no entry" signs are usually located both to the left and the right side of a corresponding road. Signs can sometimes be paired up with other signs as well, for example a speed restriction might regularly be located next to a sign indicating road works. To achieve optimal results and have good control of the desirable characteristics for the images, the most suitable solution was the creation of an artificial task-specific dataset.

This dataset, which will be referred to as Augmented Traffic Sign Dataset (ATSDS) consists of images created by inserting a singular street sign on an appropriate background, ensuring perfect knowledge of the area relevant for the classification. This enables control of the sign size and thus the importance area and limits the occurrence of the relevant sign, while disconnecting any elements of the background like street markings or road characteristics from the used signs. The created dataset allows the evaluation of the XAI methods using the approach proposed later in this thesis and a meaningful statistical analysis of the importance areas alongside it thanks to a perfect knowledge of the underlying theoretical ground truth. The following section introduces the datasets used and the steps taken when creating the images for the ATSDS dataset.

3.1.1 GTSRB

The German Traffic Sign Recognition Benchmark - GTSRB (Stallkamp et al., 2012) is a popular dataset for german traffic signs. It consists of around 50.000 close up images of 43 unique traffic classes of traffic signs. Image sizes differ, but most images are between the size of 40x40 and 60x60. All images are annotated with bounding box coordinates, making it easy to extract cutouts of the sign given knowledge of the shape. Example images from this dataset are displayed in Figure 9.

Figure 9: Example Images from the GTSRB dataset for three of the 43 classes.

3.1.2 Google Street View Dataset

The Google Street View Dataset consist of 62,058 high quality Google Street View images (Zamir and Shah, 2014). All images have a size of 1280x1024 and show American streets. The images were retrieved using Google Street view screenshots from urban or suburban areas in Pittsburgh, Orlando and Manhattan. Images are provided as so called placemarks. The description by Zamir and Shah (2014) is as follows: "For each Street View placemark (i.e. each spot on one street), the 360° spherical view is broken down into 4 side views and 1 upward view. There is one additional image per placemark which shows some overlaid markers, such as the address, name of streets, etc." Figure 10 showcases example images from the dataset.

Figure 10: Example Images from the Google Street View Dataset. Navigation Elements from Google Street View can be seen in the top left of the images.

3.1.3 ATSDS

The Augmented Traffic Sign Dataset (ATSDS) is a task-specific dataset created for this thesis. It consists of a total of 9500 images, divided into 19 classes with 450 train and 50 test images each per class.

It is very notable, that the ATSDS features less classes and images compared to the GTSRB dataset used to provide the traffic sign images. This is because the GTSRB dataset is very unbalanced, with the amount of available samples for some classes being less than 150, while other classes featured more than 1500 images. To still have a decent amount of classes, while having relevant amounts of samples per class, classes consisting of less than 500 images were excluded to rule out any distortion of the results due to unbalances in the dataset. Similarly for classes with more than 500 samples, only the first 500 images were used. For the remaining images, a cutout of the signs was extracted by cropping the image to the bounding box and applying a mask with the appropriate shape for the traffic sign. The signs had 5 possible shapes (round, triangle, reverse triangle, rhombus, octagon). A detailed rundown of the shapes and classes with class descriptions can be found in Appendix A

For the background images the four suitable side views for each placemark were used. Each image was cropped to hide items belonging to the Google Street View interface. Further cropping of the sides was applied with the goal to create a squared image, in or-

der to eliminate any distortion in the resizing process. The cropped images were resized to a 512x512 resolution.

For each class of street signs, 450 images from the selected classes in the GTSRB dataset were randomly sampled for the training set, while the remaining 50 were used for the test set. The final image was generated by first cropping the image towards the bounding box and then resizing it to 128x128. The decision for this size was based on the fact that training with smaller crops (64x64, 96x96), particularly for VGG (Section 3.2.2), was not successful. Using the mask for the corresponding shape, the area containing the actual sign from the 128x128 crop was added on top of the background image. As a result of this process, the traffic signs on average take up 5% on an image as Figure 11. It is worth noting that each sign and each background was used exclusively for a single sample, ensuring that there was no duplication of images across samples.

Figure 11: ATSDS example images featuring all of the five possible traffic sign shapes.

3.2 Convolutional Neural Networks

The performance of an XAI method can differ a lot depending on the CNN architecture it is applied to. Many CNN architecture exist today, often featuring very different and unique designs. It is essential to evaluate XAI methods across multiple architectures to ensure that results are reliable and have a high chance to work well on a wide range of other CNN architectures. Given the scope of this work, the decision was made to test the methods on four different network architectures of varying complexity and performance. A brief overview of these models and their role for this thesis will be given, before discussing them in more detail.

The first network will be called "Simple CNN" and is specifically designed for this thesis with the main focus on a low depth, limited amounts of parameters and basic building blocks. It consists of only four convolutional layers. This model is designed to test how XAI methods perform and differ when applied to a very basic CNN architecture with few parameters and a simple structure.

The second model is a VGG-16 (Simonyan and Zisserman, 2014). The architecture became popular in 2014 and achieved state of the art results on ImageNet, by using and popularizing 3x3 convolutional filters and increasing the depths of CNNs (more layers) rather than focusing on width (more channels) which was popular beforehand. In the context of this Thesis the VGG fills the role of a more advanced CNN, which is still based on very basic design elements.

While the VGG architecture had proven that a higher network depths is a good concept to achieve better performance, such deep networks were notoriously hard to train. During training, so called vanishing or exploding gradients (Pascanu et al., 2013) would prevent the deeper layers of the network from learning meaningful feature representations.

The ResNet (He et al., 2015) architecture used as third model addressed this issue by introduced so called skip connections, which directly connects the input of a convolutional block to its output and thus creates a more stable gradient flow and training process. This allowed them to train networks with up to 152 layers depth.

While both the VGG and ResNet architecture can no longer be considered state of the art CNNs, they remain extremely relevant with 20.000 (VGG) and 50.000 (ResNet) citations respectively in the last year according to Google Scholar. Additionally these CNNs are especially popular in the field of XAI user studies (Müller, 2024).

The final CNN architecture is ConvNeXt (Zhuang Liu et al., 2022). The ConvNeXt architecture was released at a time, when it seemed like CNNs could no longer keep up with modern transformer architectures like Swin Transformers (Ze Liu et al., 2021). In the paper they showed, that CNNs were still able to compete with transformer architectures achieving state of the art results, by introducing key ideas used in transformer architectures and applying them to a ResNet to turn it into a state of the art network which is able to compete with the performance of advanced transformers.

These models will be discussed in more details, as a deep understanding of their structures is necessary to be able to draw conclusions from the results.

3.2.1 Simple Model

The job of the simple model is to allow observations of XAI methods on basic building blocks. Similar to the VGG, the simple model uses an initial convolutional layer with a large kernel and a stride of 2 to reduce the spatial dimension of the input image while increasing the amount of filters from the 3 filters given by our RBG image to 32. This so called "stem layer" is followed by a 3x3 filter, which retains the spatial dimension because of padding, while increasing the amount of filters to 64. Next a single max pooling layer is used, reducing the spatial dimension to 55x55, followed by two more 3x3 convolutional layers like the previous one, increasing the amount of layers to 128/256 respectively while retaining the spatial dimension.

By using only a single max pooling layer, the spatial dimension of the feature size in the final convolutional layer remains much bigger compared to other models used, which could be especially interesting for the Grad-CAM method. The last convolutional layer is followed by a Global Average Pooling (GAP) which compresses each layer into a single creating a 1x256 feature vector which is then directly mapped to the 19 classes with a fully connected layer. After each of the mentioned convolutional layers a ReLU activation function was applied. The design choice is based on an iterative approach of adding convolutional layers, until the network achieved a sufficient accuracy of more than 90%

Since this simple architecture was very prone to overfitting, Batch Normalization was applied after every convolutional layer and 25% dropout was applied before the global average pooling. The Code for this Model can be found in Appendix B

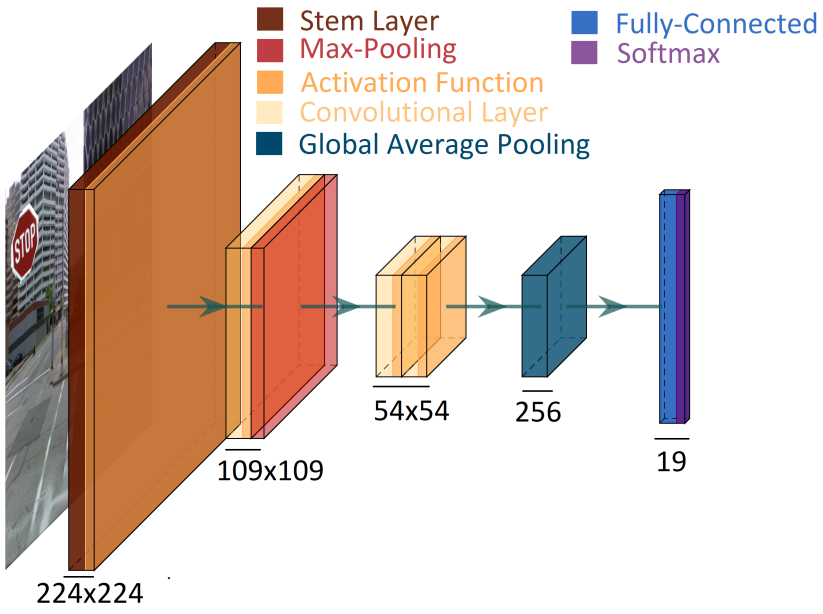

Figure 12: Visualization of the Simple CNN.

3.2.2 VGG

The VGG architecture proposed by Simonyan and Zisserman (2014) is still a very popular CNN today. While the performance on the popular ImageNet benchmark can no longer compete with other methods, the VGG16 model still achieves state of the art performance on other datasets, for example outperforming many more recent CNN designs on the InDL (Yang et al., 2023) dataset achieving equal results to the ConvNeXt architecture.

While being a much deeper network with a large amount of parameters compared to the proposed simple model, it still consist of very basic building blocks, utilizing only 3x3 convolutional filters and 2x2 max-pooling layers for the feature extraction. The receptive field of the 3x3 kernel for the convolutional layers makes it well suited to capture local patterns like edges, without requiring a large amount of parameters compared to a larger kernel. This allowed the VGG to have a higher depth without massively increasing the needed time to train the network.

While 3x3 kernels are thus a good choice, they lack the capability of reducing the spatial dimension of the feature maps, which is necessary to reduce the amount of parameters and thus enable the depth of the network. This is achieved using 2x2 max pooling layers with a stride of 2 in the VGG. Reducing the spatial dimension of the feature maps, while increasing the number of channels in the way proposed by the VGG architecture remains one of the core foundations for state of the art CNNs.

Simonyan and Zisserman (2014) use these ideas to build different CNNs. For this thesis the decision was made to use their VGG16 design. It offers a very good trade-off in terms of computational expense and performance compared to more shallow models like the VGG11 or the slightly deeper VGG19 which achieved only marginally better results.

The VGG16 architecture is composed by the feature extractor which consist of 13 convolutional layers, combined with five max pooling layers. The output of this feature extractor is flattened and fed into the classification head consisting of three fully connected layers as depicted in Figure 13.

3.2.3 ResNet

Residual Networks (He et al., 2015) and the skip connections introduced by them are an important concepts when it comes to training very deep CNNs. The name for these connections comes from the idea of adding the input of a convolutional layer to its output. These connections which are sometimes also referred to as residual or shortcut connections improve the gradient flow, which can alleviate the problem of vanishing gradients. As depicted in Figure 14 such connections are usually applied around blocks, rather than being used on each convolutional layer.

ResNet made it popular to heavily downsample the input, using a convolution with a larger kernel and a stride of 2, followed by a pooling layer on the RGB input image. ResNet specifically made use of a 7x7 kernel with stride 2 and followed by a 3x3 max pooling with stride 2, downsampling their 224x224x3 input images down to a 56x56x64 feature map, before feeding it into the rest of the network. Similar procedures were widely used afterwards for example for the RegNet (Radosavovic et al., 2020) or the DenseNet

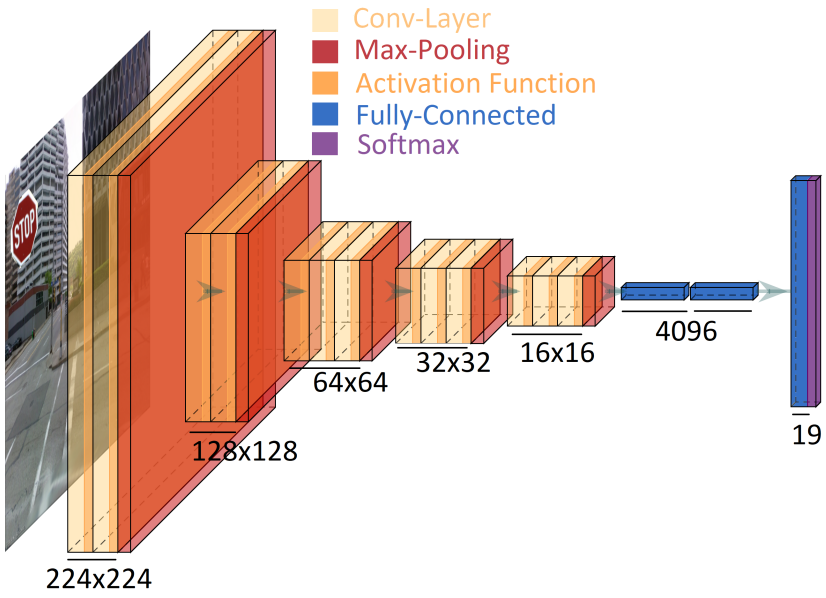

Figure 13: Simplified visualization of the VGG16 model structure.

(Huang et al., 2016) architectures. For the very deep architectures with 50, 101 and 152 layers respectively, a bottleneck block as depicted in Figure 14 was introduced, to reduce the per-layer complexity of the network and allow for training in a reasonable amount of time, while increasing the depth. As a Result the ResNet50 with the bottleneck design and the ResNet34 without it have the same complexity (FLOPs). For this Thesis the ResNet50 was used. It offers a great tradeoff in terms of computational demand and performance which makes it a popular choice in the field of XAI (Saranya and Subhashini, 2023). The model architecture is illustrated in Figure 15

3.2.4 ConvNeXt

The ConvNeXt architecture proposed by Zhuang Liu et al. (2022) showcased, that CNNs are still able to compete with transformers and achieve state of the art results on image classification tasks. In their paper they showed, that many advances in the field of image classification which had been attributed to transformers were not because of the intrinsic characteristics of the transformers, but could instead be attributed to other reasons like the use of new activation functions, better normalization methods or a multitude of small improvements to the training procedures.

Zhuang Liu et al. (2022) started off with a basic ResNet50 (He et al., 2015) and applied different improvements to the architecture. The improvements are grouped into 5 categories. The first group was called the “Macro Design” and consists of improving the initial downsampling and a better stage design.

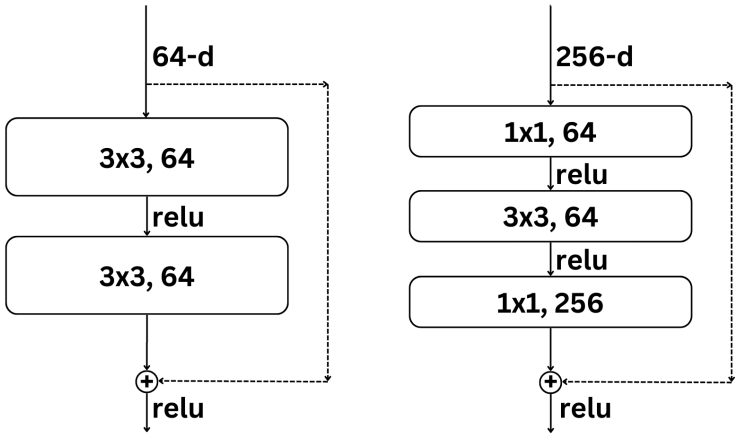

Figure 14: Proposed Block Designs for ResNet18/30 (left) and ResNet50/152 (right). Residual connections allow for better gradient flow. Typically residual connections skip multiple layers or a block (He et al., 2015).

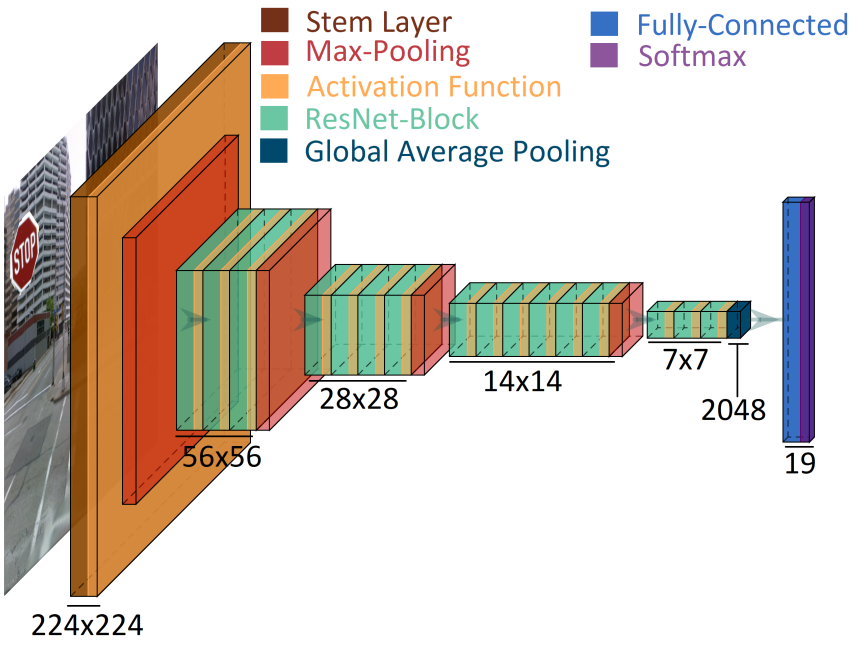

Figure 15: Simplified visualization for the ResNet50 architecture.

Many CNNs, including ResNet, achieve initial downsampling through a 7x7 kernel with stride 2 as initial stem layer, followed by a 3x3 max pooling with stride 2. Transformer

architectures on the other hand usually split the image into patches, which are embedded and merged. Drawing inspiration from the Swin Transformer (Ze Liu et al., 2021) design, the stem layer was replaced with a "patchify layer" which is a non overlapping convolution with a 4x4 kernel and stride of 4. Both the ResNet approach and the patchify layer downsample the input image to a spatial dimension of 56x56xC where C is the amount of channels.

As second improvement to the macro design, the amount of blocks per stage (Section 2.1.8) was rearranged to have the same ratio used in Swin Transformers. The four stages with a (3,4,6,3) block per stage structure were changed to a (3,3,9,3).

The second group of improvements is called "ResNeXt-ify" and as the name indicates introduces ideas from the ResNeXt (Xie et al., 2016). The key elements of ResNext is to use grouped convolutions. ConvNeXt uses group sizes of one, so called depthwise convolutions. As discussed in Section 2.1.6 converting standard convolutions to depthwise convolutions reduced the amount of operations and the amount of parameters in the network, resulting in a lower computational demand, but a decrease in capacity as well.

The reduced computational demand allows for a width increase. ConvNeXt compared to ResNet50 increases the amount of channels by 50%. This adapted design has a similar computational demand and complexity to the original design, while achieving a higher accuracy.

As a result of this so called depthwise convolution, spatial and channel mixing does not happen at the same time, which is a property found in vision transformers as well. It is important to note, that only the 3x3 layer is using depthwise convolutions as depicted in Figure 16 - b). Thus the 3x3 convolution is responsible for the spatial mixing, while the 1x1 pointwise convolutions are responsible for the channel mixing.

As third improvement taken from transformers, the bottleneck structure was changed into an inverted bottleneck. Transformers usually contain such inverted bottleneck blocks, where the inverted bottleneck has four times the amount of channels compared to the input. The changed block design depicted in Figure 16 - b) slightly improves performance and allows for the following design changes.

The fourth big improvement for the ConvNeXt block is to increase the kernel size. While it had been the gold standard for CNNs since the release of the VGG architecture to primarily use 3x3 kernels, modern transformer architectures utilize larger kernels for a bigger receptive field. In order to be able to use larger kernels, the 3x3 layer was moved up in the block design. By using the depthwise convolution with a reduced number of filters at the beginning of the block the kernel size can be increased without a massive increase in computational demand during training. The kernel size for this layer was subsequently increased to 7x7, which achieved the best accuracy when testing larger kernel sizes.

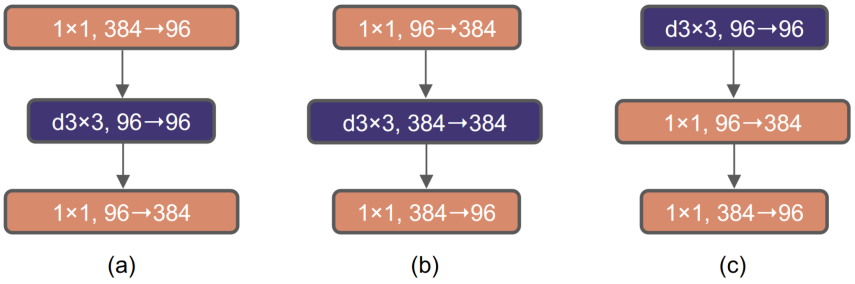

Figure 16: a) ResNeXt block b) Inverted bottleneck - c) ConvNeXt block design. The image from (Zhuang Liu et al., 2022) shows the inversion of the bottleneck, followed by moving the depthwise convolution (d3x3) upstream to allow for a kernel size change afterwards.

As final step, the "Micro Design" of the network was optimized through multiple small changes.

An improvement in accuracy was achieved, by replacing the ReLU activation function with GELU, which has been widely used in advanced transformer architectures. Additionally the GELU was only applied once per block after the first 1x1 convolution, rather than applying it after every single convolutional layer, which had been the standard for CNN designs.

Similarly a reduction of normalization layers gave another slight performance increase, thus the ConvNeXt block only uses a normalization layer after the depthwise convolution as depicted in Figure 17

Batch normalization in ResNet is done using the BatchNorm (BN). The simpler Layer Normalization (LN) on the other has become popular in transformer architectures. While this replacement doesnt improve performance if done by itself on the ResNet, it improves performance on the adapted blocks/architecture for ConvNeXt, indicating that the changed and now shared characteristics of transformer architectures could be a necessity for the successful usage of LN.

Finally the downsampling step is replaced by using 2x2 convolutions with a stride of 2 accompanied by an additional normalization layer which helps to stabilize the training.

The resulting model is called ConvNeXt-Tiny (ConvNeXt-T) (Figure 19). Larger versions of the ConvNeXt architecture exist, but using them would have significantly increased the computational expenses while the added network capacity would have likely offered no benefit for the research question of the thesis.

Zhuang Liu et al. (2022) introduced many other small improvements to the training process and the network, but discussing them in depth would not be possible within the scope of this thesis. A notable one relevant to the results is LayerScale (Touvron et al., 2021) and will thus be discussed. LayerScale like many other improvements is a method usually used to stabilize the training process of deep vision transformers. The key idea is to learn a diagonal matrix for each block, which is then multiplied on the results and

thus used to scale them versus the residual connection. As discussed in Section 3.2.3 the main reason for introducing residual connections is to stabilize the training and prevent exploding or vanishing gradients in deep networks. This means if the results of the block are much higher compared to the residual connection, the effect of the residual connection could be weakened, risking a worse training process. Similarly if the impact of the residual connection is unproportionally high, the block might not be able to convey enough relevant information, limiting the performance of the network. The LayerScale technique can thus be interpreted as a way to balance the importance of the output of each block, versus the residual connection and as an attempt at achieving an ideal ratio of the two elements for an optimal training process.

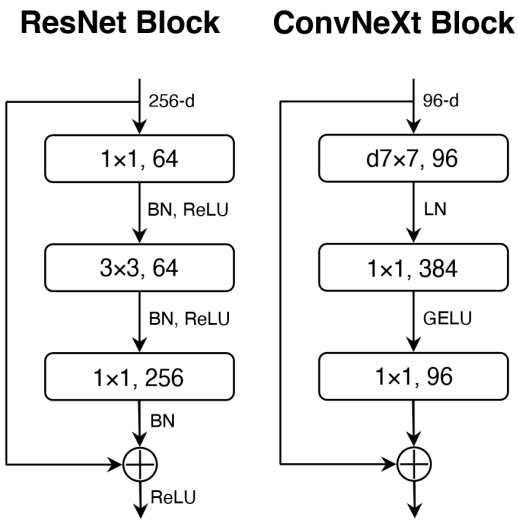

Figure 17: Comparison of the ConvNeXt block (R) with the ResNet50 block, it is based on (Zhuang Liu et al., 2022). Compared to the design shown in Figure 16 the kernel size of the depthwise convolution is increased, LN is used after the depthwise convolution and GELU is applied after the first 1x1 kernel.

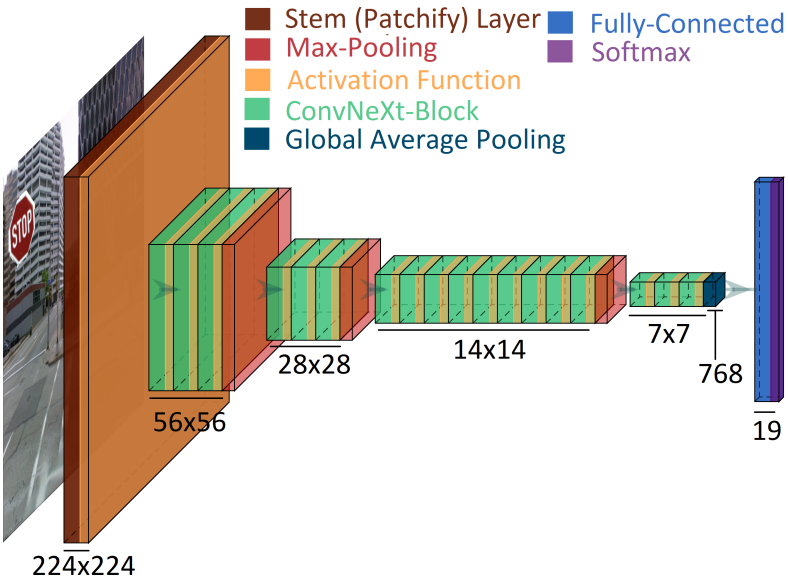

Figure 18: Simplified visualization for the ConvNeXt network.

	output size	• ResNet-50	• ConvNeXt-T
stem	56×56	7×7, 64, stride 2 3×3 max pool, stride 2	4×4, 96, stride 4
res2	56×56	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} d7\times7, 96 \\ 1\times1, 384 \\ 1\times1, 96 \end{bmatrix} \times 3$
res3	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} d7\times7, 192 \\ 1\times1, 768 \\ 1\times1, 192 \end{bmatrix} \times 3$
res4	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} d7\times7, 384 \\ 1\times1, 1536 \\ 1\times1, 384 \end{bmatrix} \times 9$
res5	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} d7\times7, 768 \\ 1\times1, 3072 \\ 1\times1, 768 \end{bmatrix} \times 3$
FLOPs		4.1×10^9	4.5×10^9
# params.		25.6×10^6	28.6×10^6

Figure 19: ConvNeXt-T architecture compared to the ResNet50 (Zhuang Liu et al., 2022).

3.2.5 CNN Training and Testing Procedures

This section will provide a short overview of the procedures for training and testing. Detailed information like used hyperparameters can be found in Appendix C.

All of the discussed CNNs were trained using the same training procedure for an amount of 200 epochs. The amount of epochs was used as a trade-off between the needed training time and the achieved accuracy. As discussed in Section 6.1 accuracy for both training and test did not show any meaningful changes toward the end of the training, while achieving accuracies sufficiently close to 100 percent for all trained models.

For training, broadly following the procedure described by He et al. (2015), the input to the final network was derived by resizing the image to 256x256. A 224x224 random crop was taken from the resized image and normalized using mean and standard deviation of the dataset. This means a slightly different view of each image is shown to the network every epoch, which can prevent overfitting and helps the model to generalize better.

The test accuracy during training was determined by using a 224x224 center crop instead of a random crop.

Training was conducted using the AdamW optimizer discussed in Section 2.1.9, using a cross-entropy loss. The learning rate was adapted throughout the training process using the cosine annealing learning rate scheduling (Section 2.1.10).

4 XAI Methods

4.1 Grad-CAM

4.1.1 CAM

One of the earliest XAI methods attempting to offer visual explanations for the decision-making of CNNs is a method called "class activation mapping" in short "CAM" (Zhou et al., 2015) The core idea behind CAM is, that features from the last convolutional layer retain their connection to the corresponding area in the input image. These features obtained from the convolutional part of the CNN are directly linked to the classes by using Global Average Pooling (GAP) to reduce the spatial dimension of the feature map followed by a single fully connected layer to the output vector as classification head. When inserting an input image into such a CNN, the weights of the fully connected layer directly correspond to the importance of each of the features for the respective class. These weighted features are then combined into a single heatmap, which is upsampled using bilinear interpolation to match the size of the input image.

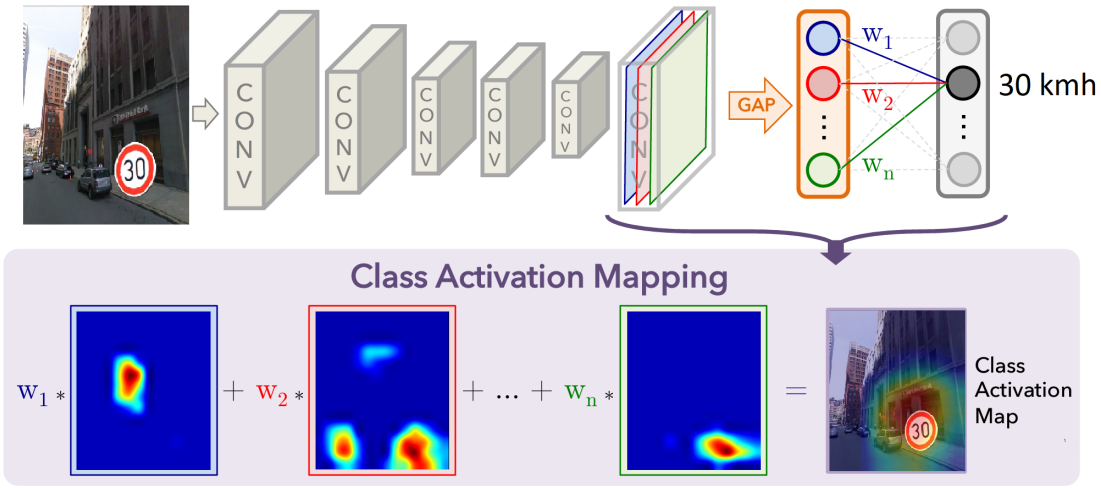

Figure 20: Adapted image from the CAM paper (Zhou et al., 2015) showcasing how features are weighted and combined to retrieve a class specific heatmap of relevant regions

4.1.2 Grad-CAM

The Grad-CAM (Selvaraju et al., 2016) method generalizes the core idea introduced by CAM to extract localization information for features from the last convolutional layer for networks that do not adhere to the restrictions set by CAM for the classification head (GAP followed by a single fully connected layer). Since such networks do not offer a direct weighting of the features, Grad-CAM instead uses the gradients of the feature

maps with respect to the desired output class as weight. The importance weight α_k^c for a feature map k and a class c is thus given by the pooled gradients of the feature maps with respect to the output Y^c of the model for the class of interest c :

$$w_k^c = \frac{1}{Z} \sum_{i=1}^{h} \sum_{j=1}^{w} \frac{\partial Y^c}{\partial A_{ij}^k} \tag{9}$$

where

- A_{ij}^k is the feature map k at location (i, j).
- h and w are the height and width of the feature map.
- Z is a normalization constant (typically $h \times w$).

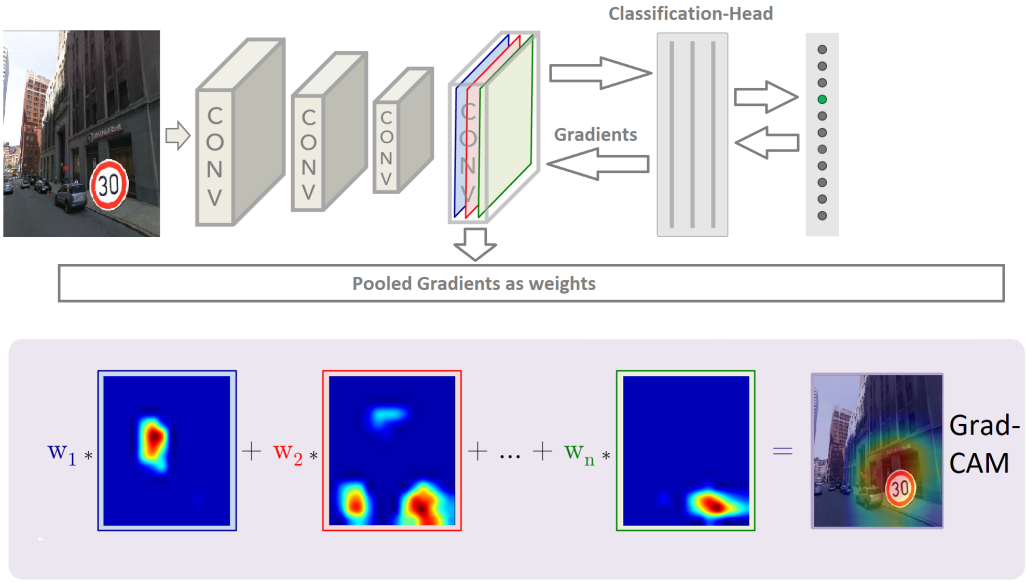

Figure 21: Adapted Diagram from CAM (Zhou et al., 2015) showing the core similarities but different derivation of weights for the Grad-CAM algorithm

4.2 PRISM

Principal Image Sections Mapping (PRISM) unlike many other methods does not depend on backpropagation. Similar to Grad-CAM PRISM assumes that the features in the final convolutional layer have a high spatial correlation to relevant areas in the input image. PRISM computes the Principal Component Analysis (PCA) for the last convolutional layer and truncates the results after the third principal component.

More precisely Singular Value Decomposition (SVD) is used to obtain principal component vectors. This is done by reshaping the final outputs $\mathrm{A}^{n\times c\times h\times w}$ where n is the batchsize, c the amount of channels and h and w denote the height and width of our featuremaps to be two dimensional $\mathrm{A}' = \mathrm{A}^{v\times c}$ where $v = n \cdot h \cdot w$. This matrix is centered

$$\mathrm{A}'' = \mathrm{A}' - \mathrm{mean}(\mathrm{A}') \tag{10}$$

and SVD is computed

$$\mathrm{A}'' = \mathrm{U} \cdot \mathrm{S} \cdot \mathrm{V}^{\mathrm{T}}. \tag{11}$$

Now the matrix

$$\mathrm{A}_{\mathrm{PCA}} ::= \mathrm{U} \cdot \mathrm{S} \tag{12}$$

is defined from which the first three columns are used and interpreted as the respective color channels: red, green, blue (Szandała and Maciejewski, 2022).

PRISM produces a RGB heatmap with a gray base color, where important regions can be identified by a bright color. To extract a heatmap from the rgb output, the image is normalized around the grey area and the max value of the channels is selected as value for the heatmap. Visualizations for this process are depicted in Figure 22

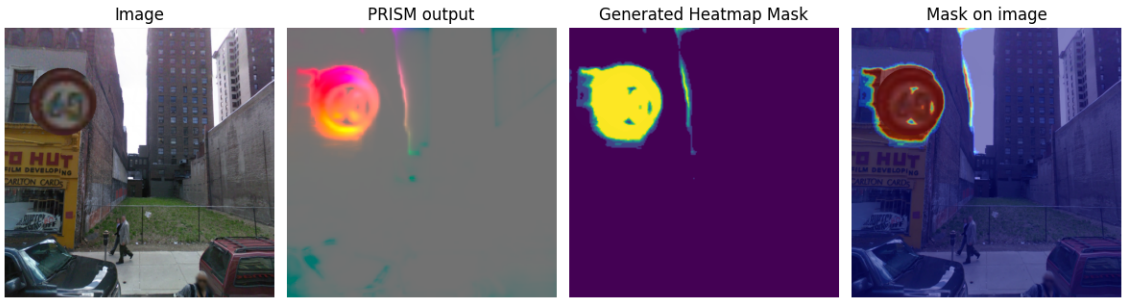

Figure 22: Example images for the PRISM method. Original image (1), PRISM output (RGB) (2), the generated single channel heatmap (3) and a visualization of the mask on top of the image using alpha blending (4).

4.3 LIME

Local Interpretable Model-agnostic Explanations (LIME) (Ribeiro et al., 2016) is, as suggested by the name, a model agnostic approach to explain models. It uses perturbations

of the input data to explore the behavior of the model for a certain local area. When applied to CNNs, so called super-pixel segments, which are essentially larger areas of the input image, are perturbed. The perturbed instances are fed into the CNN to receive the outputs. Using these outputs, a new interpretable model is trained with the goal of approximating the behavior of the CNN for each image. This interpretable model used typically and in this thesis is a sparse linear classifier. The importance of the regions/superpixels in the original input image can directly be derived from the weights of the sparse linear classifier. This allows for a ranking of the features.

To use the results for the comparisons, a heatmap is derived using only the most important feature which in tests performed better compared to adding more features. It is important to note, that alongside the amount of chosen features, many other parameters can heavily influence the results of the LIME method. Details in regards to the implementation can be found in Appendix D. An example heatmap derived this way is depicted in Figure 23. It is important to note, that LIME only highlights the feature and does not include any weighting within the important/unimportant area. This result in a binary heatmap. The problems of binary heatmaps in the context of this thesis and how it is addressed can be found in Section 4.7.

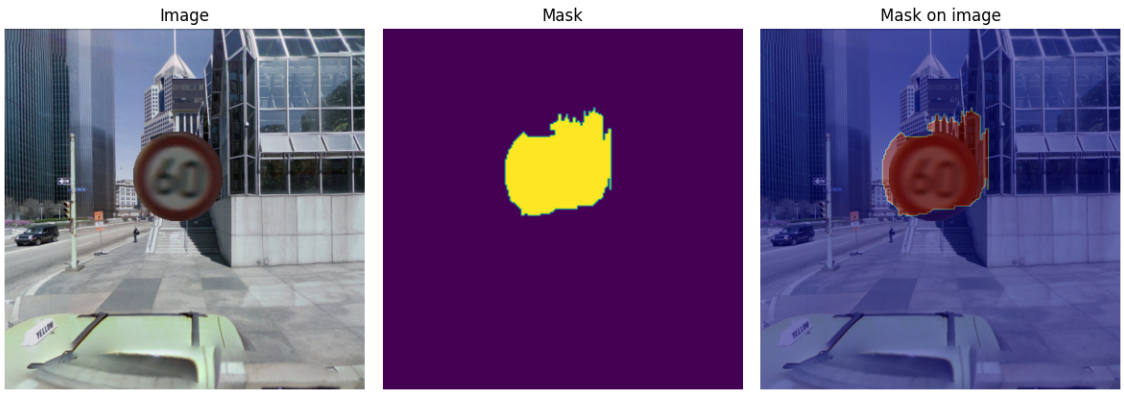

Figure 23: Example images showcasing the LIME heatmaps. Original image (L), LIME mask (M) and an overlayed mask (R). Most notably unlike other methods LIME produces specific features, resulting in a binary mask, rather than a smooth heatmap.

4.4 Integrated Gradients

Unlike the methods discussed so far, the Integrated Gradients (IG) method introduced by Sundararajan et al. (2017) operates on the input level of the network. This should in theory make it less susceptible to the network architecture. Given a function (usually referring to a neural network) F, IG computes the gradients on the path from a baseline x' to the real input x. The integrated gradients are given by

$$\text{IntegratedGrads}_i(x) ::= (x_i - x_i') \cdot \int_{\alpha=0}^{1} \frac{\partial F(x' + \alpha \cdot (x - x'))}{\partial x_i} d\alpha, \tag{13}$$

where in the context of image inputs i would refer to every single element of our input (R, G or B value of each pixel). When IG was introduced, a black image was typically used as baseline, but this has serious downsides as discussed in Section 4.5.

To efficiently compute this, the integrated gradients are instead approximated as

$$\text{IntegratedGrads}_i^{\text{approx}}(x) ::= (x - x') \cdot \sum_{j=1}^{n} \frac{\partial F(x' + \frac{j}{n} \cdot (x - x'))}{\partial x_i} \cdot \frac{1}{m}, \tag{14}$$

where n denotes the amount of steps between the baseline and the input image used in the approximation. Choosing an appropriate value for n is important, as we need enough steps in order to get a good approximation for the integrated gradient. However increasing the amount of steps increases the computational expenses as well. For this thesis a value of $n = 64$ was used, which is in line with the recommended range of 20 to 300 given by Sundararajan et al. (2017).

The output of the IG method is referred to as (IG) attributions. IG has the problem, that it does not produce a connected importance area, but rather highlights single pixels and a high value in one pixel does not necessarily imply high pixel values in the surrounding area. Figure 24 depicts the IG attribution and showcases this issues.

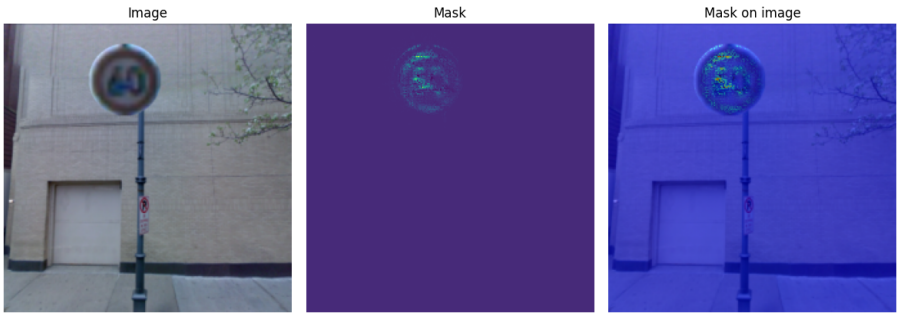

Figure 24: Example images showing the IG attributions. Original Image (L), IG attributions (M) and a visualization of the mask on top of the image using alpha blending (R). Notably the IG attributions are very scattered and thus do not offer a trivial way to extract a connected importance area.

4.5 XRAI

The XRAI method proposed by Kapishnikov et al. (2019) is a method based on IG which tries to address the issues IG has. The key problem with the IG attribution is, that pixels with different important values are scattered and do not create connected segments, making it harder to interpret the raw attributions.

XRAI approaches this problem by using precomputed segments for the input image produced with the Felzenszwalb's graph-based image segmentation algorithm (Felzenszwalb and Huttenlocher, 2004). Segments for the image are created for six different parameter settings and thus a so called oversegmentation is achieved. Each segment is

assigned an importance value by calculating the mean attribution value for the contained pixels and the highest rated segment is added to the heatmap with the corresponding weight. Afterwards pixels which have been added are excluded from the other segments. This ensures that each pixel can only influence a single segment and is important because of the oversegmentation. Segments are added and changed in this way, until 100% of the image is covered. The resulting mask is visualized in Figure 25.

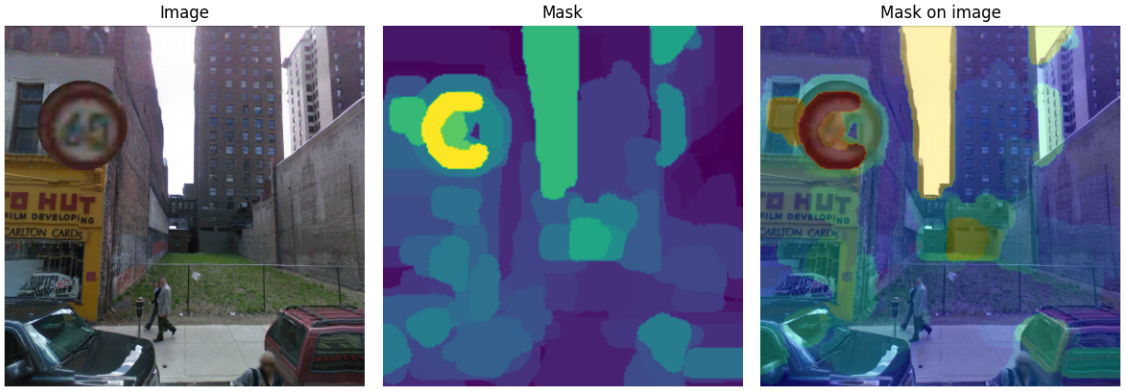

Figure 25: Example images for the XRAI Method. Original image (L), XRAI heatmap (M) and a visualization of the mask on top of the image using alpha blending (R).

Another problem Kapishnikov et al. (2019) identified is that IG can perform very poorly if many pixels in the input and the baseline are identical or very similar, because the value is likely to stay the same or show only small changes for different steps. In case of images where a black object is important, a black baseline as proposed in the IG paper can lead to very poor results. The proposed solution is to calculate the IG attributions for both a black and a white baseline and combine the results as displayed in Figure 26.

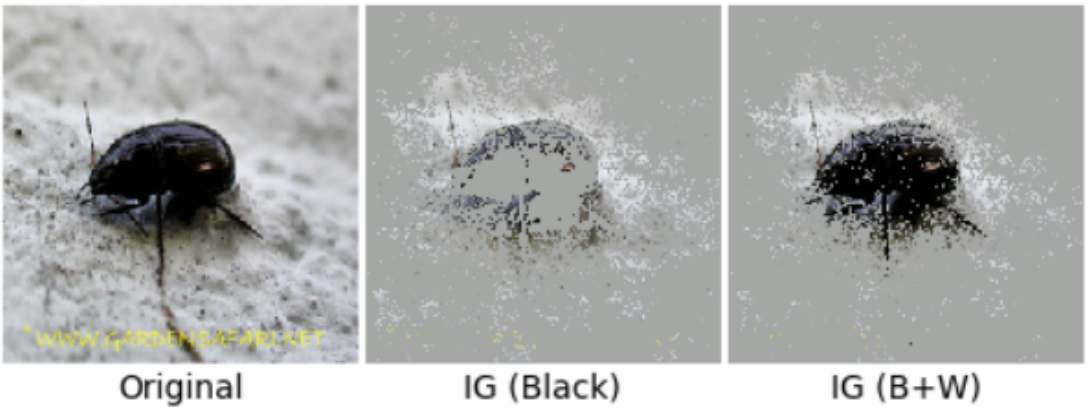

Figure 26: Image from Kapishnikov et al. (2019) showcasing the poor results provided by IG (M) if the pixel color of the baseline and the important part of the input image (L) are both black. To address this problem attributions are calculated for both a white and black baseline and combined (R).

4.6 IG-Fixpoints

As discussed previously a weakness of the IG attributions is, that they are scattered and hard to interpret by a human. While methods like XRAI can solve this problem to a certain degree, they introduce a new problem, which is the dependency on a good segmentation of the image.

In this thesis, a novel approach, "IG-Fixpoints" (IGF), is presented which addresses the problem of the scattered IG attributions, without depending on any additional segmentation algorithms. The method utilizes the idea of fixpoints in eyetracking, which are often used to create heatmaps in an attempt to understand the decision-making process by humans (Müller et al., 2023). In eyetracking heatmaps, the areas around so called fixpoints, which are pixels the user has fixated for a certain duration, are marked as important. The duration of the fixation functions as weight, where a longer fixation is considered more important. IGF creates heatmaps in the same way, by considering the top n pixel attributions as fixpoints where the attribution value takes the role of the duration as a weighting mechanism.

For this thesis the $n = 256$ most important pixels as given by IG were used as such fixpoints. The value was taken as a good tradeoff for having enough pixels impact the heatmap, while preventing a dilution effect which can happen if n is chosen too high. Further tests for different amount of used attribution pixels were concluded and are discussed in Section 6.4. The IG attributions were calculated the same way for both XRAI and IG using both a black and a white baseline as proposed in Section 4.5

The heatmap is derived from the attributions, by starting out with an empty 2D-heatmap matching the image shape and adding precomputed 2D Gaussian bells

$$f(x, y) = a \cdot e^{-s \cdot ((x-x_i)^2 + (y-y_i)^2)} \tag{15}$$

at the pixel positions (x_i, y_i) with the highest attribution values, where a is the weight given by the attribution value and s is a scaling factor. The radius of influence for the bell was limited to 64 pixels. Based on the scale parameter IGF can produce vastly different heatmaps as illustrated in Figure 28. Initially a scale parameter of $s = 0.001$ was chosen and used for the comparisons presented in Section 6.2. Further tests to optimize the scale parameter were conducted and are discussed in Section 6.4.

It is important to note, that both the parameter for the scale and the amount of used attribution values were not optimized prior to the evaluation with other XAI methods, in order to not reverse engineer the XAI method to do specifically well on the proposed evaluation method. This would have given the method an unfair advantage over other methods, as they might have performed better with different parameter settings as well, however testing different parameter settings for all of the compared methods was not possible within the scope of the thesis.

Figure 27 showcases the resulting output.

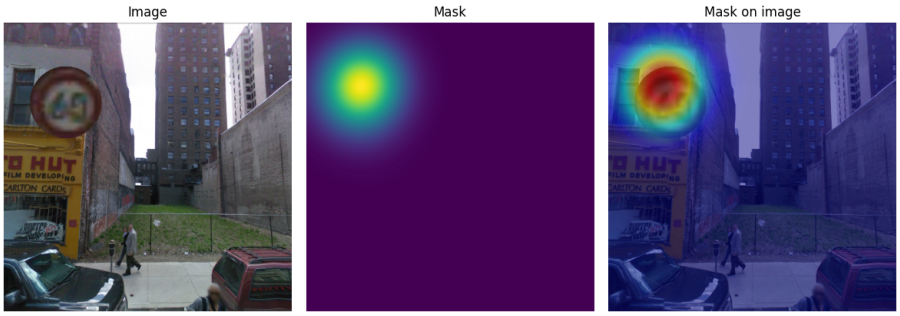

Figure 27: Example images showing the IGF method. Original Image (L), corresponding IGF mask (M) and a visualization of the mask on top of the image using alpha blending (R).

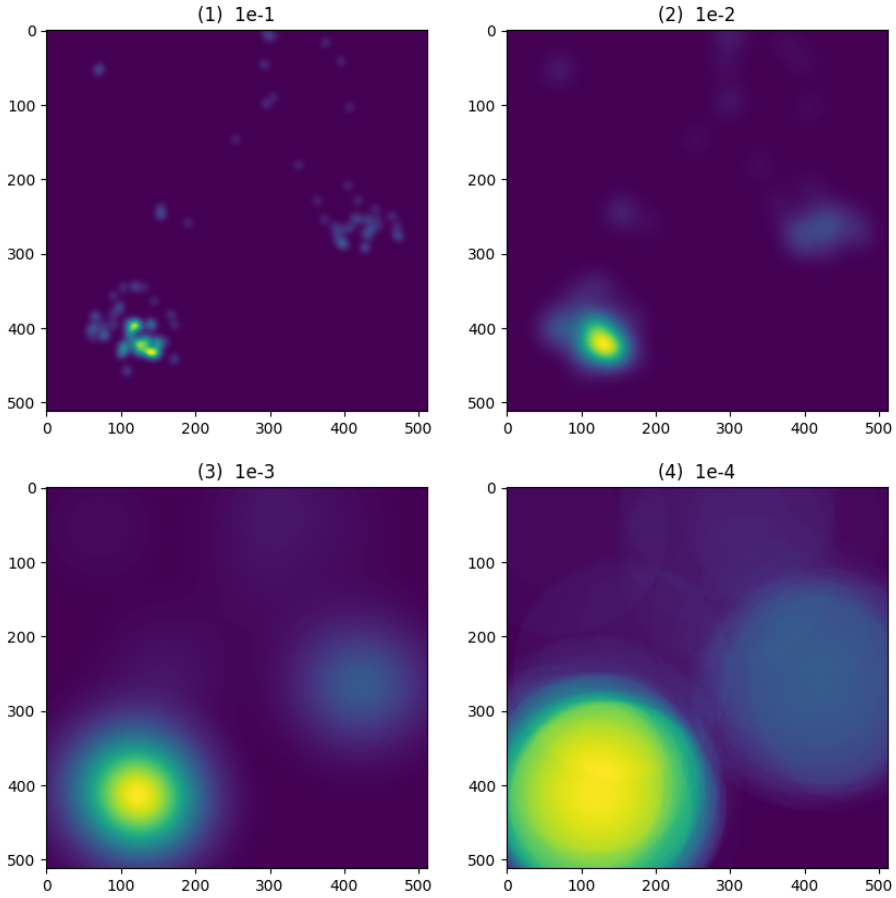

Figure 28: Example visualization of the impact the scale parameter s has on the impacted area on the heatmap around the pixel for each attribution. Small values (1), (2) might limit the impact too much and thus might not create proper segments, while larger values (4) might influence their surrounding too much. The chosen scale parameter (3) produces smooth segments without diluting the sphere of influence for a pixel attribution too much.

4.7 Heatmap Processing

During the production of the heatmaps for the XAI methods, the used test images were resized to a size of 224x224 rather than taking a crop, to ensure the whole traffic sign was available in the test image. Proportionally this slightly decreases the size of the traffic sign compared to the train and test images, which however did not impact the accuracy in a meaningful way.

This is important to know, since the presented methods produce outputs of different spatial dimensions compared to the images in the dataset (512x512) we want to apply the heatmaps on. Methods like Grad-CAM, and PRISM address this issue and have upsampling or extrapolation as intrinsic part of the method. LIME, XRAI and IGF on the other hand produce heatmaps which have the same spatial dimension as the inputs to the CNNs (224x224).

To match the these heatmaps to the spatial dimension (512x512) of the images in the dataset, bilinear upsampling is used. The choice for this upsampling method was made, because it is the same method which is used by the Grad-CAM algorithm.

As additional postprocessing step, a smoothing of the heatmaps was conducted. The reason for this is, that some of the XAI Methods do not allow for a clear distinction between the importance of individual pixels which is necessary in order to extract an area of a specific size, which is necessary for the evaluation of the XAI using the methods proposed in Section 5

Most notably LIME produces a binary heatmap. With every pixel having either a one or a zero value, extracting a specific amount of pixels depending on their importance would be impossible, since they all have the same importance value. Similar problems arise when evaluating the XRAI algorithm, where all pixels within a segment hold the same value. The problem can arise for both Grad-CAM and PRISM as well, since they sometimes only highlight a very small area of the image, making it impossible to differentiate between the pixels with a zero value when trying to extract a larger percentage of the image.

To counteract this problem all produced heatmaps are smoothed out. This is achieved by creating a smoothing mask and adding the mask to the heatmap.

The smoothing mask is obtained through average pooling. The kernel size used is 129, with 64 padding and a stride of one, where padded pixels are not taken into consideration for the results value. This means a pixel at position (0,0) would only average over the 65x65 area to its bottom-right, while a pixel in the center of the image would average over all pixels covered by the kernel. The size for the kernel was chosen in order to ensure, that as long as at least some part inside the street sign was highlighted, the smoothing mask would be able to cover the whole street sign.

The resulting gradualized map is divided by 256 to ensure it does not influence the existing order in the heatmaps. Instead the smoothing process should be considered as a tiebreaking mechanism, which does not change the importance the original XAI method has placed on regions or pixels. It rather only makes a difference when a decision is necessary which of the pixels with an identical value should be prioritized. This is especially important if a large amount of the image has zero pixels, to allow for all methods

to extract parts of the image that have the same size.

5 Evaluation of XAI-Methods using Neural Networks

The evaluation of the XAI methods is conducted by measuring how much accuracy is retained/lost, depending on whether the important pixels are progressively occluded or revealed.

5.1 Occlusion Accuracy

The first benchmarking method will be referred to as “Occlusion Accuracy” and measures the reduction in accuracy of the trained network on images where the important areas are progressively occluded starting with the most important pixels. In order to do this multiple new test datasets are created, depending on the occlusion level. Occlusion Accuracy Datasets in this thesis are defined by the occlusion level, the used CNN and the XAI method for the occlusion. An example for different occlusion levels is illustrated in Figure 29.

For occlusion, the original background when creating the image for the ATSDS dataset is used. This ensures more natural images and prevents an unnatural influence by the occluded area, which could happen if the occlusion was just done using for example black or white pixels. It is important to make an effort to minimize the influence these areas have on the results to obtain relevant results. For the ATSDS dataset for example one can imagine that using white pixels to fill the occluded areas could shift the decisionmaking of the network towards classes like "Give Way", which relatively speaking have more white pixels in the inserted streetsign.

For each XAI method and each network, Occlusion Datasets were created and evaluated up to 10% with 1% steps as a tradeoff between accuracy of results and computational expenses and storage demand when creating the Datasets. This was necessary to allow for multiple CNNs and XAI methods to be compared. A reduction in Accuracy is the desired result for a good XAI method when the important area is occluded.

Figure 29: Different Occlusion levels, starting from 1% occlusion (top left) to 10% occlusion (bottom right). As observable in the last two images, the streetsign might no longer remain in the image even with less than 10% occlusion level.

5.2 Revealing Accuracy

The second method, called "Revealing Accuracy", can be considered as counterpart to the Occlusion Accuracy.

Unlike the Occlusion Accuracy, a relatively strong increase in accuracy at lower percentages compared to other XAI methods would indicate that the heatmap produced by the XAI method is a better interpretation of the importance area for the CNN. Creation and evaluation of the Revealing Accuracy Datasets are done the same way (1% steps until 10%). Example images showcasing different percentages for the Revealing Accuracy Datasets are depicted in Figure 30

For both the Revealing Accuracy and the Occlusion Accuracy, the images are fed back into the same CNN which was used for the production of the XAI heatmaps.

Figure 30: Different revelation levels, starting from 1% revelation (top left) to 10% revelation (bottom right). As observed with the occluded images, the full streetsign is visible at less than 10%.

5.3 Statistical Evaluation

Given the knowledge of the underlying ground truth of the important areas for the images in the ATSDS dataset, it is an obvious choice to compare the results of statistical measurements for similarity and to see if a statistical evaluation might offer similar results as the proposed evaluation methods or if there are big differences. Four different popular methods to measure similarity are presented based on Freedman et al. (2007) to compare the provided heatmaps and the ground truth.

5.3.1 Pearson Correlation Coefficient

The Pearson correlation coefficient (PCC) is usually used to measures the linear correlation between two datasets. The PCC for two single channel images X, Y is defined by:

$$\rho(X,Y) = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n}(x_{ij}-\bar{x})(y_{ij}-\bar{y})}{\sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}(x_{ij}-\bar{x})^2}\sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}(y_{ij}-\bar{y})^2}}, \tag{16}$$

where x_{ij} and y_{ij} describe the pixels, $\bar{x}$ and $\bar{y}$ are the image means, and m and n correspond to the height and width of the images.

The value for ρ can range between $[-1, 1]$ with a higher value implying a higher similarity between heatmap and ground truth in our context.

5.3.2 Cosine Similarity

Another similar method is the Cosine Similarity given by

$$\mathbf{S}_{\cos}(\mathrm{X},\mathrm{Y}) = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n}x_{ij}y_{ij}}{\sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}x_{ij}^2}\sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}y_{ij}^2}}, \tag{17}$$

where the same connotations are used as for ρ and similarly the values range between $[-1, 1]$ with a higher value indicating a higher similarity.

5.3.3 Kullback-Leibler Divergence

The Kullback-Leibler (KL) divergence is a measurement of difference between distributions, which we can use to measure how well the heatmap can be used to represent the ground truth. A high similarity is indicated by a low divergence value. The formula for a 2D-image is given by

$$\mathbf{D}_{\mathrm{KL}}(X||Y) = \sum_{i=1}^{m}\sum_{j=1}^{n}x_{ij}\log\frac{x_{ij}}{y_{ij}}, \tag{18}$$

where X is the ground truth and Y is the heatmap and x_{ij} and y_{ij} are the respective pixels.

5.3.4 True Positive Rate

The True Positive Rate (TPR) is measured for different percentages of the heatmap by comparing the ground truth X with a binary mask extracted from the heatmap. The TPR is given by

$$\mathrm{TPR}(X,Y^p) = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n}(x_{ij}=1)\wedge(y_{ij}^p=1)}{\sum_{i=1}^{m}\sum_{j=1}^{n}(x_{ij}=1)}, \tag{19}$$

where Y^p is a binary map where all values are zero, unless the pixel value is in the top $p \in [1, 10]$ percent of the intensity values in the heatmap.

6 Results

6.1 Network Training Results

The networks were trained as described in Section 3.2.5 and Appendix C. Overall training was stable throughout the training for all CNNs. The train and test loss which is depicted in Figure 31 indicates a generally smooth training process with no significant overfitting. While the test loss for the ResNet50 throughout the training differed from the train loss, it started to smooth out after 150 epochs.

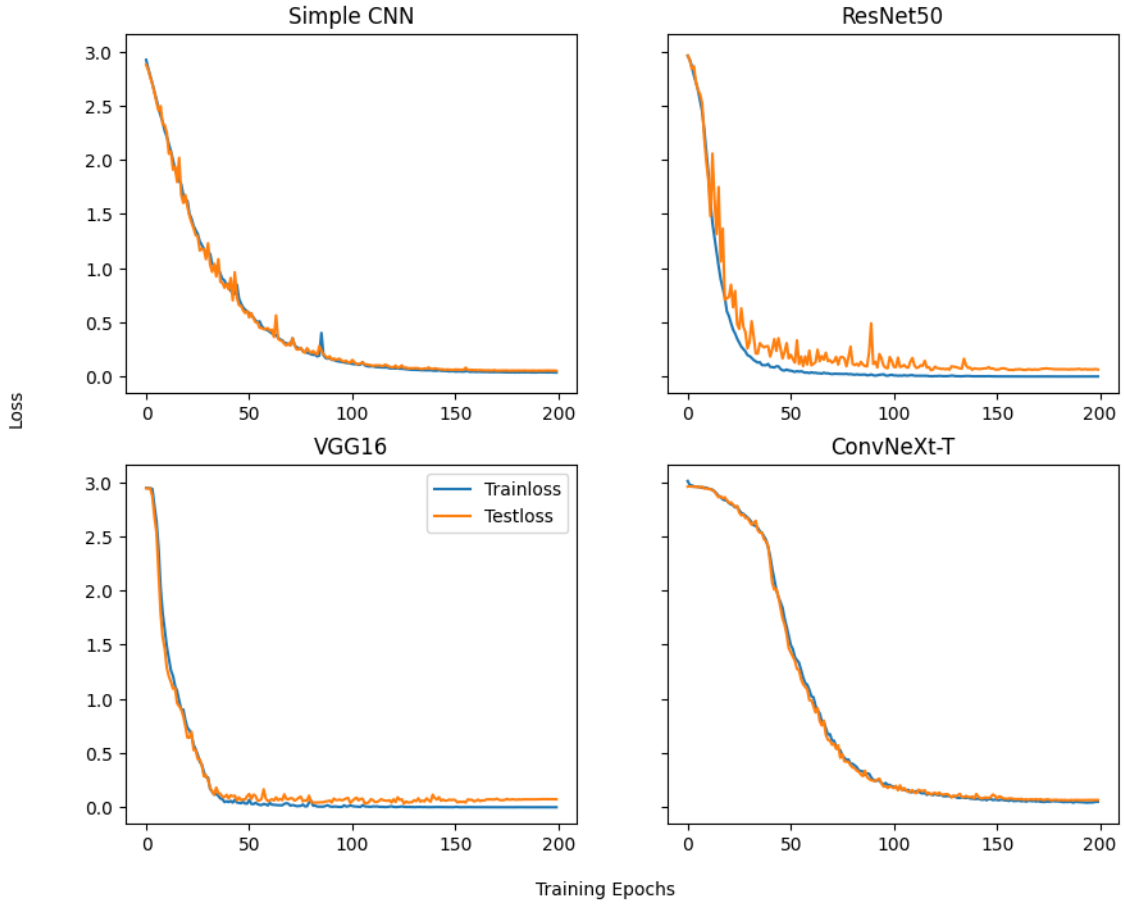

Figure 31: Train and test loss for the different CNNs over the 200 epoch training period.

The achieved test accuracy after 200 epochs was 98.8% for the Simple CNN, the VGG16 achieved 99.2%, the ResNet50 achieved 98.5% and the ConvNeXt performed worst with 97.6% accuracy. The illustration of the accuracy throughout the training process in Figure 32 shows that the training was smooth and the CNNs generalized well to the testset. Similar to the observed differences in train and test loss, the ResNet50 showed worse test accuracy throughout the training, but stabilized after 150 epochs. Notably the ConvNeXt-T model unlike the other models did not achieve a 100% train accuracy. This could be attributed to the usage of many techniques which focus on generalization (Section 3.2.4).

Another reason could be, that the network was designed for significantly longer training. According to Zhuang Liu et al. (2022) they trained their network six times longer (more epochs) compared to the ResNet50 which the ConvNeXt-T is based on, indicating that the network might have benefited from a larger amount of epochs. It should be noted that as discussed in Appendix C the ConvNeXt-T was the only model to require an increased learning rate in order to train in a timely manner which could be another contributing factor.

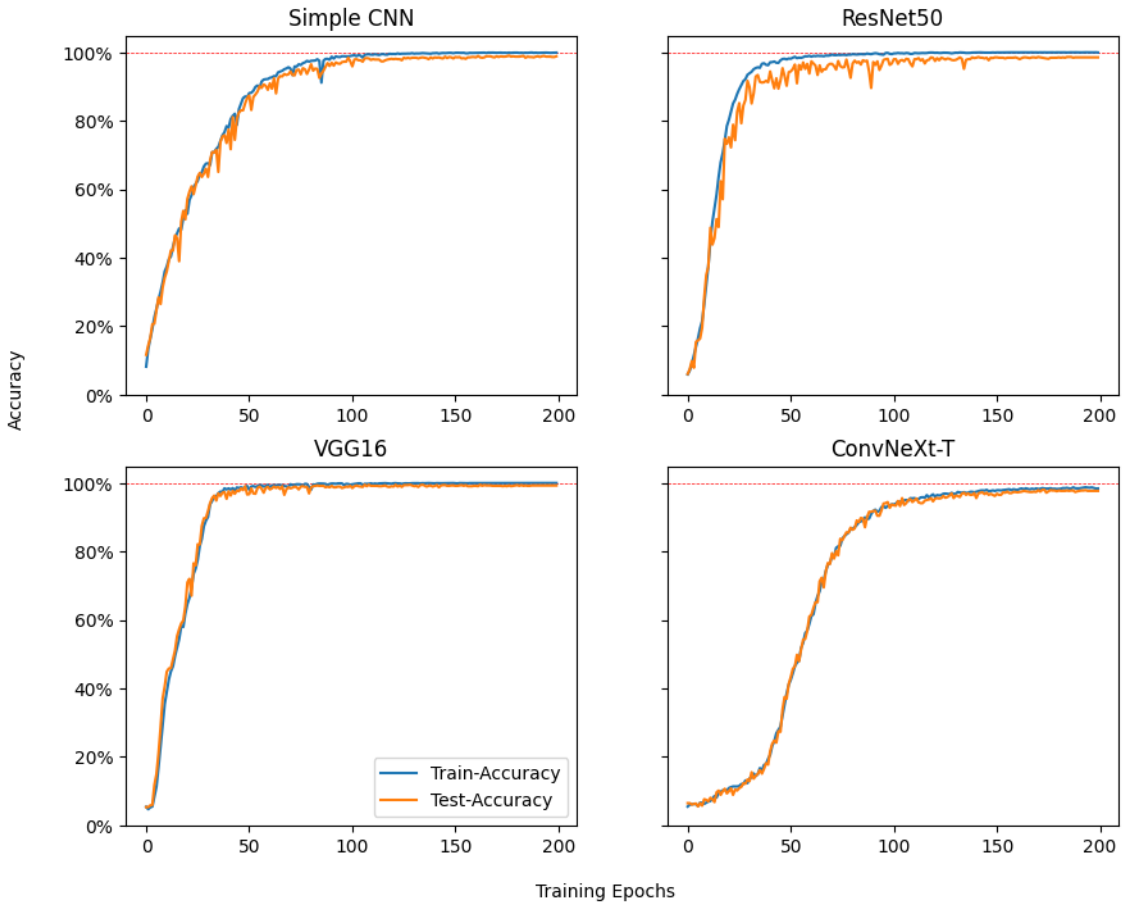

Figure 32: Train and test accuracy for the different CNNs over the 200 epoch training period. A red dashed line is drawn at 100%.

6.2 Occlusion and Revelation Accuracy

Results for the Occlusion and Revelation Accuracy produced comparable results, especially with regards to the ranking of the methods for each CNN. Therefore the Revelation Accuracy will be discussed in details, while the results for the Occlusion Accuracy for comparisons can be found in Figure 40 with exact values being available in Appendix E

Simple CNN

Despite having a very basic structure which could create the expectations that methods should work better compared to very complex networks, the Simple CNN posed a challenge to multiple XAI methods. Only Grad-CAM and LIME worked really well. The IG based methods performed significantly worse, but one could argue that the proposed IGF method slightly outperformed XRAI. This indicates that there was an underlying problem with the IG attributions. PRISM performed very bad, producing heatmaps unable to highlight the traffic signs. Details are given in Table 2 and the results are visualized in Figure 33. All Tables

Table 1: Revelation Accuracy - Simple CNN

%	Grad-CAM	IGF	LIME	PRISM	XRAI
0	0.06	0.06	0.06	0.06	0.06
1	**0.19**	0.08	0.13	0.06	0.09
2	**0.61**	0.27	0.37	0.07	0.24
3	**0.89**	0.40	0.59	0.07	0.37
4	**0.96**	0.42	0.74	0.08	0.43
5	**0.98**	0.44	0.84	0.09	0.44
6	**0.98**	0.46	0.90	0.10	0.46
7	**0.98**	0.48	0.92	0.11	0.48
8	**0.98**	0.50	0.94	0.13	0.49
9	**0.98**	0.51	0.96	0.13	0.50
10	**0.98**	0.53	0.97	0.14	0.52

Table 2: Detailed Revelation Accuracy data. Each line corresponds to the achieved network achieved for a corresponding percentage of the image being revealed. For readability the best method is highlighted in bold, while the worst method is underlined.

VGG

All XAI methods were able to perform very well for the VGG16 architecutre. With only 4% of the important area being revealed, 4 out of 5 methods already achieved a 90% accuracy. While LIME was trailing slightly behind the other methods, it can still be considered as a relatively good result. One reason for this could be, that many of the XAI methods were designed and tested using VGG, which as discussed in Section 3.2.2 remains a very popular network architectures due to its simplicity. The IGF method proposed in this thesis showed especially promising results, when only a small part of the image was revealed. This could hint at either the method being very precise or having better differentiation between important subareas of the broader importance area. This could be attributed to the fact it operates on the spatial dimension of the input image compared to e.g. Grad-CAM which operates using the outputs of the final convolutional layer which usually has a much smaller spatial dimension. Details are given in Table 3 and the results are visualized in Figure 34.

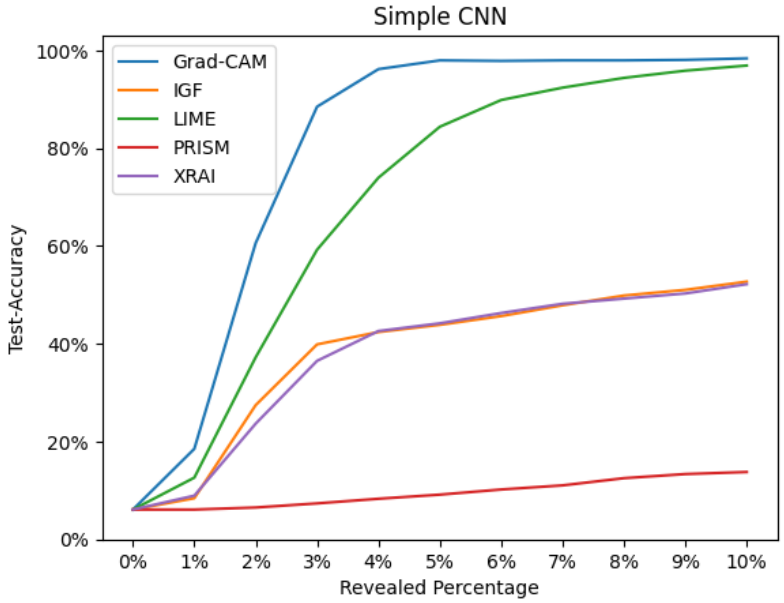

Figure 33: Revelation Accuracy for the Simple CNN showcasing the achieved accuracy for on the test dataset for revelation levels from 0 to 10%.

Table 3: Revelation Accuracy - VGG16

%	Grad-CAM	IGF	LIME	PRISM	XRAI
0	0.05	0.05	0.05	0.05	0.05
1	0.12	0.15	0.09	0.10	0.12
2	0.63	**0.72**	0.30	0.50	0.52
3	0.96	**0.96**	0.50	0.83	0.87
4	**0.98**	0.97	0.66	0.91	0.95
5	**0.99**	0.98	0.79	0.93	0.97
6	**0.99**	**0.99**	0.86	0.95	0.98
7	**0.99**	**0.99**	0.91	0.96	0.98
8	**0.99**	**0.99**	0.94	0.97	**0.99**
9	**0.99**	**0.99**	0.95	0.98	**0.99**
10	**0.99**	**0.99**	0.96	0.98	**0.99**

ResNet50

For the Resnet50 Architecture Grad-CAM and LIME showed promosing results. Interestingly PRISM, while being very bad for low percentages, continuously improved. This indicates that the method was likely not that far off the important area and started to include it in the images at higher percentages. Details are given in Table 4 and the results are visualized in Figure 35.

Notably the IG methods had a decent start below 3%, but showed hardly any increase

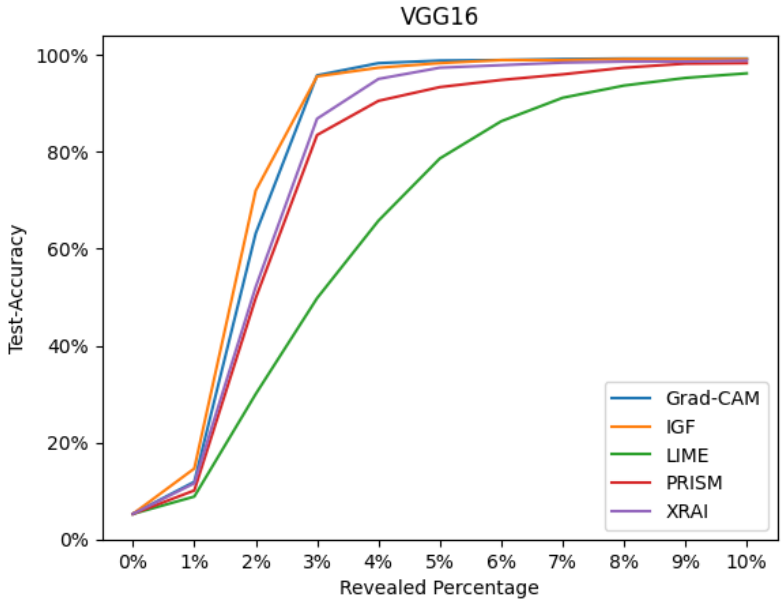

Figure 34: Revelation Accuracy for the VGG16 CNN. Revelation Accuracy for the Simple CNN showcasing the achieved accuracy for on the test dataset for revelation levels from 0 to 10%.

in accuracy from 4% to 10% resulting in a poor overall performance. When looking at heatmaps for classes with poor performance, it was notable that while the traffic signs were often highlighted, other high attribution values detracted the methods. Examples for this are given in Figure 36

Table 4: Revelation Accuracy - Resnet50

%	Grad-CAM	IGF	LIME	PRISM	XRAI
0	0.05	0.05	0.05	0.05	0.05
1	**0.10**	0.07	0.08	0.08	0.07
2	**0.32**	0.17	0.21	0.15	0.15
3	**0.63**	0.34	0.41	0.21	0.28
4	**0.82**	0.39	0.57	0.29	0.33
5	**0.87**	0.42	0.67	0.39	0.36
6	**0.92**	0.44	0.76	0.47	0.38
7	**0.95**	0.47	0.81	0.52	0.39
8	**0.96**	0.48	0.86	0.58	0.40
9	**0.96**	0.50	0.89	0.63	0.42
10	**0.97**	0.51	0.9	0.67	0.43

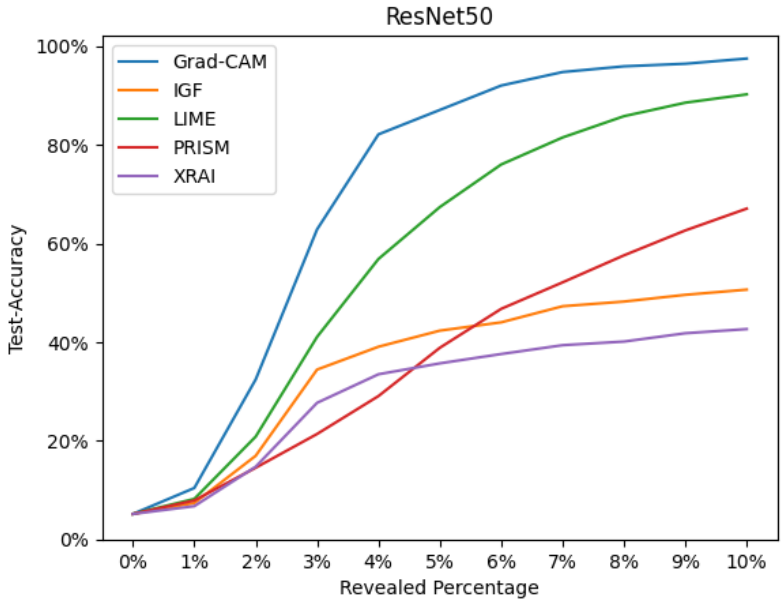

Figure 35: Revelation Accuracy for the Resnet50. Revelation Accuracy for the Simple CNN showcasing the achieved accuracy for on the test dataset for revelation levels from 0 to 10%.

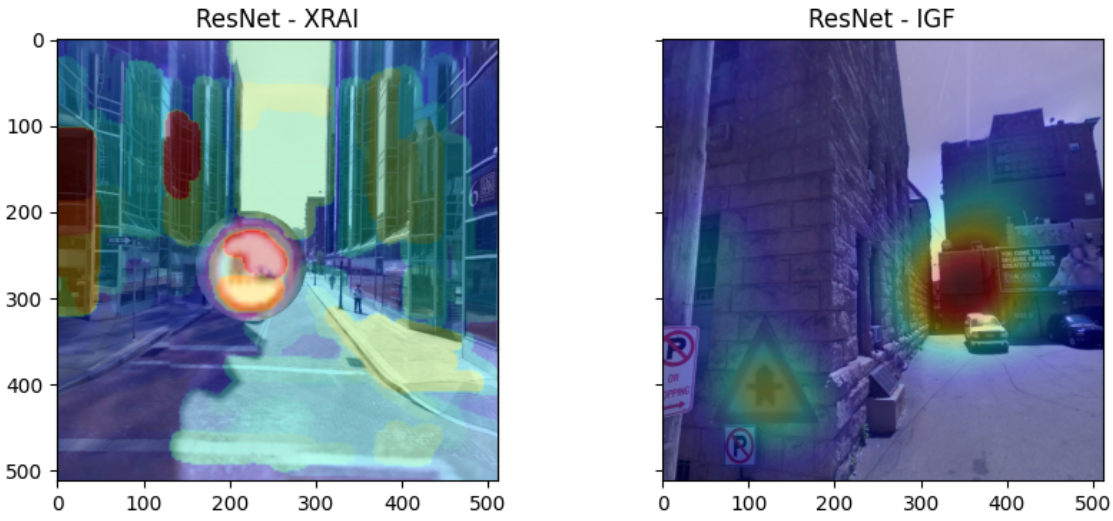

Figure 36: Example images showcasing a problem the IG methods often faced on the ResNet architecture. While the traffic sign is highlighted, segments on the left of the sign (XRAI example) or a area right to it (IGF) were highlighted stronger, which caused the method to perform poorly.

ConvNeXt

Since Grad-CAM had performed really well for the other CNNs, it was surprising, that the performance on the ConvNeXt network was very bad, showing that advances in CNN architectures might need new XAI methods to match them. The same can be said about PRISM. For both methods the increase in accuracy seems to be caused by chance rather than the quality of the heatmaps. On the other hand, both IG based methods work very well on ConvNeXt, despite their below average performance on the Simple CNN and ResNet. Details are given in Table 5 and the results are visualized in Figure 37. Figure 38 showcases the accuracy for certain classes barely improved when more of the image was revealed.

The poor performance of Grad-CAM on the ConvNeXt-T was especially surprising after it had shown great results on the ResNet50 it is based on (Section 3.2.4). While being based on the ResNet50, the ConvNeXt-T implemented a large number of changes not only to the network design, but the training procedure as well, which ideally would all be investigated in regards to their impact on the interpretability of the model. Given the time limitations of this thesis a few selected changes which intuitively seemed to be the most likely culprits for the poor performance of the Grad-CAM method were investigated in Section 6.5.

Table 5: Revelation Accuracy - ConvNeXt

%	Grad-CAM	IGF	LIME	PRISM	XRAI
0	0.05	0.05	0.05	0.05	0.05
1	0.05	**0.08**	0.07	0.05	**0.08**
2	0.06	**0.30**	0.16	0.07	**0.30**
3	0.07	**0.68**	0.36	0.08	0.65
4	0.09	**0.83**	0.52	0.09	0.82
5	0.11	**0.89**	0.68	0.11	0.87
6	0.14	**0.91**	0.76	0.12	0.90
7	0.16	**0.93**	0.84	0.14	0.91
8	0.18	**0.94**	0.88	0.15	0.91
9	0.20	**0.95**	0.91	0.17	0.93
10	0.22	**0.96**	0.92	0.19	0.93

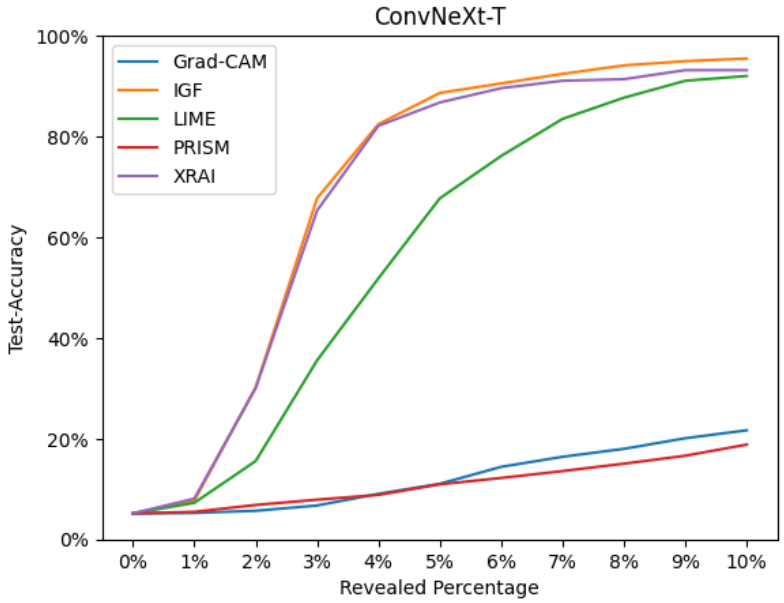

Figure 37: Revelation Accuracy for the ConvNeXt. Revelation Accuracy for the Simple CNN showcasing the achieved accuracy for on the test dataset for revelation levels from 0 to 10%.

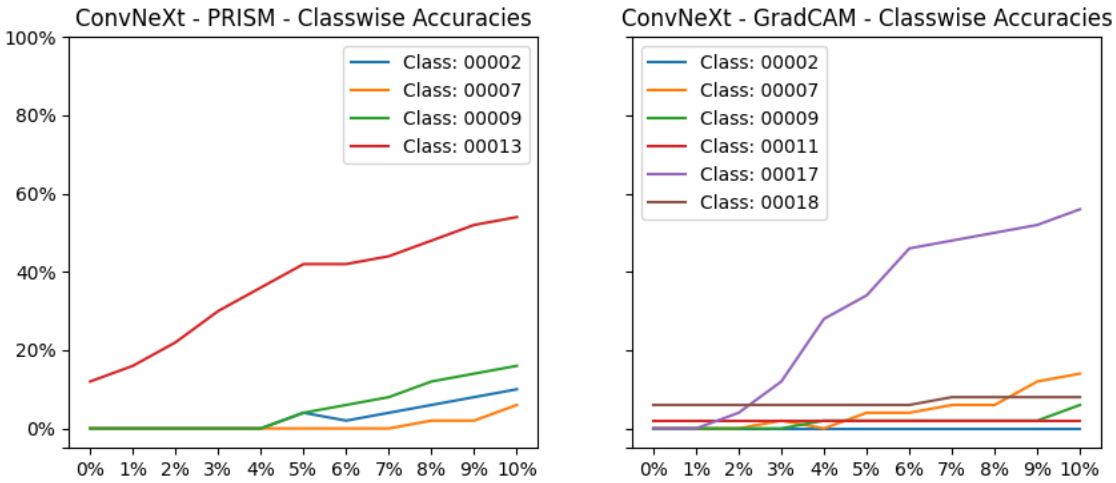

Figure 38: Classwise accuracies for PRISM (L) and Grad-CAM (R) showcasing the accuracy for selected poorly performing classes complemented by the best performing class. For PRISM some classes showed no accuracy improvement below 5% revelation and even the best performing class "00013" achieved only around 50% accuracy at 10% revelation. Grad-CAM had similar issues, with some classes ("00002","00011") not showing any improvement to the accuracy as the revelation ratio was increased. Similar to PRISM, the best performing class ("00017") stayed below 60% accuracy even at a 10% revelation level.

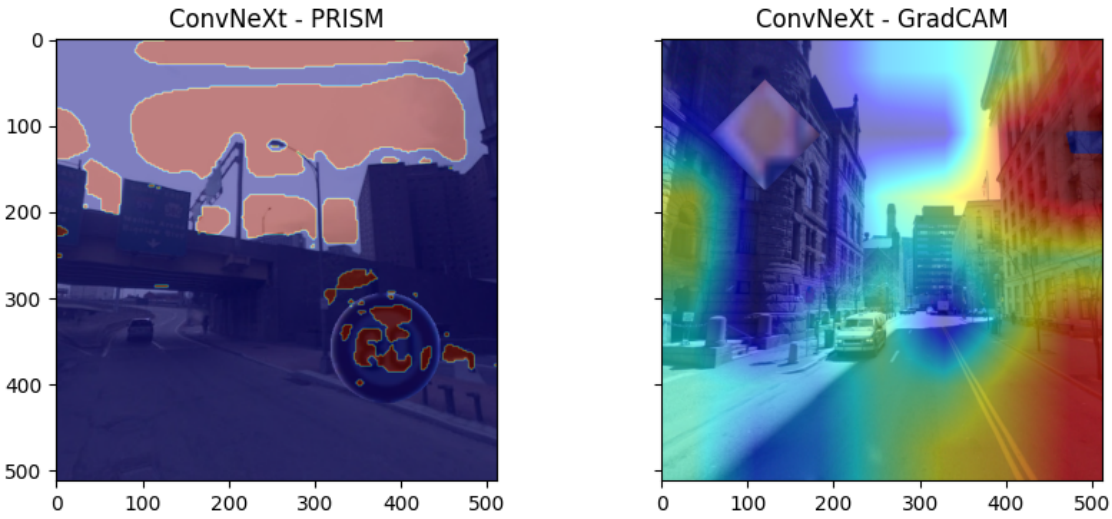

Figure 39: Heatmaps produced by PRISM or Grad-CAM failing to capture the important area. While PRISM tended to highlight small parts of the important area on the image, other areas dominated the heatmaps. GradCAM heatmaps often did not follow any comprehendable pattern and if the traffic sign was highlighted, it seemed to be mostly by chance.

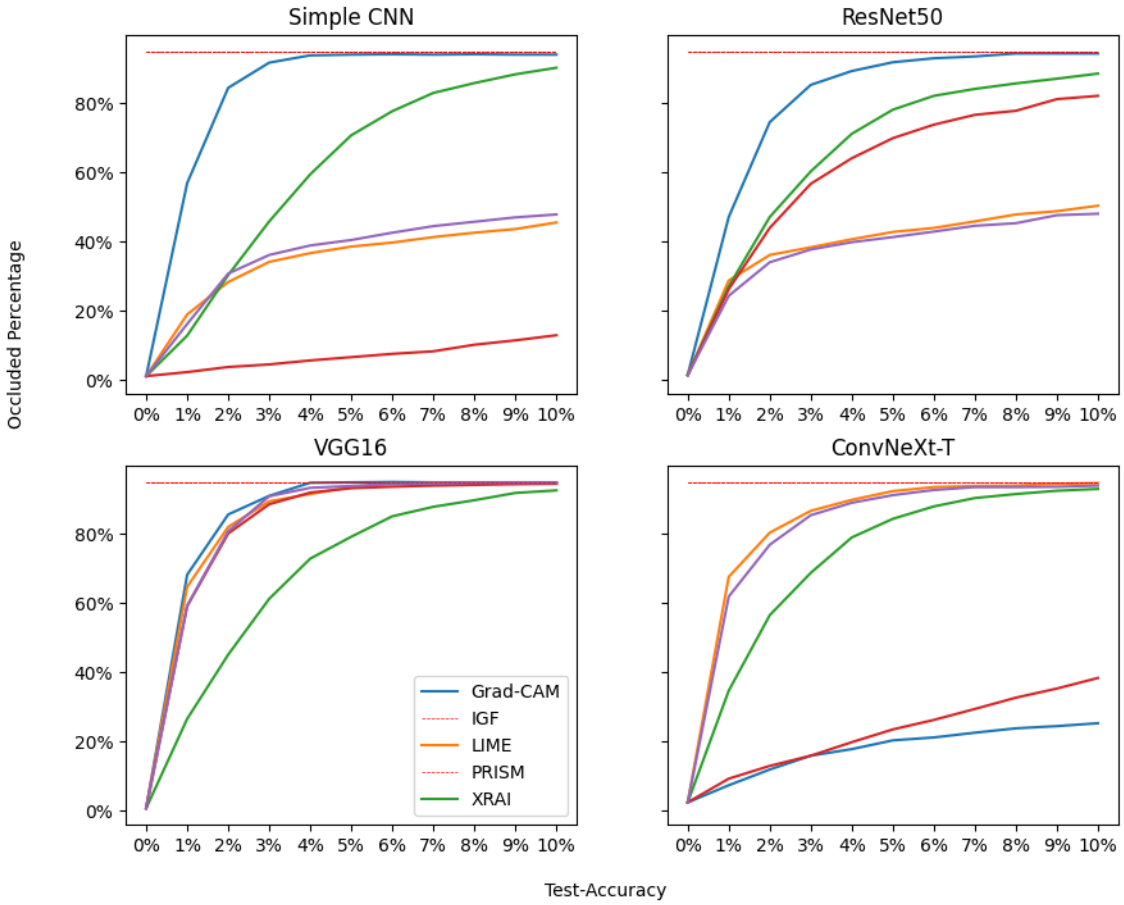

Figure 40: Results for the Occlusion Accuracy. Instead of depicting the Occlusion Accuracy itself, the plots depict the decrease in accuracy for a given occlusion level assuming a 100% accuracy for the fully visible image. A vertical line is drawn at 94,7%, which corresponds to the network making a random guess. Only the PRISM method really showed notable differences compared to the Revelation Accuracy. PRISM performed significantly better on the ResNet50, nearly matching LIME. For ConvNeXt the PRISM results were still subpar, but notably better compared to the results of the Revelation Accuracy.

6.3 Statistical Evaluation of XAI Methods

Pearson Correlation Coefficient

The results for the Pearson Coefficient (Section 5.3.1) depicted in Table 6 strongly indicate, that within the presented methods, no single one can trivially be considered as superior. XRAI was the only method which had the highest correlation to the ground truth for multiple CNNs, but at the same time had the worst result for the ResNet. Most notably there is a big variation in correlations throughout methods and networks, with only LIME having very comparable results for all CNNs.

Table 6: Results - Pearson Correlation Coefficient

	Grad-CAM	**IGF**	**LIME**	**PRISM**	**XRAI**
Simple CNN	**0.770**	0.335	0.561	0.069	0.310
VGG	0.747	0.770	0.515	0.761	**0.811**
ResNet	0.439	0.319	**0.475**	0.357	0.240
ConvNext	0.034	0.748	0.541	0.139	**0.773**

Cosine Similarity

The results for the cosine similarity (Section 5.3.2) are very comparable to those from the Pearson Coefficient. Rankings remained mostly the same (Table 7).

Table 7: Results - Cosine Similarity

	Grad-CAM	**IGF**	**LIME**	**PRISM**	**XRAI**
Simple CNN	**0.775**	0.392	0.577	0.195	0.364
VGG	0.746	0.779	0.535	0.772	**0.820**
ResNet	0.453	0.379	**0.498**	0.387	0.312
ConvNext	0.144	0.757	0.558	0.218	**0.771**

Kullback–Leibler Divergence

The KL divergence (Section 5.3.3) showed significantly different results compared to the other measurements (Table 8). Most notably LIME always has a high divergence, likely caused by the fact that Lime produces a binary mask which in many cases only covers a small fraction of the image. The results strongly indicate that if only a statistical approach is taken, the results can strongly depend on characteristics of the produced heatmap.

The results for the KL Divergence highlight the problem, that while a theoretically known ground truth allows for such statistical comparisons, we cannot be sure, that all of the ground truth was actually important to the CNNs decision. For example in theory the numbers on the speed limit signs alone could be enough for the network to make a decision, in some cases like a "Speed 80" shield even just the "8" might be enough. These

statistical comparisons are thus a good sanity check, without necessarily being able to provide conclusive evidence to the capability of an XAI method.

Table 8: Results - Kullback–Leibler Divergence

	Grad-CAM	**IGF**	**LIME**	**PRISM**	**XRAI**
Simple CNN	**0.969**	2.900	2.452	4.338	2.552
VGG	1.112	0.940	3.257	1.023	**0.658**
ResNet	**2.006**	2.849	3.875	3.565	2.774
ConvNext	10.86	**0.917**	2.69	4.872	1.075

True Positive Rate

The results for the TPR (Section 5.3.4) are depicted in Figure 41 and are most similar to the Occlusion/Revelation Accuracy results, however the Occlusion/Revelation approaches have the benefit, that a differentiation can be made for what parts inside the ground truth are actually more important. The detailed numbers can be found in Appendix F. In general the TPR did have some minor disagreements with the Occlusion/Revelation Accuracies, but the bigger picture if a method was doing very good or very bad was the same. Notably one could argue that IGF performed better compared to XRAI for this metric, especially at higher percentages for the ResNet50.

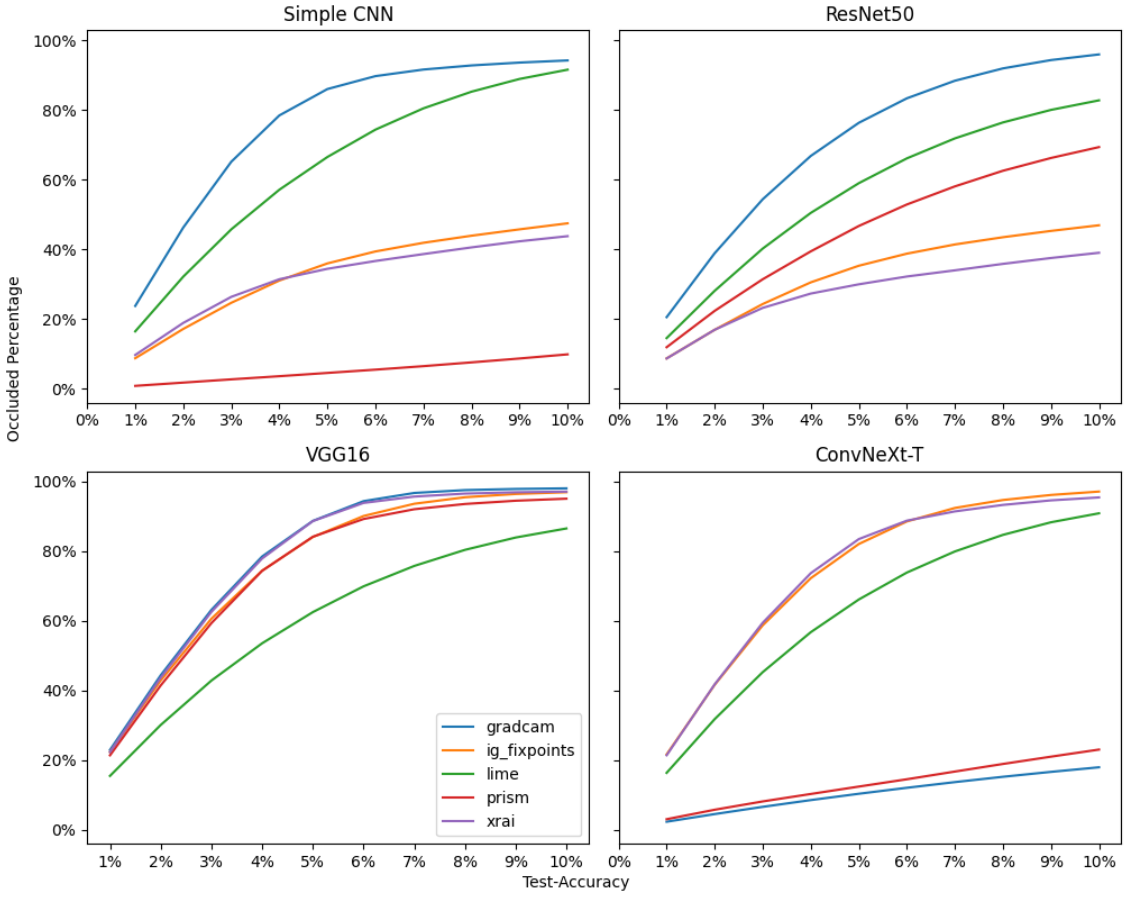

Figure 41: TPR results for different CNNs

6.4 Parameter Optimization for IGF

As discussed in Section 4.6 the introduced IGF method produces very different heatmaps depending on the amount of fixpoints used and the choice for the scale parameter for the 2-D gaussian bell. With the goal of improving the method, different parameter settings were tested. Parameter testing was conducted using the Simple CNN, since it was one of the networks the method was not able to perform to the level of other methods. Figure 42 depicts the Revelation Accuracy of the method with different values for the scale parameter. The number of attribution values used as fixpoints was 128.

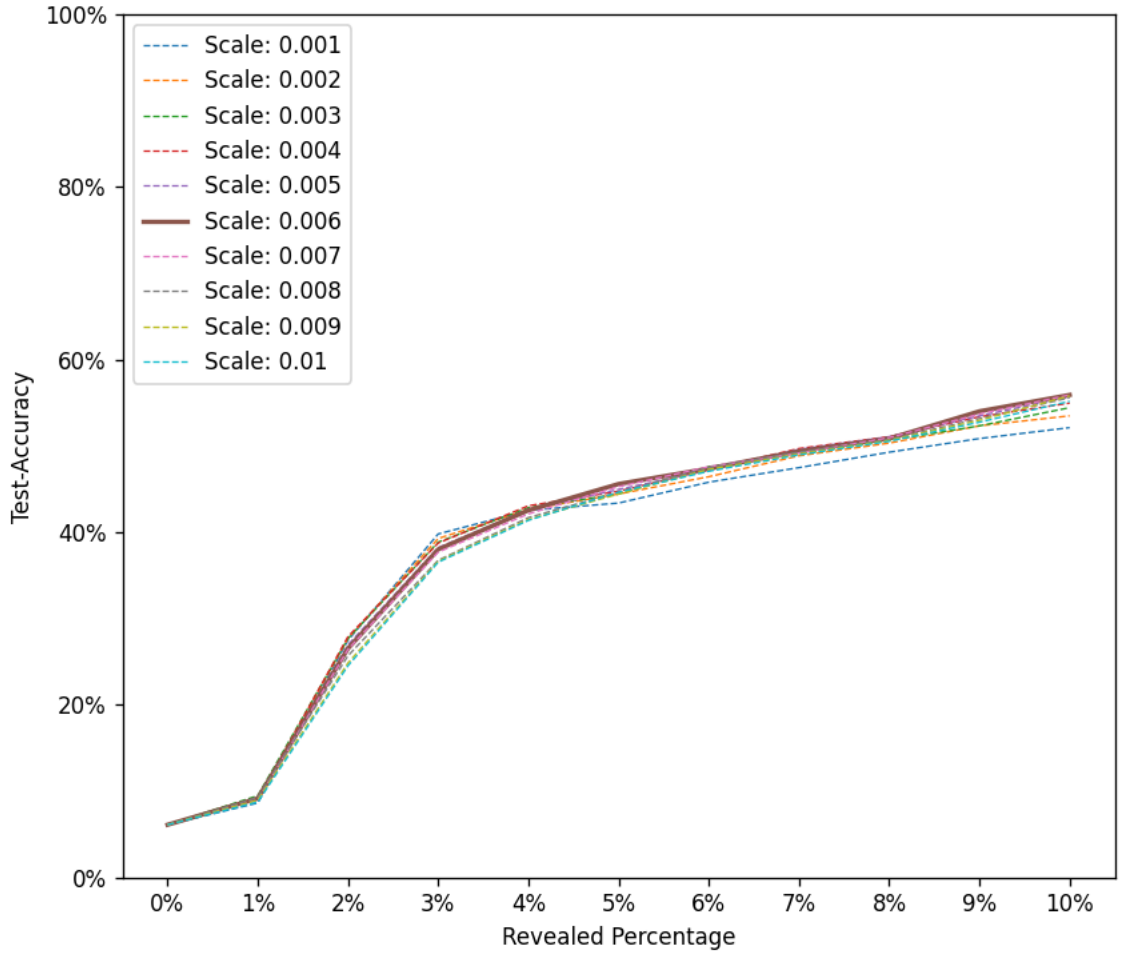

Figure 42: Revelation Accuracy of the IGF Method on the Simple CNN for different scale parameters.

The observable improvements were relatively small, but the Revelation Accuracy did increase by a few percentage points over the chosen scale parameter for the results presented in Section 6.2. As displayed in the Figure 28 the scale values had different strengths and weaknesses. For further testing a scale value of 0.006 was used. While it did not achieve the highest accuracy for all revelation percentages, as it showed the best

results at higher percentages. Using 0.006 as value for the scale parameter, further tests were conducted to improve the accuracy by changing the amount of attributions used as fixpoints for the heatmap. As depicted in Figure 43 a small amount of used fixpoints seemed preferable over a larger number.

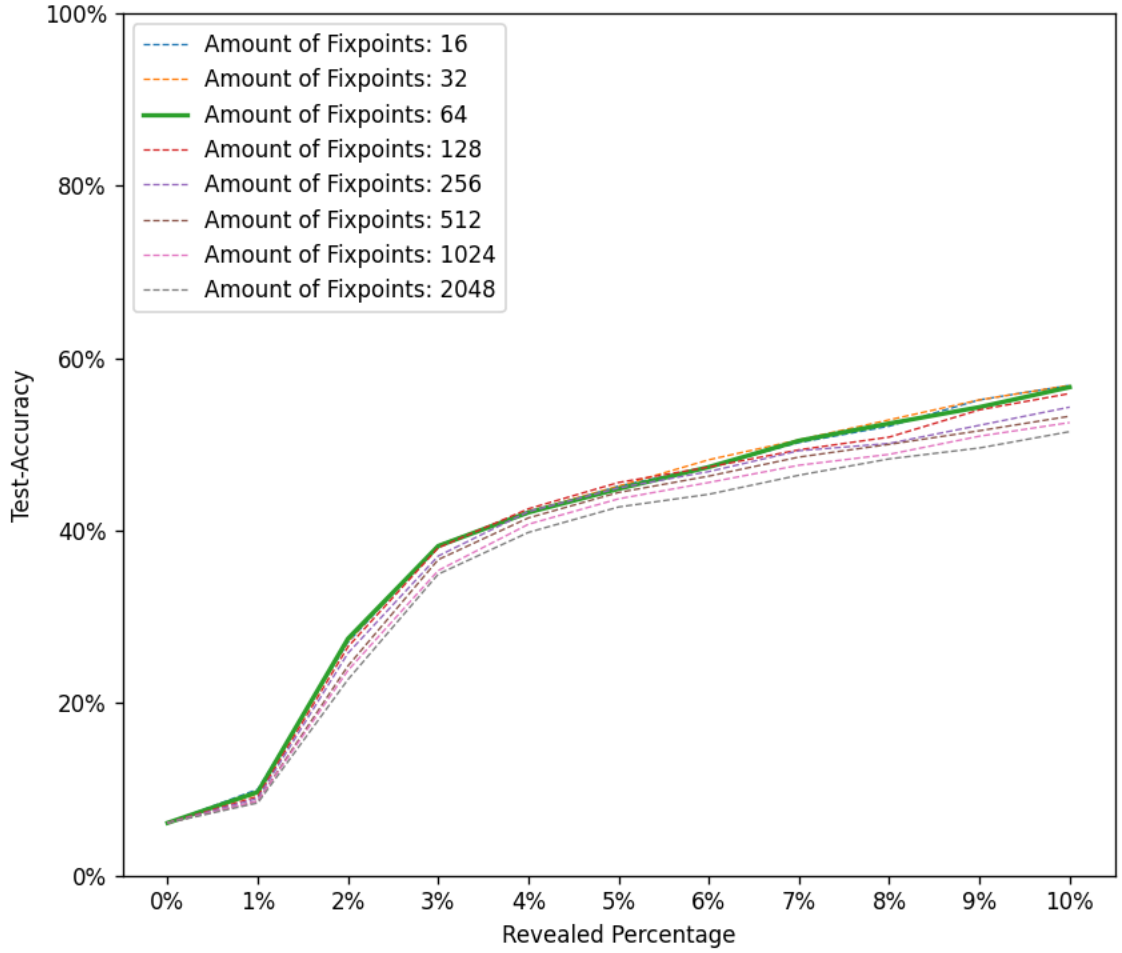

Figure 43: Revelation Accuracy of the IGF Method on the Simple CNN for different scale parameters.

The new parameter settings were compared with the initially chosen settings, but as depicted in Figure 44 the changed parameters did not increase the final Revelation Accuracy for the other CNNs and especially for the ConvNeXt-T and ResNet50 had a worse performance at lower percentages compared the parameters initially proposed in Section 4.6.

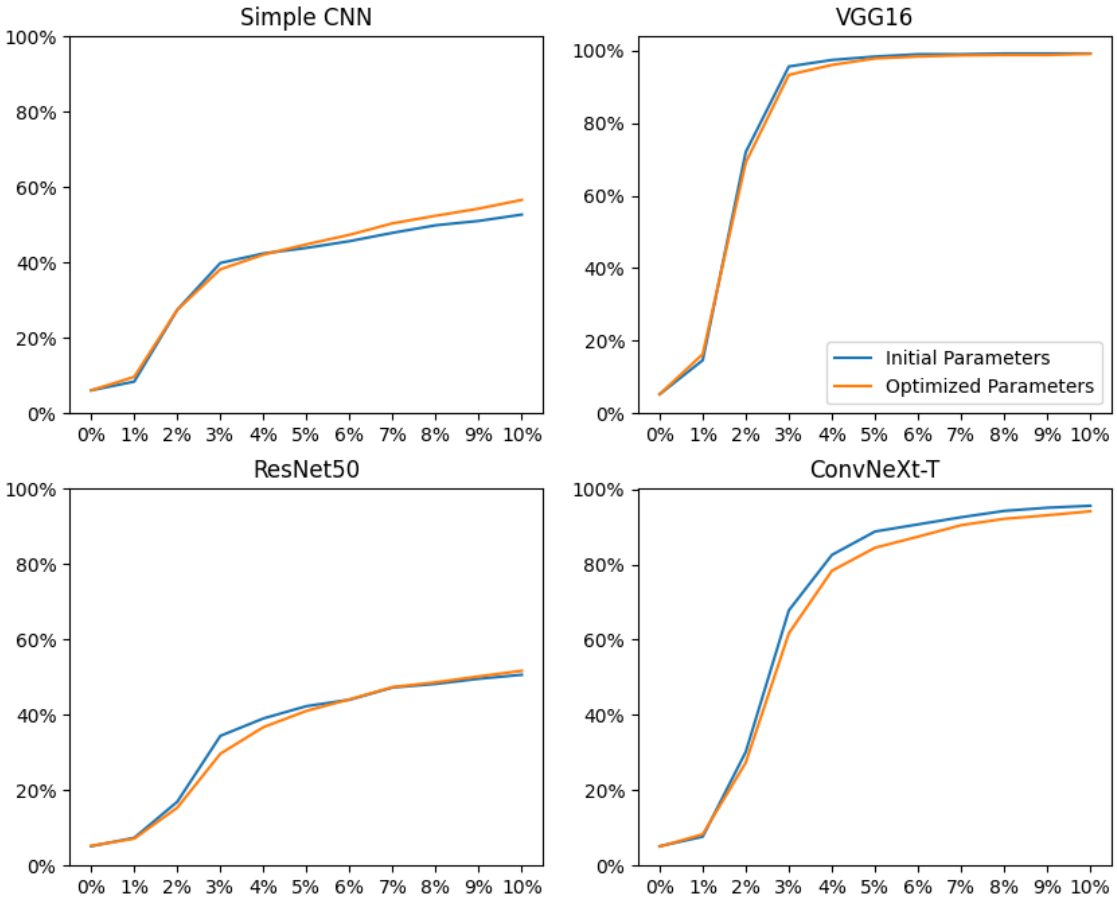

Figure 44: Revelation Accuracy of the IGF Method on the Simple CNN for different scale parameters.

6.5 Grad-CAM on Changed ConvNeXt

As discussed in Section 6.2 the results of Grad-CAM were very good on most networks, including the ResNet50, which the ConvNeXt-T is based upon.

Therefore it came as a surprise that GradCAM showed a very bad performance on this model. A plausible explanation for a decreased performance of Grad-CAM on ConvNeXt-T is that the output of the last convolutional layer, which is the basis Grad-CAM works with, has a smaller spatial dimension compared to the other networks. While this could explain a decreased preciseness and thus performance of the method, it cannot explain that Grad-CAM seemed to be completely unable to locate the important area.

An initial hypothesis was that the split in channel and spatial mixing the ConvNeXt block achieved by using a depthwise convolution followed by 1x1 kernels, could cause the information about the location of features in the input image to get lost, however when replacing three VGG layers with depthwise convolutions, it still showed good results for the GradCAM message, indicating that the depthwise convolutions were not causing the problem.

In an attempt to find the cause of the deteriorated performance of GradCAM on the ConvNeXt-T model, slightly changed versions of ConvNeXt-T were trained. Afterwards heatmaps were created for them using Grad-CAM and the Revelation Accuracy was calculated. Since not all changed models achieved the same accuracy, the Revelation Accuracy was calculated using the initially trained ConvNeXt-T to have a more equal base of comparison.

The results are depicted in Figure 45. Removing stochastic depth or replacing layer normalization with batch normalization seemed to offer no significant improvements or even made the results worse. However when training the network without the usage of LayerScale (Section 3.2.4), which in the context of the ConvNeXt paper is considered only a minor improvement to stabilize the training, rather than a key element to boost the performance.

Grad-CAM, while not perfect, offers much better results compared the the normal model when LayerScale was not used. These results showed, that even some smaller design choices might impact the interpretability of new networks, making it challenging to create a futureproof XAI method. While the conducted test helped to shine some light on the problem, given the large amount of changes to training procedures and CNN designs made in resent years, further investigation into factors that could make a proposed CNN less interpretable for some established methods would be interesting, but were not possible within the scope of this thesis.

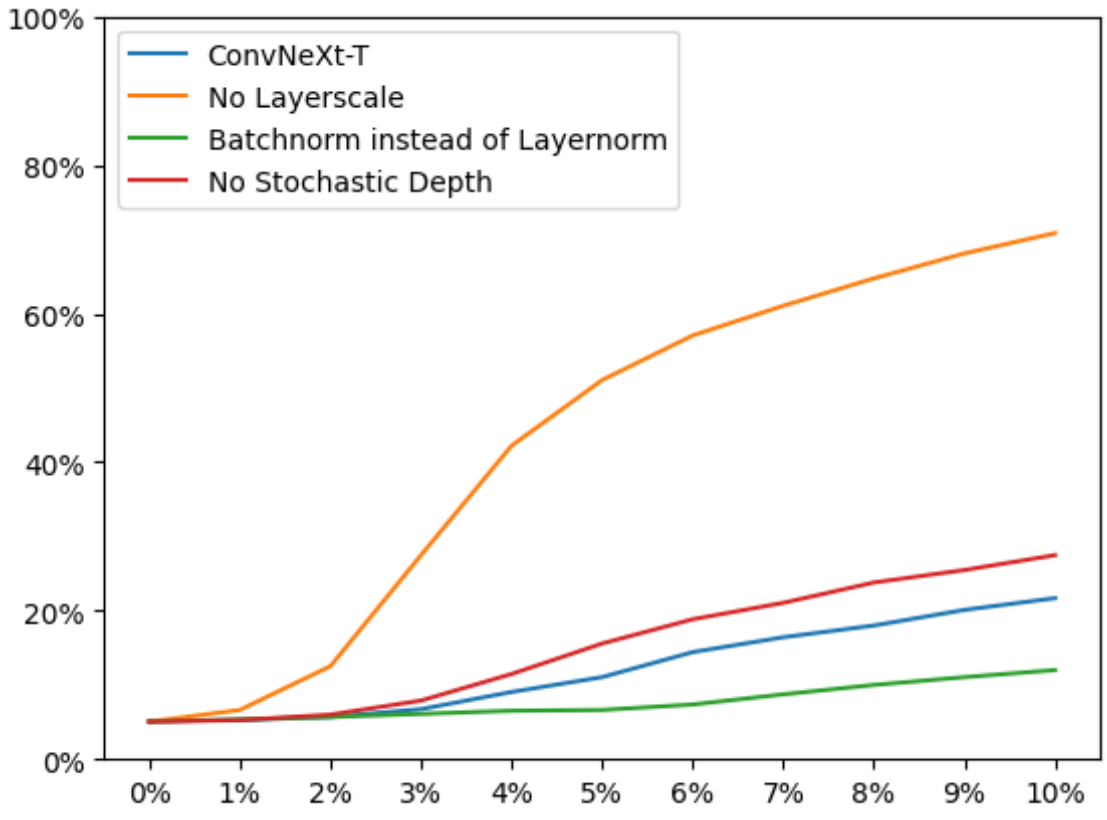

Figure 45: Grad-CAM results on different variations of the ConvNext-T design

7 Conclusion and Outlook

The main conclusion of this thesis can be considered the fact, that ranking XAI methods remains difficult. While CNNs themselves already depend on many factors, like the choice of the dataset, the augmentation and preprocessing of the training data or which scheduler and optimizer is used, XAI methods on top of that are supposed to work for different models, datasets or possibly even a broader range of AI tasks.

In this thesis it became very clear that even an established method that works really well on multiple CNNs like Grad-CAM might struggle at producing good explanations for a different model or if a certain training technique is used, as showcased by the performance Grad-CAM had for the state of the art ConvNeXt model. On the other hand the IG based methods worked really well here, yet showed mediocre performance on the ResNet50 and VGG16 models.

The only method which consistently did very well was LIME, which could be attributed to the fact that it is designed to be model agnostic and applicable on a wide variety of different ML tasks. At the same time LIME was never the ideal XAI method for a given CNN and was always outperformed by at least one of the other methods.

The novel XAI-method proposed in this thesis was able to perform fairly well and either matched or outperformed the other IG based method (XRAI) on all tested models. Most notably within the evaluated XAI-methods it had the best performance for the state of the art ConvNeXt-T model.

The results underlined the need for a good standardized benchmarking method which is based on a variety of networks, as the discussed methods showed very different results depending on the CNN they were applied to.

While this thesis can be seen as a step towards establishing such a basis to compare XAI methods on, more work needs to be done in order to have a good standardized benchmarking method for XAI.

Another takeaway from this thesis should be, that while XAI methods should work post-hoc on any network, certain considerations when designing a new CNN might prevent some otherwise capable XAI methods from working properly.

For the future it would be desirable if both research fields can work closely together, in order to ensure that newly proposed models are not only more performant to their predecessors, but are ideally still interpretable using existing XAI methods, which as presented in Section 6.5 might be achievable with minor changes to a model or training process without a major loss in performance.

A Shapes of Signs and Class Descriptions for the ATSDS

Label	Class Description	Shape
00001	Speed Limit 30	round
00002	Speed Limit 50	round
00003	Speed Limit 60	round
00004	Speed Limit 70	round
00005	Speed Limit 80	round
00007	Speed Limit 100	round
00008	Speed Limit 120	round
00009	No passing	round
00010	No passing if weight is 3.5t+	round
00011	Right of way (Intersection)	triangle
00012	Right of way (Street)	rhombus
00013	Give way.	reverse triangle
00014	STOP	octagon
00017	No Entry	round
00018	Danger	triangle
00025	Roadwork	triangle
00031	Deer crossing	triangle
00035	Drive straight	round
00038	Pass obstacle on the right	round

B Python Code for the Simple CNN using the pytorch framework

```
class SimpleCNN(nn.Module):
    def __init__(self, n_classes):
    super(SimpleCNN, self).__init__()
        self.stem = nn.Conv2d(3, 32, 7, stride=2)
        self.bn1 = nn.BatchNorm2d(32)
        self.conv1 = nn.Conv2d(32, 64, 3, padding=1)
        self.bn2 = nn.BatchNorm2d(64)
        self.conv2 = nn.Conv2d(64, 128, 3, padding=1)
        self.bn3 = nn.BatchNorm2d(128)
        self.conv3 = nn.Conv2d(128, 256, 3, padding=1)
        self.bn4 = nn.BatchNorm2d(256)

        self.dropout1 = nn.Dropout(0.25)
        self.fc1 = nn.Linear(256, n_classes)

    def forward(self, x):
        x = self.bn1(F.relu(self.stem(x)))
        x = self.bn2(F.relu(self.conv1(x)))
        x = F.max_pool2d(x, 2, stride=2)
        x = self.bn3(F.relu(self.conv2(x)))
        x = self.bn4(F.relu(self.conv3(x)))
        x = self.dropout1(x)
        x = F.adaptive_avg_pool2d(x, (1, 1))
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        return x
```

C Training Details

Item	Parameter
Number of Epochs	200
Optimizer	AdamW
Momentum β_1	0.9
Momentum β_2	0.999
Loss	Cross-Entropy
Learning Rate	1e-4
Learning Rate Scheduler	cosine decay
Weight Decay	1e-5
Batch Size	128

Changes were made to the VGG16 and the ConvNeXt-T network. VGG16 using default parameters suffered from slight overfitting and had it's weight decay increased to 4e-5 instead.

The ConvNeXt-T model was training slower than the other models and not reaching sufficient accuracy within the 200 epochs on a 1e-4 learning rate. The ConvNeXt-T was thus trained with an increased learning rate of 3e-4 instead.

Notably the proposed training procedures by Zhuang Liu et al. (2022) including label smoothing and additional data augmentation techniques and were not fully implemented, as it was unclear how good they would work on the older network architectures and even in the paper were not used for every specification of the ConvNeXt architecture.

D Implementation Details

The implementation of CNNs, training procedures, XAI methods and miscellaneous programming tasks for this thesis was done in Python. The Code, including a list of the utilized libraries of which the most important ones will be discussed below can be found on Github:

Datasets

The ATSDS dataset was implemented as a "VisionDataset" using the torchvision datasets library. The GTSRB (Stallkamp et al., 2012) dataset for the streetsigns was retreived using the torchvision datasets library. The Google Street View Dataset was downloaded from the official website of the Center for Research in Computer Vision from the University of Central Florida (https://www.crcv.ucf.edu/data/GMCP_Geolocalization). For each of the datasets 9500 images eligible as described in Section 3.1 were used. When a choice had to be made the lexicographic first images were taken, e.g. if 500 traffic signs from a class with 900 eligible images were taken or for the choice of the 9500 background images from the Google Street View dataset. For the creation of the final images, each streetsign was assigned a random background index. A random starting point on the x and y axis was chosen and the cutout of the streetsign according to its shape was inserted. The Cutout was produced using the PIL.ImageDraw library. The random seed used was "42". The code is provided in the Github repository.

CNN Training

The discussed CNNs were implemented using pytorch and specifically the torchivision library. Training was conducted using CUDA on an NVIDA A100 GPU from the high performance cluster from the "Technische Universität Dresden".

XAI-Methods

While some methods(Grad-CAM, IG, IGF, XRAI) were implemented from scratch, the "torchPRISM" and the "lime" library for python were used for the respective methods. Using existing libraries for more complex methods allowed to have a broader and more relevant comparison of XAI methods, without requiring a large time investment to implement them.

E Detailed Occlusion Accuracy Results

Table 9: Occlusion Accuracy Results - Simple CNN

%	Grad-CAM	IGF	LIME	PRISM	XRAI
0	0.99	0.99	0.99	0.99	0.99
1	**0.43**	0.81	0.87	0.98	0.84
2	**0.16**	0.72	0.70	0.96	0.69
3	**0.08**	0.66	0.54	0.95	0.64
4	**0.06**	0.63	0.41	0.94	0.61
5	**0.06**	0.61	0.29	0.93	0.60
6	**0.06**	0.60	0.22	0.92	0.57
7	**0.06**	0.59	0.17	0.92	0.56
8	**0.06**	0.57	0.14	0.90	0.54
9	**0.06**	0.56	0.12	0.89	0.53
10	**0.06**	0.55	0.10	0.87	0.52

Table 10: Occlusion Accuracy Results - VGG16

%	Grad-CAM	IGF	LIME	PRISM	XRAI
0	0.99	0.99	0.99	0.99	0.99
1	**0.32**	0.35	0.73	0.41	0.41
2	**0.15**	0.18	0.55	0.20	0.19
3	**0.09**	0.11	0.39	0.12	0.09
4	**0.05**	0.09	0.27	0.08	0.07
5	**0.05**	0.07	0.21	0.07	0.06
6	**0.05**	0.06	0.15	0.07	0.06
7	**0.05**	**0.05**	0.12	0.06	0.06
8	**0.05**	0.06	0.10	0.06	**0.05**
9	**0.05**	0.06	0.08	0.06	**0.05**
10	**0.05**	0.06	0.08	0.06	**0.05**

Table 11: Occlusion Accuracy Results - ResNet50

%	Grad-CAM	IGF	LIME	PRISM	XRAI
0	0.99	0.99	0.99	0.99	0.99
1	**0.53**	0.71	0.73	0.74	0.76
2	**0.26**	0.64	0.53	0.56	0.66
3	**0.15**	0.62	0.4	0.43	0.62
4	**0.11**	0.59	0.29	0.36	0.60
5	**0.08**	0.57	0.22	0.30	0.59
6	**0.07**	0.56	0.18	0.26	0.57
7	**0.07**	0.54	0.16	0.23	0.55
8	**0.06**	0.52	0.14	0.22	0.55
9	**0.06**	0.51	0.13	0.19	0.52
10	**0.06**	0.50	0.12	0.18	0.52

Table 12: Occlusion Accuracy Results - ConvNeXt-T

%	Grad-CAM	IGF	LIME	PRISM	XRAI
0	0.98	0.98	0.98	0.98	0.98
1	0.93	**0.33**	0.65	0.91	0.38
2	0.88	**0.20**	0.44	0.87	0.23
3	0.84	**0.13**	0.31	0.84	0.15
4	0.82	**0.10**	0.21	0.80	0.11
5	0.80	**0.08**	0.16	0.77	0.09
6	0.79	**0.07**	0.12	0.74	**0.07**
7	0.77	**0.06**	0.1	0.71	0.07
8	0.76	**0.06**	0.09	0.67	0.07
9	0.76	**0.06**	0.08	0.65	0.07
10	0.75	**0.05**	0.07	0.62	0.06

F Detailed TPR Results

Table 13: TPR Results - Simple CNN

%	Grad-CAM	IGF	LIME	PRISM	XRAI
1	**0.24**	0.09	0.16	0.01	0.10
2	**0.46**	0.17	0.32	0.02	0.19
3	**0.65**	0.25	0.46	0.03	0.26
4	**0.78**	0.31	0.57	0.04	0.31
5	**0.86**	0.36	0.67	0.05	0.34
6	**0.9**	0.39	0.74	0.05	0.37
7	**0.92**	0.42	0.81	0.06	0.39
8	**0.93**	0.44	0.85	0.08	0.41
9	**0.94**	0.46	0.89	0.09	0.42
10	**0.94**	0.47	0.92	0.10	0.44

Table 14: TPR Results - VGG16

%	Grad-CAM	IGF	LIME	PRISM	XRAI
1	**0.23**	0.22	0.15	0.21	0.22
2	**0.44**	0.43	0.30	0.41	**0.44**
3	**0.63**	0.61	0.43	0.59	**0.63**
4	**0.78**	0.74	0.54	0.74	**0.78**
5	**0.89**	0.84	0.63	0.84	**0.89**
6	**0.94**	0.90	0.70	0.89	**0.94**
7	**0.97**	0.94	0.76	0.92	0.96
8	**0.98**	0.95	0.80	0.94	0.97
9	**0.98**	0.96	0.84	0.94	0.97
10	**0.98**	0.97	0.87	0.95	0.97

Table 15: TPR Results - ResNet50

%	Grad-CAM	IGF	LIME	PRISM	XRAI
1	**0.21**	0.09	0.15	0.12	0.09
2	**0.39**	0.17	0.28	0.22	0.17
3	**0.54**	0.24	0.40	0.31	0.23
4	**0.67**	0.31	0.50	0.39	0.27
5	**0.76**	0.35	0.59	0.47	0.30
6	**0.83**	0.39	0.66	0.53	0.32
7	**0.88**	0.41	0.72	0.58	0.34
8	**0.92**	0.44	0.76	0.63	0.36
9	**0.94**	0.45	0.80	0.66	0.38
10	**0.96**	0.47	0.83	0.69	0.39

Table 16: TPR Results - ConvNeXt-T

%	Grad-CAM	IGF	LIME	PRISM	XRAI
1	0.02	**0.22**	0.16	0.03	0.21
2	0.05	**0.42**	0.32	0.06	**0.42**
3	0.07	**0.59**	0.45	0.08	**0.59**
4	0.09	0.72	0.57	0.10	**0.74**
5	0.10	0.82	0.66	0.12	**0.83**
6	0.12	**0.89**	0.74	0.15	**0.89**
7	0.14	**0.92**	0.8	0.17	0.91
8	0.15	**0.95**	0.85	0.19	0.93
9	0.17	**0.96**	0.88	0.21	0.95
10	0.18	**0.97**	0.91	0.23	0.95

References

Alejandro Barredo Arrieta, Natalia Dıéaz Rodrıéguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador Garcıéa, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera (2019). "Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI". In: *CoRR* abs/1910.10045. arXiv: 1910.10045.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton (2016). *Layer Normalization*. arXiv: 1607.06450.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba (2016). "OpenAI Gym". In: *CoRR* abs/1606.01540. arXiv: 1606.01540.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei (2009). "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255.

Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri (2021). "A Comprehensive Survey and Performance Analysis of Activation Functions in Deep Learning". In: *CoRR* abs/2109.14545. arXiv: 2109.14545.

Christian Ertler, Jerneja Mislej, Tobias Ollmann, Lorenzo Porzi, and Yubin Kuang (2019). "Traffic Sign Detection and Classification around the World". In: *CoRR* abs/1909.04422. arXiv: 1909.04422.

Pedro F. Felzenszwalb and Daniel P. Huttenlocher (2004). "Efficient Graph-Based Image Segmentation". In: *International Journal of Computer Vision* 59.2, pp. 167–181.

David Freedman, Robert Pisani, and Roger Purves (2007). "Statistics (international student edition)". In: *Pisani, R. Purves, 4th edn. WW Norton & Company, New York*.

Yunhui Guo, Yandong Li, Rogério Schmidt Feris, Liqiang Wang, and Tajana Rosing (2019). "Depthwise Convolution is All You Need for Learning Multiple Visual Domains". In: *CoRR* abs/1902.00927. arXiv: 1902.00927.

Vikas Hassija, Vinay Chamola, Atmesh Mahapatra, Abhinandan Singal, Divyansh Goel, Kaizhu Huang, Simone Scardapane, Indro Spinelli, Mufti Mahmud, and Amir Hussain (Aug. 2023). "Interpreting Black-Box Models: A Review on Explainable Artificial Intelligence". In: *Cognitive Computation* 16.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2015). "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385. arXiv: 1512.03385.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt (2020). "Measuring Massive Multitask Language Understanding". In: *CoRR* abs/2009.03300. arXiv: 2009.03300.

Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel (2013). "Detection of Traffic Signs in Real-World Images: The German Traffic Sign Detection Benchmark". In: *International Joint Conference on Neural Networks*. 1288.

Gao Huang, Zhuang Liu, and Kilian Q. Weinberger (2016). "Densely Connected Convolutional Networks". In: *CoRR* abs/1608.06993. arXiv: 1608.06993.

Sergey Ioffe and Christian Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167. arXiv: 1502.03167.

Andrei Kapishnikov, Tolga Bolukbasi, Fernanda B. Viégas, and Michael Terry (2019). "Segment Integrated Gradients: Better attributions through regions". In: *CoRR* abs/1906.02825. arXiv: 1906.02825.

Minhyeok Lee (2023). *GELU Activation Function in Deep Learning: A Comprehensive Mathematical Analysis and Performance*. arXiv: 2305.12073.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo (2021). "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows". In: *CoRR* abs/2103.14030. arXiv: 2103.14030.

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie (2022). "A ConvNet for the 2020s". In: *CoRR* abs/2201.03545. arXiv: 2201.03545.

Ilya Loshchilov and Frank Hutter (2017). "Fixing Weight Decay Regularization in Adam". In: *CoRR* abs/1711.05101. arXiv: 1711.05101.

Romy Müller (2024). "How Explainable AI Affects Human Performance: A Systematic Review of the Behavioural Consequences of Saliency Maps". In: *International Journal of Human–Computer Interaction* 0.0, pp. 1–32.

Romy Müller, Marcel Dürschmidt, Julian Ullrich, Carsten Knoll, Sascha Weber, and Steffen Seitz (2023). *Do humans and Convolutional Neural Networks attend to similar areas during scene classification: Effects of task and image type*. arXiv: 2307.13345 [cs.CV].

M.A. Nielsen (2015). *Neural Networks and Deep Learning*. Determination Press.

Razvan Pascanu, Tomás Mikolov, and Yoshua Bengio (2013). "On the difficulty of training Recurrent Neural Networks". In: *CoRR* abs/1211.5063. arXiv: 1211.5063.

Ilija Radosavovic, Raj Prateek Kosaraju, Ross B. Girshick, Kaiming He, and Piotr Dollár (2020). "Designing Network Design Spaces". In: *CoRR* abs/2003.13678. arXiv: 2003.13678.

Bharath Ramsundar and Reza Bosagh Zadeh (2018). *TensorFlow for Deep Learning: From Linear Regression to Reinforcement Learning*. 1st. O'Reilly Media, Inc.

Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin (2016). ""Why Should I Trust You?": Explaining the Predictions of Any Classifier". In: *CoRR* abs/1602.04938. arXiv: 1602.04938.

Sebastian Ruder (2016). "An overview of gradient descent optimization algorithms". In: *CoRR* abs/1609.04747. arXiv: 1609.04747.

Maria Salinas, Javiera Sepúlveda, Leonel Hidalgo, Dominga Peirano, Macarena Morel, Pablo Uribe, Veronica Rotemberg, Juan Briones, Domingo Mery, and Cristian Navarrete-Dechent (May 2024). "A systematic review and meta-analysis of artificial intelligence versus clinicians for skin cancer diagnosis". In: *npj Digital Medicine* 7.

A. Saranya and R. Subhashini (2023). "A systematic review of Explainable Artificial Intelligence models and applications: Recent developments and future trends". In: *Decision Analytics Journal* 7, p. 100230.

Iqbal H. Sarker (2021). "Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions". In: *SN Computer Science* 2.6, p. 420.

Ramprasaath R. Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra (2016). "Grad-CAM: Why did you say that? Visual Explanations from Deep Networks via Gradient-based Localization". In: *CoRR* abs/1610.02391. arXiv: 1610.02391.

Karen Simonyan and Andrew Zisserman (2014). "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556.

J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel (2012). "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition". In: *Neural Networks* 32. Selected Papers from IJCNN 2011, pp. 323–332.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan (2017). "Axiomatic Attribution for Deep Networks". In: *CoRR* abs/1703.01365. arXiv: 1703.01365.

Tomasz Szandała and Henryk Maciejewski (2022). "PRISM: Principal Image Sections Mapping". In: *Computational Science – ICCS 2022*. Ed. by Derek Groen, Clélia de Mulatier, Maciej Paszynski, Valeria V. Krzhizhanovskaya, Jack J. Dongarra, and Peter M. A. Sloot. Springer International Publishing, pp. 749–760.

Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou (2021). "Going deeper with Image Transformers". In: *CoRR* abs/2103.17239. arXiv: 2103.17239.

Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He (2016). "Aggregated Residual Transformations for Deep Neural Networks". In: *CoRR* abs/1611.05431. arXiv: 1611.05431.

Haobo Yang, Wenyu Wang, Ze Cao, Zhekai Duan, and Xuchen Liu (2023). "InDL: A New Dataset and Benchmark for In-Diagram Logic Interpretation based on Visual Illusion". In: arXiv: 2305.17716.

A.R. Zamir and M. Shah (2014). *Image Geo-localization Based on Multiple Nearest Neighbor Feature Matching using Generalized Graphs*.

Yifei Zhang, Siyi Gu, James Song, Bo Pan, Guangji Bai, and Liang Zhao (2023). *XAI Benchmark for Visual Explanation*. arXiv: 2310.08537.

Bolei Zhou, Aditya Khosla, Àgata Lapedriza, Aude Oliva, and Antonio Torralba (2015). "Learning Deep Features for Discriminative Localization". In: *CoRR* abs/1512.04150. arXiv: 1512.04150.

List of Figures

1 Artificial Neuron. 2

2 A fully connected ANN/FNN mapping eight input points (left) to four outputs (right) using three "hidden" layers in between as illustrated by Ramsundar and Zadeh (2018) 3

3 Behavior of the Sigmoid (green), ReLU (purple) and GELU (red) function close to zero (L) and when zoomed out (R). 5

4 Example for the application of a 3x3 convolutional filter on a 4x4 input without padding. The example calculation is based on the area marked with the green dashed lines. Notably the spatial dimension of the output decreased to 2x2. 6

5 Usage of zero padding to prevent the reduction in the spatial dimension. The required padding depends on the size and stride of the kernel. 6

6 Illustration of pooling methods using a 2x2 kernel size and a stride of two. Average pooling takes the average of the four elements it overlaps with, while max pooling extracts the maximum. Notably the spatial dimension is reduced by half when using the typical kernel size of 2x2 and a stride of two. 7

7 Illustration of a 3x3 kernel application on a three channel input using a standard and a depthwise convolution followed by a pointwise convolution (Guo et al., 2019). The standard convolution (L) is applied on all three channels at once, while depthwise convolution operates on a per channel basis instead and achieves the channel mixing through the pointwise convolution afterwards (R). 8

8 Example structure for a modern convolutional neural network. a) shows the basic structure consisting of a stem, the body and the head. b) shows a sample body, consisting of multiple stages. c) shows a stage consisting of multiple blocks. As explained in Section 2.1.8 the number of channels or width w_i is usually doubled between stages, while the spatial dimension r is halved. As depicted in c) this usually happens before the output of the prior stage is fed into the first block of the new stage, while number of channels and spatial dimension is retained throughout the stage. 11

9 Example Images from the GTSRB dataset for three of the 43 classes. 16

10 Example Images from the Google Street View Dataset. Navigation Elements from Google Street View can be seen in the top left of the images. . 17

11 ATSDS example images featuring all of the five possible traffic sign shapes. 18

12 Visualization of the Simple CNN. 20

13 Simplified visualization of the VGG16 model structure. 22

14 Proposed Block Designs for ResNet18/30 (left) and ResNet50/152 (right). Residual connections allow for better gradient flow. Typically residual connections skip multiple layers or a block (He et al., 2015). 23

15 Simplified visualization for the ResNet50 architecture. 23

16 a) ResNeXt block b) Inverted bottleneck - c) ConvNeXt block design. The image from (Zhuang Liu et al., 2022) shows the inversion of the bottleneck, followed by moving the depthwise convolution (d3x3) upstream to allow for a kernel size change afterwards. 25

17 Comparison of the ConvNeXt block (R) with the ResNet50 block, it is based on (Zhuang Liu et al., 2022). Compared to the design shown in Figure 16 the kernel size of the depthwise convolution is increased, LN is used after the depthwise convolution and GELU is applied after the first 1x1 kernel. 26

18 Simplified visualization for the ConvNeXt network. 27

19 ConvNeXt-T architecture compared to the ResNet50 (Zhuang Liu et al., 2022). 27

20 Adapted image from the CAM paper (Zhou et al., 2015) showcasing how features are weighted and combined to retrieve a class specific heatmap of relevant regions 29

21 Adapted Diagram from CAM (Zhou et al., 2015) showing the core similarities but different derivation of weights for the Grad-CAM algorithm . . . 30

22 Example images for the PRISM method. Original image (1), PRISM output (RGB) (2), the generated single channel heatmap (3) and a visualization of the mask on top of the image using alpha blending (4). 31

23 Example images showcasing the LIME heatmaps. Original image (L), LIME mask (M) and an overlayed mask (R). Most notably unlike other methods LIME produces specific features, resulting in a binary mask, rather than a smooth heatmap. 32

24 Example images showing the IG attributions. Original Image (L), IG attributions (M) and a visualization of the mask on top of the image using alpha blending (R). Notably the IG attributions are very scattered and thus do not offer a trivial way to extract a connected importance area. 33

25 Example images for the XRAI Method. Original image (L), XRAI heatmap (M) and a visualization of the mask on top of the image using alpha blending (R). 34

26 Image from Kapishnikov et al. (2019) showcasing the poor results provided by IG (M) if the pixel color of the baseline and the important part of the input image (L) are both black. To address this problem attributions are calculated for both a white and black baseline and combined (R). . . . 34

27 Example images showing the IGF method. Original Image (L), corresponding IGF mask (M) and a visualization of the mask on top of the image using alpha blending (R). 36

28 Example visualization of the impact the scale parameter s has on the impacted area on the heatmap around the pixel for each attribution. Small values (1), (2) might limit the impact too much and thus might not create proper segments, while larger values (4) might influence their surrounding too much. The chosen scale parameter (3) produces smooth segments without diluting the sphere of influence for a pixel attribution too much. . 36

29 Different Occlusion levels, starting from 1% occlusion (top left) to 10% occlusion (bottom right). As observable in the last two images, the streetsign might no longer remain in the image even with less than 10% occlusion level. 39

30 Different revelation levels, starting from 1% revelation (top left) to 10% revelation (bottom right). As observed with the occluded images, the full streetsign is visible at less than 10%. 40

31 Train and test loss for the different CNNs over the 200 epoch training period. 43

32 Train and test accuracy for the different CNNs over the 200 epoch training period. A red dashed line is drawn at 100%. 44

33 Revelation Accuracy for the Simple CNN showcasing the achieved accuracy for on the test dataset for revelation levels from 0 to 10%. 46

34 Revelation Accuracy for the VGG16 CNN. Revelation Accuracy for the Simple CNN showcasing the achieved accuracy for on the test dataset for revelation levels from 0 to 10%. 47

35 Revelation Accuracy for the Resnet50. Revelation Accuracy for the Simple CNN showcasing the achieved accuracy for on the test dataset for revelation levels from 0 to 10%. 48

36 Example images showcasing a problem the IG methods often faced on the ResNet architecture. While the traffic sign is highlighted, segments on the left of the sign (XRAI example) or a area right to it (IGF) were highlighted stronger, which caused the method to perform poorly. 48

37 Revelation Accuracy for the ConvNeXt. Revelation Accuracy for the Simple CNN showcasing the achieved accuracy for on the test dataset for revelation levels from 0 to 10%. 50

38 Classwise accuracies for PRISM (L) and Grad-CAM (R) showcasing the accuracy for selected poorly performing classes complemented by the best performing class. For PRISM some classes showed no accuracy improvement below 5% revelation and even the best performing class "00013" achieved only around 50% accuracy at 10% revelation. Grad-CAM had similar issues, with some classes ("00002","00011") not showing any improvement to the accuracy as the revelation ratio was increased. Similar to PRISM, the best performing class ("00017") stayed below 60% accuracy even at a 10% revelation level. 50

39 Heatmaps produced by PRISM or Grad-CAM failing to capture the important area. While PRISM tended to highlight small parts of the important area on the image, other areas dominated the heatmaps. GradCAM heatmaps often did not follow any comprehendable pattern and if the traffic sign was highlighted, it seemed to be mostly by chance. 51

40 Results for the Occlusion Accuracy. Instead of depicting the Occlusion Accuracy itself, the plots depict the decrease in accuracy for a given occlusion level assuming a 100% accuracy for the fully visible image. A vertical line is drawn at 94,7%, which corresponds to the network making a random guess. Only the PRISM method really showed notable differences compared to the Revelation Accuracy. PRISM performed significantly better on the ResNet50, nearly matching LIME. For ConvNeXt the PRISM results were still subpar, but notably better compared to the results of the Revelation Accuracy. 52

41 TPR results for different CNNs 55

42 Revelation Accuracy of the IGF Method on the Simple CNN for different scale parameters. 56

43 Revelation Accuracy of the IGF Method on the Simple CNN for different scale parameters. 57

44 Revelation Accuracy of the IGF Method on the Simple CNN for different scale parameters. 58

45 Grad-CAM results on different variations of the ConvNext-T design . . . 60

List of Tables

1 Revelation Accuracy - Simple CNN 45

2 Detailed Revelation Accuracy data. Each line corresponds to the achieved network achieved for a corresponding percentage of the image being revealed. For readability the best method is highlighted in bold, while the worst method is underlined. 45

3 Revelation Accuracy - VGG16 46

4 Revelation Accuracy - Resnet50 47

5 Revelation Accuracy - ConvNeXt 49

6 Results - Pearson Correlation Coefficient 53

7 Results - Cosine Similarity 53

8 Results - Kullback–Leibler Divergence 54

9 Occlusion Accuracy Results - Simple CNN 65

10 Occlusion Accuracy Results - VGG16 65

11 Occlusion Accuracy Results - ResNet50 66

12 Occlusion Accuracy Results - ConvNeXt-T 66
13 TPR Results - Simple CNN 67
14 TPR Results - VGG16 67
15 TPR Results - ResNet50 67
16 TPR Results - ConvNeXt-T 68