Nicolas Blystad Carbone

An overview and comparison of Explainable AI (XAI) methods

Specialization Project TTK4550

Supervisor: Anastasios Lekkas
Trondheim, December 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Innovation and Creativity

Abstract

With the increasing amount of applications relying on black box machine learning, a rising concern about their trustworthiness is emerging. Methods deciphering how models reason and why predictions are made could lead towards greater trust in these systems consequently increase their application domain. Explainable AI (XAI) aims to provide such methods and are gaining momentum both in the industry and in academia. This specialization projects focuses on these techniques and covers why such methods are needed. The current state of the art is presented with a deep dive into their underlying theory. Further, a model for image classification is implemented and two XAI methods are applied to the model, LIME and SHAP. The strength and weaknesses of these explanations are discussed and followed by an attempt to improve the model. The improved model increases the test accuracy from 84.20% to 87.27%.

Preface

This specialization project concludes my 9th semester at Norwegian University of Science and Technology (NTNU) and is an important step towards the master thesis set to be carried out spring 2020. I would like to thank my supervisor Anastasios Lekkas for his valuable insights, suggestions and feedback throughout the project as it formed. His guidance was especially critical during the beginning of the semester where relevant papers and material on Explainable AI methods were recommended. I would also like to thank Andreas Thyholt Henriksen for providing helpful examples in the utilization of LIME and the scikit-image segmentation algorithms in Section 4.5.

Multiple software tools were utilized in Chapter 4. The Python (v3.6.9) programming language along with the NumPy, Matplotlib, scikit-learn and scikit-image packages were used to accomplish the experiments. The PyTorch deep learning framework v1.3.1[1] was applied to build and speed up the training of a model. The free Jupyter notebook environment, Google Colab[2], was used to support GPU-accelerated training of the model in Chapter 4. The Pytorch framework supports training on GPU and Google Colab provides open access to GPUs, making the combination a powerful tool. The free cloud storage service Google Drive[3] was integrated with Google Colab, enabling the project to be accessible from any computer as long as a stable internet connection was available. It was therefore used for the entire implementation of methods in the project. Some effort had to be put into solving issues in the Google Colab environment normally not present on a local machine, like saving figures in vector format, calculation timeout, store and retrieve data in the cloud, but workarounds were found. The convenience of free GPU cloud computing accessible from anywhere in the world made the benefits overly outweigh the drawbacks. The version control software provided by GitHub [4] added a layer of security by allowing routinely backup of the code along with any made figures. The authors open sourced packages of LIME[5] and SHAP[6] were used to explain the model predictions in Section 4.5 and Section 4.6, respectively. The model in Section 4.2 used a starter code provided in the NTNU course TDT4265[7] as a foundation for the development. NTNU provided a workplace at the university with a Dell Inc. OptiPLex 7060 computer and dual monitor setup.

Scalars are written in lower case x, a vector in lower case bold $\mathbf{x}$ and a matrix in upper case bold $\mathbf{X}$. Finally, the figures are the author's work unless otherwise stated. Besides the mentions in this section, all the work is carried out independently.

Nicolas Blystad Carbone,
December 17, 2019

[1] https://pytorch.org/
[2] https://colab.research.google.com/
[3] https://www.google.com/drive/
[4] https://github.com/
[5] https://github.com/marcotcr/lime
[6] https://github.com/slundberg/shap
[7] https://github.com/hukkelas/TDT4265-A3-starter-code

Table of Contents

List of Tables

List of Figures

Abbreviations

AI	=	Artificial Intelligence
XAI	=	Explainable Artificial Intelligence
DL	=	Deep Learning
ML	=	Machine Learning
NN	=	Neural Network
DNN	=	Deep Neural Network
CV	=	Computer Vision
CNN	=	Convolutional Neural Network
RGB	=	Red-Green-Blue color model represented as a tuple
SA	=	Sensitivity Analysis
LRP	=	Layer-wise Relevance Propagation
LIME	=	Local Interpretable Model-Agnostic Explanations
SHAP	=	Shapley Additive Explanations

Chapter 1

Introduction

1.1 Background and motivation

The recent advancement in machine learning (ML) and deep learning (DL) combined with the rising availability of data has enabled experts to reach increasingly sophisticated results in computer vision (CV) [17], image generation [25],[14], drug discovery [26] and other intricate tasks. While the results on their own are promising, these DL systems have been criticized for their distrusting black box approach [30]. As gradually more critical applications involves machine learning there is a huge need to provide trust in their decision. For instance in a medical application where a system reaches an unexpected conclusion it would be hugely beneficial to a doctor if the system could explain why it reached the prediction. This could, as a consequence, uncover correlations unknown to experts and potentially set new theories in motion. Take Google Brain's AlphaGo as an example on how an AI lead to a paradigm shift. In March 2016 the long reigning world champion in the game of Go, Lee Sedol, faced AlphaGo, a deep reinforcement learning based AI agent[5]. The system had reached superhuman levels in Go by observing strong human amateur games supplemented by self play. During the match between the two, at move 37 in the 2nd game, AlphaGo performs an unprecedented move which for advanced players would appear rather ignorant[7]. While the first impression implied a mistakenly poor move, it turned out to be a significant decision towards gaining the upper hand and ultimately win the battle. The move was so revolutionary that it later would become a part of modern strategies together with other insights from the games [8]. If the system in some way could've explained why it decided to perform the strange move, then maybe the Go community would gain even more insights into the revolutionary thinking of the self-learnt system.

Besides the benefit of understanding why predictions are made there is also a security as-

pect involved in these applications. There has been examples of adversarial attacks tricking AI systems with imperceptible perturbations to humans [24]. One such example is shown in Figure 1.1 where small alterations of a few pixels deceives the classifier. Although these attacks require the possibility to observe the input-output relation, it is concerning how robust these methods likely will be in the future in the hands of an unethical attacker. Since the system is trained on a limited set of data there are edge cases unknown to the model and these shortcomings are exploited in these attacks. The examples in Figure 1.1 show some of the driving motivation to develop methods exposing weaknesses in the model.

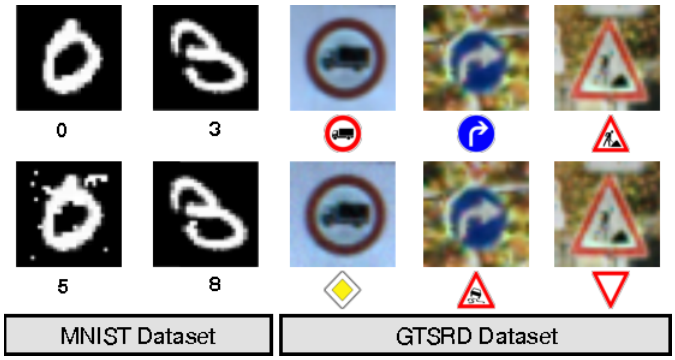


Figure 1.1: Adversarial attack showcasing the issues around an opaque system unable to provide explanations for its predictions. Input samples are shown at the top row with the model's correctly classified output beneath. The figures at the lower half are manufactured by altering pixels from the legitimate inputs resulting the model to misclassify. Figure source [24]

Explainable AI and the law

As the industry increasingly apply AI methods to support human decision making, it is expected to attain greater responsibility in a transition towards full automation. The impacts on individuals by these automated decision systems may be significant in cases such as medical treatment, access to loans, credit cards, insurance, employment and so on. The European Union's General Data Protection Regulation (GDPR) is a regulation imposed on all member states of the EU and the European Economic Area (EEA) that went into effect May 2018. It addresses data protection and privacy rights for citizens and aims to give individuals control of their personal data as well as the *right to an explanation* [1]. Recital 71 states

> The data subject should have the right not to be subject to a decision, which may include a measure, evaluating personal aspects relating to him or her which is based solely on automated processing and which produces legal effects concerning him or her or similarly significantly affects him or her, such as automatic refusal of an online credit application or e-recruiting practices without any human intervention [...]
>
> In any case, such processing should be subject to suitable safeguards, which

> should include specific information to the data subject and the right to obtain human intervention, to express his or her point of view, to obtain an explanation of the decision reached after such assessment and to challenge the decision.

The exempt clearly expresses that whenever automated decision affects an individual, they have the right to obtain an explanation for a decision and not be subjected to solely automated decisions. The automated decision making systems therefore need to abide the legislation by providing explanations to individuals who are affected by their decision. This calls for action in the field of AI where not only the importance of insights and security in such systems are relevant, but also their ability to act in accordance to legislation.

There has recently been an increasing number of proposed solution addressing the challenges with black box AI solutions. These methods are commonly referred to as eXplainable Artificial Intelligence (XAI) and tries to answer the question of ”why” the model concludes with a decision. Usually this involves visual clues on how the system responds to the input data. [30] summarizes two simple techniques, Sensitivity Analysis (SA) and Layer-wise Relevance Propagation, LRP (proposed in [4]). These methods could be used for any classification tasks and are shown to perform well on image classification, text classification and even human action in video. [35] introduces Integrated Gradients, a technique similar to Sensitivity Analysis. [36] proposes Local Interpretable Model-Agnostic Explanations (LIME) and [20] introduces SHAP. These methods are further discussed in Chapter 3 with LIME and SHAP applied in Chapter 4.

1.2 Objectives

The main objective of the project is to acquire an overview of the newly emerging field of Explainable Artificial Intelligence (XAI). This includes both the theoretical aspect and their respective implementation. Training a classifier and applying various state of the art XAI methods to the model will be essential to gain practical experience with the techniques. Consequently, a deeper understanding of the strengths and weaknesses of not only the network, but also the XAI methods themselves, is aimed to be obtained. Further, the project intends to explore the feasibility of improving a model by feedback from explanations. Finally, acquiring knowledge of how to perform literature survey, structuring a larger project and individual work is also an objective during the course of this project.

1.3 Outline

The specialization project is structured following the conventional **I**ntroduction,**M**ethods, **R**esults **a**nd **D**iscussion (IMRAD) structure.
Chapter 1 covers the background and motivation behind XAI by looking at scientific and

societal impacts of AI. The section concludes with an objective and the outline of the specialization project.
Chapter 2 intends to provide relevant background theory on neural network classifiers.
Chapter 3 presents an overview over some state of the art methods in XAI and insights from a survey on explainability according to social sciences. Chapter 4 introduces the training of a network to classify simple 32x32 pixel images into 10 classes. Further it showcases explanations by two of the XAI methods presented in Chapter 3 applied to the model. Finally, an attempt to improve the model based on information from the explanation is conducted.
The project concludes by summarizing the findings and discussing future work in Chapter 5.

Chapter 2

Fundamental Theory

The theory presented in this section is based on the recognized Deep Learning textbook by Ian Goodfellow et al [9] (Section 2.1 - section 2.6) with some inputs from Artificial Intelligence: A Modern Approach by S.Russel and P.Norvig [28] (Section 2.2). The material is intended to give a brief overview of the most relevant theory in Deep Learning to set the foundation for the methods presented in Chapter 3 and later applied in Chapter 4. The chapter starts with an introduction to Machine Learning in Section 2.1. The building block for deep neural networks, the artificial neuron, is presented next in Section 2.2 followed by multilayered deep neural networks in Section 2.3. Next, loss and optimization is introduced in Section 2.4 before a special type of network, the convolutional neural network, is presented in Section 2.5. Finally, the training process of a network is provided in Section 2.6.

2.1 Introduction - Machine Learning

The field of machine learning (ML) is based on algorithms and statistical models that can learn and infer from data without explicit instructions. The ML algorithms build a mathematical model on the given data, which may be labeled, partly labeled or unlabeled. Whenever the model learns from labeled data, meaning the data has been tagged with an appropriate label, it is called **supervised learning**. Examples include which words were spoken in an audio clip, whether the email is spam or not, if a tumor is present in an x-ray or simply what animal is seen in a photo. Since labeling datasets generally involves asking humans to annotate the data, it is costly, time consuming and may in some cases even be infeasible because of the quantity or simply by being uninterpretable by humans. In such instances the data is left as unlabeled and **unsupervised learning** techniques are used to operate on such data. The unsupervised methods aims to extract information from the unlabeled distribution by finding unknown patterns. This is typically done by clustering

data into groups such as in k-means clustering, but other methods like anomaly detection or autoencoders[1] may be appropriate depending on the application.

There are generally considered three main categories in machine learning, two of them being supervised- and unsupervised learning. In the last of the main three categories, **reinforcement learning**, a software agent performs actions in an (un)known environment to maximize its reward. This involves finding an optimal policy $\pi(s_i) = a_j$ for each state, s and action a. Reinforcement learning covers machine learning tasks that are beyond the scope of this project.

The following theory addresses the supervised learning task of *classification* using deep learning, a subset of machine learning. Though there exist a range of algorithms for this type of task[2], deep neural network is of interest as it outperforms the other methods in computer vision and image classification to which it is applied in Chapter 4. In a $classification$ task, the system is asked to predict which of the k groups some input belongs to. Formally, this means to learn a function $f : \mathbb{R}^n \to \{1, ..., k\}$, such that when the input $\mathbf{x} \in \mathbb{R}^n$ is passed through a function $f(\cdot)$ it is assigned the correct category identified by numeric code y.

2.2 Artificial Neuron

Artificial neurons are the fundamental building blocks in a neural network. They are mathematical models heavily inspired by biological neurons. x_i represents inputs flowing across a link i where each link has a numeric weight w_i portraying the strength of the connection. An input bias term b is added and, together with the weighted sum over the n inputs, passed through an activation function $g(\cdot)$. Put simply, each artificial neuron "fires" or activates based on the linear combination of its inputs. This input to output relation is defined as

$$a = g\left(\sum_{i=0}^{n} w_i x_i + b\right) \tag{2.1}$$

and a visualization is shown in Figure 2.1. Usually, the x_0 input to a unit is assigned the value 1 with an associated bias weight $w_0 = b$ such that Equation (2.1) can be rewritten as a dot product[3]

$$a_j = g\left(\mathbf{w}^\top \mathbf{x}\right). \tag{2.2}$$

$\mathbf{x}$ is the vector with activations from the previous units with the first element being $x_0 = 1$. The weight vector w consequently has the bias b as the first element followed by the n connection weights. This condenses the expression and simplifies the practical implementation by combining the weights and bias into a single vector $\mathbf{w}$. The choice of activation function g determines the amount of activation sent over the link from a neuron j to the

[1] Sometimes referred to as a feature extractor neural network .

[2] E.g. multinomial logistic regression, decision tree, random forest, nave Bayes, k-nearest neighbor and so on.

[3] This is sometimes referred to as the *bias trick*. See for instance http://cs231n.github.io/linear-classify/

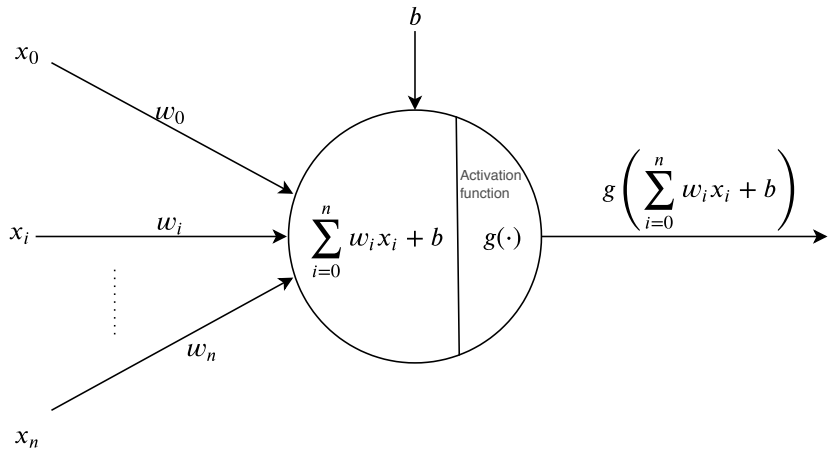


Figure 2.1: One of many possible visualizations of an artificial neuron. The input x_i could be the activation sent from another neuron or a numeric raw value originating from data.

next and is chosen to be nonlinear. This is to ensure the important property that the connected network of neurons can represent a nonlinear function. Even though there are a vast range of possible activation functions, only a few are introduced, starting with the Sigmoid activation function.

Sigmoid

The sigmoid activation function is commonly used as the first activation function introduced to beginners because of its simple interpretation. It is constrained to an output between 0 and 1 as $x \to \pm\infty$ and this may be seen as a percentage of activation from a neuron. Albeit being simple, it has been heavily criticized for its problem of *vanishing gradients*[10][4] during backpropagation[5] which is known in the literature. It saturates when the input is either large positive or negative. The definition of the function is stated as

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

with its derivative $\sigma'(x) = \sigma(x)(1 - \sigma(x))$. The sigmoid function is commonly denoted by a σ.

[4] First paragraph on page 5

[5] Backpropagation and the gradient based learning method is discussed in Section 2.4.

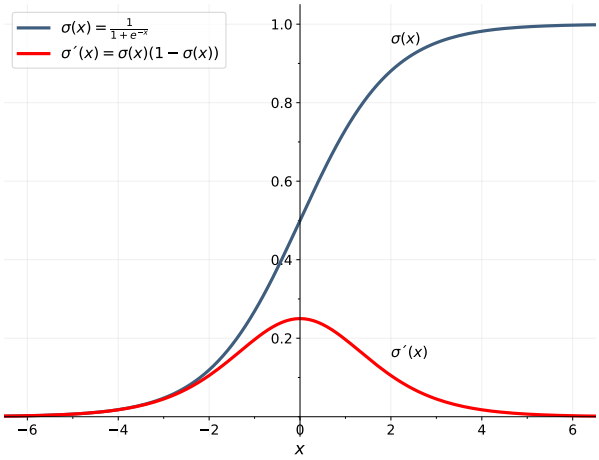


Figure 2.2: The sigmoid function plotted with its derivative. The sigmoid function constrains the output in the range $(0, 1)$. The derivative remains small and vanishes when x takes values away from the origin.

ReLU

Another nonlinear activation function is the rectifier linear unit (ReLu) defined as

$$g(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.4}$$

The derivative takes the form

$$g'(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if} x < 0 \end{cases} \tag{2.5}$$

where it is undefined at $x = 0$. A workaround in practice is to define a value for the derivative at $x = 0$, either the right or left derivative. Unlike the sigmoid, ReLu does not saturate when $x > 0$ If the learned bias b into a neuron is sufficiently negative[6], then the activation from the unit may remain 0 and causes the neuron to never fire. This is essentially the same as removing the neuron from the network and is described as the *dying ReLu problem*[7]. Once this happens, the gradient will forever remain zero as $f'(x) = 0$ when $x < 0$ and no update can correct for the learned parameters into the neuron. An attempt to mitigate this problem is to replace the 0 with αx for $x < 0$. This results in a modified ReLu, namely the LeakyReLu.

[6]This could be a symptom of learning rate set too high or a large gradient
[7]See http://cs231n.github.io/neural-networks-1/

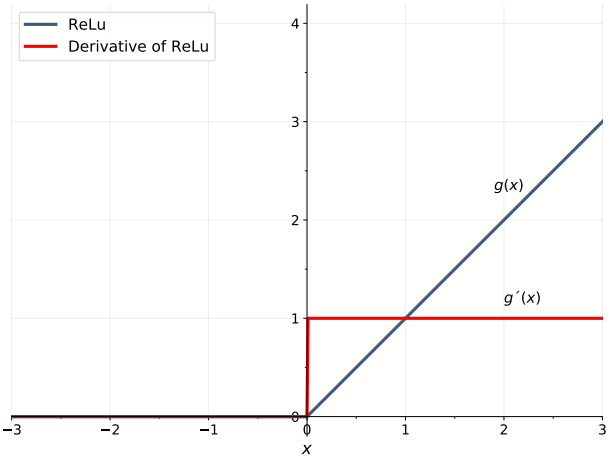


Figure 2.3: The ReLu function plotted with its derivative.

LeakyReLu

The LeakyReLu activation function is formally stated as

$$g(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha x, & \text{otherwise} \end{cases} \tag{2.6}$$

where α usually is in the range of 0.01. It allows a small, positive gradient of α when the unit is not active. The derivative is stated as

$$g'(x) = \begin{cases} 1, & \text{if } x > 0 \\ \alpha, & \text{if} x < 0 \end{cases} \tag{2.7}$$

where it is undefined at $x = 0$. This is solved, like in ReLu, by defining either the left or right derivative at $x = 0$.

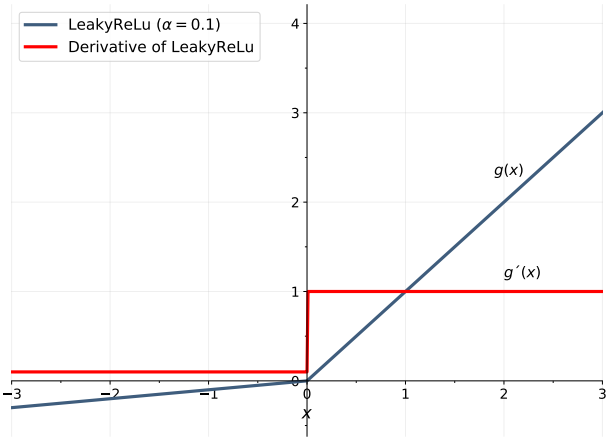


Figure 2.4: The LeakyReLu function plotted with its derivative. The α i set to 0.1 to visualize the slope and the positive gradient when $x < 0$

2.3 Deep Neural Networks

In this section Deep Neural Networks (DNN) are referring to what is sometimes called deep feed-forward neural networks or multilayer perceptrons (MLPs). The internal structure is composed by connected layers of artificial neurons. An illustration is presented in Figure 2.5. Each neuron in a layer is connected to every neuron in the next layer and activates based on the signal strength received, as discussed in section 2.2. At the output of the deep network, a vector or scalar states the final calculation by the network.

The goal in the classification task is to approximate some function f, such that $y = f(x)$ maps an input x to a class y. The feed-forward network approximates this function by defining a mapping $y = h(x; \theta)$ and finding the parameters θ for the weights through back-propagation, discussed in section 2.4. The feedforward term arise because of the direction of computation from input x through the network $h(x; \theta)$ to the output y without any internal feedback connections. The connections between the layers forms a directed acyclic graph.

The output in DNNs y are commonly **one-hot** encoded, a method used to quantify the categorical labels. Take a dataset with three different labels $\{cat, dog, human\}$ as an example for why this is needed. The computer needs to map the input $x \in \mathbb{R}^n$, say an image consisting of n pixels, to the output labels $\{cat, dog, human\}$. The labels could be translated to numeric values by mapping $cat = 1, dog = 2, human = 3$ but this would imply that dog is the average of cat and $human$ in the output space, certainly not the case. This is solved with **one-hot** encoding $cat = [1, 0, 0], dog = [0, 1, 0], human = [0, 0, 1]$. The

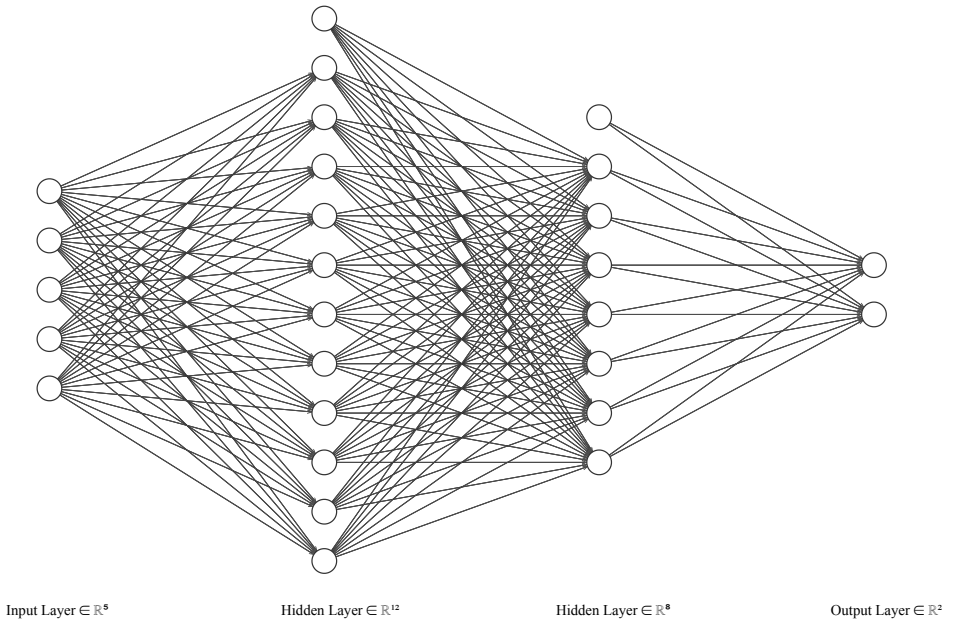


Figure 2.5: A deep neural net with two hidden layers. The bias term is depicted as a unit activation in the hidden layers. Figure made using the NN-SVG software [19]

output vector $\hat{\mathbf{y}}$ has length equal to the number of different classes presented in the labeled dataset. If the input belongs to the jth class, then the value at the jth position in $\hat{\mathbf{y}}$ should be highest, with the rest being (close to) zero. If the output vector $\hat{\mathbf{y}}$ is normalized such that $\sum_j \hat{y}_j = 1$, then each component of $\hat{\mathbf{y}}$ may be interpreted as a prediction confidence towards a class provided by the model. One such smooth normalization, *softmax*, is used to squash the output such that it may describe a probability distribution over n classes. It has the property of distributing the output vector $\hat{\mathbf{y}}$ such that each element describes a probability for the specific class c, namely $\hat{y}_c = p(y = c|\mathbf{x})$. The softmax function is stated in [9], for each class c, as

$$\text{softmax}(\hat{\mathbf{y}})_c = \frac{\exp \hat{y}_c}{\sum_j \exp \hat{y}_j} \tag{2.8}$$

where $\hat{y}_c$ is the output vale given to class c before applying softmax.

2.4 Loss function and Optimization

Once the architecture of the deep neural network is defined, tuning of weights and biases can be performed. The objective is now to fit a nonlinear function to the input-output space covered by the training data. The training data consist of an input $\mathbf{x}$ with a corresponding

label y and is used to tune the network. A cost function, sometimes referred to as a loss function or objective function, measures how the network performs on the training data. The cost function tries to punish the model by a high cost whenever confident, wrong prediction are made. A low cost is given for low confidence, correct predictions and in the ideal case of 100% confidence to the correct class, a zero cost is attained. There are multitudes of possible cost functions, two common are mean-squared loss

$$L(\mathbf{x}, y, \theta) = \text{MSE}_{\text{train}} = \frac{1}{m}\sum_{i=1}^{m} |\hat{\mathbf{y}}^{(i)} - \mathbf{y}^{(i)}|^2 \tag{2.9}$$

where the error increases whenever the Euclidian distance between prediction and the target becomes larger. The second is LogLoss

$$L(\mathbf{x}, y, \theta) = \text{LogLoss}_{\text{train}} = -\frac{1}{m}\sum_{i=1}^{m} \mathbf{y}^{(i)} \log(\hat{\mathbf{y}}^{(i)}) + (1 - \mathbf{y}^{(i)}) \log(1 - \hat{\mathbf{y}}^{(i)}) \tag{2.10}$$

where $\hat{\mathbf{y}}^{(i)}$ uses the earlier described softmax function to represent a probability. The negative sign is used to transform LogLoss to be strictly decreasing and the optimization problem becomes a minimization of the cost function. [9] recommends the LogLoss function with Softmax over MSE because of numerical stability. The LogLoss function is usually referred to as CrossEntropyLoss when it is used for multi-class classification. The loss sums over the number of examples in the training set, m, where $\hat{\mathbf{y}}^{(i)}$ is the prediction for sample number i with its corresponding label $y^{(i)}$. The loss function now quantifies a measure of the performance of the network. Whenever the loss is high, the network performs poorly, while a low loss signifies more accurate predictions. This property is used in the training by changing the weights in the layers of the network through optimization by minimizing the loss. The nonlinearities in a neural network causes the loss functions of interest to be non-convex[8], thereby ruling out useful properties obtained in convex optimization. Therefore, iterative, gradient-based optimizers are used to minimize the loss function. Unfortunately, because of the nature of nonlinear programming, the minimum is unlikely to be a global minimum, no convergence is guaranteed and the minimum is sensitive to the initial parameters. This means that the weights and biases in a feedforward neural network impacts the resulting minimum found by optimization. The optimization procedure by a gradient-based optimizer, gradient descent, uses the gradient of the loss function to change the weights such that the loss decreases. The weights are updated according to the gradient descent algorithm

$$\theta \leftarrow \theta - \eta \mathbf{g} \tag{2.11}$$

where η is the learning rate. The weights, θ, are updated based on the gradient of the loss back-propagated through the network. The loss is calculated over all training samples, but may also be updated more than once during a training iteration. This is referred to as updating the weights using *minibatches*. A *minibatch* contains a smaller set of samples, drawn uniformly from the training set, and is useful if the computer struggles to keep all

[8] Chapter 6.2 in Deep Learning book[9]

information in its memory. These are drawn until the complete training set has been used. Whenever gradient descent is used with *minbatches*, it takes the name Stochastic gradient descent (SGD). The gradient $\mathbf{g}$ in Equation (2.11) is found by computing

$$\mathbf{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta L(\mathbf{x}^{(i)}, y^{(i)}, \theta) \tag{2.12}$$

where L is a chosen loss function. The (Stochastic) gradient descent algorithm is one of many possible optimization algorithms and the book [9] gives a detailed introduction to the most common ones, such as SGD with momentum, SGD with adaptive learning rates (Adam, AdaGrad, RMSProp) and Newton's method.

2.5 Convolutional Neural Networks

Originally introduced in 1989 [18], Convolutional Neural Networks (CNNs) have been seen to perform exceptionally well in computer vision applications[17]. The CNN is a specialized kind of feedforward neural network which applies a sliding convolution operation over the input with a kernel, sometimes called a filter. The kernel consists of a grid of weights which performs a weighted sum over the input. The sum is then passed to the output, before the kernel strides a distance S and performs the convolution operation over the new input. This is done in sequence until the kernel has swept across the whole input and created a feature map at the output. A simple illustration is shown in Figure 2.6. Multiple number of kernels, K, may perform the same operation with different weights and create distinct feature maps based on the same input. This allows each filter to activate based on various features found in the input. The stacked sequence of feature maps may then be passed a new convolutional layer where a new set of kernels perform the convolution operation yet again. The stacked feature maps now takes up a width W_{out}, depth $D_{out} = K$, the kernel has size FxF and slides over the input with step size, stride, S. A zero padding around the input P could be added to fix the output to a desired width, normally such that $W_{out} = W_{in}$. The feature map then has the output size according to the formula $W_{out} = \frac{W_{in}-F+2P}{S} + 1$. Common hyperparamters are filter size FxF of 3x3 or 5x5, stride $S = 1$ and padding $P = 1$ or $P = 2$. Each element in the feature map is further passed through an nonlinear activation function, introduced in Section 2.2, to ensure nonlinear relationships between each layer. The nonlinear activation may be performed later if a pooling layer is used.

Pooling

A pooling layer helps the network to make features become invariant to small translations. The layer outputs a summary of the nearby inputs. For instance, the **max pooling** layer reports the maximum in an area of desired size, FxF. Other types of pooling layers exist, like **average pooling** taking the average of the nearby FxF pixels and the L2-norm pooling. Common sizes for the filter is 2x2 and stride 2 as depicted in Figure 2.7

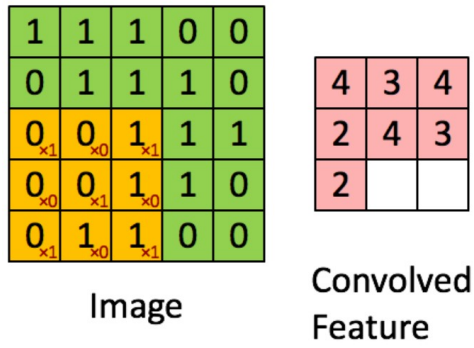


Figure 2.6: A simple convolutional layer with a 3x3 kernel illustrated in red writing. The feature map is show to the right, although not yet completed. The kernel has to stride over the last parts of the image to produce the two missing entries. Figure from towardsdatascience.com/beginners-guide-cnn

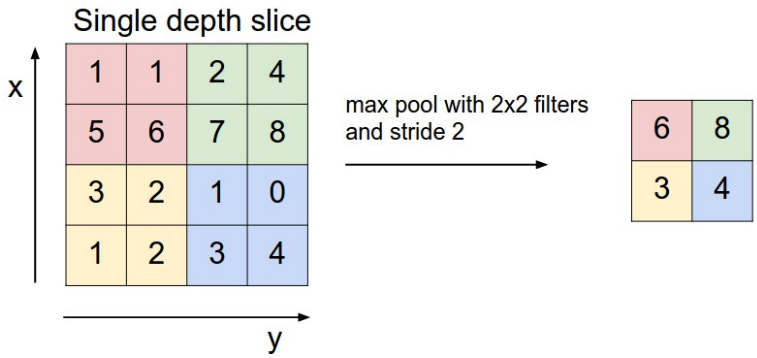


Figure 2.7: A downsampling max pooling layer with 2x2 filter and stride of 2. Figure from Stanford CS class CS231n http://cs231n.github.io/convolutional-networks/#pool

The kernel weights are updated during the training phase of model using loss and optimization as introduced in Section 2.4. Though CNNs are well suited for image applications, they are seldom used alone. Multiple CNN layers with pooling are usually stacked with a fully connected neural network (Section 2.3) to finally classify the input.

2.6 Training process

Once the architecture, loss function and optimization technique is chosen, the training of the model may begin. The model observes the training set and adjusts its network of weights by minimizing the loss measured by the loss function. Once a full sweep across the dataset is complete, the model has finished an *epoch*. Several epochs are needed to adjust the network until loss and accuracy converges. Even though the performance on

this dataset may be exceptional, it does not tell how well the model performs on new data. Therefore it is common to split the dataset into three sets: training set, validation set and test set. The validation set is used to validate the model during epochs. The expert may then evaluate the generalization by observing the accuracy on unseen data. It may further provide the designer feedback on the hyperparameters chosen to ensure convergence and desired performance. The hyperparameters could, but is not limited to, include the learning rate, weights, weight decay, network architecture etc[9]. The expert may interrupt the training at any point if the accuracy on the unseen validation set starts to decline, while it still increases on the training set. This is an indication that the model is transitioning towards *overfitting* or *overtraining*, meaning that the model learns features in the training data and not generalizing characteristics. This can be interpreted as the model is starting to 'remember' data seen in the training set and perform poorly on the unseen validation data. Finally, when the model has finished training, the test set provides an unbiased estimation on the accuracy of the final model by exposing it to unseen data. The expert may observe the percentage of accuracy and decide whether it performs sufficiently. If the accuracy decreases significantly, this may indicate overtraining and a modified architecture, more data, preprocessing, data augmentation or other generalization methods may be necessary to improve performance. One such generalization technique, **Dropout**, randomly deactivates neurons during training with probability p and has been shown to be effective [34]. A common preprocessing technique is to ensure that the data is centered with mean $\mu = 0$ and then either normalize such that the data is in range $[-1, 1]$ or normalize using a standard deviation σ. For images, **Global contrast normalization** performs such a normalization by scaling the different RGB color channels.

[9]It is worth mentioning that the fitted model is introduced to some bias as information from the validation set is used to influence the parameters in the model.

Chapter 3

Explainable Artificial Intelligence

The aforementioned methods in Chapter 2 have shown exceptional results, in cases even outperforming humans in applications discussed in Chapter 1. The techniques, however, do not obey conventional modelling where human experts carefully decide important attributes and features, but rather learns hidden patterns often unknown to the expert. These relationships may be causal and could lead to a poor generalization if the learned characteristics happen to be either oversimplified or too specific. The datasets used to train these methods are commonly intended to represent a wide array of conditions in order to obtain models which infer well on unobserved data. Explainable AI aims to help bridge the gap between the model and human interpretability.

The first three methods, Sensitivity Analysis, Layer-wise Relevance Propagation and Integrated Gradients, use the internal network structure to produce explanations. For images this would result in *saliency maps* or what is knowns as heatmaps. In contrast, LIME and SHAP performs transforms of the inputs and observe the output without relying on internal calculations in the network and are therefore model agnostic.

3.1 Requisites for interpretable explanations

Establishing a foundation for what is required from an explanation is essential before diving into some suggested XAI methods. A simple yes or no answer may be sufficient to reason for a prediction in some applications like random forest classifiers. However, in application requiring more sophisticated approaches, such as natural language processing, computer vision and robotics, binary explanations is of limited value. Instead, a visualization of where important features are present or their structure could be more relevant. This principle is illustrated in Figure 3.1. Since an explicit metric of explainability currently is unknown in the AI literature, most XAI methods are based on the researchers intuition

of a good explanation. This could limit the advancement of these methods as understandings from explanations varies greatly with the depth of knowledge in these systems. For this reason, Miller [22] argues that multiple viable strategies to attain interpretable explanations in AI should be built with consideration from known research in philosophy, psychology, and cognitive sciences. From these fields it is known that people exert biases, social expectations and explanation selection whenever reasoning is presented. Miller pinpoints four major findings in his review which he claims that researchers and practitioners currently are unaware of:

1. Explanations should explain why event A happened instead of B. People seldom ask why event A occurred, but rather seek reasons for it over another case. Looking at Figure 3.1 as an example, then this could be an understanding of why the model predicted a frog as opposed to a flower or some other animal.

2. Selective explanations induce biases. Humans select one or two reasons for an event from a pool of infinite possible causes. It is therefore, subconsciously, rarely expected a complete explanation for an event. This has the unavoidable effect of inducing cognitive biases whenever a reason is selected and being presented.

3. Statistical relationships are ineffective at delivering an explanation to humans. While they do have some value, causal relations provide more meaningful reasoning compared to probabilities. The net takeaway from this point is the accompanied value of using both at the same time and avoid likelihoods alone.

4. An explanation is a conversational transfer of knowledge presented in belief of the recipient's view. This implies that an explanation is a social interaction conveyed through a communicative medium, that being e.g. an image, natural language, body language, text etc. or any combination of these.

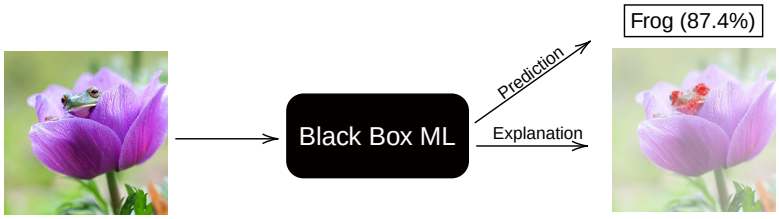


Figure 3.1: Illustration showcasing an interpretable explanation of a black box prediction. The learned features of the frog is highlighted by a heatmap, visualizing the importance of its head, eyes and colour. The expert may asserts higher trust to the classifier based upon the explanation.

Applying practices inferred from these findings will likely increase the explanation value for the layman. This is essential for societal trust to future AI driven systems as it's taking over evermore tasks in the industry, infrastructure, transportation, medicine and ultimately peoples daily lives. If - or for that matter when - AI systems are handed these responsibilities, it is vital that their decisions are deeply understood such that the public opinion

maintains trust even in the off chance of failure. The XAI methods presented next aims to open the black box models and lay the foundation for future applications.

3.2 Sensitivity Analysis - SA

Sensitivity Analysis (SA) is a simple method for explainability with traces back to at least the 60's [11]. The author shows how it's possible to explain the importance of each feature by an iterative sweep across inputs and measure the change in output. This laid the foundation of the SA discussed in [30] where it is applied to network predictions on images, text and video. Other works have applied SA to visualize different model explanations, such as in [27] where the authors use SA on a model trained on functional magnetic resonance (fMRI) data. The authors show how SA is a flexible and computationally efficient tool to visualize nonlinear kernel models[1] in neuroimaging. SA measures the output importance of each input, i, by the relationship

$$R_i = ||\frac{\partial}{\partial x_i} f(\boldsymbol{x})||. \tag{3.1}$$

Sensitivity Analysis, as a means to explanation, assumes that the most important input features are the output prediction most sensitive to. Contrary to other explainable methods, it looks at the change in output rather than the output value itself $f(\boldsymbol{x})$. This could lead to mediocre or, in the worst case, misleading explanations. For example on images, using Sensitivity Analysis produces a heatmap pointing at pixels the model prediction is sensitive to. The misleading explanations appears on images where the ground truth class is somewhat occluded. Using Figure 3.1 as an example, one would, according to Sensitivity Analysis, expect a large sensitivity around the visible parts of the frog's head. However, over the pixels where the flower petal hides the body of the frog, one could reconstruct its body by altering the pixels in a particular way. This would potentially increase the confidence for the prediction and Sensitivity Analysis might potentially. Picking up on this, Sensitivity Analysis could suggest that the petal is more important for increasing the prediction than the frog itself. An example of SA is also shown in Figure 3.2 from the paper [30]. Here both SA and LRP (introduced in next session) are compared in the task image classification. SA provides some explanation value, but is inferior to methods such as LRP when Miller's points from Section 3.1 are used as the comparison baseline. Figure 3.2 compares the two methods and a brief discussion based on Miller's points are discussed in the caption.

3.3 Layer-wise Relevance Propagation - LRP

Layer-wise Relevance Propagation, or LRP, is a method for decomposition of a classifier proposed in [4]. The authors have provided a toolbox available at http://www.

[1] Some kernel models include Support Vector Machine, nonlinear kernel logistic regression and kernel Fisher discriminant.

explain-ai.org/ with open source implementation of the method. LRP aims to understand the contribution by each individual input (e.g. pixel in an image or a word in a paragraph), x, to the prediction $f(\mathbf{x})$. This may either be a positive or a negative contribution. The classification function, f, is assumed to be positive and real-valued. This is a valid assumption as most classifiers use the softmax function[2] at their final layer, outputting a type of confidence probability. The technique explains the prediction by redistribution of the prediction $f(\mathbf{x})$ backwards with a rule for relevance score R_i to each input. The redistribution technique obeys what the authors refers to as *relevance conservation* summarized from [30] with the equation

$$\sum_i R_i = \sum_j R_j = ... = f(\mathbf{x}). \tag{3.2}$$

Equation (3.2) states, in each layer of the network, that the total relevance is constant. This means that it's only redistributed amongst the nodes in between each layer.

Finally, when the relevance score for each input is computed, then each R_i quantifies how much the input variable contributes to the final prediction $f(\mathbf{x})$.

The redistribution function is dependant on which network architecture that is used. For regular neural networks this is simply performed backwards from layer $l+1$ to layer l with

$$R_j = \sum_k \frac{x_j w_{jk}}{\sum_j x_j w_{jk} + \epsilon} R_k \tag{3.3}$$

where ϵ is a term added to prevent division by zero. In short, the relevance is redistributed by the weighted input $x_j w_{jk}$ from layer l to $l+1$ where x_j is the neuron activation and w_{jk} is the connection strength. [4] also proposes the *alpha-beta* rule as an alternative to the simplest form presented in Equation (3.3). The *alpha-beta* rule is defined as

$$R_j = \sum_k \left(\alpha \frac{(x_j w_{jk})^+}{\sum_j (x_j w_{jk})^+} - \beta \frac{(x_j w_{jk})^-}{\sum_j (x_j w_{jk})^-} \right) R_k \tag{3.4}$$

with $^+$ and $^-$ denoting positive and negative parts of the expression. Additionally, another constraint is applied to enforce the relevance conservation: $\alpha - \beta = 1$.

Other redistribution functions have been proposed for different architectures, but are outside of the scope for this project. Further literature covers these and are summarized in [29]. With the proposed redistribution functions, the LRP method is extended to a multitude of possible use cases other than regular feed forward neural networks. Meanwhile, LRP was recently criticized for a slow performance when applied to real-time video and it was therefore proposed, in the same paper, a method orders of magnitude faster with similar visualization result as LRP, namely VisualBackProp [6].

[2] Softmax has the property of transforming the input to positive values with a total sum of unity.

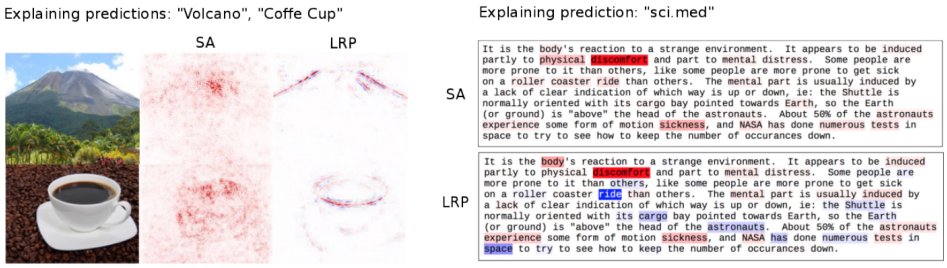


Figure 3.2: Left: Figure illustrating the explanation of a class prediction based on an image using Sensitivity Analysis and Layer-wise Relevance Propagation. Right: Another application of SA and LRP to explain text classification. The model is a black-box in both examples, and both methods provide an explanation of how the model perceives the data. Unlike SA, LRP is able to highlight both negative and positive contributions towards the prediction. Applying Miller's findings for explanations introduced in Section 3.1, LRP is superior to SA because it highlights both positive and negative reasons in its explanation, rather than only emphasizing what the output is sensitive to. The issues presented as point number two are addressed in LRP by providing two explanation, reason for and againts, rather than a single reasoning in SA, thereby reducing a biased selective explanation. Figure source [30].

3.4 Integrated Gradients

Integrated Gradients is a technique based on similar principles as SA (Section 3.2), but instead of calculating a gradient based on only the specific input, Integrated Gradients accumulates gradients in a linear path going from a baseline x' input to the input at hand, x, finally finding the average output gradient. In the original paper [35], the authors define a baseline vector $x' \in \mathbb{R}^n$ being for instance a black picture for an image model or a zero vector for a text network. Reasoning for a baseline stems from the human intuition of assigning cause for an outcome by comparing to the absence of an event. This is exactly what was pinpointed in point 1 in Section 3.1 as being essential for an interpretable explanation. While it is possible to define an image consisting of noise as the baseline, it does not preserve the human intuition of absence like the black image. This choice of baseline selection also contributes towards detecting adversarial perturbations, unlike a noisy baseline. Further, $x \in \mathbb{R}^n$ is the input with $\frac{\partial f(x)}{\partial x_i}$ being the gradient of the output prediction $f(x)$ along the i^{th} dimension. Finally, the formula is defined as

$$\text{IntegratedGrads}_i(x) := (x_i - x_i') \int_{\alpha=0}^{1} \frac{\partial f(x' + \alpha(x - x'))}{\partial x_i} d\alpha \tag{3.5}$$

where the chosen path, $x' + \alpha(x - x'), \alpha \in [0, 1]$, is a straight line between the baseline x' and the input x. The authors also discusses taking other curved paths, but concludes that a straight line should be used as it's the simplest mathematical path while also preserving symmetry[3].

[3]This is discussed in 4.2 in [35].

An example of Integrated Gradients is illustrated in Figure 3.3 where the model has to identify the type of expected answer. Since the model works with text data, the baseline used by Integrated Gradients is the zero embedding vector. Intuitive attributes such as 'how many', 'difference' and 'total number' are identified as triggers towards the 'Numeric' class. Notice in the last question how the name 'Charles' is emphasized as a feature for a simple 'Yes/No' question. This is an undesirable trait of the model and the expert may conclude that the model is unreliable towards this class even though the prediction is correct. A more reasonable attribute would be to rely on the word 'did' which intuitively expects a binary True/False or Yes/No answer. The expert may need to feature engineer the data, collect more samples and retrain the model to increase the trust.

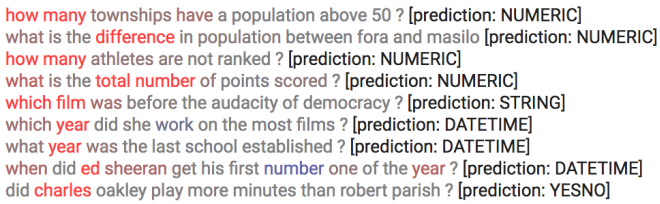


Figure 3.3: Integrated Gradients applied to a question classification model. The explanation highlights positive (red), negative (blue) and neutral (grey) attribution strength. Figure source [35].

Finally, the authors claim simplicity in the implementation of Integrated Gradients as it only needs a few calls to the standard gradient operator. This is confirmed in another paper published in 2018 [2] (in its section 3, Listing 1). One of the limitations of Integrated Gradients, argues [3], is the high computational cost by evaluating the average gradient by numerical integration several times with slightly different inputs. This is especially heavy for large models, and should be taken into account whenever explanation techniques for more complex models are considered.

3.5 LIME

LIME [36], short for Local Interpretable Model-Agnostic Explanations, is an explanation framework created to guide experts on the behaviour of a trained black box model. It aims to answer the question of "why should I trust the model?" by finding an interpretable model that is locally reliable to the classifier. The framework is open sourced and available at the lead author's GitHub [37]. Summarized, LIME approximates the black-box model around an input by assuming that the input space is locally accurate to the classifier. This means that the explanation are made at the individual level.

A vector $x \in R^d$ represents the unaltered input while $x' \in \{0,1\}^{d'}$ symbolizes a binary vector for the explainable representation. This could for instance be whether a set of pixels are present or not. An explanation is defined as a model $g \in G$, where G denotes all possible interpretable models. The model g could for instance be a linear model, decision tree or a falling rule list [38] with "IF-THEN" statements. This implies that it needs

to facilitate simple, interpretable visual or textual explanations. Furthermore, as not all possible explanation models g are simple enough for human interpretability, the authors introduce $\Omega(g)$ to be a measure of complexity of the explanation model. As an example, for a decision rule, $\Omega(g)$ could be the number of statements needed in a IF-THEN rule or the number of non-zero weights for a linear model. The fewer statements or non-zero weights needed, the simpler the model and less penalty is introduced by the $\Omega(g)$ term. The classifier being explained, $f : \mathbb{R}^d \to \mathbb{R}$, outputs the probability $f(x)$ that an input x belongs to a specific class[4]. z is a sample of x and $\pi_x(z)$ is used as a measure of proximity between z to x. The locality in LIME is obtained through the weighting term π_x. Finally, $\mathcal{L}(f, g, \pi_x)$ is defined as a measure of how inaccurate g approximates f. This could for instance be a distance measure between f and g. LIME tries to minimize the objective function stated as

$$\xi(x) = \underset{g \in G}{\operatorname{argmin}} \mathcal{L}(f, g, \pi_x) + \Omega(g) \tag{3.6}$$

in order to obtain a valid local approximation of f while reducing the complexity of g to ensure human interpretability. LIME is model-agnostic, meaning that the explanation is separated from the choice of model. The objective function in Equation (3.6) establishes this foundation by conveniently allowing individual choices for $G, \mathcal{L}$ and Ω. The term $\mathcal{L}(f, g, \pi_x)$ is approximated, since it should be independent of f, by drawing non-zero elements of x' uniformly at random. A such perturbed sample $z' \in \{0, 1\}^{d'}$ is passed through the model to obtain $f(z)$, which is the prediction probability of z' belonging to the specified class. Given enough perturbed samples, Equation (3.6) is optimized to get an explanation of x, namely $\xi(x)$.

Linear Explanations

This project bases upon the authors implementation of LIME for images[5], which utilizes a linear explanation model $g \in G$, such that $g(z') = w_g \cdot z'$. The square loss, also known as quadratic loss, is used for $\mathcal{L}$ and $\pi_x = \exp\left(-\frac{D(x,z)^2}{\sigma^2}\right)$ is the measure weighting the proximity between z and x where D is a distance function. The D is defined in the code as the l^2 norm, with $\sigma = 0.75 \cdot \sqrt{\text{number of columns}}$. The term is finally set to

$$\mathcal{L}(f, g, \pi_x) = \sum_{z, z' \in Z} \pi_x(x)(f(z) - g(z'))^2. \tag{3.7}$$

The model complexity penalization term Ω is, for images, based on "superpixels" found by segmentation techniques. The authors provide a default segmentation technique, "quick-shift", but it is possible to provide others. This is explored further in Section 4.5. The

[4]This is slightly different from conventional notation where f is defined with k-outputs, $f : \mathbb{R}^d \to \mathbb{R}^k$. The author rather defines $f(x)$ to be the prediction probability of chosen class instead, simplifying the notation in the process.

[5]The authors implementation also support explanations of tabular data and text models.

interpretable representation $x' \in \{0,1\}^{d'}$ indicates 1 for superpixels present in the original image x, and 0 for removed superpixels displayed in grey. Ω is approximated using K features from the image, and finding the weights w with a least squares regression (Lasso[6]). Algorithm 1 finds the weights w_g used in the linear model g and is stated as

Algorithm 1 Sparse Linear Explanations using LIME — *Source: [36]*

Require: Classifier f, Number of samples N
Require: Instance x and its interpretable version x'
Require: Similarity kernel π_x and the length of explanation K

```
Z ← {}
for i ∈ {1, 2, ..., N} do
    z'_i ← sample_around(x')
    Z ← Z ∪ < z'_i, f(z_i), π_x(z_i) >
end for
w ← K − Lasso(Z, K)          ▷ using z'_i as features, f(z) as target
return w
```

(a) Original Image (b) Explaining *Electric guitar* (c) Explaining *Acoustic guitar* (d) Explaining *Labrador*

Figure 3.4: Linear LIME applied to an image passed to Google's Inception neural network. LIME highlights superpixels which contribute towards the class being explained. Figure source [36]

3.6 SHapley Additive exPlanations - SHAP

SHapley Additive exPlanations or SHAP for short, is a proposed framework to interpret predictions provided by a model [20]. It is a game theoretic approach to explain any black box model, and is therefore model agnostic. The authors have open sourced their work and published it freely available at GitHub [21]. The authors propose SHAP values, similar to Shapley values, as a unified measure of feature importance.

3.6.1 Shapley values

A brief introduction to Shapley values is given as it forms the basis on which the SHAP method is built upon. The Shapley values were introduced by Lloyd S. Shapley in 1953

[6] https://scikit-learn.org/stable/modules/linear_model.html#lasso

[31] as a concept in cooperative game theory. A Shapley value is assigned to each player in a game stating how important they are in the cooperation to a surplus. It provides one way of fairly distributing the payout to players based on their contribution in the game. The coalition game is described using the set N of n players. Let a coalition of players S be a subset of N, i.e $S \subseteq N$, and define a payout function v, describing the expected payout to each possible coalition. In a coalition game (v, N), the amount player i collects is the Shapley value

$$\phi_i(v) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} [v(S \cup \{i\}) - v(S)]. \tag{3.8}$$

The term $[v(S \cup \{i\}) - v(S)]$ can be interpreted as calculating the payout to a coalition with player i minus the payout without player i. The term is weighted by $\frac{|S|!(|N|-|S|-1)!}{|N|!}$ and together with the sum $\sum_{S \subseteq N \setminus \{i\}}$, represents, for each player, the average contribution over different possible permutations in which the particular coalition can be formed. Summarized, the Shapley value for player i (Equation (3.8)) states the importance of the player by comparing the payout with and without them in a coalition of players. Since the order in which the players contributes may affect the payout, the contribution is calculated with all possible permutations of the coalition, across all possible coalitions.

The Shapley value obeys properties important to obtain a fair and unique distribution. It is also the only attribution method that satisfies these desirable properties [13]. The original paper [31] states these properties in the definition as *symmetry*, *efficiency* and *law of aggregation* (=linearity). Finally, the property of a dummy or null player is explicitly stated in the paper's definition section. These properties are

Symmetry

If two players are equal and contribute the same, then they receive equal payout. Formally, if $v(S \cup \{i\}) = v(S \cup \{j\})$ then $\phi_i(v) = \phi_j(v)$.

Efficiency

The sum of the Shapley values of all players equal the payout for the total coalition

$$\sum_N \phi_i(v) = v(N) \tag{3.9}$$

Law of aggregation

If two independent games are combined, then the payout equal the payout sum for each individual game

$$\phi_i(v + w) = \phi_i(v) + \phi_i(w) \tag{3.10}$$

Null player

A player that does not contribute to the payout receives no payout. If $v(S \cup \{i\}) = v(S)$ for all coalitions S without i, then $\phi_i(v) = 0$

3.6.2 SHAP Method overview

Even though the concept of Shapley values was proposed in the field of cooperative game theory, it may be used as a means to explain a prediction. By assuming that each feature is a player and the prediction is the total payout, then the Shapley value tells how much each individual feature contributed towards the prediction. Unfortunately, calculating the Shapley value going through all possible combinations of features becomes computationally infeasible. Computing the marginal contribution of every feature to every coalition is $O(2^{|N|})$ [13]. Assuming each pixel in a 32x32 image is a feature, this results in 1024 features, giving $2^{1024} \approx 1.79x10^{308}$ possible combinations[7], making the direct calculation infeasible. Approximations using Monte Carlo sampling has been proposed in [] Strumbelj et al. (2014)., Shapley sampling value estimation, Quantitative Input Influence also similar approximations, SHAP values are based on the Shapley values and obeys their properties, adding strong mathematical theory behind it. The authors propose a model agnostic method to obtain the SHAP values, namely Kernel SHAP.

Kernel SHAP

Kernel SHAP is presented in the SHAP paper[20] as model agnostic approximation method to obtain Shapley values. It has similar accuracy as other Shapley estimation techniques like Shapley sampling values and Quantitative Input Influence. Kernel SHAP uses eq. (3.6) to obtain the Shapley values, but unlike LIME, avoids heuristically choosing the parameters for loss function $\mathcal{L}$, weighting kernel π_x and regularization term Ω. Rather, they are chosen such that the properties of Shapley values are retained. These are shown to be

$$\Omega(g) = 0, \tag{3.11}$$

$$\pi'_x(z') = \frac{(M-1)}{(M \text{ choose } |z'|)|z'|(M-|z'|)}, \tag{3.12}$$

$$\mathcal{L}(f, g, \pi_x) = \sum_{z,z' \in Z} (f(h_x(z')) - g(z'))^2 \pi'_x(z'). \tag{3.13}$$

M is the maximum number of features in a coalition, being the size of set N, $|N|$, in the original description of Shapley values. M choose $|z'|$ is the binomial coefficient $\binom{M}{|z'|} = \frac{M!}{|z'|!(M-|z'|)!}$. The g in Equation (3.13). The kernel π'_x in Kernel SHAP essentially merges the theory of Shapley values with the model agnostic approach of LIME. Even though Kernel SHAP is model agnostic like LIME, it is unfortunately slow. Deep SHAP is presented

[7] In comparison, the commonly stated number of atoms in the Universe is around 10^{80}.

in the paper to utilize the characteristics of deep neural networks to enhance computational performance.

Deep SHAP

Deep SHAP was developed to be a faster model specific approximation method to the SHAP values. It is faster, but only approximate and is based on connections between SHAP and another explanation technique, DeepLIFT proposed in [32]. DeepLIFT decomposes the output prediction by backpropagating the contributions of all neurons to the input, similar to LRP. What mainly differs DeepLIFT and LRP is that it uses a different redistribution function. The details are further described in [32] and in [2]. DeepSHAP modifies the redistribution function to include the SHAP values, leading to effective computation. The method requires background samples to approximate the expected output of the model. The expected output is used to approximate the SHAP values. The found SHAP values sum up to the difference between the expected model output and the current model output. This adds some confusion to what the SHAP values represent as it is not the difference of the predicted value after removing the feature. Further details of Deep SHAP are described in [20]. An example of a plot using Deep SHAP is shown in Figure 3.5. An interesting observation in this explanation is that the absence of pixels on top of the 'four' is a strong indicator of being four rather than nine. The lack of pixels in the middle of 'zero' is important for the explanation of the class. This is likely the case because the other digits usually have pixels in the middle of the image

Figure 3.5: A plot of SHAP values using Deep SHAP on a model trained on the MNIST dataset. The red pixels increase the output while the blue decrease the output. Figure source [21]

Chapter 4

Experiments and Results

The chapter walks through the training of a convolutional neural network on a labeled image dataset in Section 4.1 and Section 4.2. Having trained a model, Section 4.3 contains an analysis of its performance. Section 4.4 visualizes the feature maps in between layers to showcase the abstract features learned by the model. Section 4.5 and Section 4.6 shows how the XAI methods perform on the model followed by a comparison and discussion in Section 4.7. Finally, Section 4.8 showcases how data augmentation based on model explanations improve the model training. Note that the terms *model*, *system*, *network* and *the classifier* are used interchangeably throughout Chapter 4 referring to the trained deep neural network in the chapter.

4.1 CIFAR-10

In order to demonstrate the strength of the aforementioned XAI methods in Chapter 3, a thorough experiment on the known CIFAR-10 dataset[1] is conducted. The dataset consists of 60,000 colored (RGB) images of pixel size 32x32 labeled with 10 different classes [15], [16]. The 10 labels include: plane, car, bird, cat, deer, dog, frog, horse, ship, truck and are uniformly distributed with 6,000 images each. The training dataset contains 50,000 of the images (5,000 for each class) resulting in 10,000 images ~~remaining~~ for the test set. Each class is mutually exclusive, implying no overlap. Cars, for instance, include SUVs, sedans and other small cars. The truck class only consist of big, cargo, road-driven vehicles such as trailers, concrete mixers and fire trucks. The two classes exclude pickup trucks in their set. A portion of the training set is showcased in Figure 4.1.

[1] CIFAR - Canadian Institute For Advanced Research

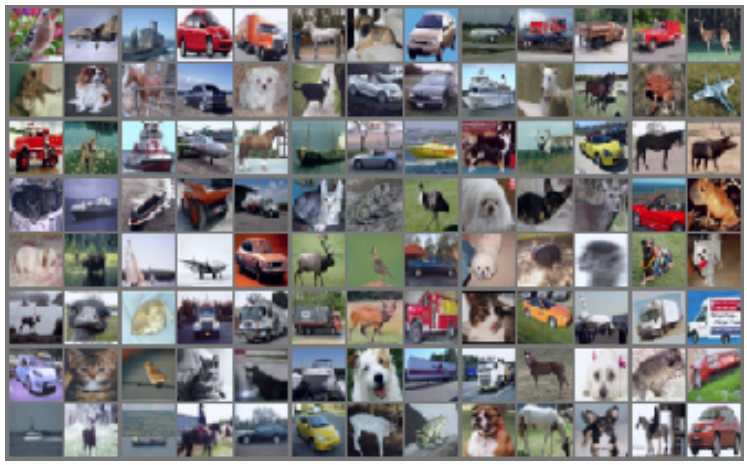


Figure 4.1: 104 arbitrarily collected training images from the CIFAR-10 dataset. The resolution of each individual image in the figure is somewhat degraded.

4.2 Model Architecture and Training

A simple neural network structure is used to ease the visualization of the neural network in Section 4.4. The implementation of the model is based on a starter code provided in the NTNU course TDT4265[2] and further functionality is built upon it. The architecture of the model trained on the dataset consist of a 6-layered CNN followed by a fully connected neural network with a single hidden layer. Each of the 6 convolution layers uses a kernel size of 5x5 with batch normalization. Convolutional layer number 2, 4 and 6 also includes a 2x2 max pooling and dropout. The random dropout of weights aims to restrain the network from overfitting, resulting in better generalization. The downsampling by max-pooling helps the network with invariance, meaning that features detected are invariant to small translations of the input [9]. If an image is ever so slightly shifted, then most of the output from pooled outputs do not change. This is, like other methods, to improve the generalization. The 6th and final convolution layer is connected to a dense neural network with 64 hidden neurons. The 64 hidden neurons are finally connected to 10 output neurons each one representing a class. This is often referred to as a one-hot encoded output, discussed in Section 2.3. The last layer utilizes softmax, described in Section 2.3, to normalize the output sum to 1, such that each output can be interpreted as a percentage of confidence. Table 4.1 summarizes the architecture in detail and a visualization of the

[2] https://github.com/hukkelas/TDT4265-A3-starter-code

network is depicted in Figure 4.2.

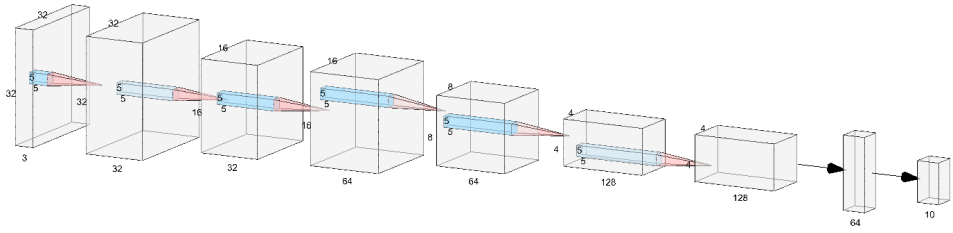


Figure 4.2: The model architecture visualized. The leftmost cuboid depicts an input image of 32x32 pixels with a depth of 3 being the rgb values. Six cubes depicts the CNN layers and the last two vertical cuboids represents the dense neural network. The sixth (and final) layer in the CNN has $128 * 4 * 4 = 2048$ datapoints. These are connected to the hidden layer with 64 neurons which again are connected to the 10 output neurons. Figure made with the NN-SVG software [19]

The 50,000 training images are further randomly split 10% to a validation set of 5,000 images with the remaining 90% left in the training set. This split allows the training to be stopped when the training loss consistently decreases while the validation loss increases. This is, as mentioned in Section 2.6, a sign of overfitting and early stopping aims to prevent the phenomena. The model is trained using the described architecture in Table 4.1 with a learning rate set to 0.02 and early stop epoch count of 3. Figure 4.3 shows the training procedure. The test training loss and accuracy is added to the figure to show how the validation set is able to estimate the model performance on the test data. Note that the model only updates its weights with the training data. In summary, the model is trained on the training data, uses the validation data to early stop and reports final accuracy using the test data.

Layer	Filter size Nx(DxWxH)	Output DxWxH
InputImage	-	3x32x32
Conv2D	32x(3x5x5 kernel)	32x32x32
BatchNorm2D	-	32x32x32
LeakyRelu	-	32x32x32
Conv2D	32x(32x5x5 kernel)	32x32x32
MaxPool2D	32x2x2 kernel	32x16x16
BatchNorm2D	-	32x16x16
Dropout $(p = 0.2)$	-	32x16x16
LeakyRelu	-	32x16x16
Conv2D	64x(32x5x5 kernel)	64x16x16
BatchNorm2D	-	64x16x16
LeakyRelu	-	64x16x16
Conv2D	64x(64x5x5 kernel)	64x16x16
MaxPool2D	64x2x2	64x8x8
BatchNorm2D	-	64x8x8
Dropout $(p = 0.3)$	-	64x8x8
LeakyRelu	-	64x8x8
Conv2D	128x(64x5x5 kernel)	128x8x8
BatchNorm2D	-	128x8x8
Conv2D	128x(128x5x5 kernel)	128x8x8
MaxPool2D	128x2x2	128x4x4
BatchNorm2D	-	128x4x4
Dropout $(p = 0.4)$	-	128x4x4
LeakyRelu	-	128x4x4
Reshape	-	2048
FullyConnected	-	64
BatchNorm1D	-	64
Relu	-	64
FullyConnected	-	10
SoftMax	-	10

Table 4.1: The full model architecture used in the training phase. Each consecutive output is the input to the next step. The model is set in evaluation mode, removing dropout, when the test dataset is employed. A visualization of the layers is presented in Figure 4.2. It is debatable whether the activation function (here LeakyRelu $\alpha = 0.01$ presented in Section 2.2) should be applied before or after the batch normalization. In the paper proposing batch normalization, the activation function was applied after the normalization and is followingly in the experiment [12]. Though newer experimentation in the area show that it is slightly better to use activation function afterwards.

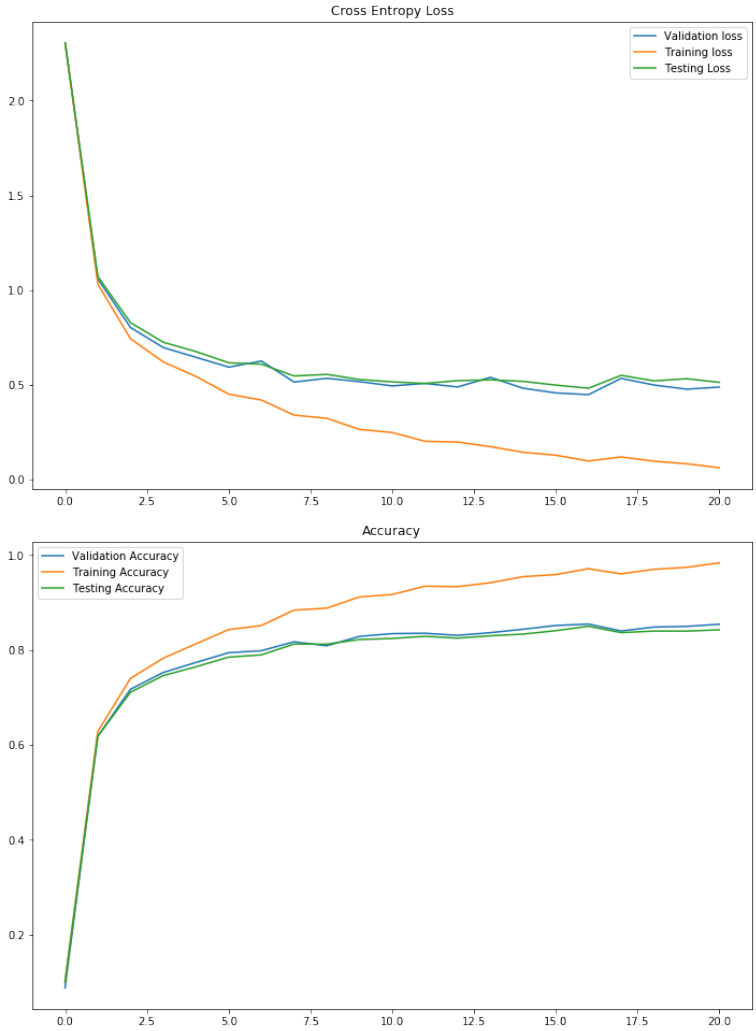


Figure 4.3: The loss and accuracy plotted over epoch count. An epoch contains a full training iteration across the whole training data. The validation is used for finding optimal hyperparameters with an early stop. Finally, the model achieves an accuracy over the test set at 84.20%.

4.3 Performance analysis

The accuracy across the 10,000 test images for each class is listed in Table 4.2 with an average of 84.20%. The expected prediction accuracy over 10 classes using random guessing is 10%, signifying that the network has learned features to distinct the images from each other. Observing Table 4.2, Cat, Dog and Bird suffers from lowest prediction accuracy with 72.0%, 76.0% and 76.9% respectively. This could indicate for instance either that the

training and test data are vastly different for these classes or alternatively that these classes have similar features, somehow confusing the model.

Next, a confusion matrix is used. The confusion matrix is a metric used to attain an overview of the model's performance across all predictions. It visualizes which classes that confuses the model and summarizes the performance even in cases where the dataset is imbalanced. Using only the accuracy as a metric would result in an average accuracy, while using a confusion matrix showcases which classes perform better than others, highlighting inaccuracies in the model likely occurring from dataset imbalances. Correct predictions are reported along the diagonal. The values are either a count or a normalized percentage of predictions over the ground truth label. Looking at Figure 4.4, it appears that most errors arise from predicting dogs as the wrong label cats, 14.3%, and vice versa, 11.5%. Not unexpectedly, the model struggles in some instances to distinguish between cars and trucks, reporting 4.8% trucks as cars, and 4.5% of cars as trucks. Furthermore, 4.5% ships are reported as planes, 4.1% birds are mistaken as planes but only 3.0% ships are stated as planes. Supringsly, the model seems to have found features where horses and deers are vastly different with only deers predicted as horses 1.9% and horses predicted as deers 2.7%. For instance, deers are predicted as cats 5.8%, frogs 4.0% and birds 3.9%, all higher than the 1.9% predicted as horses. From the perspective of humans, it seems rather strange to predict cats, frogs and even birds rather than a horse in the case where the true category is a deer. The argument holds the other way around by the same analysis on the true class of horse. 4.1% of the horses are reported as dogs, 3.1% as cats, 2.7% as deers but only 1.6%, 0.8% and 0.3% as birds, planes and frogs respectively. Again, the model predicts some other smaller animal, here dogs and cats, over the horses. Finally, inferring from the analysis of the confusion matrix is that the features found by the model clusters the classes in regions where animals are close to each other and the vehicles are clustered in another region, separating the two superclasses.

Class	**Test accuracy**
Plane	85.8%
Car	93.0 %
Bird	76.9 %
Cat	72.0 %
Deer	80.4 %
Dog	76.0 %
Frog	89.7 %
Horse	87.0%
Ship	92.5 %
Truck	88.9 %

Table 4.2: Data showcasing the accuracy over each individual class. Finally, the model achieves an average accuracy of 84.20%

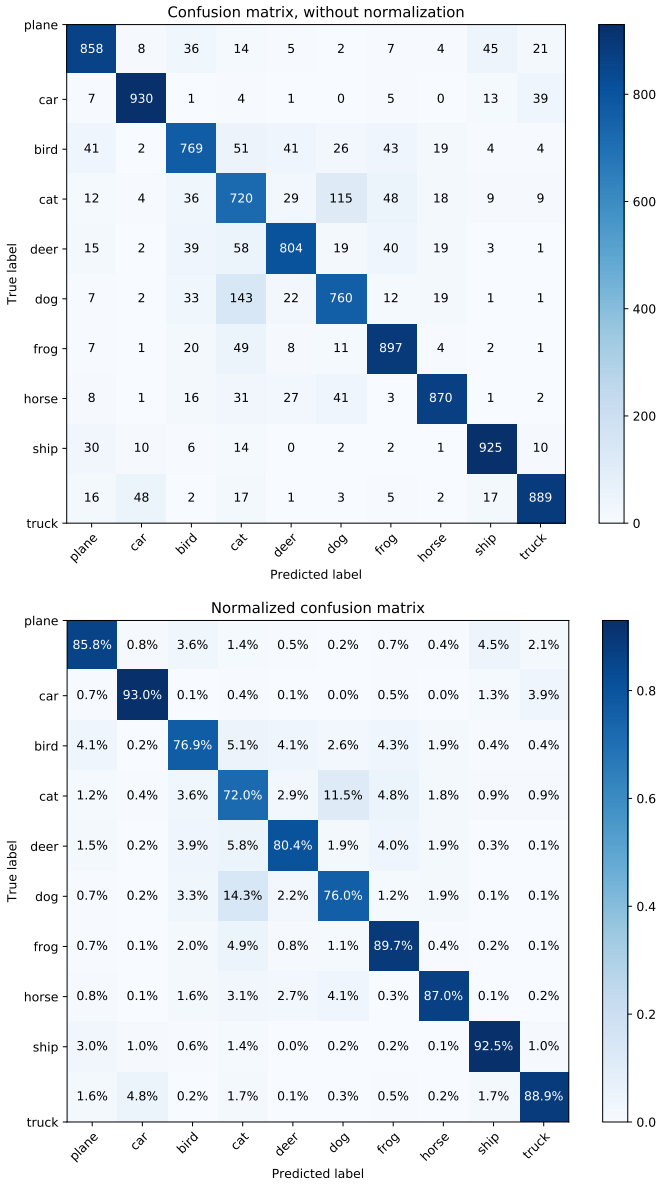


Figure 4.4: Confusion matrix without and with normalization, respectively. The figure is plotted by slightly altering the scikit-learn plot_confusion_matrix() function.

4.4 Visualizing the Convolutional Neural Network

The analysis of the model presented in the previous section provides some understanding of the network performance. However, the performance metrics do not address how the model interprets and reaches a prediction, thereby providing little to no explanation of its internal workings. A simple method to visualize the model's understanding of input data is to pass an image and observe activation at individual layers throughout the network. Another method is to generate an image which maximizes the class score, as the authors do in the paper *Deep Inside Convolutional Networks* [33]. The first approach is performed in this section.

A single image labeled as a plane, seen in Figure 4.5, is passed through the first convolutional layer. The convolution operation is performed with kernels of size 5x5 as described in Table 4.1. The first layer produces 32 feature maps, or activation maps, and are portrayed in Figure 4.6. Figure 4.7 shows the feature maps for layer 2, Figure 4.8 for layer 3 and 4, Figure 4.9 for layer 5 and Figure 4.10 for the 6th and final CNN layer. Visualization of the values over the fully connected layer towards the output are omitted.

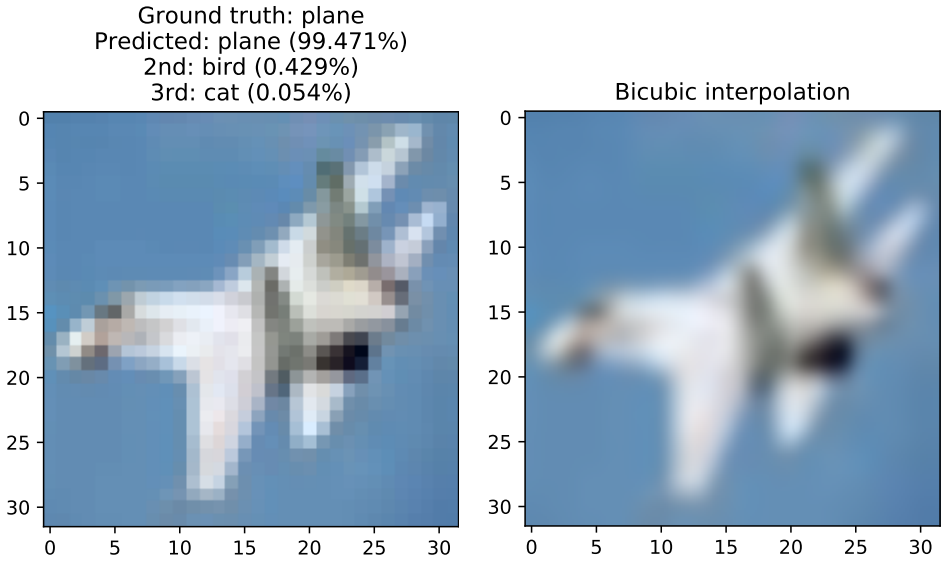


Figure 4.5: To the left: the input image to the trained network. The network classifies this correctly as an airplane with 99.471% confidence. The 2nd and 3rd class predictions are given as 'bird' and 'cat'. To the right: the same plane image displayed using bicubic interpolation to enhance the visibility.

The 32 feature maps in Figure 4.6 are all a transformation of the input image, but what kind of transformation they represent is not entirely clear. The attributes and traits each map is picking up on up may be understood as an edge or feature detection. However, it is not explicitly clear what the network is doing. Going further into the network and looking at layer 2, 3 and 4 in Figure 4.7 and Figure 4.8, less and less of the plane is visible. At

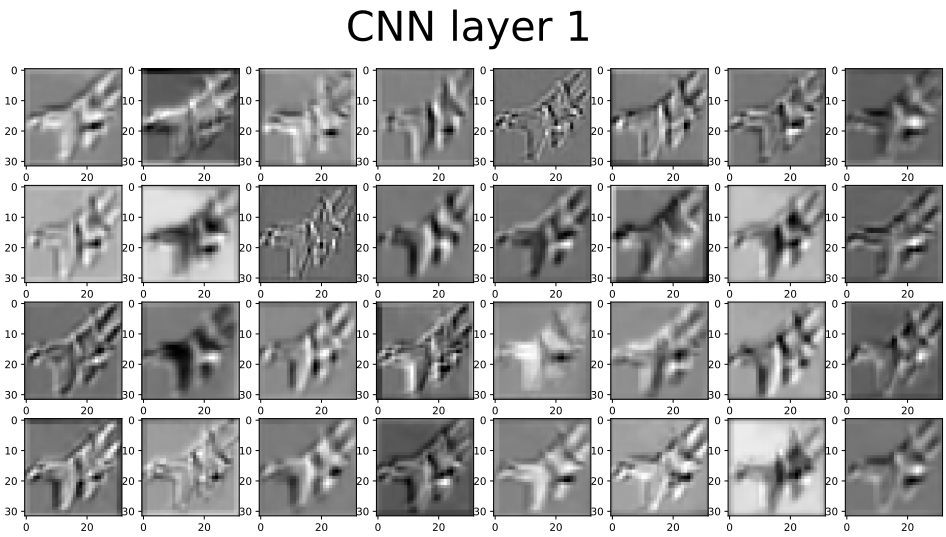


Figure 4.6: The resulting 32 feature maps after the first convolutional layer. As the image is passed in as a 3-layered RGB picture, each feature map produces a transformed output, here visualized in grayscale.

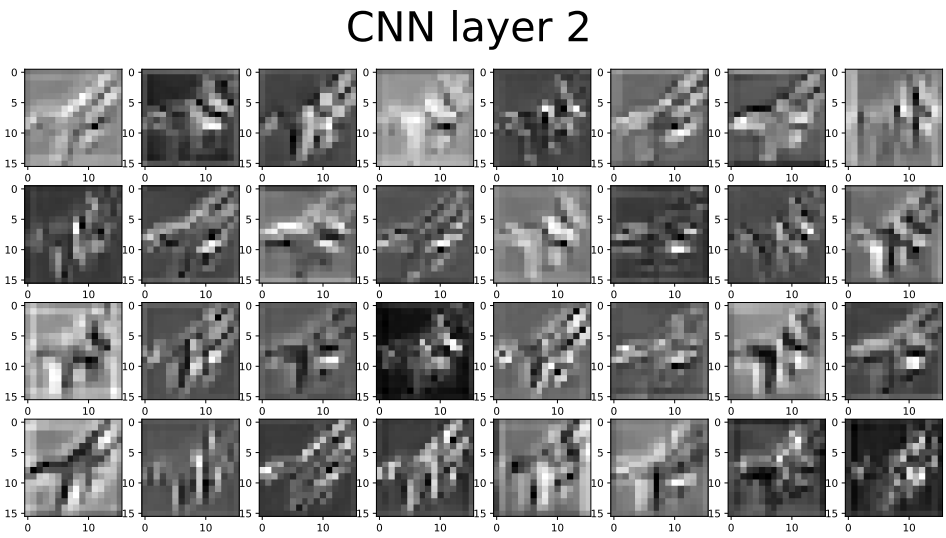


Figure 4.7: The feature maps in the second layer.

layer 4 there appears to be blobs of high activation (in white) at different positions in each feature map.

Analyzing the final layer transformation before the fully connected neural network may

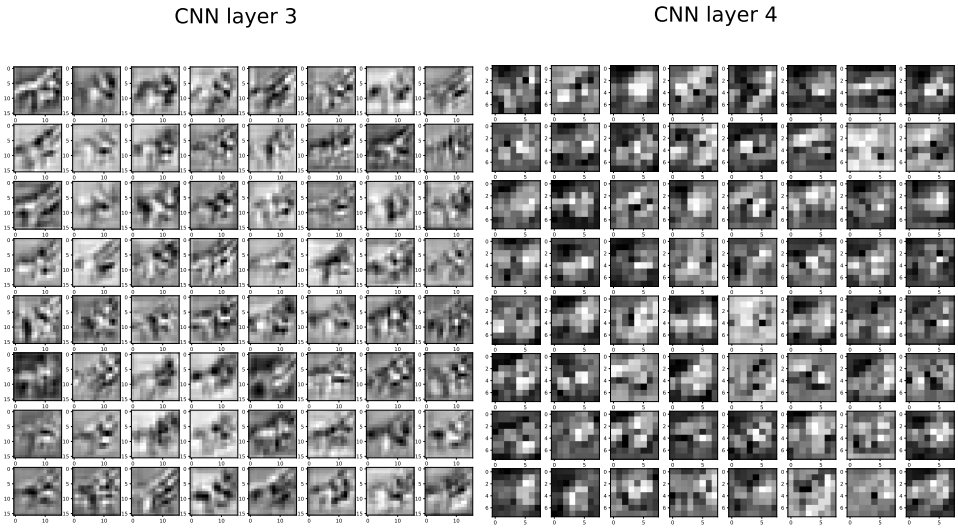


Figure 4.8: The feature maps of the third and fourth layers. The features appears more abstract as the image is passed forward in the convolutional layers.

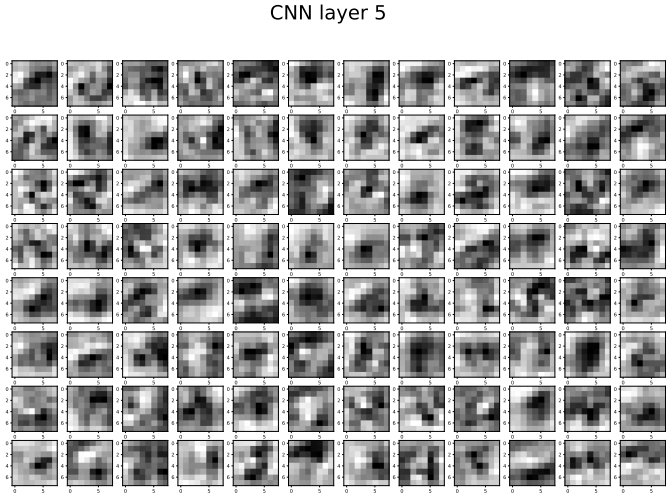


Figure 4.9: After the fifth layer, no resemblance of the original image is left.

give an understanding the model's decipherment. This is depicted in Figure 4.10. Even the last layer does not give an intuitive explanation of important features. Arguably, the final convolutional layer performs an abstract conjunction of the previous layer's output in order to classify the image. What the math behind the model indisputably expresses is that the weight space has converged values where the loss is minimized. It may therefore be argued that it is highly unlikely for humans to understand the individual transformations between

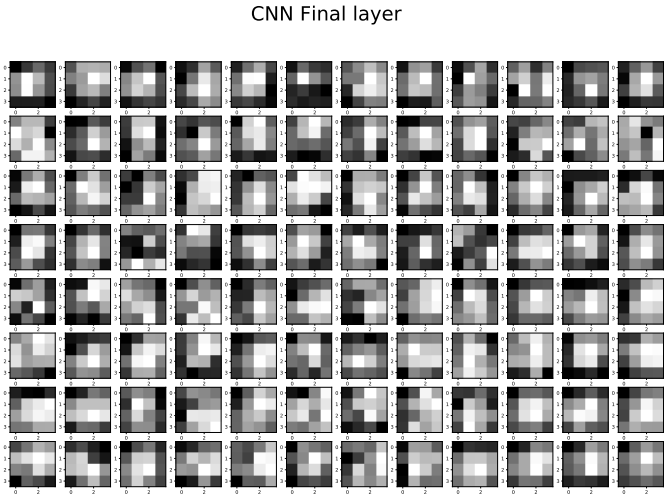


Figure 4.10: Output from the last CNN layer (layer 6) before fully connected neural network to prediction, showing how abstract the model interprets the image. The different points contain information which the final neural network uses to classify the input.

the model's layers looking at each feature map. The black box transformation performed by the network is not interpretable by human inspection and it becomes evident that a better approach to explain the model is needed. It is here the tools from XAI methods, introduced in Chapter 3 comes into play.

4.5 LIME explanation

The average test accuracy of the trained model is 84.20% as described in Section 4.3. Although the network performs fairly well, it is unclear whether it has learned what humans describes as important features or if it exploits biases in the dataset. These biases could for instance be that all dogs in the dataset are white, all cats are black or that all horses are brown. The surroundings could also be biased in the sense that all cars and trucks images have asphalt in the background while the sky is blue in airplane and boat images. To avoid these biases one may include a diverse enough dataset with all possible varieties of the given class, angles, light shading and so on. This is in practice not only a tedious task, but it might also be infeasible. Whether the dataset diversification is balanced or not, directly impacts the features picked up by the model. This imbalance is however not easily discovered, especially in datasets containing ten to hundreds of thousands different samples. An expert may discover these biases, correct for them and hopefully increase performance with help from explainable methods such as the ones discussed in Chapter 3.

One such method, the LIME framework (introduced in Section 3.5), is applied to the trained model to inspect predictions and their respective LIME explanations. Since the

method requires superpixels to find areas that impacts the prediction, it is crucial to ensure that the segmentation divides areas of fairly equal sizes. This is because LIME uses perturbed versions of the image, randomly hiding segments while observing the accuracy. If the segmented areas have disproportional sizes, small but highly important parts of the image may be present in these larger superpixels, thereby creating an illusion of the whole superpixel being of high importance. Even though a weighting of the similarity between the occluded image and its original version is performed, larger superpixels should be avoided because of the inaccuracies they may produce in the linear approximation. The linear approximation used to find the weights of g might struggle as a consequence of the few samples it is provided if only bigger areas are present. It therefore needs a balance between enough samples and also large enough samples to provide impact towards the prediction. The use of a low resolution model may struggle because of these drawbacks. The authors do not stress this in their work, but is an observation noted through this project. Six segmentation techniques are used to inspect the performance on some of the test images. Starting with an image of a horse in Figure 4.11, the different segmentations are performed on the image displayed in Figure 4.12.

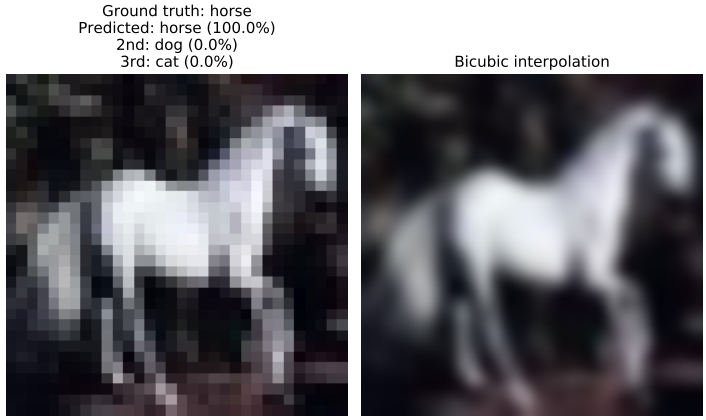


Figure 4.11: Image from test set with top three predictions. The right image is visualized using bicubic interpolation.

As seen in the figure, the default segmentation technique provided in the authors GitHub implementation, ”Quickshift”, fails to produce fair-sized superpixels. The larger patches covers vastly bigger areas compared to the smaller sized segments. The three segmentation algorithms at the bottom row in Figure 4.12, all fail to produce clear and general segmentation. Finally, Felzenszwalb and SLIC are able to produce superpixels which separate the image into regions based on edges, while preserving areas of about similar sizes. SLIC is slightly more consistent in the area sizes compared to Felzenszwalb. Different results for the segmentation is obtained by experimenting with different parameters, and the ones selected generalized well over a variety of test images[3].

[3]Further details on the segmentation techniques are available at the package documentation https://scikit-image.org/docs/dev/api/skimage.segmentation.html

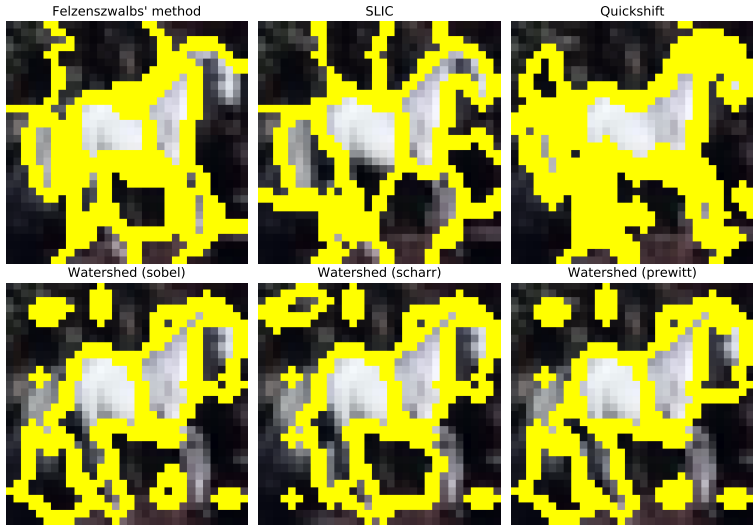


Figure 4.12: Six different segmentation techniques to showcase their performance on the image. The segmented areas become superpixels which LIME uses to build a local model. LIME uses the superpixels to perturb the image by randomly removing some of them. The three Watershed based segmentation algorithms uses three different edge detectors: Sobel, Scharr and Prewit. The segmentations are performed using the scikit-image segmentation module https://scikit-image.org/docs/dev/api/skimage.segmentation.html.

Next, the segmented image is sent through LIME where a local explainable model is approximated. This is done six times, one for each segmentation algorithm. The obtained explanation for the predicted class 'horse' is shown in Figure 4.13. Looking at the figure it becomes evident that the choice of segmentation algorithm heavily influences the explanation. "Quickshift", "Watershed (sobel)", "Watershed (scharr)" and "Watershed (prewitt)" struggles to produce a explanation, likely a result of poor segmentation. Finally Felzenszwalb and SLIC provide an explanation highlighting parts of the horse as an important feature.

Next a couple of examples using LIME with Felzenswalb and SLIC are shown. In Figure 4.15b the model gives the prediction of the class Dog with about 59.8% confidence, and explains this with both segmentation techniques by highlighting the dogs head and parts of its body. The two explanations appear consistent with each other, though some parts of the dog is highlighted slightly differently because of the different segmentation techniques. An interesting observation produced by LIME is that the model uses parts of the background as a feature. This is prevalent in Figure 4.15b, Figure 4.16, Figure 4.17a and Figure 4.18b. This suggest that the model is slightly biased by using parts of the background to provide a prediction. A better model may be produced if such features are removed, and is further discussed in Section 4.8.

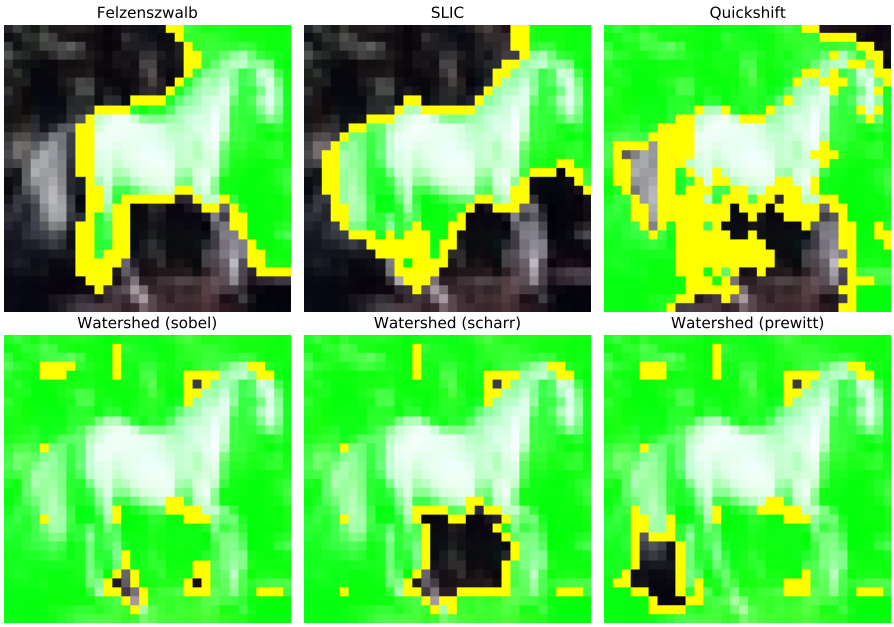


Figure 4.13: The six different segmented images are sent through LIME to obtain the shown explanation. The top 3 pro (in green) features are highlighted in each explanation image. The head is part of the larger superpixel in the LIME explanation using Felzenszwalb segmentation technique. The segmentation techniques makes the explanation slightly inconsistent, showing some of the issues with LIME on low resolution images.

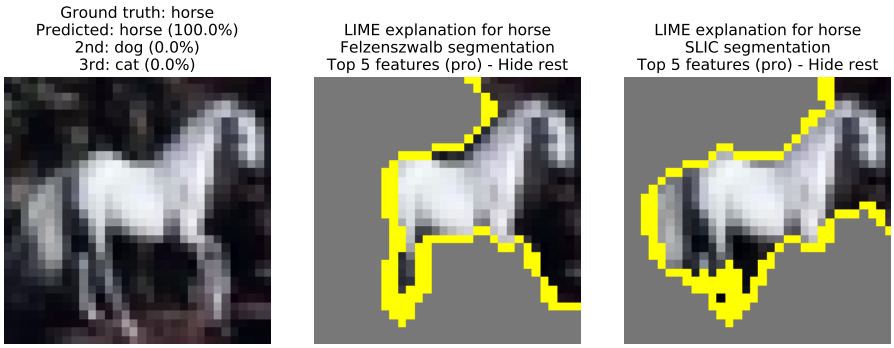
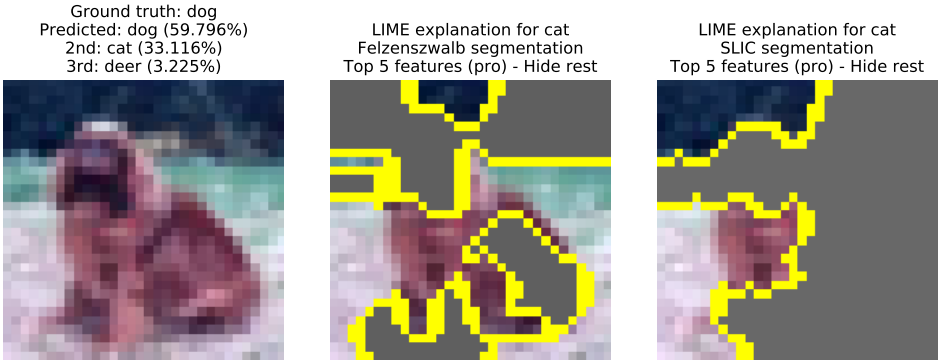


Figure 4.14: The figure shows the superpixels which LIME finds are most important towards the top prediction 'horse'. The head is part of the larger superpixel which also includes the background in the LIME explanation using Felzenszwalb segmentation technique. This does not happen in SLIC, and the head is still part of the explanation. This illustrates how the segmentation techniques makes the explanation slightly inconsistent, showing some of the issues with LIME on low resolution images.

(a) Image from test set with top three predictions according to the model. The middle and rightmost image shows the LIME explanation of top 5 features (pro) with the rest hidden.



(b) The same image with LIME explanation of the second top prediction 'cat'. Interestingly, by occluding the head of the dog the model sees a cat. It is understandable, looking from a humans perspective when the face is occluded.

Figure 4.15: By observing the two LIME explanation for both 'dog' and 'cat'. The model may be trusted since it picks up the dog's head and body as an important attribute.

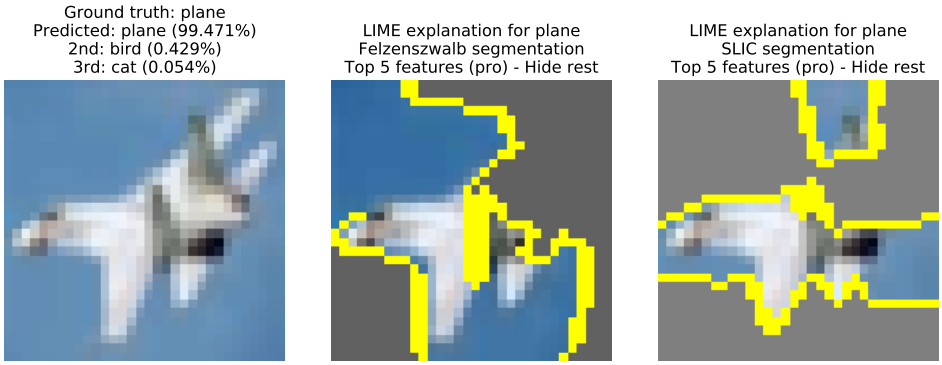
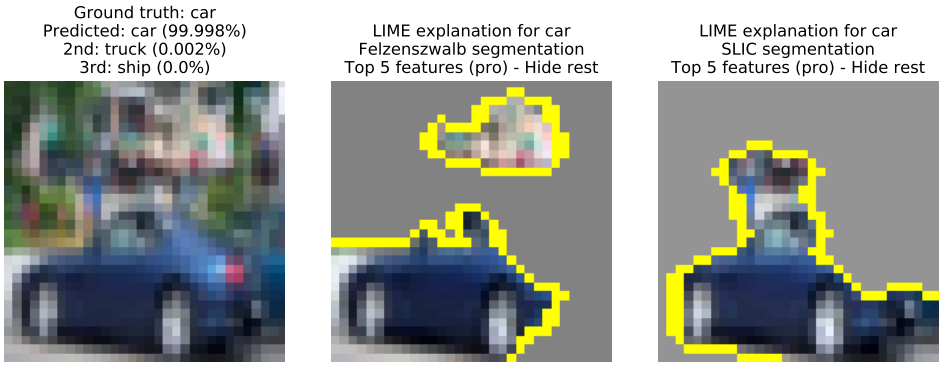
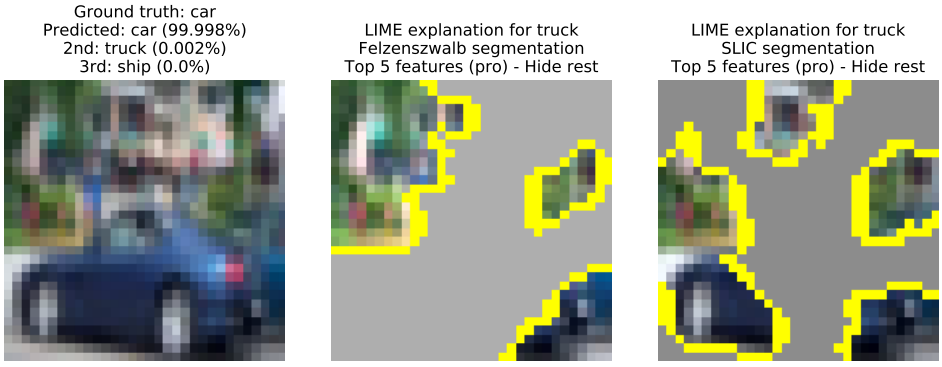


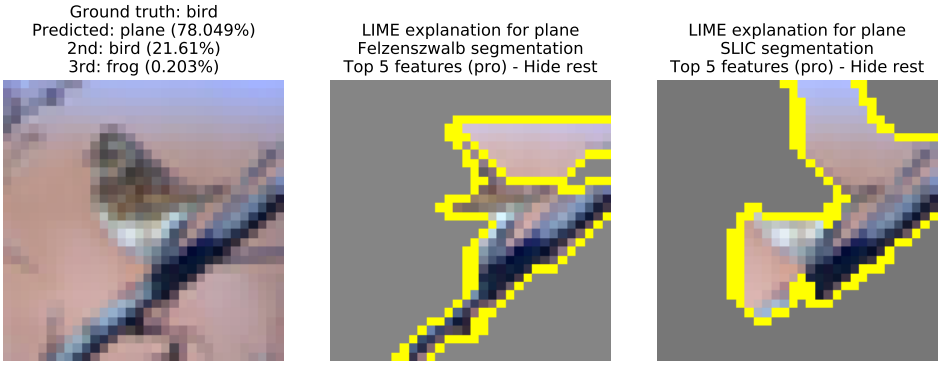
Figure 4.16: Image from test set with top three predictions according to the model. The middle and rightmost image shows the LIME explanation of top 5 features (pro) with the rest hidden.



(a) A highly confident prediction towards a 'car'. The wheels along with the lower chassis of the car makes up the most part of the features towards the prediction. Trust may be asserted to the model following the explanation given by LIME.



(b) The second top prediction class 'truck'. The model explains the class by looking only at the wheels and occluding parts of the car. The background also becomes a part of the explanation towards truck.



(a) Explanation for the top prediction 'plane' which is wrong compared to ground truth. The explanation highlights the stem of the three on which the bird is standing on. When occluding the bird, the feature does resemble a wing or parts of a plane even though the model is wrong.

(b) Explanation for the second top prediction 'bird', which is the correct class. The explanation highlights the body of the bird, but unfortunately this feature is not strong enough according to the model. Rest assure, the model is slightly uncertain whether this is a plane or bird, and trust may be asserted to the model because of the features it is picking up on, even though the prediction is wrong.

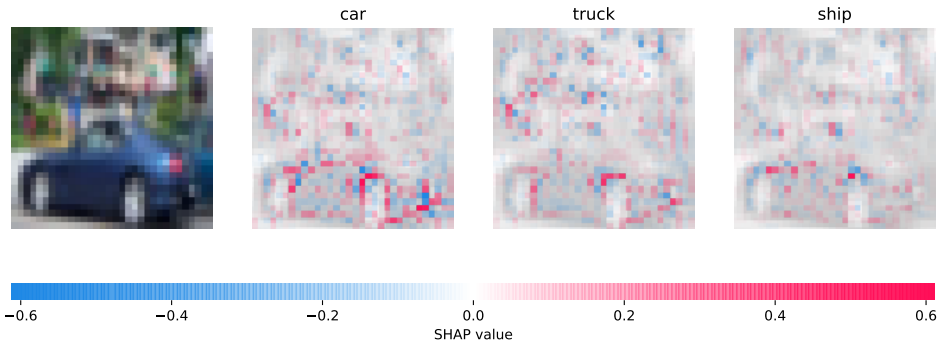
4.6 SHAP explanation

SHAP calculates the average contribution of a feature value to the prediction, as described in Section 3.6. 200 random test images are loaded from the test set and these are used throughout the experiment as the background set. These images are used to produce the expected model output and to approximate the SHAP values. By using the same randomly drawn images for calculating the SHAP values, variance across the explanations are kept to a minimum even though an unwanted bias might be present. The number of images for each class for the random baseline images may be imbalanced. This is not addressed in the documentation of the package, but it is recommended to use at least 100 samples to obtain the expected model output.

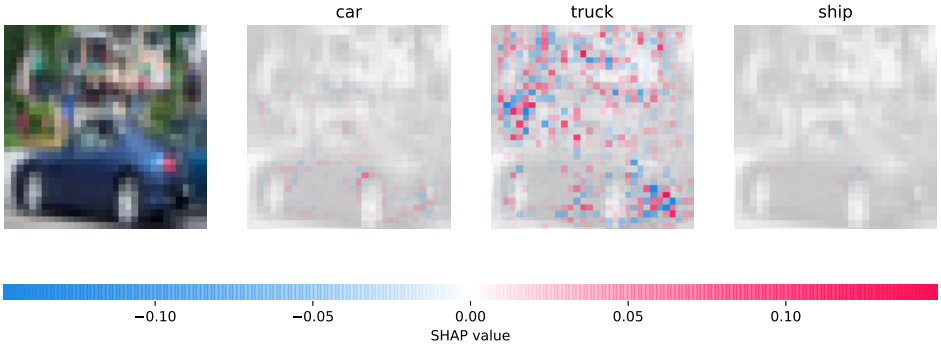
Next, it is mentioned that the explanation is sensitive to the output representation created by the model[4]. If a softmax is used at the output, then each value represents the probability and confidence towards the predicted class. Explaining such a model is referred to as using SHAP on the probability space. If no squashing function is used, then the explanation is performed on the margin space. The margin space is unbounded and in units of information, while the probability space uses units of probability. The choice of output space directly impacts the explanation provided by SHAP. The probability space SHAP gives larger weight to evidence increasing the output from 40%-50% compared to 98%-99%. This is because it requires far less information to go from 40% to 50% in comparison to 98% to 99%. The margin space directly corresponds to evidence and changes in the output are not saturated by squashing.

The displayed explanations of SHAP are shown in Figure 4.19, Figure 4.20 and Figure 4.21 with a discussion of their explanation in their respective captions. Even though the underlying theory behind SHAP is understood, the resulting explanation provided on this model appear rather complex and somewhat confusing. This could be a result of either a poor adaption of the model to SHAP or that it performs poorly on this type of data.

[4] See https://github.com/slundberg/shap/blob/master/notebooks/kernel_explainer/Squashing%20Effect.ipynb

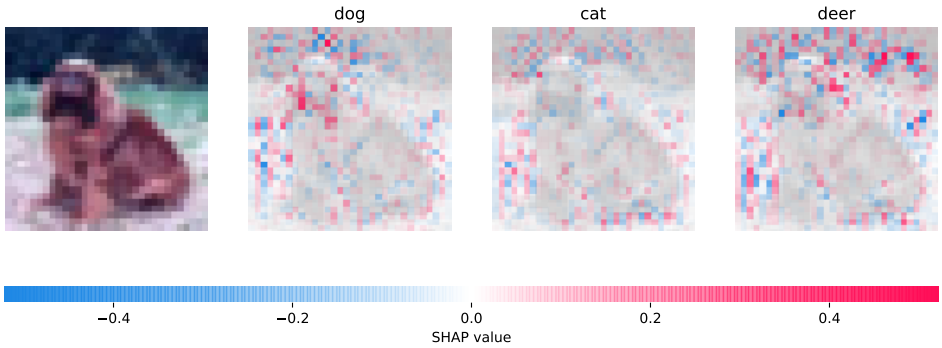


(a) Margin space SHAP.

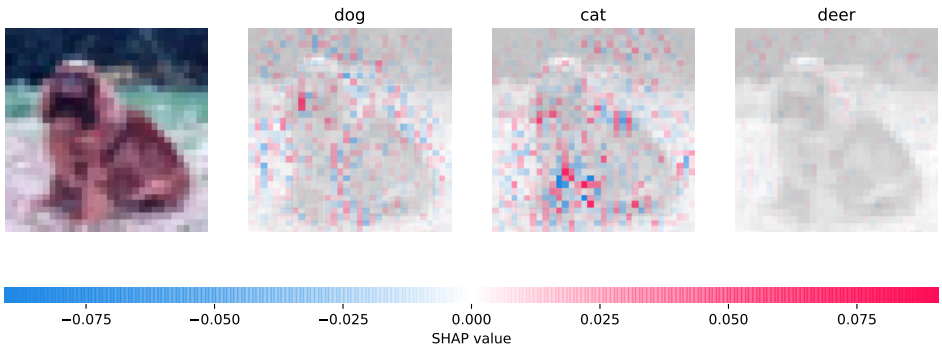


(b) Probabilistic space SHAP.

Figure 4.19: The effects on feature importance by choosing the output space. This on a correctly predicted 'car' 99.998%, followed by 'truck' 0.002% and third 'ship'. The probabilistic space SHAP gives low value the features in the explanation of 'car' as there is little change in the output. The SHAP values around the fender of the car are highlighted in red, meaning they contribute towards the prediction of a car.

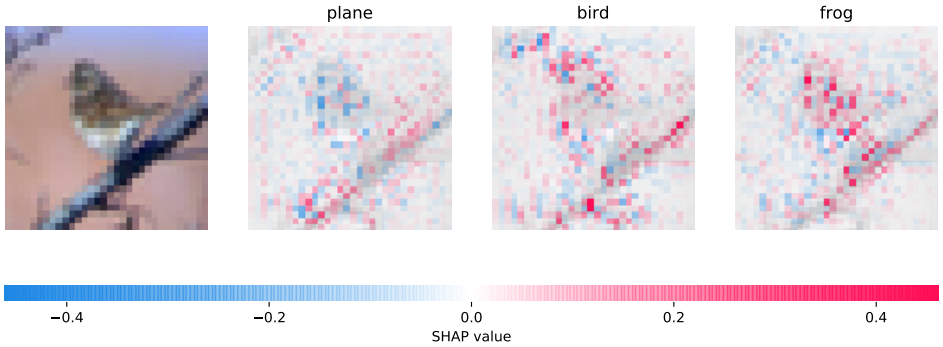


(a) Margin space SHAP.

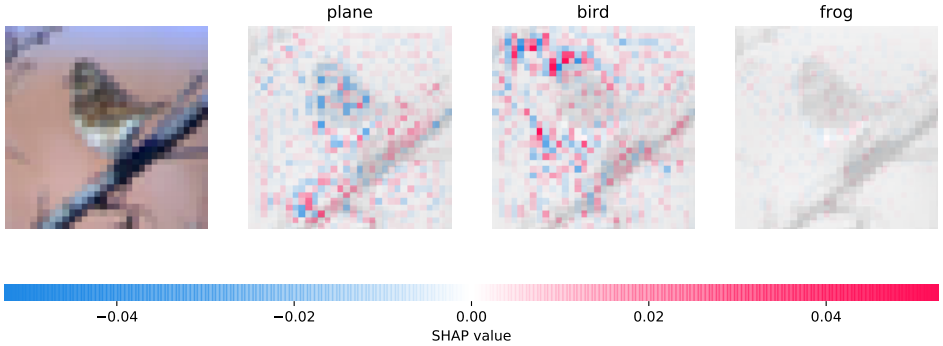


(b) Probabilistic space SHAP.

Figure 4.20: The effects on feature importance by choosing the output space. This on a correctly predicted 'dog' 59.796 %, followed by 'cat' 33.116% and 'deer' 3.225%. The dogs head appears to be an important feature towards the prediction of a dog. Some of the pixels in the background also appears to contribute. These observations are prevalent in both the margin and probabilistic space SHAP. For the margin explanation towards a 'deer', most of the background appears to contribute, while most parts of the dog itself appears to be ignored. The SHAP values for 'deer' in the probabilistic space SHAP attain a low value compared to the other explanations and are barely visible.



(a) Margin space SHAP.



(b) Probabilistic space SHAP.

Figure 4.21: The effects on feature importance by choosing the output space. The model has predicted the wrong class 'plane' 78.049% followed second by 'bird' 21.61%(the actual class) and third 'frog' 0.203%. The model sees the body of the bird as a negative feature and the parts of the stick as positive feature according to the explanation for 'plane'. This is consistent with the LIME explanation of the same image. The head of the bird appears important for the explanation of 'bird', also similar to the LIME explanation.

4.7 Comparing explanations by LIME and SHAP

Looking the four points by Miller presented in section 3.1, LIME satisfies the requisites to a greater extent in comparison to SHAP. The first point, explaining why event A happened over B, is possible with both methods. They are able to explain for and against any class, essentially explaining why a class was predicted over any of the other. On the second point where humans expects few reasons for a cause, LIME conforms better than SHAP. LIME presents only a few superpixels rather than showing the SHAP value for each pixel. This simplifies the explanation to a few regions highlighting the pros and cons. Unfortunately this means that the LIME explanation is a simplification, but for the layman the literature researched by Miller suggest such a method to be superior. The third point outlines the ineffectiveness of statistical explanations alone. Both SHAP and LIME are tools used together with the prediction confidence to explain an instance. Miller's third point therefore argues that there's greater value whenever probabilities and an explanation technique is used together, rather than a prediction confidence alone. The fourth point, an explanation being a conversational transfer of knowledge, implies that the explanation should be conveyed through a communicative medium. The image explanations provided by both methods comply with this point. Assuming an explanation is a social interaction between the model and the user, the fourth point may be interpreted as stating that an explanation should mimic how humans explain to each other. It would therefore imply that an improved explanation would include more elements used in human explanations such as natural language, body language, text, signs, expressions and other possible communicative interactions. Since a model is limited by its nature, this might not be feasible for most interaction types, but simple combinations of text, language, images etc. could be a viable strategy for future explainable methods.

The Shapley values, in which the SHAP values are based on, can easily be misinterpreted. The Shapley value does not represent the difference in prediction by removing a feature. It rather represents the change of contribution of a feature value compared to the mean output[23]. In other words, the Shapley value is the average marginal contribution of a feature.

A major drawback of the LIME method on images is the fact that the explanation model relies on the given segments. The superpixel from one segmentation technique to another, or even the same with different parameters, results in different explanations. An expert therefore needs to consider the choice of segmentation before applying LIME to the model. It is likely that the segmentations would perform better on higher resolution images, and the issues only apply to lower resolution datasets such as CIFAR-10. The utilized SHAP method on the other hand, does not use superpixels and avoids the issues obtained by segmentation, but the interpretation of the values are more difficult in comparison to LIME. Next, in both SHAP and LIME for images, the feature explanations are local and only applies to the instance being explained. This means that a global explanation of the model is not possible and a new explanation has to be computed for each instance. SHAP do, however, have a global explanation technique where the SHAP value of a feature is plotted over all the examples in a dataset, but this is mainly applicable for tabular data[5].

[5] An example is illustrated in the SHAP documentation https://github.com/slundberg/shap#

For the layman, LIME would likely be a better choice, as it greatly simplifies the explanation. For an expert, LIME is a great tool, but unfortunately it does not have the same mathematical strength as SHAP. As presented in Section 3.6.1, the game theoretic Shapley values obeys properties important for a fair and unique distribution of feature importance. For automated decision systems where decisions affects individuals, a fair and unique explanation may only be attainable by Shapley value based explanation methods. Therefore, as also stated in *the interpretable machine learning book* [23], SHAP may be the only novel technique which obeys the law by guaranteeing equality in the explanation.

Finally, it is worth noting that the implemented SHAP method is the Deep SHAP with single pixel explanations. However, Kernel SHAP, presented in section 3.6, allows for explanations using superpixels as features. The superpixels are, like LIME, found by segmentation algorithms and their respective SHAP values are found using the Kernel SHAP method. This was attempted to implement for the model, but unfortunately unsuccessful. It is likely that a Kernel SHAP segmented method would be as good or better than LIME.

4.8 Automated data augmentation and retraining model

In an attempt to improve the model, an automated augmentation of the dataset is performed. Since the background appears to be a part of the explanation from both LIME and SHAP, it would be interesting to see if partly removing the color information in the background could improve the performance. By removing the background colour, features prevalent in the foreground are undisturbed and the network is encouraged to use features in the foreground. Multiple efforts at detecting the background using edge detection and morphology were attempted to little success. Inspired by an online blogpost[6], a solution with edge detection and blurring is implemented with a satisfying performance. Each image is filtered with the sobel filter[7] to detect the edges in the image. The object of interest in the datasets appears in the foreground, leaving the background with less prominent edges. Next, the image is blurred using a two-dimensional gaussian blur[8]. This enlarges the edges such that areas with several edges obtain a higher value which saturates to 1. Components without any detected edges receives a value close to zero. Tuning the gaussian blur kernel results in either too large portions grayed out or none at all. A reasonable performance occured using a kernel of $\sigma = 3$. Finally, the parts of the image with a blur ≤ 0.20 are converted to grayscale. The result after graying the background with the aforementioned technique are displayed in Figure 4.21. A subset of the augmented training data are shown in Figure 4.22.

The original, unprocessed, dataset is used in combination with the preprocessed training data. This essentially doubles the data, half of which with the original color, and final half

tree-ensemble-example-with-treeexplainer-xgboostlightgbmcatboostscikit-learnpyspark-model

[6] https://flothesof.github.io/removing-background-scikit-image.html

[7] Using the scikit-image package https://scikit-image.org/docs/dev/api/skimage.filters.html#skimage.filters.sobel

[8] https://scikit-image.org/docs/dev/api/skimage.filters.html#skimage.filters.gaussian

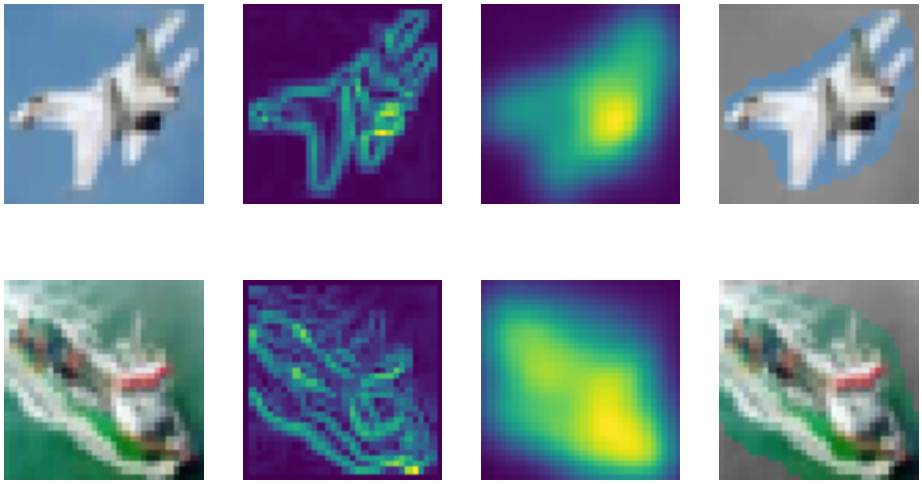


Figure 4.21: From left: The original images. 2nd: The images after the sobel transform. 3rd: A gaussian filter is applied to blur and enlarge the edges. 4th: The images with its background grayed out.

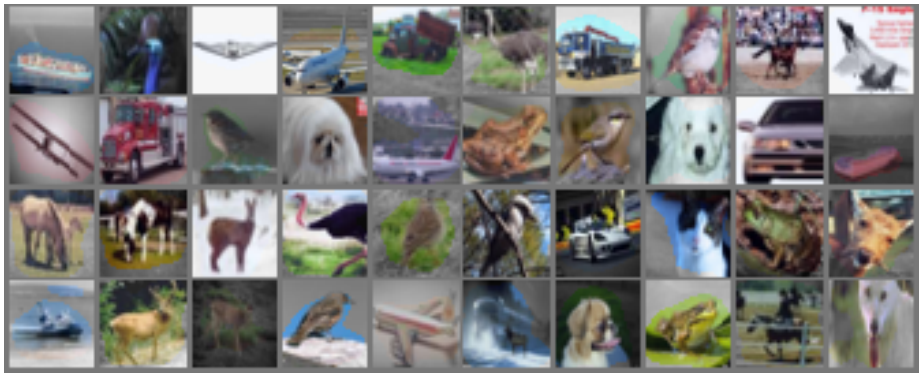


Figure 4.22: A sample of 40 images from the augmented training set after automatically graying out the background.

with parts of the background grayed out. This results in an augmented training set. The same architecture from Section 4.2 is used to obtain an unbiased performance comparison.

Retraining a new model with the augmented dataset results in a final test accuracy of 84.54%. This is an increased accuracy of 0.14% percentage points by comparison to the previous model with 84.20%. Therefore, the training was performed again, this time resulting in a test accuracy of 85.53%, an increase of 1.33% percentage points to the old model. Although the accuracy increases, it is unlikely that the augmented data is the rea-

son alone for the increased performance. The random initialization of weights, random dropout and early stopping may have a larger impact on the final accuracy.

As the augmented training data contains two sets of identical images where half have the background partly grayed out, it may help generalization to mirror the images about the vertical axis. This is a common strategy in data augmentation. A new model is trained with the preprocessed images **mirrored** and the parts of the background grayed out. The resulting model achieves a test accuracy of 87.27%, a noticeable increase of 3.07% percentage point compared to the old model. This shows a clear model improvement by performing data augmentation. Finally, a table with a comparison of the best performing model's and the old model's individual class accuracy is listed in Table 4.3.

Class	**New model test accuracy**	**Old model test accuracy**
Plane	91.1%	85.8%
Car	93.0 %	93.0 %
Bird	78.0 %	76.9 %
Cat	70.0 %	72.0 %
Deer	88.8 %	80.4 %
Dog	83.7 %	76.0 %
Frog	89.6 %	89.7 %
Horse	91.0%	87.0%
Ship	93.8 %	92.5 %
Truck	93.7 %	88.9 %
Average	**87.27%**	**84.20 %**

Table 4.3: Table showcasing the accuracy over each individual class for both the new and the previous model. The new model trained on the augmented data generalizes better and achieves a higher test accuracy.

Chapter 5

Conclusion

Explainable AI is an emerging field highly necessary in a world where evermore models are made using black-box approaches. Due to concerns regarding trust in safety critical applications, medical AI systems, automated decision systems amongst other, XAI aims to provide explanations and increase trust. Such AI systems also need to abide legislation by providing fair and unique explanations for their decision. A greater understanding of the theory behind prominent methods and some of their implementations has been obtained through this specialization project. LIME and SHAP were used on a computer vision model trained from scratch. The explanations from these two methods suggested parts of the background as a learned feature in the model and this information was used to augment the training data. The background was grayed out using a combination of edge detection and blur to identify the rich parts of the foreground. The images were mirrored and added to the training data, doubling its size. The resulting test accuracy increased from 84.20% to 87.27%. It is unclear whether the mirroring alone or the mirroring together with the grayed out background helped the model to generalize better. Nevertheless, the explanations provided insights in the data, helping to understand features the black box model activates on images. This was especial insightful for an example where a wrong prediction was made. Finally, a deeper knowledge on how to work towards a larger project, literature survey and individual work was achieved.

5.1 Further work

While the outcomes of the project itself are exciting, there is still endless potential in the field of XAI:

- The previous work presented in Chapter 3 only covers explanations utilizing input-output pairs to a single type of data. It would therefore be of interest to investigate if XAI methods could be applied to cyber-physical systems where multiple sensor

inputs are needed. This could as an example be a fusion of sensors on a robotic system, e.g. cameras, LIDARs, pressure sensors, GPS, gyroscopes and so on.

- Another interesting approach would be to combine training with metaprogramming and XAI. Using metaprogramming, the training process could be adjusted with feedback loops from XAI explanations.
- Improving current XAI methods such that more human centered explanation are provided.
- Investigate how XAI may help detect adversarial attacks aiming at tricking the model.

Bibliography

[1] , 2016. General data protection regulation.
URL https://eur-lex.europa.eu/eli/reg/2016/679/oj

[2] Ancona, M., Ceolini, E., Öztireli, A. C., Gross, M. H., 2017. A unified view of gradient-based attribution methods for deep neural networks. CoRR abs/1711.06104.
URL http://arxiv.org/abs/1711.06104

[3] Ancona, M., Ceolini, E., Öztireli, C., Gross, M., 2019. Gradient-Based Attribution Methods. Springer International Publishing, Cham, pp. 169–191.
URL https://doi.org/10.1007/978-3-030-28954-6_9

[4] Bach, S., Binder, A., Montavon, G., Klauschen, F., Mller, K.-R., Samek, W., 07 2015. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. PLOS ONE 10 (7), 1–46.
URL https://doi.org/10.1371/journal.pone.0130140

[5] BBC, March 2016. Artificial intelligence: Google's alphago beats go master lee sedol.
URL https://www.bbc.com/news/technology-35785875

[6] Bojarski, M., Choromanska, A., Choromanski, K., Firner, B., Ackel, L. J., Muller, U., Yeres, P., Zieba, K., 2018. Visualbackprop: Efficient visualization of cnns for autonomous driving. 2018 IEEE International Conference on Robotics and Automation (ICRA), 1–8.

[7] DeepMind, March 2016. Match 2 - google deepmind challenge match: Lee sedol vs alphago.
URL https://youtu.be/l-GsfyVCBu0?t=4696

[8] DeepMind, April 2017. Innovations of alphago.
URL https://deepmind.com/blog/article/innovations-alphago

[9] Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning. MIT Press, http://www.deeplearningbook.org.

[10] Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J., et al., 2001. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

[11] Howard, R. A., Sep. 1968. The foundations of decision analysis. IEEE Transactions on Systems Science and Cybernetics 4 (3), 211–219.

[12] Ioffe, S., Szegedy, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. CoRR abs/1502.03167.
URL http://arxiv.org/abs/1502.03167

[13] Jia, R., Dao, D., Wang, B., Hubis, F. A., Hynes, N., Gurel, N. M., Li, B., Zhang, C., Song, D., Spanos, C., 2019. Towards efficient data valuation based on the shapley value. arXiv preprint arXiv:1902.10275.

[14] Karras, T., Aila, T., Laine, S., Lehtinen, J., 2017. Progressive growing of gans for improved quality, stability, and variation. ArXiv abs/1710.10196.

[15] Krizhevsky, A., 2009. Learning multiple layers of features from tiny images.

[16] Krizhevsky, A., Nair, V., Hinton, G., 2009. Cifar-10 (canadian institute for advanced research).
URL http://www.cs.toronto.edu/~kriz/cifar.html

[17] Krizhevsky, A., Sutskever, I., Hinton, G., 01 2012. Imagenet classification with deep convolutional neural networks. Neural Information Processing Systems 25.

[18] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., Jackel, L. D., 1989. Backpropagation applied to handwritten zip code recognition. Neural Computation 1 (4), 541–551.
URL https://doi.org/10.1162/neco.1989.1.4.541

[19] LeNail, A., 1 2019. Nn-svg: Publication-ready neural network architecture schematics. Journal of Open Source Software 4 (33), 747.
URL http://dx.doi.org/10.21105/joss.00747

[20] Lundberg, S. M., Lee, S.-I., 2017. A unified approach to interpreting model predictions. In: Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (Eds.), Advances in Neural Information Processing Systems 30. Curran Associates, Inc., pp. 4765–4774.
URL http://papers.nips.cc/paper/7062-a-unified-approach-to-interpret
pdf

[21] Lundberg, S. M., Lee, S.-I., n.d n.d. Shap (shapley additive explanations).
URL https://github.com/slundberg/shap

[22] Miller, T., 2017. Explanation in artificial intelligence: Insights from the social sciences. Artif. Intell. 267, 1–38.

[23] Molnar, C., 2019. Interpretable Machine Learning. https://christophm.github.io/interpretable-ml-book/.

[24] Papernot, N., McDaniel, P. D., Goodfellow, I. J., Jha, S., Celik, Z. B., Swami, A., 2016. Practical black-box attacks against deep learning systems using adversarial examples. CoRR abs/1602.02697.
URL http://arxiv.org/abs/1602.02697

[25] Radford, A., Metz, L., Chintala, S., 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. CoRR abs/1511.06434.

[26] Ramsundar, B., Kearnes, S., Riley, P., Webster, D., Konerding, D., Pande, V., Feb 2015. Massively Multitask Networks for Drug Discovery. arXiv e-prints, arXiv:1502.02072.

[27] Rasmussen, P. M., Madsen, K. H., Lund, T. E., Hansen, L. K., 2011. Visualization of nonlinear kernel models in neuroimaging by sensitivity maps. NeuroImage 55 (3), 1120 – 1131.
URL http://www.sciencedirect.com/science/article/pii/S1053811910016198

[28] Russell, S., Norvig, P., 2009. Artificial Intelligence: A Modern Approach, 3rd Edition. Prentice Hall Press, Upper Saddle River, NJ, USA.

[29] Samek, W., Müller, K.-R., 2019. Towards Explainable Artificial Intelligence. Springer International Publishing, Cham, pp. 5–22.
URL https://doi.org/10.1007/978-3-030-28954-6_1

[30] Samek, W., von Wiegand, T. E., Mueller, K., 2017. Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. ArXiv abs/1708.08296.

[31] Shapley, L. S., 1953. A value for n-person games. Contributions to the Theory of Games 2 (28), 307–317.

[32] Shrikumar, A., Greenside, P., Kundaje, A., 2017. Learning important features through propagating activation differences.

[33] Simonyan, K., Vedaldi, A., Zisserman, A., 2013. Deep inside convolutional networks: Visualising image classification models and saliency maps.

[34] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., 2014. Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research 15, 1929–1958.
URL http://jmlr.org/papers/v15/srivastava14a.html

[35] Sundararajan, M., Taly, A., Yan, Q., 2017. Axiomatic attribution for deep networks. CoRR abs/1703.01365.
URL http://arxiv.org/abs/1703.01365

[36] Tulio Ribeiro, M., Singh, S., Guestrin, C., Feb 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. arXiv e-prints, arXiv:1602.04938.

[37] Tulio Ribeiro, M., Singh, S., Guestrin, C., n.d n.d. Lime (ocal interpretable model-agnostic explanations,).
URL https://github.com/marcotcr/lime

[38] Wang, F., Rudin, C., 2014. Falling rule lists.