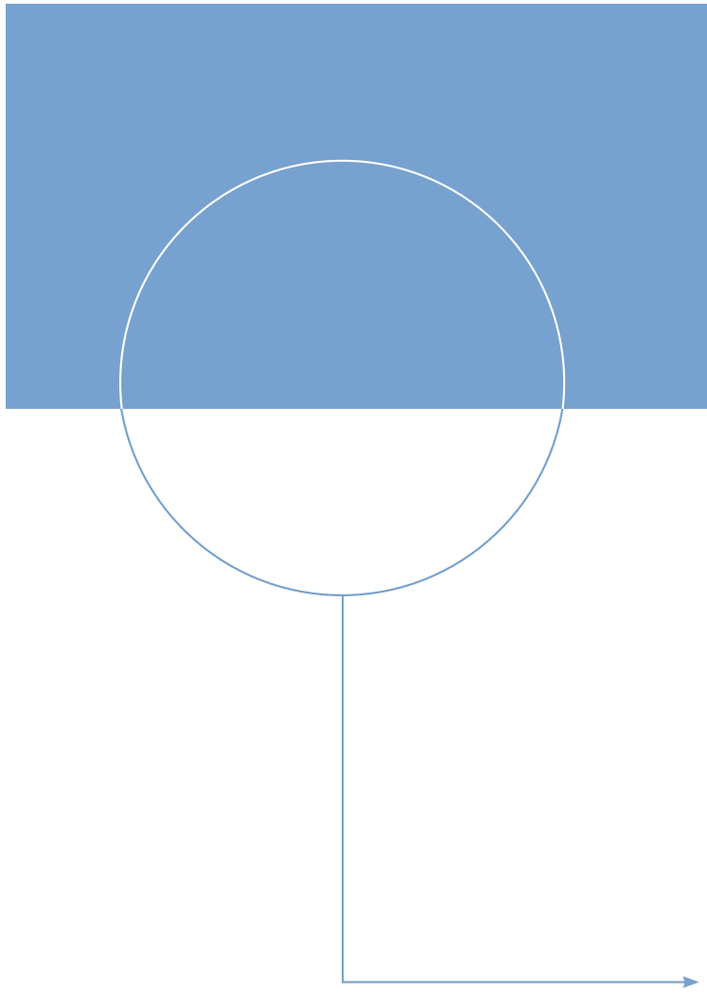


Master's thesis

2021

Yanzhe Bekkemoen

NTNU
Norwegian University of
Science and Technology
Faculty of Information Technology and Electrical
Engineering
Department of Computer Science

Yanzhe Bekkemoen

Correcting Classification

A Bayesian Framework Using Explanation Feedback to Improve Classification Abilities

May 2021

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

Correcting Classification

A Bayesian Framework Using Explanation Feedback to Improve Classification Abilities

Yanzhe Bekkemoen

Computer Science
Submission date: May 2021
Supervisor: Helge Langseth

Norwegian University of Science and Technology
Department of Computer Science

Abstract

Neural networks (NNs) have shown high predictive performance, however, with shortcomings. Firstly, the reasons behind the classifications are not fully understood. Several explanation methods have been developed, but they do not provide mechanisms for users to interact with the explanations. Explanations are social, meaning they are a transfer of knowledge through interactions. Nonetheless, current explanation methods contribute only to one-way communication. Secondly, NNs tend to be overconfident, providing unreasonable uncertainty estimates on out-of-distribution observations. We overcome these difficulties by training a Bayesian convolutional neural network (CNN) that uses explanation feedback. After training, the model presents explanations of training sample classifications to an annotator. Based on the provided information, the annotator can accept or reject the explanations by providing feedback. Our proposed method utilizes this feedback for fine-tuning to correct the model such that the explanations and classifications improve. We use existing CNN architectures to demonstrate the method's effectiveness on one toy dataset (decoy MNIST) and two real-world datasets (Dogs vs. Cats and ISIC skin cancer). The experiments indicate that few annotated explanations and fine-tuning epochs are needed to improve the model and predictive performance, making the model more trustworthy and understandable.

Sammendrag

Nevrale nettverk (NNs) har vist høy prediktiv ytelse, men med mangler. For det første er ikke årsakene bak klassifiseringene fullstendig forstått. Flere forklaringsmetoder er utviklet, men de har ikke mekanismer for brukere å samhandle med forklaringene. Forklaringer er sosiale, noe som betyr at de er en overføring av kunnskap gjennom interaksjoner. Likevel bidrar nåværende forklaringsmetoder bare til enveiskommunikasjon. For det andre har NN-er en tendens til å være for selvsikre, og gir urimelige usikkerhetsestimater på observasjoner som ikke kommer fra samme distribusjon. Vi overvinner disse vanskelighetene ved å trene et Bayesiansk Konvolusjonell nevrale nettverk (CNN) som bruker forklaringstilbakemeldinger. Etter trening presenterer modellen forklaringer på klassifiseringer av treningseksempler til en orakel. Basert på gitt informasjon kan orakelet godta eller avvise forklaringene ved å gi tilbakemelding. Den foreslåtte metoden vår bruker denne tilbakemeldingen til finjustering for å korrigere modellen slik at forklaringene og klassifiseringene forbedres. Vi bruker eksisterende CNN-arkitekturer for å demonstrere metodens effektivitet på ett leketøydatasett (decoy MNIST) og to ekte datasett (Dogs vs. Cats og ISIC skin cancer). Eksperimentene indikerer at få tilbakemeldinger- og finjusteringsepoker er nødvendig for å forbedre modellen og prediktiv ytelse, noe som gjør modellen mer pålitelig og forståelig.

Preface

I want to thank my supervisor Helge Langseth for his guidance, support and encouragement.

Yanzhe Bekkemoen

Trondheim, May 1, 2021

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The following chapter outlines the background and motivation for this master's thesis. Based on the background and motivation, we will concretize our goal for this thesis and divide it into research questions. We will end the chapter with an overview of this thesis's contribution and its structure. Appendix 3 can be read as a condensed version of this thesis's core contributions for those with sufficient background knowledge.

1.1 Background and Motivation

Machine learning (ML) is a branch of artificial intelligence (AI) consisting of methods that automatically learns from data to perform a predefined task using a performance measure to guide the learning process [Mitchell, 1997]. ML has many applications and is widely used, from commerce [Tintarev and Masthoff, 2011] to more critical applications, such as credit risk assessment [Hand and Henley, 1997] and medical diagnosis [Kononenko, 2001]. The influence ML has on economical and societal decision-making has been increasing, and more of the decisions previously taken by humans alone are now a joint task of human and machines, and some just machines [Makridakis, 2017; Furman and Seamans, 2018]. In recent years, ML has leaped forward in performance. In some games such as Go, Chess, Shogi, and Atari, the performance has even surpassed humans [Mnih et al., 2013]. Many of these accomplishments are due to the recent development in deep learning (DL) [LeCun et al., 2015].

Despite these accomplishments, there are still obstacles that make the adoption of ML on a wider scale difficult [Chui and Malhotra, 2018; Patel et al., 2008].

One of the problems that make the adaptation difficult is the explainability of ML models. Interpretability or explainability refers to a model's ability to explain its decision-making process in a human-understandable way [Doshi-Velez and Kim, 2017]. Explainability is important to prove many desired properties with ML models such as fairness and safety, and induce trust. In some cases the interpretability of ML model is even a legal requirement [Goodman and Flaxman, 2017]. Different metrics, such as accuracy, recall, and precision, can verify a model's predictive performance. However, it can neither prove nor guarantee that the models are fair, reliable, or safe in practice. Not only that, research has shown that people are often reluctant in algorithmic decision making without them verifying [Dawes, 1979; Dietvorst et al., 2015; Binns et al., 2018].

DL models are known to have high predictive performance but suffer from low explainability [Gunning and Aha, 2019]. Much research has gone into making DL models interpretable. One approach focuses on developing interpretable ML models that are inherently interpretable because of their simplicity [Letham et al., 2015]. This approach circumvents the difficulty of explaining DL models by replacing them with less expressive models. Thus, trading predictive performance for explainability. Another approach try to interpret existing DL models by introducing a method that works with the existing models [Selvaraju et al., 2020a; Ribeiro et al., 2016a; Shrikumar et al., 2019; Simonyan et al., 2014] without modifying them. Methods that take on this approach are referred to as post-hoc methods [Murdoch et al., 2019; Lipton, 2018]. These methods try to explain DL models' predictions by looking at the importance of individual features (e.g., pixels for image data and words for textual data). Hence, avoiding the compromise between predictive performance and explainability.

Related to interpretability, there is the issue with DL models' robustness. DL models are bad at quantifying uncertainty and tend to provide overconfident predictions [Lakshminarayanan et al., 2017]. An overconfident model can be dangerous, especially in more critical applications such as autonomous cars and medical applications. Furthermore, it has been shown that an overconfident model can be perceived as offensive [Amodei et al., 2016]. Although explainability methods can provide insight into model overconfidence, there is still a need for techniques to resolve overconfidence. One way to resolve overconfidence is to use probabilistic techniques to explicitly incorporate uncertainty; both Bayesian [MacKay, 1992; Neal, 1996; Maddox et al., 2019; Blundell et al., 2015; Louizos and Welling, 2017] and non-Bayesian [Malinin and Gales, 2018; Lakshminarayanan et al., 2017; Osband et al., 2016] approaches are investigated in the literature.

There has been much literature on explaining DL models. However, there has been little work on using classification explanations beyond understanding the models' reasoning process for decision making. We define the use of explanations

as ways to utilize the information provided through those explanations to improve a model's classification abilities. More specifically, a user gets asked to evaluate classification explanations. Based on the explanations, the user might agree or disagree with the model. If the user disagrees, it can provide an alternative explanation. This alternative explanation is added to the dataset as additional data the model gets trained with during a fine-tuning phase to correct the model's understanding of the problem.

In this thesis, we want to develop a method that uses classification explanations to correct a model if the practitioner thinks the explanations are wrong. Moreover, we want the model to quantify the uncertainty since robustness, trust, and safety are closely related. For that reason, we propose a novel model correction method that is compatible with Bayesian inference and its principled approach to uncertainty quantification. Model correction refers to methods that employ classification explanations to improve the models' explanations and predictive performance. After training, a Bayesian CNN presents explanations of training sample classifications and explanations to a human annotator. The annotator can accept or reject the explanations by giving feedback as additional evidence. This feedback is used during fine-tuning to correct the model such that the explanations and predictive performance improve.

1.2 Goals and Research Questions

In this section, we outline the goal of this master's thesis. Furthermore, we will concretized this goal into three more specific research questions. Our goal, motivated by the issues in Section 1.1 is

Goal Increase the robustness and explainability of NNs through model interaction, correction and Bayesian inference for uncertainty quantification.

In order to develop new ML methods, the research field must first be mapped. This thesis takes on research from two fields, Bayesian DL and explainable artificial intelligence (XAI). Thus, our first research question is:

Research question 1. *What is the state-of-the-art methods within Bayesian Deep Learning and Explainable AI?*

The second research question we investigate in this thesis concerns the possibility of correcting models using their classification explanations. There has been much research on explaining the decision-making process of NNs. Would it be possible to extend the use of these explanations to understand the NNs and improve their classification abilities? This leads us to our second research question.

Research question 2. *Can a user interact with a NN using its classification explanations to correct the model and improve its classification abilities?*

Provided a positive answer to whether we can use the classification explanations to improve NNs. Could we take a step further and use this method with Bayesian inference? Thus, not only resulting in a model that provides correct explanations but also robust predictions. Our third research question is the following:

Research question 3. *Can the model correction method be combined with Bayesian inference to obtain a robust and explainable NN model?*

1.3 Contributions

This section gives an brief overview of the contributions of this thesis. A more elaborated overview of the contributions are given in Section 7.2. The contributions are as follows:

1. An interactive framework to probe a human annotator for feedback on classification explanations (see Figure 4.1). This framework augments the "standard" ML pipeline with two steps

   Explain and Provide Feedback. During this step, a model asks a human annotator for feedback on classification explanations, similar to active learning (AL).

   Fine-tune. Takes the feedback gathered from the previous step and fine-tunes the model to correct it.

2. A Bayesian framework utilizing explanation feedback to correct a model. The application of Bayesian inference results in a mathematically grounded objective function.

3. Experimental results on one toy dataset and two real-world datasets that demonstrate the method's effectiveness. This indicates that few annotated explanations and fine-tuning epochs are required to improve explanations justifying the classifications and the predictive performance.

1.4 Thesis Structure

The rest of the thesis is structured as follows.

Chapter 2 Background Theory. We will in Chapter 2 outline the necessary background materials needed to understand the rest of this thesis. The background section starts with describing NNs in Section 2.1 and CNN in

Section 2.2. After finishing the theory concerning NNs, Section 2.3 moves to the topic of XAI. Section 2.4 outlines the classification explanations methods that are used throughout this thesis. The last part of this chapter, Section 2.5 reviews Bayesian inference and one approach to estimate the posterior distribution, namely, variational inference (VI).

Apart from providing background knowledge, this chapter also gives the reader an overview of some of the state-of-the-art methods within Bayesian DL and XAI. Thus, this chapter tries to answer Research question 1.

Chapter 3 Framework. Chapter 3 outlines the programming libraries and the computational frameworks used to realize this thesis. Also, this chapter gives a brief introduction to automatic differentiation (AD) which is the generalization of backpropagation.

Chapter 4 Active Feedback. Chapter 4 introduces the model correction method that is linked to Research question 2. Moreover, this chapter proposes a framework to interact with a ML model to generate explanation feedback for the model correction method.

Chapter 5 Bayesian model correction framework. Chapter 5 extends the model correction method proposed in Chapter 4 to be compatible with Bayesian NNs. This chapter is related to Research question 3 that asks whether or not the model correction method can be used with Bayesian inference.

Chapter 6 Experiments and Results. Chapter 6 provides the experimental setup with the necessary information to reproduce the experiments. Moreover, this chapter outlines the results of the experiments that indicate positive answers to both Research questions 2 and 3.

Chapter 7 Evaluation and Conclusion. Chapter 7 concludes this thesis with an evaluation, discussion, and future work. Furthermore, this chapter elaborates on the contributions that were briefly introduced in this chapter. The evaluation discusses the experimental results from Chapter 6. The discussion reviews the advantages and the shortcomings of this work. Finally, future work considers some of the possible future directions for this work.

Appendices. The appendices contain the mathematical notation and abbreviations used throughout this thesis. Moreover, they provide a distilled version of this thesis as an article. The article contains all of the core contributions of this thesis.

Chapter 2

Background Theory

This chapter aims to provide the reader with sufficient background material to understand the rest of the thesis. The material provided in this chapter is suitable for those who need a quick recap. Not every detail will be outlined in this chapter, and the readers that require a more in-depth explanation are asked to consult the reference material.

2.1 Neural Network

This section outlines the core concepts of NNs. It starts with a brief introduction to supervised ML, and then slowly builds up the theory around NNs. From how a NN makes prediction/classification to how the learning process works. Notice that although the main focus is on NNs, we will always give a general ML description before narrowing the content onto NNs.

2.1.1 Supervised Learning

ML is collection methods and techniques that enable computers to improve their performance automatically with experience. The improvement or learning happens with an experience tailored to a specific task. It is said that models learn when their performance (it is assumed that a performance measure exists) increases at a given task using the experience [Mitchell, 1997]. ML is used in a wide variety of applications, such as translation and spam filtering [Guzella and Caminhas, 2009; Bahdanau et al., 2016]. The methods differ in how they work, how the experience is structured, and what tasks they are intended to solve [Jordan and Mitchell, 2015]. The learning is often divided into supervised learning,

unsupervised learning, and reinforcement learning. However, the methods can also have other types of divisions. In this thesis, we are mainly going to deal with supervised learning.

In a supervised learning setting, we have data $\mathcal{D} = \{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_{i=1}^{n}$ or $\mathcal{D} = (\boldsymbol{X}, \boldsymbol{Y})$ in matrix form. We assume that the data has the following relation, $\boldsymbol{y}^{(i)} = f(\boldsymbol{x}^{(i)}) + \boldsymbol{\epsilon}$. $\boldsymbol{\epsilon}$ is the random error term or measurement noise, independent of $\boldsymbol{x}^{(i)}$ and has mean zero that covers all the unobserved covariates that influence $\boldsymbol{y}^{(i)}$. Using the data, the goal is to approximate the unknown function $f(\boldsymbol{x}^{(i)})$ that describes the relationship between the covariates $\boldsymbol{x}^{(i)}$ and the response $\boldsymbol{y}^{(i)}$ for samples in our data, but also samples not seen. In other words, the data serves as a proxy for the function f we wish to find. We want to find the function either for prediction/classification or inference. This can be done by finding a function $\hat{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \approx f(\boldsymbol{x}^{(i)})$ (we will write $\hat{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})$ as $\hat{f}_{\boldsymbol{\theta}}$ from now on to lighten notation) that is parameterized by $\boldsymbol{\theta}$ (not all ML models are parameterized by some $\boldsymbol{\theta}$, but for now let us assume this is true).

The goal is to choose a model that represents the approximating function $\hat{f}_{\boldsymbol{\theta}}$ and find its parameter $\boldsymbol{\theta}^*$ that minimizes the "difference" (how the difference is measured or defined will be dealt in Section 2.1.3) between the real function f and the approximating function $\hat{f}_{\boldsymbol{\theta}}$ using the data $\mathcal{D}$. By using the data $\mathcal{D}$, we assume that the empirical distribution $\hat{p}_{\text{data}}$ defined by $\mathcal{D}$ to a certain degree represents the data generating distribution p_{data} "well" enough. Finding the parameter $\boldsymbol{\theta}^*$ that minimizes the "difference" is done by applying an optimization algorithm specific to the model representing $\hat{f}_{\boldsymbol{\theta}}$, guided by some performance measure $\mathcal{L}$ which measures this "difference" between the functions using the experience or data $\mathcal{D}$. A NN is an example of a model that can represent $\hat{f}_{\boldsymbol{\theta}}$. However, it can also be some other type of ML model.

The empirical data $\mathcal{D}$ used to train the model is often divided into different parts. It is usually divided into training data $\mathcal{D}_{\text{train}}$, validation data $\mathcal{D}_{\text{valid}}$, and test data $\mathcal{D}_{\text{test}}$. Here, $\mathcal{D}_{\text{train}}$ is used to teach the model to solve the given task. Besides the parameters $\boldsymbol{\theta}$ that the model derives from training, it also has hyperparameters that are parameters set before the learning process. $\mathcal{D}_{\text{valid}}$ is used to tune these hyperparameters so that the model can perform better at the given task. When tuning and training are done, $\mathcal{D}_{\text{test}}$ is used to verify the model's final performance. It is not allowed to tune or train the model after $\mathcal{D}_{\text{test}}$ has been applied. $\mathcal{D}_{\text{test}}$ can be seen as a proxy measure for how well the model will perform after it is deployed in production.

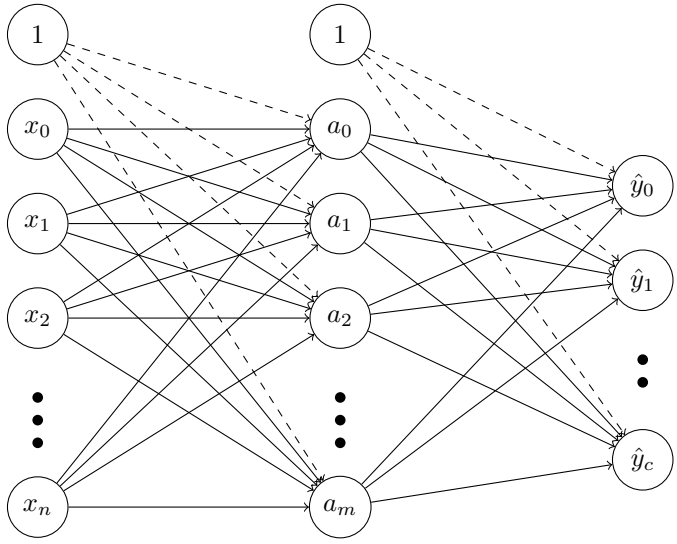


Figure 2.1: A FNN, where the circles represent neurons and a collection of neurons on the same level, are named a layer. The depicted network is also sometimes called a single-layer perceptron. Also, the figure provides the different terms used for different layers in a FNN.

2.1.2 Feedforward Neural Network

A NN can be seen as a function approximator $\hat{f}_{\boldsymbol{\theta}}$ able to approximate some function f. The term NN encompasses a large variety of network architectures or learning methods. In this section, we will describe the most widely used FNN, or often called a multi(single)layer perceptron.

If we take a closer look, a NN is a network of nodes with directed edges connected in a structured way, as seen in Figure 2.1. In a NN, nodes are called neurons and edges connections. As seen in the figure, we have special nodes 1 called the bias node. The bias is a neuron that is constant with trainable weight. It serves the same purpose as the intercept in a regression model. The bias node provides the NN with flexibility and allows the output of the neurons to differ from 0 (before the activation function is applied) when all the inputs are 0. Only being able to produce 0 when all the inputs are 0 might yield an inferior fit.

A neuron can be seen in Figure 2.2, where we have n inputs into the neuron and a bias neuron. Each input x_i is multiplied with the corresponding weight w_i. All of these values are summed together after the multiplications, $s = w_{b_0} + \sum_{i=0}^{n} w_i x_i$. The weights $\boldsymbol{w}$ are the parameters $\boldsymbol{\theta}$ described in Section 2.1.1 which implies that the goal is to find $\boldsymbol{w}^*$ that minimizes the "difference". The sum s passes through an activation function $\varphi(z)$, such as the Sigmoid function or rectified linear unit (ReLU) function to get $\hat{y} = \varphi(s)$. The output y of the activation function is passed on to a new neuron in the network that its edge directs to. If the neuron belongs to the last layer of the network, the activation value $\hat{y}$ is the output of the network. It is important to note that the activation functions used for the output layer and the hidden layers are, in most cases, different. It is most common to use a nonlinear activation function for the hidden layers. However, a linear activation function $\varphi(z) = z$ can be used. The activation function of the output layer depends on the task the NN tries to solve. For regression problems, the linear activation function is used, while for classification problems the Sigmoid activation function is used, or softmax activation function when the number of classes $k > 2$.

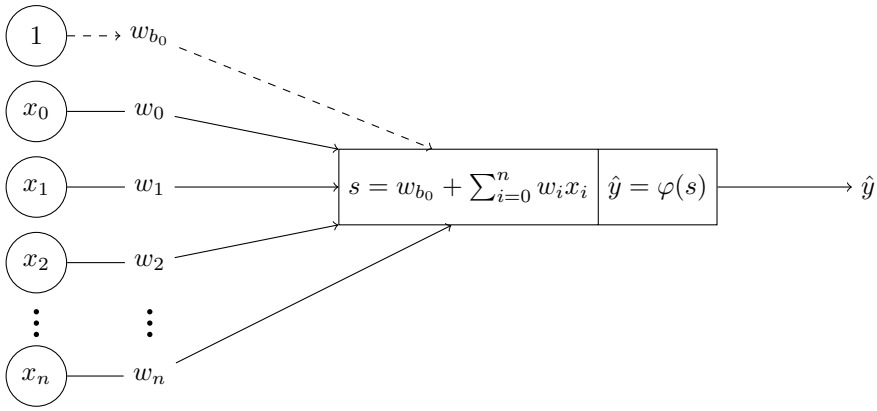
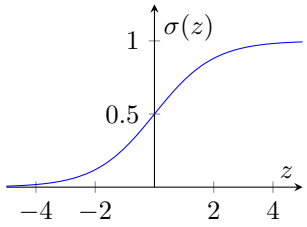
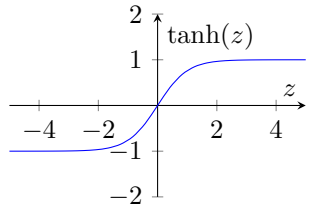


Figure 2.2: A single neuron with n input and the activation function $\varphi(x)$. The figure depicts how the input gets processed and turned into output in a single neuron.

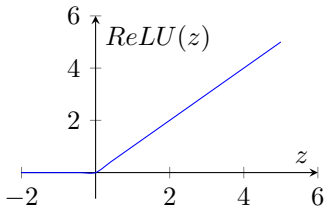
Nonlinear activation functions (shown in Figure 2.3) are used since they enable a NN to learn nonlinear relationships between the input and output space. If all layers have linear activation functions, the resulting network will be linear since the sum of linear functions is still a linear function. No matter how wide or deep the network is, it will still not capture nonlinear relationships in the data. The "Universal approximation theorem" states that a FNN with a single hidden layer



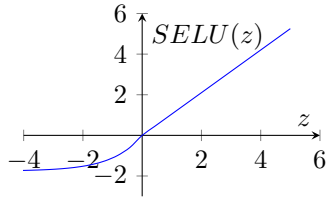
(a) Sigmoid activation function



(b) Hyperbolic tangent activation function



(c) ReLU activation function



(d) Scaled exponential linear unit activation function

Figure 2.3: Commonly used activation functions, Sigmoid $\sigma(z)$, hyperbolic tangent $tanh(z)$, rectified linear unit $ReLU(z)$ and scaled exponential linear unit $SELU(z)$ [Klambauer et al., 2017]

containing a finite number of neurons can approximate continuous functions on compact subsets $\mathbb{R}^n$. One of the first versions of this theorem was proved by Cybenko [1989] for the Sigmoid activation function. Later Hornik [1991] showed that the theorem does not depend on a specific activation function, but rather the architecture of the NN itself.

A neuron can be seen as a collection of weights that enable learning for a NN. As neurons tie together weights, layers tie together neurons. Therefore, next, we have layers that are a collection of nodes. In the most commonly known type of NN, FNN, neurons in the same layer are not interconnected. Instead, layers are stacked together. Each neuron in one layer is connected to all of the neurons in the next layer. The connections are directed edges, which means the signal only flows one way. This can be seen in Figure 2.1. The layers have dedicated names; the first layer of a network is called the input layer, and the last, output layer. Every layer in between the input and output layers is called the hidden layers.

Assuming a NN has learned the relationship between the input and output space.

It can make prediction/classification or inference by propagating signals from input space to output space. More specifically, let us illustrate this with a single hidden layer FNN, like the one seen in Figure 2.1.

Assume that we have the following setup:

Prediction

1. Input nodes: x_i for $i = 1, .., n$, or as a vector $\boldsymbol{x}$.
2. Hidden layer nodes: a_i for $i = 1, .., m$, or as a vector $\boldsymbol{a}$.

   The hidden layer has the activation function φ_h such that the output of the hidden layer becomes

   $$a_j(\boldsymbol{x}) = \varphi_h(w_{b_{0j}} + \sum_{i=1}^{n} w_{ij}x_i),\ j = 1, ..., m, \tag{2.1}$$

   where w_{ij} is the weight from input node x_i to hidden node a_j, and $w_{b_{0j}}$ is the bias term for the hidden node a_j.
3. Output nodes: $\hat{y}_i$ for $i = 1, .., c$, or as a vector $\hat{\boldsymbol{y}}$. Depending on the task the network is aimed to solve, c can range from 1 to a finite large number in $\mathbb{Z}^+$.

   The output layer have the activation function φ_o such that the output of the output layer becomes

   $$\hat{y}_j(\boldsymbol{x}) = \varphi_o(w_{b_{1j}} + \sum_{i=1}^{m} w_{ij}a_i),\ j = 1, ..., c, \tag{2.2}$$

   where w_{ij} is the connection from hidden node a_i to output node y_j, and $w_{b_{1j}}$ is the bias term for the output node $\hat{y}_j$.

This implies that the result for output node y_j is

$$\hat{y}_j(\boldsymbol{x}) = \varphi_o(w_{b_{1j}} + \sum_{i=1}^{m} w_{ij}a_i) = \varphi_o(w_{b_{1j}} + \sum_{i=1}^{m} w_{ij}(\varphi_h(w_{b_{0i}} + \sum_{l=1}^{n} w_{li}x_l))), \tag{2.3}$$

when the equation is expanded to the input. Equation (2.3) presents how a NN works during a forward pass. $\hat{\boldsymbol{y}}$ is our predicted result based on the input $\boldsymbol{x}$ that has been propagated through the network, in this case, inputted into Equation (2.3) and calculated. With a larger network, the basic calculations are still the same; however, more tedious.

2.1.3 Learning

Having defined a model $\hat{f}_{\boldsymbol{\theta}}$, we need a performance measure $\mathcal{L}$ in order to guide the model during learning (also known as training). $\mathcal{L}$ is an objective function that takes the data and the model $\mathcal{L}(\hat{f}_{\boldsymbol{\theta}}, \mathcal{D})$ and quantifies how "well" $\hat{f}_{\boldsymbol{\theta}}$ approximates the relationship between the covariates $\boldsymbol{x}$ and the responses $\boldsymbol{y}$ measured using the data $\mathcal{D}$. Notice that the objective function provided not only gets affected by the model chosen but also by how "well" the empirical distribution $\hat{p}_{\text{data}}$ defined by data $\mathcal{D}$ approximates the data generating distribution p_{data}.

An optimization algorithm can be applied after defining an objective function, for example, gradient descent for NNs. The algorithm tries to minimize (it can be maximizing too, depending on how the function is defined) $\mathcal{L}$ by adjusting the parameters $\boldsymbol{\theta}$ in $\hat{f}_{\boldsymbol{\theta}}$. Gradient descent is an optimization algorithm that tries to minimize a function iteratively by moving in the direction of the steepest descent defined by the negative gradient. For NN, the weights $\boldsymbol{w}$ are the parameters $\boldsymbol{\theta}$ we want to adjust. The loss function is minimized by adjusting the weights. As a consequence of how the algorithm works, the loss function we choose must be differentiable. Gradient descent works quite straightforward for a NN without hidden layers. For deeper networks, backpropagation is needed. We will first tackle gradient descent before moving on to the more general process, namely, backpropagation.

Loss function

In the context of NN, the term loss function is used to describe the objective function. The loss function measures the error between the true response and the prediction/classification for a single sample. When we aggregate the loss over several data points, the term cost function is often used (we will use the terms loss and cost interchangeably). The choice of loss function depends on the task and other factors. To explain how a loss function is, we will take a closer look at the mean squared error (MSE) loss function used for prediction tasks. The MSE loss function is given in Equation (2.4) where we have n predictions (we assume that the samples are independent and identically distributed (i.i.d.)). $\boldsymbol{y}$ is the ground truths, and $\hat{\boldsymbol{y}}$ is the predictions the model $\hat{f}_{\boldsymbol{\theta}}$ makes. MSE computes the square of the difference between y_i and $\hat{y}_i$ for all n samples averaged (2 in denominator is added to make derivation more convenient). Since MSE takes the square, the error will always be non-negative. Another consequence of the square is that it gives more weight to larger differences/distances. Because MSE measures the distance between vy and $\hat{\boldsymbol{y}}$, there needs to be a natural ordering between the response values. In classification, a natural ordering of the classes often does not exists. Therefore MSE loss can not be used for classification (not

true for binary) and is mainly used for prediction.

$$MSE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{2n} \sum_{i=1}^{n} (y^{(i)} - \hat{y}^{(i)})^2. \tag{2.4}$$

For classification tasks, CE loss function is applied (it is also known as the negative log-likelihood, negative since we want it to be minimization). The loss function can be seen in Equation (2.5) where we assume that there are C classes, $\hat{y}_c^{(i)}$ is the output from the Sigmoid output node for binary classification and softmax for classification where $C > 2$, and $y_c^{(i)}$ is the observed one-hot encoding for class c.

$$CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\frac{1}{n} \sum_{i=1}^{n} \frac{1}{C} \sum_{c=1}^{C} y_c^{(i)} \log \hat{y}_c^{(i)}. \tag{2.5}$$

By using a function like Equation (2.4) or Equation (2.5), the optimization algorithm is getting feedback. The loss provides information on how the parameters should be adjusted for each iteration.

Gradient descent

Now that we have a rough idea of how things work let us illustrate these concepts by outlining the gradient descent algorithm. Assume that we have a loss function $\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})$, for example, Equation (2.4). Further, assume that we have a model $\hat{f}_{\boldsymbol{w}}$ parameterized with weights $\boldsymbol{w}$, and a training set $\mathcal{D}_{\text{train}} = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$. Making these assumptions, we can derive the gradient descent algorithm that is shown in Algorithm 1. If we, for example, have a network like the one displayed in Figure 2.4 (assuming binary classification $C = 2$, we get that the gradient for the weight using the chain rule from calculus is:

$$\begin{aligned}
\frac{\partial \mathcal{L}(y, \hat{y}; w)}{\partial w} &= \frac{\partial s}{\partial w} \frac{\partial \hat{y}}{\partial s} \frac{\partial \mathcal{L}}{\partial \hat{y}} \\
&= \frac{\partial (wx + b)}{\partial w} \frac{\partial \phi(s)}{\partial s} \frac{\partial [y \log \hat{y} + (1 - y) \log(1 - \hat{y})]}{\partial \hat{y}} \\
&= x \cdot \hat{y}(1 - \hat{y}) \cdot (\frac{y}{\hat{y}} + \frac{y - 1}{1 - \hat{y}}).
\end{aligned}$$

As we can see, for a single neuron, the derivation is quite simple. However, this holds for networks with hidden layers and multiple neurons each layer too, which we will go into detail in Section 2.1.3.

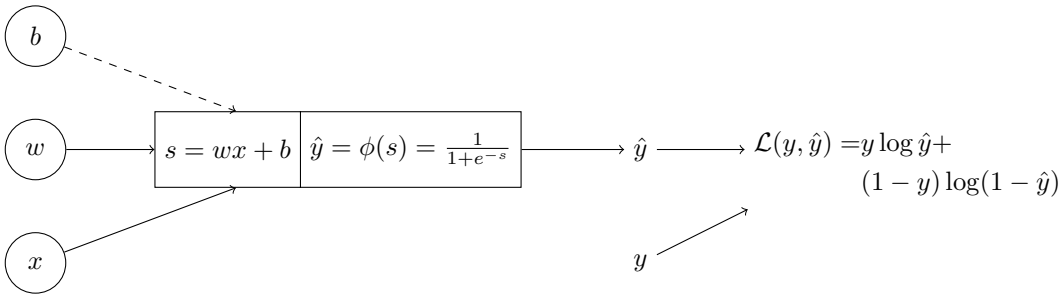


Figure 2.4: A single neuron with a single input and the Sigmoid activation function. The binary CE loss function is used to compare the prediction and the true response. This figure illustrates how the gradient of weights can be computed using the gradient descent algorithm.

Algorithm 1 describes what is known as SGD since we make updates to the weights $\boldsymbol{w}$ after only a single sample. It is called mini-batch SGD if we instead calculate the loss over several samples, but not the whole dataset. The other extreme of calculating the loss over the whole training set before making an update is called batch SGD.

When we calculate the loss over a small number of samples, the parameters are updated more often, making the training process faster. Besides, having noisy updates can make the model avoid local minimums. However, the noise can also be disadvantageous since it can make the error oscillate. On the other hand, having a large mini-batch size gives a more stable error gradient. Nevertheless, having a more stable error can also make it easier for the training procedure to get stuck at a local minimum.

When it comes to hardware aspects of learning, having too small batches can underutilize the tensor operations done efficiently in one go with large batches. On the other hand, having too large a batch size can also be disadvantageous since we need to load more data into the computer memory at once that might not be available. Overall, all batch sizes have pros and cons, and the choice depends on the task and resources available.

Algorithm 1 Stochastic gradient descent (SGD)

1. Let $t = 0$ and the initial values for the weights be $\boldsymbol{w}^{(0)}$ (initialized randomly drawn from some distribution).
2. Until finding a optimum (might not be the global minimum) or reaching some criteria (number of epochs), repeat this step:
   (a) For sample i in $1, ..., n$:
   (b) i. Calculate the prediction $\hat{y}^{(i)} = \hat{f}_{\boldsymbol{w}^{(t)}}(\boldsymbol{x}^{(i)})$. This step is known as the forward pass.
   ii. Calculate the loss function $\mathcal{L}(y^{(i)}, \hat{y}^{(i)}; \boldsymbol{w}^{(t)})$.
   iii. Find the gradient (direction) in the p-dimensional space of the weights, and evaluate this at the current weight values:

$$\nabla\mathcal{L}(y^{(i)}, \hat{y}^{(i)}; \boldsymbol{w}^{(t)}). \tag{2.6}$$

   This step is known as the backward pass.
   iv. Update the weights using a given step length (learning rate) λ in the direction of the negative of the gradient of the loss function:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \lambda \cdot \nabla\mathcal{L}(y^{(i)}, \hat{y}^{(i)}; \boldsymbol{w}^{(t)}). \tag{2.7}$$

   v. Increment $t = t + 1$.
3. The final values of the weights in the p-dimensional space $\boldsymbol{w}^{(\text{final})}$ are our parameter estimates $\boldsymbol{w}^*$.

Backpropagation

The differentiation of the loss function for a NN with a single neuron is quite straightforward. However, adding more layers and neurons make things a bit more complicated. Finding the analytical expression for $\nabla\mathcal{L}$ is not tricky, but the numerical evaluation is not cheap. Therefore, backpropagation is introduced, which essentially is a way to calculate the gradient for any weight in a multi-layer perceptron using the chain rule from calculus, making the calculations less expensive. The term backpropagation and its use were first announced by Rumelhart et al. [1986]. Backpropagation was later popularized and closer studied by Rumelhart and McClelland [1987]. However, the technique was rediscovered several times by different researchers independently from each other.

A NN can be seen as a computational graph like the one seen in Figure 2.5. Computational graphs are an important part of backpropagation and are a way to graphically display mathematical functions as nodes and edges in a graph. Nodes in a computations graph represent variables and operations such as addition,

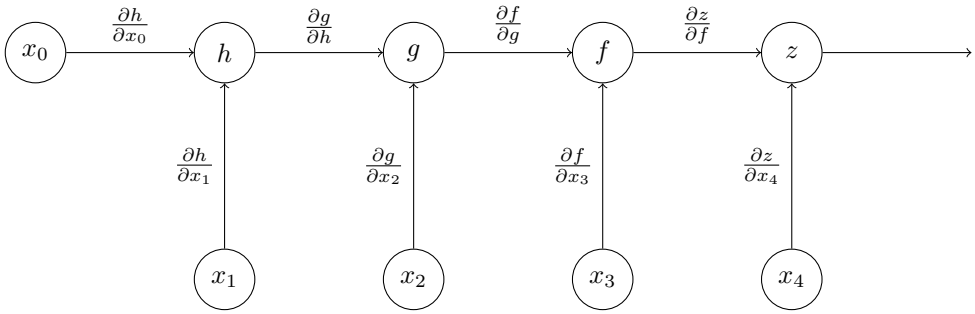


Figure 2.5: Computational graph

subtraction, multiplication, and division. Edges show how the computation is applied. In general, this means that if we want the gradient of a node in the graph with respect to some other node further down the graph. We only need to multiply the gradients along that path. To illustrate this, let us look at an example using the graph displayed in Figure 2.5. The edges on the graph already show the partial derivative of the node that the edge goes to with respect to the node where the edge comes from. Nevertheless, the question remains, how the partial derivative can be calculated if intermediate nodes are along the path. If we, for example, want the partial derivative of $\frac{\partial z}{\partial h}$, how can it be calculated? As we said, this can be done by multiplying the edges along path, for $\frac{\partial z}{\partial h}$ the answer is $\frac{\partial z}{\partial h} = \frac{\partial z}{\partial f}\frac{\partial f}{\partial g}\frac{\partial g}{\partial h}$ which is just the path between the nodes. More formally, we are traversing the graph in reverse topological order and applying the chain rule. Even though our example uses a tree, the algorithm can be applied to arbitrary computational (directed and acyclic) graphs. Furthermore, the algorithm is not limited to scalar values and applies to tensors too.

Since a NN is a computational graph, we can calculate the partial derivative of the loss for any arbitrary weight by following the edges along the path. That is essentially how backpropagation works and how the updates of weights can be done. Because the paths share partial derivatives, backpropagation lends itself to dynamic programming, making the computations more efficient.

2.1.4 Generalization

We have already mentioned that our model $\hat{f}_{\boldsymbol{\theta}}$ will not be exactly as the true function f due to some noise $\boldsymbol{\epsilon}$. The loss provides us with more details that can be analyzed and decomposed to get a better understanding of where the error stems from. The concepts we outline in the section apply to prediction but can

be transferred to a classification setting with some modifications.

Let us assume we have a model $\hat{f}_{\boldsymbol{\theta}}$ trained on the training data $\mathcal{D}_{\text{train}}$ using Equation (2.4) as our loss function. Furthermore, assume that we want to use our model to predict a new unseen test observation $(x^{(0)}, y^{(0)})$. Now, we want to find the expected value of the error between the predicted response and the true response $\mathbb{E}[y^{(0)} - \hat{f}_{\boldsymbol{\theta}}(x^{(0)})]^2$. Using the fact that $y^{(0)} = f(x^{(0)}) + \epsilon$, we can decompose the expected value into three different terms as seen in Equation (2.8).

$$
\begin{aligned}
&\mathbb{E}[y^{(0)} - \hat{f}_{\boldsymbol{\theta}}(x^{(0)})]^2 \\
=&\mathbb{E}[(y^{(0)})^2 + \hat{f}_{\boldsymbol{\theta}}(x^{(0)})^2 - 2y^{(0)}\hat{f}_{\boldsymbol{\theta}}(x^{(0)})] \\
=&\mathbb{E}[(y^{(0)})^2] + \mathbb{E}[\hat{f}_{\boldsymbol{\theta}}(x^{(0)})^2] - \mathbb{E}[2y^{(0)}\hat{f}_{\boldsymbol{\theta}}(x^{(0)})] \\
=&\text{Var}[y^{(0)}] + \mathbb{E}[y^{(0)}]^2 + \text{Var}[\hat{f}_{\boldsymbol{\theta}}(x^{(0)})] + \mathbb{E}[\hat{f}_{\boldsymbol{\theta}}(x^{(0)})]^2 \\
&- 2\mathbb{E}[y^{(0)}]\mathbb{E}[\hat{f}_{\boldsymbol{\theta}}(x^{(0)})] \\
=&\text{Var}[y^{(0)}] + f(\boldsymbol{x}^{(0)})^2 + \text{Var}[\hat{f}_{\boldsymbol{\theta}}(x^{(0)})] + \mathbb{E}[\hat{f}_{\boldsymbol{\theta}}(x^{(0)})]^2 \\
&- 2f(x^{(0)})\mathbb{E}[\hat{f}_{\boldsymbol{\theta}}(x^{(0)})] \\
=&\text{Var}(\boldsymbol{\epsilon}) + \text{Var}[\hat{f}_{\boldsymbol{\theta}}(x^{(0)})] + (f(x^{(0)}) - \mathbb{E}[\hat{f}_{\boldsymbol{\theta}}(x^{(0)})])^2 \\
=&\text{Var}(\boldsymbol{\epsilon}) + \text{Var}[\hat{f}_{\boldsymbol{\theta}}(x^{(0)})] + (\text{Bias}[\hat{f}_{\boldsymbol{\theta}}(x^{(0)})])^2.
\end{aligned}
\tag{2.8}
$$

Equation (2.8) displays the decomposition known as the bias-variance trade-off. Let us take a close look at Equation (2.8).

$\text{Var}(\epsilon) = \sigma$ can not be reduced regardless of changing the model or tuning its parameters. The term stems from measurement error, and can not be reduced unless we have measurements without error. Hence this error is beyond our control even if the true $f(x^{(0)})$ is known.

$\text{Var}(\hat{f}_{\boldsymbol{\theta}}(x^{(0)}))$ is the expected deviation around the mean at $x^{(0)}$. In other words, the variance of the prediction at $x^{(0)}$. In ML, when there is too much variance, we say that the model is overfitting the data. That means the model actually is modelling the random noise rather than the relationship between the covariates and the response.

$(\text{Bias}[\hat{f}_{\boldsymbol{\theta}}(x^{(0)})])^2$ measures how much the prediction differs from the true mean. In ML, when there is too much bias, we say that the model is underfitting the data. Since the model is not able to model the relevant relationships between the covariates and the response.

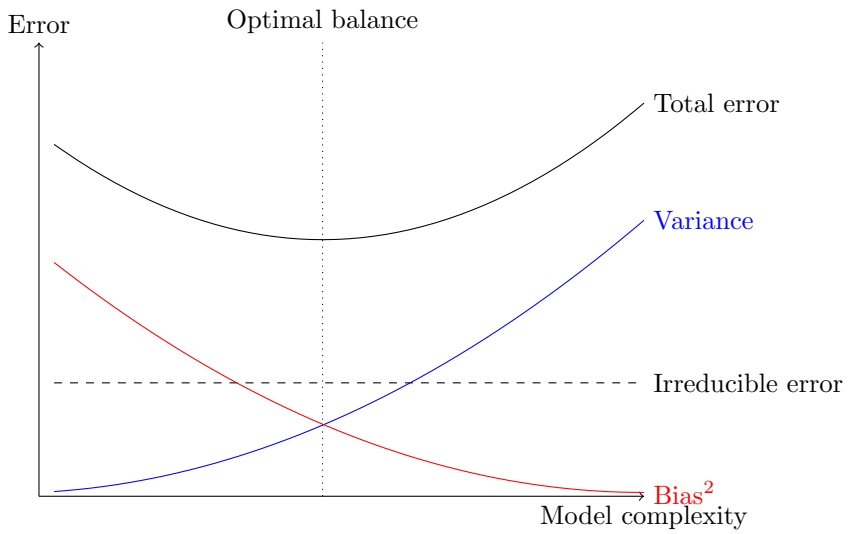


Figure 2.6: Bias-variance trade-off

Equation (2.8) is called the bias-variance trade-off since when we try to decrease the bias, the variance increases, and the same applies when we try to decrease the variance. Nevertheless, the increase and decrease rates are not necessarily the same. Hence, the goal is to find the minimum of the sum of these quantities and not their individual minimum. Figure 2.6 illustrate these concepts, and as we can see, the goal is to find the optimal balance implying that we want to minimize the sum of the quantities. Figure 2.6 clearly illustrates which components the total error consists of and that no matter what the model complexity is, the irreducible error can not be reduced by the selected model. In the same figure, we can see that the x-axis is denoted model complexity. In terms of NN, model complexity mostly equates to the number of hidden layers and the number of neurons in each layer. Nevertheless, there are other components of a NN that also influence its model complexity. Note that the behavior between the variance and squared bias displayed in Figure 2.6 holds in general. However, the slopes of the curves displayed are just an example. The slops can look a lot different from what is shown in the figure.

Generalization for neural network

A NN learns iteratively, thus, the term model complexity applies in a different way for a NN than other types of ML models. When we talk about model complexity

for NN, we need to consider both the architecture and how many optimization steps the learning algorithm takes. For each step the learning algorithm takes, the weights change, and so does the function it represents. At the initialization, the weights are often initialized to small values, so the approximate function is more "linear". As we start to train, the approximate function gets less "linear" because the weights are updated to fit the data. Hence, Figure 2.7 provides a better meaning of the bias-variance trade-off for a NN than Figure 2.6.

For a NN, we can look at the bias-variance trade-off from a training perspective, what it means to overfit and underfit. Figure 2.7 illustrates these concepts in terms of the data $\mathcal{D}$ and the number of training iterations. When a NN fits the data, it should fit the data better for each iteration of training, and the training loss should decrease. However, if the training loss decreases but the test loss starts to increase, we know the model is starting to overfit the training data. On the other hand, if the training process stopped, and the training and test loss is still decreasing, the model is underfitting. Since if we continue to train, the loss can be significantly decreased. Another type of underfitting is when the training and test error converge, but we know it should be significantly lower. At that point, we have a model not expressive enough to capture the underlying relationship in the data, which is also underfitting. Overall, the goal is to minimize the training error without the test error increasing.

Generalization Techniques

We have already seen in Figure 2.6 that the main component to achieve model generalization is to adjust model complexity. In this section, we are going to look at techniques that can help combat overfitting. Fixing underfitting is usually easier than fixing overfitting in a NN. If the model underfits, we can fix that by adding more hidden units or the number of layers for a NN since it gives the network more flexibility to capture nonlinearities in the data. Overfitting, on the other hand, is harder since restricting the number of hidden units and the number of layers hurts the expressiveness of the model. By restricting expressiveness, the model might not be able to model interesting latent relationships in the data. From statistics, we know that as the amount of data increases, the influence of the prior we have on the model diminishes. In other words, one way to combat overfitting is to increase the amount of training data. However, getting more data can be expensive or not possible. Hence, there are numerous techniques to fight overfitting. We will briefly go through some of the most widely adopted techniques. Notice that the list is not exhaustive.

Early Stopping comes in several variants in terms of stopping criteria in case the loss does not decrease [Prechelt, 2012]. The goal of early stopping is to avoid overfitting. It is done by monitoring the validation loss during

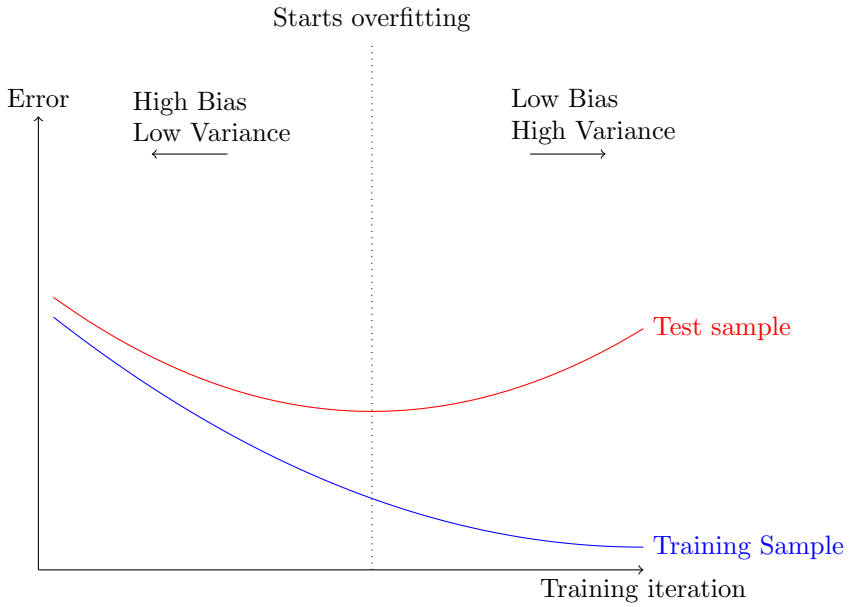


Figure 2.7: Bias-variance trade-off in terms of the data $\mathcal{D}$ and number of training iteration.

training, as shown in Figure 2.7 (The figure displays test loss, but during training of a NN, validation samples are used, not test samples). Early training allows the network to train as long as the validation error decreases. However, if, for example, the validation error does not improve for three iterations, the training procedure is stopped. In the beginning, the NN starts with small weights, thus represents a more "linear" function. As the training goes on, the approximate function gets less linear since the weights are updated to fit the data. Hence, early stopping tries to stop the training procedure before the approximating function becomes too complex.

Parameter Norm Penalty adds a new term to the loss function that penalizes the model for the weight sizes [Tibshirani, 1996; Hoerl and Kennard, 1970]. Smaller weights provide a less complex approximate function, and thus prevents overfitting. On the other hand, if the weights get too constrained, the model will underfit as we have already discussed. Hence the amount of penalty is a hyperparameter. Two of the most known variants are L^1 and L^2 regularization. L^2 penalty is also known as weight decay, which is analogous to ridge regression for linear models. It adds the square of

the magnitude of the weights to the loss function. This means that larger weights are given a higher loss. This is the MSE loss with L^2 penalty added (in this example there is only a single weight),

$$\frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2 + \lambda w^2. \tag{2.9}$$

From a statistical point of view, having Gaussian priors in a Bayesian NN is equivalent to using a "normal" NN with L^2 regularizers. Similarly, using Laplacean priors in a Bayesian NN is equivalent to applying L^1 regularizers in a "normal" NN. To prove this for the Gaussian prior case. Assume that we have the data set $\mathcal{D}$. Furthermore, assume that the input and output have the following relationship

$$\begin{aligned} y^{(i)} &= wx^{(i)} + \epsilon \text{ for } i = 1, ..., n \\ \epsilon &\sim \mathcal{N}(0, \sigma^2) \end{aligned} \tag{2.10}$$

where we want to infer the parameter w. From a probabilistic perspective, Equation (2.10) provides the Gaussian likelihood,

$$\prod_{i=1}^{n}\mathcal{N}(y^{(i)}|wx^{(i)}, \sigma^2). \tag{2.11}$$

We can regularize the parameter w in Equation (2.11) by imposing a Gaussian prior $\mathcal{N}(w|0, \lambda^{-1})$, where λ is a strictly positive scalar. As a result of combining the prior and the likelihood, we get,

$$\prod_{i=1}^{n}\mathcal{N}(y^{(i)}|wx^{(i)}, \sigma^2)\mathcal{N}(w|0, \lambda^{-1}). \tag{2.12}$$

If we take the natural logarithm of Equation (2.12), we get:

$$\sum_{i=1}^{n}\frac{1}{\sigma^2}(y^{(i)} - wx^{(i)})^2 - \lambda w^2 + \text{constant}. \tag{2.13}$$

By looking at Equation (2.13), it should be quite clear why L^2 penalty is equivalent to applying Gaussian prior, since L^2 penalty equals MSE $+ \lambda w^2$.

A similar approach can be used to demonstrate the relation between L^1 regularization and Laplacean prior.

Data Augmentation tries to synthesize more training data without access to the underlying data generation process p_{data}. This can be done, for example, by duplicating the data and add noise to the duplicates so that we get new "samples". By having more data, it becomes harder to overfit [Shorten and Khoshgoftaar, 2019].

Dropout is a regularization technique for a NN by Srivastava et al. [2014]. During training, each neuron in a network has some probability p to be turned off (known as the dropout rate) following a Bernoulli distribution. It makes it harder for neurons to collaborate and overfit the training data since there is always a chance for them to be dropped. During testing, on the other hand, none of the neurons will be turned off. However, the output of the neurons will be scaled down by the dropout rate because none of the neurons will be turned off, and so the values get bigger.

Ensemble training trains several copies of the same model with different datasets and different initializations and combines their predictions [Opitz and Maclin, 1999; Freund and Schapire, 1996; Breiman, 1996]. Dropout can be seen as a form of ensemble training since only a subnetwork is used for the forward pass during training. However, using several networks and taking their average for prediction or maximum voting for classification is not common because training several networks requires a lot of computational resources. Combining several networks can make the prediction/classification better if we can make several training sets that are not fully correlated.

We can show this by observing the variance of the mean. Assume that we have n samples of the random variable x that are identically distributed, each with mean μ and variance σ^2, and that the variables have a positive

correlation ρ, $\mathrm{Cov}(\mathrm{x}^{(i)}, \mathrm{x}^{(j)}) = \rho\sigma^2,\ i \neq j$. The average variance is:

$$\begin{aligned}
\mathrm{Var}(\bar{\mathrm{x}}) &= \mathrm{Var}(\frac{1}{n}\sum_{i=1}^{n}\mathrm{x}^{(i)}) \\
&= \sum_{i=1}^{n}\frac{1}{n^2}\mathrm{Var}(\mathrm{x}^{(i)}) + 2\sum_{i=2}^{n}\sum_{j=1}^{i-1}\frac{1}{n^2}\mathrm{Cov}(\mathrm{x}^{(i)}, \mathrm{x}^{(j)}) \\
&= \frac{1}{n}\sigma^2 + 2\frac{n(n-1)}{2}\frac{1}{n^2}\rho\sigma^2 \\
&= \frac{1}{n}\sigma^2 + \rho\sigma^2 - \frac{1}{n}\rho\sigma^2 \\
&= \rho\sigma^2 + \frac{1-\rho}{n}\sigma^2 \\
&= \frac{1-(1-n)\rho}{n}\sigma^2.
\end{aligned} \tag{2.14}$$

As we can see, as long as the correlation between the networks is not perfect, we get reduced variance, and that is why ensemble learning can be used to combat overfitting.

2.1.5 Techniques to improve neural network

This section outlines techniques that we have not examined yet to improve a NN. The methods we show are mainly those used in this thesis. As such, this section is not an exhaustive list of techniques to improve NNs.

Batch normalization

Batch normalization (often called batch norm) is a technique that normalizes the output of a layer in a NN. Batch norm can be used for any layer in a NN, and not limited to the input layer. Assume that we have a mini batch B of size m. The empirical mean and variance of the batch is thus

$$\begin{aligned}
\mu_B &= \frac{1}{m}\sum_{i=1}^{m}x^{(i)} \\
\sigma_m^2 &= \frac{1}{m}\sum(x^{(i)} - \mu_B)^2.
\end{aligned}$$

The normalized $x^{(i)}$ is

$$\hat{x}^{(i)} = \frac{x^{(i)} - \mu_B}{\sqrt{\sigma_m^2 + \epsilon}}, \tag{2.15}$$

where ϵ is a small constant added for numerical stability. The output $\hat{x}^{(i)}$ has mean zero and unit variance. Notice that if $\boldsymbol{x}^{(i)}$ is a vector, each dimension is normalized separately.

Ioffe and Szegedy [2015] proposed this technique to reduce the covariate shift, but it has many other benefits. The original authors define the covariate shift as a shift in the distribution of network activations in each layer caused by a change in the network parameters during training. This behavior is unwanted since each successive layer need to adjust to a new distribution each time. The covariate shift is especially true for deeper networks where the shift can be significant in deep layers with minor changes in shallow layers. The training process is sped up and more reliable by removing this unwanted behavior since the layers will not receive activations of arbitrary distributions.

Batch norm uses the mean and variance of a mini-batch to normalize the output. Thus, there will be noise added. This noise makes it harder for the network to overfit the dataset and more robust to different weight initialization schemes and learning rates. Thus, the networks using batch norm can have a higher learning rate without exploding and vanishing gradients.

Active learning

Sssume that we have a dataset that is labeled. However, we also have more data from the same distribution but unlabeled. For example, there are many tweets, but the amount of tweets that have been labeled for sentiment is much less. If we train a model and the performance of our model is not satisfactory, and we have a lot of unlabeled data. One way to improve the model is to use AL.

AL is a framework to increase a model's performance by intelligently getting more training data. AL tries to do this by choosing which unlabeled data sample to label instead of randomly labeling more data. AL tries to ask an "oracle" to label samples that it thinks will have a significant impact on the training of the model (Figure 2.8 illustrates this concept). AL is valuable and vital since getting labeled samples is time-consuming, very difficult, and expensive, depending on the task to be solved. Although human labor can be cheap relative to the reward in some situations, it is not valid for all data types. Some types of data might require experts. For example, annotating disease and gene mentions for biomedical information extraction often requires PhD-level biologists.

In AL the process of selecting observations to label is called sampling. There have been developed numerous sampling strategies for AL. We are not going to go into details on different sampling strategies in this thesis. Nevertheless, Figure 2.9 displays why AL is useful and that the sampling strategy chosen matter.

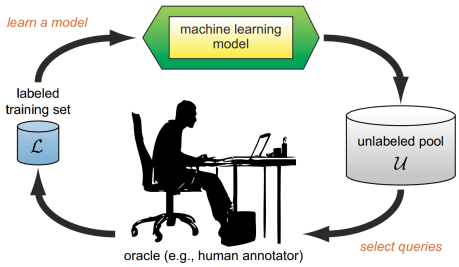


Figure 2.8: Pool-based active learning cycle.

Source: Settles [2012]

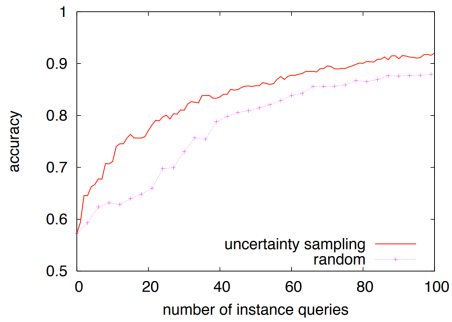


Figure 2.9: Learning curves for text classification: baseball vs. hockey. Curves plot classification accuracy as a function of the number of documents queried for two selection strategies: uncertainty sampling (AL) and random sampling (passive learning). We can see that the AL approach is superior here because its learning curve dominates random sampling.

Source: Settles [2012]

2.2 Convolutional Neural Network

A CNN is a class of neural networks with unique layers called convolutional layers. Convolutional layers consist of kernels that are tensors of weights. CNNs are famous and high performing in the field of computer vision, and the considerable success in this field can largely be attributed to CNNs [Krizhevsky et al., 2012]. However, CNNs are not limited to computer vision but can also be applied to, for example, natural language processing (NLP) [dos Santos and Gatti, 2014]. The reason CNNs are good at image processing is due to their ability to craft good and useful features automatically. Before CNNs entered the stage, features were handcrafted, both time-consuming, laborious, and subjective. Whereas CNNs only need the computational resources to compute the features and find feature extractors that no one would have thought of. The idea of using convolution to detect features in, for example, images is not new. Nevertheless, coming up with useful kernels is hard. The feature extraction process automated also increases the number of feature extractors used since finding them is just training a network.

2.2.1 Properties

The main difference between convolutional layers in CNN and dense layers in FNN is the patterns they learn. Dense layers try to find global patterns in the data, whereas convolutional layers try to find local patterns. Having layers that learn local patterns gives CNN two interesting properties, translation invariance and spatial hierarchy of patterns that we will briefly go over:

Translation invariance. When dense layers learn a pattern in the data, the location matters since which neuron activates matters. Remember that every input neuron is connected to every neuron in the layer after the input layer with its weight in a dense layer. Thus different transformations are applied at every location. Convolutional layers, on the other hand, apply the same transformation at every location, which makes the neurons, not location-sensitive [LeCun, 2012; Jaderberg et al., 2015; Gens and Domingos, 2014]. Thus, if we want to classify images of cats, it does not matter where in the image the cats appear.

Spatial hierarchy of patterns. This property is fairly natural to us humans since the visual world is spatially hierarchical. Figure 2.10 displays what this property essentially means. We see that each layer of the network extracts and combines the feature to a higher level of abstraction [Mahendran and Vedaldi, 2016; Bengio et al., 2013; Zeiler and Fergus, 2013]. This hierarchy of patterns also applies to NLP, where we have phonemes that combine into morphemes, which in turn combines into words and continues.

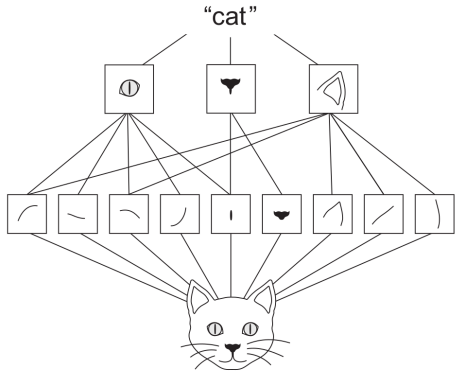


Figure 2.10: Spatial hierarchy of patterns. The network is fed an image of a cat and extracts its features. The first convolutional layer tries to find local features, and the layers coming after combines the features extracted into more and more abstract features.

Source: Chollet [2018]

A NN having local connections reduces the number of parameters that need to be learned during training. This implies it will be faster to train the network and harder to overfit. As we have seen from Section 2.1.4, as the number of learnable parameters (model complexity) decreases, so does the chance for overfitting.

2.2.2 Components

In this section, we will look at the different building blocks that make up a CNN. Not all of the components are necessary, but they are commonly used in a CNN. When we go over the different examples in this section, we will use 2-dimensional tensors. However, the operations described in this section can be applied to arbitrary-dimensional tensors. Note that when we refer to the tensors inputted and outputted from the different layers in a CNN, we refer to them as feature maps.

Convolution

The two dimensional convolution operation (convolution is not limited to two dimensional) in a CNN is defined as

$$\boldsymbol{O}(l,j) = (\boldsymbol{I} * \boldsymbol{K})(l,j) = \sum_{m=1}^{M}\sum_{n=1}^{N} \boldsymbol{I}(l+m, j+n)\boldsymbol{K}(m,n), \tag{2.16}$$

where we assume that the kernel is of size $M \times N$. The final feature map $\boldsymbol{O}$ is computed by sliding the kernel over $L \cdot J$ number of windows assuming that the final feature map is of size $L \times J$ and use stride $s = 1$. The notion of stride used in CNN deals with the distance the kernel is moved between two successive windows. Using stride of, for example, $s = 2$ downsamples the feature map by a factor of 2. Although stride can be used to downsample feature maps, it is most common to use pooling (discussed in section 2.2.2) to downsample the feature maps. If we do not want the output feature map to be downsampled, padding can be used. Padding is adding the input feature map with a border of **0**s to increase its size. The width of the padding can be controlled, and it is common to pad either input and output feature maps such they have the same size, or to use no padding at all.

Before moving on, let us look at an example of how the convolution operation works in practice. Figure 2.11 illustrates Equation (2.17):

$$\begin{aligned}
\boldsymbol{O}(l,j) = (\boldsymbol{I} * \boldsymbol{K})(l,j) &= \sum_{m=1}^{M}\sum_{n=1}^{N} \boldsymbol{I}(l+m, j+n)\boldsymbol{K}(m,n) \\
\boldsymbol{O}(1,4) = (\boldsymbol{I} * \boldsymbol{K})(1,4) &= \sum_{m=1}^{3}\sum_{n=1}^{3} \boldsymbol{I}(1+m, 4+n)\boldsymbol{K}(m,n) \\
&= 1\cdot 1 + 0\cdot 0 + 0\cdot 1 + 1\cdot 0 + 1\cdot 1 + 0\cdot 0 + 1\cdot 1 + 1\cdot 0 + 1\cdot 1 \\
&= 4.
\end{aligned} \tag{2.17}$$

The stride $s = 1$ in this example, after sliding the windows over $\boldsymbol{I}$ and doing the same operation as shown in Equation (2.17) at every location. We get the resulting feature map $\boldsymbol{O}$ of size 5×5. Notice that only a single kernel is used for this example. Moreover, that the number of output feature maps equals the number of kernels and is independent of the number of input feature maps. In conclusion, the convolution layers are essentially a stack of kernels $\boldsymbol{K}$ with trainable weights that are convolved over the input feature map(s) $\boldsymbol{I}$.

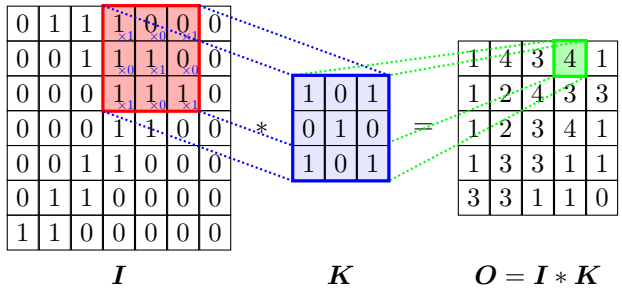


Figure 2.11: How the convolution operation works. Here we have a 7×7 input $\boldsymbol{I}$ and a single 3×3 kernel $\boldsymbol{K}$. $\boldsymbol{O}$ is the resulting output after applying the kernel with stride $s = 1$.

Pooling

The pooling operation is similar to convolution because we slide a window over a feature map. Instead of taking the convolution operation, pooling often takes the max or average operation over the window. That means there is no kernel applied to the window. Instead, we take the max or average operation over all the values in the window. The output will be a single value (or a tensor, depending on the dimensions) similar to the convolution operation. Figure 2.12 displays an example of how pooling, or more specifically, max-pooling, works on a feature map. The pooling operation is essentially used to downsample feature maps. Feature maps are downsampled to reduce the number of parameters, making it harder to overfit. Besides, downsampling makes it possible for each successive convolutional layer to get a broader view of the initial feature map. The kernel covers a more significant proportion of the input feature map for each layer. Hence, the successive layers can combine and build more abstract and complicated features [Le and Borji, 2018; Luo et al., 2017].

As we already know, a larger stride can be used to downsample. However, pooling has proven to be better, especially max-pooling [Scherer et al., 2010]. Feature maps tend to encode the presence of a concept over the different tiles. If we use max-pooling, the resulting feature map will look for the maximal presence of concepts that the kernels extract. If average-pooling is used, then the average presence is encoded, which has been proven to be less valuable than maximal presence. We know that another alternative is to use a larger stride to downsample, but that approach produces sparse feature maps. Both sparse feature maps and average-pooling can miss or weaken the information concerning feature presence. Therefore, it is most common to use max-pooling to downsample feature

maps. On the other hand, using a larger stride for downsampling can also be advantageous since the network can learn specific downsampling strategies that can be handy in some situations. It has been shown, for example, in variational autoencoders (VAEs) or generative adversarial networks (GANs) that getting rid of the pooling layer is essential for training good generative models.

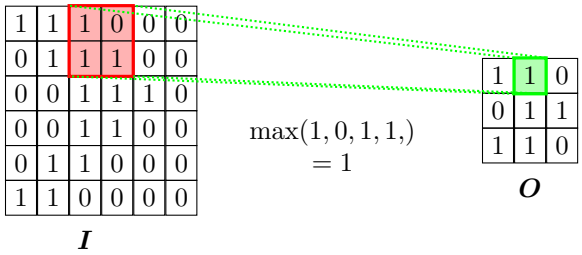


Figure 2.12: How the max-pooling operation works. Here we have a 6×6 input feature map $\boldsymbol{I}$ and the output feature map 3×3 $\boldsymbol{O}$. $\boldsymbol{O}$ is the resulting output after applying the max-pooling operation over windows of size 2×2 with stride $s = 2$.

2.2.3 Tying the building blocks together

Now that we have looked at the different operations of a CNN, it is time to fit all the components together. Since the convolutional layers utilize spatial hierarchy of patterns, the convolutional layers are often stacked along with other types of layers. Nevertheless, a CNN can contain dense layers as long as convolutional layers are included. Figure 2.13 displays what a CNN architecture can look like when the different building blocks are fit together. The typical setup is first to use a convolutional layer, then an activation function is added, as in dense layers. After the activation function comes the pooling layer that reduces the size of the feature map. These three types of layers are often considered as one "unit" that are stacked together, as shown in Section 2.2. The activation function serves the same purpose in a CNN as it does for dense layers. After those "units" comes the dense layer(s) which is/are not mandatory, but often improves the result. Since the convolutional layers have extracted useful features, dense layers can be used to extract global relationships without being affected by the locality of objects. As always, the network ends with an output layer that has an activation function and number of neurons depending on the task to be solved. Although we have stated in which order the building blocks are usually used, other combinations are also possible and can sometimes be even better.

As we progress down the layers in a CNN, it is common that the number of feature

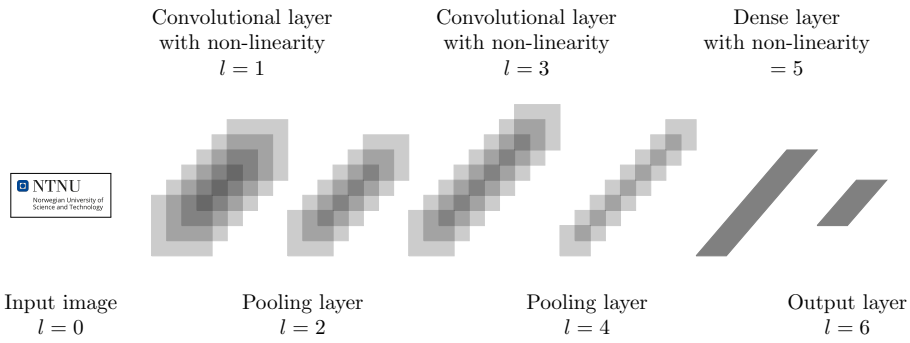


Figure 2.13: The architecture of the original convolutional neural network, as introduced by LeCun et al. [1989], alternates between convolutional layers, including hyperbolic tangent activation function and pooling layers. In this illustration, the convolutional layers already include non-linearities. The feature maps of the final pooling layer are fed into the actual classifier consisting of an arbitrary number of fully connected layers. The output layer usually uses softmax activation function if number of classes $C > 2$, else the sigmoid activation function is used.

maps increases and their size decrease. As the feature maps' size decreases, each map contains a larger portion of the input [Araujo et al., 2019; Luo et al., 2017]. In other words, more information is stored in each feature map. Moreover, as each feature map contains a feature, the increasing number of feature maps implies more features are constructed.

2.3 Explainable Artificial Intelligence

XAI is a collection of techniques and methods that aim to make ML models interpretable for humans. Realizing the usefulness and need for interpretability, the field of XAI is gaining much momentum. Researchers use the term XAI for a broad range of methods that produce a wide range of different outputs. Thus, the term interpretability is not concise in its meaning. This section opens with an attempt to give the term interpretability a more concise meaning and motivation for why it is needed. We will look at the different aspects of interpretation methods like properties and how they can be divided and evaluated. The section ends with a closer look at some specific interpretation methods used in this thesis.

2.3.1 Definition

There is no widespread agreement as to what interpretability means in the context of ML. Furthermore, terms like interpretability and explainability are often used interchangeably in the literature. According to the Cambridge Dictionary, interpret means "to decide what the intended meaning of something is". The Cambridge Dictionary definition is a general definition of the word interpret, not anchored to a specific context. In this section, we will look at what interpretability means in the context of ML.

Many research articles are trying to define the term interpretability. Doshi-Velez and Kim [2017] define interpretability as "the ability to explain or to present in understandable terms to a human". However, they agree that a formal definition of interpretability is still missing. Another definition by Murdoch et al. [2019] says that interpretability or more specifically, interpretable ML as "the use of ML models for the extraction of *relevant* knowledge about domain relationships contained in data". They define knowledge as *relevant* if it provides insight for a specific audience into a chosen domain problem. Miller [2019] builds his definition upon Biran and Cotton [2017]'s definition of interpretability that states "the degree to which an observer can understand the cause of a decision".

Lipton [2018] does not give a single definition of interpretability. According to him, interpretability is not a monolithic concept but reflects several distinct ideas. Hence a single closed definition of interpretability can not be given. There are also other definitions of interpretability from other research fields, such as psychology by Lombrozo [2006]. She states that "explanations are... the currency in which we exchanged beliefs".

As we can see, there is no unanimous agreement upon what interpretability is. However, we still have a vague idea of what interpretability means. Like the literature, we will also use the different terms of interpretability, such as explainability, interchangeably throughout this thesis.

2.3.2 Motivation

There is a need for XAI since complex models are black-boxes (as displayed in Figure 2.14) where we do not understand the model's internal reasoning process. Black-box in this context is a term used for models that do not reveal anything about its internal reasoning process in a way that a human can understand. Having an interpretable ML model in itself is essential to detect the failures, thereby improving the model [Hoiem et al., 2012]. Currently, with black-box models, the only insight the user gets is through evaluation metrics. The metrics are helpful but do not provide the user with an explanation. In this section,

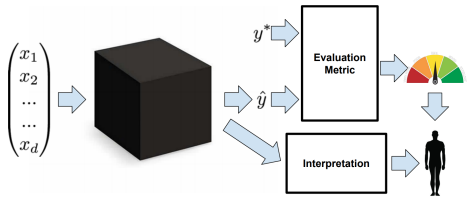


Figure 2.14: Typically, when ML models are evaluated, only the output $\hat{y}$ and ground truth y are needed. However, to understand a black-box model, some explainability method is needed to increase our understanding of the model.

Source: Lipton [2018]

we are going to look at the motivations for explanations for ML models. There is a need for interpretable ML that comes from the demand of satisfying other essential criteria of ML models. These criteria are fairness, privacy, robustness, trust, and so on. In this section, we are going to take a closer look at these criteria.

According to Doshi-Velez and Kim [2017] the need for interpretability comes from incompleteness in the problem formalization. The incompleteness can be thought of as goals we want the model to satisfy but is hard or impossible to quantify in numbers. If we have explainable models, gaps that come from not satisfying these goals and their effect become visible to us, making us aware of the models' flaws. Lipton [2018] says that the need for interpretability comes from a mismatch between the real world cost in a deployment setting and the cost from the formal objective of supervised learning (such as the MSE loss for prediction). He implies that the metrics that can be produced automatically do not fully reflect the goals we want the model to fulfill. Both these papers imply that we want to optimize the ML models for an objective that can not be quantified using metrics. We need interpretable ML models to ensure that we can somehow fulfill those objectives to some extent, or at least be aware of what is missing.

Some of the criteria we may want the ML model to fulfill through interpretability are:

Scientific understanding. As we have said earlier, we create ML systems for prediction/classification or gaining insights into the data. Interpretable ML models can help us with that. By explaining a prediction, we humans can gain knowledge by converting the explanation into knowledge. Even if the classification/prediction is useful, it does not help us expand our

understanding of the data, which is often desired.

Trust and safety. For most tasks, testing a system for all possible scenarios is impossible. It would be impossible to list all cases where the system might fail. Even if it is possible, the resources needed to verify all the outputs would not be possible. Hence, interpretable systems are needed to gain the trust of the system users. In order to gain trust, the model can, for example, explain where it is making mistakes in the input space, instead of only providing metrics that measure which observation was given correct classification. Having good metrics certainly gives users confidence but can not ensure how the model performs in a real-world scenario.

Fairness and ethics. If the data the model is trained on contains some undesired biases, the model will likely learn those biases. However, we do not know what kind of biases the model has learned, and we probably do not want it deployed without knowing that. Even if we could encode against biases, the notion of fairness is often too abstract to be fully encoded. Assuming that we can list all biases before training is unreasonable.

Mismatched objectives. What we want the model to learn might not be possible to formalize as pairs of covariates and response. Instead, we train our model to optimize for a proxy function that does not entirely cover our objective. If we have explanations, we can use those to help the model learn our original objective. Alternatively, at least verify the trade-off that the model has made after training.

Trade-offs between objectives. We might want to optimize the model for several criteria that we have encoded into the model. However, seeing how the model trades off the criteria globally or for each observation might be essential to know since it is impossible to satisfy them. By seeing the trade-off, we can have better evaluations of the model, whether it satisfies our needs or not, and is the trade-off acceptable to the user.

This list of criteria is not exhaustive. However, we have tried to condense our findings in the literature to these criteria that we find essential.

2.3.3 Properties of interpretation methods

XAI methods can be characterized with certain properties [Robnik-Šikonja and Bohanec, 2018; Murdoch et al., 2019]. The list of properties presented in the literature is often massive, and differs between different research articles. This section introduces those most common properties and summarizes them in a concise manner.

Accuracy. According to Murdoch et al. [2019], accuracy can be divided into **predictive** and **descriptive** accuracy. Predictive accuracy is the accuracy provided through the training of the model. If the predictive accuracy is not high, it is reasonable to expect that the extracted explanations will not be used. Also, the extracted relationships will not be very accurate. Notice that when the word predictive accuracy is used, it is expected to reflect the model's overall performance. If the training and test data are skewed, and the model performs badly, the predictive accuracy should reflect that fact. Predictive accuracy should be a measure of accuracy that reflects the model's weaknesses.

The authors define descriptive accuracy as "the degree to which an interpretation method objectively captures the relationships learned by ML models". We want the interpretation method we use to be faithful to the relationship the model learns. However, that is often not the case. As some very flexible models can learn highly non-linear functions, it is difficult to extract and explain the learned relationship.

Ideally, we want both predictive and descriptive accuracy to be high, but it is often not possible. As the flexibility of a model is increased, the predictive accuracy should increase. However, it also becomes more challenging to explain the learned relationship. This is related to the bias-variance trade-off that we described in Section 2.1.4.

Relevancy. An interpretation is "relevant" if it provides insight for a particular audience into a chosen domain problem [Murdoch et al., 2019]. The explanation extracted does not necessarily need to be understood by the general public. It depends on who is going to use the explanation. If the aim is scientific understanding or improving the ML model, the explanation does not need to be understood by everybody and can contain scientific terms. On the other hand, if the model is used to evaluate if someone can get a loan in a bank, the explanation must be easily understood by the general public. In conclusion, the relevancy of an interpretation method should match the user group of the model.

Stability. As we have talked about the trust of the ML models in Section 2.3.2, there is also a need for trust in the explanation methods. This property is highly related to the accuracy, but we are making it a separate point to emphasize its importance. If an explanation changes significantly because of small perturbations of a sample, it is probably not a good idea to trust the explanation method. Hence, when an explanation method is evaluated, stability is essential. However, it does not mean that the explanation method is not useful because it can be stable in other parts of the input

space. Thus it might not provide value in a certain area of the input space, but can still be useful in other parts of the input space.

2.3.4 Taxonomy of interpretability

There is an agreement in the literature [Lipton, 2018; Murdoch et al., 2019; Ribeiro et al., 2016a; Molnar, 2020] that there are two types of interpretability. The first type is often known as model-specific interpretability, which assumes that the model itself is interpretable without any further intervention. The other type is known as post-hoc interpretability that assumes that interpretability can be achieved by using methods and techniques to probe the model after training. Since post-hoc methods are applied after training, they are often model agnostic, meaning that they are not tailored to a specific model type.

Model-specific interpretability

Model-specific interpretability is about models that are readily interpretable. The models often encode the relationship learned or learn in a way that is easy for the user of the model to understand. The models are constructed in a specific way that increases the descriptive accuracy. However, the increase in descriptive accuracy often comes with a cost in terms of predictive accuracy due to model flexibility. We are now going to take a look at different ways model-specific interpretability is achieved. Note that this division is not exclusive, and a lot of the ML models employ several of these methods.

Modularity: One way to obtain interpretable models is to use models that can be decomposed. That is, the prediction process can be divided and studied separately. This form of interpretability assumes that every module admits an intuitive explanation. One way, for example, is to look at the features separately in a generalized additive model [Hastie and Tibshirani, 1990], or have conditional independence in a probabilistic model that can be studied separately [Pearl, 1985; Koller and Friedman, 2009]. In DL, the attention mechanism [Kim and Canny, 2017] provides insight into a NN's inner working. This notion of interpretability assumes that independent structures exist in the model and the data. In this type of interpretability, the practitioner should be careful not to overinterpret independence that not necessarily exist.

Simulatability: This form of model-specific interpretability assumes that a human (the group of people the explanation is intended for) can understand the whole prediction process without looking at the modules separately. While the term simulatability might be ambiguous, we know that the human cognition limits the coverage of the term as it is intended for a human

to do the simulation. This type of interpretability places a considerable constraint on model flexibility to achieve interpretability since contemplating an entire model at once is not an easy feat. This type of interpretability is only possible with data that has few features, and the underlying relationship is simple. List of rules [Letham et al., 2015; Friedman and Popescu, 2008] and decision trees [Breiman et al., 1984] are often cited as simulatable methods [Murdoch et al., 2019] due to hierarchical decision-making and simplicity. However, this is only true for a list with a small number of rules and decision trees with limited depth.

Sparsity: This form of interpretability assumes that the model is interpretable because it is sparse, achieved by limiting the number of non-zero parameters. This type of interpretability assumes that a sparse set of features can summarize the underlying relationship, hopefully without a considerable drop in predictive performance. The sparse set of features should be intuitive and represent concepts that can easily be related to the model's output.

Sparsity is often achieved by penalizing the loss function for the coefficient's magnitude, such as Least absolute shrinkage and selection operator [Tibshirani, 1996] and sparse encoding [Olshausen and Field, 1997]. There are also feature selection methods like Akaike information criterion [Akaike, 1974] and Bayesian information criterion [Schwarz, 1978] that select a sparse set of features in order for the model to generalize, which in turn increases the interpretability. These methods rely on that a sparse set of feature can model the problem without a significant drop on predictive accuracy.

Feature engineering: Another way to achieve model-specific interpretability is to engineer a small set of intuitive features that covers the original features "well enough". Another way to achieve model-specific interpretability is to engineer a small set of intuitive features that covers the original features "well enough". This can be done by domain experts or using dimension reduction techniques like principal components analysis [Wold et al., 1987]. That said, all these methods still assume that the ML model can describe the relationship between the features and response, and achieve good predictive accuracy. Otherwise, having interpretable features would not help much in understanding the model.

Although there are several ways to classify model-specific interpretability as we have seen. They all place some constraints that most likely will decrease the predictive accuracy. Some problems might need very flexible models that this kind of interpretability can be hard to achieve. Thus there has been much research on post-hoc interpretability that we are going to look at now.

Post-hoc interpretability

Post-hoc interpretability is another way to obtain explanations from ML models. These methods do not place constraints on the ML models. As a result of that, the predictive accuracy is fixed. However, the explanations produced pay the price in the form of not giving an exact reflection of the model's inner working [Adebayo et al., 2020] (the degree varies). Nevertheless, they are still useful, especially in situations where the data is highly non-linear and requires models with high model complexity. The explanations the post-hoc methods provide often fall into one of two categories, namely global and local explanations. A local explanation tries to explain a single prediction without considering other observations in the data except for neighboring observations. While on the other hand, a global explanation tries to explain the underlying relationships in the data that should be valid for a large population. As this division exists in post-hoc interpretability, we will look at different types of post-hoc explanation methods from these two perspectives.

Global interpretation: These methods provide explanations and relationships in the data that holds for a large population, but it can also be adapted to a subpopulation.

Feature importance and interaction: These types of methods try to extract the importance or how much contribution each feature provide in the prediction of the different classes. However, this is a hard task, because there is nearly always interactions between features, and looking at individual features can be misleading. These types of methods can be designed to be applicable to all types of models [Altmann et al., 2010]. In some cases like Random Forest [Breiman, 2001], this kind of measure can be directly accessed without any additional tools. In addition to feature importance, there are methods to extract feature interactions [Tsang et al., 2018; Kumbier et al., 2018].

Statistical feature importance: If we can make assumptions about the underlying data generation process that is reasonable, we can construct a confidence interval and do hypothesis testing. This can help us determine which features are statistically significant, and is considered a form of global interpretation.

Visualization: These methods try to visualize the inner workings of a model and are mostly applied to NNs. For a CNN, for example, researchers have tried to visualize the feature maps [Zeiler and Fergus, 2013; Olah et al., 2017], to figure out what the model has learned. For recurrent networks, researchers have tried to visualize the state vectors [Karpathy et al., 2015] to understand the model.

Local interpretation: While global interpretation tries to find contributions and relationships that hold for a large population, local interpretation focuses on a single prediction and neighboring observations. This type of interpretation is useful when the practitioner is interested in individual observations and not essential relationships in a population.

Feature importance: Like in the global setting, our goal is to find an importance score for each feature. However, in this case, the importance score only needs to be valid locally. An example of this kind of method is the local interpretable model-agnostic explanations [Ribeiro et al., 2016b] that provides an explanation that holds for the given sample, and observations that are close to the given sample. It estimates feature importance by fitting a linear model in a limited space in the input space where the given sample is located.

Example based explanation: This approach justifies a prediction by providing observations that the model thinks are similar and are given the same prediction [Caruana et al., 1999]. The similarity can be measured by comparing the observation features using a distance measure like euclidean distance. Another approach is, for example, to use the hidden layer activation of a NN when the samples are fed to the network to measure the similarity of observations.

2.3.5 Evaluation of interpretability

Evaluating interpretability methods is hard and not clear [Murdoch et al., 2019; Doshi-Velez and Kim, 2017]. Researchers have cherry-picked samples from their experiments to support their claims, but it has been proven that, for example, visual assessments of the attention maps can be misleading [Adebayo et al., 2020]. The community has not yet settled on standard evaluation protocols. However, there are some promising works [Lipton, 2018; Gilpin et al., 2019; Doshi-Velez and Kim, 2017] ongoing. There is still a need for evaluation methods for descriptive accuracy and relevancy since claims need to be evaluated, and the evaluation should match the claimed contribution. Doshi-Velez and Kim [2017] propose a taxonomy for the evaluation of interpretability that is divided into three segments, application-grounded, human-grounded, and functionally-grounded evaluation.

Application-grounded Evaluation evaluates the interpretability of a system by conducting experiments within a real application with domain experts. It requires that the system is grounded in a real application with a specific user group in mind. However, using domain experts increases the cost in terms of time and resources, which is not always feasible. In addition, the

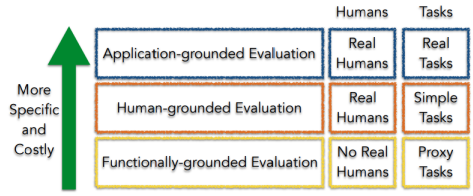


Figure 2.15: Taxonomy of evaluation approaches for interpretability

Source: adapted from Doshi-Velez and Kim [2017]

application-grounded evaluation is not an easy evaluation metric [Antunes et al., 2008]. However, the system is tested on a real scenario with real users, which provides strong evidence of its usefulness.

Human-grounded Evaluation is like an application-grounded evaluation in the sense it uses human subjects. However, trained experts can be expensive and not always possible, thus using not trained people is an alternative. These subjects will provide less accurate evaluation, nevertheless still useful since we still get to test a general notion of interpretability. Having cheaper labor enables us to have a more substantial amount of subjects, thus getting more feedback. The tasks used in this kind of evaluation is often simplified to accommodate that the test subjects are not experts in the application domain. This type of evaluation can be evaluated using multiple-choice questions instead of open-ended questions that will be easier for the subjects to answer.

Functionally-grounded Evaluation does evaluation without human subjects using proxy tasks. We might want to test the system before it is done and mature enough to see the progress. Thus this type of evaluation provides a cheap way to assess the progress. Choosing a proxy task is no easy feat, but after it has been chosen, the evaluation mainly equates to an optimization problem. This type of evaluation is not as accurate as the application-grounded, nevertheless, it still provides value, and can be done automatically.

The evaluation of interpretability remains an open problem. However, knowing this is still important when we want to demonstrate new methods. This provides us with a more rigid framework for evaluating new interpretability methods and support our claims.

2.4 Explanation methods

In the sections to come, we are going to look at explanation methods that are used in this thesis. Most of the examples used to present these methods are images. However, these methods can be applied to other types of data. All of these methods provide local explanations and are tailored specifically towards NNs.

Before we continue, we will give a general description of how the methods work and how they are classified. These methods produce an explanation by highlighting "important" features on an input observation. More specifically, assume that we have a NN that takes in the input $\boldsymbol{x} = (x_1, ..., x_n) \in \mathbb{R}^n$ and produces the output $\boldsymbol{y} = (y_1, ..., y_c)$ where c is the number of output neurons. Given a specific target neuron y_j, the goal of an attention method is to determine the contribution $\boldsymbol{r}^j = (r_1^j, ..., r_n^j) \in \mathbb{R}^n$ of each input feature x_i to the output y_j. $\boldsymbol{r}^j$ is what we are going to call the attention map in this thesis, which has the same size as the input $\boldsymbol{x}$. For classification, we usually want to find the contribution of the inputs to a neuron associated with a class of interest.

If we were to classify the methods that fall under this description according to the taxonomy given in Section 2.3.4, these would be post-hoc interpretability. However, since they use the gradients of the models, they are not strictly model-agnostic. The explanations are valid only for single samples, the interpretability methods are considered local interpretations.

There is two main ways to obtain $\boldsymbol{r}^j$ using methods that are specific to a NN, perturbation-based [Zeiler and Fergus, 2014; Zintgraf et al., 2017; Zhou and Troyanskaya, 2015] or backpropagation-based [Bach et al., 2015; Simonyan et al., 2014; Springenberg et al., 2015; Shrikumar et al., 2019; Zeiler and Fergus, 2014; Sundararajan et al., 2017; Shrikumar et al., 2017].

Perturbation-based methods make changes to input in order to measure feature importance. They do it by systematically making changes to the input, do a forward pass for each modification, and measure the difference in the output for each change. These methods can estimate the importance of the area that has been masked, removed, or altered. The downsides of these methods are the computational resources needed for the forward passes for each modification, and the feature "importance" measured is very sensitive to how big the occluded area is [Ancona et al., 2019].

On the other hand, the backpropagation-based methods are much more efficient since only one pass is needed. The importance signal is backpropagated from the neuron of interest to the input space. It is similar to the gradient calculation during training. However, this calculation is applied after training to interpret

the model, not updating its parameters. Also, it is called backpropagation based methods since not every method in this category uses the gradient to obtain feature importance. Nevertheless, they all work in a backpropagation fashion.

2.4.1 Gradient-weighted Class Activation Mapping

Grad-CAM [Selvaraju et al., 2020a] is a method used to interpret CNNs that builds upon class activation mapping (CAM) method by Zhou et al. [2015a]. CAM is used to identify discriminative regions in an image for explaining the classification for a limited number of CNN architectures. In essence, CAM trades-off predictive accuracy for descriptive accuracy. Grad-CAM, on the other hand, generalizes CAM so it can by applied to any CNN architecture without any modifications to the model or requiring retraining. Thus avoiding the trade-off between the predictive and descriptive accuracy.

Grad-CAM assumes that deeper representations in a CNN capture higher-level visual concepts. Furthermore, it assumes that the feature maps retain spatial information. Both of these properties are important for Grad-CAM to obtain good results, and are discussed in Section 2.2.1. Grad-CAM is mainly used in the last convolution layer since it assumes that this layer trades-off best when it comes to high-level semantic and spatial information [Selvaraju et al., 2020a]. It uses the gradient in the last convolutional layer to explain a CNN's decision making. Grad-CAM explains the decision making by producing an attention map that the authors call a class-discriminative localization map $L^c_{\text{Grad-CAM}} \in \mathbb{R}^{U \times V}$ of width U and height V for any class c. An example of a class-discriminative localization map can be seen in Figure 2.16b.

Since the final attention map is a 2-dimensional matrix, and the last convolutional layer nearly always consists of more than 1 feature map, the feature maps must first be combined. Grad-CAM combines the feature maps by doing a linear combination of the feature maps. Therefore, the first step is to find the importance of each feature map k, denoted $\boldsymbol{A}^k$, for class c. It starts by computing the gradient of the score for class c, y_c (before softmax), with respect to the feature map activation $\boldsymbol{A}^k$ of a convolutional layer. That is, we want to find the gradients $\frac{\partial y_c}{\partial \boldsymbol{A}^k_{i,j}}$ for all i, j. The reason we do not want to involve the softmax function is due to that softmax normalizes with respect to all classes. The result is that the gradient of the probability after softmax contains not only the class of interest but also the other classes. However, we only want to know how the model reasons for a specific class. $\frac{\partial y_c}{\partial \boldsymbol{A}^k_{i,j}}$ are global-averaged-pooled (works best based on the empirical findings [Selvaraju et al., 2020a]) over all the gradients to find α^c_k, the importance of feature map k for class c. If we assume that the shape

of the feature maps is $U \times V$ with a size Z, the importance α_k^c is:

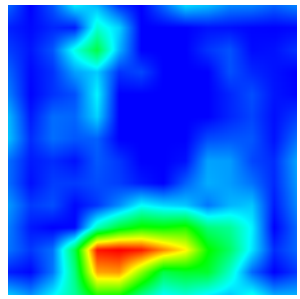
$$\alpha_k^c = \frac{1}{Z} \sum_i^U \sum_j^V \frac{\partial y_c}{\partial \boldsymbol{A}_{i,j}^k}.$$

The localization map for class c is obtained by taking ReLU of the weighted sum of the N feature maps:

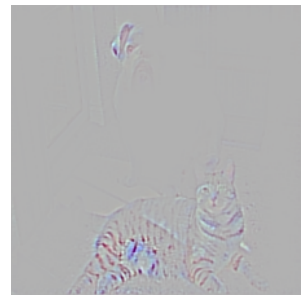
$$L_{\text{Grad-CAM}}^c = ReLU(\sum_k^K \alpha_k^c \boldsymbol{A}^k).$$

The shape of the localization map is the same as the shape of each feature map. The application of ReLU is not necessary, but this is how the original paper presents Grad-CAM. ReLU is used since the authors wanted only the area that positively influenced the classification of class c. In other words, the area that should be increased to increase the class score y_c. The authors hypothesized that the negative contributions belong to other classes, and by excluding them, we are only focusing on a single class. Note that the y_c does not need to be the class score of class c, it can be any differentiable activation.

(a) Image of a "boxer dog" and a "tiger cat"



(b) Grad-CAM localization map of "tiger cat"



(c) Guided Grad-CAM localization map of "tiger cat"

Figure 2.16: Results of Grad-CAM and Guided Grad-CAM

Source: Selvaraju et al. [2020a]

The main advantage of Grad-CAM is that its explanations are based on concepts that are easier to reason with. On the other hand, the size of the localization map can be too coarse based on the CNN architecture and the input size. In order to fix the size of the localization map issue, the authors proposed Guided

Grad-CAM. It is called Guided since originally in the paper, they used the backpropagation method called Guided Backpropagation [Springenberg et al., 2015]. Guided Backpropagation is a gradient-based method that combines the work of Simonyan et al. [2014] and Zeiler and Fergus [2014] together. Guided Grad-CAM is obtained by taking the attention map of any pixel-based backpropagation method (in the paper, they used Guided Backpropagation) and elementwise multiply it with an upsampled version of the localization map (the authors used bilinear interpolation). According to the authors, by doing this, they obtain an explanation that is both high-resolution and class-discriminative, such as the one seen in Figure 2.16c. In Figure 2.16b, we get an explanation that points at the cat (class-discriminative). On the other hand, Figure 2.16c points at the cat (class-discriminative) and justifies the decision by highlighting the stripes, pointy eyes, and ears of the cat (high-resolution).

2.4.2 Using explanations to improve the model

We have so far look at some methods to explain a NN by pointing out important input features that contribute to and highly affect the prediction of individual samples. In this section, we will look at some previous works that use the explanations to improve the model generalization and performance. There has been much work on the extraction of explanations from models. However, the amount of research on using the extracted explanation to improve the model has not been much. The number of research articles published on this topic is only a handful [Rieger et al., 2020; Mitsuhara et al., 2019; Burns et al., 2019; Wu et al., 2017; Ross et al., 2017; Erion et al., 2020] to the best of our knowledge.

Drucker and Cun [1992] showed that penalizing the input gradient can improve the model's generalization abilities. Their motivation for doing this is that small perturbations in the input space should not change the output components. By regularizing the input gradient, they make sure that small changes in the input will not significantly affect the output. Thus, making the network weights smaller and keeping the neurons' output longer in the linear region. Ross et al. [2017] suggest penalizing the input gradient to correct wrong explanations (adding domain knowledge). By correcting explanations, it will make the model generalize and perform better. Ross et al. [2017] do this by adding a penalty to the loss function that penalizes the input gradient at certain regions that the user specifies as unimportant or wrong explanation.

Let us now go into details of how Ross et al. [2017] does it. Assume that we have a binary matrix $\boldsymbol{A} \in \{0,1\}^{N \times D}$ where 1 indicates that the feature d is irrelevant for the prediction. Further, assume that we have our data $\boldsymbol{X}$ and the classification $\hat{\boldsymbol{Y}}$ that is obtained by forward passing $\boldsymbol{X}$. The goal is to minimize

$\frac{\partial \boldsymbol{X}_d}{\hat{\boldsymbol{Y}}}$ for ds that is irrelevant for the classification. Assume that we uses the CE loss function with N samples and C classes, thus by adding this new term, the loss function becomes:

$$\mathcal{L}(\boldsymbol{Y}, \hat{\boldsymbol{Y}}) = \underbrace{-\frac{1}{N}\sum_{i=1}^{N}\frac{1}{C}\sum_{c=1}^{C}\boldsymbol{Y}_c^{(i)}\log\hat{\boldsymbol{Y}}_c^{(i)}}_{\text{Right answer}} + \underbrace{\lambda_1\sum_j\theta_j^2}_{\text{Regularization}}$$
$$\underbrace{+\lambda_2\frac{1}{N}\sum_{i=1}^{N}\frac{1}{D}\sum_{d=1}^{D}(\boldsymbol{A}_d^{(i)}\frac{\partial}{\partial\boldsymbol{X}_d^{(i)}}\sum_{c=1}^{C}\log\hat{\boldsymbol{Y}}_c^{(i)})^2}_{\text{Right reasons}}.$$

What the "right reasons" term does is to discourage large input gradient in regions marked by $\boldsymbol{A}$. Also, the authors use the gradient of log probability because it works best based on empirical findings. In the original paper, the input gradient's suppression is over all classes as they assume that a feature d is irrelevant for all of the classes. However, the input gradient can be suppressed separately for different classes, giving a finer feedback matrix. The main problem, as we see it, is to specify the annotation matrix $\boldsymbol{A}$ beforehand. Also, it works on a pixel-level that can make the feedback process harder for the domain expert since annotating too many or too few pixels can, in the worst case, have a big impact.

In the same paper, the authors propose a method to automatically finding multiple plausible annotation matrices without human intervention, since it is not always possible, and the "right reason" might not be known beforehand. Let us illustrate the algorithm to get a better idea of how it works. By using Algorithm 2,

Algorithm 2 Find another explanation

1. Set the annotation matrix $\boldsymbol{A}_{(0)} = 0$.
2. For i in (0 to as long as the performance does not drop significantly and the annotation matrices change):
   (a) Train a model $\mathcal{M}_i$ using the annotation matrix $\boldsymbol{A}_{(i)}$
   (b) If the model performance is not impacted significantly negatively keep $\boldsymbol{A}_{(i)}$ as one explanation (except for $i = 0$).
   (c) Set $\boldsymbol{A}_{(i+1)} = 1$ where the input gradient $\boldsymbol{I} \in \mathbb{R}^{N\times D}$ of $\mathcal{M}_i$ is $\frac{\boldsymbol{I}}{\max_d \boldsymbol{I}} > c$ for some constant c.
3. Return $\boldsymbol{A}_{(1)}, ...$ as possible annotation matrices.

a domain expert can obtain annotation matrices it can choose from instead of trying to figure out by itself. All of the resulting models using the annotation matrices are accurate, but based on different reasons. The number of annotation matrices produced provides a measure of redundancy in the dataset.

Other related works are, one by Erion et al. [2020] that suggest enforcing priors (not as in Bayesian statistics) over the explanation in the model to make the model behave more intuitively. They propose three different priors, 1) on image data, encourage the model to produce piece-wise smooth attention maps, that its, neighboring pixels should have similar attributions, 2) on gene expression data, encourage the model to treat functionally related genes similarly, and 3) on health care data, encourage the model to produce sparse attribution maps. They mainly add a term to the loss functions that measure how well the model enforces the prior selected (assuming that there is a differentiable function that can measure this prior). They call their method for *attribution priors* and suggest this as a way to encode meaningful domain knowledge. Another is by Mitsuhara et al. [2019] that suggest having domain experts to edit attention maps extracted from the model that they think is wrong. After obtaining human-edited attention maps, they fine-tune the model to minimize the difference between the edited attention maps and the one extracted from the model. This is done by adding a penalty to the loss function that measures the difference between the attention maps. The biggest issue, as we see it is asking an expert giving the correct explanation from scratch. Doing that is difficult and limits the network's possibility of finding new interesting explanations that the experts might not have thought about.

2.5 Bayesian inference

Consider a probabilistic model $P(\mathcal{D}|\boldsymbol{w})$, the prior distribution $P(\boldsymbol{w})$ that encodes the prior knowledge about the parameter $\boldsymbol{w}$ and the evidence

$$\mathcal{D} = \{(\mathbf{X}^{(i)}, \boldsymbol{y}^{(i)})\}_{i=1}^{n} \tag{2.18}$$

with n i.i.d. samples. We want to compute the posterior distribution

$$P(\boldsymbol{w}|\mathcal{D}) = \frac{P(\mathcal{D}|\boldsymbol{w})P(\boldsymbol{w})}{P(\mathcal{D})} \tag{2.19}$$

using Bayes rule. $P(\mathcal{D}) = \int P(\mathcal{D}, \boldsymbol{\theta}) d\boldsymbol{\theta}$ is the marginal likelihood. To solve it, we need to marginalize over $\boldsymbol{\theta}$ which is typically intractable. This can be solved analytically, but only for simple models. For example, a model with discrete parameters $\boldsymbol{\theta}$ where the integral becomes a sum, the integral is simple enough, or when we have distributions from the exponential family. Even in some situations, when the integral is analytically solvable, the computational cost might be too high.

The posterior can be approximated several ways. One way is to frame the problem of finding the posterior as an optimization problem. To do this, we need to find

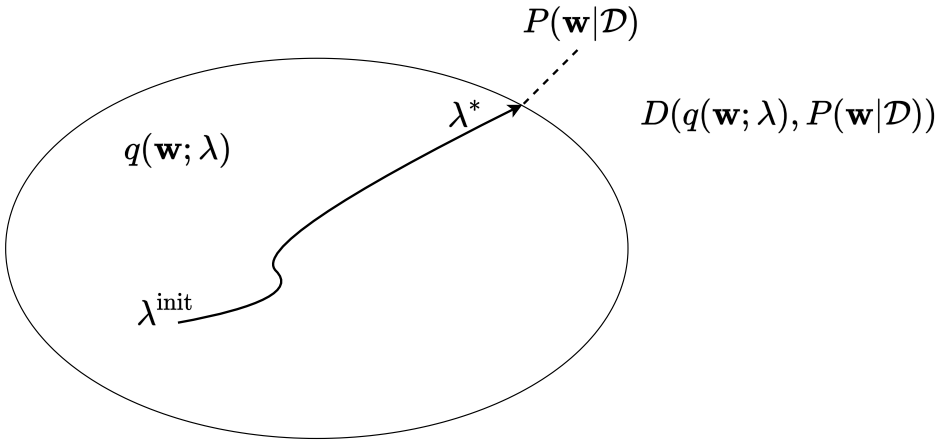


Figure 2.17: Minimizing the dissimilarity between $q(\boldsymbol{\theta};\boldsymbol{\lambda})$ and $P(\boldsymbol{\theta}|\mathcal{D})$ using some measure D.

a variational distribution $q(\boldsymbol{\theta};\boldsymbol{\lambda})$ parameterized by $\boldsymbol{\lambda}$ that is an approximation of the true posterior distribution $P(\boldsymbol{\theta}|\mathcal{D}) \approx q(\boldsymbol{\theta};\boldsymbol{\lambda})$. $q(\boldsymbol{\theta};\boldsymbol{\lambda})$ is selected to belong to a family of distributions of simpler form than the true posterior that makes the calculations tractable, and is as close to the true posterior as possible. The optimal λ is found by minimizing the dissimilarity between $P(\boldsymbol{\theta}|\mathcal{D})$ and $q(\boldsymbol{\theta};\boldsymbol{\lambda})$ with respect to some dissimilarity measure $D(q\|P)$. This minimization process is illustrated in Figure 2.17.

2.5.1 Kullback–Leibler divergence

The most common measure of dissimilarity between distributions for variational Bayes is the Kullback–Leibler (KL) divergence [Kullback and Leibler, 1951]. KL divergence is a measure that makes the minimization tractable. KL-divergence is defined as

$$D_{\mathrm{KL}}(P_1\|P_2) = \int_{\boldsymbol{\theta}} P_1(\boldsymbol{\theta}) \log \frac{P_1(\boldsymbol{\theta})}{P_2(\boldsymbol{\theta})} d\boldsymbol{\theta}, \tag{2.20}$$

that measures the dissimilarity between the distributions $P_1(\cdot)$ and $P_2(\cdot)$, and has the following properties

- $D_{\mathrm{KL}}(P_1\|P_2) \geq 0$.
- $D_{\mathrm{KL}}(P_1\|P_2) = 0 \iff \forall\boldsymbol{\theta} P_1(\boldsymbol{\theta}) = P_2(\boldsymbol{\theta})$.

- KL divergence is not symmetric: $D_{\mathrm{KL}}(P_1 \| P_2) \neq D_{\mathrm{KL}}(P_1 \| P_2)$.
- KL divergence does not satisfy the triangle inequality $D_{\mathrm{KL}}(P_1, P_2) \leqslant D_{\mathrm{KL}}(P_1 \| P_3) + D_{\mathrm{KL}}(P_3 \| P_2)$. For example, assume $\Omega = \{0, 1\}$, $P_1(0) = \frac{1}{2}$, $P_3(0) = \frac{1}{4}, P_2(0) = \frac{1}{10}$. We get that $D_{\mathrm{KL}}(P_1 \| P_2) \leqslant D_{\mathrm{KL}}(P_1 \| P_3) + D_{\mathrm{KL}}(P_3 \| P_2) \implies 0.51 \leqslant 0.24$ which is not true.

We can see that for $P_1(\boldsymbol{x}) = 0$, we have $P_1(\boldsymbol{x}) \log \frac{P_1(\boldsymbol{x})}{P_2(\boldsymbol{x})} = 0$, but for

$$\lim_{P_2(\boldsymbol{x}) \to 0} P_1(\boldsymbol{x}) \log \frac{P_1(\boldsymbol{x})}{P_2(\boldsymbol{x})} \to \infty, \text{ if } P_1(\boldsymbol{x}) > 0.$$

This implies that the support of $P_1(\cdot)$ has to be within the support of $P_2(\cdot)$, if not the divergence goes to ∞. In other less precise words, when choosing the approximate distribution $P_1(\cdot)$, we want $P_1(\cdot)$ to be (proportionally) small when $P_2(\cdot)$ is small. This also implies that $P_1(\cdot)$ minimizing the divergence to $P_2(\cdot)$ will in general have smaller variance.

2.5.2 Exponential family

The exponential family is a parametric set of probability density functions and mass functions that can be expressed as

$$f(\boldsymbol{x}|\boldsymbol{\theta}) = h(\boldsymbol{x}) c(\boldsymbol{\theta}) e^{\sum_{i=1}^{k} w_i(\boldsymbol{\theta}) t_i(\boldsymbol{x})}, \tag{2.21}$$

where $h(\boldsymbol{x}) \geq 0$, $t_1(\boldsymbol{x}), ..., t_k(\boldsymbol{x})$ are real-valued functions of the observation $\boldsymbol{x}$ (which cannot depend on $\boldsymbol{x}$), $c(\boldsymbol{\theta}) \geq 0$ and $w_1(\boldsymbol{\theta}), ..., w_k(\boldsymbol{\theta})$ are real-valued functions of parameter $\boldsymbol{\theta}$ (which cannot depend on $\boldsymbol{x}$).

The advantage of the exponential family is that it makes algebraic manipulation easier and identical for many distributions. Some continuous distributions that are part of the family are Normal, Gamma, and Beta. For discrete distributions, we have Binomial, Poisson, and Negative binomial.

Another important property with the exponential family representation is the conjugate distributions. If when the prior $P(\boldsymbol{\theta})$ is multiplied by the likelihood $P(\mathbf{x}|\boldsymbol{\theta})$ and divided by the evidence $P(\mathcal{D})$, the resulting posterior distribution $P(\boldsymbol{\theta}|\mathcal{D})$ is in the same family of distributions as the prior, then we have a conjugate prior. In general, if the posterior distribution and prior are in the same probability distribution family, they are called conjugate distributions, and the prior is called a conjugate prior for the likelihood.

2.5.3 Variational approximation

Now that we have some background, we can continue with the derivation of finding $\boldsymbol{\lambda}$ so that the divergence between the variational distribution $q(\boldsymbol{\theta};\boldsymbol{\lambda})$ and the true posterior $P(\boldsymbol{\theta}|\mathcal{D})$ is minimized $\min_{\boldsymbol{\lambda}} D_{\mathrm{KL}}(q(\boldsymbol{\theta};\boldsymbol{\lambda})\|P(\boldsymbol{\theta}|\mathcal{D}))$. First, we need to specify a family of densities over the latent variables such that $q(\boldsymbol{\theta};\boldsymbol{\lambda}) \in \mathcal{Q}$, where each $q(\boldsymbol{\theta};\boldsymbol{\lambda})$ is a candidate approximation to the exact conditional. It is important to notice that the order of distribution measured is important because KL divergence is not symmetric. Distribution $q(\boldsymbol{\theta};\boldsymbol{\lambda})$ is chosen as the first argument, since integration over $q(\boldsymbol{\theta};\boldsymbol{\lambda})$ is tractable (we choose the distribution). If the unknown posterior $P(\boldsymbol{\theta}|\mathcal{D})$ was chosen as the first argument. The derivation would be harder because it involves point-wise evaluation of $P(\boldsymbol{\theta}|\mathcal{D})$. This formulation gives us the objective

$$\mathcal{L}_{KL}(\boldsymbol{\lambda}) = \int q(\boldsymbol{\theta};\boldsymbol{\lambda}) \log \frac{q(\boldsymbol{\theta};\boldsymbol{\lambda})}{P(\boldsymbol{\theta}|\mathcal{D})} d\boldsymbol{\theta}, \tag{2.22}$$

that can not be computed directly, since it depends on $\log P(\boldsymbol{\theta}|\mathcal{D})$. The Equation (2.22) can be re-written as

$$\begin{aligned} \mathcal{L}_{KL}(\boldsymbol{\lambda}) &= \int q(\boldsymbol{\theta};\boldsymbol{\lambda}) \log \frac{q(\boldsymbol{\theta};\boldsymbol{\lambda})P(\mathcal{D})}{P(\boldsymbol{\theta}|\mathcal{D})P(\mathcal{D})} d\boldsymbol{\theta} \\ &= \int q(\boldsymbol{\theta};\boldsymbol{\lambda}) \log \frac{q(\boldsymbol{\theta};\boldsymbol{\lambda})P(\mathcal{D})}{p(\boldsymbol{\theta},\mathcal{D})} d\boldsymbol{\theta}, \end{aligned} \tag{2.23}$$

that is still hard to compute, yet easier than the original formulation since we no longer have to deal with the posterior distribution. Equation (2.22) can further be written as

$$\begin{aligned} \mathcal{L}_{KL}(\boldsymbol{\lambda}) &= \int q(\boldsymbol{\theta};\boldsymbol{\lambda}) \log \frac{q(\boldsymbol{\theta};\boldsymbol{\lambda})}{P(\boldsymbol{\theta},\mathcal{D})} d\boldsymbol{\theta} + \log P(\mathcal{D}) \\ &= D_{\mathrm{KL}}(q(\boldsymbol{\theta};\boldsymbol{\lambda})\|P(\boldsymbol{\theta},\mathcal{D})) + \log P(\mathcal{D}) \\ &= \mathbb{E}_{q(\boldsymbol{\theta};\boldsymbol{\lambda})}[q(\boldsymbol{\theta};\boldsymbol{\lambda})] - \mathbb{E}_{q(\boldsymbol{\theta};\boldsymbol{\lambda})}[\log P(\boldsymbol{\theta},\mathcal{D})] + \log P(\mathcal{D}) \\ &= \mathcal{H}(q(\boldsymbol{\theta};\boldsymbol{\lambda})) - \mathbb{E}_{q(\boldsymbol{\theta};\boldsymbol{\lambda})}[\log P(\boldsymbol{\theta},\mathcal{D})] + \log P(\mathcal{D}). \end{aligned} \tag{2.24}$$

The first term is called negative expected log-density, while the second is the negative entropy of the approximation. Even though we no longer have to deal with the posterior distribution, we still have evidence $\log P(\mathcal{D})$, which is hard to compute and is the reason why we appeal to approximate inference in the first place.

We, therefore, need to rewrite the objective, and it can be done as follows

$$\begin{aligned}
\log P(\mathcal{D}) &= \mathcal{L}_{KL}(\boldsymbol{\lambda}) - D_{\mathrm{KL}}(q(\boldsymbol{\theta};\boldsymbol{\lambda})\|P(\boldsymbol{\theta},\mathcal{D})) \\
&= \mathcal{L}_{KL}(\boldsymbol{\lambda}) - \int q(\boldsymbol{\theta};\boldsymbol{\lambda}) \log \frac{q(\boldsymbol{\theta};\boldsymbol{\lambda})}{P(\boldsymbol{\theta},\mathcal{D})} d\boldsymbol{\theta} \\
&= \mathcal{L}_{KL}(\boldsymbol{\lambda}) - \int q(\boldsymbol{\theta};\boldsymbol{\lambda}) \log \frac{q(\boldsymbol{\theta};\boldsymbol{\lambda})}{P(\mathcal{D}|\boldsymbol{\theta})P(\boldsymbol{\theta})} d\boldsymbol{\theta} \\
&= \mathcal{L}_{KL}(\boldsymbol{\lambda}) - \int q(\boldsymbol{\theta};\boldsymbol{\lambda}) \log \frac{q(\boldsymbol{\theta};\boldsymbol{\lambda})}{P(\boldsymbol{\theta})} d\boldsymbol{\theta} + \int q(\boldsymbol{\theta};\boldsymbol{\lambda}) \log P(\mathcal{D}|\boldsymbol{\theta}) d\boldsymbol{\theta} \\
\log p(\mathcal{D}) &= \mathbb{E}_{q(\boldsymbol{\theta};\boldsymbol{\lambda})}[\log P(\mathcal{D}|\boldsymbol{\theta})] - D_{\mathrm{KL}}(q(\boldsymbol{\theta};\boldsymbol{\lambda})\|p(\boldsymbol{\theta})) + \mathcal{L}_{KL}(\boldsymbol{\lambda}).
\end{aligned} \tag{2.25}$$

Now, the left side of Equation (2.25) consist of evidence $\log P(\mathcal{D})$ which is a constant, and $\mathcal{L}_{KL}(\boldsymbol{\lambda})$ which is always positive. We can see that maximizing $\mathbb{E}_{q(\boldsymbol{\theta};\boldsymbol{\lambda})}[\log P(\mathcal{D}|\boldsymbol{\theta})] - D_{KL}(q(\boldsymbol{\theta};\boldsymbol{\lambda})\|P(\boldsymbol{\theta}))$ is the same as minimizing $\mathcal{L}_{KL}(\boldsymbol{\lambda})$. We therefore get a new objective that is (to be maximized)

$$\mathcal{L}_{ELBO}(\boldsymbol{\lambda}) = \mathbb{E}_{q(\boldsymbol{\theta};\boldsymbol{\lambda})}[\log P(\mathcal{D}|\boldsymbol{\theta})] - D_{KL}(q(\boldsymbol{\theta};\boldsymbol{\lambda})\|P(\boldsymbol{\theta})). \tag{2.26}$$

The first term in Equation (2.26) is the expected log-likelihood, and the second term is the divergence between the approximation and the prior. We see that the first term encourages $q(\boldsymbol{\theta};\boldsymbol{\lambda})$ to place its mass on $\boldsymbol{\theta}$ that explains the observed data. While the second term $q(\boldsymbol{\theta};\boldsymbol{\lambda})$ wants to place mass on densities close to the prior. Hence, the objective trades between likelihood and prior. The data amount weights the trade-off. Thus, having more data places more mass on the likelihood term. We can see that $\log P(\mathcal{D}) \geq \mathcal{L}_{ELBO}(\boldsymbol{\lambda})$ for any $q(\boldsymbol{\theta};\boldsymbol{\lambda})$, because $D_{\mathrm{KL}}(q(\boldsymbol{\theta};\boldsymbol{\lambda})\|P(\boldsymbol{\theta})) \geq 0$ shown by Kullback and Leibler [1951].

The new objective is called evidence lower bound (ELBO) and contains only expectations of known parts of the model over the approximation. However to be able to compute the expectations, we need to assume that $q(\boldsymbol{\theta};\boldsymbol{\lambda})$ is simple enough. This new objective tries to balance a trade-off between satisfying the simplicity of the prior $P(\boldsymbol{\theta})$, and the complexity of the data $\mathcal{D}$ as mentioned. Hence, the maximizing variational distribution is

$$q^*(\boldsymbol{\theta};\boldsymbol{\lambda}) = \arg\max_{q^* \in \mathcal{Q}} \mathcal{L}_{ELBO}. \tag{2.27}$$

2.5.4 Mean-field approximation

In the section above, we mentioned that we need to assume that $q(\boldsymbol{\theta};\boldsymbol{\lambda})$ is simple enough. One way is to assume that $q(\boldsymbol{\theta};\boldsymbol{\lambda})$ can be factorized into a product

of lower-dimensional terms, which is called the mean-field approximation. The approximated posterior can then be written as

$$q(\boldsymbol{\theta};\boldsymbol{\lambda}) = \prod_{i=1}^{N} q(\theta_i;\boldsymbol{\lambda}_i), \tag{2.28}$$

where we assume that the dimension is N and that each λ_i can be a vector.

Assuming mean-field approximation the first term $\mathbb{E}_{q(\boldsymbol{\theta};\boldsymbol{\lambda})}[\log P(\mathcal{D}|\boldsymbol{\theta})]$ in Equation (2.26) can be expressed as

$$\begin{aligned}
\mathbb{E}_{q(\boldsymbol{\theta};\boldsymbol{\lambda})}[\log P(\mathcal{D}|\boldsymbol{\theta})] &= \int q(\boldsymbol{\theta};\boldsymbol{\lambda}) P(\mathcal{D}|\boldsymbol{\theta}) d\boldsymbol{\theta} \\
&= \int \prod_{i=1}^{N} q(\theta_i;\boldsymbol{\lambda}_i) \log P(\mathcal{D}|\theta_i) d\boldsymbol{\theta} \\
&= \int \ ... \ \int \prod_{i=1}^{N} q(\theta_i;\boldsymbol{\lambda}_i) \log P(\mathcal{D}|\theta_i) d\theta_1 \ .... \ d\theta_N \\
&= \int q(\theta_n;\boldsymbol{\lambda}_n) \mathbb{E}_{q_{-n}}[\log P(\mathcal{D}|\boldsymbol{\theta})] d\theta_n.
\end{aligned}$$

While the second term $D_{KL}(q(\boldsymbol{\theta};\boldsymbol{\lambda})\|P(\boldsymbol{\theta}))$ can written as

$$\begin{aligned}
D_{KL}(q(\boldsymbol{\theta};\boldsymbol{\lambda})\|P(\boldsymbol{\theta})) &= \int q(\boldsymbol{\theta};\boldsymbol{\lambda}) \log \frac{q(\boldsymbol{\theta};\boldsymbol{\lambda})}{P(\boldsymbol{\theta})} d\boldsymbol{\theta} \\
&= \int \prod_{i=1}^{N} q(\theta_i;\boldsymbol{\lambda}_i) \log \frac{q(\theta_i;\boldsymbol{\lambda}_i)}{P(\theta_i)} d\boldsymbol{\theta} \\
&= \int \ ... \ \int \prod_{i=1}^{N} q(\theta_i;\boldsymbol{\lambda}_i) \log \frac{q(\theta_i;\boldsymbol{\lambda}_i)}{P(\theta_i)} d\theta_1 \ .... \ d\theta_N \\
&= \int q(\theta_n;\boldsymbol{\lambda}_n) \mathbb{E}_{q_{-n}}[\log \frac{q(\boldsymbol{\theta};\boldsymbol{\lambda})}{P(\boldsymbol{\theta})}] d\theta_n.
\end{aligned}$$

Knowing this, we can then rewrite the equation Equation (2.26) as

$$\begin{aligned}
\mathcal{L}_{ELBO}(\boldsymbol{\lambda}) &= \int q(\theta_n;\boldsymbol{\lambda}_n)\mathbb{E}_{q_{-n}}[\log P(\mathcal{D}|\boldsymbol{\theta})]d\theta_n - \int q(\theta_n;\boldsymbol{\lambda}_n)\mathbb{E}_{q_{-n}}[\log \frac{q(\boldsymbol{\theta};\boldsymbol{\lambda})}{P(\boldsymbol{\theta})}]d\theta_n \\
&= \int q(\theta_n;\boldsymbol{\lambda}_n)\mathbb{E}_{q_{-n}}[\log \frac{P(\boldsymbol{\theta},\mathcal{D})}{q(\boldsymbol{\theta};\boldsymbol{\lambda})}]d\theta_n \\
&= \int q(\theta_n;\boldsymbol{\lambda}_n)\mathbb{E}_{q_{-n}}[\log P(\boldsymbol{\theta},\mathcal{D})]d\theta_n - \int q(\theta_n;\boldsymbol{\lambda}_n)\mathbb{E}_{q_{-n}}[\log q(\boldsymbol{\theta};\boldsymbol{\lambda})]d\theta_n \\
&= \int q(\theta_n;\boldsymbol{\lambda}_n)\mathbb{E}_{q_{-n}}[\log P(\boldsymbol{\theta},\mathcal{D})]d\theta_n - \int q(\theta_n;\boldsymbol{\lambda}_n)\log q(\boldsymbol{\theta}_n;\boldsymbol{\lambda}_n)d\theta_n \\
&\quad - \mathbb{E}_{q_{-n}}[\log q(\boldsymbol{\theta}_{-n};\boldsymbol{\lambda}_{-n})]
\end{aligned} \tag{2.29}$$

Lagrange multipliers

Lagrange Multipliers is used to solve constrained extreme-value problem where the variables of the function to be optimized must satisfy one or more constraint equations or inequalities. A technique for optimizing a function $f(x, y)$ subject to the equality constraint $g(x, y) = 0$ is based on the following theorem from Adams and Essex [2013]:

Theorem 2.2.1 (The Methods of Lagrange Multipliers)**.** *Suppose the f and g have continuous first partial derivatives near the point $P_0 = (x_0, y_0)$ on the curve $\mathcal{C}$ with equation $g(x, y) = 0$. Suppose also that, when restricted to points on $\mathcal{C}$, the function $f(x, y)$ has a local maximum or minimum value at P_0. Finally, suppose that*

- *P_0 is not an endpoint of $\mathcal{C}$, and*
- *$\nabla g(P_0) \neq 0$.*

Then there exists a number Λ_0 such that (x_0, y_0, Λ_0) is a critical point of the ***Lagrange function****.*

$$L(x, y, \Lambda) = f(x, y) + \Lambda g(x, y)$$

The theorem doesn't guarantee that a solution exists, but if it does exist, solving the Lagrange function $\nabla L(x, y, \Lambda) = f(x, y) + \Lambda g(x, y) = 0$ will give us maximum or minimum of $f(x, y)$ constrained to $g(x, y) = 0$. The intuition is based on that $f(x, y)$ will be minimized or maximized when it is perpendicular

to the constraint $g(x, y) = 0$, which implies that the $\nabla g(x, y)$ is proportional to $\nabla f(x, y)$ at that moment when they are perpendicular.

2.5.5 Coordinate ascent

Coordinate ascent is a optimization algorithm used to solve extreme-value problem for a multivariable function $f(\boldsymbol{x})$ where $\boldsymbol{x} \in \mathbb{R}^N$. The idea is that the multivariable function can be minimized by minimizing it along one direction x_i at a time, keeping all of the other variables $\boldsymbol{x}_{-i}$ fixed. This is done for all dimensions, N. Solving univariable optimization problems in a loop is easier, but it only converges to a local optimum for non-convex objectives.

Having Equation (2.29), we can use Lagrange multipliers and coordinate ascent to optimize Equation (2.29) for each $q(\theta_n; \boldsymbol{\lambda}_n)$ separately. We need Lagrange multipliers because $q(\theta_n; \boldsymbol{\lambda}_n)$ is a probability distribution, hence $\int q(\theta_n; \boldsymbol{\lambda}_n) d\theta_n = 1$ must be satisfied. Setting the constraint $\int q(\theta_n; \boldsymbol{\lambda}_n) d\theta_n = 1$, the Lagrangian of the optimization problem for a term n becomes

$$\begin{aligned} L_{ELBO}(n) = &\int q(\theta_n; \boldsymbol{\lambda}_n) \mathbb{E}_{q_{-n}}[\log P(\boldsymbol{\theta}, \mathcal{D})] d\theta_n - \int q(\theta_n; \boldsymbol{\lambda}_n) \log q(\boldsymbol{\theta}_n; \boldsymbol{\lambda}_n) d\theta_n \\ &+ \Lambda(\int q(\theta_n; \boldsymbol{\lambda}_n) d\theta_n - 1). \end{aligned} \tag{2.30}$$

Equation (2.30) can be optimized by solving $\nabla_{q(\theta_n;\boldsymbol{\lambda}_n),\Lambda} L_{ELBO}(n) = 0$. By doing that, we get

$$q(\theta_n; \boldsymbol{\lambda}_n) \propto e^{\mathbb{E}_{q_{-n}}[\log P(\boldsymbol{\theta}, \mathcal{D})]}. \tag{2.31}$$

Having everything that underlies coordinate ascent variational inference (CAVI). We can now outline the CAVI algorithm where we iterate through the variational factors $q(\theta_n; \boldsymbol{\lambda}_n)$, and updating them using equation Equation (2.31).

Algorithm 3 Coordinate ascent variational inference

- While the $\mathcal{L}_{ELBO}(\boldsymbol{\lambda})$ has not converged
  - For $i \in \{1, ..., n\}$
    - Set $q_i(\theta_i; \boldsymbol{\lambda}_i) \propto e^{\mathbb{E}_{q_{-n}}[\log P(\boldsymbol{\theta}, \mathcal{D})]}$
    - Compute $\mathcal{L}_{ELBO}(\boldsymbol{\lambda})$
- Return $q(\boldsymbol{\theta}; \boldsymbol{\lambda}) = \prod_{i=1}^{N} q_i(\theta_i; \boldsymbol{\lambda}_i)$

2.5.6 Bayes by Backprop

Bayes by Backprop (BBB) [Blundell et al., 2015] introduces a backpropagation-compatible algorithm for learning posterior distribution on the weights of a NN using variational inference. The goal is to find an approximate posterior distribution $q(\mathbf{w};\boldsymbol{\theta})$ parameterized $\boldsymbol{\theta}$ close to the true posterior $P(\mathbf{w}|\mathcal{D})$ since exact inference is intractable. The first step is to choose the form of the approximate posterior, expressive enough to be close to the true posterior. Next, we find $\boldsymbol{\theta}$ that minimizes the KL-divergence between the approximate and true posterior

$$
\begin{aligned}
\boldsymbol{\theta}^* &= \arg\min_{\boldsymbol{\theta}} D_{\mathrm{KL}}(q(\mathbf{w};\boldsymbol{\theta}) \| P(\mathbf{w}|\mathcal{D})) \\
&= \arg\min_{\boldsymbol{\theta}} \int q(\mathbf{w};\boldsymbol{\theta}) \log \frac{q(\mathbf{w};\boldsymbol{\theta})}{P(\mathbf{w}|\mathcal{D})} d\mathbf{w} \\
&= \arg\min_{\boldsymbol{\theta}} \int q(\mathbf{w};\boldsymbol{\theta}) \log \frac{q(\mathbf{w};\boldsymbol{\theta}) P(\mathcal{D})}{P(\mathcal{D}|\mathbf{w}) P(\mathbf{w})} d\mathbf{w} \\
&\quad (P(\mathcal{D}) \text{ can be omitted since it is constant and independent from } \boldsymbol{\theta}) \\
&= \arg\min_{\boldsymbol{\theta}} \int q(\mathbf{w};\boldsymbol{\theta}) \log \frac{q(\mathbf{w};\boldsymbol{\theta})}{P(\mathbf{w})} d\mathbf{w} - \int q(\mathbf{w};\boldsymbol{\theta}) \log P(\mathcal{D}|\mathbf{w}) d\mathbf{w} \\
&= \arg\min_{\boldsymbol{\theta}} D_{\mathrm{KL}}(q(\mathbf{w};\boldsymbol{\theta}) \| P(\mathbf{w})) - \mathbb{E}_{\mathbf{w}\sim q(\mathbf{w};\boldsymbol{\theta})}[\log P(\mathcal{D}|\mathbf{w})].
\end{aligned}
$$

This gives us the ELBO that we write as

$$
\begin{aligned}
\mathcal{F}(\mathcal{D},\boldsymbol{\theta}) &= \underbrace{D_{\mathrm{KL}}(q(\mathbf{w};\boldsymbol{\theta}) \| P(\mathbf{w}))}_{\text{Complexity}} - \underbrace{\mathbb{E}_{\mathbf{w}\sim q(\mathbf{w};\boldsymbol{\theta})}[\log P(\mathcal{D}|\mathbf{w})]}_{\text{Likelihood}} \\
&= \int q(\mathbf{w};\boldsymbol{\theta}) \log q(\mathbf{w};\boldsymbol{\theta}) d\mathbf{w} - \int q(\mathbf{w};\boldsymbol{\theta}) \log P(\mathbf{w}) d\mathbf{w} - \mathbb{E}_{\mathbf{w}\sim q(\mathbf{w};\boldsymbol{\theta})}[\log P(\mathcal{D}|\mathbf{w})] \\
&= \mathbb{E}_{\mathbf{w}\sim q(\mathbf{w};\boldsymbol{\theta})}[\log q(\mathbf{w};\boldsymbol{\theta}) - P(\mathbf{w}) - \log P(\mathcal{D}|\mathbf{w})].
\end{aligned}
\tag{2.32}
$$

Equation (2.32), on one hand, tries to satisfy the simplicity of the prior termed the complexity cost. On the other hand, the likelihood cost that tries to satisfy the complexity of the data $\mathcal{D}$. To minimize Equation (2.32), gradient descent and various approximations are used, since it is computationally infeasible to optimize it naively. If gradient descent is used to minimize Equation (2.32), it must be possible to differentiate Equation (2.32). To differentiate Equation (2.32) the reparameterization trick is used, which essentially expresses the derivative of an expectation as the expectation of a derivative.

The reparameterization trick

Consider a function f with derivative in $\mathbf{w}$ and that we want to compute

$$\begin{aligned}\nabla_{\boldsymbol{\theta}}\mathbb{E}_{\mathbf{w}\sim q(\mathbf{w};\boldsymbol{\theta})}[f(\mathbf{w},\boldsymbol{\theta})] &= \nabla_{\boldsymbol{\theta}}\int q(\mathbf{w};\boldsymbol{\theta})f(\mathbf{w},\boldsymbol{\theta})d\mathbf{w} \\ &= \int \nabla_{\boldsymbol{\theta}} q(\mathbf{w};\boldsymbol{\theta})f(\mathbf{w},\boldsymbol{\theta})d\mathbf{w} \\ &= \int f(\mathbf{w},\boldsymbol{\theta})\nabla_{\boldsymbol{\theta}} q(\mathbf{w};\boldsymbol{\theta})d\mathbf{w} + \int q(\mathbf{w};\boldsymbol{\theta})\nabla_{\boldsymbol{\theta}} f(\mathbf{w},\boldsymbol{\theta})d\mathbf{w} \\ &= \underbrace{\int f(\mathbf{w},\boldsymbol{\theta})\nabla_{\boldsymbol{\theta}} q(\mathbf{w};\boldsymbol{\theta})d\mathbf{w}}_{\text{Can not be computed}} + \mathbb{E}_{\mathbf{w}\sim q(\mathbf{w};\boldsymbol{\theta})}[\nabla_{\boldsymbol{\theta}} f(\mathbf{w},\boldsymbol{\theta})].\end{aligned}$$

However, we can not solve the integral analytically in general. To solve this, we rewrite the random vector $\mathbf{w}$ as $\mathbf{w} = g(\boldsymbol{\epsilon},\boldsymbol{\theta})$, where $\boldsymbol{\epsilon}\sim q(\boldsymbol{\epsilon})$ and $g(\boldsymbol{\epsilon},\boldsymbol{\theta})$ is a deterministic function. This gives us

$$\begin{aligned}\nabla_{\boldsymbol{\theta}}\mathbb{E}_{\mathbf{w}\sim q(\mathbf{w};\boldsymbol{\theta})}[f(\mathbf{w},\boldsymbol{\theta})] &= \nabla_{\boldsymbol{\theta}}\mathbb{E}_{\boldsymbol{\epsilon}\sim q(\boldsymbol{\epsilon})}[f(g(\boldsymbol{\epsilon};\boldsymbol{\theta}),\boldsymbol{\theta})] \\ &= \int q(\boldsymbol{\epsilon})\nabla_{\boldsymbol{\theta}} f(g(\boldsymbol{\epsilon};\boldsymbol{\theta}),\boldsymbol{\theta})d\mathbf{w} \\ &= \mathbb{E}_{\boldsymbol{\epsilon}\sim q(\boldsymbol{\epsilon})}[\nabla_{\boldsymbol{\theta}} f(g(\boldsymbol{\epsilon};\boldsymbol{\theta}),\boldsymbol{\theta})] \\ &= \mathbb{E}_{\boldsymbol{\epsilon}\sim q(\boldsymbol{\epsilon})}[\frac{\partial f(g(\boldsymbol{\epsilon};\boldsymbol{\theta}),\boldsymbol{\theta})}{\partial g(\boldsymbol{\epsilon};\boldsymbol{\theta})}\frac{\partial g(\boldsymbol{\epsilon};\boldsymbol{\theta})}{\partial\boldsymbol{\theta}} + \frac{\partial f(g(\boldsymbol{\epsilon};\boldsymbol{\theta}),\boldsymbol{\theta})}{\partial\boldsymbol{\theta}}],\end{aligned}$$

which is something we can solve.

To calculate Equation (2.32), Blundell et al. [2015] approximate it by drawing Monte Carlo samples from the variational posterior $q(\mathbf{w};\boldsymbol{\theta})$ and using the reparameterization trick on $\mathbf{w}$. Thus, we get

$$\begin{aligned}\mathbf{w} &= g(\boldsymbol{\epsilon},\boldsymbol{\theta}) \\ \mathcal{F}(\mathcal{D},\boldsymbol{\theta}) &\approx \frac{1}{n}\sum_{i=1}^{n}\log q(\mathbf{w}) - P(\mathbf{w}) - \log P(\mathcal{D}|\mathbf{w}).\end{aligned} \tag{2.33}$$

2.5.7 Local Reparameterization Trick

Equation (2.33) provides an objective that can be optimized using gradient descent. However, to increase sampling performance, Kingma et al. [2015] propose to sample an intermediate variable $g(\cdot)$ rather than the weights $\boldsymbol{w}$ that they coined the local reparameterization trick (LRT). By sampling an intermediate

variable with lower dimension, less random noise needs to be sample. It is called local since the reparameterization is applied on a intermediate variable instead of directly on the weights.

Let us now consider a simplified problem where the intermediate variable is the activation. For simplicity, we are only going to consider a single layer l and ignore the bias term. $\boldsymbol{b}$ is the output of the previous layer $l-1$ of size 1×100 which is fixed. $\mathbf{W}$ is the weights of layer l of size 100×100. Thus, the activation for layer l is $\boldsymbol{a} = \boldsymbol{b}\mathbf{W}$ of size 1×100. Let $q(\mathrm{w}_{i,j}; \boldsymbol{\theta}) = \mathcal{N}(\mu_{i,j}, \sigma_{i,j})$, which implies that $q(\mathrm{a}_j|\boldsymbol{b}) = \mathcal{N}(\hat{\mu}_j, \hat{\sigma}_j)$ where

$$\begin{aligned}\hat{\mu}_j &= \sum_i b_i \mu_{i,j} \\ \hat{\sigma}_j &= \sqrt{\sum_{i=1} b_i^2 \sigma_{i,j}^2}.\end{aligned}$$

Hence, if we sample activation rather than weight, Equation (2.33) can be rewritten as

$$\begin{aligned}\mathcal{F}(\mathcal{D}, \boldsymbol{\theta}) &= D_{\mathrm{KL}}(q(\mathbf{w}|\boldsymbol{\theta}) \| P(\mathbf{w})) - \mathbb{E}_{\mathbf{a} \sim q(\mathbf{a}|\mathbf{b})}[\log P(\mathcal{D}|\mathbf{a})] \\ &\approx D_{\mathrm{KL}}(q(\mathbf{w}; \boldsymbol{\theta}) \| P(\mathbf{w})) - \sum_{i=1}^{n} \log P(\mathcal{D}|\mathbf{a}^{(i)}) \\ &\approx \underbrace{D_{\mathrm{KL}}(q(\mathbf{w}; \boldsymbol{\theta}) \| P(\mathbf{w}))}_{\text{Complexity cost}} - \sum_{i=1}^{n} \log P(y|\boldsymbol{x}, \boldsymbol{w}, \boldsymbol{a}^{(i)}).\end{aligned} \tag{2.34}$$

The complexity cost in Equation (2.34) needs to be solved analytically since we can no longer draw Monte Carlo (MC) samples from $q(\mathrm{w}_{i,j}; \boldsymbol{\theta})$ to estimate the KL divergence. This restricts us to only Gaussian prior opposed to Equation (2.33) that allows for non-Gaussian prior and non-Gaussian approximate posterior. The derivation of the complexity cost for the Gaussian case can be found in the appendix of Kingma and Welling [2014].

Chapter 3

Frameworks

This chapter provides an overview of the frameworks used to make the research possible and used in the experiments.

3.1 Frameworks

Making DL models is not easy. However, it has never been easier than now due to various DL frameworks. The DL libraries often rely on graphics processing unit (GPU)-accelerated libraries to offer high-performance multi-GPU accelerated training. They all have the built-in features needed to make DL models, thus making DL available to the masses. This section gives an overview of the tools and frameworks used in implementing the models, which not only consists of DL frameworks.

Before moving on, we will give a high-level overview of DL frameworks. There are various DL frameworks, and they are implemented differently. However, certain traits are common for all of them. We will describe those properties before moving on to a more detailed look at a specific one used in this thesis.

AD [Rall, 1981]. AD is a set of ways or techniques to evaluate the derivative of a function that is essential for backpropagation. We will go more into detail on how AD works in Section 3.1.1.

Hardware integration. DL frameworks provide automatic utilization of available hardware to train NNs. The hardware for each user differs. It can be hard to set up, especially if the user, for example, has more than one GPU.

However, most DL frameworks offer automatic setup, which has made DL research much easier to do.

DL building blocks. Some components like regularizers and optimizers are standard for all DL models. Offering these components out of the box makes it easier for the user and less time needed to implement models.

Computation graphs. NNs can be represented as plain tensors. Nevertheless, DL frameworks offer to represent them as computation graphs. It makes them more manageable to look at and enables AD since backpropagation is implemented using computation graphs.

3.1.1 PyTorch

PyTorch [Paszke et al., 2019] is a scientific computing package targeted at two groups of audiences, those who want to use NumPy [Walt et al., 2011] (which adds array and matrix along with a large collection of high-level mathematical functions support for Python) on GPU and those who want to do deep learning research. PyTorch is essentially NumPy on GPU plus AD, and also allows converting from and to NumPy arrays seamlessly. However, this simple definition of PyTorch does not justify how vast and complex the technology is. It includes features such as different activation functions, optimizer, and early stopping schemes that make DL convenient.

Automatic Differentiation

Before continuing, let us take a detour and look at AD. What lies at the core of AD is the chain rule that allows the decomposition of differentials. AD can be done in two ways, forward accumulation or reverse accumulation, which specifies how the chain rule should be traversed. Consider the following example:

$$
\begin{aligned}
y = z(f(g(h(x)))) = z(f(g(h(w_o)))) = z(f(g(w_1))) = z(f(w_2)) = z(w_3) \\
w_0 &= x \\
w_1 &= h(w_0) \\
w_2 &= g(w_1) \\
w_3 &= f(w_2) \\
w_4 &= z(w_3) = y
\end{aligned}
\tag{3.1}
$$

We can calculate $\frac{dy}{dx}$ in Equation (3.1) by applying the chain rule that gives:

$$\begin{aligned}\frac{dy}{dx} &= \frac{dw_4}{dw_3}\frac{dw_3}{dw_2}\frac{dw_2}{dw_1}\frac{dw_1}{dw_0} \\ &= \frac{dy}{dw_3}\frac{dw_3}{dw_2}\frac{dw_2}{dw_1}\frac{dw_1}{x}.\end{aligned} \tag{3.2}$$

Forward accumulation would have solved Equation (3.2) in the following way:

$$\frac{dw_i}{dw_0} = \frac{dw_i}{dw_{i-1}}\frac{dw_{i-1}}{dw_0} \text{ for } i = 2, 3, 4. \tag{3.3}$$

While reverse accumulation would have solved Equation (3.2) in the following way:

$$\frac{dw_4}{dw_i} = \frac{dw_4}{dw_{i+1}}\frac{dw_{i+1}}{dw_i} \text{ for } i = 0, 1, 2. \tag{3.4}$$

Using Equation (3.3), the solution to Equation (3.1) is as follows:

$$\frac{dy}{dx} = \frac{dw_4}{dw_0} = \frac{dw_4}{dw_0}(\frac{dw_3}{dw_2}(\frac{dw_2}{dw_1}\frac{dw_1}{dw_0})). \tag{3.5}$$

If Equation (3.4) is used instead to solve Equation (3.1), the solution would be as follows:

$$\frac{dy}{dx} = \frac{dw_4}{dw_0} = ((\frac{dw_4}{dw_3}\frac{dw_3}{dw_2})\frac{dw_2}{dw_1})\frac{dw_1}{dw_0}. \tag{3.6}$$

AD is used to do backpropagation in a NN, described in Section 2.1.3.

PyTorch does the AD with reverse accumulation which makes it possible to change the way the network behaves arbitrarily with zero lag or overhead.

Components

PyTorch consists of several packages that make up the DL framework. We will take a look at three of the most important packages of PyTorch used to make a NN.

`torch.Tensor` represents tensors in PyTorch that can store, for example, weights and biases, and is similar to `numpy.ndarray`. `torch.Tensor` has the attribute `.requires_grad` that can be set to `True` for tensors requiring gradients. The method `.backward()` can be invoked to have all the gradients computed automatically. `torch.Tensor` along with `torch.autograd.Function` (keeps track of how tensors are computed) build up the computational graph in PyTorch. the `torch.autograd.Function` object is stored in a tensor's `.grad_fn` attribute.

`torch.autograd` provides AD for all differentiable operations on tensors, and is central to all NNs made using PyTorch. `torch.autograd.Function` mentioned earlier is part of `autograd` that provides methods like `.backward()` in addition to keeping track of operations.

`torch.nn` is the package that contains everything needed to construct a NN, such as activation functions and regularizers like Dropout. One of the most essential sub-module of `torch.nn` is `nn.Module` that contains layers, and a method `forward(method)` that returns the `output`. `nn.Module` represents a NN in PyTorch, and can contain `nn.Module` inside of itself making it easier for practitioners to organize their NNs into multiple modules. Hence, `nn.Module` is a way to organize low-level building blocks such as tensors into a NN.

Besides these packages, PyTorch offer, for example, `torch.optim.Optimizer` that contains many different optimizers such as Adam [Kingma and Ba, 2017] that can be used to train a NN.

Pretrained models

PyTorch offers pre-trained models that can be handy since the computational power needed to train large models are not always available, or the user might want to do transfer learning. Having pre-trained models can be handy, and it also provides a mean to see how the state-of-the-art models performs on our tasks.

3.1.2 PyTorch Lightning

PyTorch Lightning [Falcon and .al, 2019] is a way to organize PyTorch code, making it faster and easier to develop and train models. According to the developers, it can be thought of as a style guide for PyTorch code. It decouples the science code from the engineering code, which means that PyTorch Lightning takes care of the training code while the user can focus on their models and algorithms. Lightning removes all the boilerplate code making it easier to develop new algorithms and models.

PyTorch Lightning comes with additional facilities that make it easier to do ML research and keeping the experiments reproducible. Also, it interoperates with other frameworks made for PyTorch. PyTorch Lightning is created for researchers and Ph.D. students working in AI. The interface is simple, thus suitable for model developers and newcomers.

3.1.3 Captum

Captum [Kokhlikyan et al., 2020] ("comprehension" in Latin) is a framework built on PyTorch that consists of various interpretability methods for models made with PyTorch. It contains state-of-the-art interpretation methods that make it easier for researchers and developers to understand their models better. For ML researchers, it will be more accessible to benchmark and use different explanation methods in their research. While for model developers, it will be easier to troubleshoot and improve their models. Captum also comes with visualization tools, making it easier to understand the explanations since even with tools like Captum, explanations can be hard to understand without proper visualization. The visualization tools work for both images, text, and other types of features.

The intended audience for Captum is model developers that try to improve their models and researchers that are working on XAI. In addition, it can be used by people who use trained models that want to deliver better explanations to their users.

The algorithms Captum implements are separated into three groups, primary attribution, layer attribution, and neuron attribution, and are increasing continuously. They define these three different groups as:

Primary attribution evaluates contribution of each input feature to the output of a model.

Layer attribution evaluates contribution of each neuron in a given layer to the output of the model.

Neuron attribution evaluates contribution of each input feature on the activation of a particular hidden neuron.

3.1.4 TensorBoard

TensorBoard is TensorFlow's [Abadi et al., 2016] visualization toolkit that consists of web applications that provides tracking and visualization of ML experimentation. TensorBoard can track metrics like loss and accuracy, visualize the model graph, view histograms of weights, biases, or other tensors as they change over time. TensorBoard is a part of the TensorFlow ecosystem and can be used for PyTorch. Since TensorBoard runs like a website, it can be deployed as a website and be shared with others.

Chapter 4

Active Feedback

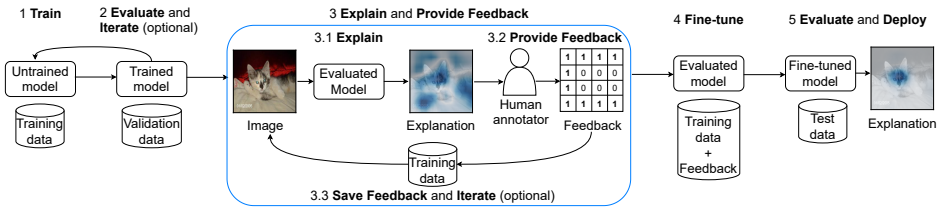


Figure 4.1: A "standard" ML pipeline with steps for annotating explanations and correcting a model. During Step 3, a model explains training sample classifications to a human annotator who gives feedback on those explanations. A feedback $\boldsymbol{F}^{(i)}$ for a sample i is a matrix of the same width and height as the sample. If a feature k, j is irrelevant $\boldsymbol{F}^{(i)}_{k,j} = 1$, otherwise $\boldsymbol{F}^{(i)}_{k,j} = 0$. In Step 4, a model is fine-tuned with training data and feedback. The goal is to improve the reasons behind the classifications (explanation in Step 3 vs. Step 5) and predictive performance.

During the 20th century, there was a horse named Clever Hans was claimed to have performed arithmetic and other intellectual tasks. It was later revealed that the horse observed its trainer and responded to the trainer's involuntary cues in the body language. The trainer was not aware that he provided the horse with hints causing it to display intelligent behavior. This behavior is today referred to as the Clever Hans effect [Pfungst, 1911]. A NN could be focusing on spurious features and display the Clever Hans effect. If that is the case, the effect could potentially be revealed through explanation methods. New research has started to explore the possibility of providing feedback on classification explanations to a

NN to prevent or remove the Clever Hans effect rather than just revealing it [Ross et al., 2017; Teso and Kersting, 2019; Erion et al., 2020; Rieger et al., 2020]. This feedback will be referred to as either feedback or explanation feedback in case of ambiguity. The primary approach has been using the gradients in the input space to act on explanations [Ross et al., 2017; Erion et al., 2020] to correct a model.

Explanations are social, meaning they are a transfer of knowledge through interactions or conversations [Miller, 2019]. Many model correction methods work as one-way communication where annotations of irrelevant features need to be provided before training. However, defining such knowledge about the data before training does not satisfy the explanations' social aspect. Additionally, defining such knowledge without considering what the model will learn through training can be challenging, making model correction methods hard to use in practice.

In this section, we propose a new ML pipeline, active feedback (AF). AF augments a "standard" ML pipeline with two additional steps. These steps try to improve classification explanations given by a model and its predictive performance by adding a human annotator into the loop. Additionally, taking a step towards making explanations in ML social. AF works by having a trained model explain some of its classifications using a backpropagation-based explanation method. An annotator provides the model with feedback on these classification explanations if needed. A feedback of a sample i is a matrix $\boldsymbol{F}^{(i)} \in \{0,1\}^{w \times h}$ (same width and height as input sample i) provided by a human annotator. If a feature k, j is irrelevant $\boldsymbol{F}^{(i)}_{k,j} = 1$, otherwise $\boldsymbol{F}^{(i)}_{k,j} = 0$. For Step 4, the model is corrected using fine-tuning[1] with the feedback provided by the annotator and the training dataset. Like AL, AF queries a user for more information. However, unlike AL, AF does not ask for labels, but feedback on classification explanations. The AF pipeline can be seen in Figure 4.1.

Step 3 of AF works by having an annotator by hand select pixels when dealing with images. However, having a method to select regions can assist a user with constructing feedback. We provide the user with such a method. Our method provides the user with possible feedback (see Figure 4.2 (right); each color is a possible feedback). The regions are divided into "concepts". Throughout the literature, the term concept is used to express more meaningful units to humans than individual pixels [Ancona et al., 2019; Zhou et al., 2018; Kim et al., 2018]. These concepts can be used as feedback in our model correction method. The model correction method extends the work of Ross et al. [2017], which proposes

[1]Fine-tuning is a form of transfer learning where we change the weights of a pre-trained model by continuing the backpropagation.

a new loss that can use classification explanations to regularize a model. Unlike Ross et al. [2017], we use the loss with transfer learning and do not expect explanation feedback without a trained model.

To summarize this chapter

1. Section 4.1 details the AF pipeline. AF is a "standard" ML pipeline extended with two additional steps. These two steps consists of, gather feedback from a human annotator and to correct a model using explanation feedback.
2. Section 4.2 proposes a feedback selection method to assist human annotators with the construction of explanation feedback.
3. Section 4.3 outlines how the loss function introduced in Ross et al. [2017] can be used for fine-tuning with explanation feedback to correct a model.

4.1 Pipeline

This section outlines AF pipeline in greater details (see Figure 4.1). AF extends a "standard" ML training procedure with three steps: explain, provide feedback, and finetune. Step 1, 2 and 5 are part of "standard" ML training and evaluation operations. Step 3 and 4 are the steps introduced by AF. The steps in AF pipeline are as follows:

1. **Train.** This step consists of training a DL model on a training dataset.
2. **Evaluate and iterate.** Using a validation dataset, the trained model is evaluated. Optionally, the trained model can be tuned using the validation during this step.
3. **Explain and provide feedback.** The model explains classifications of samples from the training dataset. The goal of this step is for a user to verify if a model makes classifications based on a "reasonable" explanation.

   The user can after inspecting an explanation, give feedback on it. The feedback is used to correct the model in case of incorrect patterns learned. The user can manually highlight regions it does not want the model to focus on. However, since this is the most time-consuming part, we want to provide the user with possible feedback. Using our region selection method, the user is provided with possible regions that it can select as feedback to the model (see Figure 4.2). This is described more in Section 4.2.2.
4. **Fine-tune** The model is fine-tuned with user-provided feedback and the original training dataset. In this step, the model correction algorithm is used

to fine-tune the model. An in-depth outline of the procedure is given in Section 4.3.

5. **Evaluate and deploy** The test dataset is applied to see if the model performs satisfactorily to be deployed in real use. Figure 4.1 depicts how an explanation changes when a model is corrected/improved. The explanation shows that the model focuses much less on background information when being told that it is irrelevant through feedback on classification explanation.

4.2 Feedback selection method

Previous research has shown that pixel-based explanations do not increase human trust and understanding in the model [Poursabzi-Sangdeh et al., 2019; Kim et al., 2018]. Also, there is evidence that regions are easier to understand than pixel-based [Sundararajan et al., 2019]. Furthermore, we know that referring to the cause is more effective than looking at exact numbers [Miller, 2019]. Hence, we conjecture that providing the user with possible regions beforehand eases providing the model with feedback.

This section presents a feedback selection algorithm that can optionally be used during Step 3 of the AF pipeline to reduce human labor. We want to provide an annotator with possible feedback, in this case, continuous regions, instead of manually highlighting pixels. The method is built on top of attribution maps produced using a backpropagation-based method. Hence, providing the annotator with regions encompassing high attribution.

4.2.1 How the feedback selection method works

We are now going to outline the method by dividing it into six steps:

1. Construct the underlying attribution maps.
2. Select the sign of attributions.
3. Threshold the attribution maps.
4. Combine attribution maps.
5. Find and select segments.
6. Merge regions using semantic segmentation (Optional).

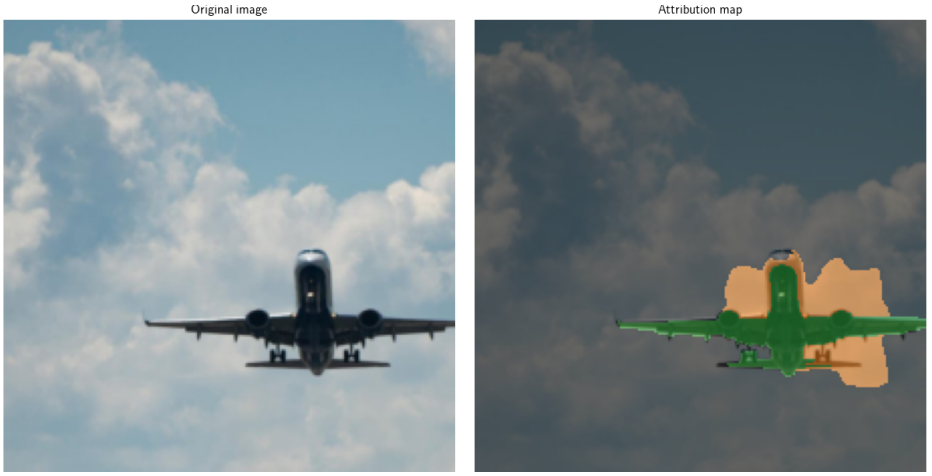


Figure 4.2: A VGG-19 model with batch normalization, trained on the ImageNet dataset. The image fed into the classifier (left) and the segmentation map of the classification (right). The segmentation map displays a continuous region that is important to the classifier. The region is further divided into smaller segments to separate concepts within the region. Each color matches a concept, here with orange indicating the background and green highlighting the airplane. This sample is classified as "airliner" with a softmax output of 0.7.

1. Construct the underlying attribution maps

Consider an input sample $\mathbf{X}^{w \times h \times c}$ and a CNN model $f_{\boldsymbol{\theta}}(\mathbf{X}) = \hat{\boldsymbol{y}}^c$ parametrized by $\boldsymbol{\theta}$. $\hat{y}_c$ of $\hat{\boldsymbol{y}}$ is the logit of class c that an annotator selects as interesting.

The method starts with a sample $\mathbf{X}$ (Figure 4.2 (left)), and calculates $\mathbf{V} = \nabla_{\mathbf{X}} y_c$ (Figure 4.4c) which is the logit gradient of class c with respect to the input. Grad-CAM (see Section 2.4.1) is employed to find attribution map $\mathbf{U}_0$ for the last convolutional layer (forward direction) (Figure 4.4b). Since the attribution map $\mathbf{U}_0$ is produced using a convolutional layer, it need to be interpolated before element-wise product with $\mathbf{V}$. The function interpolate$(\mathbf{X}_1, \mathbf{X}_2)$ interpolates $\mathbf{X}_1$ to $\mathbf{X}_2$'s size using bilinear interpolation. Bilinear interpolation is used throughout the literature to upsample the activation maps to input size [Selvaraju et al., 2020a; Bau et al., 2020; Zhou et al., 2015b; Long et al., 2014]. Hence, we get:

$$\mathbf{V} = \nabla_{\mathbf{X}} y_c \tag{4.1}$$

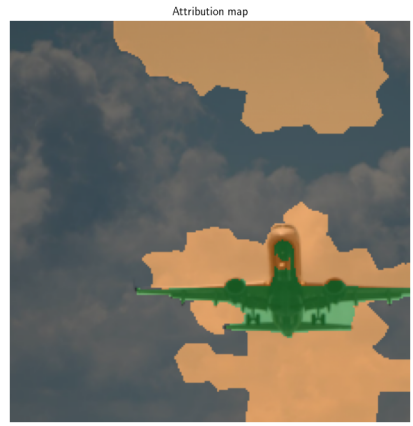
$$\mathbf{U}_0 = L^{y_c}_{\text{Grad-CAM}}(\mathbf{X}) \tag{4.2}$$

$$\mathbf{U} = \text{interpolate}(\mathbf{U}_0, \mathbf{V}) \tag{4.3}$$

For Grad-CAM, the ReLU[2] function is not applied in order to retain the negative attribution.

2. Select the sign of attributions

(a) Negative attribution



(b) Absolute attribution. The region is divided into smaller segments to separate concepts within the region (different color).

Figure 4.3: Negative and absolute attribution

The second part consists of selecting to either focus on the positive, negative, or absolute value of the attribution. Figure 4.2 displays regions that attributes positively to the class "airliner". However, in some situations it might be desirable or necessary to look at what negatively effects the model too (for the "airliner", Figure 4.3a), since negative attribution can also cause overfitting. For example, if the model learns that removing a chair from a picture of a cat increases the probability for the class "cat". We can say that the model has learned a wrong negative relationship. This negative relationship should not exists if there is no class "chair". If there was a class "chair", it would make sense since it "absorbs" some of the softmax output. The method also offers to display the absolute value of the attribution for convenience so that the user can see all regions affecting the model (for the "airliner", Figure 4.3b). However, the understanding of negative attribution is not as clear as positive attribution. Hence, in most cases only positive attribution is needed.

[2]In the original implementation, the ReLU function is used on the output attribution map

If the user selects to focus on the positive attribution:

$$\mathbf{V} = \mathbf{V} \odot \mathbf{1}_{\mathbf{V}>0}$$
$$\mathbf{U} = \mathbf{U} \odot \mathbf{1}_{\mathbf{U}>0},$$

the negative attribution:

$$\mathbf{V} = \mathbf{V}^{\text{abs}} \odot \mathbf{1}_{\mathbf{V}<0}$$
$$\mathbf{U} = \mathbf{U}^{\text{abs}} \odot \mathbf{1}_{\mathbf{U}<0},$$

or the absolute attribution:

$$\mathbf{V} = \mathbf{V}^{\text{abs}}$$
$$\mathbf{U} = \mathbf{U}^{\text{abs}}.$$

3. Threshold the attribution maps

We want the attribution of a constrained region, hence, the attribution maps need to be thresholded where the smallest attribution values are removed. We do this with both attribution maps $\mathbf{V}$ and $\mathbf{U}$.

$$\theta_1' = \arg\min_{\theta_1} \quad \mathbf{V} \odot \mathbf{1}_{\mathbf{V}>\theta_1} \tag{4.4}$$

$$\text{s.t.} \quad \frac{||\operatorname{vec}(\mathbf{V} \odot \mathbf{1}_{\mathbf{V}>\theta_1})||_1}{||\operatorname{vec}(\mathbf{V})||_1} \leq \lambda_1. \tag{4.5}$$

Thus, we get $\mathbf{V} = \mathbf{V} \odot \mathbf{1}_{\mathbf{V}>\theta_1'}$. And likewise,

$$\theta_2' = \arg\min_{\theta_2} \quad \mathbf{U} \odot \mathbf{1}_{\mathbf{U}>\theta_2} \tag{4.6}$$

$$\text{s.t.} \quad \frac{||\operatorname{vec}(\mathbf{U} \odot \mathbf{1}_{\mathbf{U}>\theta_2})||_1}{||\operatorname{vec}(\mathbf{U})||_1} \leq \lambda_1, \tag{4.7}$$

$\mathbf{U} = \mathbf{U} \odot \mathbf{1}_{\mathbf{U}>\theta_2'}$. $\lambda_1 \in [0, 1]$ is a parameter.

4. Combine attribution maps

$\mathbf{U}$ can be too coarse depending on the NN architecture. While $\mathbf{V}$ can be too spread out making it hard to construct a continuous region. Hence, we combine $\mathbf{V}$ and $\mathbf{U}$ into a single attribution map $\mathbf{W}$ (Figure 4.4h)

$$\mathbf{W} = \mathbf{V} \odot \mathbf{U},$$

to gain the benefit of both sizes. Humans tend to reason using higher-level abstract "concepts" [Rosch, 2002]. We know that the last convolutional layer represents high-level "concepts" [Zeiler and Fergus, 2014; Zhou et al., 2015b; Bau et al., 2020]. Thus, by including information from the last convolutional layer may make the attribution map align better with human intuition.

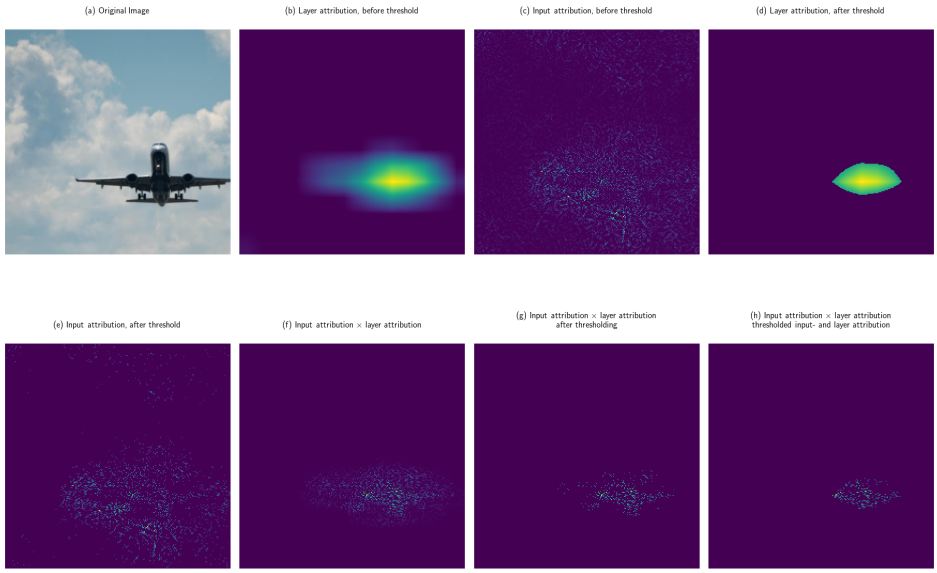
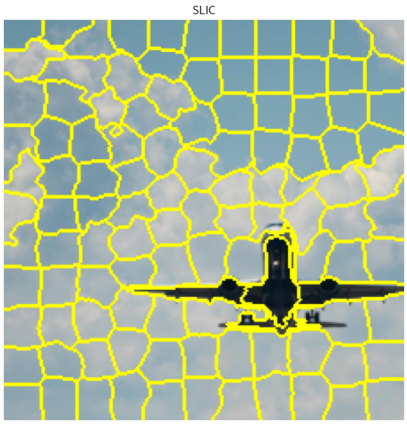


Figure 4.4: Attribution maps using Saliency and Grad-CAM. (a) The original image fed to the model. (b) Attribution map of the deepest convolutional layer. (c) Attribution map of the input. (d) Attribution map (b) after thresholding using $\lambda = 0.2$. (e) Attribution map (c) after thresholding using $\lambda = 0.2$. (g) Attribution map (b) element-wise multiplied with (c) and then thresholding with $\lambda = 0.2$. (h) Attribution map (d) element-wise multiplied with (e).

5. Find and select segments

The definition of "good" segmentation of depends on the application. We tried four different segmentation algorithms Felzenszwalb's efficient graph based segmentation [Felzenszwalb and Huttenlocher, 2004], Quickshift image segmentation [Vedaldi and Soatto, 2008], Compact watershed segmentation of gradient images [Neubert and Protzel, 2014] and SLIC [Achanta et al., 2012]. Based on the empirical results with these algorithms for our application, we found that SLIC worked satisfactory at most examples. Therefore, SLIC is used to segment the images. SLIC is an simple algorithm that uses color similarity and proximity of pixels in the image plane to cluster pixels into superpixels[3] using k-means clustering [Lloyd, 1982].

[3] A term used for a group of pixels with similar color and other low-level properties



(a) SLIC applied om Figure 4.2 (left) with n_segments = 120.

(b) Chosen segments using the attribution map $\mathbf{W}$ in figure 4.4 and $\boldsymbol{S}_i$ in Figure 4.5a.

Figure 4.5: Segmentation using SLIC and chosen segments

Following on, we segment $\mathbf{X}$ into segments using SLIC (Figure 4.5a). Since the quality of the segmentation depends on a parameter "number of segments" given to SLIC, we segment $\mathbf{X}$ with n different values for the parameter "number of segments". Therefore, we get $\boldsymbol{S}_1 \in \{0, .., t_1\}^{m \times n}, ..., \boldsymbol{S}_n \in \{0, .., t_n\}^{m \times n}$. For each $\boldsymbol{S}_i$, we select $t_j \in \boldsymbol{S}_i \odot \mathbf{1}_{\mathrm{W}>0}$:

$$\boldsymbol{M}_i = \boldsymbol{S}_i \in \boldsymbol{S}_i \odot \mathbf{1}_{\mathrm{W}>0} \; i = 1, ..n,$$

Finally, we want to select the $\boldsymbol{M}_1, ..., \boldsymbol{M}_n$ that covers the smallest area (Figure 4.3b):

$$j^* = \arg\min_j \; ||\operatorname{vec}(\boldsymbol{M}_j)||_0, \text{ for } j = 1, ..., n,$$
$$\boldsymbol{L} = \boldsymbol{M}_{j^*}$$

6. Merge regions using semantic segmentation (Optional)

To further improve the quality of the attribution map, a segmentation network can be used to segment $\mathbf{X}$ into $\boldsymbol{C}$ (Figure 4.6 (left)). The segments in $\boldsymbol{L}$ are merged if they fall within in the same bounding box in $\boldsymbol{C}$. Hence, the number of segments in $\boldsymbol{L}$ can be reduced (Figure 4.2 (right) after merge vs. Figure 4.5b before merge).

Figure 4.6 (left) is not used as an explanation, since an explanation is not the same as object detection. An explanation should highlight where the model "looks" to make its classification [Dabkowski and Gal, 2017], which does not necessary imply the whole object.

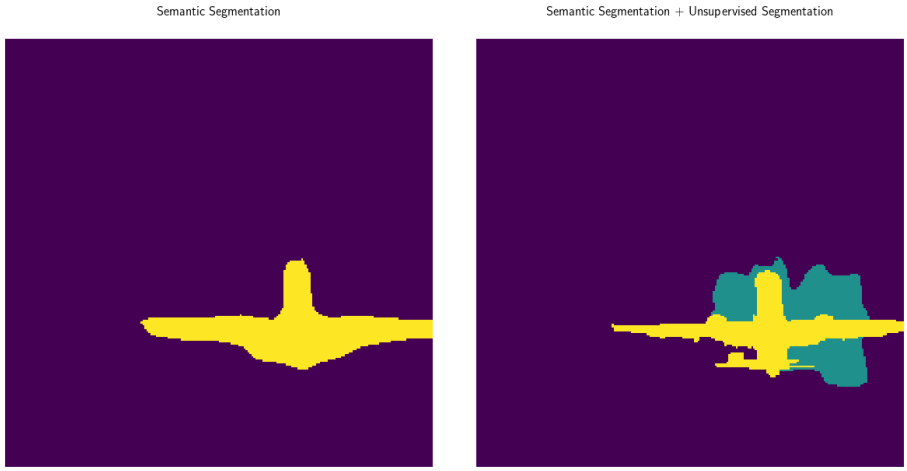


Figure 4.6: (left) Bounding box using a semantic segmentation network on Figure 4.2 (left). (right) Merging segments in Figure 4.3b that occupy the same bounding box.

The final attribution map $\boldsymbol{L}$ (Figure 4.2 (right)) consists of continuous regions that is further divided into smaller segments if the region consists of more than one concept. Each segment should consist of one concept that is meaningful and important to the model on its own. To fulfill the meaningfulness property [Ghorbani et al., 2019] for concepts, we use the SLIC algorithm and optionally a semantic segmentation network to further improve the quality of the segments. Since the segments chosen overlap with the thresholded attribution map, it should be clear that the segments highlight important areas for the model. However, it requires that the underlying attribution method is faithful to the model.

4.2.2 Representing feedback

We propose to present a collection of the possible feedback to the user using the segmentation map our feedback selection method produces. The feedback are represented as continuous regions that represent concepts in the input space. These feedback encompass regions with high attribution that will likely affect the model's decision boundary. Examples of this can be seen in Figure 4.2 where each

color represent a possible feedback. The feedback the user selects for a sample forms a binary matrix used in the fine-tuning process.

4.3 Incorporating Feedback

This section describes Step 4 of the pipeline outlined in Section 4.1, namely how to correct a model with explanation feedback provided by a human annotator.

Assume that we have a sample $\mathbf{X}^{(i)}$ with the one-hot encoding of the label $\boldsymbol{y}^{(i)}$. The softmax classification output $f(\mathbf{X}^{(i)}) = \hat{\boldsymbol{y}}^{(i)}$ is obtained by forward passing $\mathbf{X}^{(i)}$. The explanation feedback that an annotator provides for the sample i is represented as a binary matrix $\boldsymbol{F}^{(i)} \in \{0,1\}^{w \times h}$ (same height and width as the sample). $\boldsymbol{F}^{(i)}_{k,j} = 1$ indicates that a feature k, j is irrelevant for the class of interest, otherwise $\boldsymbol{F}^{(i)}_{k,j} = 0$. For samples without explanation feedback, $\boldsymbol{F}^{(i)} = \mathbf{0}$. We want to minimize

$$\mathbf{H} = \boldsymbol{F}^{(i)} \otimes \nabla^{(i)}_{\mathbf{X}}(\hat{y}^{(i)}_c) \tag{4.8}$$

to incorporate the explanation feedback. $\otimes$ signifies broadcasting and an element-wise product. Equation (4.8) signifies the attribution of features that the annotator thinks are irrelevant for a given class c. Given Equation (4.8), we have the attributions we want to minimize, thus introducing the new loss

$$\mathcal{L}^*(\boldsymbol{y}^{(i)}, \hat{\boldsymbol{y}}^{(i)}) = \mathcal{L}(\boldsymbol{y}^{(i)}, \hat{\boldsymbol{y}}^{(i)}) + \frac{\lambda_2}{w \times h \times c} \sum_{i}^{N} || \operatorname{vec}(\mathbf{H})||_1. \tag{4.9}$$

Equation (4.9) penalizes the attribution that the annotator specified as irrelevant. Additionally, it minimizes $\mathcal{L}(\boldsymbol{y}^{(i)}, \hat{\boldsymbol{y}}^{(i)})$ that represents any additional loss terms used to train the model. λ_2 is a parameter that controls the importance of the feedback penalty. The loss for the training dataset is the mean of the individual losses

$$\mathcal{L}^{\text{Tot}} = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}^*(\boldsymbol{y}^{(i)}, \hat{\boldsymbol{y}}^{(i)}), \tag{4.10}$$

assuming that the samples in the dataset are i.i.d.. Given a feedback $\boldsymbol{F}^{(i)}$, the training set and the loss function given in Equation (4.9), we incorporate the feedback into the model by fine-tuning. Fine-tuning is Step 5 of the pipeline outlined in Section 4.1. Our experimental results show that the model does not need to be trained from scratch. This method can be used together with any existing regularization techniques. Also, it does not restrict the model during training since it is applied during fine-tuning. Another advantage of the method is that the annotator is not overwhelmed with possible feedback to provide the

model beforehand. Instead, the method provides some reasons that the user can select after the fact.

Chapter 5

Bayesian model correction framework

Humans' trust in models is affected by a model's confidence in its predictive performance [Zhang et al., 2020]. NNs are bad at quantifying uncertainty and tend to produce overconfident classifications [Lakshminarayanan et al., 2017]. An overconfident model can be perceived as dangerous or offensive [Amodei et al., 2016]. The model can be seen as trying to gain false trust that will backlash against the model if it makes a wrong decision. Consequently, being overconfident can reduce trust in the model rather than increase it. Providing reasonable explanations induce trust [Ribeiro et al., 2016b], but explanations generated by these aforementioned explanation methods only capture local behaviors, meaning an explanation applies only in the vicinity of a sample. This does not give insight into how a model will perform on out-of-distribution samples. Hence, to resolve model overconfidence, the model must capture aleatoric and epistemic uncertainty [Lakshminarayanan et al., 2017]. There are several ways to capture epistemic uncertainty; one way is to frame the NNs within a Bayesian framework that applies inference on the model's weights [MacKay, 1992; Blundell et al., 2015].

Model correction and Bayesian inference contribute to robustness and trust in the model. However, they are difficult to use together because most model correction methods modify the objective function. The modifications do not necessarily follow probability calculus and can introduce elements with unknown distributions.

For a model to have the correct explanation, classification and model confidence, this work bridges the gap between model correction and Bayesian inference. Additionally, it takes a step towards making explanations in ML social. We present a Bayesian framework that uses explanation feedback. After training, a Bayesian CNN presents explanations of training sample classifications to a human annotator. The annotator can accept or reject the explanations by giving feedback as additional evidence. Feedback is represented as a matrix with the same width and height as a sample specifying which attribution regions to reject. This feedback is used during fine-tuning to correct the model such that the explanations and predictive performance improve.

In a Bayesian framework, finding the maximum likelihood estimate of the weights is replaced by finding the approximate posterior distribution of the weights closest to the true posterior using variational inference. Variational inference is used since exact inference is not tractable. Finding the approximate posterior distribution can be achieved by minimizing Kullback–Leibler (KL) divergence between the approximate posterior and the true posterior of the weights using, for example, Bayes by Backprop (BBB) (see Section 2.5.6). BBB uses gradient descent to optimize the ELBO for determine the the approximate posterior of the weights closest to the true posterior distribution.

5.1 Preliminary

Consider a probabilistic model $P(\boldsymbol{e}|\boldsymbol{w})$, the prior distribution $P(\boldsymbol{w})$ that encodes the prior knowledge about the parameter $\boldsymbol{w}$ and the evidence

$$\boldsymbol{e} = \{(\mathbf{X}^{(i)}, \boldsymbol{y}^{(i)})\}_{i=1}^{n} \tag{5.1}$$

with n i.i.d. samples. We want to compute the posterior distribution

$$P(\boldsymbol{w}|\boldsymbol{e}) = \frac{P(\boldsymbol{e}|\boldsymbol{w})P(\boldsymbol{w})}{P(\boldsymbol{e})} \tag{5.2}$$

using Bayes rule. However, exact Bayesian inference is intractable, so Markov Chain Monte Carlo (MCMC) algorithms or VI are used to approximate the weights' posterior distribution. BBB [Blundell et al., 2015] is a VI and backpropagation-compatible method for learning weight distributions. BBB uses the reparametrization trick [Kingma and Welling, 2014] to calculate a MC estimator of the ELBO and uses Gaussian variational distributions. To compute the MC estimator of ELBO, samples are drawn from the weights, which is computationally expensive. To reduce the number of samples needed, the LRT [Kingma et al., 2015] was proposed. LRT considers uncertainty at the activation level (before any nonlinear

activation function) by computing the activations before drawing samples. This significantly reduces the number of samples drawn and makes it computationally cheaper.

Example 1 (LRT for a fully connected layer)**.** *Let layer l be a fully connected layer with weights $\boldsymbol{W}^{m\times r}$ where $q_{\boldsymbol{\lambda}}(W_{i,j}) = \mathcal{N}(\mu_{W_{i,j}}, \sigma^2_{W_{i,j}})\ \forall W_{i,j} \in \boldsymbol{W}$ and biases $\boldsymbol{B}^{t\times r}$ where $q_{\boldsymbol{\lambda}}(B_{i,j}) = \mathcal{N}(\mu_{B_{i,j}}, \sigma^2_{B_{i,j}})\ \forall B_{i,j} \in \boldsymbol{B}$. Given an input $\boldsymbol{Z}^{t\times m}$, the activation of layer l is $\boldsymbol{U} = \boldsymbol{ZW} + \boldsymbol{B}$ of size $t \times r$ where*

$$q_{\boldsymbol{\lambda}}(U_{i,j}|\boldsymbol{Z}) = \mathcal{N}(\gamma_{i,j}, \delta_{i,j})\ \forall U_{i,j} \in \boldsymbol{U} \tag{5.3}$$

$$\gamma_{i,j} = \sum_{k=1}^{m} Z_{i,k}\mu_{W_{k,j}} + \mu_{B_{i,j}},\ \delta_{i,j} = \sum_{k=1}^{m} Z^2_{i,k}\sigma^2_{W_{k,j}} + \sigma^2_{B_{i,j}}. \tag{5.4}$$

A new weight matrix is sampled for every training sample to reduce estimator variance, which has a computational complexity of $\mathcal{O}(tmr)$. By sampling activations rather than weights, the computational complexity is reduced to $\mathcal{O}(tr)$.

Example 2 (LRT for a convolutional layer)**.** *Let layer l be a convolutional layer with a convolutional filter $\mathbf{W}^{w'\times h'\times \bar{c}}$ where $q_{\boldsymbol{\lambda}}(\mathrm{vec}(\mathbf{W})) = \mathcal{N}(\mathrm{vec}(\mathbf{M}), \mathrm{diag}(\mathrm{vec}(\mathbf{V}^2)))$ is a fully factorized Gaussian. Given an input $\mathbf{Z}^{\bar{w}\times\bar{h}\times\bar{c}}$, the activation of layer l is $\boldsymbol{U} = \mathbf{Z} * \mathbf{W}$ of size $\hat{w} \times \hat{h}$ where*

$$q_{\boldsymbol{\lambda}}(\mathrm{vec}(\boldsymbol{U})|\mathbf{Z}) = \mathcal{N}(\boldsymbol{\mu_U}, \boldsymbol{\sigma^2_U}) \tag{5.5}$$

$$\boldsymbol{\mu_U} = \mathrm{vec}(\mathbf{Z} * \mathbf{M}),\ \boldsymbol{\sigma^2_U} = \mathrm{diag}(\mathrm{vec}(\mathbf{Z}^2 * \mathbf{V}^2)) \tag{5.6}$$

is treated as a fully factorized Gaussian. We define $\mathrm{vec}(\cdot)$ *as a vectorization function,* $(\cdot)^2$ *as an element-wise operator,* $*$ *as a convolution operator and* $\mathrm{diag}(\cdot)$ *as a diagonal matrix.*

5.2 Variational Inference and Explanation Feedback

Sections 5.2.1 and 5.2.2 outline the objective function used in the fine-tuning phase and how feedback is induced into a model by adding additional evidence to $\boldsymbol{e}$.

5.2.1 Add Explanation Feedback to Evidence

We want to compute the posterior distribution $P(\boldsymbol{w}|\boldsymbol{e})$ of the weights of a NN. The evaluation involves computation of intractable integrals. Therefore, approximation techniques are used. VI is one of the most widely used and defines a variational distribution $q_{\boldsymbol{\lambda}}(\boldsymbol{w})$ that is used to approximate the posterior distribution by minimizing the KL divergence $D_{\mathrm{KL}}(q_{\boldsymbol{\lambda}}(\boldsymbol{w})\|P(\boldsymbol{w}|\boldsymbol{e}))$. Minimizing the KL divergence is equivalent to maximizing the ELBO

$$\mathcal{L}_{\mathrm{ELBO}} = \underbrace{\mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{w})}[\log P(\boldsymbol{e}|\boldsymbol{w})]}_{\text{Likelihood}} - \underbrace{D_{\mathrm{KL}}(q_{\boldsymbol{\lambda}}(\boldsymbol{w})\|P(\boldsymbol{w}))}_{\text{Complexity}}. \tag{5.7}$$

Let ${\mathbf{X}^{(i)}}^{w\times h\times c}$ be an observation with the label $\boldsymbol{y}^{(i)}$, a feedback of the sample i is a matrix $\boldsymbol{F}^{(i)} \in \{0,1\}^{w\times h}$ given by a human annotator. If a feature k, j is irrelevant $\boldsymbol{F}^{(i)}_{k,j} = 1$, otherwise $\boldsymbol{F}^{(i)}_{k,j} = 0$. Furthermore, let f be a Bayesian CNN that takes $\mathbf{X}^{(i)}$ as input and outputs the logit vector $f(\mathbf{X}^{(i)}) = \hat{\boldsymbol{y}}^{(i)}$.

Consider input gradients $\nabla_{\mathbf{X}^{(i)}} \sum_j \hat{y}^{(i)}_j$ of the observation $\mathbf{X}^{(i)}$ that can be used as an explanation that specifies feature attributions.

$$\mathbf{H}^{(i)} = \boldsymbol{F}^{(i)} \otimes (\nabla_{\mathbf{X}^{(i)}} \sum_j \hat{y}^{(i)}_j) \tag{5.8}$$

is the attributions on irrelevant features that we want to minimize. $\otimes$ signifies broadcasting and an element-wise product. To include explanation feedback, we add additional evidence to $\boldsymbol{e}$

$$\boldsymbol{e}_{\mathrm{grad}} = \boldsymbol{e} \cup \{\mathbf{H}^{(i)} = \mathbf{0}\}_{i=1}^{n} \tag{5.9}$$

where we set $\mathbf{H}^{(i)} = \mathbf{0}$ to constrain attributions on irrelevant features, which is similar to previous work [Ross et al., 2017].

However, the distribution of the input gradients is unknown and it is hard to find analytically. Not knowing the distribution makes us unable to calculate the likelihood in Equation (5.7) with $\boldsymbol{e}_{\mathrm{grad}}$ as evidence. One way to solve this issue is to look at some quantity with known distribution instead of the input gradients. Although weight distributions are known, it is not trivial to map feedback given in the input space to the parameter space. Therefore, we use the LRT to consider the activation distributions and map the feedback to activation space. Examples 1 and 2 describe how to obtain activation distributions.

In the next section, we show how considering the uncertainty at the activation level enables us to approximate the evidence $\boldsymbol{e}_{\mathrm{grad}}$ with an alternative formulation for which the likelihood can be expressed analytically.

We use CNNs since the activations in convolutional layers retain spatial information. The spatial information is needed to map feedback defined in the input space to corresponding activations. An additional advantage of using activations rather than input gradients is reduced computational complexity since second-order partial derivatives are not needed.

5.2.2 Methodology

This section details how to add feedback to $\boldsymbol{e}$ using activations rather than input gradients. We do this by describing how to map the feedback from input to activation space. Moreover, we describe how to compute Equation (5.7) with the new evidence $\boldsymbol{e}_{\text{act}}$ (to be defined in Equation (5.13)); something we could not do with $\boldsymbol{e}_{\text{grad}}$.

Consider the same setup as in Section 5.2.1. Let $\mathbf{A}^{(i)} \in \mathbb{R}^{u \times v \times d}$ $(u \leq w,\ v \leq h)$ be feature maps (before any nonlinear activation function) of the last convolutional layer (forward direction) obtained by forward passing $\mathbf{X}^{(i)}$. The last convolutional layer is used since it contains the highest abstraction level of features [Zeiler and Fergus, 2013; Zhou et al., 2015b; Selvaraju et al., 2019].

Map Feedback from Input to Activation Space

The feedback will be mapped to the activation space to include it in the evidence $\boldsymbol{e}$. We do this in two steps:

1. downsample the feedback to the width and height of the feature maps $\mathbf{A}^{(i)}$ and
2. find activations in $\mathbf{A}^{(i)}$ spatially overlapping with the feedback and affecting the classification.

We downsample the feedback $\boldsymbol{F}^{(i)}$ to the width and height of $\mathbf{A}^{(i)}$ by computing

$$\boldsymbol{J}^{(i)} = \text{AdaptiveMaxPool}(\boldsymbol{F}^{(i)}, (u, v)). \tag{5.10}$$

The function $\text{AdaptiveMaxPool}(\boldsymbol{Z}, \boldsymbol{o})$ takes a matrix $\boldsymbol{Z}$ and uses max-pooling to transform it to size $\boldsymbol{o} = (o_1, o_2)$. This operation is similar to how Grad-CAM interpolates attribution maps to the input space to visualize explanations. Next, we find entries in $\boldsymbol{J}^{(i)}$ that overlap with $\mathbf{A}^{(i)}$ by calculating

$$\mathbf{G}^{(i)} = \mathbf{A}^{(i)} \otimes \boldsymbol{J}^{(i)}. \tag{5.11}$$

CNNs often use rectified linear unit and max-pooling, implying that classifications are only affected by a subset of the activations. Therefore, activations not used

in the classification will be excluded by computing

$$\mathbf{L}^{(i)} = \mathbf{G}^{(i)} \odot \mathbf{1}_{(\nabla_{\mathbf{A}^{(i)}} \sum_j \hat{y}_j^{(i)}) \neq 0} \tag{5.12}$$

to zero out entries in $\mathbf{G}^{(i)}$ that do not affect the classification. $\mathbf{L}^{(i)}$ is a tensor representing the activations that affect the classification and overlap with all indices k, j where $\boldsymbol{J}_{k,j}^{(i)} = 1$. The goal is to find the random variables where values in $\mathbf{L}^{(i)} \neq 0$ are drawn from. We will refer to those activations as $\boldsymbol{a}_{\mathcal{F}}^{(i)}$. The remaining activations in the network will be denoted $\boldsymbol{a}_{\mathcal{F}^c}^{(i)}$. We substitute input gradients with activations and add feedback to $\boldsymbol{e}$ by defining

$$\boldsymbol{e}_{\text{act}} = \boldsymbol{e} \cup \{\mathbf{a}_{\mathcal{F}}^{(i)} = \mathbf{0}\}_{i=1}^{n} \tag{5.13}$$

The process of finding $\boldsymbol{a}_{\mathcal{F}}^{(i)}$incurs an extra forward pass.

Compute the ELBO with Additional Evidence

In this section, we show how to compute the ELBO with $\boldsymbol{e}_{\text{act}}$. We follow Kingma et al. [2015] and consider uncertainty at the level of activations. This implies that the expectation in Equation (5.7) will be computed with respect to $q_{\boldsymbol{\lambda}}(\boldsymbol{a}_{\mathcal{F}^c}^{(i)}, \boldsymbol{a}_{\mathcal{F}}^{(i)} | \mathbf{X}^{(i)})$ rather than $q_{\boldsymbol{\lambda}}(\boldsymbol{w})$. $q_{\boldsymbol{\lambda}}(\boldsymbol{a}_{\mathcal{F}^c}^{(i)} | \mathbf{X}^{(i)}, \mathbf{a}_{\mathcal{F}}^{(i)} = \mathbf{0})$ will be written as $q_{\boldsymbol{\lambda}}(\boldsymbol{a}_{\mathcal{F}^c}^{(i)})$ for short. $\mathcal{L}_{\text{ELBO}}(\mathbf{X}^{(1)}, ..., \mathbf{X}^{(n)}) = \sum_{i=1}^{n} \mathcal{L}_{\text{ELBO}}^{(i)}(\mathbf{X}^{(i)})$ since the samples are i.i.d. We can reformulate Equation (5.7) as a loss function using activations and the evidence $\boldsymbol{e}_{\text{act}}$ for a single sample i as

$$\mathcal{L}_{\text{LRT}}^{(i)} = \frac{1}{n} D_{\text{KL}}(q_{\boldsymbol{\lambda}}(\boldsymbol{w}) \| P(\boldsymbol{w})) - \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{a}_{\mathcal{F}^c}^{(i)})}[\log P(\boldsymbol{e}_{\text{act}}^{(i)} | \boldsymbol{a}_{\mathcal{F}^c}^{(i)})]. \tag{5.14}$$

Expanding $\boldsymbol{e}_{\text{act}}^{(i)}$ in the log probability in Equation (5.14) gives

$$\begin{aligned} \log(\boldsymbol{e}_{\text{act}}^{(i)} | \boldsymbol{a}_{\mathcal{F}^c}^{(i)}) &= \log P(\boldsymbol{y}^{(i)}, \mathbf{a}_{\mathcal{F}}^{(i)} = \mathbf{0} | \mathbf{X}^{(i)}, \boldsymbol{a}_{\mathcal{F}^c}^{(i)}) \\ &= \log P(\boldsymbol{y}^{(i)} | \mathbf{a}_{\mathcal{F}}^{(i)} = \mathbf{0}, \boldsymbol{a}_{\mathcal{F}^c}^{(i)}) + \log P(\mathbf{a}_{\mathcal{F}}^{(i)} = \mathbf{0} | \mathbf{X}^{(i)}, \boldsymbol{a}_{\mathcal{F}^c}^{(i)}) \end{aligned} \tag{5.15}$$

where $\boldsymbol{a}_{\mathcal{F}^c}^{(i)}$ and $\boldsymbol{a}_{\mathcal{F}}^{(i)}$ are treated as fully factorized Gaussians. To compute Equation (5.14), we follow the result of BBB and approximate the negative log-likelihood by applying MC sampling. When both $q_{\boldsymbol{\lambda}}(\boldsymbol{w})$ and $P(\boldsymbol{w})$ are fully factorized Gaussians, the complexity term can be calculated in closed form [Kingma and Welling, 2014]. Equation (5.14) is an objective function that uses the evidence $\boldsymbol{e}_{\text{act}}$ to incorporate feedback from an annotator and is theoretically justified from a probabilistic perspective, making it mathematically grounded. Moreover, we do not need a hyperparameter that deals with feedback penalty; instead, the objective function trades-off between complexity and data.

Example 3 (How to approximate Equation (5.14))**.** *Consider* $\boldsymbol{w} \in \mathbb{R}^j$*, where the prior* $P(\boldsymbol{w}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ *and the approximate posterior* $q_{\boldsymbol{\lambda}}(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{\mu}, diag(\boldsymbol{\sigma}^2))$ *are both fully factorized Gaussians. Hence, Equation* (5.14) *can be approximated as*

$$\begin{aligned}\mathcal{L}_{LRT}^{(i)} \approx & \frac{1}{2n}\sum_{i=1}^{j}(\mu_i^2 + \sigma_i^2 - 1 - \log\sigma_i^2) \\ & - \frac{1}{m}\sum_{t=1}^{m}[\log P(\boldsymbol{y}^{(i)}|\mathbf{a}_{\mathcal{F}}^{(i,t)} = \mathbf{0}, \boldsymbol{a}_{\mathcal{F}^c}^{(i,t)}) + \log P(\mathbf{a}_{\mathcal{F}}^{(i,t)} = \mathbf{0}|\mathbf{X}^{(i)}, \boldsymbol{a}_{\mathcal{F}^c}^{(i,t)})].\end{aligned} \tag{5.16}$$

m is the number of MC samples.

To summarize, feedback defined in the input space by an annotator is mapped to the activations. The feedback is included by adding it as additional evidence to $\boldsymbol{e}$. During Step 2, the model is trained with $\boldsymbol{e}$ as the evidence and Equation (5.7) as the objective function. As for Step 4, to fine-tune the model, Equation (5.14) is used as the objective function with $\boldsymbol{e}_{\text{act}}$ as the evidence.

Chapter 6

Experiments and Results

In this chapter, we outline the experiments from planning to execution. Section 6.1 describes the planned experiments and the questions those experiments should answer, which is the Research questions 2 and 3. Section 6.2 provides an overview of the experimental setup in order for others to recreate the experiments. Finally, Section 6.3 shows the experimental results of the experiments outlined in Section 6.1.

6.1 Experimental Plan

In this section, we will outline the planned experiments that will be carried out in Section 6.3. Three datasets, decoy MNIST, Dogs vs. Cats, and International Skin Imaging Collaboration (ISIC) skin cancer, will be used to demonstrate the effectiveness of the method introduced in Chapter 4 and Chapter 5. Chapter 4 describes a method to correct NNs based on explanation feedback provided by a human annotator. Furthermore, in the same chapter, we propose a pipeline for the feedback process. Chapter 5 extends the idea of using explanation feedback to correct a model to include Bayesian NNs. To show the proposed method's effectiveness, we will combine the pipeline outlined in Section 4.1 and model correction method described in Chapter 5 and do the experiments.

There are two questions that the experiments should answer:

Model correction. The first part of the experiment is to see whether it is possible to correct a model by having an annotator provide explanation feedback. Apart from whether the method works, an important aspect is how much

feedback is needed to correct a model. For the method to be effective, the number of explanation feedback needed must be "small" compared to the dataset size. This is especially true for this method as it requires human intervention. Hence, one of the questions the experiment should answer is the effectiveness of the presented model correction method. The effectiveness embodies the model performance and the size of the feedback set.

Model correction with Bayesian NNs. Given that the first part provides a positive answer. In the second part of the experiment, we want to see if it is possible to extend the work in Section 4.3 to Bayesian NNs (introduced in Chapter 5). In the Bayesian setting, we want to find the posterior distribution of the parameters in a model. This requires a new method to induce explanation feedback into a model without invalidating the principled method of finding the posterior distribution.

6.2 Experimental Setup

This section gives an overview of the datasets and model architectures used in the experiments described later in this chapter. Furthermore, we describe the model architectures used to generate the results presented in Chapter 4. The purpose of this section is to give the reader a sufficient amount of information to recreate the experiments done in Section 6.3.

6.2.1 Dataset

This section outlines the datasets used in the experiments described in this chapter. Moreover, it presents the explanation feedback used for the datasets to simulate the feedback process by a human annotator.

Decoy MNIST

The MNIST [Lecun et al., 1998] dataset consists of 28×28 grayscale images of digits. The training set has 60000 samples, and the test set has 10000 samples. We divided the training dataset into 54000 samples for training and 6000 samples for validation during the experiments. The dataset has 10 classes, the digits $0, 1, \ldots, 9$. The decoy MNIST is inspired by Ross et al. [2017]. However, our dataset has some modifications compared to the one found in Ross et al. [2017]. Every sample has a 4×4 square in each corner (see Figure 6.1). In the training data, the decoys' colors correspond with the label of the digit y, $(255 - 25y)$, and in the validation and test data, the colors are randomly drawn. The decoys on the training dataset are robust since it appears on every sample without noise. This makes it hard for classifiers to learn the non-decoy features. We modified

Figure 6.1: Decoy MNIST dataset

the dataset since when the decoy rule is not apparent, the CNNs models do not overfit. As these decoys are synthetically induced, we know the ground truth feedback. Thus, the feedback $\boldsymbol{F}_{k,j}^{(i)} = 1$ when index k, j overlaps with a white patch.

Dogs vs. Cats

The Dogs vs. Cats [Kaggle, 2014] dataset consists of RGB images of dogs and cats (see Figure 6.2). The images are of different sizes, but is scaled to 227×227. The dataset has a training set of 25000 samples. The test set consists of 12500 samples, but does not have labels. Thus, the training set is divided into 90% training, 5% validation and 5% test. For the construction of feedback, a pre-trained DeepLabV3 ResNet101 [Chen et al., 2017] semantic segmentation network

Figure 6.2: Dogs vs. Cats dataset

was used. $\boldsymbol{F}_{k,j}^{(i)} = 1$ when index k, j does not overlap with an animal.

6.2.2 ISIC dataset

The version of the ISIC skin cancer data by Rieger et al. [2020] that we used has 21654 samples where 2284 are malignant (label 1) and 19370 are benign (label 0). Of those benign samples, 48% (9209 samples) have colorful patches (see Figure 6.3). Feedback for samples in this dataset are regions that contain those colorful patches because they are irrelevant for the classifications.

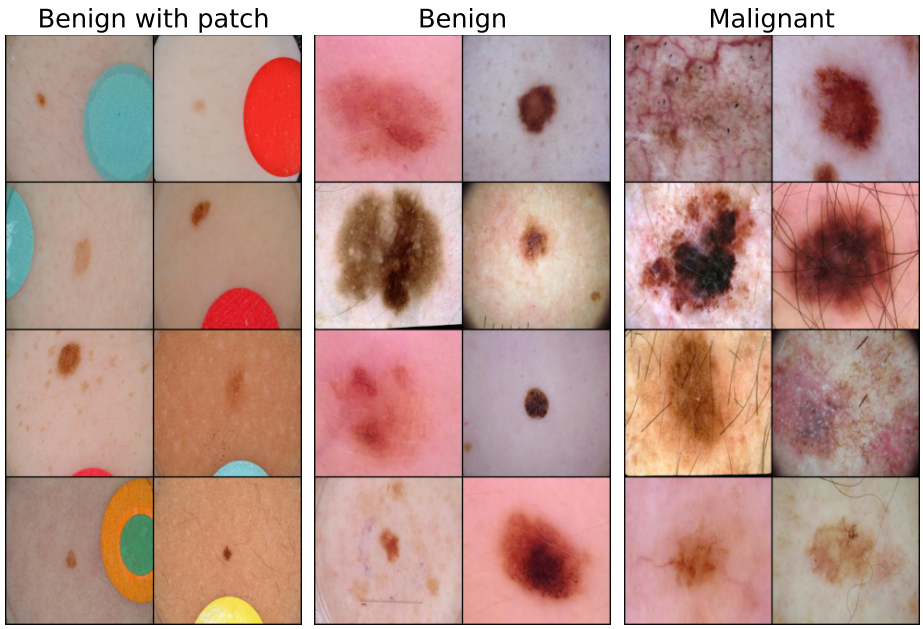


Figure 6.3: International Skin Imaging Collaboration (ISIC) dataset. In this dataset, there are two classes, benign and malignant. However, some of the samples in the benign class have colorful patches (column 1).

6.2.3 Models

In this section, we outline the NN architectures we used in the experiments.

LeNet

LeNet [LeCun et al., 1989] is a CNN architecture and one of the earliest of its kind. The architecture consists of 5 layers, 3 convolutional layers, each followed by a pooling layer except for the last convolutional layer. The last two layers in this architecture are fully connected layers. The architecture can be seen in Figure 6.4.

AlexNet

For some of the experiments, we used a pre-trained CNN AlexNet [Krizhevsky et al., 2012] trained on the ImageNet dataset [Russakovsky et al., 2015] as our model. AlexNet is what made CNNs famous, known for the ImageNet Large

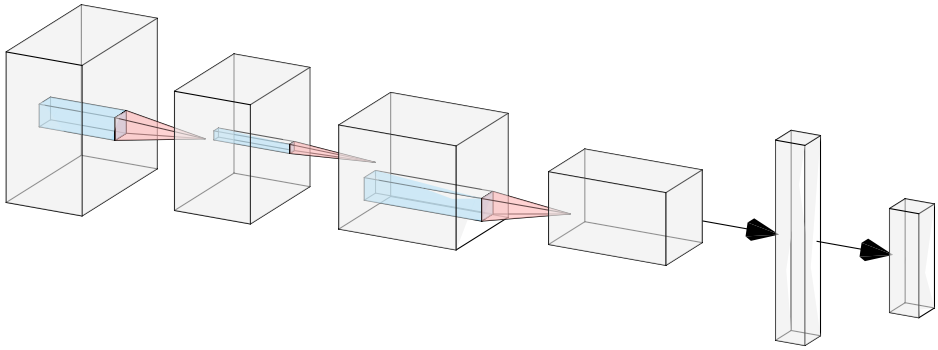


Figure 6.4: LeNet architecture

Scale Visual Recognition Challenge (ILSVRC) 2012 which it won by a large margin (15.3% vs 26.2% (second place) error rates). The AlexNet architecture consists of 5 convolutional layers with overlapping max-pooling layers and two fully connected layers, as seen in Figure 6.5. ReLU activation is applied after every convolution and fully connected layers. The overlapping max-pooling is applied in the first, second, and fifth convolution layer after the ReLU activation. For the fully connected layers, dropout is applied with a dropout rate of 0.5. The original AlexNet architecture had local response normalization (LRN) [Krizhevsky et al., 2012] after ReLU activations, but before max-pooling in the first and second convolutional layer. It was showed later that LRN does not improve the performance on the ImagNet dataset but leads to increased memory and computation time [Simonyan and Zisserman, 2015].

For the experiment used to demonstrate the method outline in Chapter 5, we removed the dropout layers and added batch normalization after every layer. We did it since it gave us more stable training.

VGG16

VGG16 [Simonyan and Zisserman, 2015] is one of the most famous model architectures submitted to ILSVRC 2014. The model achieves 92.7% top-5 test accuracy in the ImageNet dataset. The model significantly outperforms the previous generation of models in the ILSVRC 2012 and ILSVRC 2013 competitions. It improves AlexNet by replacing large convolutional filters with smaller 3×3 filters that are stacked. The model uses ReLU activation after every layer and uses max-pooling after stacks of convolutional layers. In the fully connected layers, dropout with a dropout rate of 0.5 is applied. We used a pre-trained VGG16 trained on the ImageNet dataset by PyTorch [Paszke et al., 2019] to demonstrate

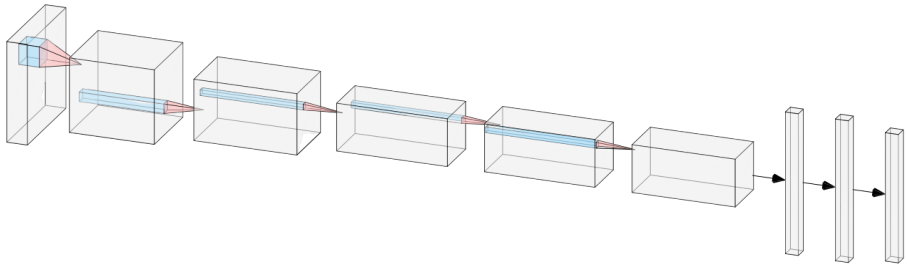


Figure 6.5: AlexNet architecture for the ImageNet dataset

the feedback selection method presented in Section 4.2.

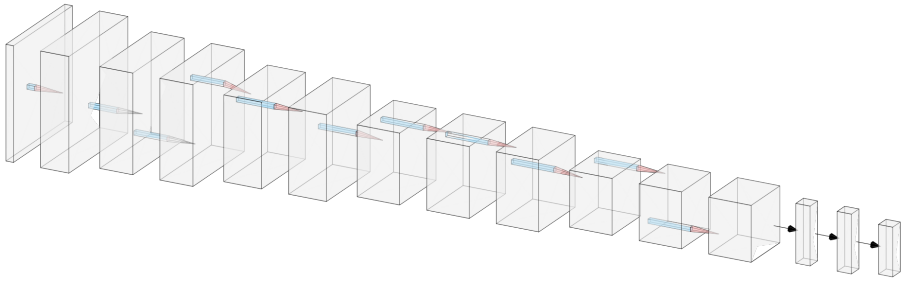


Figure 6.6: VGG16 architecture for the ImageNet dataset

6.2.4 Hyperparameters

This section gives an overview of the hyperparameters used to train the models. Table 6.1 shows the hyperparameters that are commonly used for all experiments. If a parameter is not listed, the default value from PyTorch is used.

Hyperparameter	Value
Learning rate	0.001
Batch size size	256

Table 6.1: Hyperparameters used in the experiments. Same hyperparameters were used for all experiments

6.3 Experimental Results

In this section, we present the experimental results on the datasets presented in Section 6.2.1 in the following order, Section 6.3.1 presents Decoy MNIST, Section 6.3.2 presents Dogs vs. Cats and Section 6.3.3 presents ISIC. The results show the effectiveness of the method presented in Chapter 5. The method in Chapter 4 is a subset of the final method presented in Chapter 5. Thus, the results apply to the method from Chapter 4 too. The method from Chapter 5 is used with the pipeline presented in Chapter 4.

6.3.1 Decoy MNIST dataset

To observe how much of the original predictive performance is recovered through model correction, the model was trained with MNIST (7 epochs) and decoy MNIST. The model trained without decoys has 98.7% accuracy. After adding decoys, the accuracy drops down to 68.8%. By fine-tuning the model with feedback, the difference between the original accuracy is reduced to 2.6%. The decoy rule has no randomness, which makes it more difficult to recover the original accuracy. Most previous works have feedback on all training samples. In contrast, this approach has feedback on only 0.3% of the training data, making it more appealing.

Figure 6.8 and Table 6.4 show that the model focuses less on the decoys after fine-tuning on feedback. Thus, confirming that the reason the accuracy increases is because of less focus on decoys after fine-tuning. Figure 6.7 indicates that fine-tuning increases the accuracy of all classes. However, the model still struggles more with classes 8 and 9, same as without feedback.

Attribute	No feedback	Feedback	Without decoy
Epochs	1	15	7
Feedback data size	-	0.003 (162)	-

Table 6.2: Training statistics. For model with feedback, the epoch is number of finetuning epochs. The feedback data size is the percentage of the training dataset size.

Metric	No feedback	Feedback	No decoy
Accuracy	0.688	**0.961**	0.987
F1	0.688	**0.961**	0.987
Precision	0.688	**0.961**	0.987
Recall	0.688	**0.961**	0.987

Table 6.3: Metrics of test dataset. The F1, precision and recall scores are calculated using macro average since micro average will yield the same result as accuracy.

Explanation method	No feedback	Feedback
Saliency	0.040	**0.026**
DeepLIFT	0.025	**0.018**
Occlusion	0.156	**0.082**
Grad-CAM	0.071	**0.026**

Table 6.4: Attributions overlapping with irrelevant features averaged over all samples in the test dataset with annotated explanation. The attribution overlap score is bounded $[0, 1]$ and a **lower** score is better because it implies less attention is focused on irrelevant features. For Occlusion, a sliding window of size 3×3 was used.

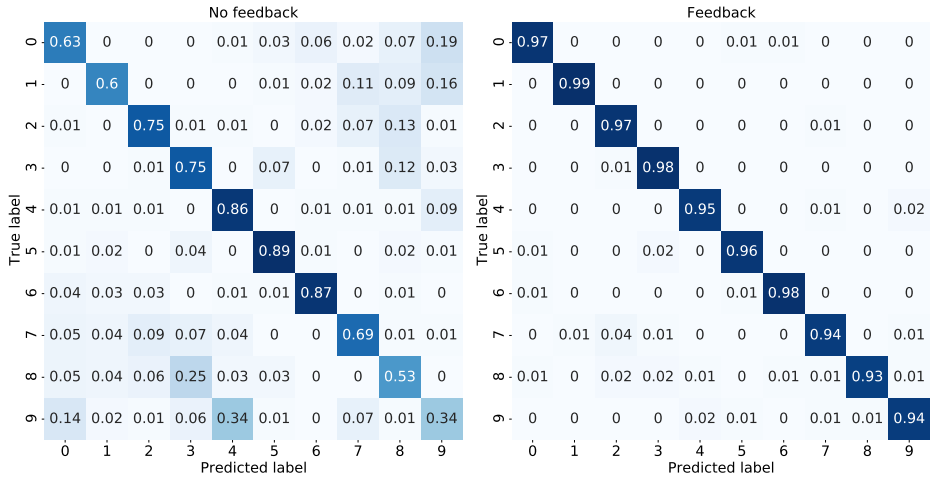


Figure 6.7: MNIST dataset confusion matrix on test dataset.

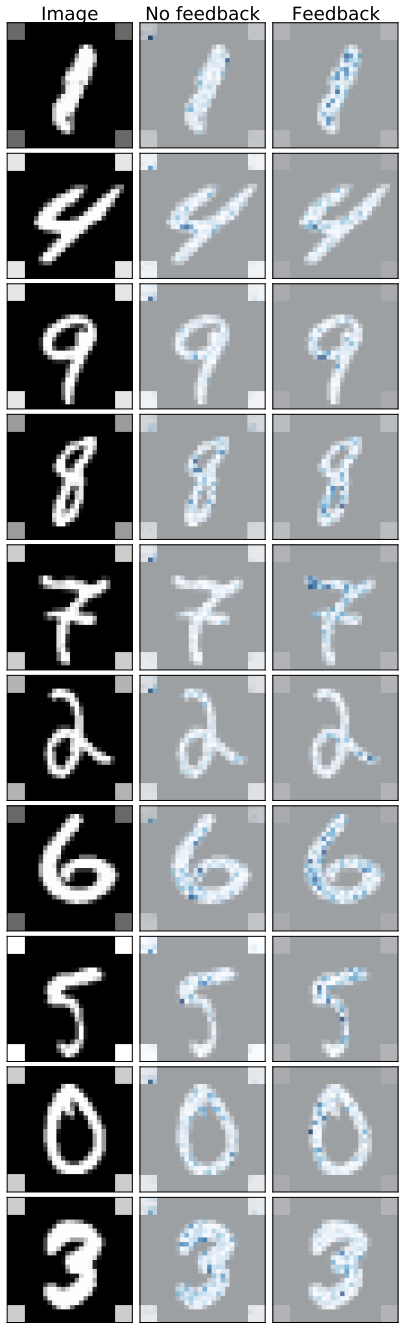


Figure 6.8: Samples from the training dataset. Column 2 and 3 display heatmaps of the samples in column 1. A **darker color** implies a higher attribution. The model focuses on both the decoys and digits before feedback. After feedback, only the digits are used for classifications.

6.3.2 Dogs vs. Cats dataset

Dogs vs. Cats is the only dataset without any apparent irrelevant features repeating in several samples. Therefore, only a slight improvement in the model's predictive performance can be seen in Table 6.6 and Figure 6.9. Furthermore, more feedback data is needed to affect the model as shown in Table 6.5. After fine-tuning with feedback, attributions overlap more with the animals (see Figure 6.10 and Table 6.7). Interestingly, explaining not to focus on background information results in the model focusing on the animals' faces instead of the whole body.

	No feedback	Feedback
Epochs	765	16
Feedback data size	-	0.045 (1012)

Table 6.5: Training statistics. For model with feedback, the epoch is number of finetuning epochs. The feedback data size is the percentage of the training dataset size.

Metric	No feedback	Feedback
Accuracy	0.886	**0.894**
F1	0.887	**0.896**
Precision	0.918	**0.923**
Recall	0.857	**0.870**

Table 6.6: Metrics of test dataset

Explanation method	No feedback	Feedback
Saliency	0.396	**0.384**
DeepLIFT	**0.405**	0.407
Occlusion	0.441	**0.379**
Grad-CAM	0.393	**0.380**

Table 6.7: Attributions overlapping with irrelevant features averaged over all samples in the test dataset with annotated explanation. The attribution overlap score is bounded $[0, 1]$ and a **lower** score is better because it implies less attention is focused on irrelevant features. For Occlusion, a sliding window of size 23×23 was used.

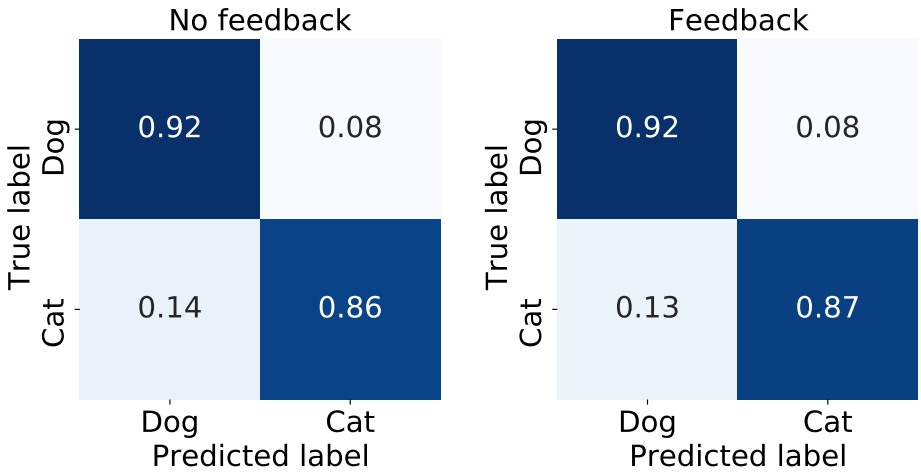


Figure 6.9: Dogs vs. cats dataset confusion matrix on test dataset.

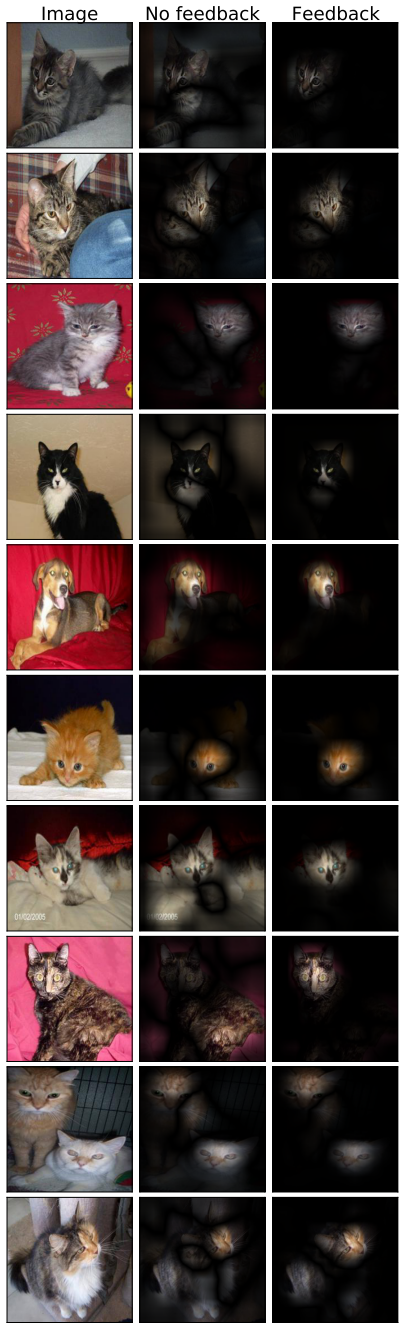


Figure 6.10: Samples from the test dataset. After feedback, the model becomes sharper and focuses more on the animals than the background.

6.3.3 ISIC dataset

The models were tested both with and without patches to observe the patches' effect. Table 6.9 and Figure 6.11 show that with patch data, the model without feedback has better accuracy than the model fine-tuned with feedback. However, without patch data, the model fine-tuned with feedback has better accuracy. Moreover, it is more trustworthy based on the explanations (Figure 6.12).

Before feedback, the model uses patches to "cheat" on benign samples. When patches are removed, the model without feedback has the same number of false positives and true negatives, classifying benign samples randomly. In contrast, the model fine-tuned with feedback does not do that. Figure 6.12 demonstrates that the model with feedback focuses much less on the patches. Table 6.10 displays the same result quantitatively. Also, the positions of the patches appear not to matter.

Figure 6.11 indicates that the model without feedback predicts random on benign samples when patches are included. Moreover, the confusion matrix demonstrates that without feedback, the model is better on malignant samples. We conjecture that it might be because it uses the information that the malignant samples do not contain a patch.

	No feedback	Feedback
Epochs	997	1
Feedback data size	-	0.015 (292)

Table 6.8: Training statistics. For model with feedback, the epoch is number of finetuning epochs. The feedback data size is the percentage of the training dataset size.

Patch	Precision		Recall		F1		Accuracy	
	NF	F	NF	F	NF	F	NF	F
Yes	0.280	**0.320**	**0.904**	0.798	0.427	**0.457**	**0.815**	0.799
No	0.289	**0.335**	**0.904**	0.798	0.437	**0.472**	0.702	**0.721**

Table 6.9: Performance metrics of the model trained with no feedback (NF) and feedback (F). The dataset is tested with and without patch data and accuracy is computed with macro average recall, also known as balanced accuracy.

Explanation method	No feedback	Feedback
Saliency	0.154	**0.111**
DeepLIFT	0.145	**0.124**
Grad-CAM	0.221	**0.071**
Occlusion	0.250	**0.148**

Table 6.10: Attributions overlapping with irrelevant features averaged over all samples in the test dataset with annotated explanation. The attribution overlap score is bounded $[0, 1]$ and a **lower** score is better because it implies less attention is focused on irrelevant features. For Occlusion, a sliding window of size 23×23 was used.

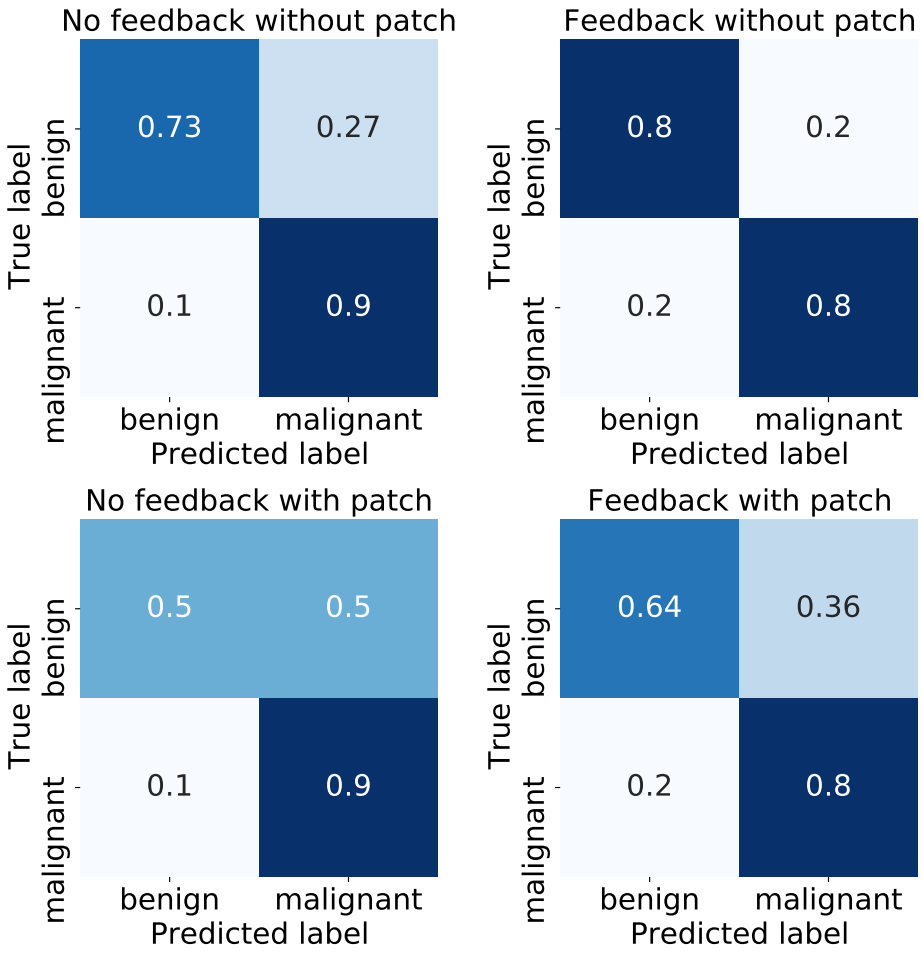


Figure 6.11: ISIC confusion matrix on test data. First row without spurious features, and second with.

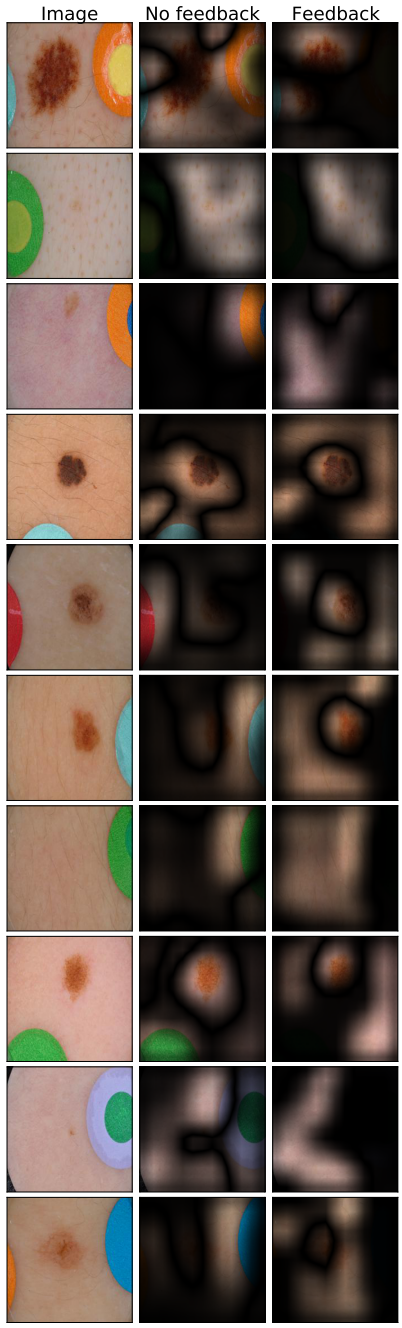


Figure 6.12: Samples from the test dataset with spurious features. Before feedback, the model uses irrelevant patches and moles to classify samples. After feedback, it uses only moles on these samples.

Chapter 7

Evaluation and Conclusion

This chapter evaluates and discusses our proposed method and its experimental results presented in Chapter 6. Furthermore, the section details the contributions made in this thesis and presents an overview of future work directions.

7.1 Evaluation and Discussion

This thesis presents a Bayesian framework that uses explanation feedback to correct a model so that classification explanations and predictive performance improve, making the model more trustworthy and understandable. The effectiveness of the proposed approach is shown on one toy dataset (see Section 6.3.1) and two real-world datasets (see Sections 6.3.2 and 6.3.3). The results indicate that focus on dataset biases are reduced, and the features representing the data's true underlying relationship are targeted more distinctly. The proposed method, therefore, pays less attention to irrelevant features. Furthermore, only a few annotated explanations and fine-tuning epochs are needed to observe this effect. We will in this section evaluate and discuss the experimental results from Section 6.3.

The model trained on the decoy MNIST dataset shows that providing explanation feedback and fine-tuning the model corrects it. We can observed the improvement in the different metrics shown in Tables 6.3 and 6.4 and the confusion matrix in Figure 6.7. Moreover, the improvement is qualitatively seen in Figure 6.8.

The change is expected since the decoys are syntactically added. However, we can notice that the predictive performance is lower than on the model trained on MNIST even after model correction. We conjecture that it is due to the robustness of the decoys. Thus, it is hard for the model to ignore these features even after fine-tuning. Nevertheless, these results indicate positive answers on Research questions 2 and 3.

The Dogs vs. Cats dataset has no obvious spurious features the model can overfit on. As a result, the model performs well without model correction as seen in Table 6.6 and Figure 6.9. After the model is fine-tuned with feedback, the predictive performance does not change much, which is as expected. However, qualitatively, it is possible to see changes in Figure 6.10. We observe that the model's focus is more tightly bounded. Furthermore, this can quantitatively be observed in Table 6.7. Unlike the decoy MNIST dataset, it is neither easy to verify nor conclude that explanation feedback has any positive effect on the model trained on this dataset.

The ISIC skin cancer dataset is similar to the decoy MNIST dataset since they contain spurious features. Table 6.9 and Figure 6.11 display metrics that shows model correction improves the model trained on this dataset. Especially noticeable is when we test the model without samples that contain spurious features. When samples with spurious features are excluded, the uncorrected model classifies benign samples randomly. The corrected model, on the other hand, does not exhibit this behavior. Qualitatively shown in Figure 6.12, we can notice that most of the attribution on spurious features is removed by model correction. Furthermore, a similar outcome can be reached by looking at Table 6.10. Even though the improvement is not as huge as on the decoy MNIST dataset, we can conclude that model correction works and positively affects the model's classification abilities.

The overall results indicate that model correction has a positive effect on a model's classification abilities. Only a few explanation feedback is needed during fine-tuning to improve a model's classification abilities even with a random sampling strategy. The improvement does not generalize well to the Dog vs. Cat dataset, as that dataset has no apparent spurious features like the other datasets.

On the downside, we have no good quantification of the actual change in explanations, except for the explanation overlap metrics found in Tables 6.4, 6.7 and 6.10. Those numerical values are lacking since we have no comparison with other model correction methods. Without comparison with other methods, we have no intuition about what constitutes good values. Moreover, our experimental results lack a human-centric evaluation. An explanation may serve different

purposes, such as raising trust in a model or assisting a user in decision making [Chander et al., 2018]. Furthermore, explanations may have different stakeholders and not just ML researchers and domain experts [Ras et al., 2018]. How "good" an explanation also depends on factors such as user satisfaction and trust assessment [Miller, 2019]. Hence, the evaluation should include not only algorithmic validation but also human-centric validation, which is lacking.

7.2 Contributions

We propose a pipeline and a Bayesian framework using explanation feedback. The pipeline makes it possible for users to interact with a model after training to create explanation feedback. The users create feedback based on the model's explanations of training sample classifications. Our Bayesian framework uses this feedback to correct the model so that explanations and predictive performance improve, making the model more trustworthy and understandable. The contributions made can be summarized as:

1. A pipeline that provides feedback to a model after training (see Figure 4.1). This avoids imposing restrictions on the model during the training phase and results in knowledge transfer being interactive and model specific.
2. A Bayesian framework utilizing explanation feedback to correct a model. The application of Bayesian inference results in a mathematically grounded objective function.
3. Experimental results, on one toy dataset and two real-world datasets, that demonstrate the method's effectiveness. This indicates that only a few annotated explanations and fine-tuning epochs are required to improve explanations justifying the classifications and the predictive performance.

7.3 Future Work

We see several opportunities for future works. This section describes those possibilities:

Extending beyond CNN architectures. Model correction methods require access to the loss gradient. However, we do not know the analytic form of the gradient's distribution. Hence, we cannot use the gradient itself as a target for our analysis. As a result, we proposed a new model correction method that exploits locality information in feature maps in a CNN. An exciting and unexplored direction is to extend our methods to other NN architectures.

Richer feedback semantics. Rejecting attribution regions is the only supported feedback semantic for our model correction method. However, a user might want the model to concentrate on specific regions with low attribution. Alternative feedback semantics have been explored [Selvaraju et al., 2019]. However, this is an emerging field, and much has yet to be researched. We see this as an essential future direction in order for humans to communicate with NNs.

Beyond image data. In this thesis, the focus of model correction has been on image data. Nevertheless, model correction can also be helpful in other research fields, such as NLP. Model correction methods can be applied to fix models that are, for example, gender-biased. Therefore, a possible future direction for this work is textual data. There are already some works on model correction for textual data that can be of interest to those who want to explore this path [Liu and Avci, 2019; Selvaraju et al., 2020b; Lertvittayakumjorn et al., 2020].

Sampling strategies. In this thesis, we used random sampling to find samples to query for annotation. It is known that random sampling gives the worst-case performance, and other clever strategies can be used [Settles, 2012]. Our method makes model correction compatible with Bayesian methods. Thus, we can explore interesting Bayesian active learning strategies [Kirsch et al., 2019; Gal et al., 2017; Tran et al., 2019] to select samples for annotation.

Appendices

1 Mathematical notation

The following notation system is adapted from Goodfellow et al. [2016].

Numbers and Arrays

a	A scalar (integer or real)
$\boldsymbol{a}$	A vector
$\boldsymbol{A}$	A matrix
$\mathbf{A}$	A tensor
$\boldsymbol{I}_n$	Identity matrix with n rows and n columns
$\text{diag}(\boldsymbol{a})$	A square, diagonal matrix with diagonal entries given by $\boldsymbol{a}$
a	A scalar random variable
$\mathbf{a}$	A vector-valued random variable
$\mathbf{A}$	A matrix-valued random variable

Sets and Graphs

$\mathbb{A}$	A set
$\mathbb{R}$	The set of real numbers
$\{0, 1\}$	The set containing 0 and 1
$\{0, 1, \dots, n\}$	The set of all integers between 0 and n
$[a, b]$	The real interval including a and b
$(a, b]$	The real interval excluding a but including b
$\mathbb{A} \backslash \mathbb{B}$	Set subtraction, i.e., the set containing the elements of $\mathbb{A}$ that are not in $\mathbb{B}$

Indexing

a_i	Element i of vector $\boldsymbol{a}$, with indexing starting at 1
a_{-i}	All elements of vector $\boldsymbol{a}$ except for element i
$A_{i,j}$	Element i, j of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{i,:}$	Row i of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{:,i}$	Column i of matrix $\boldsymbol{A}$
$\mathsf{A}_{i,j,k}$	Element (i, j, k) of a 3-D tensor $\mathsf{\mathbf{A}}$
$\mathsf{\mathbf{A}}_{:,:,i}$	2-D slice of a 3-D tensor
a_i	Element i of the random vector $\mathbf{a}$

Linear Algebra Operations

$\boldsymbol{A} \odot \boldsymbol{B}$	Element-wise (Hadamard) product of $\boldsymbol{A}$ and $\boldsymbol{B}$
$\boldsymbol{A} \otimes \boldsymbol{B}$	Broadcasting and element-wise (Hadamard) product of $\boldsymbol{A}$ and $\boldsymbol{B}$

Calculus

$\frac{dy}{dx}$	Derivative of y with respect to x
$\frac{\partial y}{\partial x}$	Partial derivative of y with respect to x
$\nabla_{\boldsymbol{x}} y$	Gradient of y with respect to $\boldsymbol{x}$
$\nabla_{\boldsymbol{X}} y$	Matrix derivatives of y with respect to $\boldsymbol{X}$
$\nabla_{\mathbf{X}} y$	Tensor containing derivatives of y with respect to $\mathbf{X}$
$\int f(\boldsymbol{x}) d\boldsymbol{x}$	Definite integral over the entire domain of $\boldsymbol{x}$
$\int_{\mathbb{S}} f(\boldsymbol{x}) d\boldsymbol{x}$	Definite integral with respect to $\boldsymbol{x}$ over the set $\mathbb{S}$

Probability and Information Theory

$\mathrm{a} \perp \mathrm{b}$	The random variables a and b are independent
$\mathrm{a} \perp \mathrm{b} \mid \mathrm{c}$	They are conditionally independent given c
$P(\mathrm{a})$	A probability distribution
$\mathrm{a} \sim P$	Random variable a has distribution P
$\mathbb{E}_{\mathrm{x} \sim P}[f(x)]$ or $\mathbb{E} f(x)$	Expectation of $f(x)$ with respect to $P(\mathrm{x})$
$\mathrm{Var}(f(x))$	Variance of $f(x)$ under $P(\mathrm{x})$
$\mathrm{Cov}(f(x), g(x))$	Covariance of $f(x)$ and $g(x)$ under $P(\mathrm{x})$
$H(\mathrm{x})$	Shannon entropy of the random variable x
$D_{\mathrm{KL}}(P \Vert Q)$	Kullback-Leibler divergence of P and Q
$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$	Gaussian distribution over $\boldsymbol{x}$ with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$

Functions

$f : \mathbb{A} \to \mathbb{B}$	The function f with domain $\mathbb{A}$ and range $\mathbb{B}$
$f(\boldsymbol{x}; \boldsymbol{\theta})$	A function of $\boldsymbol{x}$ parametrized by $\boldsymbol{\theta}$. (Sometimes we write $f(\boldsymbol{x})$ and omit the argument $\boldsymbol{\theta}$ to lighten notation)
$\log x$	Natural logarithm of x
$\|\|\boldsymbol{x}\|\|_p$	L^p norm of $\boldsymbol{x}$
$\|\|\boldsymbol{x}\|\|$	L^2 norm of $\boldsymbol{x}$
x^+	Positive part of x, i.e., $\max(0, x)$
$\mathbf{1}_{\text{condition}}$	is 1 if the condition is true, 0 otherwise
$\boldsymbol{X}^{\text{abs}}$	Element-wise absolute value of $\boldsymbol{X}$

Sometimes we use a function f whose argument is a scalar but apply it to a vector, matrix, or tensor: $f(\boldsymbol{x})$, $f(\boldsymbol{X})$, or $f(\mathbf{X})$. This denotes the application of f to the array element-wise. For example, if $\mathbf{C} = \sigma(\mathbf{X})$, then $C_{i,j,k} = \sigma(X_{i,j,k})$ for all valid values of i, j and k.

Datasets and Distributions

p_{data}	The data generating distribution
$\hat{p}_{\text{data}}$	The empirical distribution defined by the training set
$\mathbb{X}$	A set of training examples
$\boldsymbol{x}^{(i)}$	The i-th example (input) from a dataset
$y^{(i)}$ or $\boldsymbol{y}^{(i)}$	The target associated with $\boldsymbol{x}^{(i)}$ for supervised learning
$\boldsymbol{X}$	The $m \times n$ matrix with input example $\boldsymbol{x}^{(i)}$ in row $\boldsymbol{X}_{i,:}$

2 Acronyms

AD automatic differentiation. 5, 59–62

AF active feedback. 66–68

AI artificial intelligence. 1, 3, 62

AL active learning. ix, 4, 25, 26, 66

BBB Bayes by Backprop. 55, 78, 82

CAM class activation mapping. 43

CAVI coordinate ascent variational inference. 54

CE cross entropy. ix, 14, 15, 46

CNN convolutional neural network. i, 3, 4, 27–29, 31, 39, 43, 44, 69, 78, 80, 81, 87, 89, 105

DeepLIFT deep learning important features. 93, 96, 100

DL deep learning. 1–3, 5, 37, 59–61, 67

ELBO evidence lower bound. 51, 54, 55, 78, 80, 82

F feedback. xiii, 99

FNN feedforward neural network. ix, 9–12, 27

GPU graphics processing unit. 59, 60

Grad-CAM gradient-weighted class activation mapping. x, xi, 43–45, 69, 70, 72, 81, 93, 96, 100

i.i.d. independent and identically distributed. 13, 47, 75, 78, 82

ILSVRC ImageNet Large Scale Visual Recognition Challenge. 89, 90

ISIC International Skin Imaging Collaboration. i, xi, 85, 88, 89, 92, 104

KL Kullback–Leibler. 48–50, 57, 78, 80

LRN local response normalization. 90

LRT local reparameterization trick. 56, 78–80, 82, 83

MC Monte Carlo. 57, 78, 82, 83

MCMC Markov Chain Monte Carlo. 78

ML machine learning. x, 1–5, 7, 8, 18, 19, 32–34, 36–39, 62, 63, 65–67, 78, 105

MNIST Modified National Institute of Standards and Technology. i, xi, 85–87, 92, 94, 103, 104

MSE mean squared error. 13, 14, 22, 34

NF no feedback. xiii, 99

NLP natural language processing. 27, 106

NN neural network. i, 3–5, 7–13, 16, 17, 19–24, 28, 37, 39, 40, 42, 45, 55, 59–62, 65, 66, 71, 77, 80, 85, 86, 89, 105, 106

OOD out-of-distribution. i, 77

ReLU rectified linear unit. 10, 11, 44, 70, 81, 90

SGD stochastic gradient descent. 15, 16

SLIC simple linear iterative clustering. xi, 72–74

VI variational inference. 5, 78, 80

XAI explainable artificial intelligence. 3, 5, 32, 33, 35, 63

3 Article

CORRECTING CLASSIFICATION: A BAYESIAN FRAMEWORK USING EXPLANATION FEEDBACK TO IMPROVE CLASSIFICATION ABILITIES

Yanzhe Bekkemoen
Department of Computer Science
Norwegian University of Science and Technology
Trondheim, Norway

Helge Langseth
Department of Computer Science
Norwegian University of Science and Technology
Trondheim, Norway

ABSTRACT

Neural networks (NNs) have shown high predictive performance, however, with shortcomings. Firstly, the reasons behind the classifications are not fully understood. Several explanation methods have been developed, but they do not provide mechanisms for users to interact with the explanations. Explanations are social, meaning they are a transfer of knowledge through interactions. Nonetheless, current explanation methods contribute only to one-way communication. Secondly, NNs tend to be overconfident, providing unreasonable uncertainty estimates on out-of-distribution observations. We overcome these difficulties by training a Bayesian convolutional neural network (CNN) that uses explanation feedback. After training, the model presents explanations of training sample classifications to an annotator. Based on the provided information, the annotator can accept or reject the explanations by providing feedback. Our proposed method utilizes this feedback for fine-tuning to correct the model such that the explanations and classifications improve. We use existing CNN architectures to demonstrate the method's effectiveness on one toy dataset (decoy MNIST) and two real-world datasets (Dogs vs. Cats and ISIC skin cancer). The experiments indicate that few annotated explanations and fine-tuning epochs are needed to improve the model and predictive performance, making the model more trustworthy and understandable.

Keywords Explainable/Interpretable Machine Learning · Bayesian Inference · Deep Learning

1 Introduction

During the 20th century, a horse named Clever Hans was claimed to have performed arithmetic and other intellectual tasks. However, it was later revealed that the horse responded to the trainer's involuntary body language cues. The trainer was unaware that he provided the horse with hints causing it to display intelligent behavior. This behavior is today referred to as the Clever Hans effect [Pfungst, 1911].

NNs have displayed high predictive performance in many application areas in recent years, but they can focus on irrelevant features, thereby displaying the Clever Hans effect. We define irrelevant features as features not encoding the data's true underlying relationship. To make NNs and other machine learning (ML) methods interpretable and detect such behavior, several methods for explaining classifications have been developed, specifically for NNs [Zeiler and Fergus, 2013, Simonyan et al., 2014, Selvaraju et al., 2020, Shrikumar et al., 2019] and other model-agnostic methods [Ribeiro et al., 2016]. However, these explanation methods do not offer a way to act on the explanations and remove the Clever Hans effect. As a result, new research has started to explore the possibility of leveraging explanations to remove or prevent the effect [Ross et al., 2017, Erion et al., 2020, Rieger et al., 2020, Teso and Kersting, 2019, Selvaraju et al., 2019]. We will refer to these methods as model correction methods.

Explanations are social, meaning they are a transfer of knowledge through interactions or conversations [Miller, 2019]. Many model correction methods work as one-way communication where annotations of irrelevant features need to be provided before training. However, defining such knowledge about the data before training does not satisfy the

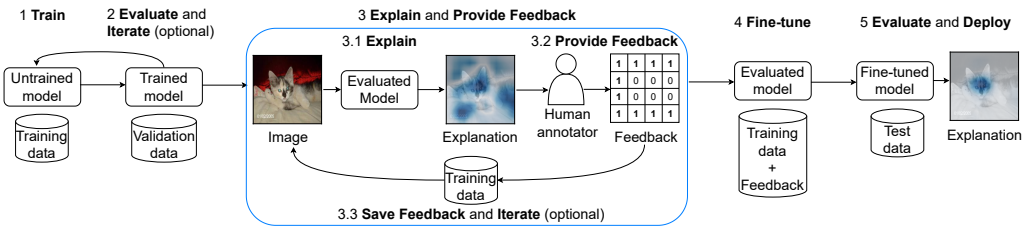


Figure 1: A "standard" ML pipeline with steps for annotating explanations and correcting a model. During Step 3, a model explains training sample classifications to a human annotator who gives feedback on those explanations. A feedback $\boldsymbol{F}^{(i)}$ for a sample i is a matrix of the same width and height as the image. If a feature k, j is irrelevant $\boldsymbol{F}^{(i)}_{k,j} = 1$, otherwise $\boldsymbol{F}^{(i)}_{k,j} = 0$. In Step 4, a model is fine-tuned with training data and feedback. The goal is to improve the reasons behind the classifications (explanation in Step 3 vs. Step 5) and predictive performance.

explanations' social aspect. Additionally, defining such knowledge without considering what the model will learn through training can be challenging, making model correction methods hard to use in practice.

Humans' trust in models is affected by a model's confidence in its predictive performance [Zhang et al., 2020]. NNs are bad at quantifying uncertainty and tend to produce overconfident classifications [Lakshminarayanan et al., 2017]. An overconfident model can be dangerous or offensive [Amodei et al., 2016]. The model can be seen as trying to gain false trust that will backlash against the model if it makes a wrong decision. Consequently, being overconfident can reduce trust in the model rather than increase it. Providing reasonable explanations induce trust [Ribeiro et al., 2016], but explanations generated by these aforementioned explanation methods only capture local behaviors, meaning an explanation applies only in the vicinity of a sample. This does not give insight into how a model will perform on out-of-distribution samples. Hence, to resolve model overconfidence, the model must capture aleatoric and epistemic uncertainty [Lakshminarayanan et al., 2017]. There are several ways to capture epistemic uncertainty; one way is to frame the NNs within a Bayesian framework that applies inference on the model's weights [MacKay, 1992, Blundell et al., 2015].

Model correction and Bayesian inference contribute to robustness and trust in the model. However, they are difficult to use together because most model correction methods modify the objective function. The modifications do not necessarily follow probability calculus and can introduce elements with unknown distributions.

For a model to have the correct explanation, classification and model confidence, this work bridges the gap between model correction and Bayesian inference. Additionally, it takes a step towards making explanations in ML social. We present a Bayesian framework that uses explanation feedback. After training, a Bayesian CNN presents explanations of training sample classifications to a human annotator. The annotator can accept or reject the explanations by giving feedback as additional evidence. Feedback is represented as a matrix with the same width and height as a sample specifying which attribution regions to reject. This feedback is used during fine-tuning to correct the model such that the explanations and predictive performance improve.

Our main contributions are: (1) A pipeline that provides feedback to a model after training (see Figure 1). This avoids imposing restrictions on the model during the training phase and results in knowledge transfer being interactive and model specific. (2) A Bayesian framework utilizing explanation feedback to correct a model, resulting in a mathematically grounded objective function from a probabilistic view. (3) Experimental results, on one toy dataset and two real-world datasets, that demonstrate the method's effectiveness. This indicates that few annotated explanations and fine-tuning epochs are required to improve explanations justifying the classifications and the predictive performance.

2 Related Works

Explanation Methods. Several explanation methods have been developed during the past years. These explanation methods can be divided into model-specific vs. model-agnostic and global vs. local scope [Lipton, 2017]. This work builds upon methods that create model-specific local explanations for NNs [Zeiler and Fergus, 2013, Simonyan et al., 2014, Selvaraju et al., 2020, Shrikumar et al., 2019, Sundararajan et al., 2017]. These model-specific local explanation methods produce importance scores assigned to individual features, e.g., pixels for image data or words for textual data. We will refer to the importance scores as feature attributions or attributions for short.

Model Correction Methods. The first work in this area is "Right for the Right Reasons" (RRR) [Ross et al., 2017] that regularize input gradients to suppress irrelevant features in the input space. Similarly, Erion et al. [2020] use Expected Gradients (a modified version of Integrated Gradients [Sundararajan et al., 2017]) to regularize explanations. Used in a similar fashion as RRR, contextual decomposition explanation penalization [Rieger et al., 2020] can regularize feature importance and feature interaction. To encourage NNs to focus on the same input regions as humans, Human Importance-aware Network Tuning (HINT) [Selvaraju et al., 2019] uses gradients in the last convolutional layer (forward direction). HINT encourages NNs to focus on certain regions instead of suppressing attributions. Similar to active learning, but for explanations, Teso and Kersting [2019] proposed using data augmentation to actively correct a model.

Bayesian Inference. Consider a probabilistic model $P(\boldsymbol{e}|\boldsymbol{w})$, the prior distribution $P(\boldsymbol{w})$ that encodes the prior knowledge about the parameter $\boldsymbol{w}$ and the evidence

$$\boldsymbol{e} = \{(\mathbf{X}^{(i)}, \boldsymbol{y}^{(i)})\}_{i=1}^{n} \tag{1}$$

with n independent and identically distributed (i.i.d.) samples. We want to compute the posterior distribution

$$P(\boldsymbol{w}|\boldsymbol{e}) = \frac{P(\boldsymbol{e}|\boldsymbol{w})P(\boldsymbol{w})}{P(\boldsymbol{e})} \tag{2}$$

using Bayes rule. However, exact Bayesian inference is intractable, so Markov Chain Monte Carlo (MCMC) algorithms or variational inference (VI) are used to approximate the weights' posterior distribution. Although VI is more scalable than MCMC methods, MCMC methods have an asymptotic guarantee that VI methods do not have. Bayes by Backprop (BBB) [Blundell et al., 2015] is a VI and backpropagation-compatible method for learning weight distributions. BBB uses the reparametrization trick [Kingma and Welling, 2014] to calculate a Monte Carlo (MC) estimator of the evidence lower bound (ELBO) and uses Gaussian variational distributions. To compute the MC estimator of ELBO, samples are drawn from the weights, which is computationally expensive. To reduce the number of samples needed, the local reparameterization trick (LRT) [Kingma et al., 2015] was proposed. LRT considers uncertainty at the activation level (before any nonlinear activation function) by computing the activations before drawing samples. This significantly reduces the number of samples drawn and makes it computationally cheaper.

Example 1 (LRT for a fully connected layer) *Let layer l be a fully connected layer with weights $\boldsymbol{W}^{m\times r}$ where $q_{\boldsymbol{\lambda}}(W_{i,j}) = \mathcal{N}(\mu_{W_{i,j}}, \sigma^2_{W_{i,j}})\ \forall W_{i,j} \in \boldsymbol{W}$ and biases $\boldsymbol{B}^{t\times r}$ where $q_{\boldsymbol{\lambda}}(B_{i,j}) = \mathcal{N}(\mu_{B_{i,j}}, \sigma^2_{B_{i,j}})\ \forall B_{i,j} \in \boldsymbol{B}$. Given an input $\boldsymbol{Z}^{t\times m}$, the activation of layer l is $\boldsymbol{U} = \boldsymbol{Z}\boldsymbol{W} + \boldsymbol{B}$ of size $t \times r$ where*

$$q_{\boldsymbol{\lambda}}(U_{i,j}|\boldsymbol{Z}) = \mathcal{N}(\gamma_{i,j}, \delta_{i,j})\ \forall U_{i,j} \in \boldsymbol{U} \tag{3}$$

$$\gamma_{i,j} = \sum_{k=1}^{m} Z_{i,k}\mu_{W_{k,j}} + \mu_{B_{i,j}},\ \delta_{i,j} = \sum_{k=1}^{m} Z^2_{i,k}\sigma^2_{W_{k,j}} + \sigma^2_{B_{i,j}}. \tag{4}$$

A new weight matrix is sampled for every training sample to reduce estimator variance, which has a computational complexity of $\mathcal{O}(tmr)$. By sampling activations rather than weights, the computational complexity is reduced to $\mathcal{O}(tr)$.

Example 2 (LRT for a convolutional layer) *Let layer l be a convolutional layer with a convolutional filter $\mathbf{W}^{w'\times h'\times \bar{c}}$ where $q_{\boldsymbol{\lambda}}(\mathrm{vec}(\mathbf{W})) = \mathcal{N}(\mathrm{vec}(\mathbf{M}), \mathrm{diag}(\mathrm{vec}(\mathbf{V}^2)))$ is a fully factorized Gaussian. Given an input $\mathbf{Z}^{\bar{w}\times \bar{h}\times \bar{c}}$, the activation of layer l is $\boldsymbol{U} = \mathbf{Z} * \mathbf{W}$ of size $\hat{w} \times \hat{h}$ where*

$$q_{\boldsymbol{\lambda}}(\mathrm{vec}(\boldsymbol{U})|\mathbf{Z}) = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{U}}, \boldsymbol{\sigma}^2_{\boldsymbol{U}}) \tag{5}$$

$$\boldsymbol{\mu}_{\boldsymbol{U}} = \mathrm{vec}(\mathbf{Z} * \mathbf{M}),\ \boldsymbol{\sigma}^2_{\boldsymbol{U}} = \mathrm{diag}(\mathrm{vec}(\mathbf{Z}^2 * \mathbf{V}^2)) \tag{6}$$

is treated as a fully factorized Gaussian. We define $\mathrm{vec}(\cdot)$ as a vectorization function, $(\cdot)^2$ as an element-wise operator, $*$ as a convolution operator and $\mathrm{diag}(\cdot)$ as a diagonal matrix.

3 Variational Inference and Explanation Feedback

Section 3.1 outlines the pipeline to construct feedback and correct a model (Step 3 and 4 in Figure 1). Sections 3.2 and 3.3 outline the objective function used in the fine-tuning phase and how feedback is induced into a model by adding additional evidence to $\boldsymbol{e}$.

3.1 Feedback Pipeline

There might be prior knowledge available about a dataset, e.g., digits are centered in the Modified National Institute of Standards and Technology (MNIST) [LeCun et al., 1989] dataset. In this case, we can make a model focus on the center and ignore the edges during training. However, this kind of knowledge might not be needed and can unnecessarily regularize the model. Moreover, constructing such knowledge without knowing what the model will learn through training can be difficult. If irrelevant features were known before training, training data could be preprocessed instead of introducing a new technique. We propose a pipeline where, unlike most previous work, the model asks for feedback after training. By asking for feedback after training, we can define knowledge specifically targeted at the model.

The pipeline adds two additional steps to a "standard" ML pipeline (Step 3 and 4 in Figure 1). During Step 3, a human annotator provides explanation feedback on training sample classifications. For explanation-generation, any saliency-based methods can be used, e.g., Gradient-weighted Class Activation Mapping (Grad-CAM) [Selvaraju et al., 2020]. To determine samples for annotation, active learning sampling strategies can be used. However, it is not obvious how to measure explanations' informativeness, so we leave this issue for future work. During the experiments, samples are annotated uniformly at random because the goal of our analysis is a proof of concept and not a fully specified pipeline.

As for Step 4, a model is fine-tuned using feedback and training data. The evidence and the objective function used for fine-tuning will be described in the coming sections.

3.2 Add Explanation Feedback to Evidence

We want to compute the posterior distribution $P(\boldsymbol{w}|\boldsymbol{e})$ of the weights of a NN. The evaluation involves computation of intractable integrals. Therefore, approximation techniques are used. VI is one of the most widely used and defines a variational distribution $q_{\boldsymbol{\lambda}}(\boldsymbol{w})$ that is used to approximate the posterior distribution by minimizing the Kullback–Leibler (KL) divergence $D_{\mathrm{KL}}(q_{\boldsymbol{\lambda}}(\boldsymbol{w})\|P(\boldsymbol{w}|\boldsymbol{e}))$. Minimizing the KL divergence is equivalent to maximizing the ELBO

$$\mathcal{L}_{\mathrm{ELBO}} = \underbrace{\mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{w})}[\log P(\boldsymbol{e}|\boldsymbol{w})]}_{\text{Likelihood}} - \underbrace{D_{\mathrm{KL}}(q_{\boldsymbol{\lambda}}(\boldsymbol{w})\|P(\boldsymbol{w}))}_{\text{Complexity}}. \tag{7}$$

Let $\mathbf{X}^{(i)^{w\times h\times c}}$ be an observation with the label $\boldsymbol{y}^{(i)}$, a feedback of the sample i is a matrix $\boldsymbol{F}^{(i)} \in \{0,1\}^{w\times h}$ given by a human annotator. If a feature k, j is irrelevant $\boldsymbol{F}^{(i)}_{k,j} = 1$, otherwise $\boldsymbol{F}^{(i)}_{k,j} = 0$. Furthermore, let f be a Bayesian CNN that takes $\mathbf{X}^{(i)}$ as input and outputs the logit vector $f(\mathbf{X}^{(i)}) = \hat{\boldsymbol{y}}^{(i)}$.

Consider input gradients $\nabla_{\mathbf{X}^{(i)}} \sum_j \hat{y}^{(i)}_j$ of the observation $\mathbf{X}^{(i)}$ that can be used as an explanation that specifies feature attributions.

$$\mathbf{H}^{(i)} = \boldsymbol{F}^{(i)} \otimes (\nabla_{\mathbf{X}^{(i)}} \sum_j \hat{y}^{(i)}_j) \tag{8}$$

is the attributions on irrelevant features that we want to minimize. $\otimes$ signifies broadcasting and an element-wise product. To include explanation feedback, we add additional evidence to $\boldsymbol{e}$

$$\boldsymbol{e}_{\text{grad}} = \boldsymbol{e} \cup \{\mathbf{H}^{(i)} = \mathbf{0}\}_{i=1}^{n} \tag{9}$$

where we set $\mathbf{H}^{(i)} = \mathbf{0}$ to constrain attributions on irrelevant features, which is similar to previous work [Ross et al., 2017].

However, the distribution of the input gradients is unknown and it is hard to find analytically. Not knowing the distribution makes us unable to calculate the likelihood in Equation (7) with $\boldsymbol{e}_{\text{grad}}$ as evidence. One way to solve this issue is to look at some quantity with known distribution instead of the input gradients. Although weight distributions are known, it is not trivial to map feedback given in the input space to the parameter space. Therefore, we use the LRT to consider the activation distributions and map the feedback to activation space. Examples 1 and 2 describe how to obtain activation distributions.

In the next section, we show how considering the uncertainty at the activation level enables us to approximate the evidence $\boldsymbol{e}_{\text{grad}}$ with an alternative formulation for which the likelihood can be expressed analytically.

We use CNNs since the activations in convolutional layers retain spatial information. The spatial information is needed to map feedback defined in the input space to corresponding activations. An additional advantage of using activations rather than input gradients is reduced computational complexity since second-order partial derivatives are not needed.

3.3 Methodology

This section details how to add feedback to $\boldsymbol{e}$ using activations rather than input gradients. We do this by describing how to map the feedback from input to activation space. Moreover, we describe how to compute Equation (7) with the new evidence $\boldsymbol{e}_{\text{act}}$ (to be defined in Equation (13)); something we could not do with $\boldsymbol{e}_{\text{grad}}$.

Consider the same setup as in Section 3.2. Let $\mathbf{A}^{(i)} \in \mathbb{R}^{u \times v \times d}$ $(u \leq w,\ v \leq h)$ be feature maps (before any nonlinear activation function) of the last convolutional layer (forward direction) obtained by forward passing $\mathbf{X}^{(i)}$. The last convolutional layer is used since it contains the highest abstraction level of features [Zeiler and Fergus, 2013, Zhou et al., 2015, Selvaraju et al., 2019].

3.3.1 Map Feedback from Input to Activation Space

The feedback will be mapped to the activation space to include it in the evidence $\boldsymbol{e}$. We do this in two steps: (1) downsample the feedback to the width and height of the feature maps $\mathbf{A}^{(i)}$ and (2) find activations in $\mathbf{A}^{(i)}$ spatially overlapping with the feedback and affecting the classification.

We downsample the feedback $\boldsymbol{F}^{(i)}$ to the width and height of $\mathbf{A}^{(i)}$ by computing

$$\boldsymbol{J}^{(i)} = \text{AdaptiveMaxPool}(\boldsymbol{F}^{(i)}, (u, v)). \tag{10}$$

The function AdaptiveMaxPool$(\boldsymbol{Z}, \boldsymbol{o})$ takes a matrix $\boldsymbol{Z}$ and uses max-pooling to transform it to size $\boldsymbol{o} = (o_1, o_2)$. This operation is similar to how Grad-CAM interpolates attribution maps to the input space to visualize explanations. Next, we find entries in $\boldsymbol{J}^{(i)}$ that overlap with $\mathbf{A}^{(i)}$ by calculating

$$\mathbf{G}^{(i)} = \mathbf{A}^{(i)} \otimes \boldsymbol{J}^{(i)}. \tag{11}$$

CNNs often use rectified linear unit and max-pooling, implying that classifications are only affected by a subset of the activations. Therefore, activations not used in the classification will be excluded by computing

$$\mathbf{L}^{(i)} = \mathbf{G}^{(i)} \odot \mathbf{1}_{(\nabla_{\mathbf{A}^{(i)}} \sum_j \hat{\mathrm{y}}_j^{(i)}) \neq 0} \tag{12}$$

to zero out entries in $\mathbf{G}^{(i)}$ that do not affect the classification. $\mathbf{L}^{(i)}$ is a tensor representing the activations that affect the classification and overlap with all indices k, j where $\boldsymbol{J}_{k,j}^{(i)} = 1$. The goal is to find the random variables where values in $\mathbf{L}^{(i)} \neq 0$ are drawn from. We will refer to those activations as $\boldsymbol{a}_{\mathcal{F}}^{(i)}$. The remaining activations in the network will be denoted $\boldsymbol{a}_{\mathcal{F}^c}^{(i)}$. We substitute input gradients with activations and add feedback to $\boldsymbol{e}$ by defining

$$\boldsymbol{e}_{\text{act}} = \boldsymbol{e} \cup \{\mathbf{a}_{\mathcal{F}}^{(i)} = \mathbf{0}\}_{i=1}^{n} \tag{13}$$

The process of finding $\boldsymbol{a}_{\mathcal{F}}^{(i)}$incurs an extra forward pass.

3.3.2 Compute the ELBO with Additional Evidence

In this section, we show how to compute the ELBO with $\boldsymbol{e}_{\text{act}}$. We follow Kingma et al. [2015] and consider uncertainty at the level of activations. This implies that the expectation in Equation (7) will be computed with respect to $q_{\boldsymbol{\lambda}}(\boldsymbol{a}_{\mathcal{F}^c}^{(i)}, \boldsymbol{a}_{\mathcal{F}}^{(i)} | \mathbf{X}^{(i)})$ rather than $q_{\boldsymbol{\lambda}}(\boldsymbol{w})$. $q_{\boldsymbol{\lambda}}(\boldsymbol{a}_{\mathcal{F}^c}^{(i)} | \mathbf{X}^{(i)}, \mathbf{a}_{\mathcal{F}}^{(i)} = \mathbf{0})$ will be written as $q_{\boldsymbol{\lambda}}(\boldsymbol{a}_{\mathcal{F}^c}^{(i)})$ for short. $\mathcal{L}_{\text{ELBO}}(\mathbf{X}^{(1)}, ..., \mathbf{X}^{(n)}) = \sum_{i=1}^{n} \mathcal{L}_{\text{ELBO}}^{(i)}(\mathbf{X}^{(i)})$ since the samples are i.i.d. We can reformulate Equation (7) as a loss function using activations and the evidence $\boldsymbol{e}_{\text{act}}$ for a single sample i as

$$\mathcal{L}_{\text{LRT}}^{(i)} = \frac{1}{n} D_{\text{KL}}(q_{\boldsymbol{\lambda}}(\boldsymbol{w}) \| P(\boldsymbol{w})) - \mathbb{E}_{q_{\boldsymbol{\lambda}}(\boldsymbol{a}_{\mathcal{F}^c}^{(i)})}[\log P(\boldsymbol{e}_{\text{act}}^{(i)} | \boldsymbol{a}_{\mathcal{F}^c}^{(i)})]. \tag{14}$$

Expanding $\boldsymbol{e}_{\text{act}}^{(i)}$ in the log probability in Equation (14) gives

$$\begin{aligned} \log(\boldsymbol{e}_{\text{act}}^{(i)} | \boldsymbol{a}_{\mathcal{F}^c}^{(i)}) &= \log P(\boldsymbol{y}^{(i)}, \mathbf{a}_{\mathcal{F}}^{(i)} = \mathbf{0} | \mathbf{X}^{(i)}, \boldsymbol{a}_{\mathcal{F}^c}^{(i)}) \\ &= \log P(\boldsymbol{y}^{(i)} | \mathbf{a}_{\mathcal{F}}^{(i)} = \mathbf{0}, \boldsymbol{a}_{\mathcal{F}^c}^{(i)}) + \log P(\mathbf{a}_{\mathcal{F}}^{(i)} = \mathbf{0} | \mathbf{X}^{(i)}, \boldsymbol{a}_{\mathcal{F}^c}^{(i)}) \end{aligned} \tag{15}$$

where $\boldsymbol{a}_{\mathcal{F}^c}^{(i)}$ and $\boldsymbol{a}_{\mathcal{F}}^{(i)}$ are treated as fully factorized Gaussians. To compute Equation (14), we follow the result of BBB and approximate the negative log-likelihood by applying MC sampling. When both $q_{\boldsymbol{\lambda}}(\boldsymbol{w})$ and $P(\boldsymbol{w})$ are fully factorized Gaussians, the complexity term can be calculated in closed form [Kingma and Welling, 2014]. Equation (14) is an objective function that uses the evidence $\boldsymbol{e}_{\text{act}}$ to incorporate feedback from an annotator and is theoretically justified from a probabilistic perspective, making it mathematically grounded. Moreover, we do not need a hyperparameter that deals with feedback penalty; instead, the objective function trades-off between complexity and data.

Dataset	Precision		Recall		F1		Accuracy	
	NF	F	NF	F	NF	F	NF	F
Decoy MNIST	0.725	**0.970**	0.725	**0.970**	0.725	**0.970**	0.725	**0.970**
Dogs vs. Cats	0.918	**0.923**	0.857	**0.870**	0.887	**0.896**	0.886	**0.894**
Skin cancer	0.280	**0.320**	**0.904**	0.798	0.427	**0.457**	**0.815**	0.799
Skin cancer NP	0.289	**0.335**	**0.904**	0.798	0.437	**0.472**	0.702	**0.721**

Table 1: Performance metrics of the model trained with no feedback (NF) and feedback (F). For decoy MNIST, all of the metrics are calculated using micro average. The skin cancer dataset is tested with and without patch data (NP) and accuracy is computed with macro average recall, also known as balanced accuracy.

Example 3 (How to approximate Equation (14)) *Consider $\boldsymbol{w} \in \mathbb{R}^j$, where the prior $P(\boldsymbol{w}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and the approximate posterior $q_{\boldsymbol{\lambda}}(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{\mu}, diag(\boldsymbol{\sigma}^2))$ are both fully factorized Gaussians. Hence, Equation* (14) *can be approximated as*

$$\begin{aligned}\mathcal{L}_{LRT}^{(i)} \approx & \frac{1}{2n}\sum_{i=1}^{j}(\mu_i^2 + \sigma_i^2 - 1 - \log\sigma_i^2) \\ & - \frac{1}{m}\sum_{t=1}^{m}[\log P(\boldsymbol{y}^{(i)}|\mathbf{a}_{\mathcal{F}}^{(i,t)} = \mathbf{0}, \boldsymbol{a}_{\mathcal{F}^c}^{(i,t)}) + \log P(\mathbf{a}_{\mathcal{F}}^{(i,t)} = \mathbf{0}|\mathbf{X}^{(i)}, \boldsymbol{a}_{\mathcal{F}^c}^{(i,t)})].\end{aligned} \tag{16}$$

m is the number of MC samples.

To summarize, feedback defined in the input space by an annotator is mapped to the activations. The feedback is included by adding it as additional evidence to $\boldsymbol{e}$. During Step 2, the model is trained with $\boldsymbol{e}$ as the evidence and Equation (7) as the objective function. As for Step 4, to fine-tune the model, Equation (14) is used as the objective function with $\boldsymbol{e}_{\text{act}}$ as the evidence.

4 Experiments and Results

The goal of our experiments is to display the effect of our proposed pipeline and model correction method. We demonstrate our work on one toy dataset, decoy MNIST [Ross et al., 2017] and two real-world datasets, Dogs vs. Cats [Kaggle, 2014] and The International Skin Imaging Collaboration (ISIC) skin cancer [Codella et al., 2019].

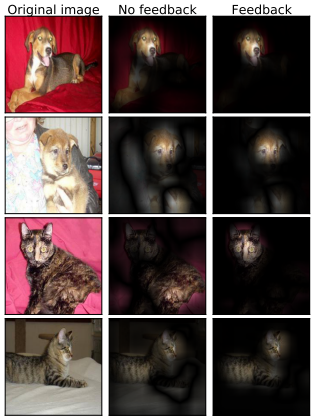
The LeNet [Lecun et al., 1998] architecture was used for decoy MNIST. LeNet models were trained for 100 epochs with early stopping. The AlexNet [Krizhevsky et al., 2012] architecture with batch normalization [Ioffe and Szegedy, 2015], but without dropout was used for Dogs vs. Cats and ISIC skin cancer. AlexNet models were trained for 1000 epochs with early stopping. Both LeNet and AlexNet models were saved at the best validation F1 score. All models were trained with a batch size of 256. The Adam optimizer [Kingma and Ba, 2017] with a learning rate of 0.001 and a prior $P(\boldsymbol{w}) = \mathcal{N}(\mathbf{0}, 0.1^2\boldsymbol{I})$ were used. Samples to annotate with feedback were chosen uniformly at random, as stated in Section 3.1.

To quantify attribution changes before and after feedback, attributions overlapping with irrelevant features were computed using four methods (see Table 2): Saliency [Simonyan et al., 2014], Deep Learning Important FeaTures (DeepLIFT) [Shrikumar et al., 2019], Grad-CAM and Occlusion [Zeiler and Fergus, 2013]. Consider an attribution mask $\mathbf{T}^{(i)^{w\times h\times v}}$ (v is the number of channels that depends on the method and the model) of sample i for one of the attribution methods. k is the number of samples in the test dataset with annotated explanation (not used in the fine-tuning process). The attribution overlap in Table 2 is computed as

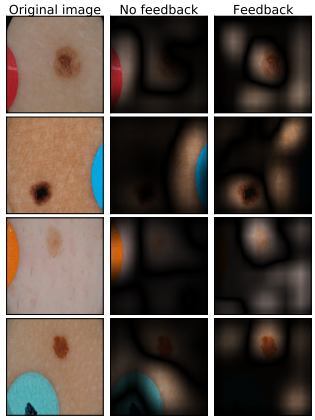
$$\frac{1}{k}\sum_{i=1}^{k}\frac{||\operatorname{vec}(\boldsymbol{F}^{(i)} \otimes \mathbf{T}^{(i)})||_1}{||\operatorname{vec}(\mathbf{T}^{(i)})||_1} \tag{17}$$

where $||\cdot||_1$ is the ℓ_1 norm operator. The effect of fine-tuning with feedback for a few samples from the test datasets (not used in the fine-tuning process) can be seen in Figure 2.

(a) Decoy MNIST. The model focuses on both the decoys and digits before feedback. After feedback, only the digits are used for classifications.



(b) Dogs vs. Cats. After feedback, the model becomes sharper and focuses more on the animals than the background.



(c) ISIC skin cancer. Before feedback, the model uses irrelevant patches and moles to classify samples. After feedback, it uses only moles on these samples.

Figure 2: Explanations before and after fine-tuning with feedback visualized on samples from test datasets. DeepLIFT was used for decoy MNIST and Grad-CAM for both Dogs vs. Cats and ISIC skin cancer to visualize explanations.

Dataset	Saliency		DeepLIFT		Grad-CAM		Occlusion	
	NF	F	NF	F	NF	F	NF	F
Decoy MNIST	0.080	**0.031**	0.078	**0.019**	0.063	**0.025**	0.181	**0.050**
Dogs vs. Cats	0.396	**0.384**	**0.405**	0.407	0.441	**0.379**	0.393	**0.380**
Skin cancer	0.154	**0.111**	0.145	**0.124**	0.221	**0.071**	0.250	**0.148**

Table 2: Attributions overlapping with irrelevant features averaged over all samples in the test dataset with annotated explanation, as described in Section 4. The attribution overlap score is bounded $[0, 1]$ and a **lower** score is better because it implies less attention is focused on irrelevant features. For Occlusion, a sliding window of size 3×3 was used for decoy MNIST and 23×23 for the other two datasets.

4.1 Decoy MNIST

The MNIST dataset consists of 28×28 grayscale images of digits. The training data has 60000 samples and the test data has 10000 samples. There are ten classes: $0, 1, ..., 9$. Decoy MNIST is a modified version of MNIST where every sample in the dataset has a 4×4 square in each corner (see Figure 2a). In the training data, the decoys' colors correspond with the label of the digit y, $(255 - 25y)$ and in the test data, the colors are randomly drawn. This decoy rule is a modified version of the one found in Ross et al. [2017], where only one corner has a decoy and is drawn randomly. The training data was divided into 90% (54000 samples) for training and 10% (6000 samples) for validation. The model was trained for 1 epoch without feedback and fine-tuned for 17 epochs. 0.4% (189 samples) of the training samples were annotated with feedback.

To observe how much of the original predictive performance is recovered through model correction, the model was trained with MNIST (7 epochs) and decoy MNIST. The model trained without decoys has 98.7% accuracy. After adding decoys, the accuracy drops down to 72.5%. By fine-tuning the model with feedback, the difference between the original accuracy is reduced to 1.7%. The decoy rule has no randomness, which makes it more difficult to recover the original accuracy. Most previous works have feedback on all training samples. In contrast, this approach has feedback on only 0.4% of the training data, making it more appealing.

Figure 2a and Table 2 show that without feedback, the model focuses on the decoys. After fine-tuning with feedback, the attributions on the decoys are significantly reduced.

4.2 Dogs vs. Cats

The Dogs vs. Cats dataset consists of RGB images of dogs (label 0) and cats (label 1) (see Figure 2b). All images in the dataset are scaled to 227×227. It has training data of 25000 samples and test data of 12500 samples, but the latter does not have labels. Thus, only the training data is used and divided as follows: 90% (22500 samples) training, 5% (2500 samples) validation and 5% (2500 samples) test. For the construction of feedback, a pretrained DeepLabV3 ResNet101 [Chen et al., 2017] semantic segmentation network was used. $\boldsymbol{F}^{(i)}_{k,j} = 1$ when index k, j does not overlap with an animal. This tells the model to not focus on background information. The model was trained for 765 epochs without feedback and fine-tuned for 16 epochs. 4.5% (1012 samples) of the training samples were annotated with feedback.

Dogs vs. Cats is the only dataset without any apparent irrelevant features repeating in several samples. Therefore, only a slight improvement in the model's predictive performance can be seen in Table 1. Furthermore, more feedback data is needed to affect the model. After fine-tuning with feedback, attributions overlap more with the animals (see Figure 2b and Table 2). Interestingly, explaining not to focus on background information results in the model focusing on the animals' faces instead of the whole body.

4.3 ISIC Skin Cancer

The version of the ISIC skin cancer dataset by Rieger et al. [2020] that we used has 21654 samples where 2284 are malignant (label 1) and 19370 are benign (label 0). Of those benign samples, 48% (9209 samples) have colorful patches (see Figure 2c). Feedback for samples in this dataset are regions that contain those colorful patches because they are irrelevant for the classifications. The dataset was divided into 90% (19488 samples) training, 5% (1083 samples) validation, and 5% (1083 samples) test. The model was trained for 997 epochs without feedback and fine-tuned for 1 epoch. 1.5% (292 samples) of the training samples were annotated with feedback.

The models were tested both with and without patches to observe the patches' effect. Table 1 shows that with patch data, the model without feedback has better accuracy than the model fine-tuned with feedback. However, without patch data, the model fine-tuned with feedback has better accuracy. Moreover, it is more trustworthy based on the explanations (Figure 2c).

Before feedback, the model uses patches to "cheat" on benign samples. When patches are removed, the model without feedback has the same number of false positives and true negatives, classifying benign samples randomly. In contrast, the model fine-tuned with feedback does not do that. Figure 2c demonstrates that the model with feedback focuses much less on the patches. Table 2 displays the same result quantitatively. Also, the positions of the patches appear not to matter.

5 Conclusion and Future Works

We propose a pipeline and a Bayesian framework using explanation feedback. The pipeline makes it possible for users to interact with a model after training to create explanation feedback (summarized in Figure 1). The users create feedback based on the model's explanations of training sample classifications. Our Bayesian framework uses this feedback to correct the model so that explanations and predictive performance improve, making the model more trustworthy and understandable. The effectiveness of the proposed approach is shown on one toy dataset and two real-world datasets. The results indicate that focus on dataset biases are minimized, and the features representing the data's true underlying relationship are targeted more distinctly. Thus, paying less attention to irrelevant features. Furthermore, few annotated explanations and fine-tuning epochs are needed to observe this effect.

We see several opportunities for future works, including: (1) sampling strategies for finding samples to ask for annotation, (2) extending beyond CNN architectures and (3) richer feedback semantics; currently, only rejection of attribution regions are supported.

References

O. Pfungst. *Clever Hans:(the horse of Mr. Von Osten.) a contribution to experimental animal and human psychology*. Holt, Rinehart and Winston, 1911.

M. D. Zeiler and R. Fergus. Visualizing and Understanding Convolutional Networks. *arXiv:1311.2901*, 2013. URL `http://arxiv.org/abs/1311.2901`.

K. Simonyan, A. Vedaldi, and A. Zisserman. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *arXiv:1312.6034*, 2014. URL `http://arxiv.org/abs/1312.6034`.

R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. *International Journal of Computer Vision*, 128(2):336–359, 2020. ISSN 0920-5691, 1573-1405. doi:10.1007/s11263-019-01228-7. URL http://arxiv.org/abs/1610.02391. arXiv: 1610.02391.

A. Shrikumar, P. Greenside, and A. Kundaje. Learning Important Features Through Propagating Activation Differences. *arXiv:1704.02685*, 2019. URL http://arxiv.org/abs/1704.02685.

M. T. Ribeiro, S. Singh, and C. Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. *arXiv:1602.04938*, 2016. URL http://arxiv.org/abs/1602.04938.

A. S. Ross, M. C. Hughes, and F. Doshi-Velez. Right for the Right Reasons: Training Differentiable Models by Constraining their Explanations. *arXiv:1703.03717*, 2017. URL http://arxiv.org/abs/1703.03717.

G. Erion, J. D. Janizek, P. Sturmfels, S. Lundberg, and S. Lee. Improving performance of deep learning models with axiomatic attribution priors and expected gradients. *arXiv:1906.10670*, 2020. URL http://arxiv.org/abs/1906.10670.

L. Rieger, C. Singh, W. J. Murdoch, and Bin Yu. Interpretations are useful: penalizing explanations to align neural networks with prior knowledge. *arXiv:1909.13584*, 2020. URL http://arxiv.org/abs/1909.13584.

S. Teso and K. Kersting. Explanatory Interactive Machine Learning. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, AIES '19, pages 239–245, New York, NY, USA, January 2019. Association for Computing Machinery. ISBN 978-1-4503-6324-2. doi:10.1145/3306618.3314293. URL https://doi.org/10.1145/3306618.3314293.

R. R. Selvaraju, S. Lee, Y. Shen, H. Jin, S. Ghosh, L. Heck, D. Batra, and D. Parikh. Taking a HINT: Leveraging Explanations to Make Vision and Language Models More Grounded. *arXiv:1902.03751*, 2019. URL http://arxiv.org/abs/1902.03751.

T. Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 267:1–38, February 2019. ISSN 0004-3702. doi:10.1016/j.artint.2018.07.007. URL http://www.sciencedirect.com/science/article/pii/S0004370218305988.

Y. Zhang, Q. V. Liao, and R. K. E. Bellamy. Effect of Confidence and Explanation on Accuracy and Trust Calibration in AI-Assisted Decision Making. *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, pages 295–305, 2020. doi:10.1145/3351095.3372852. URL http://arxiv.org/abs/2001.02114. arXiv: 2001.02114.

B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. *arXiv:1612.01474*, 2017. URL http://arxiv.org/abs/1612.01474.

D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané. Concrete Problems in AI Safety. *arXiv:1606.06565*, 2016. URL http://arxiv.org/abs/1606.06565.

D. J. C. MacKay. A Practical Bayesian Framework for Backpropagation Networks. *Neural Computation*, 4(3):448–472, May 1992. ISSN 0899-7667. doi:10.1162/neco.1992.4.3.448.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight Uncertainty in Neural Network. In *International Conference on Machine Learning*, pages 1613–1622. PMLR, June 2015. URL http://proceedings.mlr.press/v37/blundell15.html. ISSN: 1938-7228.

Z. C. Lipton. The Mythos of Model Interpretability. *arXiv:1606.03490*, 2017. URL http://arxiv.org/abs/1606.03490.

M. Sundararajan, A. Taly, and Q. Yan. Axiomatic Attribution for Deep Networks. *arXiv:1703.01365*, 2017. URL http://arxiv.org/abs/1703.01365.

D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. *arXiv:1312.6114*, 2014. URL http://arxiv.org/abs/1312.6114.

D. P. Kingma, T. Salimans, and M. Welling. Variational Dropout and the Local Reparameterization Trick. *arXiv:1506.02557*, 2015. URL http://arxiv.org/abs/1506.02557.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, December 1989. ISSN 0899-7667. doi:10.1162/neco.1989.1.4.541. URL https://doi.org/10.1162/neco.1989.1.4.541.

B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Object Detectors Emerge in Deep Scene CNNs. *arXiv:1412.6856*, 2015. URL http://arxiv.org/abs/1412.6856.

Kaggle. Dogs vs. Cats, January 2014. URL https://kaggle.com/c/dogs-vs-cats. Microsoft Research. https://kaggle.com/c/dogs-vs-cats (accessed 28/11/2020).

N. Codella, V. Rotemberg, P. Tschandl, M. E. Celebi, S. Dusza, D. Gutman, B. Helba, A. Kalloo, K. Liopyris, M. Marchetti, H. Kittler, and A. Halpern. Skin Lesion Analysis Toward Melanoma Detection 2018: A Challenge Hosted by the International Skin Imaging Collaboration (ISIC). *arXiv:1902.03368*, 2019. URL http://arxiv.org/abs/1902.03368.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998. ISSN 1558-2256. doi:10.1109/5.726791.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, 25:1097–1105, 2012. URL https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html.

S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167*, 2015. URL http://arxiv.org/abs/1502.03167.

D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980*, 2017. URL http://arxiv.org/abs/1412.6980.

L. Chen, G. Papandreou, F. Schroff, and H. Adam. Rethinking Atrous Convolution for Semantic Image Segmentation. *arXiv:1706.05587*, 2017. URL http://arxiv.org/abs/1706.05587.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2016). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv:1603.04467 [cs]*. arXiv: 1603.04467.

Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., and Süsstrunk, S. (2012). SLIC Superpixels Compared to State-of-the-Art Superpixel Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11):2274–2282. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

Adams, R. and Essex, C. (2013). *Calculus: A Complete Course*. Pearson.

Adebayo, J., Gilmer, J., Muelly, M., Goodfellow, I., Hardt, M., and Kim, B. (2020). Sanity Checks for Saliency Maps. *arXiv:1810.03292 [cs, stat]*. arXiv: 1810.03292.

Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723. Conference Name: IEEE Transactions on Automatic Control.

Altmann, A., Toloşi, L., Sander, O., and Lengauer, T. (2010). Permutation importance: a corrected feature importance measure. *Bioinformatics*, 26(10):1340–1347.

Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., and Mané, D. (2016). Concrete Problems in AI Safety. *arXiv:1606.06565*.

Ancona, M., Ceolini, E., Öztireli, C., and Gross, M. (2019). Gradient-Based Attribution Methods. In Samek, W., Montavon, G., Vedaldi, A., Hansen, L. K., and Müller, K.-R., editors, *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, Lecture Notes in Computer Science, pages 169–191. Springer International Publishing, Cham.

Antunes, P., Herskovic, V., Ochoa, S. F., and Pino, J. A. (2008). Structuring dimensions for collaborative systems evaluation. *ACM Computing Surveys*, 44(2):8:1–8:28.

Araujo, A., Norris, W., and Sim, J. (2019). Computing Receptive Fields of Convolutional Neural Networks. *Distill*, 4(11):e21.

Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.-R., and Samek, W. (2015). On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation. *PLOS ONE*, 10(7):e0130140. Publisher: Public Library of Science.

Bahdanau, D., Cho, K., and Bengio, Y. (2016). Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv:1409.0473 [cs, stat]*. arXiv: 1409.0473.

Bau, D., Zhu, J.-Y., Strobelt, H., Lapedriza, A., Zhou, B., and Torralba, A. (2020). Understanding the role of individual units in a deep neural network. *Proceedings of the National Academy of Sciences*, 117(48):30071–30078. Publisher: National Academy of Sciences Section: Colloquium on the Science of Deep Learning.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

Binns, R., Van Kleek, M., Veale, M., Lyngs, U., Zhao, J., and Shadbolt, N. (2018). 'It's Reducing a Human Being to a Percentage': Perceptions of Justice in Algorithmic Decisions. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pages 1–14, New York, NY, USA. Association for Computing Machinery.

Biran, O. and Cotton, C. (2017). Explanation and justification in machine learning: A survey. In *IJCAI-17 workshop on explainable AI (XAI)*, volume 8, pages 8–13. Issue: 1.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight Uncertainty in Neural Network. In *International Conference on Machine Learning*, pages 1613–1622. PMLR. ISSN: 1938-7228.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2):123–140.

Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1):5–32.

Breiman, L., Friedman, J., Stone, C., and Olshen, R. (1984). *Classification and Regression Trees*. Taylor & Francis.

Burns, K., Hendricks, L. A., Saenko, K., Darrell, T., and Rohrbach, A. (2019). Women also Snowboard: Overcoming Bias in Captioning Models. *arXiv:1803.09797 [cs]*. arXiv: 1803.09797.

Caruana, R., Kangarloo, H., Dionisio, J. D., Sinha, U., and Johnson, D. (1999). Case-based explanation of non-case-based learning methods. *Proceedings. AMIA Symposium*, pages 212–215.

Chander, A., Srinivasan, R., Chelian, S. E., Wang, J., and Uchino, K. (2018). Working with Beliefs: AI Transparency in the Enterprise. In *IUI Workshops*.

Chen, L., Papandreou, G., Schroff, F., and Adam, H. (2017). Rethinking Atrous Convolution for Semantic Image Segmentation. *arXiv:1706.05587*.

Chollet, F. (2018). *Deep learning with Python*, volume 361. Manning New York.

Chui, M. and Malhotra, S. (2018). AI adoption advances, but foundational barriers remain. *Mckinsey and Company*.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314.

Dabkowski, P. and Gal, Y. (2017). Real Time Image Saliency for Black Box Classifiers. *Advances in Neural Information Processing Systems*, 30.

Dawes, R. M. (1979). The robust beauty of improper linear models in decision making. *American Psychologist*, 34(7):571–582. Place: US Publisher: American Psychological Association.

Dietvorst, B. J., Simmons, J. P., and Massey, C. (2015). Algorithm aversion: people erroneously avoid algorithms after seeing them err. *Journal of Experimental Psychology. General*, 144(1):114–126.

dos Santos, C. and Gatti, M. (2014). Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 69–78, Dublin, Ireland. Dublin City University and Association for Computational Linguistics.

Doshi-Velez, F. and Kim, B. (2017). Towards A Rigorous Science of Interpretable Machine Learning. *arXiv:1702.08608 [cs, stat]*. arXiv: 1702.08608.

Drucker, H. and Cun, Y. L. (1992). Improving generalization performance using double backpropagation. *IEEE Transactions on Neural Networks*, 3(6):991–997. Conference Name: IEEE Transactions on Neural Networks.

Erion, G., Janizek, J. D., Sturmfels, P., Lundberg, S., and Lee, S.-I. (2020). Improving performance of deep learning models with axiomatic attribution priors and expected gradients. *arXiv:1906.10670 [cs, stat]*. arXiv: 1906.10670.

Falcon, W. and .al (2019). PyTorch Lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, 3.

Felzenszwalb, P. F. and Huttenlocher, D. P. (2004). Efficient Graph-Based Image Segmentation. *International Journal of Computer Vision*, 59(2):167–181.

Freund, Y. and Schapire, R. E. (1996). Experiments with a New Boosting Algorithm.

Friedman, J. H. and Popescu, B. E. (2008). Predictive learning via rule ensembles. *The Annals of Applied Statistics*, 2(3):916–954. Publisher: Institute of Mathematical Statistics.

Furman, J. and Seamans, R. (2018). AI and the Economy. *Innovation Policy and the Economy*, 19:161–191. Publisher: The University of Chicago Press.

Gal, Y., Islam, R., and Ghahramani, Z. (2017). Deep Bayesian Active Learning with Image Data. *arXiv:1703.02910 [cs, stat]*. arXiv: 1703.02910.

Gens, R. and Domingos, P. M. (2014). Deep Symmetry Networks. *Advances in Neural Information Processing Systems*, 27.

Ghorbani, A., Wexler, J., Zou, J. Y., and Kim, B. (2019). Towards Automatic Concept-based Explanations. *Advances in Neural Information Processing Systems*, 32.

Gilpin, L. H., Bau, D., Yuan, B. Z., Bajwa, A., Specter, M., and Kagal, L. (2019). Explaining Explanations: An Overview of Interpretability of Machine Learning. *arXiv:1806.00069 [cs, stat]*. arXiv: 1806.00069.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. (2016). *Deep learning*, volume 1. MIT press Cambridge.

Goodman, B. and Flaxman, S. (2017). European Union Regulations on Algorithmic Decision-Making and a "Right to Explanation". *AI Magazine*, 38(3):50–57. Number: 3.

Gunning, D. and Aha, D. (2019). DARPA's Explainable Artificial Intelligence (XAI) Program. *AI Magazine*, 40(2):44–58. Number: 2.

Guzella, T. S. and Caminhas, W. M. (2009). A review of machine learning approaches to Spam filtering. *Expert Systems with Applications*, 36(7):10206–10222.

Hand, D. J. and Henley, W. E. (1997). Statistical Classification Methods in Consumer Credit Scoring: A Review. *Journal of the Royal Statistical Society. Series A (Statistics in Society)*, 160(3):523–541. Publisher: [Wiley, Royal Statistical Society].

Hastie, T. J. and Tibshirani, R. J. (1990). *Generalized additive models*, volume 43. CRC press.

Hoerl, A. E. and Kennard, R. W. (1970). Ridge Regression: Biased Estimation for Nonorthogonal Problems. *Technometrics*, 12(1):55–67. Publisher: Taylor & Francis _eprint: https://www.tandfonline.com/doi/pdf/10.1080/00401706.1970.10488634.

Hoiem, D., Chodpathumwan, Y., and Dai, Q. (2012). Diagnosing Error in Object Detectors. In Fitzgibbon, A., Lazebnik, S., Perona, P., Sato, Y., and Schmid, C., editors, *Computer Vision – ECCV 2012*, Lecture Notes in Computer Science, pages 340–353, Berlin, Heidelberg. Springer.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257.

Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*, pages 448–456. PMLR. ISSN: 1938-7228.

Jaderberg, M., Simonyan, K., Zisserman, A., and Kavukcuoglu, K. (2015). Spatial Transformer Networks. *Advances in Neural Information Processing Systems*, 28.

Jordan, M. I. and Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260. Publisher: American Association for the Advancement of Science.

Kaggle (2014). Dogs vs. Cats. Microsoft Research. https://kaggle.com/c/dogs-vs-cats (accessed 28/11/2020).

Karpathy, A., Johnson, J., and Fei-Fei, L. (2015). Visualizing and Understanding Recurrent Networks. *arXiv:1506.02078 [cs]*. arXiv: 1506.02078.

Kim, B., Wattenberg, M., Gilmer, J., Cai, C., Wexler, J., Viegas, F., and Sayres, R. (2018). Interpretability Beyond Feature Attribution: Quantitative Testing with Concept Activation Vectors (TCAV). In *International Conference on Machine Learning*, pages 2668–2677. PMLR. ISSN: 2640-3498.

Kim, J. and Canny, J. (2017). Interpretable Learning for Self-Driving Cars by Visualizing Causal Attention. *arXiv:1703.10631 [cs]*. arXiv: 1703.10631.

Kingma, D. P. and Ba, J. (2017). Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*. arXiv: 1412.6980.

Kingma, D. P., Salimans, T., and Welling, M. (2015). Variational Dropout and the Local Reparameterization Trick. *arXiv:1506.02557*.

Kingma, D. P. and Welling, M. (2014). Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]*. arXiv: 1312.6114.

Kirsch, A., van Amersfoort, J., and Gal, Y. (2019). BatchBALD: Efficient and Diverse Batch Acquisition for Deep Bayesian Active Learning. *Advances in Neural Information Processing Systems*, 32.

Klambauer, G., Unterthiner, T., Mayr, A., and Hochreiter, S. (2017). Self-Normalizing Neural Networks. *arXiv:1706.02515 [cs, stat]*. arXiv: 1706.02515.

Kokhlikyan, N., Miglani, V., Martin, M., Wang, E., Alsallakh, B., Reynolds, J., Melnikov, A., Kliushkina, N., Araya, C., Yan, S., and Reblitz-Richardson, O. (2020). *Captum: A unified and generic model interpretability library for PyTorch*. _eprint: 2009.07896.

Koller, D. and Friedman, N. (2009). *Probabilistic graphical models: principles and techniques*. MIT press.

Kononenko, I. (2001). Machine learning for medical diagnosis: history, state of the art and perspective. *Artificial Intelligence in Medicine*, 23(1):89–109.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, 25.

Kullback, S. and Leibler, R. A. (1951). On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86. Publisher: Institute of Mathematical Statistics.

Kumbier, K., Basu, S., Brown, J. B., Celniker, S., and Yu, B. (2018). Refining interaction search through signed iterative Random Forests. *arXiv:1810.07287 [cs, stat]*. arXiv: 1810.07287.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017). Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. *arXiv:1612.01474*.

Le, H. and Borji, A. (2018). What are the Receptive, Effective Receptive, and Projective Fields of Neurons in Convolutional Neural Networks? *arXiv:1705.07049 [cs]*. arXiv: 1705.07049.

LeCun, Y. (2012). Learning Invariant Feature Hierarchies. In Fusiello, A., Murino, V., and Cucchiara, R., editors, *Computer Vision – ECCV 2012. Workshops and Demonstrations*, Lecture Notes in Computer Science, pages 496–505, Berlin, Heidelberg. Springer.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444. Number: 7553 Publisher: Nature Publishing Group.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551. Publisher: MIT Press.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324. Conference Name: Proceedings of the IEEE.

Lertvittayakumjorn, P., Specia, L., and Toni, F. (2020). FIND: Human-in-the-Loop Debugging Deep Text Classifiers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 332–348, Online. Association for Computational Linguistics.

Letham, B., Rudin, C., McCormick, T. H., and Madigan, D. (2015). Interpretable classifiers using rules and Bayesian analysis: Building a better stroke prediction model. *The Annals of Applied Statistics*, 9(3):1350–1371. Publisher: Institute of Mathematical Statistics.

Lipton, Z. C. (2018). The Mythos of Model Interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57.

Liu, F. and Avci, B. (2019). Incorporating Priors with Feature Attribution on Text Classification. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6274–6283, Florence, Italy. Association for Computational Linguistics.

Lloyd, S. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137. Conference Name: IEEE Transactions on Information Theory.

Lombrozo, T. (2006). The structure and function of explanations. *Trends in Cognitive Sciences*, 10(10):464–470. Publisher: Elsevier.

Long, J. L., Zhang, N., and Darrell, T. (2014). Do Convnets Learn Correspondence? *Advances in Neural Information Processing Systems*, 27.

Louizos, C. and Welling, M. (2017). Multiplicative Normalizing Flows for Variational Bayesian Neural Networks. In *International Conference on Machine Learning*, pages 2218–2227. PMLR. ISSN: 2640-3498.

Luo, W., Li, Y., Urtasun, R., and Zemel, R. (2017). Understanding the Effective Receptive Field in Deep Convolutional Neural Networks. *arXiv:1701.04128 [cs]*. arXiv: 1701.04128.

MacKay, D. J. C. (1992). A Practical Bayesian Framework for Backpropagation Networks. *Neural Computation*, 4(3):448–472.

Maddox, W., Garipov, T., Izmailov, P., Vetrov, D., and Wilson, A. G. (2019). A Simple Baseline for Bayesian Uncertainty in Deep Learning. *arXiv:1902.02476 [cs, stat]*. arXiv: 1902.02476.

Mahendran, A. and Vedaldi, A. (2016). Visualizing Deep Convolutional Neural Networks Using Natural Pre-Images. *arXiv:1512.02017 [cs]*. arXiv: 1512.02017.

Makridakis, S. (2017). The forthcoming Artificial Intelligence (AI) revolution: Its impact on society and firms. *Futures*, 90:46–60.

Malinin, A. and Gales, M. (2018). Predictive Uncertainty Estimation via Prior Networks. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Miller, T. (2019). Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 267:1–38.

Mitchell, T. (1997). *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill.

Mitsuhara, M., Fukui, H., Sakashita, Y., Ogata, T., Hirakawa, T., Yamashita, T., and Fujiyoshi, H. (2019). Embedding Human Knowledge into Deep Neural Network via Attention Map. *arXiv:1905.03540 [cs]*. arXiv: 1905.03540.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. *arXiv:1312.5602 [cs]*. arXiv: 1312.5602.

Molnar, C. (2020). *Interpretable machine learning*. Lulu. com.

Murdoch, W. J., Singh, C., Kumbier, K., Abbasi-Asl, R., and Yu, B. (2019). Definitions, methods, and applications in interpretable machine learning. *Proceedings of the National Academy of Sciences*, 116(44):22071–22080. Publisher: National Academy of Sciences Section: Physical Sciences.

Neal, R. M. (1996). *Bayesian Learning for Neural Networks*. Lecture Notes in Statistics. Springer-Verlag, New York.

Neubert, P. and Protzel, P. (2014). Compact Watershed and Preemptive SLIC: On Improving Trade-offs of Superpixel Segmentation Algorithms. In *2014 22nd International Conference on Pattern Recognition*, pages 996–1001. ISSN: 1051-4651.

Olah, C., Mordvintsev, A., and Schubert, L. (2017). Feature Visualization. *Distill*, 2(11):e7.

Olshausen, B. A. and Field, D. J. (1997). Sparse coding with an overcomplete basis set: A strategy employed by V1? *Vision Research*, 37(23):3311–3325.

Opitz, D. and Maclin, R. (1999). Popular Ensemble Methods: An Empirical Study. *Journal of Artificial Intelligence Research*, 11:169–198.

Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. (2016). Deep Exploration via Bootstrapped DQN. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *arXiv:1912.01703 [cs, stat]*. arXiv: 1912.01703.

Patel, K., Fogarty, J., Landay, J. A., and Harrison, B. (2008). Examining difficulties software developers encounter in the adoption of statistical machine learning. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 3*, AAAI'08, pages 1563–1566, Chicago, Illinois. AAAI Press.

Pearl, J. (1985). Bayesian netwcrks: A model cf self-activated memory for evidential reasoning. In *Proceedings of the 7th Conference of the Cognitive Science Society, University of California, Irvine, CA, USA*, pages 15–17.

Pfungst, O. (1911). *Clever Hans:(the horse of Mr. Von Osten.) a contribution to experimental animal and human psychology*. Holt, Rinehart and Winston.

Poursabzi-Sangdeh, F., Goldstein, D., Hofman, J., Vaughan, J. W., and Wallach, H. (2019). Manipulating and Measuring Model Interpretability.

Prechelt, L. (2012). Early Stopping — But When? In Montavon, G., Orr, G. B., and Müller, K.-R., editors, *Neural Networks: Tricks of the Trade: Second Edition*, Lecture Notes in Computer Science, pages 53–67. Springer, Berlin, Heidelberg.

Rall, L. B. (1981). *Automatic Differentiation: Techniques and Applications*. Lecture Notes in Computer Science. Springer-Verlag, Berlin Heidelberg.

Ras, G., van Gerven, M., and Haselager, P. (2018). Explanation Methods in Deep Learning: Users, Values, Concerns and Challenges. *arXiv:1803.07517 [cs, stat]*. arXiv: 1803.07517.

Ribeiro, M. T., Singh, S., and Guestrin, C. (2016a). Model-Agnostic Interpretability of Machine Learning. *arXiv:1606.05386 [cs, stat]*. arXiv: 1606.05386.

Ribeiro, M. T., Singh, S., and Guestrin, C. (2016b). "Why Should I Trust You?": Explaining the Predictions of Any Classifier. *arXiv:1602.04938 [cs, stat]*. arXiv: 1602.04938.

Rieger, L., Singh, C., Murdoch, W., and Yu, B. (2020). Interpretations are Useful: Penalizing Explanations to Align Neural Networks with Prior Knowledge. In *International Conference on Machine Learning*, pages 8116–8126. PMLR. ISSN: 2640-3498.

Robnik-Šikonja, M. and Bohanec, M. (2018). Perturbation-Based Explanations of Prediction Models. In Zhou, J. and Chen, F., editors, *Human and Machine Learning: Visible, Explainable, Trustworthy and Transparent*, Human–Computer Interaction Series, pages 159–175. Springer International Publishing, Cham.

Rosch, E. (2002). *Principles of categorization*. Foundations of cognitive psychology: Core readings. MIT Press, Cambridge, MA, US. Pages: 270.

Ross, A. S., Hughes, M. C., and Doshi-Velez, F. (2017). Right for the Right Reasons: Training Differentiable Models by Constraining their Explanations. *arXiv:1703.03717 [cs, stat]*. arXiv: 1703.03717.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536. Number: 6088 Publisher: Nature Publishing Group.

Rumelhart, D. E. and McClelland, J. L. (1987). Learning Internal Representations by Error Propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*, pages 318–362. MIT Press. Conference Name: Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252.

Scherer, D., Müller, A., and Behnke, S. (2010). Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition. In Diamantaras, K., Duch, W., and Iliadis, L. S., editors, *Artificial Neural Networks – ICANN 2010*, Lecture Notes in Computer Science, pages 92–101, Berlin, Heidelberg. Springer.

Schwarz, G. (1978). Estimating the Dimension of a Model. *The Annals of Statistics*, 6(2):461–464. Publisher: Institute of Mathematical Statistics.

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Batra, D. (2020a). Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. *International Journal of Computer Vision*, 128(2):336–359.

Selvaraju, R. R., Lee, S., Shen, Y., Jin, H., Ghosh, S., Heck, L., Batra, D., and Parikh, D. (2019). Taking a HINT: Leveraging Explanations to Make Vision and Language Models More Grounded. *arXiv:1902.03751*.

Selvaraju, R. R., Tendulkar, P., Parikh, D., Horvitz, E., Ribeiro, M. T., Nushi, B., and Kamar, E. (2020b). SQuINTing at VQA Models: Introspecting VQA Models With Sub-Questions. pages 10003–10011.

Settles, B. (2012). Active Learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–114.

Shorten, C. and Khoshgoftaar, T. M. (2019). A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, 6(1):60.

Shrikumar, A., Greenside, P., and Kundaje, A. (2019). Learning Important Features Through Propagating Activation Differences. *arXiv:1704.02685 [cs]*. arXiv: 1704.02685.

Shrikumar, A., Greenside, P., Shcherbina, A., and Kundaje, A. (2017). Not Just a Black Box: Learning Important Features Through Propagating Activation Differences. *arXiv:1605.01713 [cs]*. arXiv: 1605.01713.

Simonyan, K., Vedaldi, A., and Zisserman, A. (2014). Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *arXiv:1312.6034 [cs]*. arXiv: 1312.6034.

Simonyan, K. and Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556 [cs]*. arXiv: 1409.1556.

Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. (2015). Striving for Simplicity: The All Convolutional Net. *arXiv:1412.6806 [cs]*. arXiv: 1412.6806.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

Sundararajan, M., Taly, A., and Yan, Q. (2017). Axiomatic Attribution for Deep Networks. *arXiv:1703.01365 [cs]*. arXiv: 1703.01365.

Sundararajan, M., Xu, J., Taly, A., Sayres, R., and Najmi, A. (2019). Exploring Principled Visualizations for Deep Network Attributions. In *IUI Workshops*, volume 4.

Teso, S. and Kersting, K. (2019). Explanatory Interactive Machine Learning. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, AIES '19, pages 239–245, New York, NY, USA. Association for Computing Machinery.

Tibshirani, R. (1996). Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288. Publisher: [Royal Statistical Society, Wiley].

Tintarev, N. and Masthoff, J. (2011). Designing and Evaluating Explanations for Recommender Systems. In Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B., editors, *Recommender Systems Handbook*, pages 479–510. Springer US, Boston, MA.

Tran, T., Do, T.-T., Reid, I., and Carneiro, G. (2019). Bayesian Generative Active Deep Learning. In *International Conference on Machine Learning*, pages 6295–6304. PMLR. ISSN: 2640-3498.

Tsang, M., Cheng, D., and Liu, Y. (2018). Detecting Statistical Interactions from Neural Network Weights. *arXiv:1705.04977 [cs, stat]*. arXiv: 1705.04977.

Vedaldi, A. and Soatto, S. (2008). Quick Shift and Kernel Methods for Mode Seeking. In Forsyth, D., Torr, P., and Zisserman, A., editors, *Computer Vision – ECCV 2008*, Lecture Notes in Computer Science, pages 705–718, Berlin, Heidelberg. Springer.

Walt, S. v. d., Colbert, S. C., and Varoquaux, G. (2011). The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science Engineering*, 13(2):22–30. Conference Name: Computing in Science Engineering.

Wold, S., Esbensen, K., and Geladi, P. (1987). Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52. Publisher: Elsevier.

Wu, M., Hughes, M. C., Parbhoo, S., Zazzi, M., Roth, V., and Doshi-Velez, F. (2017). Beyond Sparsity: Tree Regularization of Deep Models for Interpretability. *arXiv:1711.06178 [cs, stat]*. arXiv: 1711.06178.

Zeiler, M. D. and Fergus, R. (2013). Visualizing and Understanding Convolutional Networks. *arXiv:1311.2901*.

Zeiler, M. D. and Fergus, R. (2014). Visualizing and Understanding Convolutional Networks. In Fleet, D., Pajdla, T., Schiele, B., and Tuytelaars, T., editors, *Computer Vision – ECCV 2014*, Lecture Notes in Computer Science, pages 818–833, Cham. Springer International Publishing.

Zhang, Y., Liao, Q. V., and Bellamy, R. K. E. (2020). Effect of Confidence and Explanation on Accuracy and Trust Calibration in AI-Assisted Decision Making. *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, pages 295–305. arXiv: 2001.02114.

Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., and Torralba, A. (2015a). Learning Deep Features for Discriminative Localization. *arXiv:1512.04150 [cs]*. arXiv: 1512.04150.

Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., and Torralba, A. (2015b). Object Detectors Emerge in Deep Scene CNNs. *arXiv:1412.6856 [cs]*. arXiv: 1412.6856.

Zhou, B., Sun, Y., Bau, D., and Torralba, A. (2018). Interpretable Basis Decomposition for Visual Explanation. pages 119–134.

Zhou, J. and Troyanskaya, O. G. (2015). Predicting effects of noncoding variants with deep learning–based sequence model. *Nature Methods*, 12(10):931–934. Number: 10 Publisher: Nature Publishing Group.

Zintgraf, L. M., Cohen, T. S., Adel, T., and Welling, M. (2017). Visualizing Deep Neural Network Decisions: Prediction Difference Analysis. *arXiv:1702.04595 [cs]*. arXiv: 1702.04595.