Bachelor Thesis

Where is the Uncertainty in Bayesian Neural Networks?

An Exploratory Analysis

Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Wilhelm-Schickard-Institut für Informatik
Methods of Machine Learning
Moritz Kniebel, `moritz.kniebel@student.uni-tuebingen.de`, 2020

Bearbeitungszeitraum:	September-Dezember 2020

Gutachter:	Prof. Dr. Philipp Hennig, Universität Tübingen
Betreuer:	Marius Hobbhahn, Universität Tübingen

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Bachelorarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Moritz Kniebel (Matrikelnummer 4059837), December 8, 2020

Abstract

The concept of uncertainty in machine learning is crucial to employ associated methods for real-world applications, such as autonomous driving or advanced screening and diagnosis of malicious diseases. Despite the complexity of deep learning models, probabilistic methods make it possible to approximate a posterior distribution and thus to have information about the model's uncertainty. Besides its practicability, the uncertainty could hold further information that could be exploited to improve deep learning methods. For instance, if a specific region in a Neural Network tends to have high uncertainty, improved training methods could focus on more certain parts first. Likewise, such information could be used to restructure a network's architecture. In this thesis, we tackled an experimental analysis of the uncertainty, since it was yet to be the subject of known research. Two Convolutional Neural Network architectures and a selection of classification-datasets formed the basis for our research. To obtain information about the uncertainty, we used two related methods, namely: The Laplace approximation for Neural Networks and the Kronecker-factored approximate curvature. In several experiments, possible properties and correlations of the uncertainty were examined. Visualizations of the uncertainty were a main focus of the analysis. We found out that there is a correlation between the size of a weight and its uncertainty. Further, we found several indications that the complexity of a network determines the uncertainty over its parameter space.

Acknowledgments

tbd.

Contents

1 Introduction **11**

2 Background **13**

2.1 From Perceptron to Deep Neural Network 13

2.1.1 The Perceptron 13

2.1.2 Convolutional Neural Networks 14

2.2 Uncertainty in Deep Neural Networks 16

2.3 Bayesian Neural Networks 18

2.4 Bayesian Modelling 18

2.4.1 Bayesian Inference 18

2.4.2 The Intractable Posterior 19

2.5 On BackPACK for PyTorch 20

3 Related Work **21**

3.1 Variational Inference 21

3.2 Bayes by Dropout 22

3.3 Deep Ensembles 22

4 Theory **23**

4.1 About the Hessian 23

4.2 Laplace Approximation of the Weights in Neural Networks 24

4.3 Kronecker-factored Approximate Curvature (KFAC) 25

5 Setup for Experiments **27**

5.1 Networks 27

5.1.1 Simple CNN 27

5.1.2 VGG-11 27

5.2 Datasets 30

5.2.1 MNIST Dataset 30

5.2.2 SVHN 31

5.2.3 CIFAR-10 and CIFAR-100 31

5.2.4 Toy Dataset 32

5.3 Obtaining the Uncertainty 33

5.3.1 Implementation of the Laplace Approximation 33

5.3.2 Implementation of KFAC 33

6 Experiments **35**
6.1 Influence of The Prior Precision 35
6.1.1 Results: Influence of the Prior Precision 35
6.2 Visualization of the Uncertainty 37
6.2.1 Results: Visualization of the Uncertainty 37
6.3 Comparing Weight-Size with Uncertainty 39
6.3.1 Results: Comparing Weight-Size with Uncertainty 40
6.4 Examine Computation 42
6.4.1 Results: Examine Computation 43
6.5 Comparing Laplace Approximation with KFAC 43
6.5.1 Results: Comparing Laplace Approximation with KFAC . . . 44
6.6 Comparing Uncertainties with Activations 44
6.6.1 Results: Comparing Uncertainties With Activations 44
6.7 Transferring Experiments to VGG-11 44
6.7.1 Results: Transferring Experiments to VGG-11 45
6.8 Uncertainty in Different Width Networks 46
6.8.1 Results: Uncertainty in Different Width Networks 46
6.9 Influence of Accuracy and Network Width on Uncertainty 47
6.9.1 Results: Influence of Accuracy and Network Width on Uncertainty 48
6.10 Extensive Examination of Uncertainty for Different Widths 49
6.10.1 Results: Extensive Examination of Uncertainty for Different Widths 49
6.11 Examination of Layers with Static Width 51
6.11.1 Results: Examination of Layers With Static Width 51

7 Conclusion **53**
7.1 Future Outlook 54

1 Introduction

Giving computers the ability to learn from given data and solve problems for us is one of the breakthroughs in computer science if not in modern times. Due to its possibilities, Machine Learning (ML) is one of the fastest-growing fields in Computer Science. This growth can be attributed to a shift from mere academic to real-world applications of ML methods. Especially the employment of Artificial Neural Networks yielded results that previously were not deemed possible, such as beating the world's top Go player (Silver et al. 2017). However, when giving a computer the authority to make a decision for us, we can hardly accept any errors. That is why further improvement demands new quantities from ML methods, such as a reliable bound of confidence and a measure of uncertainty about a prediction. A vast number of parameters influence how a Neural Network (NN) forms its prediction, given any input. Due to this complexity, a NN is often referred to as a black bock. Regardless of this description, probabilistic methods make it possible to obtain a measure of uncertainty about every prediction. With this information, we can have better trust in the decision of a computer and employ it for real-world applications, such as autonomous driving or advanced screening and diagnosis of malicious diseases.

Logically, a trained NN should make predictions with low uncertainty. As countless research aims to optimize or replace an established method, there is also scope for improving uncertainty estimation itself or other methods by exploiting the uncertainty estimates. An aspect that could determine the latter improvement is to explore how the uncertainty the uncertainty is distributed in a NN's parameter-space, i.e. how much uncertainty there is in which part of the network. Knowing the distribution and further properties of the uncertainty could help to improve associated methods. Training methods could focus on certain parts with low uncertainty, as they may be more important to make a right prediction. Consequently, areas with high uncertainty could be pruned from the network, as they may not significantly determine the predictions.

This research in this work tackles to explore the distribution of the uncertainty in NNs, as it was yet to be the topic of known research. The thesis starts by providing the required background knowledge to understand our research in the subsequent chapter 2. After presenting some related work in chapter 3, we will introduce the concepts of the specific methods we use to get information about the uncertainty in chapter 4. The experimental studies we do in this work are presented by first describing the setup in chapter 5 and the concrete experiments in chapter 6. Moreover, a main goal of the experiments is to provide visualizations of the uncertainty, as a purely numerical analysis could be problematic.

2 Background

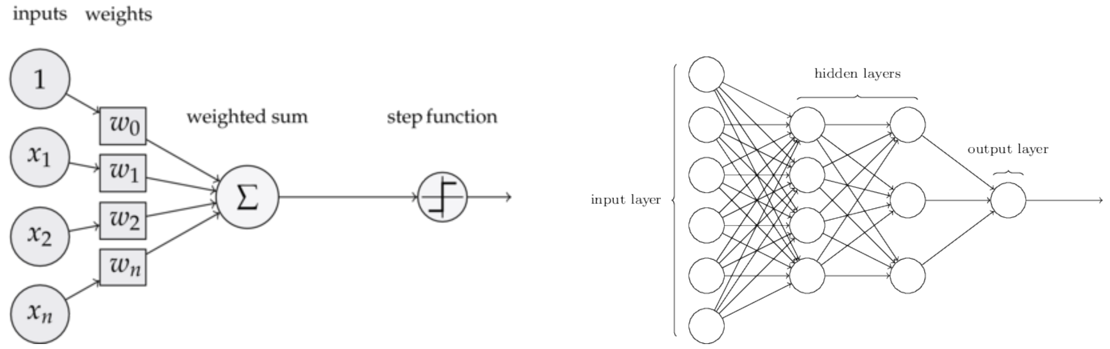
To provide a good lead-in to this work, this chapter gives an introduction to the background and underlying concepts of our research. We start with a survey of NNs that finishes with an introduction to Convolutional Neural Networks (CNNs), which is the foremost network class in this work. After that, the concept of uncertainty in deep learning is described and an introduction to Bayesian Neural Networks (BNNs) is given. Following is a section about Bayesian modeling that includes the mathematical basis of this work and the need for approximating methods. Lastly, the Python package BackPACK for PyTorch is presented, as we use it to implement approximate inference methods.

2.1 From Perceptron to Deep Neural Network

Computers showed great performance in handling vast amounts of data. Nevertheless, they always follow the instructions of humans or a program written by humans. Artificial Neural Networks were groundbreaking, as they gave computers the ability to learn from given data and perform tasks that were not directly programmed. As of today, NNs outperform humans in specific tasks, like image recognition or playing chess (Silver et al. 2017). This success is based on modeling neurons in the brain mathematically, which was pioneered in Rosenblatt's perceptron (Rosenblatt 1958). From this basis, a variety of specialized NN classes originated, such as Convolutional Neural Networks (Fukushima and Kunihiko n.d.) and Recurrent Neural Networks. Given some data $D = X, Y$, NNs can perform different tasks, such as classification or regression.

2.1.1 The Perceptron

The perceptron was the first artificial neuron and is the basis of NNs, as we know them today. It is a classifier for linearly separable tasks in the setting of supervised learning. As the perceptron can learn every function it can represent, it was pioneering for deep learning.
Inspired by a neuron in the human brain, the perceptron takes an input and fires an output. Simple mathematics can model this function by converting an input $X = \{x_1, ..., x_i\}$ into an output y_i. Every input x_i is weighted and therefore the weight-value quantifies its importance for forming the output y_i. The weighted sum of all



(a) (b)

Figure 2.1: (a): Schematic representation of the Perceptron [1]. (b) Structure of MLP with two hidden layers [2].

inputs forms an intermediate output v, which is given into a binary step function ϕ called activation function and returns the final output o (Figure 2.1a).

$$
\begin{aligned}
v &= \sum_i w_i x_i \\
o &= \phi(v)
\end{aligned}
$$

One of the inputs called the bias b has a constant value of 1. This property helps the perceptron to generalize the input data by controlling the activation function.
As mentioned, a single perceptron can classify linear separable datasets. Combining plural perceptrons extends the number of tasks it can perform. A multilayer perceptron (MLP) arranges several perceptrons layer-wise to form a feed-forward network as shown in Figure 2.1b. The MLP consists of three kinds of layers: the input layer, the hidden layer, and the output layer. An MLP can classify more complex tasks as the number of layers increases. With one hidden layer, the MLP can classify convex polygons. By adding a second hidden layer, the MLP can classify sets of arbitrary form.
The MLP is the most basic example of a Deep Neural Network (DNN) as we know it today. Nevertheless, in today's various deep learning models, the MLP is only a subset of DNNs.

2.1.2 Convolutional Neural Networks

The Convolutional Neural Network is a popular class of DNNs for image-related tasks. Over the last decade, CNNs showed impressive performance in the realm of

[2] Figure adopted from https://blog.dbrgn.ch/2013/3/26/perceptrons-in-python/

[2] Figure adopted from https://github.com/rcassani/mlp-example

image-classification. The approach builds on the application of filters to the input image. Since every pixel in an image is of different importance for specific content, filtering is a suitable concept when handling images. The filters, also referred to as kernels, are realized by an $m \times n$ matrix with feature-specific values in every field. A simple filter for a horizontal line could have the value 1 throughout every entry of a single row and the value 0 for every other entry. The filters are applied by traversing over every possible $m \times n$ pixels segment of a given image. Every color value in a segment is multiplied element-wise with the filter matrix, i.e. the dot-product of the filter and the image is returned as shown in Figure 2.2. The product of a thoroughly filtered image is called a convoluted feature, in which each neuron represents a small region of the input image. There are also two special filters commonly used in CNNs, namely: average pooling and max pooling. They return the mean value and the maximal value of a segment, respectively. The filtering in CNNs exploits local correlations in the input image by finding filters that maximally respond to small regions in the input image (Ke et al. 2018).

The architecture of a CNN combines three types of layers: the characteristic ConvLayers, the pooling layers (e.g. maxpool, avgpool), and the fully connected layers known from feedforward networks. A CNN starts with a sequence of ConvLayers and is followed by some fully connected layers. The pooling layers are located after specific ConvLayers to reduce the input's spatial size and thus the number of parameters. Each ConvLayer applies a defined number of filters. For example, a ConvLayer of size (1, 32) returns 32 convoluted features of one input image. The outputs produced by a ConvLayer feed into the following ConvLayer, whereby the input image is filtered down to its relevant features for classification (Figure 2.3). Backpropagation training adjusts the filters and weights to minimize the expected error. During training, the filters evolve to detect complicated structures or specialize in complementing another filter. In the example of handwritten digits, trained filters learned to detect whole digits, as visualized in Figure 2.3.

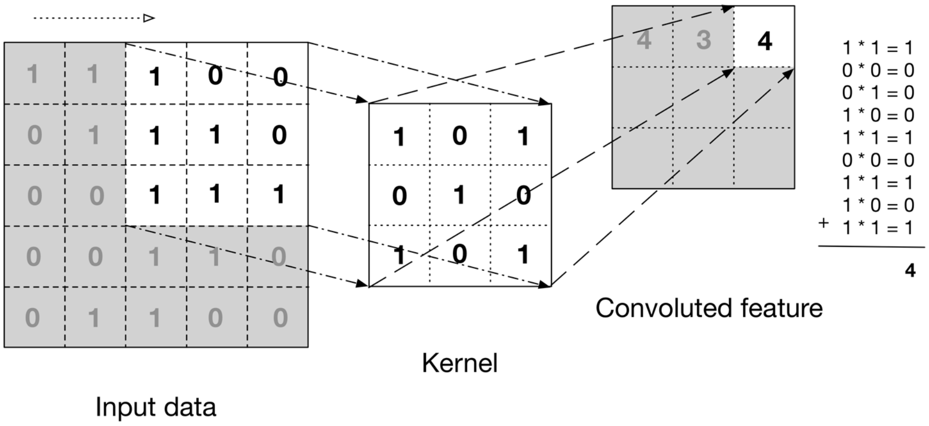


Figure 2.2: Applying a 3x3 kernel on a 5x5 input. This results in an 3x3 convoluted feature [3].

2.2 Uncertainty in Deep Neural Networks

After the above survey on NN we will now thematize the problems of deploying them for real-world applications.
A trained NN performs well for data inside or similar to the training data. The predictions for such data are accurate and assigned with high confidence. In contrast, NNs should predict with low accuracy and confidence for inputs far away from the training data. But in fact, NNs tend to be overconfident for out-of-distribution inputs (Hein et al. 2019). At the same time, NNs can easily be fooled by an input, for instance can an added perturbation lead to misclassification of an input. Such misconceptions entail severe problems in safety-critical applications, such as autonomous driving or medical use cases. Real-world data inherits variation, whereby events far away from the training data naturally occur. An example of such an event could be a self-driving car entering a new environment. To use NNs for real-world applications, they need the ability to tell how certain or rather uncertain they are about a prediction. There are two types of uncertainty, which are present in deep learning:

- *epistemic uncertainty* is knowledge uncertainty, i.e. the uncertainty in the parameter-space and structure of a model, observing dataset D.

- *aleatoric uncertainty* is the inherent variation in the observations, the intrinsic uncertainty of the world they take place in (stochastic uncertainty). One example is noisy data.

[3] Figure adopted from Patterson and Gibson (2017)

[4] Figure adopted from https://towardsdatascience.com/mnist-handwritten-digits-classification-using-a-convolutional-neural-network-cnn-af5fafbc35e9

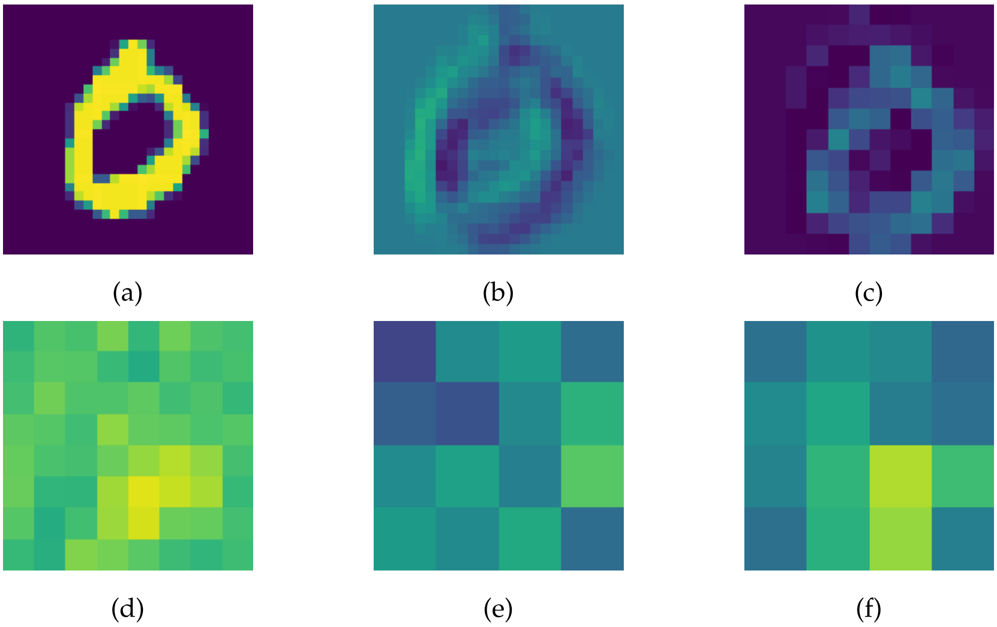


Figure 2.3: A CNN was trained on the MNIST dataset. The intermediate activations for a specific filter were visualized for every layer. (a) The input image. (b) The feature map returned by the first ConvLayer. (c) The inputfeed of the second ConvLayer. (d) The feature map returned by the second ConvLayer. (e) The inputfeed to the linear layer. (f) The feature map returned by the linear layer (for visualization-purposes, the flattened weights were reassembled to a 4×4 matrix). As the input image gets filtered down, the convoluted features become less interpretable for the human eye.

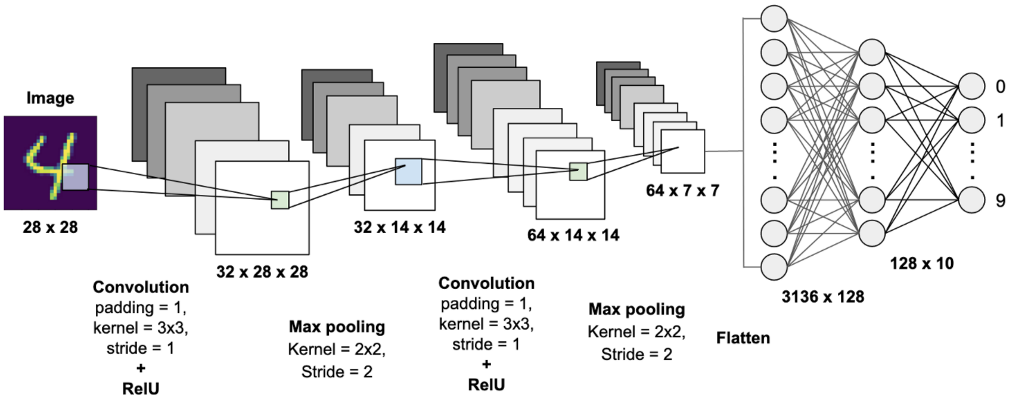


Figure 2.4: CNN architecture with two ConvLayers, Maxpooling and two linear layers [4].

Supplementing ML, i.e. deep learning with Bayesian statistics incorporates the above mentioned uncertainty into a DNN. This probabilistic view on ML makes NNs more reliable for real-world application.

2.3 Bayesian Neural Networks

A probabilistic view on ML comprises a specific treatment of DNNs, which will be introduced hereinafter.
DNNs lack the ability to form a posterior distribution. This is due to the use of concrete values in the weight space of a network. The Bayesian Neural Network (BNN) tackles this problem. BNNs are not specifically a new network class but rather describe a Bayesian framework for any kind of DNN (MacKay 1992). This framework comprises a set of tools to transform a DNN into a probabilistic model by inferring distributions over a DNN's weight space. Given some training data, the prior distributions over the weights are adjusted to attain the most probable set of weights. Further, a posterior distribution can be obtained to get uncertainty estimates from a prediction. To gain intuition, a BNN can be interpreted as an ensemble of all possible models of given architecture after observing dataset D.

2.4 Bayesian Modelling

The probabilistic methods that provide us with information about a models uncertainty are based on Bayesian statistics. Encountering new evidence modifies our beliefs about the occurrence of an event. For example, our belief about the upcoming weather updates when we see clouds in the sky. The described probabilities are mathematically described by Bayes' theorem, which is the fundamental rule to do inference.

2.4.1 Bayesian Inference

In probability theory, inference describes the process of reasoning a probability by analyzing new data. Bayesian inference leverages Bayes' theorem to deduce probabilities from given data. Recall Bayes' theorem:

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)},$$

where events A and B have prior probabilities $p(A)$ and $p(B)$ and $p(B|A)$ is the likelihood, i.e. the conditional probability of event B given the occurrence of event A. Here, the probabilities of A and B were exact probabilities. However, for several applications, such as deep learning models, probability distributions are better representations of A and B. For deep learning models, the notation of Bayes' theorem changes to:

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)},$$

where θ denotes the parameter space of the used model, and D is the observed data. Bayesian inference adjusts the prior distribution $p(\theta)$ after observing data D. This adjustment forms the posterior distribution $p(\theta|D)$. Comparing the posterior distribution with the prior distribution shows the effect of the new data D. The most probable value in the posterior distribution is called the maximum a posteriori (*MAP*) estimate. Since the *MAP* estimate will not have a probability of 100%, the posterior distribution holds information on how uncertain the model is about the outcome.
Moreover, the posterior distribution gives the possibility to perform further inference. Given a new data point x^* the output is predicted by integrating:

$$p(y^*|x^*, D) = \int p(y^*|x^*, \theta)p(\theta|D)d\theta$$

Having a posterior distributions brings great possibilities to deep learning. Analytically speaking, the posterior distribution has its downsides, though.

2.4.2 The Intractable Posterior

Bayesian modeling provides the mathematical basis to form a posterior distribution. In practice, the posterior distribution is only tractable for simple models, such as Bayesian linear regression. More complex models have an analytically intractable posterior distribution. This intractability is due to marginalization, which is the key to a meaningful posterior distribution. The model evidence $p(D)$ is responsible for marginalization by integrating:

$$p(D) = \int p(D|\theta)p(\theta)d\theta$$

This integral gets analytically intractable, as it includes all possible parameter values for θ. The number of all possible values for θ grows exponentially with the model complexity. Since we are still interested in the properties of the posterior distribution, an approximation is needed.

Approximate Inference

Approximate inference is an essential field of today's machine learning. It makes reasoning about engaging and real-world data for many applications possible and, more importantly, assessable. From a high-level perspective, approximate inference solves Bayes' theorem for complex cases. Chapter 3 and 4 will describe some approaches to perform approximate inference.

2.5 On BackPACK for PyTorch

The implementation of approximating inference is significantly important for out research. BackPACK (Dangel et al. 2020) is a library for PyTorch extending the backwards pass of a NN to compute additional information. Such information could be the individual gradient of a minibatch or in the context of this thesis curvature approximations of the Hessian [5]. To receive this second-order information only a few lines of code are necessary. As we can extract the mentioned information after training a NN, BackPACK made it easy to integrate DNNs into the Bayesian framework and gather information about the uncertainty.

[5] BackPACK website: https://backpack.pt

3 Related Work

Bayesian Neural Networks have been gaining attention since the early 1990s, and the Bayesian toolbox is used broadly in connection with deep learning. Many methods to perform approximated inference have been developed. This chapter gives a high-level summary of some of the most popular methods to get an idea of possible approximate inference methods.

3.1 Variational Inference

Variational inference is frequently mentioned in the context of uncertainty estimation. A family of distributions $q(w)$ parameterized with θ is defined to approximate the posterior distribution $p(w|D)$. The goal is to find the setting of parameters θ^* that minimizes the difference between q and p. The Kullback-Leibler divergence (KL) formalizes the difference between the two distributions as:

$$KL(q(w|\theta)||p(w|D)) = \int q(w|\theta)\ \log \frac{q(w|\theta)}{p(w|D)} dw$$

The parameters in q determine the difference to p. This formulates the objective minimization-problem:

$$\begin{aligned}\theta^* &= \arg\min_{\theta} \int_{\theta} q(w|\theta)\ \log \frac{q(w|\theta)}{p(w|D)} dw \\ &= \arg\min_{\theta} \int_{\theta} q(w|\theta)\ log \frac{q(w|\theta)}{p(D|w)p(w)} dw\end{aligned}$$

As this objective is dependent on the intractable posterior, the minimization is not possible in this form. But the KL has some properties that help to approximate $p(w|D)$:

1. $\forall p, q : KL(q||p) > 0$
2. $KL(q||p) = 0 \iff q = p$
3. $\log p(D) = KL(q(w|\theta)||(p(w|D)) + \mathbb{E}_{q(w|\theta)} [\log p(D, w) - \log q(w|\theta)]$

Decomposing the log evidence reveals the equivalence of minimizing the KL and maximizing the ELBO. The non-negative KL and the ELBO add up to the log evidence. Therefore, the KL is minimal when the ELBO is maximal. Maximizing the ELBO is computationally possible, as it only depends on q. Stated insights reformulate the above minimization problem to:

$$\theta^* = \arg\max_\theta \mathbb{E}_{q(w|\theta)} \left[\log p(D, w) - \log q(w|\theta)\right]$$

3.2 Bayes by Dropout

Dropout is a technique that was first introduced to improve the training of NNs in Srivastava et al. (2014). Excluding random neurons during each iteration of training reduces the network's generalization error. A dropout is implemented by setting the weights of a randomly chosen neuron to zero, virtually eliminating it from the network. Theoretically, training with dropout optimizes a number of sparse networks with shared weights on the same task. The network generalizes much better as dropout forces it to learn redundant and independent features.

Gal and Ghahramani (2016) proposed to use dropout during test time for Bayesian approximation. Doing this, we get different outputs for a series of identical inputs. Given a sequence of identical input, the network returns a sequence of slightly different outputs, as different neurons are dropped out for every individual input. Such sequence of outputs is interpreted as a set of MC-samples. Such samples give the possibility to form a predictive distribution. Uncertainty measurements, such as the variance, can be estimated from this predictive distribution.

3.3 Deep Ensembles

Deep ensembles describe a simple method to assess model uncertainty using DNNs (Lakshminarayanan et al. 2017). The idea of this approach is to have several NNs trained on the same task. In such a way, a NN's strengths and weaknesses complement each other. This synergy goes back to Nicolas de Condorcet's famous jury theorem: If each juror (in a jury) has a probability greater than 50% to decide correctly on a problem, then the probability of a correct jury verdict converges to 100%, as the size of the jury increases. Likewise, numerous networks with a mediocre validation accuracy know more about the data than one network with high accuracy.

Diversification in weight initialization, data shuffing, and architectures induce the differences between ensembled NNs. Taking every member's output into account gives insights into uncertainty, such as the variance. Deep ensembles perform notably well for out-of-distribution inputs. With its large number of parameters, an NN can represent observed data with many different functions. The variance within all possible functions' outputs is significant for out-of-distribution inputs. Hence, we get a precise measure of uncertainty from an ensemble of networks. As ensembling needs to train numerous networks, the method has its downsides in time and memory consumption.

4 Theory

As the previous chapter introduced some related work, we will now introduce the approximating methods that we took into account in our research. We employ two methods, which use second-order information to approximate the posterior distribution. To provide good intuition, we will start by introducing the Hessian, as both methods depend on it. After that, the Laplace approximation for NNs is introduces, which is the foremost method in this work. The second methods is the Kronecker-factored approximate curvature and will be introduced lastly.

4.1 About the Hessian

The Hessian w.r.t. the Loss holds second-order information, which are essential properties of the gradient, i.e. information about the rate of change in the gradient. In a high-dimensional space, like the loss surface of a NN, the gradient has many directions. For each of which directions, the rate of change is different. For every direction, there is one row in the Hessian matrix w.r.t. Loss L, which is defined as:

$$H_{ij} = \begin{bmatrix} \frac{\partial^2 L}{\partial W_1^2} & \frac{\partial^2 L}{\partial W_1 \partial W_2} & \cdots & \frac{\partial^2 L}{\partial W_1 \partial W_i} \\ \frac{\partial^2 L}{\partial W_2 \partial W_1} & \frac{\partial^2 L}{\partial W_2^2} & \cdots & \frac{\partial^2 L}{\partial W_2 \partial W_i} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 L}{\partial W_j \partial W_1} & \frac{\partial^2 L}{\partial W_j \partial W_2} & \cdots & \frac{\partial^2 L}{\partial W_j^2} \end{bmatrix}$$

The entries of this matrix fall into two classes of second-order derivates. The diagonal entries are pure partial second-order derivatives, whereas the non-diagonal entries are mixed partial second-order derivatives. The large number of parameters in DNNs makes the computation of the full Hessian infeasible, due to the storage it would occupy. An approximation of the generalized Gauss-Newton (GNN) diagonal will be used in this thesis to avoid this problem. The GGN and the Hessian are equivalent if the associated NN uses piece-wise linear activation functions (Dangel et al. 2020). As we will use ReLU networks, the equivalence holds in this work. The diagonal of the GNN gets approximated as [1]:

$$diag(G(\theta^{(i)})) \approx \frac{1}{N}\sum_{n=1}^{N} diag([(\mathbf{J}_{\theta^{(i)}}\, z_n^{(i)})^T \hat{S}(z_n^{(i)})]\, [(\mathbf{J}_{\theta^{(i)}}\, z_n^{(i)})^T \hat{S}(z_n^{(i)})]^T),$$

where **J** is and Jacobian and

$$\nabla_f^2 \ell(f(x_n, \theta), y_n) = S(z_n^{(L)})S(z_n^{(L)})^T,$$

where $z_n^{(L)}$ is the input of the last layer L.

4.2 Laplace Approximation of the Weights in Neural Networks

The Laplace approximation is a way to approximate a posterior distribution with a Gaussian. Using it for Neural Networks was pioneered by MacKay (1992).The Laplace approximation places a Gaussian at the mode of the posterior distribution (Figure 4.1). The curvature of this Gaussian is found in the inverse of the covariance matrix w.r.t. the Loss L. The single steps will be explained in more detail to get a more specific intuition of Laplace approximation for NNs.
The first step is to locate the mode W_{MAP} of the posterior distribution via optimization, denoted as $\hat{\theta}$. This is done by merely training the network, whereby the optimizer seeks to minimize the expected loss with backpropagation:

$$\hat{\theta} = \arg\min_{\theta} L(\theta)$$

The next step is to perform a second-order Taylor expansion around the mode $\hat{\theta}$. This is where the approximation takes place:

$$log\, p(\theta|D) = log\, p(\hat{\theta}|D) + (\theta - \hat{\theta})^T \mathbf{J}(\theta - \hat{\theta}) - \frac{1}{2}(\theta - \hat{\theta})^T \mathbf{H}(\theta - \hat{\theta}),$$

where the first-order term **J** can be dropped since the gradient at W_{MAP} is equal to zero:

$$log\, p(\theta|D) = log\, p(\hat{\theta}|D) - \frac{1}{2}(\theta - \hat{\theta})^T \mathbf{H}(\theta - \hat{\theta}),$$

where **H** is the Hessian w.r.t. the loss $\mathbf{H}_{ij} = \frac{\partial^2 L}{\partial W_i \partial W_j}$ evaluated at W_{MAP}. As the derived equation is of Gaussian functional form, we approximated the posterior as a normal distribution:

$$p(W|\mathcal{D}) \approx \mathcal{N}(\hat{\theta}, \mathbf{H}^{-1})$$

In this thesis we don't focus on the approximated posterior distribution itself, but on the measure of uncertainty it provides. This measure of uncertainty is the curvature we approximated. Therefore we use the inverse of the Hessian as a proxy for the uncertainty.

[1] calculated with the BackPACK package. More details in the Appendix of Dangel et al. (2020)

[2] Figure adopted from https://www.r-bloggers.com/2013/11/easy-laplace-approximation-of-bayesian-models-in-r/

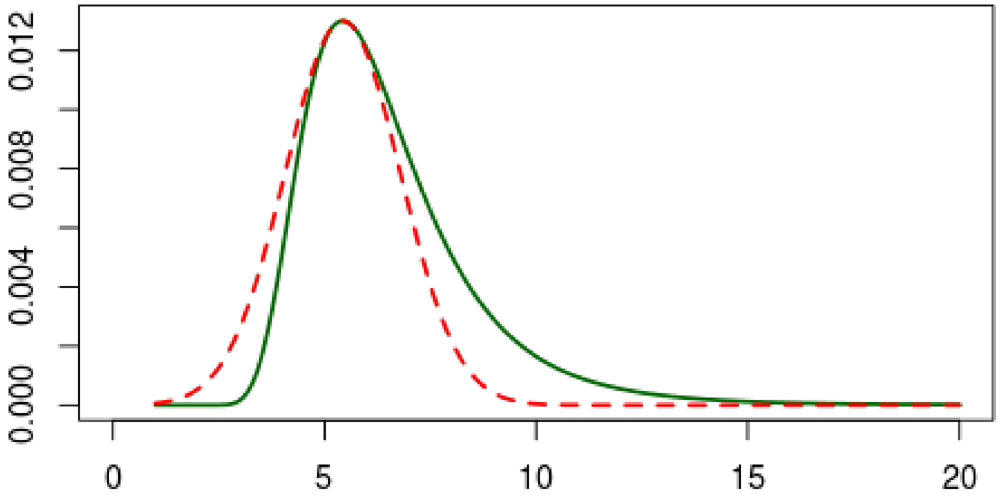


Figure 4.1: Example of the Laplace approximation. The solid green line shows the posterior distribution that needs to be approximated. The red dotted line shows the approximated posterior, which is a Gaussian centered at the mode of the true posterior.[2].

4.3 Kronecker-factored Approximate Curvature (KFAC)

Looking at state of the art Neural Networks, with millions of weight parameters, the Laplace approximation proposed in MacKay (1992) reaches its limit. This is due to a huge amount of storage (several terabytes) the Hessian would occupy, as mentioned above. Another approximation besides the diagonal GNN was presented in Ritter et al. (2018). They approximate the Hessian by a block matrix of independent Kronecker-factorized matrices. This brings some benefits to it, as Kronecker products reduce the computational complexity and can be inverted separately. The method's idea is to block-diagonalize the Hessian, whereby every block corresponds to one layer of the associated NN. Each of those diagonal blocks is approximated by Kronecker-factorization (Figure 4.2). Thus, the sample Hessian H_λ of each layer λ is approximated with:

$$[H_\lambda]_{(a,b)(c,d)} \equiv \frac{\delta^2 L}{\delta W^\lambda_{a,b} \delta W^\lambda_{b,c}} = a_b^{\lambda-1} a_d^{\lambda-1} [\mathcal{H}_\lambda]_{a,c},$$

where a_λ are the activation values of layer λ and $\mathcal{H}$ is the pre-activation Hessian of layer λ:

$$[\mathcal{H}_\lambda]_{a,b} = \frac{\delta^2 E}{\delta h_a^\lambda \delta h_b^\lambda},$$

where h_λ is the pre-activation of layer λ. Additionally, we can denote the covariance of the incoming activations $a_{\lambda-1}$ as:

$$[\mathcal{Q}_\lambda]_{c,d} = (a_c^{\lambda-1} a^T{}_d^{\lambda-1})$$

With the approximated kronecker products $[\mathcal{Q}_\lambda]$ and $[\mathcal{H}_\lambda]$ the sample Hessian of each layer is:

$$\begin{aligned} H_\lambda &= \frac{\delta^2 L}{\delta \text{vec}(W_\lambda) \delta \text{vec}(W_\lambda)} \\ &= (a_{\lambda-1} a_{\lambda-1}^T) \otimes \frac{\delta^2 L}{\delta h_\lambda \delta h_\lambda} \\ &= \mathcal{Q}_\lambda \otimes \mathcal{H}_\lambda, \end{aligned}$$

where $\otimes$ denotes the Kronecker-product.
Since the layerwise Hessian is split into two Kronecker products, both of which should be described separately. $[\mathcal{Q}_\lambda]$ is the Hessian of the layerwise output w.r.t. the loss L. $[\mathcal{H}_\lambda]$ is the auto-correlation matrix of the layer-wise inputs (Wu et al. 2020). With those Kronecker factors, the posterior of the weights in layer λ can be approximated as:

$$W_\lambda \sim \mathcal{MN}(W_\lambda^*, \mathcal{Q}^{-1}, \mathcal{H}^{-1}),$$

where $\mathcal{MN}$ is the matrix Gaussian distribution.

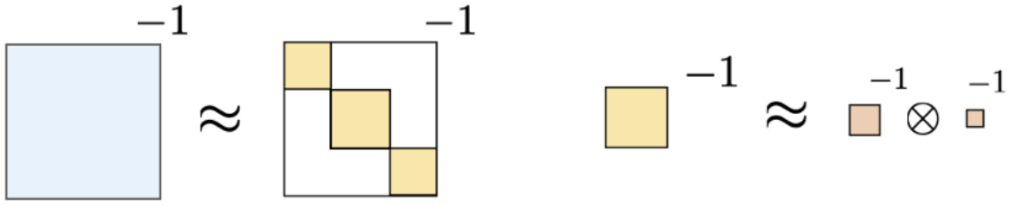


Figure 4.2: visualization of KFAC. The Hessian (blue) gets approximated by block-diagonals. Each of the diagonal blocks is approximated by a krockecker product of two smaller matrices. Inverting the Hessian can be achieved by inverting the kronecker products [3].

[3] Figure adopted from https://towardsdatascience.com/introducing-k-fac-and-its-application-for-large-scale-deep-learning-4e3f9b443414

5 Setup for Experiments

In this work, we conduct several experiments to understand and analyze the uncertainty in BNNs. Due to the variety in the scope of deep learning, we need to limit this work by focusing on one network class. We choose to use CNNs for image classification, as classification generally pleases to incorporate a measure of uncertainty. Besides that, the realm of image classification is a significant field of deep learning.

This chapter introduces the setup we use for the experimental analysis of the uncertainty. First, we will introduce the used network architectures. Next, the selection of datasets, which we use during the experiments is introduced. To support the summary of every dataset, an excerpt of its contents is shown. Finally, the implementations of Laplace Approximation and KFAC are explained.

5.1 Networks

Two network architectures are used in this thesis. Both networks are CNNs, where one has a simple, and the other one a more complex architecture. Each architecture and its configuration is described hereinafter.

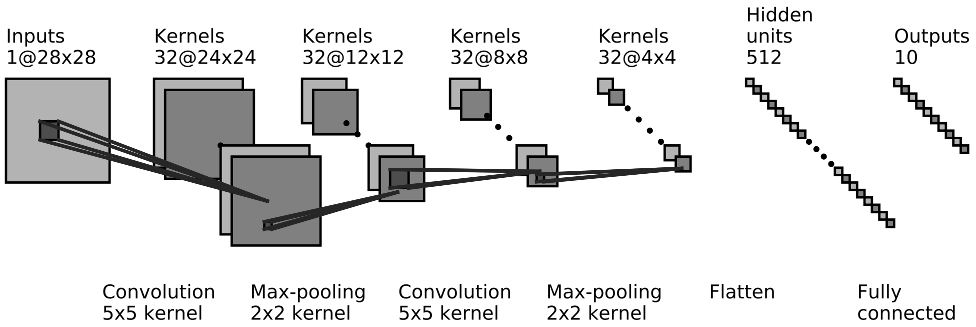
5.1.1 Simple CNN

As we enter the experiments with this network, it needs to fulfill some basic requirements. It has to be simple and achieve good validation accuracy on a standard benchmark dataset. Since we work with CNNs for image classification, the first network should generalize well on the MNIST dataset.

We use a CNN architecture that includes two ConvLayers with ReLU activations and one linear layer at the end. Cross-entropy loss and Adam are used to minimize the expected error during training. We will refer to this CNN as *simpleNet* from now on. A visualization of simpleNet and its exact specifications can be found in Figure 5.1a and Table 5.1, respectively.

5.1.2 VGG-11

To make possible findings comparable to state-of-the-art deep learning models, we need a CNN architecture that is capable of achieving state-of-the-art results.



(a) Schematic structure of simpleNet for a 28x28 input with 10 classes, such as MNIST.

Table 5.1: simpleNet architecture and hyperparameters. ConvLayer parameters are of form (input_size, output_size, kernel_size). MaxPool2d parameters are of form (kernel_size, stride)

Network architecture		
Layer	**Name**	**Parameters**
	Conv2d	$(1, 32, 5)$
1	ReLU	-
	MaxPool2d	$(2, 2)$
	Conv2d	$(32, 32, 5)$
2	ReLU	-
	MaxPool2d	$(2, 2)$
3	Linear	$(512, 10)$
Network Hyperparameters		
Hyperparameter	**specification**	
Lossfunction	Cross-Entropy	
Optimizer	Adam	
Learning-Rate	0.001	
Weight-Decay	5e-4	

For image-classification, the VGG (Simonyan and Zisserman 2014) architecture satisfies this requirement as it achieves decent results on ImageNet (Deng et al. 2009). There are several VGG architectures, which primarily differ in their depths. In this work, we choose the VGG-11 with eight ConvLayers and three linear layers. We adjusted the specifications of VGG-11 to achieve good performance on the used datasets, i.e. CIFAR and SVHN. The detailed specifications are listed in Table 5.2.

Table 5.2: VGG-11 architecture and hyperparameters. ConvLayer parameters are of form (input_size, output_size, kernel_size, stride, padding). MaxPool2d parameters are of form (kernel_size, stride, padding, dilation). We decrease the learning rate during training to find the best parameters for good accuracy.

Network architecture		
Layer	**Name**	**Parameters**
1	Conv2d ReLU MaxPool2d	(3, 64, 3, 1, 1) - (2, 2, 0, 1)
2	Conv2d ReLU MaxPool2d	(64, 128, 3, 1, 1) - (2, 2, 0, 1)
3	Conv2d ReLU	(128, 256, 3, 1, 1) -
4	Conv2d ReLU MaxPool2d	(256, 256, 3, 1, 1) - (2, 2, 0, 1)
5	Conv2d ReLU	(256, 512, 3, 1, 1) -
6	Conv2d ReLU MaxPool2d	(512, 512, 3, 1, 1) - (2, 2, 0, 1)
7	Conv2d ReLU	(512, 512, 3, 1, 1) -
8	Conv2d ReLU MaxPool2d	(512, 512, 3, 1, 1) - (2, 2, 0, 1)
9	Dropout Linear ReLU	0.5 (512, 512) -
10	Dropout Linear ReLU	0.5 (512, 512) -
11	Linear	(512, 10)
Network Hyperparameters		
Hyperparameter	**specification**	
Lossfunction	Cross-Entropy	
Optimizer	SGD	
Learning-Rate	0.1, 0.01, 0.001	
Momentum	0.9	
Weight-Decay	5e-4	

5.2 Datasets

We train the above introduced CNN architectures on several datasets. To provide a good basis for comparison, those datasets incorparate different degrees of complexity.

5.2.1 MNIST Dataset

The Modified National Institute of Standards and Technology database (n.d.) of handwritten digits or MNIST dataset for short is the primary benchmark for image classification. MNIST consists of 70000 black and white images split into 60000 training and 10000 test images. Each image has a size of 28×28 pixels and shows a handwritten digit between zero and nine at its center. Due to the varying styles of handwritten digits, the dataset gives robustness to a properly trained model. Therefore MNIST is gladly used for method comparison and entry-level machine learning examples. Figure 5.2 shows a set of images contained in the MNIST dataset.

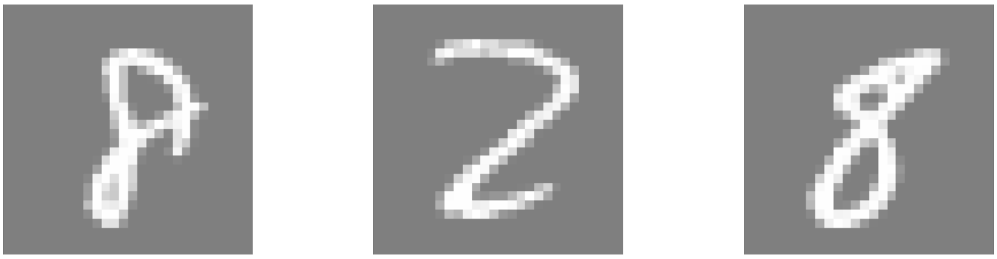
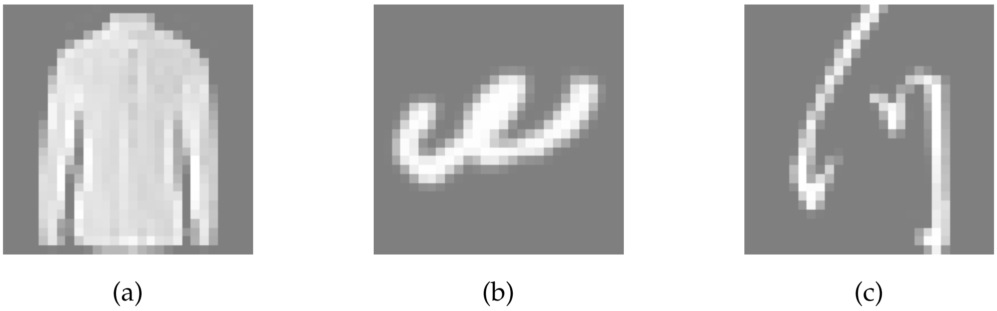


Figure 5.2: Examples of the MNIST dataset.

Alternative MNIST

The MNIST dataset is extensively used for image-related tasks in machine learning. Complementary, there are some drop-in replacements for the MNIST dataset, namely: Extended MNIST (EMNIST) (Cohen et al. 2017), Fashion-MNIST (FMNIST)(Xiao et al. 2017), and Kuzushiji-MNIST (KMNIST)(Clanuwat et al. n.d.). Those alternative datasets share specifications with MNIST but incorporate different content. EMNIST contains images of handwritten characters. FMNIST, developed by Zalando, pictures various pieces of clothing. Images in KMNIST show phonetic letters of hiragana, a Japanese syllabary. A sample of each dataset is shown in Figure 5.3.



(a) (b) (c)

Figure 5.3: Examples of (a) FashionMNIST,(b) EMNIST,(c) KMNIST.

5.2.2 SVHN

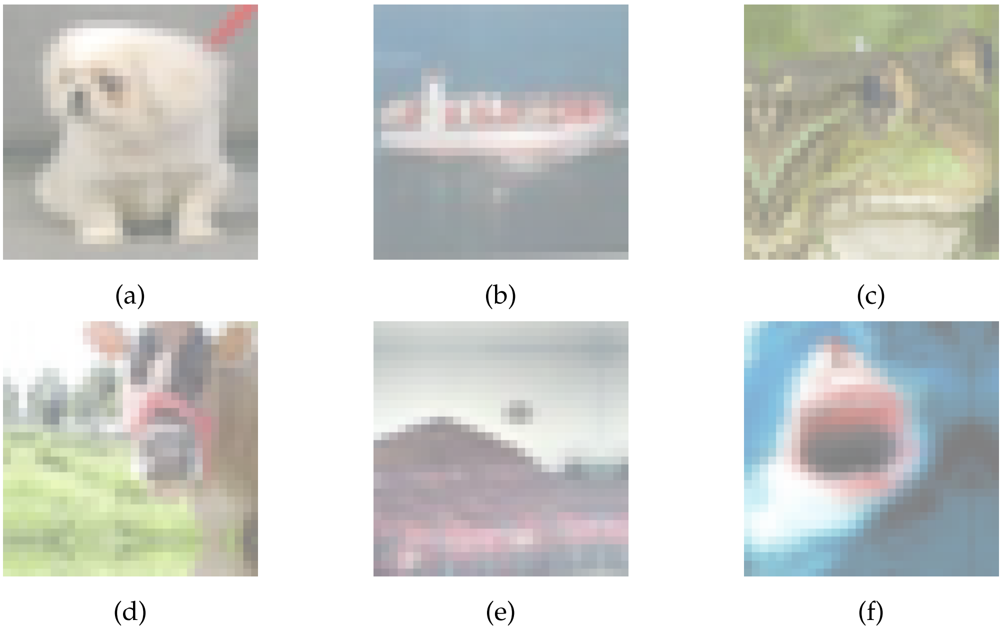
The Street View House Numbers (SVHN) dataset (Netzer et al. 2011) is a popular computer vision dataset containing real-world images (colored). The dataset is widely used to develop and benchmark machine learning and object detection algorithms, as it requires minimal data preprocessing and formatting. Taking pictures from a natural scene adds real-world complexity to the data. Therefore, SVHN is the next step to simpler datasets, such as MNIST. The added difficulty is visible in the lack of contrast normalization, overlapping digits, and distracting features. SVHN includes 73257 training and 26032 test images (32 × 32 pixels) with an additional 531131, more simplistic, images to extend the training set. The pictures in SVHN show photographs of house numbers differing in style, size, and perspective. Every house number contains numbers from zero to nine, which are labeled as ten distinct classes. An excerpt from SVHN is shown in Figure 5.4.

Figure 5.4: Examples from SVHN dataset.

5.2.3 CIFAR-10 and CIFAR-100

The Canadian Institute For Advanced Research (CIFAR) created two established datasets for object recognition and image classification tasks (Krizhevsky 2012). Both CIFAR-10 and CIFAR-100 are subsets of the 80 Million Tiny Images dataset. The CIFAR datasets were developed to test and optimize machine learning, i.e. computer vision algorithms.

CIFAR-10 is more frequently used to benchmark image classification methods. The dataset consists of 60000 images, divided into 50000 training and 10000 test images. A relatively low resolution of 32 × 32 pixels makes fast training and testing possible. The ten different classes in CIFAR-10 show vehicles like cars or airplanes and animals like cats and deer. Every class is independent, which means that there is no overlap existent in the images.
CIFAR-100 is similar to CIFAR-10, except the number of classes is raised to 100. Those classes consist of 20 superclasses, where each superclass contains five subclasses. An example of a superclass is reptiles containing the subclasses crocodile, dinosaur, lizard, snake, and turtle. To get an idea of both CIFAR datasets, Figure x and y show examples of CIFAR-10 and CIFAR-100, respectively.


(a) (b) (c)

(d) (e) (f)

Figure 5.5: Examples from CIFAR-10 (a - c) and CIFAR-100 (d - f) datasets: (a) dog, (b) ship, (c) frog, (d) cattle, (e) bed, (f) shark.

5.2.4 Toy Dataset

We defined a Toy Dataset to use a perceptron for linear regression. The dataset consists of 500 input-target pairs $x, y = (x, x \cdot c + \epsilon)$, where c is a constant and ϵ is Gaussian noise. We can manipulate c to specify the data-pairs x, y. Figure x shows a plot of all datapoints for a specific value of c.

5.3 Obtaining the Uncertainty

Two methods to perform approximate inference were introduced earlier in Chapter 4. This section gives a high-level description of how both methods are implemented.[1]

5.3.1 Implementation of the Laplace Approximation

Chapter 4.2 explained the theory of the Laplace approximation. However, we are not exactly interested in the network's posterior distribution obtained by the Laplace approximation. Rather, we are exclusively interested in the magnitude of every weights uncertainty. Since $\mathbf{H}^{-1}$ determines the curvature of the posterior distribution, we employ the inverted Hessian $\mathbf{H}_{\lambda}^{-1}$ of each layer λ as a proxy for its weights uncertainties. To avoid the computational difficulties of the exact Hessian, we approximate the diagonal Hessian w.r.t. the Loss for every layer. A prior precision is added to every entry of the approximation and the obtained matrix is inverted. For individual layers λ, the diagonal entries of $\mathbf{H}_{\lambda}^{-1}$ are the variances of each weight $w_{\lambda i}$. We decided to use the standard deviation as a interpretation of a weight's uncertainty. Therefore we apply the square-root to every entry in the approximated matrix $\textit{diagonal}(\mathbf{H}_{\lambda}^{-1})$. For each layer in a NN, this method yields a tensor of the associated layer's size, filled with the standard deviations of its weights.

5.3.2 Implementation of KFAC

We implement the KFAC method similar to its theory, described in Chapter 4.3. For each layer, the implementation works with the weight matrix **W** and the bias matrix **b**. The factorizations **U** and **V** are approximated from **W** and an additional matrix **B** is approximated from **b**. A prior precision is then added to the obtained matrices and their values are corrected by the minibatch-size. More details on the treatment of the factorization can be found in Dangel et al. (2020). Our implementation returns **U**, **V**, **B** and their inverses $\mathbf{U}^{-1}$, $\mathbf{V}^{-1}$, and $\mathbf{B}^{-1}$ for each layer.

[1] The exact implementations can be found in *insert GitHub Link*

6 Experiments

This chapter contains records of all experiments that are conducted to explore the uncertainty in BNNs. Successively, every experiment is introduced with its procedure, and after that its results are presented. The initial experiments revolve around simpleNet and later experiments primarily take the VGG-11 into account. A main goal of the experimental analysis is to provide visualizations of the uncertainty.

6.1 Influence of The Prior Precision

Choosing an appropriate prior for the network's weights is discussed in numerous research, like Silvestro and Andermann (2020).
In this first experiment, we want to find how different prior precisions influence the weight's resulting uncertainty. We train simpleNet on MNIST, FashionMNIST, KMNIST, and EMNIST. For a series of prior precisions p_0, the uncertainty of the trained network is evaluated, where $0.00001 \leq p_0 \leq 1$. Our Laplace approximation implementation obtains the uncertainties of simpleNet's weights, given a prior precision. The goals of this experiment are threefold. 1) We want to find out how much of an influence the prior precision has on the resulting uncertainty. If the prior precision strongly determines the uncertainty, we need to decrease its value. Otherwise, the resulting uncertainties are not meaningful. 2) An overall view of the uncertainties in every layer for different prior precision shows whether the prior precision influences a particular layer. 3) We can find an appropriate prior precision to use in the following experiments. Additionally, the gathered data shows how uncertainty is distributed throughout the network.

6.1.1 Results: Influence of the Prior Precision

To provide the best overview of the experiment's results, we listed all of them into Table 6.1. Since we use the standard deviation to measure uncertainty, the maximal uncertainty given a prior precision p_i is:

$$\sigma_{max}(p_i) = \sqrt{\frac{1}{p_i}}$$

The numbers in Table 6.1 show that a larger prior precision strongly determines the uncertainty. For instance, the uncertainties for prior precision of 1 are very

Table 6.1: Average uncertainties in layers of simpleNet for different prior precision and datasets. Every value in the table is the mean value of ten seeds.

MNIST						
Prior Precision	**0.weight**	**0.bias**	**3.weight**	**3.bias**	**7.weight**	**7.bias**
1	0.9989	0.9945	0.9999	0.9996	0.9954	0.9988
0.1	3.1284	3.0128	3.1587	3.1483	3.0441	3.1265
0.01	9.1788	7.5771	9.8925	9.5997	8.2553	9.0651
0.001	20.8763	13.0099	29.0171	24.2665	18.1852	19.0138
0.0001	31.0823	15.7159	66.6582	41.2746	34.2091	25.4718
0.00001	34.9811	16.8371	112.8993	67.5562	64.5365	27.2228
KMNIST						
Prior Precision	**0.weight**	**0.bias**	**3.weight**	**3.bias**	**7.weight**	**7.bias**
1	0.9983	0.9864	0.9995	0.9986	0.9848	0.9971
0.1	3.1117	2.8497	3.1476	3.1199	2.8321	3.0737
0.01	8.8461	6.2232	9.591	8.9309	6.4772	8.0303
0.001	18.106	8.7472	24.6546	18.6511	12.0998	12.9998
0.0001	23.7982	9.5417	46.9226	31.0698	23.552	14.4653
0.00001	25.8735	10.2955	84.1586	63.1653	55.973	14.6531
EMNIST						
Prior Precision	**0.weight**	**0.bias**	**3.weight**	**3.bias**	**7.weight**	**7.bias**
1	0.9987	0.993	0.9999	0.9995	0.9956	0.9989
0.1	3.1244	2.9798	3.1589	3.1477	3.049	3.1288
0.01	9.1122	7.2894	9.8977	9.5889	8.3134	9.1183
0.001	20.4832	12.1048	29.1892	24.3872	18.5495	19.4224
0.0001	30.3415	14.3021	69.1649	44.6483	36.2236	26.2278
0.00001	34.6683	15.2217	135.9278	78.8104	73.955	29.9223
FashionMNIST						
Prior Precision	**0.weight**	**0.bias**	**3.weight**	**3.bias**	**7.weight**	**7.bias**
1	0.9827	0.952	0.9997	0.9976	0.9845	0.9914
0.1	2.7998	2.3673	3.1515	3.089	2.852	2.9301
0.01	6.0461	3.9018	9.6955	8.3981	6.8064	6.5646
0.001	8.6955	4.5509	26.0211	16.3853	13.6235	9.2794
0.0001	9.9294	4.8534	52.2461	29.5735	27.8975	9.9714
0.00001	10.9271	5.5394	69.1919	67.5701	67.192	10.0624

similar in every layer. Smaller prior precision results in a bigger diversity in the layer's uncertainties. Consequently, we can infer that the size of the prior precision determines its influence on the resulting uncertainty. With the gained knowledge, we decide to use a prior precision of 0.0001 for the following experiments. The level of diversity in the resulting uncertainties ensures a good quantification of the uncertainty. Besides that, varying uncertainties are better for visualization, as important characteristics become more noticeable.

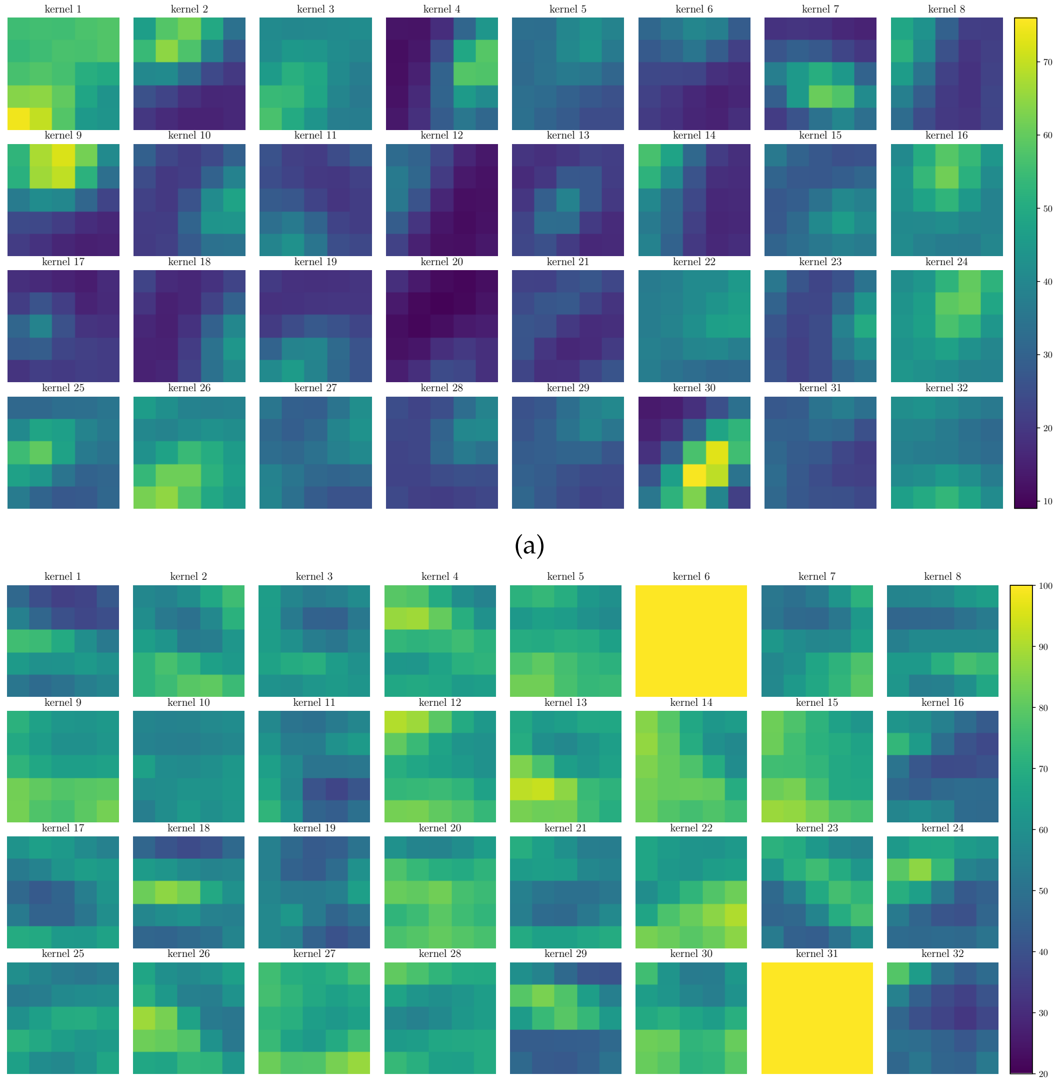
As a further result, Table 6.1 shows how the uncertainty is distributed in simpleNet. In most cases, the layers are in the same order w.r.t the size of their uncertainty. In all cases, the second layer has the highest uncertainty. The described order applies to all datasets. Hence, we can infer that the dataset does not significantly influence the uncertainty.

6.2 Visualization of the Uncertainty

This experiment aims to visualize the uncertainty in the individual layers of simpleNet. Again, we train simpleNet on MNIST and its three drop-in alternatives. We choose a prior precision of $p_0 = 0.0001$ and apply our Laplace approximation implementation to obtain the weight's uncertainties. As there are two kinds of layers, i.e. ConvLayers and linear layers, we approach their visualization differently. For ConvLayers, the visualization individually considers the uncertainty in the kernels, where every $m \times m$ kernel is represented as a heatmap. A uniform color-spectrum is used to provide good comparability between the kernels. After the last ConvLayer, flattening converts the data into a one-dimensional array that feeds into the linear layer. Therefore, the linear layer is represented as one heatmap of size 512×10. With this experiment, we can examine the visualizations for noticeable patterns, which could give insights into the characteristics of the uncertainty. Such patterns could occur within a single layer or throughout more than one layer.

6.2.1 Results: Visualization of the Uncertainty

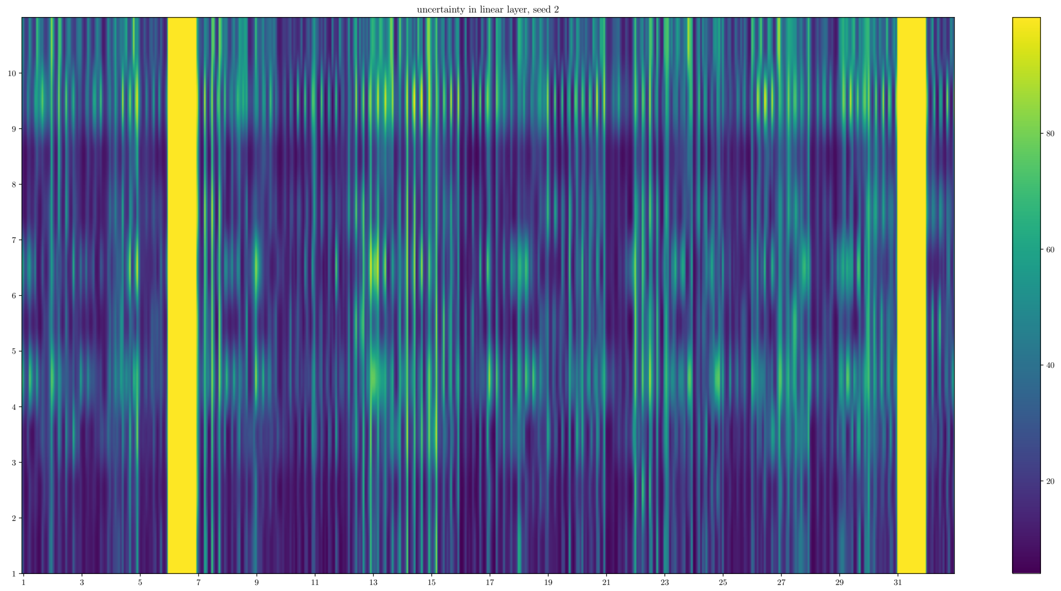
Table 6.1 already showed that the uncertainties look similar for each dataset. So do the visualizations of the uncertainty. For the sake of saving space, we will only present the results we got from simpleNet trained on MNIST for one seed.
The visualizations of the weight's uncertainties are shown in Figure 6.1. One can see, that the second ConvLayer and the linear layer share a pattern in their weight's uncertainty. Kernels 6 and 31 and the associated weights in the linear layer both have the highest uncertainty for the used prior precision of 0.0001. However, this is no characteristic of the uncertainty, but a phenomenon that is called *dying ReLU*. In ReLU networks, some gradients can cause weights to become zero. Such weights will not be activated from future gradients and are therefore called *dead* (Lu et al. 2020). Interesting about this phenomenon is that those dead weights have the maximal uncertainty. Consequently, we suggest comparing the weight's sizes with their uncertainty in the following analysis.



(a)

(b)

Figure 6.1: Uncertainty over the parameter space of simpleNet for seed 2. (a) represents the kernels in the first ConvLayer. (b) represents the kernels in the second ConvLayer.



(c)

Figure 6.1: Uncertainty over the parameter space of simpleNet for seed 2. (c) represents the uncertainty in the linear layer.

6.3 Comparing Weight-Size with Uncertainty

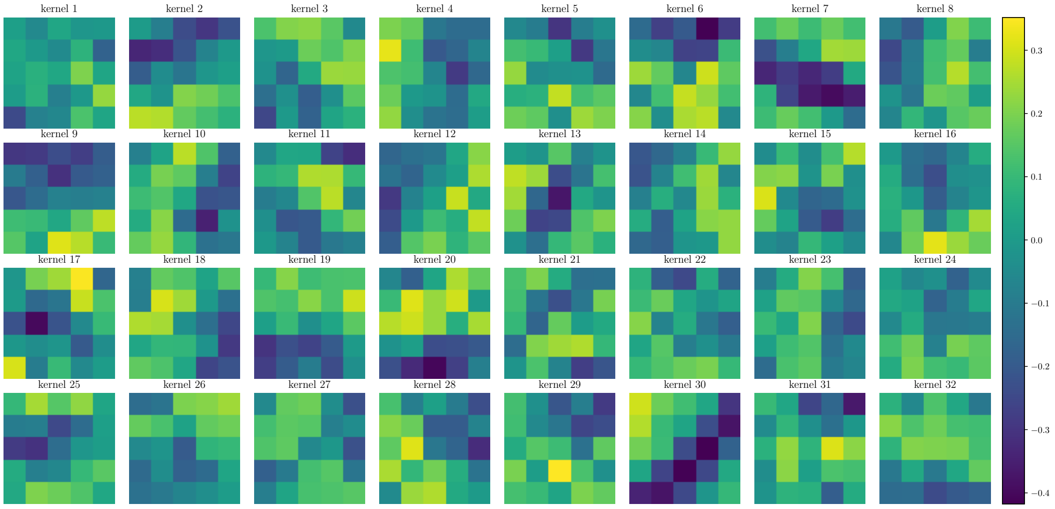
Examining the visualizations from the prior experiment suggested exploring how the size of a weight correlates with its uncertainty. This experiment visualizes the size of the weights of simpleNet in the fashion of the previous visualizations. A comparison of the visualized weights and uncertainties could reveal possible correlations. 1) Weights and associated uncertainties could show similar patterns. 2) We could get insights if the size of a weight correlates with its uncertainty.
The experiment continues with a more detailed comparison. The gathered data about weight size and uncertainty is plotted under specific aspects. Scatter plots will show the correlation between every weight's size and its uncertainty in every individual layer. Besides that, we will calculate Pearson's correlation coefficient to quantify a possible correlation. Knowing if uncertainty and weight size correlate could determine how we should treat the weights during training. If the uncertainty decreased for smaller weights, we should keep the weights small while optimizing the network. Other plots will show the distribution of the weights and the uncertainties respectively for the individual layers. We can examine those plots to see if the weight size or uncertainty is similarly distributed.

6.3.1 Results: Comparing Weight-Size with Uncertainty

The visualizations of the weights are shown in Figure 6.2. The second ConvLayer and the linear layer show the same pattern in kernels 6 and 31, similar to the visualizations of the uncertainty. The weights of said kernels have values close to zero as they were affected by the dying ReLU. A closer comparison of weight-size and uncertainty shows that they also share the same pattern besides kernels 6 and 31. This similarity indicates that there is some correlation between weight-size and uncertainty.

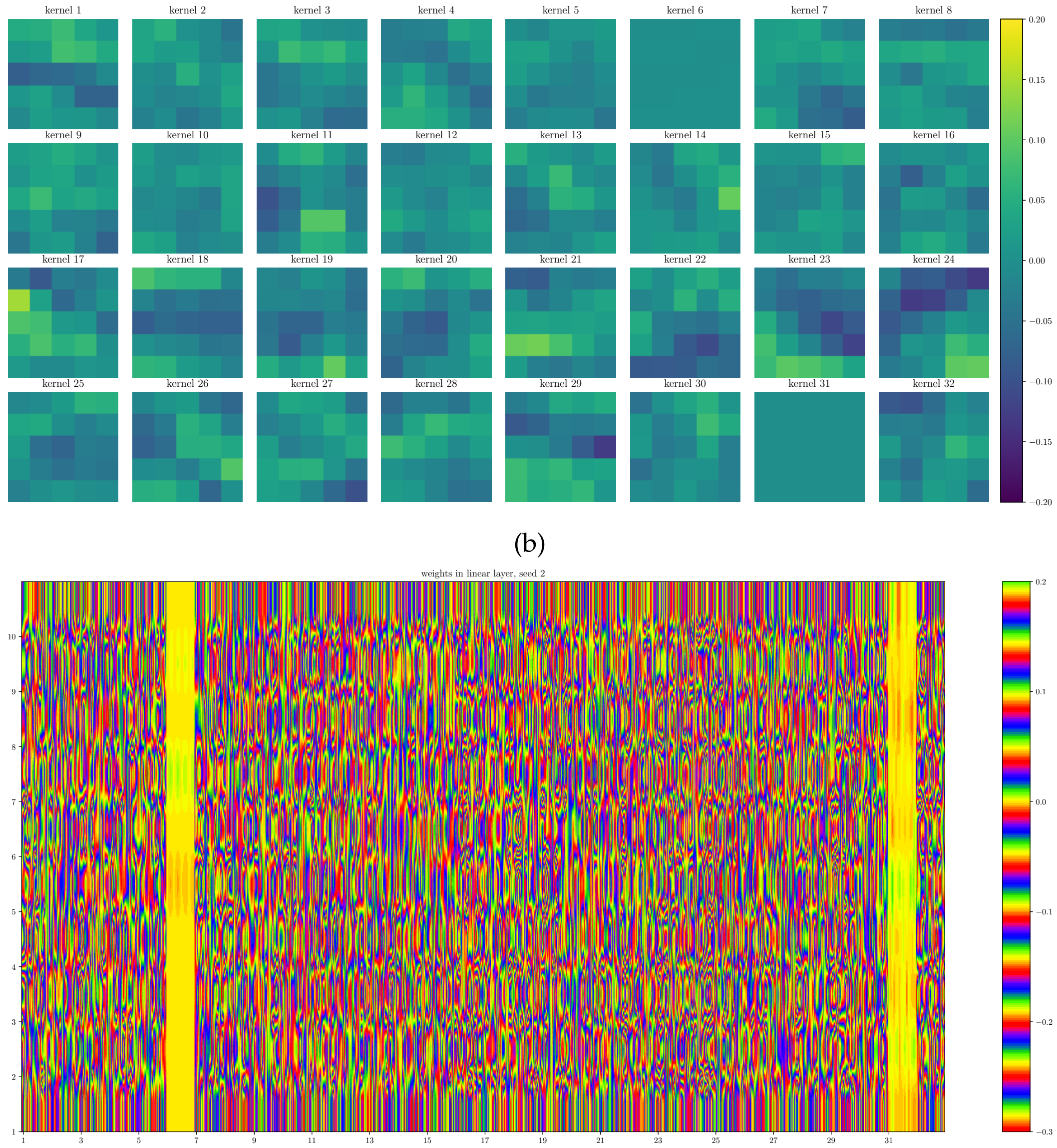
For a more detailed analysis of the shared patterns, the plots in Figure 6.3a exclusively visualizes the correlation between weight-size and uncertainty. In every layer of simpleNet, we find a negative correlation between weight-size and uncertainty.

To complete the current comparison, the distributions of uncertainty and weight-size are shown in Figure 6.7a and 6.7b. Even though the visualizations share patterns and some correlation is verified, the distributions look different. The distribution of the weight-size fits under a Gaussian. Other than that, we are not able to fit a curve over the distributions of the uncertainty.



(a)

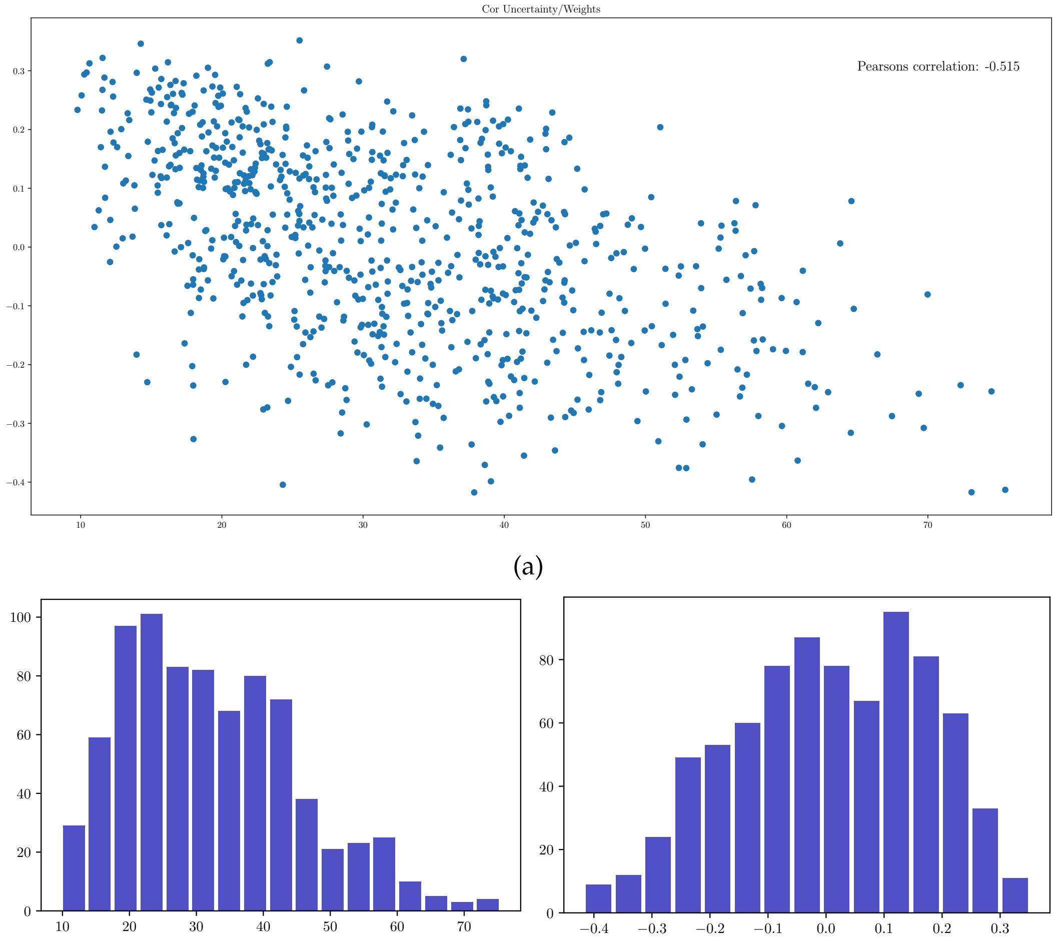
Figure 6.2: Weights of simpleNet for seed 2. (a) represents the kernels in the first ConvLayer.



(b)

(c)

Figure 6.2: Weights of simpleNet for seed 2. (b) represents the kernels in the second ConvLayer. (c) represents the uncertainty in the linear layer. We chose a different color-scheme for the latter plot, which better highlights the similarities to previous plots of the uncertainty.



(a)

(b)

(c)

Figure 6.3: Uncertainties and weights in the first ConvLayer of simpleNet: (a) shows the correlation between size of the uncertainties and size of the weights. Pearson's correlation coefficient quantifies this correlation as −0.515. (b) and (c) show the distriubtion of the uncertainty and the weights, respectively.

6.4 Examine Computation

The previous experiment found a correlation between a weight's size and its uncertainty. In this experiment, we examine how our implementation of the Laplace approximation determines the uncertainty. We define a single-layer network with one weight, i.e. a perceptron, and train it on the Toy dataset described in chapter 5.2.4. The diagonal Hessian that we approximate during the Laplace approximation has one entry, which is the curvature of the perceptron's weight. The experiment intends to see if the implementation of the Laplace approximation could cause a correlation between the size of weights and their uncertainty. Since there are no other influences besides the single weight, the perceptron is most appropriate for this experiment. For any input x, the perceptron returns $o = w \times x + b$, where w is the weight and b is the bias. The Toy dataset contains input-target-pairs of form $(x, x \cdot c + \epsilon)$. By setting different values for c, we can control the value of w, since $w \approx c$ minimizes the expected error. By comparing the weight's uncertainty with its size, we can detect if the approximation returns greater uncertainties for greater weight values.

6.4.1 Results: Examine Computation

We used several values between 1 and 200 for c and obtained very similar results for the weight's uncertainty. Hence, the experiment detects no connection between our implementation of the Laplace approximation and the negative correlation between the uncertainty and the size of a weight, like was found in the previous experiments.

6.5 Comparing Laplace Approximation with KFAC

For the simplest case of a NN, we found out that our implementation of the Laplace approximation does not yield higher uncertainty for larger weight-size. A further mathematical analysis of the Laplace approximation gets very complicated as soon as we add some more complexity to a network. Therefore, this experiment compares the uncertainties obtained by the Laplace approximation with the results from the KFAC method. We use the simpleNet trained on MNIST and approximate the Kronecker factorization with our implementation of KFAC, as it was explained in chapter 4.3. This experiment focuses on visualizing the results to make a comparison with the earlier visualizations of the uncertainty possible. For each layer, we will visualize the obtained matrices U, V, and B in individual heatmaps. From the diagonal entries of the matrix V , we can visualize the kernels of the ConvLayers and the associated weights in the linear layer. This visualization only takes one of the factorized matrices into account. Hence, this is no representation of the actual uncertainty. The stated visualization takes diagonal segments of length 5×5 and 4×4 for ConvLayers and linear layer, respectively. The segments are represented in heatmaps of the mentioned size.
By comparing the results from the Laplace approximation and KFAC, we might see similarities or differences between the methods. In particular, we want to test if we find a negative correlation between the uncertainties obtained by KFAC and the size of the associated weights.

6.5.1 Results: Comparing Laplace Approximation with KFAC

in progress ...

6.6 Comparing Uncertainties with Activations

The prior experiments already found some interesting properties of the uncertainty. In this experiment, we want to examine if the uncertainty influences how the network handles the input data. More precisely, we are interested if a kernel's uncertainty may determine the intensity of its activation for a given input. As we hold information about the weight's uncertainties in every layer, we also need to receive the convoluted features from every layer individually. To get the convoluted features from a layer λ, we remove all layers past layer λ and then perform a forward pass with an input drawn from MNIST. The convoluted features returned by each layer are then visualized as heatmaps. By comparing a kernel's uncertainty with the convoluted features it returns, we can see if the uncertainty correlates to the activation. Practitioners could exploit such a correlation to improve training methods, for instance, by focusing on kernels with small uncertainty.

6.6.1 Results: Comparing Uncertainties With Activations

For one input, the convoluted features of every layer are shown in Figure 6.4. There is no outstanding similarity when comparing the results with the visualizations of the uncertainty. As mentioned earlier, some kernels were affected by the dying ReLU. If there was a connection between the convoluted features and the uncertainties of the kernels we should see it by focusing on affected kernels. The comparison of kernels 6 and 31 with the convoluted features 6 and 31 show no similarity. We only find some convoluted features to have low activation, for which we cannot determine a connection to the dead kernels. Hence, we conclude that there is no significant correlation.

6.7 Transferring Experiments to VGG-11

The previous experiments revolved around the uncertainty in a network with simple architecture. As mentioned earlier, the next step is to transfer certain experiments to VGG-11. We train our VGG-11 on CIFAR-10 and obtain the uncertainties by applying our Laplace approximation implementation. Similar to previous experiments, we then plot the received information about the weights under different aspects. First, we visualize the kernel's weight-values and their uncertainties with heatmaps. Next, we plot the distributions of all weights and their uncertainty in all individual layer. Finally, we visualize the correlation of weight-size and weight-uncertainty in a scatter-plot for every layer. Computing Pearson's correlation coefficient provides

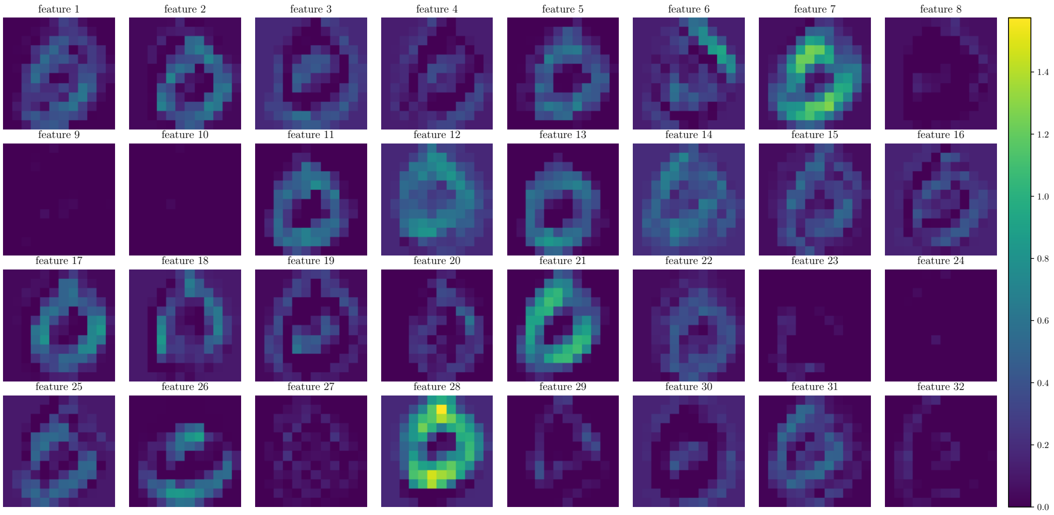
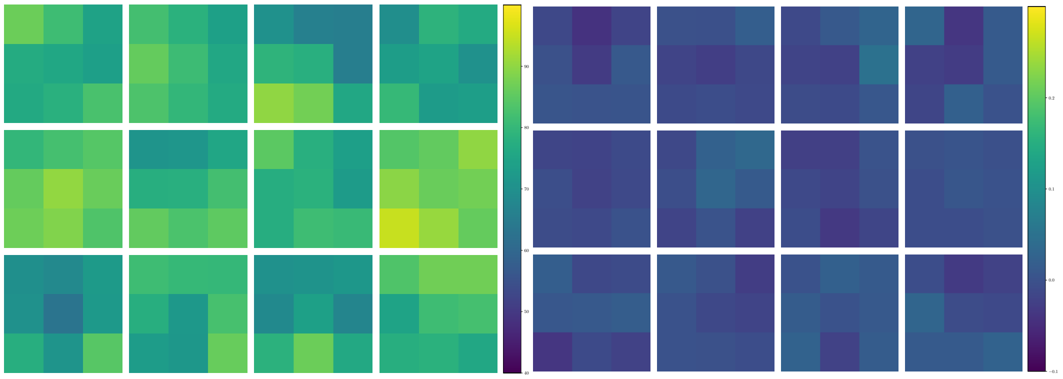


Figure 6.4: The featuremaps, which are returned from the second ConvLayer of simpleNet, given an input from the MNIST dataset.

the degree of a possible correlation in latter plots.
Similar to before, we can examine the plots for unknown aspects of uncertainty in more complex networks. Additionally, we can compare the results to the ones we received in previous experiments on simpleNet. This comparison might give insights into the behavior of uncertainty w.r.t. the network's complexity.

6.7.1 Results: Transferring Experiments to VGG-11

Figure 6.5 shows some representative examples of the uncertainty and the weight-size in the network's kernels. The results reveal that the uncertainty in VGG-11 trained on CIFAR-10 is relatively high in every layer. The high uncertainty and small weight-values in the kernels indicate a negative correlation between uncertainty and weight-size. Indeed we encounter such a negative correlation throughout the network, with a correlation coefficient between -0.015 and -0.904. Furthermore, the results reveal that the weights in every layer are normally distributed and, as in simpleNet, we are not able to assign a distribution to the uncertainties of the weights.
Compared to the previous experiments on simpleNet, the results show both similarities and differences. 1) A negative correlation between uncertainty and weight-size is discovered for both network architectures. 2) The distributions of the weights and the uncertainties show similarity. 3) A striking difference is that the more complex VGG-11 has a higher uncertainty than simpleNet. This difference gives room for follow-up experiments focusing on the influence of a network's size on its uncertainty.



(a)

(b)

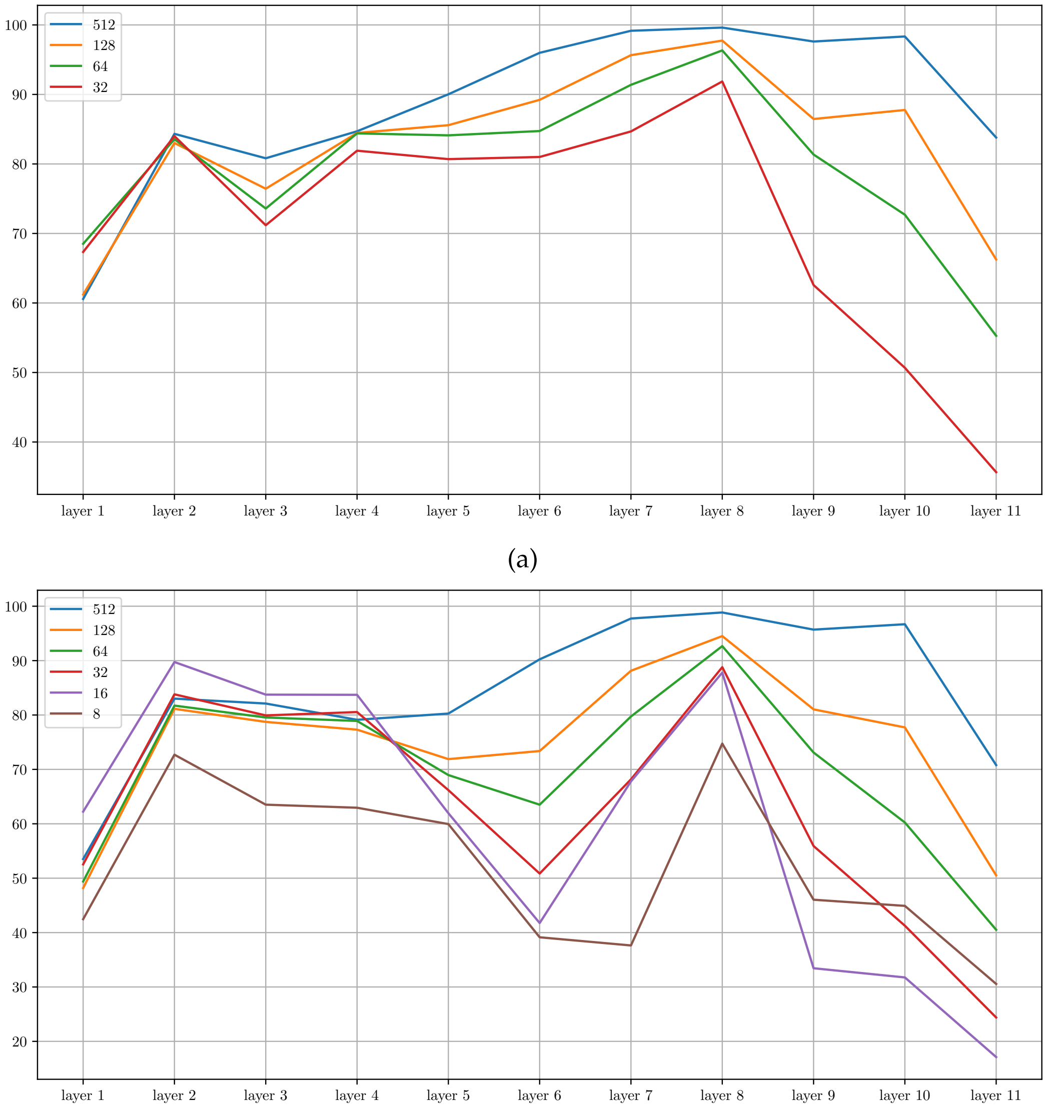
Figure 6.5: Representative excerpt of kernels in the fourth ConvLayer of VGG-11: (a) shows the uncertainty in the kernels. (b) shows the weights in the kernels.

6.8 Uncertainty in Different Width Networks

A common problem in deep learning is to find a suitable network architecture, which also takes the network's width into account. We conduct this experiment to explore how the network's width affects the uncertainty of its weights. Therefore, we successively implemented the width l for every layer past layer 5 of the VGG-11 (Table 5.2), where $2 \leq l \leq 512$. For every width, the modified network is trained on CIFAR-10 and SVHN. As before, the Laplace approximation obtains the associated uncertainties. We plot the average uncertainty per layer for every width.
This experiment might show new aspects that need to be taken into account when defining a network architecture. A practitioner may want to keep the uncertainty in the used network low. Therefore, it is helpful to know if the width of a network or an individual layer affects its weight's uncertainty.

6.8.1 Results: Uncertainty in Different Width Networks

The resulting plots from VGG-11 trained on CIFAR-10 and SVHN are shown in Figure 6.6. The uncertainties for different widths of the network show similar tendencies in every layer, as the plotted lines are moving in the same direction w.r.t. the implemented width. It seems that different widths shift the lines on the y-axis of the plots, where a smaller width determines a smaller value on the y-axis. However, the validation accuracy likewise decreases with smaller widths. To make statements about the results we need to further analyze the influence of the network's width and the validation accuracy on the uncertainty.



(a)

(b)

Figure 6.6: Uncertainties in layers of VGG-11 for different widths trained on: (a) CIFAR-10 (b) SVHN

6.9 Influence of Accuracy and Network Width on Uncertainty

The previous experiment leaves the question if the validation accuracy or the network's width determines the uncertainty of the weights in the network. This experiment makes a direct comparison of both factors possible. Similar to the

previous experiment, we train different widths of VGG-11, i.e. 512, 128, 64, 32, on CIFAR-10. During the training process, we save the network's state for different validation accuracies, namely: 60%, 70%, 80%, > 90%. For all widths and states, we use the Laplace approximation to obtain the associated uncertainties.
The experiment provides information that we can analyze under the aspects of the network's width and validation accuracy.

6.9.1 Results: Influence of Accuracy and Network Width on Uncertainty

All results of this experiment are shown numerically in Table 6.2. The validation accuracy does not correlate to the size of the uncertainty in most cases. Surprisingly, we rather find a slightly positive correlation between uncertainty and accuracy. For different widths of the network, we mostly find a positive correlation between uncertainty and width. As the latter correlation predominates, we should further examine how the width of a network influences its uncertainty. Nevertheless, we will not leave the accuracy out of consideration.

Table 6.2: Average uncertainties in layers 1-11 of VGG-11 for different widths and different validation accuracy (60% - 90%).

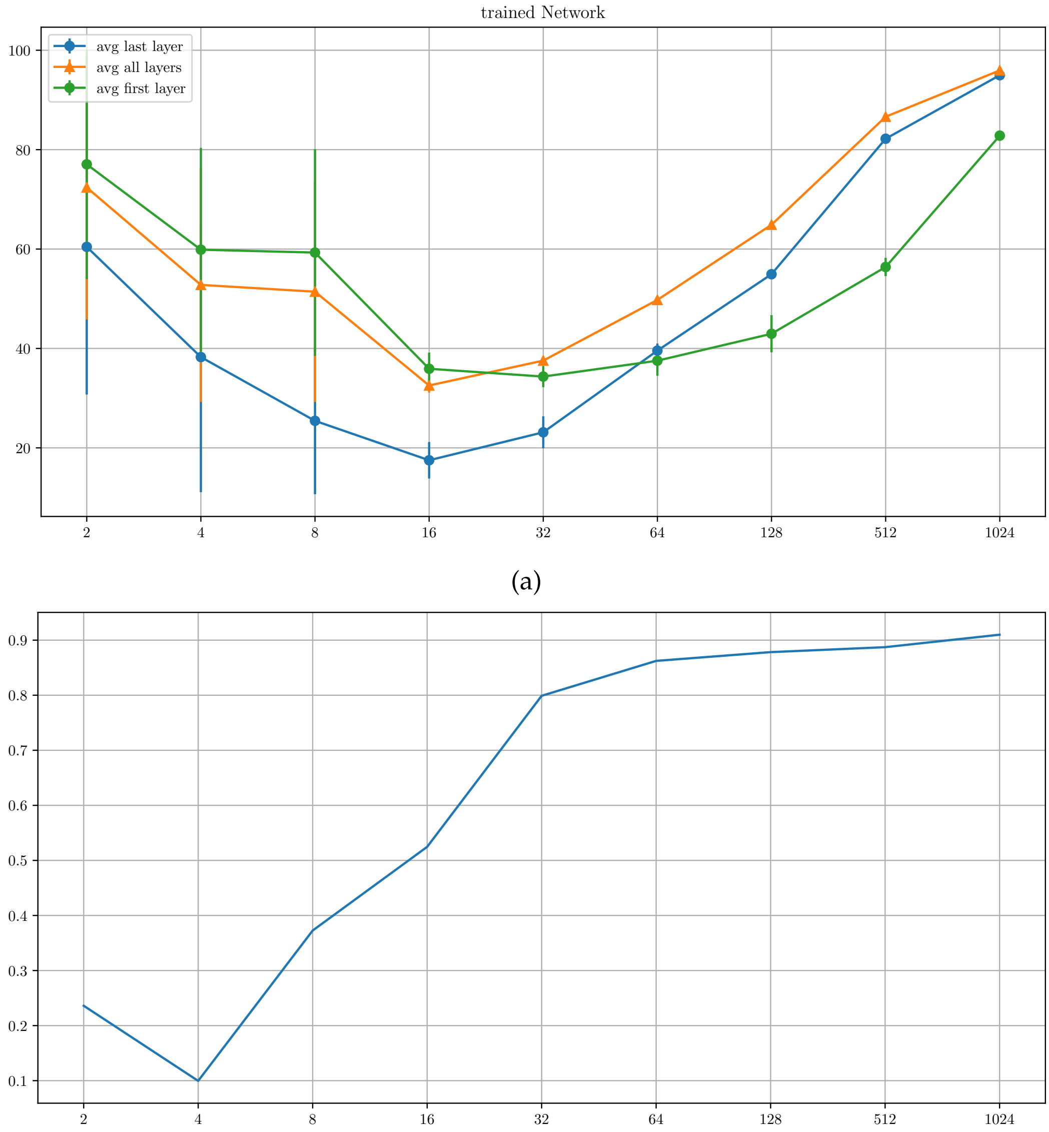
Width: 32											
Acc	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
60	60.62	84.18	81.45	86.68	75.46	64.19	78.21	90.97	55.67	50	23.40
70	60.84	83.74	78.95	87.37	78.66	70.78	79.05	90.54	56.19	55.57	25.10
80	62.03	84.20	79.48	88.74	81.71	76.40	82.72	92.46	62.20	57.40	27.58
90	59.34	80.05	69.56	81.97	80.18	79.01	84.51	93	67.57	53.56	35.44
Width: 64											
Acc	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
60	61.55	85.35	84.45	87.96	79.47	70.80	81.72	91.66	61.59	54.11	32.08
70	60.45	83.64	80.97	85.27	77.57	71.71	83.21	92.96	65.66	55.66	31.73
80	59.30	80.54	74.24	83.70	78.44	74.75	82.93	92.74	65.82	53.87	29.02
90	61.58	81.47	73.62	84.31	83.50	84.47	91.53	96.27	81.20	72.55	55.08
Width: 128											
Acc	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
60	54.56	83.99	83.90	87.02	82.83	80.69	87.64	93.18	68.03	69.40	41.70
70	55.18	82.48	80.43	84.87	79.94	79.95	88.30	94.02	70.60	68.53	42.06
80	55.98	79.08	74.22	82.00	79.61	82.05	90.26	94.54	71.29	69.01	41.75
90	59.57	81.88	75.76	84.32	85.21	89.10	95.60	97.60	86.33	87.65	66.70
Width: 512											
Acc	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
60	56.13	84.43	84.92	88.25	91.55	95.67	97.77	98.28	88.31	91.31	55.10
70	54.93	82.40	82.09	86.08	89.86	95.13	97.94	98.52	89.25	91.62	57.52
80	54.96	80.26	77.02	81.40	86.92	94.02	98.03	98.74	89.19	91.90	59.52
90	59.75	83.59	80.90	84.99	90.10	96.02	99.17	99.63	97.65	98.37	83.99

6.10 Extensive Examination of Uncertainty for Different Widths

As the previous experiment suggested, this experiment further examines how the width of a network determines its uncertainty. Besides the information we hold from previous experiments, we gather more information by using additional datasets. We train different widths of VGG-11 on MNIST and CIFAR-100 and compute the associated uncertainties with the Laplace approximation. A plot with two lines visualizes the uncertainty for each width, where one line represents the network's average uncertainty and one line the uncertainty in the network's last layer. As we do all computations for ten seeds, the mentioned lines represent the mean values and have added error bars to see how strong the values differ between the ten seeds. An additional plot shows the mean validation accuracy for every width.
To extend this experiment's information content, we also create the stated plots for simpleNet, trained on MNIST, FMNIST, EMNIST, and KMNIST. Similar to before, we use different widths, which are implemented in every layer past the first layer (Table 5.1).
The received plots are compared and individually examined for unknown characteristics of the uncertainty. Due to differences in the datasets and the use of several network configurations, we might find meaningful similarities, correlations, or differences in the resulting uncertainties. Especially by comparing the uncertainty for different widths networks with the validation accuracy, we might find a way to calibrate a neural network to have good accuracy and low uncertainty.

6.10.1 Results: Extensive Examination of Uncertainty for Different Widths

The results of the examination can be seen in Figure 6.7. A comparison of all received plots indicates one overarching tendency. The standard deviation of the obtained uncertainties, i.e. the uncertainty of the uncertainty, decreases with an increasing network's width. As mentioned earlier, the validation accuracy also improves with the use of a wider network. Hence, we cannot make a well-founded statement about the strength of the influence a network's width has on its uncertainty.
A further finding is that there are different trends in the plots associated with VGG-11 and simpleNet, respectively. The lines in the plots of VGG-11 slightly tend to first go down to a low and then go back up. However, the exact course of each line seems somewhat arbitrary, which makes it hard to extract meaningful information. The plots of simpleNet show a uniform course, where the uncertainty constantly grows with the width of the network.
In conclusion, the results indicate that there is an influence of the network's width on its uncertainty. The question still arises how to quantify the aforementioned influence. A follow-up experiment uses the setup of this experiment for further examination.



(a)

(b)

Figure 6.7: Average uncertainties for different width VGG-11 networks, trained on CIFAR 10: (a) The lines represent the average uncertainty in the last layer (blue), in the whole network (orange), and is the first two layers (green). (b) shows the mean validation accuracy for different widths.

6.11 Examination of Layers with Static Width

As mentioned, there is the possibility to extend the previous experiment. Since we did not vary the width of every layer in VGG-11, there are some layers with a static width, namely: layer 1 and layer 2 (5.2). The behavior of those static layers is what we analyze in this experiment. We integrate the average uncertainty of the stated layer into the plots created in the previous experiment. The comparison between manipulated and static layers might give hints on a certain measure of influence the width has on the uncertainty. If so, future work could try to factorize the aforementioned influence.

6.11.1 Results: Examination of Layers With Static Width

The results of the static layers can be seen in Figure 6.7. In comparison to the previous lines, the added line shows a slightly different course in the individual plots. This difference, again, speaks for a connection between a network's width and its uncertainty. However, to give a well-founded statement, the findings are still too vague.

7 Conclusion

The research in this thesis provides an exploratory analysis of the uncertainty in the weights of a BNN. Visualizations of the uncertainty support this process throughout the thesis. The analysis took two network architectures and two methods to perform approximate inference into account. One of those methods, the Laplace approximation, was chiefly used to get information about the uncertainty. The two network architectures are an entry-level CNN and a state-of-the-art CNN by name simpleNet and VGG-11. Starting with a simpleNet ensured both a good lead-in and good intuition. We understood how the Laplace approximation functions and examined present connections of the weight's states and their uncertainties. The second method to perform approximate inference, KFAC, was explored briefly, and the obtained uncertainties from both methods were compared. With moving to VGG-11, we classified state-of-the-art datasets, such as CIFAR-10. Transferring previous analysis to VGG-11 gave insights into the influence of a network's complexity on its uncertainty. With the increase in complexity, we went into a more detailed analysis of the uncertainty. Indications for the influence of a layer's width on its uncertainty were found. We followed those indications in the further examination.
The described procedure included many experiments, of which the results yield several conclusions. Performing the Laplace approximation with different prior precision showed that larger prior precision (e.g. >1, 0.1) strongly determines the uncertainty. For smaller prior precisions, the uncertainty has more freedom and is more meaningful. Seeing the visualizations of the uncertainty in figure x gave hints toward a negative correlation between the size of the uncertainty and the weight-size. This negative correlation was confirmed for both CNN architectures in several scatterplots (Figure x, y, z). With a perceptron and a Toy dataset, it was checked whether the implemented Laplace approximation could cause such a negative correlation. Since the uncertainty was steady for different weight sizes, no such connection was found. A comparison between the results of Laplace approximation and KFAC showed xy (tbd). The visualizations of the uncertainty in VGG-11 (Figure x) revealed that the uncertainties are higher than in simpleNet. This finding motivated to do a further analysis of a possible connection between network complexity and uncertainty. Several experiments with different width networks indicated such a connection. One experiment found that smaller width layers tend to have smaller uncertainties. However, the validation accuracy of a network also decreases for smaller widths, which prohibits a hasty conclusion. Table x showed numerically that there is a positive correlation between accuracy and uncertainty, as well as between the width of a layer and its uncertainty. The numbers prove that the latter

correlation is stronger and more frequent. Subsequent experiments with different width networks gave further indications for a specific influence of a network's width on its uncertainty. However, the results were imprecise and only gave hints. Therefore, no well-founded conclusions can be given, whereas there is scope for future work.

7.1 Future Outlook

This work found several indications of an influence of a network's complexity, i.e. its width, on the resulting uncertainty. If a practitioner has a clear factorization of the mentioned influence, they could use it to find a suitable network architecture or to calibrate a network. This work found several indications that the complexity of a network scales its uncertainty. Motivated from those indications, future work could build up on the findings of this work to prove mathematically or exploratory how the influence of a network's complexity could be extracted from its uncertainty.

Bibliography

: n.d.

Clanuwat, T., Bober-Irizar, M., Kitamoto, A., Lamb, A., Yamamoto, K. and Ha, D.: n.d., Deep learning for classical japanese literature.

Cohen, G., Afshar, S., Tapson, J. and van Schaik, A.: 2017, EMNIST: an extension of MNIST to handwritten letters, *CoRR* **abs/1702.05373**.

Dangel, F., Kunstner, F. and Hennig, P.: 2020, Backpack: Packing more into backprop, *International Conference on Learning Representations*.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. and Fei-Fei, L.: 2009, ImageNet: A Large-Scale Hierarchical Image Database, *CVPR09*.

Fukushima and Kunihiko: n.d., Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position.

Gal, Y. and Ghahramani, Z.: 2016, Dropout as a bayesian approximation: Representing model uncertainty in deep learning.

Hein, M., Andriushchenko, M. and Bitterwolf, J.: 2019, Why relu networks yield high-confidence predictions far away from the training data and how to mitigate the problem.

Ke, Q., Liu, J., Bennamoun, M., An, S., Sohel, F. and Boussaid, F.: 2018, *Computer Vision for Human-Machine Interaction*.

Krizhevsky, A.: 2012, Learning multiple layers of features from tiny images.

Lakshminarayanan, B., Pritzel, A. and Blundell, C.: 2017, Simple and scalable predictive uncertainty estimation using deep ensembles, *in* I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett (eds), *Advances in Neural Information Processing Systems 30*, pp. 6402–6413.

Lu, L., Shin, Y., Su, Y. and Karniadakis, G. E.: 2020, Dying relu and initialization: Theory and numerical examples.

MacKay, D. J. C.: 1992, A practical bayesian framework for backpropagation networks, *Neural Computation* **4**(3), 448–472.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B. and Ng, A.: 2011, Reading digits in natural images with unsupervised feature learning.

Patterson, J. and Gibson, A.: 2017, *Deep Learning: A Practitioner's Approach*.

Ritter, H., Botev, A. and Barber, D.: 2018, A scalable laplace approximation for neural networks, *International Conference on Learning Representations*.

Rosenblatt, F.: 1958, The perceptron: A probabilistic model for information storage and organization in the brain, *Psychological Review* pp. 65–386.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K. and Hassabis, D.: 2017, Mastering chess and shogi by self-play with a general reinforcement learning algorithm.

Silvestro, D. and Andermann, T.: 2020, Prior choice affects ability of bayesian neural networks to identify unknowns.

Simonyan, K. and Zisserman, A.: 2014, Very deep convolutional networks for large-scale image recognition.

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.

Wu, Y., Zhu, X., Wu, C., Wang, A. and Ge, R.: 2020, Dissecting hessian: Understanding common structure of hessian in neural networks.

Xiao, H., Rasul, K. and Vollgraf, R.: 2017, Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.