5. Experiments: Setup

This work conducts several experiments to understand and analyze the uncertainty in deep learning models. Due to the variety in the scope of deep learning, we need to delimit this work by focusing on one network class. We chose to use CNNs for image classification. The reason for that is that classification generally pleases to incorporate a measure of uncertainty. Also, the realm of image classification is a significant field of deep learning.
This chapter introduces the setup we use for the experiments. First, we will briefly describe how we structure the experiments. After that, all involved NNs will be introduced by their architecture and specification. Next, the selection of datasets, which we access during the experiments is introduced. Besides the summary of every dataset is an excerpt of its contents to provide a good intuition. Lastly, the implementations of Laplace Approximation and K-FAC are explained.

5.1. Procedure

We divide the conducted experiments in this work into two phases. Each of those phases mainly revolves around one network architecture. We begin with a simple CNN in the first phase and move to a more complex CNN in the second phase. This division provides an appropriate lead-in to understand and analyze the uncertainty in deep learning models. The gained understanding can then be transferred to the more complex model. The stated structure is implemented by describing the experiments and the related results for both phases successively.

5.2. Networks

5.2.1. Simple CNN

We have some requirements for the CNN we use in the first phase. It needs to be simple but also able to fit nicely on a standard benchmark dataset. This minimum of required complexity ensures that our results have a reasonable basis. Since we work with CNNs for image classification, the first network should generalize well on the MNIST dataset.
We came up with 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 5.1 and Table 5.1, respectively.

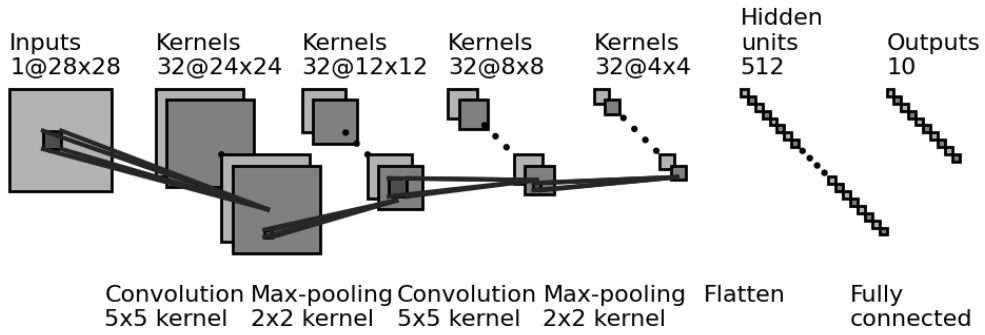

Figure 5.1.: Schematic structure of simpleNet for a 28x28 input with 10 classes, such as MNIST.

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**
1	Conv2d	(1, 32, 5)
	ReLU	-
	MaxPool2d	(2, 2)
2	Conv2d	(32, 32, 5)
	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	

5.2.2. VGG-11

To make possible findings comparable to state-of-the-art deep learning models, we need a CNN architecture capable of achieving state-of-the-art results. For image-classification, the VGG [15] architecture satisfies this requirement as it achieves

decent results on ImageNet [5]. A VGG architecture is defined by its depth. 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. The detailed specifications are listed in table 5.2.

5.3. Datasets

5.3.1. MNIST Dataset

The Modified National Institute of Standards and Technology database [8] 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 testing images. Each image has a normalized 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 x shows a set of images contained in 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) [3], Fashion-MNIST (FMNIST)[18], and Kuzushiji-MNIST (KMNIST)[2]. Those alternative datasets share specifications with MNIST but incorporate different content. EMNIST contains images of handwritten characters. FMNIST, developed by Zalando, picture various pieces of clothing. KMNIST images show phonetic letters of hiragana, a Japanese syllabary. A sample of each dataset is shown in Figure x.

5.3.2. SVHN

The Street View House Numbers (SVHN) dataset [10] 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 lack of contrast normalization, overlapping digits, and distracting features. SVHN includes 73257 training and 26032 testing images (32×32 pixels) with an additional 531131 more simplistic images to extend the training set. The pictures in SVHN show real-world photographs of house numbers, which differ 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 x.

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).

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

5.3.3. CIFAR-10 and CIFAR-100

The Canadian Institute For Advanced Research (CIFAR) created two established datasets for object recognition and image classification tasks [7]. 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 testing 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 deers. 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.

5.3.4. Iris Dataset

The Iris Dataset is often used in classification and clustering tasks. It holds four attributes of three different iris-flowers: Iris setosa, Iris virginica, and Iris versicolor. The features that set the flowers apart are the length and width of petal and sepal. The usual classification task should identify the species from given attributes. Table x shows one set of features for each species.

5.3.5. Toy Dataset

We defined a Toy Dataset to use a perceptron as a linear binary classifier. The dataset consists of 5000 input-target pairs $x, y = (x, x \times c + \epsilon)$, 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.

5.4. Obtaining The Uncertainty

For NNs, where $\mathbf{w}$ are the networks' weights, approximate inference provides us with a posterior distribution $p(\mathbf{w}|D)$. Practitioners may use the posterior distribution to get predictive uncertainty. In this work, we focus on the uncertainty within a networks' weight space, i.e. the posterior distribution of every weight $w_i \in \mathbf{w}$. Two methods are used to obtain the weights posterior distributions: The Laplace

Approximation of the Weights and K-FAC, both introduced in chapter 4.

5.4.1. Implementation of The Laplace Approximation

Chapter 4.1 explained the theory of the Laplace approximation. However, we are not exactly interested in the networks' posterior distribution obtained by the Laplace approximation. 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 Hessian $\mathbf{H}_\lambda$ 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 weights uncertainty. Therefore we applied the square-root to every entry in the approximated matrix $diagonal(\mathbf{H}_\lambda^{-1})$. 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.

5.4.2. Implementation of K-FAC

– not refined yet –

6. Experiments: Phase I

6.1. Experiments

Phase I of experiments revolves around simpleNet. We use MNIST, FashionMNIST, KMNIST, and EMNIST to train simpleNet. Training the network on ten different seeds regularizes the randomness in weight space and training data. The results of different seeds are either compared or averaged.

6.1.1. Influence of The Prior Precision

Choosing an appropriate prior for the weights of a NN is discussed in numerous research papers, like Silvestro und Andermann [14]. In this first experiment, we want to determine how different prior precision influence the weights' resulting uncertainty. SimpleNet is trained on the datasets mentioned above. We define a geometric series of prior precision $p_0 \in \{1, 0.1, 0.01, 0.001, 0.0001, 0.00001\}$, for which we want to evaluate the trained network's uncertainty. Our Laplace approximation implementation obtains the standard deviations 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 highly 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 if the prior precision might influence 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.

6.1.2. Visualization of The Uncertainty

This experiment aims to visualize the uncertainty for individual layers of simpleNet. Again, we train simpleNet on all available datasets mentioned above. We choose a prior precision $p_0 = 0.0001$ and invoke our Laplace approximation implementation to obtain the weights' standard deviations. Different approaches are used to visualize the uncertainty in the different layers, i.e. ConvLayers and linear layers. For ConvLayers, we visualize the uncertainty of the kernels. Every $m \times m$ kernel is represented as a heat-map. A uniform color-spectrum is used to provide good comparability between the kernels. The linear layer is represented in a single heat-map of size [10, 512].

Since the weights are flattened, a single heat-map is a suitable representation. With this experiment, we can examine the visualizations for interesting patterns. Such patterns could occur within a single layer or throughout more than one layer.

6.1.3. Comparison of Weights and Uncertainty

The prior experiment provided visualizations of the uncertainty of the weights. This experiment identically visualizes the size of the associated weights of simpleNet. 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.
This experiment extends with a more detailed comparison. The gathered data about weight size and uncertainty is plotted for specific interests. Scatter plots will show the correlation between every weight's size and its uncertainty for 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 would decrease 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 weight size or uncertainty show similar structure.

6.1.4. Examine Computation

The previous experiment studied the correlation between a weights size and its uncertainty. In this experiment, we examine how our implementation of the Laplace approximation determines the uncertainty. We start the procedure by defining a single-layer network with one weight, i.e. a perceptron. To train this perceptron, we use the Toy dataset described in chapter x. Like earlier, we use our Laplace approximation implementation to obtain the standard deviation of the weight. For this most simplistic case of a network, we also calculated the diagonal Hessian manually to gain intuition. The experiment intends to see if the computation causes a possible correlation between the size of weights and its uncertainty. Since there are no other influences besides the single weight, we use the perceptron in 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 $(x, x \times c + noise)$. Consequently, we can control the value of w, since $w \approx c$ minimizes the expected error. Observing different values for c makes it possible to quantify the weight size's influence on its uncertainty in the simplest case.

6.2. Results

–not refined yet–

Bibliography

[1] Agarwal, Sameer ; Furukawa, Yasutaka ; Snavely, Noah ; Simon, Ian ; Curless, Brian ; Seitz, Steven M. ; Szeliski, Richard: Building Rome in a day. In: *Commun. ACM* 54 (2011), Oktober, Nr. 10, S. 105–112. – URL http://doi.acm.org/10.1145/2001269.2001293. – ISSN 0001-0782

[2] Clanuwat, Tarin ; Bober-Irizar, Mikel ; Kitamoto, Asanobu ; Lamb, Alex ; Yamamoto, Kazuaki ; Ha, David: Deep Learning for Classical Japanese Literature. In: *CoRR* abs/1812.01718 (2018). – URL http://arxiv.org/abs/1812.01718

[3] Cohen, Gregory ; Afshar, Saeed ; Tapson, Jonathan ; Schaik, André van: EMNIST: an extension of MNIST to handwritten letters. In: *CoRR* abs/1702.05373 (2017). – URL http://arxiv.org/abs/1702.05373

[4] Dangel, Felix ; Kunstner, Frederik ; Hennig, Philipp: BackPACK: Packing more into Backprop. In: *International Conference on Learning Representations*, URL https://openreview.net/forum?id=BJlrF24twB, 2020

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

[6] Gal, Yarin ; Ghahramani, Zoubin: *Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning*. 2016

[7] Krizhevsky, Alex: Learning Multiple Layers of Features from Tiny Images. In: *University of Toronto* (2012), 05

[8] LeCun, Yann ; Cortes, Corinna: MNIST handwritten digit database. (2010). – URL http://yann.lecun.com/exdb/mnist/

[9] MacKay, David J. C.: A Practical Bayesian Framework for Backpropagation Networks. In: *Neural Computation* 4 (1992), Nr. 3, S. 448–472. – URL https://doi.org/10.1162/neco.1992.4.3.448

[10] Netzer, Yuval ; Wang, Tao ; Coates, Adam ; Bissacco, Alessandro ; Wu, Bo ; Ng, Andrew: Reading Digits in Natural Images with Unsupervised Feature Learning. In: *NIPS* (2011), 01

[11] Patterson, Josh ; Gibson, Adam: *Deep Learning: A Practitioner's Approach*. 1st. O'Reilly Media, Inc., 2017. – ISBN 1491914254

[12] Ritter, Hippolyt ; Botev, Aleksandar ; Barber, David: A Scalable Laplace Approximation for Neural Networks. In: *International Conference on Learning Representations*, URL https://openreview.net/forum?id=Skdvd2xAZ, 2018

[13] Rosenblatt, F.: The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain. In: *Psychological Review* (1958), S. 65–386

[14] Silvestro, Daniele ; Andermann, Tobias: Prior choice affects ability of Bayesian neural networks to identify unknowns. (2020), 05

[15] Simonyan, Karen ; Zisserman, Andrew: Very Deep Convolutional Networks for Large-Scale Image Recognition. In: *arXiv 1409.1556* (2014), 09

[16] Srivastava, Nitish ; Hinton, Geoffrey ; Krizhevsky, Alex ; Sutskever, Ilya ; Salakhutdinov, Ruslan: Dropout: A Simple Way to Prevent Neural Networks from Overfitting. In: *Journal of Machine Learning Research* 15 (2014), Nr. 56, S. 1929–1958. – URL http://jmlr.org/papers/v15/srivastava14a.html

[17] Wu, Yikai ; Zhu, Xingyu ; Wu, Chenwei ; Wang, Annie ; Ge, Rong: *Dissecting Hessian: Understanding Common Structure of Hessian in Neural Networks*. 2020

[18] Xiao, Han ; Rasul, Kashif ; Vollgraf, Roland: *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. 2017