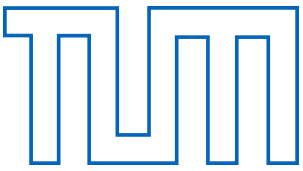


Technical University of Munich

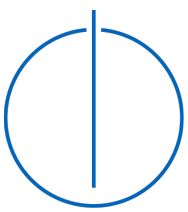
Department of Informatics

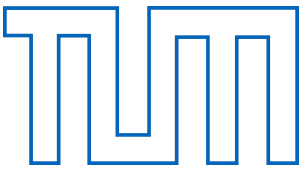
Master's Thesis in

Robotics, Cognition, Intelligence

Laplace Approximation for Uncertainty Estimation of Deep Neural Networks

Matthias Humt





Technical University of Munich

Department of Informatics

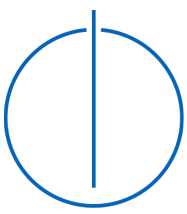
Master's Thesis in

Robotics, Cognition, Intelligence

Laplace Approximation for Uncertainty Estimation of Deep Neural Networks

Laplace Annäherung zur Unsicherheitsschätzung tiefer neuronaler Netzwerke

Author:	Matthias Humt
Professor:	PD Dr. habil. Rudolph Triebel
Supervisor:	Lee Jongseok
Date:	July 15, 2019



I confirm that this master's thesis is my own work and I have documented all sources and material used.

Germany | Munich – July 15, 2019

Matthias Humt

Abstract

Deep learning is at the frontier of machine learning and automation, using deep neural networks as the main workhorse, which have revolutionized the way we extract information from large amounts of data in computer vision, natural language processing and other domains. Moving from purely academic to real world scenarios has renewed interest into the way those powerful algorithms draw their conclusions and how to quantify the quality of their predictions beyond mere accuracy.

From a practitioner's point of view, the most important information to obtain alongside the algorithm's prediction is the uncertainty attached to it, which is the basis for an accurate assessment of confidence. Unfortunately, extracting this information from large models and datasets has proven to be difficult. A common approach so far has been to devise a method that makes as many approximations as necessary to render the problem tractable while still yielding at least somewhat useful uncertainty estimates.

This work looks at the problem from a slightly different angle: By first choosing a tractable and comparatively simple method, the burden then lies on the model design to lend itself to the chosen approximation. To this end, the most popular deep neural network architectures are compared based on their compliance to uncertainty estimation by Laplace approximation, assessing empirically the methods potentials and deficiencies as well as its applicability to large models and datasets while working towards an understanding how architectural choices correlate with the quality of obtained uncertainty estimates.

Keywords — Machine Learning, Deep Neural Networks, Uncertainty Estimation, Laplace Approximation

Table of contents

List of Figures

List of Tables

Nomenclature

Notation

$\boldsymbol{A}$	matrix
$\boldsymbol{b}$	vector
a	scalar
N	Number of data points in the dataset
D	Dimensionality of the data
K	Number of classes
$\boldsymbol{x}_i \in \mathbb{R}^D$	Single observation
$\boldsymbol{y}_i \in \mathbb{R}^K$	Single target
$(\boldsymbol{x}_i, \boldsymbol{y}_i)$	Single data point (observation, target pair)
$\boldsymbol{X} \in \mathbb{R}^{N \times D}$	Observations
$\boldsymbol{Y} \in \mathbb{R}^{N \times K}$	Targets
$\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N = \boldsymbol{X}, \boldsymbol{Y}$	Dataset
$\hat{\boldsymbol{y}}_i$	Model output for observation $\boldsymbol{x}_i$; Predicted class probabilities
$\hat{y}_i$	Predicted target for observation $\boldsymbol{x}_i$
$\boldsymbol{W}$	Weight matrix
$E(\boldsymbol{W})$	Error for a single data point
$\mathcal{L}(\boldsymbol{W})$	Loss: Average error of the entire dataset
$\mathcal{N}$	The normal distribution
ln	The natural logarithm
log	The logarithm with base 10

Acronyms & Abbreviations

AI	Artificial intelligence	ECE	Expected calibration error
ML	Machine learning	MCE	Maximum calibration error
NN	Neural network	JSD	Jensen-Shannon distance
MLP	Multi-layer perceptron	EMA	Exponential moving average
DNN	Deep neural network	ECDF	Empirical cumulative distribution function
BNN	Bayesian neural network	pdf	Probability density function
MSE	Mean-squared error	pmf	Probability mass function
GNN	Generalized Gauss-Newton	p.s.d.	Positive semi-definite
MC	Monte Carlo	w.r.t.	With respect to
MCMC	Markov chain Monte Carlo	i.e.	It est ("it is")
VI	Variational inference	e.g.	Exempli gratia ("for the sake of an example")
MAP	Maximum a Posteriori	s.t.	Such that
MLE	Maximum likelihood estimate		
OOD	Out of distribution		
FGSM	Fast gradient sign method		
KL	Kullback-Leibler		
NLL	Negative log likelihood		
TPE	Tree-structured Parzen estimator		
GP	Gaussian process		
RF	Random forest		
VB	Variational Bayes		
ACE	Average calibration error		

1 Introduction

1.1 Problem statement

With the growing field of *Artificial Intelligence* (AI) and *Machine Learning* (ML) as one of its most active branches in the form of Deep Learning (LeCun et al., 2015), a shift from purely academic to real world applications is taking place (Hinton et al., 2012; Krizhevsky et al., 2012; Mikolov et al., 2013).

From this, new needs arise in the form of interpretable algorithmic decision making and an accurate assessment of the confidence in an algorithms' predictions.

And while some great research to this end has been conducted (Carter et al., 2019; Olah et al., 2017, 2018; Zeiler and Fergus, 2014), the most iconic algorithm in use remains somewhat inscrutable: The *deep neural network* (DNN). Due to the complex structure with millions of parameters and nested non-linear transformations, it remains notoriously difficult to gain an intuitive understanding of what a trained network has actually learned and how the gained knowledge is being represented.

Furthermore, the output of such algorithms is usually a point estimate, which does not provide the often crucial additional uncertainty information from which confidence can be inferred. It has been observed that DNNs are often miscalibrated (Guo et al., 2017), assigning erroneously high confidence to wrong predictions as well as being underconfident in regions of excellent performance, making them unreliable and unpredictable, which are undesirable properties in a safety-critical environment like autonomous driving or for medical diagnosis. In other words, the algorithm should have introspective qualities (Grimmett et al., 2016), knowing what it does and does not know, which all too often is not the case.

Additionally, many network architectures are overcomplex for the task at hand, making them too slow for real-time applications like robotics and increasing energy consumption as well as computational demands unnecessarily. At the same time they are harder to train and require larger datasets to do so (Iandola et al., 2016).

While we are still far from fully understanding knowledge representation in DNNs, the renewed interest into Bayesian methods for deep learning has provided us at least with the tools to obtain uncertainty information from such models, which goes a long way to mitigate the aforementioned problems (Federici et al.,

2017; Louizos et al., 2017).

Still, the devised methods are in part too complex or computational intensive or simply not accurate enough to have gained widespread adoption among practitioners (Gal, 2016). For the deepest networks and largest datasets, scalable methods became available only very recently and comparatively little research have been directed to this domain.

1.2 Structure and contribution

The main contribution of this work is twofold. Firstly, the scalability of Laplace approximation to a variety of popular state-of-the-art DNN architectures in conjunction with very large datasets is demonstrated, making use of *Bayesian optimization* (BO) for hyperparameter search. This allows to obtain predictive uncertainty estimates that are then shown empirically to improve performance on a variety of tasks like calibration, *out-of-distribution* (OOD) detection and increased robustness to adversarial attacks.

Secondly, the influence of the architecture on the shape of the network's weight posterior distribution is studied. By analyzing the eigenvalues of the obtained curvature matrices in conjunction with high resolution one- and two-dimensional visualizations of the model's loss surfaces, we establish first insights into which architectural choices justify a multivariate normal posterior approximation and which do not.

The next section introduces some essential concepts from probability theory to allow us to then look at the *neural network* NN from a Bayesian point of view. Existing methods for uncertainty estimation with varying degrees of practicality and scaleability are contrasted to arrive at one of the most promising approaches for DNNs: Laplace approximation. Afterwards we put the method to the test, first introducing the experimental setup on a small NN and dataset to then move on to the deep network architectures, discussing results and insights along the way where applicable and referring further discussion to its dedicated section. Finally, a conclusion is drawn and an outlook for future work is given.

2 Background

2.1 Probability and information

This thesis is based on concepts from linear algebra, probability theory and information theory as well as ML basics and NNs. While none of these fields can be covered here in any length, some of the most important concepts will be highlighted, also introducing the notation, which will be used throughout the work[1].

Probability theory is the mathematical toolbox to reason in the presence of uncertainty. Specifically, it allows to quantify our degree of belief in the outcome of an event. We thereby adopt the Bayesian view on probability throughout this work as opposed to the frequentist view, the latter treating probabilities as an inherent property of the objects being studied, which manifest themselves as long run frequencies of the outcomes (Murphy, 2012, p. 27). A good introduction to Bayesian probability theory and its differences compared to frequentist statistics can be found in Jaynes (1986) and a general and more recent introduction to probability theory in Sheldon et al. (2002).

Working with uncertainty means to first cast all quantities of which we lack perfect knowledge as probability distributions and then to quantify the amount of uncertainty they inherit using information theory, where likely events have lower information content than less likely ones (Goodfellow et al., 2016, p. 71).

To update our beliefs in light of new information we can make use of Bayes' theorem (Bishop, 2006, p. 22).

$$p(\theta \mid \mathcal{D}, \mathscr{H}) = \frac{p(\mathcal{D} \mid \theta, \mathscr{H}) p(\theta \mid \mathscr{H})}{p(\mathcal{D} \mid \mathscr{H})} \tag{2.1}$$

This reads: the posterior distribution over the parameters θ – which models the most likely and less likely parameters in light of our observations – conditioned on the data $\mathcal{D}$ and the hypothesis $\mathscr{H}$ – which subsumes the choice of model and all other assumptions – is the product of the data likelihood and parameter prior divided by the evidence.

[1] For an in-depth treatment of all these topics, see standard textbooks on ML like Bishop (2006), Murphy (2012), Barber (2012) or Goodfellow et al. (2016).

For a uniform prior $p(\theta \mid \mathscr{H})$ and because the evidence $p(\mathcal{D} \mid \mathscr{H})$ is constant, the posterior distribution $p(\theta \mid \mathcal{D}, \mathscr{H})$ is proportional to the likelihood $p(\mathcal{D} \mid \theta, \mathscr{H})$ (Bishop, 2006, p. 22; Barber, 2012, p. 184). This is important to keep in mind when we later discuss all results only in light of the posterior distribution while being equally applicable to the likelihood. If we want to predict the outcome of an event, we first need to perform inference i.e. find the model parameters that best explain the data. This can be as simple as finding the mode of the posterior or likelihood distribution resulting in the *maximum a posteriori* (MAP) and *maximum likelihood estimates* (MLE) respectively (Barber, 2012, p. 184). Those are point estimates however, which do not provide any information about the shape of their distributions. Full Bayesian inference on the other hand considers all possible parameter constellations resulting in a predictive distribution, but requires marginalizing over the parameters, i.e. solving the integral

$$p(\boldsymbol{y}^\star \mid \boldsymbol{x}^\star, \mathcal{D}) = \int p(\boldsymbol{y}^\star \mid \boldsymbol{x}^\star, \theta) p(\theta \mid \mathcal{D}) \mathrm{d}\theta \tag{2.2}$$

The result is the distribution over targets called the posterior predictive distribution. Here, $\boldsymbol{x}^\star$ is a new unseen observation for which we would like to predict the correct target[2] $\boldsymbol{y}^\star$ and θ are the model parameters. the most likely target is then evaluated as (Bishop, 2006, p. 279)

$$\hat{y}^\star = \arg\max p(\boldsymbol{y}^\star \mid \boldsymbol{x}^\star, \mathcal{D}) \tag{2.3}$$

.

2.2 Neural networks and deep learning

As NNs are of central importance to this thesis, we will review the most important aspects here[3]. Those are the structure, the training objective, the training procedure, inference and application as well as a general framework for uncertainty estimation in deep learning: The *Bayesian neural network* (BNN).

NNs find ubiquitous use in a wide variety of domains whenever there are complex non-linear connections and patterns to be found in the data, which are too complex to encode by hand or even too complex to be found by a human. Among them are

[2]Using a one-hot encoding for the K classes (Murphy, 2012, p. 35).

[3]This section is based on material from Bishop (2006); Goodfellow et al. (2016); Murphy (2012). For a good conceptual overview with focus on intuitive understanding through visualization, see this superb video series on NNs `https://www.3blue1brown.com/neural-networks`

speech recognition (Hannun et al., 2014; Hinton et al., 2012), image classification, detection and recognition (Krizhevsky et al., 2012; Wu et al., 2015) as well as natural language processing (Collobert and Weston, 2008; Sutskever et al., 2014).

2.2.1 Structure

A NN is a non-linear mapping from input $\boldsymbol{x}$ to (predicted) output or target $\hat{\boldsymbol{y}} = f_{\boldsymbol{W}}(\boldsymbol{x})$ parameterized by $\boldsymbol{W}$, where we assume the true target $\boldsymbol{y}$ was generated from our deterministic function f plus noise so that $\boldsymbol{y} = f_{\boldsymbol{W}}(\boldsymbol{x}) + \boldsymbol{\epsilon}$.

The mapping can be visualized as a network, consisting of a number of units, often called neurons due to their loose functional similarity to biological neurons, which are organized into layers where each unit in one layer is connected to each unit in the subsequent layer.

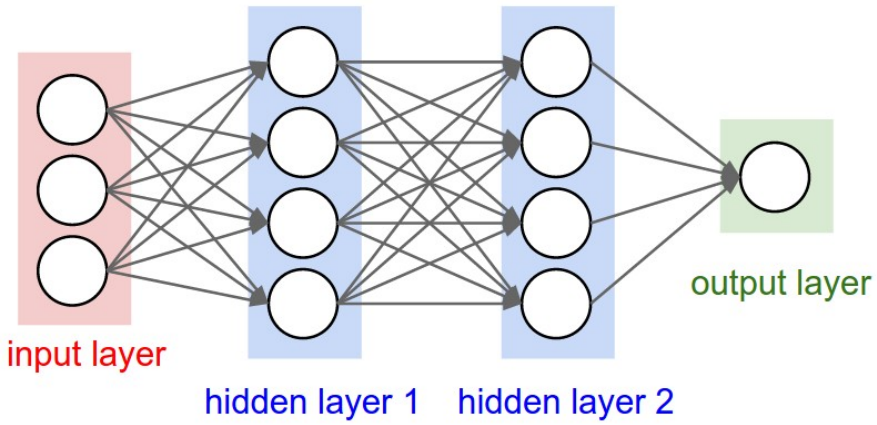


Figure 2.1: A NN with one input and one output layer as well as two hidden layers (Karpathy, 2019a). Circles denote units and the connecting lines denote the weights.

Each NN has at least three layers, where the first and last layer are called input and output layer and the intermediate layers are called hidden layers respectively. This kind of network is called a fully connected NN or multi-layer perceptron (MLP). By stacking many of these layers, a *dee* NN is created. Each connection has its own factor by which any signal traversing this connection is multiplied. Those factors are commonly called the weights or parameters of the network $\boldsymbol{W}$. When information only flows from input to output forming a directed, acyclic graph, as is the case for all networks used in this work, we call the network a feed-forward NN.

As depicted below, each unit computes one output from all of its inputs by first summing them up and then transforming the result by a non-linear (activation)

function σ.

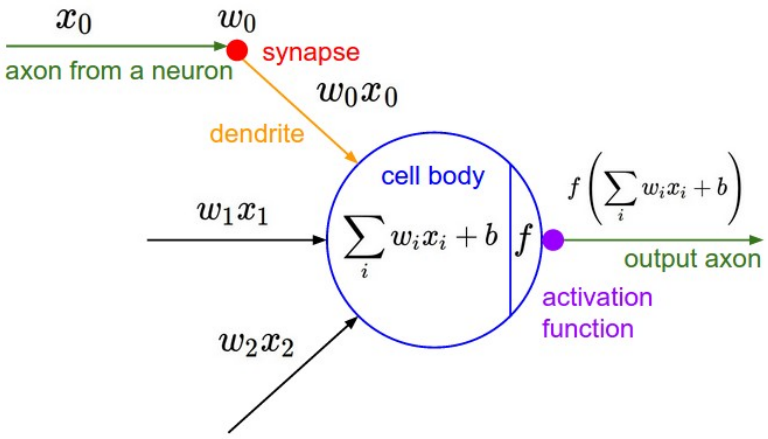


Figure 2.2: The mathematical model of an artificial neuron (Karpathy, 2019b).

When grouping all inputs of one layer into a vector and all weights between this and the subsequent layer into a matrix, the output of any layer can be computed conveniently and efficiently by a matrix-vector product followed by an elementwise application of the non-linear function,

$$\boldsymbol{a}_\ell = \boldsymbol{W}_\ell \boldsymbol{z}_{\ell-1} \tag{2.4}$$

$$\boldsymbol{z}_\ell = \sigma_\ell(\boldsymbol{a}_\ell) \tag{2.5}$$

where $\boldsymbol{a}_\ell$ are the *pre-activations* of layer ℓ with $1 \leq \ell \leq L$ and $\boldsymbol{z}_\ell$ are the activation values. Repeating this operation for all layers, we obtain the desired input-output mapping.

$$\hat{\boldsymbol{y}} = \sigma_L(\boldsymbol{W}_L(\sigma_{L-1}\,(\boldsymbol{W}_{L-1}(...(\sigma_1(\boldsymbol{W}_1\sigma_0(\boldsymbol{W}_0\boldsymbol{x}))$$

Note that without the non-linear transformation, the entire network would reduce to a single matrix-vector product. According to the universal approximation theorem (Csáji, 2001; Cybenko, 1989; Hanin, 2017; Hornik, 1991; Lu et al., 2017), a fully connected, feed-forward NN is able to approximate any function to an arbitrary degree of precision with only a single hidden layer, provided the layer contains a sufficient number of units, making it a general function approximator.

2.2.2 Objectives

In an untrained NN, all weights are initialized to small random numbers. Therefore, any input results in a random output. To improve the networks' prediction, we first need to define a metric to measure its performance, which depends on the task we want to solve. The two most common applications for NNs and ML in general

are regression where the targets $\boldsymbol{y}$ are real-valued numbers and classification where $\boldsymbol{y}$ are discrete classes. In regression the objective is to find the curve that best fits the data at hand and the performance is commonly measured by the average squared distance between the observed and predicted data, also referred to as the *mean-squared error* (MSE) or *sum-of-squares error* (Bishop, 2006, p. 146).

$$E(\boldsymbol{W}) = \frac{1}{2}(\boldsymbol{y} - \hat{\boldsymbol{y}})^\top(\boldsymbol{y} - \hat{\boldsymbol{y}}) \tag{2.6}$$

$$\mathcal{L}(\boldsymbol{W}) = \frac{1}{N}\sum_{i=1}^{N} E(\boldsymbol{W}) \tag{2.7}$$

Here, E denotes the error of a single prediction-target pair and $\mathcal{L}$ denotes the loss: the average error over all data points. For the subsequent discussion about uncertainty in deep learning, it is interesting to note that equation 2.6 has a probabilistic interpretation, as it can be expressed as the negative logarithm of a multivariate normal distribution $\mathcal{N}$[4] with precision β.

$$E(\boldsymbol{W}) = -\ln \mathcal{N}(\boldsymbol{y} \mid \hat{\boldsymbol{y}}, \beta^{-1}\boldsymbol{I}) \tag{2.8}$$

We will refer to this quantity as the negative log likelihood (NLL). The standard choice to measure classification performance is the average cross entropy between the target class and the prediction (Bishop, 2006, p. 209). Here, one measures the difference in entropy of the target (which is zero because each observation belongs to one distinct class) and the entropy of the prediction, which is largest if all possible classes are assigned equal probability and smallest if one class is assigned a probability of one while all other classes are assigned a probability of zero (see Section 4.4.5 for a proper introduction of entropy as a metric).

$$E(\boldsymbol{W}) = -\sum_{c=1}^{K} \boldsymbol{y}_c \log \hat{\boldsymbol{y}}_c = -\ln \mathrm{Cat}(\hat{\boldsymbol{y}}) \tag{2.9}$$

Again, we see that the error term can be expressed as the negative logarithm of a probability distribution, namely the categorical distribution.

2.2.3 Learning and inference

To increase the predictive performance, the objective function, i.e. the loss, needs to be minimized. This procedure is commonly referred to as *learning* or *training* of the network. The only way to influence the output of the network is by either

[4]The terms “normal distribution” and “Gaussian” will be used interchangeably.

increasing or decreasing the magnitude of the weights. The direction of change for each weight can be determined by taking the partial derivative of the objective function w.r.t. each weight. By stacking all partial derivatives into a vector we obtain the gradient of the objective function w.r.t. the weights, which points in the direction of steepest ascent. Consequently, one minimizes the loss by iteratively adjusting the weights in the direction of the negative gradient, an algorithm aptly titled *gradient descent*.

This however only works for the output layer, as the gradient of the loss w.r.t. the weights cannot be computed directly for weights of layers prior to the output. To do this, we need to resort to the chain rule from calculus. By traversing the network from output to input, that is to say in a backward manner, one first computes the gradient of the loss w.r.t the weights of the output layer, as mentioned before. One then needs to compute the gradient of the weights w.r.t. the inputs to the output layer and finally the gradient of the inputs w.r.t the weights of the previous layer. Multiplying those partial derivatives, we obtain the gradient of the loss w.r.t. the weights of the second to last layer. From here we can reiterate. This idea is known as the *Backpropagation* algorithm (Rumelhart et al., 1995, 1988), as we propagate the desired changes to our parameters backwards through the network from the output to the input. We can express this mathematically as

$$\frac{\partial E(\boldsymbol{W})}{\partial \boldsymbol{W}_{ij}^{\ell}} = \sum_k \frac{\partial \boldsymbol{a}_k^{\ell}}{\partial \boldsymbol{W}_{ij}^{\ell}} \frac{\partial E(\boldsymbol{W})}{\partial \boldsymbol{a}_k^{\ell}} = \boldsymbol{z}_j^{\ell-1} \frac{\partial E(\boldsymbol{W})}{\partial \boldsymbol{a}_i^{\ell}} \tag{2.10}$$

$$\Rightarrow \nabla_{\boldsymbol{W}} E(\boldsymbol{W})_{\ell} = \frac{\partial E(\boldsymbol{W})}{\partial \mathrm{vec}(\boldsymbol{W}_{\ell})} \tag{2.11}$$

where vec vectorizes the weight matrix $\boldsymbol{W}$ by stacking its columns. From a Bayesian point of view, learning in NNs corresponds to finding the most likely parameters that explain the data at hand. If we allow the weights to assume any value, Backpropagation learning results in the MLE of the weights $\boldsymbol{W}_{MLE}$. Regularization corresponds to placing a prior on the weights which produces a MAP estimate $\boldsymbol{W}_{MAP}$ where each regularization technique corresponds to a specific choice of prior (e.g. L_2-regularization $\Leftrightarrow$ Gaussian prior) (Bishop, 2006, p. 153, pp. 277).

Once the weights have been learned, the network can be used as a classifier or regressor for new, unseen data, which is called *inference*. Using the language of probability theory, a prediction for a new observation $\boldsymbol{x}^{\star}$ is made by evaluating $\hat{y}^{\star} = \arg\max p(\boldsymbol{y}^{\star}|\boldsymbol{x}^{\star}, \boldsymbol{W}_{MLE/MAP})$ as we have already seen in equation 2.3.

2.2.4 Convolutional neural networks

In *convolutional neural networks* (CNN), some of the fully-connected hidden layers are replaced by so-called convolutional layers. In mathematics, a convolution, which is used in computer vision for edge detection and image smoothing, describes the process and the effect of applying an operation on two functions.

To do so, a convolutional filter, which can be represented by a two dimensional grid where each cell is defined by real valued numbers (i.e. a matrix), is slid over the input (the image) and the underlying pixel intensity (brightness) is multiplied by the number of its corresponding grid cell. As an example, a horizontal edge in the image can then be detected by a grid where the top half is filled with zeros and the bottom half with ones, because the result of the convolution operation is large where we move from a dark to a bright region.

In a CNN, the magnitude of the values in each grid cell are not predefined but learned and as such become the weights of the network[5]. For each layer we can have multiple filters, i.e. one per color channel. The advantage over MLPs is the reduction in the number of parameters, as weights need to be shared, which is especially important when high resolution images are used, and the ability to capture spatial relations i.e. between neighboring pixels in images or temporal relations when used with frequencies in audio signals.

2.2.5 Bayesian neural networks

Even though the name might suggest otherwise, BNNs (Denker and Lecun, 1991; MacKay, 1992c; Neal, 1995) are not just another flavor of NN, but their main idea – to place a probability distribution on the networks' weights, casting them as random variables – can be applied to any network architecture (Gal, 2016). Conceptually this is equivalent to an ensemble of NNs of infinite size, all with the same architecture but with all possible combinations of weights. This allows, in conjunction with Bayesian inference, among others, to estimate the uncertainty of each weight as the variance of its distribution, to calibrate the networks predictions, to detect OOD or adversarial examples or to perform model selection. Performing inference in this setup however is challenging, as we move from an optimization to an integration problem as described in section 2.1, and scalable methods are only recently being investigated, further explored in section 3.

[5]See `http://cs231n.github.io/understanding-cnn/` for some examples of what a CNN actually learns.

2.3 Uncertainty

While we can express uncertainty through probability theory and quantify it using information theory, doing so requires an understanding about where it comes from and why it is important.

Broadly speaking we can arrange sources of uncertainty into two groups (Gal, 2016; Goodfellow et al., 2016):

1. Uncertainty that arises from the inherent randomness of the process being observed or our inability to measure it properly as well as uncertainty that arises from incomplete observability, which results in incomplete knowledge of the underlying process. This is called *aleatoric uncertainty*.

2. Uncertainty about which model parameters or model structure best explain the given observations, called *epistemic uncertainty*.

Typically, both types of uncertainty are present in a ML context, especially in real world applications like robotics and medical diagnosis. Together they induce the *predictive uncertainty* on which we base our confidence in the prediction.

To model aleatoric uncertainty, we can place probability distributions on the outputs of the model while epistemic uncertainty is modeled by placing a prior distribution on the parameters of the model and measuring how much they vary when given some data.

A helpful shortcut to decide whether an uncertainty at hand is aleatoric or epistemic is to ask the question: "Do I see a possibility of reducing the uncertainty by gathering more data or by refining my model (in which case it is probably epistemic) or not (in which case it is probably aleatoric)?" (Der Kiureghian and Ditlevsen, 2009).

As the application of ML algorithms moves from purely academic use into the real world, the need for predictable and reliable solutions is growing, especially if decisions made affect human life.

To this end, it would be desirable to understand how the algorithm came to its decision, but as we typically employ deep learning solutions in domains where we lack the knowledge to model the input to output mapping by hand, we mostly also lack the knowledge to understand the automated decision making.

Alternatively, one can at least expect an algorithm to reliably know when it does not know, in other words, for it to be well calibrated. As further discussed in section 4.4.3, good calibration means to be as confident or uncertain in a prediction

as is warranted by the empirical frequency of being correct. As was demonstrated by Guo et al. (2017), this is not the case for state-of-the-art DNN architectures.

Uncertainty estimates can be useful in almost all applications, but as the additional computational burden is often prohibitive or at least undesirable, they are mostly used in safety critical applications. Among those are physical diagnosis of patients suffering from hard to detect but dangerous or fatal diseases like cancer, where a falsely positive diagnosis induces a high but completely unnecessary level of stress and a falsely negative diagnosis can potentially lead to loss of life. If along those erroneous predictions the doctor also has access to the models' confidence in them, she can take over whenever the algorithm exhibits high uncertainty or she can even be alerted automatically in such cases.

Another popular example is autonomous driving, where NNs are frequently employed to detect and classify the objects in the vicinity of the car. Here, a misclassification can lead to accidents with other traffic participants and in the worst case it can lead to harm to or even death of the driver or pedestrians involved. If the wrong prediction is made with high uncertainty, the vehicle can take steps to mitigate the danger by slowing down or alerting the driver.

3 Related Work

Several attempts have been made to integrate NNs into the Bayesian framework since they were first conceived, coming with the benefits of uncertainty estimates and principled approaches for learning information integration, making all assumptions explicit and guaranteeing optimality of the obtained estimates conditioned on the assumptions made (Berger, 2013; Horvitz et al., 1986). Here we want to explore four of the most notable strategies which gained popularity due to their practicality and scalability: 1. *Variational inference* (VI) (also known as *Variational Bayes* - VB) and *expectation propagation* (EP), 2. *Monte Carlo* (MC) approximation, 3. Dropout VI (also known as MC dropout) and ensembles of NNs and 4. Laplace's Method/Laplace approximation of NN posterior distributions.

3.1 Variational inference and expectation propagation

In a sense, VI returns to the roots of NN inference, reformulating the integration problem of BNNs back into an optimization problem as is the case for deterministic NNs. This is achieved by devising a parameterized variational distribution $q(\boldsymbol{W} \,|\, \theta)$ of the same family as is assumed for the posterior distribution $p(\boldsymbol{W} \mid \mathcal{D})$. The goal is then to minimize the difference between this variational distribution and the posterior by optimization of its parameters, where the distance is measured as the *Kullback-Leibler divergence*[1] (KL divergence) between the two distributions (Blei et al., 2017). EP is similar to VI in the sense that it also computes the KL divergence between the true and approximated posterior distribution, but swaps the order of both distributions (Minka, 2001).

Hinton and Van Camp (1993) were the first to experiments with VI for BNNs, who used a multivariate Gaussian with diagonal covariance matrix as variational distribution to infer the posterior of a small network with one hidden layer of four units. The approach was extended to use full covariance matrices by Barber and Bishop (1998).

[1] See Section 4.4.6 for details.

Graves (2011) proposed the first scalable variant of VI, which works in two stages: first, the intractable true posterior is approximated by a variational distribution as in classical VI to then approximate the variational distribution by MC integration. The performance of this approach is limited by the use of a simple diagonal Gaussian as posterior approximation.

Jylänki et al. (2014) used EP for posterior inference, but updates are noisy as they rely on batch computation. Several approximation factors for each data point need to be kept in memory, which is not feasible for large datasets. Further, the performance of the proposed method was only demonstrated on relatively small networks with two hidden layers.

Soudry et al. (2014) devised an extension to classical Backpropagation called Expectation Backpropagation where no learning rate is required, though it can only be applied to classification (binary instead of continuous targets) and only updates the mean of the Gaussians (mean-field approximation) when applied to continuous weights. Additionally, the prior variance is not learned but fixed to a large value.

Hernández-Lobato and Adams (2015) introduced *Probabilistic Backpropagation* (PBP), which uses a product of Gaussians to approximate the posterior over the weights. During Backpropagation, instead of updating the weights directly, the mean and variance of their respective normal distribution are updated, where the direction of the update is computed using VI. The method has only been demonstrated on small networks with one hidden layer of 50 to 100 units. Initially, PBP was only designed for regression but was later extended to include classification tasks (Benatan et al., 2018).

A similar approach to PBP was taken by Blundell et al. (2015) who also devised a VI driven algorithm for Backpropagation training called *Bayes by Backprop*. The arising intractable integral is approximated using MC sampling and its gradient w.r.t. the mean and variance of the factorized variational Gaussian distribution is used during Backpropagation. As mean and variance parameters are learned for the variational distributions of the weights, the number of parameters is doubled. It was extended to allow training of CNNs by Shridhar et al. (2019).

A disadvantage of VI for inference in BNNs in general is the inability to use existing well performing and well trained architectures. Instead, the network needs to by redesigned and retrained entirely.

3.2 Markov chain Monte Carlo

Markov chain Monte Carlo (MCMC) is a class of algorithms based on the MC method (Gilks et al., 1996) for sampling from high dimensional probability distributions without knowing what the distribution actually looks like and for estimating its expected value.

Neal (1992) pioneered the use of MC methods, specifically the Hybrid MC (HMC) method (nowadays known as Hamiltonian Monte Carlo) for posterior inference in NNs, allowing to approximate the true posterior arbitrarily closely. This was unfeasible with the Metropolis-Hastings algorithm, as it is too slow for large scale problems, because it does not make use of gradient information. Neal (1995) reviewed and extended on his work in his Phd thesis.

The main disadvantages of HMC are potentially poor performance on large datasets, the need to tune hyperparameters and the inability to scale to large models with millions of parameters.

A general summary of the Bayesian approach for NNs, with a focus on MCMC, is given in Lampinen and Vehtari (2001).

3.3 Dropout VI/MC dropout and ensembles

Despite its name, the technique developed by Gal and Ghahramani (2016a) building on Gal and Ghahramani (2016b) is neither VI nor MC sampling, yet it bears some resemblance to both. The idea is enticing in its simplicity.

Many modern NN architectures use •itDropout (Srivastava et al., 2014) as a regularizer against overfitting. During each iteration of the training, a randomly selected subset of activations or weights (Wan et al., 2013) between previously specified layers is set to zero, effectively removing the connections from the network. The network now needs to rely on the remaining parameters to solve the task, which is similar to training an ensemble of different but similar networks.

This behavior is normally prohibited after the network is trained, to allow the network to make use of its full capacity. If dropout is also active during inference, one obtains samples from a hypothetical Bernoulli posterior over the weights for each forward pass, which can be averaged for superior calibration and predictive performance or used to compute predictive variance estimates. To obtain better results, Gal and Ghahramani also introduced a modified form of dropout after convolutional layers.

The technique was used by Kendall and Gal (2017) to obtain epistemic uncertainty estimates in computer vision tasks like image segmentation and depth regression and was improved by Kwon et al. (2018) who successfully modeled the uncertainty in automatic cancer detection from brain scans.

The slightly modified version of MC dropout called *Variational dropout* (Kingma et al., 2015) was applied to weight pruning (model compression) by Molchanov et al. (2017) with great success. In the field of robotics, MC dropout was successfully employed by Miller et al. (2018b) to increase recall and precision for object detection under open-set conditions (Miller et al., 2018a).

Disadvantages are the reliance on dropout layers in the network architecture, computational overhead for multiple forward passes during inference, and limited expressiveness of the uncertainty estimate. Additionally, MC dropout tends to severely underestimate epistemic uncertainty, which was improved by Li and Gal (2017) who used the α instead of KL divergence objective (Hernández-Lobato et al., 2016). On the other hand, the network architecture does not need to be changed when dropout layers are present and already trained networks can be reused.

Instead of sampling weight masks of a single network to obtain a distribution of outputs, one can also directly use multiple different networks or even classifier architectures to form an ensemble.

Lakshminarayanan et al. (2017) showed that this matches or even outperformes the uncertainty estimates obtained from MC dropout, with the disadvantage of having to train one network per sample output in addition to the computational overhead due to a dedicated forward pass for each sample, shared by both methods. Pearce et al. (2018) remarked that ensemble methods are not inherently Bayesian (a view shared by Gal (2016)) and proposed a modified ensembling method to address this shortcoming.

3.4 Laplace's method

The idea behind Laplace's Method or Laplace approximation (Azevedo-Filho and Shachter, 1994; Laplace, 1774) can be stated in a single sentence: Locate the mode of the posterior distribution through optimization and construct a Gaussian distribution around it, where the covariance matrix is the inverse curvature around the mode. We will explore the meaning of this more thoroughly in section 4.1, as it is the workhorse of all results obtained in this thesis.

Buntine and Weigend (1991) already made use of Laplace approximation for posterior inference in NNs in 1991, but also pioneered the use of penalty terms interpreted as prior probabilities, pruning of insignificant weights, estimating the uncertainty of weights, approximate marginalization (MC integration) through network ensembles, detection of OOD examples and more. To estimate the approximated Gaussian distribution around the posterior mode, they used the inverse of the observed and expected Fisher information matrix as its' covariance matrix.

In a series of papers, (MacKay, 1992a,b,c) established a proper probabilistic interpretation of network regression and classification tasks and used the Bayesian framework to reason about NN components. This allowed to determine network hyperparameters like the number of units per layer or the type of regularizer to use from the training data alone without having to rely on (cross-) validation and the associated hyperparameter optimization, but he questioned the applicability to large scale problems.

The contributions mentioned above were summarized and extended independently by Bishop and MacKay in 1995 (Bishop et al., 1995; MacKay, 1995).
More recently, Ritter et al. (2018) demonstrated the scaleability of Laplace approximation, using it to obtain uncertainty estimates from a wide residual network (Zagoruyko and Komodakis, 2016) on the CIFAR-100 dataset[2] (Krizhevsky and Hinton, 2009). As the required curvature matrices become intractable for larger networks, they relied heavily on a Kronecker factored approximation scheme (Botev et al., 2017; Grosse and Martens, 2016; Martens and Grosse, 2015), which we will take a closer look at in Section 4.1.3.

[2]The much larger ImageNet dataset (5.1.2) was not considered but is the principal dataset of this thesis.

4 Concept

This section provides a detailed overview of the theory behind all empirical results presented in this thesis. We discuss how to obtain uncertainty estimates from DNNs along with and attempt to provide an intuitive understanding of their connection to NN learning. We then examine possibilities of quantifying the quality of the uncertainties and how to fool an unsuspecting NN.

4.1 Scalable Laplace approximation

In Section 3.4, Laplace approximation was introduced as a long-established method to obtain uncertainty estimates from various models, including NNs, that has only recently been extended to deal with deep networks (Ritter et al., 2018).

4.1.1 Being normal around the extreme

The goal is to approximate the intractable posterior distribution of the weights of a NN given the data. The approximation is done using the second-order Taylor expansion around the mode of the distribution $\boldsymbol{W}_{MAP}$, which we have already obtained through standard Backpropagation training of the network.

$$\log p(\boldsymbol{W} \mid \mathcal{D}) \approx \log p(\boldsymbol{W}_{MAP} \mid \mathcal{D}) - \frac{1}{2}(\boldsymbol{W} - \boldsymbol{W}_{MAP})^\top H (\boldsymbol{W} - \boldsymbol{W}_{MAP}) \quad (4.1)$$

Note that the first order term of the Taylor expansion is missing, as the gradient at the maximum $\boldsymbol{W}_{MAP}$ is assumed to be zero. The curvature around the mode is the matrix of second derivatives, i.e. the average Hessian H of the negative log posterior[1]. This approximation is only well defined if H is positive (semi-)definite (p.s.d), i.e. $\boldsymbol{W}_{MAP}$ is indeed a local maximum of the posterior (Bishop, 2006, pp. 214, 215; Ritter et al., 2018). Taking the exponential we obtain

$$p(\boldsymbol{W} \mid \mathcal{D}) \approx p(\boldsymbol{W}_{MAP} \mid \mathcal{D}) \exp\left(-\frac{1}{2}(\boldsymbol{W} - \boldsymbol{W}_{MAP})^\top H (\boldsymbol{W} - \boldsymbol{W}_{MAP})\right) \quad (4.2)$$

[1]The logarithm is introduced as the optimization is done w.r.t. loss, which is equivalent to the NLL or neg. log posterior, if we consider regularization. See section 2 for details.

which is the *probability density function* (pdf) of an unnormalized Gaussian distribution, meaning it does not necessarily integrate to one.

$$\frac{1}{Z}p(\boldsymbol{W} \mid \mathcal{D}) \approx \mathcal{N}(\boldsymbol{W}_{MAP}, H^{-1}) \tag{4.3}$$

To obtain the normalization factor Z, Laplace's method can be used, which was conceived to approximate integrals of the form $\int_a^b \exp(Mf(x))\mathrm{d}x$. Suppose $f(x_0)$ is the unique global maximum of $f(x)$. Taking the exponential of $f(x)$ multiplied by M reduces the influence of values x on the integral that are far from x_0 compared to those in the neighborhood. For large M, the original integral can be reasonably approximated by the integral of the second-order Taylor expansion around x_0.

$$Z = p(\boldsymbol{W}_{MAP} \mid \mathcal{D}) \int_{-\infty}^{\infty} \exp\left(-\frac{1}{2}(\boldsymbol{W} - \boldsymbol{W}_{MAP})^\top H (\boldsymbol{W} - \boldsymbol{W}_{MAP})\right) \tag{4.4}$$

$$= p(\boldsymbol{W}_{MAP} \mid \mathcal{D}) \cdot \sqrt{(2\pi)^{|\boldsymbol{W}|} \det H} \tag{4.5}$$

$$p(\boldsymbol{W} \mid \mathcal{D}) \approx \frac{1}{\sqrt{(2\pi)^{|\boldsymbol{W}|} \det H}} \exp\left(-\frac{1}{2}(\boldsymbol{W} - \boldsymbol{W}_{MAP})^\top H (\boldsymbol{W} - \boldsymbol{W}_{MAP})\right) \tag{4.6}$$

$$= \mathcal{N}\left(\boldsymbol{W}_{MAP}, H^{-1}\right) \tag{4.7}$$

From eq. 4.4 we obtain eq. 4.5 because the second term is a Gaussian integral[2]. From there we normalize eq. 4.3 to arrive at eq. 4.6, which is indeed a multivariate normal distribution. In practice, Z does not need to be evaluated, because under most conditions, the posterior distribution is asymptotically normally distributed as the number of data points goes to infinity (Bishop, 2006, pp. 214, 215). While the available data in real world applications is always finite, it is often still sufficiently large for the approximation to work. Another caveat is however that many posterior distributions – especially those of DNNs – are multi-modal, leading to a different (though often equivalent) Laplace approximation for each mode.

We perform approximate Bayesian inference by evaluating the posterior predictive distribution (eq. 4.8) using $T = 30$ MC samples[3] $\boldsymbol{W}_t$ from the approximate posterior (Ritter et al., 2018).

[2] $|\boldsymbol{W}|$ denotes the number of weights of the NN.
[3] Following Maddox et al. (2019). Convergence using $T = 100$ can be found in Section 5.2.2.

$$p(\boldsymbol{y}^\star \mid \boldsymbol{x}^\star, \mathcal{D}) = \int p(\boldsymbol{y}^\star \mid \boldsymbol{x}^\star, \boldsymbol{W}) p(\boldsymbol{W} \mid \mathcal{D}) \mathrm{d}\boldsymbol{W} \approx \frac{1}{T} \sum_{t=1}^{T} p(\boldsymbol{y}^\star \mid \boldsymbol{x}^\star, \boldsymbol{W}_t) \tag{4.8}$$

4.1.2 Amenable curvature

The Hessian is the matrix of second derivatives and as such describes the local curvature of the function in question. In this work, we are interested in the Hessian of the loss w.r.t. the weights. For a single sample $(\boldsymbol{x}, \boldsymbol{y})$ we obtain

$$[H]_{ij} = \frac{\partial^2}{\partial \boldsymbol{W}_i \partial \boldsymbol{W}_j} E(\boldsymbol{W}) = -\frac{\partial^2}{\partial \boldsymbol{W}_i \partial \boldsymbol{W}_j} \log p(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{W}) \tag{4.9}$$

which is a $|\boldsymbol{W}| \times |\boldsymbol{W}|$ matrix, a problem we will deal with in the next section. Another inconvenience is the need of an additional backward pass through the network to compute the second derivative.

Luckily, under certain assumptions[4], the Hessian can be approximated by the *Fisher information matrix* (Fisher) $\tilde{F}$ or the *Generalized Gauss-Newton matrix* (GNN), which is often computationally less expansive. We will refer to Hessian, Fisher and GNN as *curvature matrices*. For a single weight w and $f(w) = -p(\boldsymbol{y} \mid \boldsymbol{x}, w)$ we get

$$H_w = \frac{\partial^2}{\partial w^2} \log f(w) = \frac{\partial}{\partial w} \left(\frac{1}{f(w)} \frac{\partial}{\partial w} f(w) \right) \tag{4.10}$$

$$= \left(\frac{1}{f(w)} \frac{\partial}{\partial w} f(w) \right)^2 - \frac{1}{f(w)} \frac{\partial^2}{\partial w^2} f(w) \tag{4.11}$$

$$= \left(\frac{\partial}{\partial w} \log f(w) \right)^2 - \frac{1}{f(w)} \frac{\partial^2}{\partial w^2} f(w) \tag{4.12}$$

If we now take the expectation over the likelihood, we see that in fact the expected Hessian[5] of the loss (NLL) is the Fisher of the log likelihood.

[4] GNN and Hessian are equivalent when piece-wise linear activation functions like ReLU are used while GNN and Fisher are equivalent for exponential family loss functions like squared and cross-entropy loss (Martens, 2014).

[5] Not to be confused with the Hessian averaged over all samples from the dataset, as the expectation is taken w.r.t. the likelihood $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{W})$ and not the empirical distribution $p(\boldsymbol{x}, \boldsymbol{y})$.

$$\mathbb{E}\left[\frac{1}{p(\boldsymbol{y} \mid \boldsymbol{x}, w)} \frac{\partial^2}{\partial w^2} p(\boldsymbol{y} \mid \boldsymbol{x}, w)\right] = \frac{\partial^2}{\partial w^2} \int p(\boldsymbol{y} \mid \boldsymbol{x}, w) \mathrm{d}\boldsymbol{y} = 0 \tag{4.13}$$

$$\Rightarrow \mathbb{E}[H_w] = \mathbb{E}\left[-\left(\frac{\partial}{\partial w} \log p(\boldsymbol{y} \mid \boldsymbol{x}, w)\right)^2\right] = \tilde{F}_w \tag{4.14}$$

Often, the empirical Fisher F is used instead of the *true* Fisher as $p(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{W})$ is usually unknown. The empirical Fisher is defined as the outer product of the gradient of the log likelihood w.r.t the weights. The average empirical Fisher $\bar{F}$ can then be expressed as

$$\bar{F}_{\boldsymbol{W}} = \frac{1}{N} \sum_{i=1}^{N} \left[\nabla_{\boldsymbol{W}} \log p(\boldsymbol{y}_i \mid \boldsymbol{x}_i, \boldsymbol{W}) \nabla_{\boldsymbol{W}} \log p(\boldsymbol{y}_i \mid \boldsymbol{x}_i, \boldsymbol{W})^\top\right] \tag{4.15}$$

where N is the number of data points. For the sake of convenience, the average empirical Fisher $\bar{F}$ is simply called Fisher F from here on.

A major problem when using Laplace's method in conjunction with state-of-the-art NN architectures is the size of the resulting curvature matrix, because the number of weights $|\boldsymbol{W}|$ is typically in the order of several millions. Further, F needs to be inverted when used as the covariance matrix of the multivariate normal posterior distribution approximation (eq. 4.3). We will now look at an alternative way of representing the Fisher, which allows for efficient computation, storage and inversion. The terms Fisher and curvature matrix as well as inverse Fisher and covariance matrix will be used interchangeably.

4.1.3 Kronecker factorization

The first approximation we make to shrink the size of the curvature matrix is to divide it into blocks, where each block corresponds to all the weights of one layer of the network. In the full curvature matrix, those blocks are found on the diagonal, so this is referred to as a *block-diagonal* approximation. This is a reasonable approximation, as the curvature matrix is in practice block-diagonal dominant (Martens and Grosse, 2015). However, this approach should not be confused with the standard diagonal approximation of a covariance matrix, where all covariances are ignored. Rather, the block-diagonal approximation only neglects the covariances between layers and the covariances within each layer are retained. But even the curvature matrix of the parameters for a single layer might be too large to be reasonably stored and inverted. Similar to the computation of the

gradient of the loss w.r.t. the weights (Section 2.2.3), we can obtain the sample Hessian[6] by differentiating twice. Again, for a single sample $(\boldsymbol{x}, \boldsymbol{y})$ and layer ℓ, we can build on eq. 4.9 to obtain

$$[H_\ell]_{(ab),(cd)} \equiv \frac{\partial^2 E(\boldsymbol{W})}{\partial \boldsymbol{W}^\ell_{ab} \partial \boldsymbol{W}^\ell_{cd}} = \boldsymbol{z}^{\ell-1}_b \boldsymbol{z}^{\ell-1}_d \frac{\partial^2 E(\boldsymbol{W})}{\partial \boldsymbol{a}^\ell_a \partial \boldsymbol{a}^\ell_c} \tag{4.16}$$

Just as we did for the gradient, the Hessian of a single layer can also be re-expressed conveniently in matrix form.

$$H_\ell = \frac{\partial^2 E(\boldsymbol{W})}{\partial \mathrm{vec}(\boldsymbol{W}_\ell) \partial \mathrm{vec}(\boldsymbol{W}_\ell)} = \left(\boldsymbol{z}_{\ell-1} \boldsymbol{z}^\top_{\ell-1}\right) \otimes \frac{\partial^2 E(\boldsymbol{W})}{\partial \boldsymbol{a}_\ell \partial \boldsymbol{a}_\ell} = \mathcal{Q}_\ell \otimes \mathcal{H}_\ell \tag{4.17}$$

Here, $\otimes$ denotes the Kronecker product, which is defined as $\{A \otimes B\}_{ij} = a_{ij} B$. So as it turns out, the sample Hessian can in fact be decomposed into two much smaller matrices (Martens and Grosse, 2015). Additionally, the inverse of a Kronecker product is equal to the Kronecker product of the inverses, which is an extremely convenient property.

$$(A \otimes B)^{-1} = A^{-1} \otimes B^{-1} \tag{4.18}$$

So far we have only looked at the curvature matrix for a single sample and while $\mathbb{E}[H] = \mathbb{E}[\mathcal{Q}_\ell \otimes \mathcal{H}_\ell]$, in general, $\mathbb{E}[\mathcal{Q}_\ell \otimes \mathcal{H}_\ell] \neq \mathbb{E}[\mathcal{Q}_\ell] \otimes \mathbb{E}[\mathcal{H}_\ell]$, which is the final, but potentially major approximation we need to make.
In practice, $\mathcal{Q}_\ell$ can be directly obtained from the outer product of the input vector to layer ℓ while $\mathcal{H}_\ell$ can be computed recursively, starting at the output layer, just like the gradient in the Backpropagation algorithm (Botev et al., 2017; Dangel and Hennig, 2019; Grosse and Martens, 2016; Martens and Grosse, 2015). In Section 4.1.1 we saw how to use the inverse curvature matrix as a covariance matrix to sample from a multivariate normal distribution. To do the same using the newly obtained Kronecker factors, we need to resort to the *matrix normal distribution* (Gupta and Nagar, 1991). In contrast to the multivariate normal distribution, it is defined over an entire matrix instead of a vector of random variables and is parameterized by two p.s.d. covariance matrices, which indicate the covariance of the rows and columns of the matrix of random variables it describes.

$$\mathcal{MN}(\boldsymbol{W}_{MAP}, \mathcal{Q}^{-1}, \mathcal{H}^{-1}) = \mathcal{N}(\mathrm{vec}(\boldsymbol{W}_{MAP}), (\mathcal{Q} \otimes \mathcal{H})^{-1}) \tag{4.19}$$

[6] As we have established equivalence between the Hessian and Fisher, everything discussed here also holds for the Fisher.

4.1.4 Efficient sampling

To be able to sample from the matrix normal distribution efficiently, we can describe it as a series of affine transformations. Let $\boldsymbol{Q}\boldsymbol{Q}^\top = \mathcal{Q}^{-1} \in \mathbb{R}^{q\times q}$ as well as $\boldsymbol{H}\boldsymbol{H}^\top = \mathcal{H}^{-1} \in \mathbb{R}^{h\times h}$ be the Cholesky decompositions of the inverse curvature factors for one layer ℓ and $\boldsymbol{W}_{MAP}$ be the corresponding trained weight configuration reshaped to a $q \times h$ matrix. We can then draw values from the matrix normal distribution, since

$$\boldsymbol{W} = \boldsymbol{W}_{MAP} + \boldsymbol{Q}\boldsymbol{S}\boldsymbol{H}^\top \tag{4.20}$$

where $\boldsymbol{S}$ are samples from a standard normal distribution with zero mean and unit variance, reshaped to a matrix of dimension $q \times h$ (Ritter et al., 2018). To summarize, Laplace's method for NNs consists of the following steps:

1. Select a network architecture and dataset (5.1).
2. Train the network to convergence.
3. Compute the curvature factors (4.1.2, 4.1.3).
4. Sample weight configurations (4.1.4).
5. Repeat for each sampled weight configuration:
   a) Set the weights.
   b) Compute the output for an unseen datum.
6. Collect the results to evaluate the uncertainty (4.4).

4.2 Loss, curvature, posterior

Now that we have a basic understanding of uncertainty estimation in NNs, we can connect it to some high level intuition of one central topic of interest in this thesis: How does the loss landscape and its shape, slope and curvature as a function of the network architecture relate to the network's weight posterior distribution?

Our uncertainty of the correct model parameters on a given dataset can be described by the posterior probability distribution, which is proportional to the product of the data likelihood and our (subjective) prior beliefs about the parameter distribution (see Section 2.1).

$$p(\boldsymbol{W} \mid \mathcal{D}) \propto p(\mathcal{D} \mid \boldsymbol{W})p(\boldsymbol{W})$$

To improve the predictive performance of our model, we typically try to maximize the likelihood of correctly explaining the data given a specific choice of parameters. As we have seen, this is equivalent to minimizing the negative log likelihood, a quantity commonly referred to as the loss (2.2.2). If we constrain the optimization by e.g. prohibiting overly large parameters, this is called regularization. From a probabilistic point of view, this means we introduce a prior belief about the distribution of our parameters and thus we change our optimization objective from maximizing the likelihood to instead maximizing the posterior. Therefore, we see a close connection between the loss as the optimization objective and the network's weight posterior distribution: the former is just the negative logarithm of the latter scaled by a proportionality factor.

This means that we should theoretically be able to infer information about the posterior from looking at the loss and vice versa. For example, it does not seem reasonable to approximate the posterior by a Gaussian if the loss around the extremum is not convex or bell-shaped. Conversely, the concavity of the posterior can be assessed by looking at the eigenvalues of the curvature matrix whose inverse is used as the covariance matrix of the Gaussian. A concave posterior would require only non-negative eigenvalues from the curvature of the loss, while predominantly small eigenvalues should indicate a flat loss landscape around the minimum. We will try to validate some of these observations in the Sections 5.2.4 and 5.3.1.

In order to look at the loss landscape we first need to visualize it. The problem with state-of-the-art NNs and the datasets they are trained on is that both are huge, so obtaining enough loss samples to make a detailed visualization is computationally expensive.

Further, the space we want to visualize is extremely high-dimensional, several orders of magnitude larger than what our brain can work with. To close this dimensionality gap, we will use filter-normalized one-dimensional line plots and two-dimensional contour plots as presented by Li et al. (2018).

The main idea is to first choose a specific point in space, which is defined by the parameters of the network. We will choose the weights the network has acquired after having been trained. Second, we generate a random direction of the same dimension as the network parameters, which is drawn from a standard multivariate normal distribution. Adding or subtracting fractions of this random direction scaled by the *Frobenius norm* of the weight matrix for each layer (to which Li et al. (2018) refer as *filter normalization*) to the current parameter values lets us traverse the space. For the contour plots, two random directions are generated.

This procedure can be described mathematically as

$$f(\alpha) = L(\boldsymbol{W}_{MAP} + \alpha \boldsymbol{W}_{norm}) \tag{4.21}$$

$$f(\alpha, \beta) = L(\boldsymbol{W}_{MAP} + \alpha \boldsymbol{W}_{norm1} + \beta \boldsymbol{W}_{norm2}) \tag{4.22}$$

where f is the loss $\mathcal{L}$ at a point of distance $\pm\alpha$ times the normalized random direction $\boldsymbol{W}_{norm}$ from our starting point $\boldsymbol{W}_{MAP}$ and similarly for the two-dimensional case.

4.3 Fantastic parameters and how to find them

In Section 4.1 we have seen how to obtain uncertainty estimates from DNNs. However, as seen in eq. 4.19, the Kronecker factors need to be inverted to serve as covariances in the multivariate Gaussian posterior we want to sample weight configurations from.

4.3.1 Regularizing the uncertainty

While in theory a matrix computed by an outer product like the Fisher should always be p.s.d. and symmetric, i.e. invertible, in reality this might not necessarily be the case due to numerical reasons. Additionally, we might want to regularize the curvature matrices for two reasons (Ritter et al., 2018):

1. The approximations introduced to Laplace's method in order to make it tractable for application in DNNs, like layer independence (ignoring covariances between layers) and factor independence (approximation of expectation), might lead to an overestimation of the variance in certain directions.

2. Laplace approximation itself might place probability mass in low probability areas of the true posterior.

Ritter et al. (2018) introduced a simple regularization scheme, which we adopt for this thesis. It makes use of two parameters N and τ in the following way:

$$NF_\ell + \tau \boldsymbol{I} \tag{4.23}$$

The average Fisher F_ℓ of a layer ℓ is first scaled by N, which is typically set to the number of data points in the dataset. Then, a multiple τ of the identity matrix $\boldsymbol{I}$ is added to it. We can interpret N as pseudo-observations, if it is set to a value larger than the number of data points in the dataset[7] and τ as the precision of a Gaussian prior on the weights (Bishop, 2006, pp. 279, 280), but they can also both be treated simply as hyperparameters to be optimized for some objective on a validation set (Ritter et al., 2018). Because curvature matrices are Kronecker factored, we need to approximate eq. 4.23.

$$NF_\ell + \tau \boldsymbol{I} \approx \left(\sqrt{N}\mathcal{Q}_\ell + \sqrt{\tau}\boldsymbol{I}\right) \otimes \left(\sqrt{N}\mathcal{H}_\ell + \sqrt{\tau}\boldsymbol{I}\right) \tag{4.24}$$

Ritter et al. (2018) further showed empirically that damping[8] can be reduced drastically if the curvature factors are computed using data augmentation, similar to NN training. This can be explained by the fact that augmentation schemes usually distort the images, leading to a higher loss and steeper gradients, which in turn increases the curvature and therefore have a regularizing effect on the covariance matrices obtained from the inverse curvature factors.

Finding the optimal damping parameters however is challenging, as we have little initial knowledge from which range to choose and each evaluation requires to run one forward pass per weight sample through the network using the entire validation set. As an example, using the LeNet5 NN architecture (5.1.1) on the CIFAR-10 dataset (5.1.2) and 30 posterior samples on $10,000$ validation images takes less than a minute on a modern GPU, but almost one hour for even the smallest deep network, using the same number of posterior samples and $25,000$ validation images from ImageNet. Clearly, a better approach than grid or even random search (Bergstra and Bengio, 2012) is needed to find good parameters for each network.

4.3.2 Bayesian optimization

Bayesian optimization (BO) is useful whenever it is very expensive to evaluate the optimization objective and we can spare some time to think about which set of hyperparameters to test next. A typical example is the training of NNs, where we have many hyperparameters, like the learning rate, regularization strength or dropout rate, and the training procedure can take a very long time.

[7] Or to simply account for the fact that we increased the size of the dataset by using augmentation techniques.

[8] "Damping" and "regularization" is used interchangeably.

In our case, luckily there are only two hyperparameters, but the range of possible values is large and each evaluation is very expensive. Thinking about the next best set of hyperparameters in a Bayesian context means to make use of the already evaluated combinations to inform the ongoing search. Internally, we try to learn a model of the underlying objective function, called the surrogate model, using a learning algorithm like a *Gaussian Process* (GP), a *Random Forest* (RF) or a *Tree-Structured Parzen Estimator* (TPE)[9]. An *acquisition function* is used to decide which set of parameters are the most promising in light of the knowledge gained from previous evaluations, balancing exploration against exploitation. The the most popular of these functions are *expected improvement* (EI) and *lower confidence bound* (LCB). We make use of BO after an initial grid search on LeNet5 and after some manual search on the deeper networks respectively to find a sensible range of values from which the BO algorithm is allowed to pick.

One open question remains: Which metric should be maximized or minimized that captures all properties we care about? The next section introduces some possibilities to quantify the results and the quality of the obtained uncertainty estimates, but they mostly focus on one specific property like accuracy, calibration or distance to OOD data.

One obvious choice for classification is cross-entropy. It is used ubiquitously throughout the literature as it is the loss function of choice for NN training while at the same time, it has a probabilistic interpretation as the NLL (see 2.2.2). Guo et al. (2017) showed however that there seems to be a disconnect between NLL and accuracy. They argue that the network can overfit to NLL in a way that the NLL on the test set increases while at the same time the test accuracy improves further. This means that the NN sacrifices good calibration in favor of increased classification accuracy and the overfitting takes place w.r.t. probabilistic rather than classification error. To see why, consider again the mathematical formulation of the cross-entropy error (eq. 2.9). It is minimized by making correct predictions with maximum confidence. So even after most training examples are classified correctly, NLL can be further decreased by minimizing the entropy of the predicted class probability vector, hence leading to overconfident predictions.

Another possibility is the *Brier score*, which computes the average squared distance between the true and predicted class probabilities. It features some interesting theoretical properties like decomposability into uncertainty, reliability and resolution (Murphy, 1973), but it was found to be too insensitive to small improvements due to its squared instead of logarithmic distance relationship.

[9]Please refer to Bergstra et al. (2011) for an in depth discussion.

Unfortunately, we cannot directly optimize for calibration, as a model which outputs uniform class probabilities is perfectly calibrated in conjunction with a level of chance accuracy. Neither can we directly minimize the entropy on the training data, as is desirable for a good in- and out-of-distribution separation, as this does not guarantee high accuracy. Certainly, an ad-hoc cost function could be designed that tries to balance and capture all desirable properties, as is often the case in reinforcement learning, but apart from the loss of theoretical interpretability, this is still a difficult and time consuming task.

4.4 Quantifying uncertainty

In this section, metrics are introduced to quantify the results and the quality of the uncertainty estimates, focusing on the DNN domain. We will define accuracy, confidence, *average calibration error* (ACE) and *expected calibration error* (ECE) (Guo et al., 2017; Naeini et al., 2015) as well as information theoretic entropy and the KL divergence (Bishop, 2006, pp. 53-55).

4.4.1 Accuracy

The accuracy of a classifier is the fraction of correctly classified observations where $\mathbb{1}$ is the indicator function, which is one if the condition is met and zero otherwise.

$$\mathrm{Acc}(\boldsymbol{Y}, \hat{\boldsymbol{Y}}) \equiv \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}(\hat{\boldsymbol{y}}_i = \boldsymbol{y}_i) \tag{4.25}$$

4.4.2 Confidence

The confidence of a classifier in its prediction is the highest predicted class probability of its output $\hat{\boldsymbol{y}}$.

$$\mathrm{Conf}(\hat{\boldsymbol{y}}) \equiv \max \hat{\boldsymbol{y}} \tag{4.26}$$

4.4.3 Average Calibration Error

The Average Calibration Error is the difference between a classifier's average confidence and accuracy, which is zero for a perfectly calibrated model.

$$ACE \equiv \frac{1}{N} \sum_{i=1}^{N} \mathrm{Conf}(\hat{\boldsymbol{y}}_i) - \mathrm{Acc}(\boldsymbol{Y}, \hat{\boldsymbol{Y}}) \tag{4.27}$$

If $ACE > 0$ the model is overconfident, while $ACE < 0$ means it is underconfident with a slight caveat we will discuss in the next section. Intuitively, for a classifier to be perfectly calibrated, it should correctly classify 80 out of 100 examples for which the predictions were made with 80% confidence (Dawid, 1982). In a way, good calibration occurs when Bayesian and frequentist agree.

4.4.4 Expected Calibration Error

The ACE can be misleading, as a classifier that is underconfident on one half of the domain and overconfident on the other will get a very low calibration error. To mitigate that problem, we can partition predictions into M equally-spaced bins based on their confidence score, compute the absolute ACE for each bin, then take the weighted average over the bins to obtain the expected calibration error,

$$ECE \equiv \sum_{m=1}^{M} \frac{|B_m|}{N} |\text{Acc}(B_m) - \text{Conf}(B_m)| \tag{4.28}$$

where B_m refers to all samples in bin m and N is the number of samples in the dataset. We therefore sacrifice the ability to discriminate between over- and underconfidence, but cannot be fooled into believing that the classifier is perfectly calibrated. We choose $M = 10$ for all our experiments and note that we did not observe large deviations between different sensible values, whereas the *Maximum Calibration Error* (MCE) seems to depend heavily on the binning scheme (Guo et al., 2017). Note that, other than e.g. the Brier Score (4.3.2), the ECE does only measure the calibration of the predicted class and not that of the entire class probability vector, as was observed by Miller et al. (2018a).

4.4.5 Entropy

Information theoretic or *Shannon* entropy measures the uncertainty of a probability distribution as the average information it provides. This can be understood as the level to which the outcome of an experiment would surprise the observer. When flipping a fair coin, the expectation to see `heads` is identical to that of `tails`, which is why a fair coin gets assigned maximum entropy. But if the coin is biased, having e.g. a much higher probability to show `heads` than `tails`, a knowing observer would be surprised to see it land `tails` when flipping it. Consequently, the entropy of a biased coin is lower than that of a fair one. For a discrete random

variable X the entropy is defined as

$$H(X) \equiv -\sum_{x \in \mathcal{X}} p(x) \log p(x) = -\mathbb{E}[\log p(x)] \tag{4.29}$$

For continuous random variables it is called *differential entropy* where the sum is replaced by an integral. When applied to the predicted class probabilities of a classifier it is called *predictive entropy* (Gal and Ghahramani, 2016a).

$$H(\hat{\boldsymbol{y}}) \equiv -\sum_{c=1}^{K} p(y = c \mid \boldsymbol{x}, \mathcal{D}) \log p(y = c \mid \boldsymbol{x}, \mathcal{D}) = -\sum_{c=1}^{K} \hat{y}_c \log \hat{y}_c \tag{4.30}$$

4.4.6 Kullback-Leibler divergence

To compare two probability distributions, we can compare their respective entropies by looking at the relative average information of those distributions, a quantity known as the Kullback-Leibler divergence (Murphy, 2012, p. 57). For two discrete probability distributions with *probability mass function* (pmf) $q(\boldsymbol{y})$ and $p(\boldsymbol{y})$ it is defined as

$$\mathbb{KL}(p \| q) \equiv \sum_{c=1}^{K} p(y_c) \log \frac{p(y_c)}{q(y_c)} \tag{4.31}$$

and similarly for two continuous probability distributions with pdf $p(\hat{\boldsymbol{y}})$ and $q(\hat{\boldsymbol{y}})$, where the sum is replaced by an integral. KL divergence is not symmetric, meaning $\mathbb{KL}(p \| q) \neq \mathbb{KL}(q \| p)$. If symmetry is required, it can by obtained using $\mathbb{KL}(p \| q) + \mathbb{KL}(q \| p)$. We will call a symmetrized, discretized KL divergence *Jensen-Shannon distance* (JSD), following Maddox et al. (2019).

4.5 How to fool a neural network

While NNs are remarkable at recognizing patterns, it was found that they can be fooled quite easily by deliberately constructing distortions that are added to the input in order to make the network predict a wrong class with high confidence (Goodfellow et al., 2014; Szegedy et al., 2013). Such intentionally perturbed inputs have been coined *adversarial examples* and the methods are known as *adversarial attacks*. During training, the loss w.r.t. the weights is minimized through small steps in the direction of steepest *descent*. To generate an adversarial image on the other hand, we make a small step ϵ in the direction of steepest *ascent* of the error w.r.t. the image $\boldsymbol{x}$ instead of the weights. Doing so yields a vector of change $\boldsymbol{\eta}$ with one value per input pixel, which can be directly added to the image. Interestingly, those changes are typically too small for the human eye to detect but fool a NN reliably.

$$\boldsymbol{\eta} = \epsilon \cdot \operatorname{sign}(\nabla_{\boldsymbol{x}} E(\boldsymbol{W})) \tag{4.32}$$

$$\boldsymbol{x}_{adv} = \boldsymbol{x} + \boldsymbol{\eta} \tag{4.33}$$

5 Empirical Analysis

To evaluate Laplace approximation we make use of the metrics introduced in Section 4.4. More precisely, we measure calibration using the ACE and ECE and detection of OOD data as well as adversarial attacks using the JSD. We compute the covariance matrices (4.1.1) and find suitable hyperparameters, employing both grid and random search as well as BO (4.3.2). We begin the analysis with the setup, implementation and methodology making use of small scale examples and then move on to the main results of this work, namely the performance of Laplace approximation for DNNs on large datasets and its ties to architectural decisions.

All introduced methods are implemented using the Python programming language (Python Software Foundation, `python.org`) and TensorFlow (Abadi et al., 2016; Girija, 2016).

5.1 Setup and implementation

In this Section we will first introduce the considered network architectures and datasets and then present the methodology by means of a small network and dataset.

5.1.1 Networks

The choice of network architectures to include into the comparison is largely based on Huang et al. (2017b) who looked at the speed-accuracy trade-off of different popular networks especially as feature extractors for object detection algorithms like (Fast, Faster) R-CNN (Girshick et al., 2014; Ren et al., 2015; Wang and Manning, 2013), SSD (Liu et al., 2016) and YOLO (Redmon et al., 2016) in e.g. robotic applications.

To generate reproducible results and minimize the specification overhead, we only used the official network implementations from TensorFlows high-level API Keras (Chollet et al., 2015), which come pre-trained on the ILSVRC-2012 dataset (5.1.2). The networks are summarized in table 5.1[1].

[1]Showing Top-1 accuracy on the official ILSVRC-2012 validation set without pre-processing

1. To validate the implementation and facilitate experimentation, a **LeNet5** (LeCun et al., 1998) variant with ReLU (Nair and Hinton, 2010) activation functions was used in conjuction with small datasets presented in the next section. LeNet5 consists of three convolutional layers with increasing filter size followed by two fully-connected layers. Weight decay (L_2-regularization) and batch-normalization (Ioffe and Szegedy, 2015), added between all inner layers, which is used in most modern architectures, turned out to be unnecessary for the small datasets.

2. The first deep network family we consider follows the **VGG** (Simonyan and Zisserman, 2014) architecture. It won the ILSVRC-2014 by building on the work of AlexNet (Krizhevsky et al., 2012), but pushed the depth from eight to 16 and 19 layers. It consists of consecutive convolutional layers followed by max pooling layers and three final fully-connected layers. Weight decay and Dropout were used for regularization. Both **VGG16** and **VGG19** are considered in this work. They are by far the largest networks in terms of the number of parameters with more than 100 million weights.

3. The **Inception** architecture (Ioffe and Szegedy, 2015; Szegedy et al., 2017, 2015, 2016) has evolved over the years from the initial Inception v1 (also called GoogLeNet) to the current fourth iteration. The main idea was not only to increase the depth of the network, but also its width by introducing so-called *Inception* modules. Great care was taken to minimize the number of parameters while increasing the depth resulting in a network with 42 layers with six times fewer parameters than VGG16. We use the **Inception v3** variant in this work, as it is available in Keras.

4. **ResNet** (He et al., 2016) is similar to VGG in its simplicity but introduces several important changes. The most notable is the addition of so-called *residual connections* (also called *skip connections*) where information from previous layers is passed directly into layers further ahead, skipping the layers in between. The result is a *residual block* of which multiple are stacked to form the network. In this work we consider the 50, 101 and 152 layer variants of this architecture.

5. **DenseNet** (Huang et al., 2017a) takes the idea of ResNet one step further by connecting each layer to every other layer in a feed-forward manner,

(except brightness range adjustment) using single-crop evaluation of the pre-trained Keras network implementations.

forming so-called *dense blocks*, while ResNet only connects the first to the last layer in a residual block. This approach facilitates training considerably and also yields improved performance compared to the previously presented networks. We will employ 121, 169 and 201 layer variants.

6. *Initially it was planned to include MobileNet (Howard et al., 2017) into the comparison, but due to its special architecture, which makes use of depth-wise separable convolutions, the curvature factor approximation needs to be extended, which is outside the scope of this work.

Architecture	Year	Accuracy	ECE	#Parameters
VGG16	2014	71.23%	2.70%	**138, 357, 544**
VGG19	2014	71.21%	2.29%	143, 667, 240
Inception v3	2015	**78.04%**	**1.63%**	23, 851, 784
ResNet50	2015	74.91%	5.23%	25, 636, 712
ResNet101	2015	76.31%	6.75%	44, 707, 176
ResNet152	2015	76.58%	6.78%	60, 419, 944
DenseNet121	2018	73.26%	3.16%	**8, 062, 504**
DenseNet169	2018	75.03%	5.98%	14, 307, 880
DenseNet201	2018	76.09%	3.93%	20, 242, 984

Table 5.1: Network architecture overview (best or highest/lowest value in bold).

5.1.2 Datasets

While the focus of this work lies on large networks and large datasets like ImageNet, there are several other datasets used predominantly for validation and as OOD data.

1. The smallest dataset used is synthetic and is the only one used for regression. It is inspired by Hernández-Lobato and Adams (2015) but similar experiments can be found throughout the literature. It consists of 20 uniformly distributed points $x \sim \mathcal{N}(-4, 4)$ and targets $y \sim \mathcal{N}(x^3, 3^2)$. We will be referred to this as the **Toy** dataset.

2. The Modified National Institute of Standards and Technology (**MNIST**) dataset (LeCun, 1998) consists of handwritten digits from 0 to 9, each 28×28 grey scale pixels large. There are $60,000$ images of which the last $10,000$ are usually reserved as validation set and additional $10,000$ test images.

3. The **notMNIST** dataset[2] is used as OOD data for MNIST. It has identical dimensions but instead of handwritten digits, it features grey scale letters from A to J in various computer fonts.

4. The Canadian Institute For Advanced Research (**CIFAR-10**) dataset (Krizhevsky and Hinton, 2009) features 10 classes (airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks), which are represented by 32×32 pixel RGB images. There is a total of $60,000$ images for training, of which we reserve $10,000$ for validation, and an additional $10,000$ test images.

5. The OOD dataset for CIFAR-10 is called Street View House Numbers (**SVHN**) (Netzer et al., 2011) and features single house numbers from 0 to 9 in RGB of size 32×32 pixels. There are around $100,000$ such images.

6. The dataset of central importance to this work is the Large Scale Visual Recognition Challenge (ILSVRC) dataset from 2012. The images stem from the much larger ImageNet database (Russakovsky et al., 2015) part of which is used for the annual visual recognition challenge. The 2012 variant has become a benchmark used throughout the literature to compare novel image recognition approaches and was used to train the networks used in this work. It consists of around 1.3 million training and $50,000$ validation images with an average size of 400×350 (Deng et al., 2009). There are 1000 classes, which range from different kinds of animals and plants to everyday objects. We will refer to this dataset as **ImageNet** or **ILSVRC-2012** interchangeably.

7. The ImageNet dataset is quite large and diverse. Consequently, it is not entirely clear what type of image can be regarded as being "out-of-distribution". One possibility are artistic interpretations of real objects and environments such as paintings and drawings from different artists and epochs, which is the approach taken in this work. We will refer to this as the **Art** dataset[3].

5.1.3 Libraries

Several high level Python libraries where used that implement some of low level functionality. The three most important ones are briefly presented subsequently.

- **Curvature approximation:** To approximate the Kronecker factored curvature matrices, the Kronecker-Factored Approximate Curvature (**K-FAC**)

[2] Available at `http://yaroslavvb.blogspot.co.uk/2011/09/notmnist-dataset.html`
[3] Available at `https://www.kaggle.com/c/painter-by-numbers/data`

library was used[4], which implements the approximate second-order optimization method presented in Martens and Grosse (2015) for TensorFlow. It supports the approximation of the Fisher, empirical Fisher and curvature propagation matrix (Chen et al., 2018) while the GNN matrix used in Botev et al. (2017) in an approach known as KFRA is not implemented to date.

- **Bayesian optimization:** Multiple algorithms are used to find good values for the two damping hyperparameters of the curvature factors. Most are implemented in the library **Scikit-Optimize**[5] while the TPE is implemented in **Hyperopt**[6].

- **Adversarial attacks:** The adversarial examples used to determine the resilience of Laplace's method applied to NNs are generated using the FGSM of Goodfellow et al. (2014), which is implemented in the **CleverHans**[7] (Papernot et al., 2016) library.

5.2 Introduction and validation

To validate the implementation and introduce tools of uncertainty quantification and visualization, the method is first demonstrated on the small LeNet5 CNN architecture and the MNIST and CIFAR-10 datasets.

5.2.1 Training

Contrary to the deep network architectures we will work with later, which come with pre-trained weights on the ImageNet dataset, the custom build LeNet5 needs to be trained on the two small datasets.

Using Keras this is straight forward, especially as we do not strive for competitive classification performance as our focus lies on the uncertainty estimation using Laplace approximation. We use the Adam optimizer (Kingma and Ba, 2014) and a batch size of 32 and slowly decay the learning rate when training on CIFAR-10. No regularization like batch-normalization, weight decay or Dropout is required. We augment the CIFAR-10 dataset during training by performing random vertical flips and photometric distortions like brightness and saturation changes. This

[4]Available at `https://github.com/tensorflow/kfac`
[5]Available at `https://github.com/scikit-optimize/scikit-optimize`
[6]Available at `https://github.com/hyperopt/hyperopt`
[7]Available at `https://github.com/tensorflow/cleverhans`

is done in order to study the effects of data augmentation when subsequently applied to the curvature factor computation.

On MNIST, we obtain an accuracy of 99.18% and on CIFAR-10 an accuracy of 71.12% on the respective validation sets.

5.2.2 Curvature factors and sampling

Having obtained the MAP estimate of the network parameters, the next step is to compute the Kronecker factored curvature around it. Using the K-FAC library, we compute the empirical Fisher matrix on the MNIST and CIFAR-10 training sets for the LeNet5 network and on the ImageNet training set for the remaining networks, using an exponential moving average (EMA) with 0.05 decay rate[8].

We adopt data augmentation for all curvature factors evaluated on the ImageNet dataset as explained in Section 4.3.1. The specific distortions and adjustments for the individual networks closely follow the procedure used for training described in the respective papers of each network. Among those are cropping image patches of random size, flipping as well as color, saturation and brightness changes, referred to as photometric distortions. We only deviate from the prescribed methodology where unavoidable in order to provide input dimensions and ranges that the pre-trained models expect.

Once the factors are computed, weight configurations can be sampled from the matrix normal distribution for each network as described in Section 4.1.4.

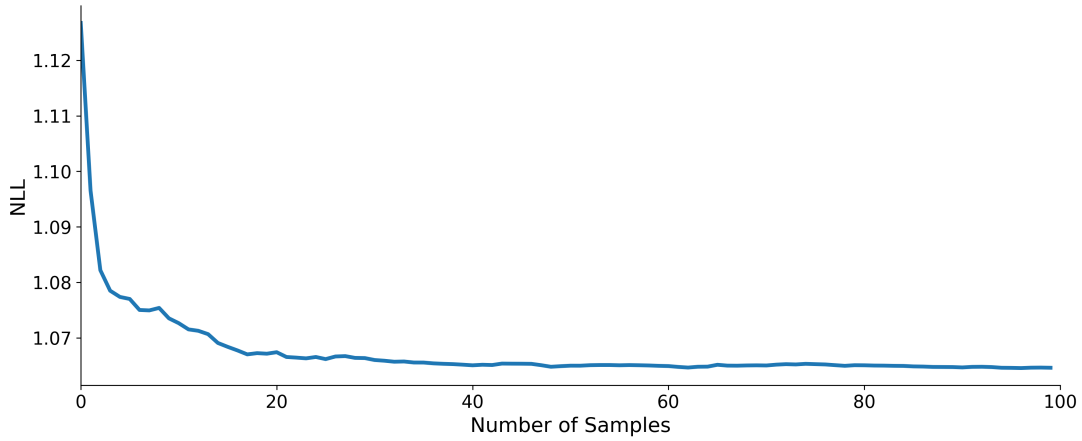


Figure 5.1: NLL convergence with increasing number of posterior samples.

[8] As we only have access to a finite amount of data and thus need to resort to the empirical Fisher, the EMA helps to smooth out fluctuations and to converge to stable values.

5.2.3 Hyperparameters

For the small LeNet5 network, a grid search over the hyperparamters can be performed. This is done on validation sets consisting of 10,000 samples reserved from the MNIST and CIFAR-10 training data. Because we have little initial knowledge about a suitable range of values for the two hyperparameters τ and N, both of them are picked from log-space, that is to say, searched over in powers of 10 to cover a wider range of values. Below the results of the NLL as a function of the two damping parameters is visualized. Lower values are represented by dark violet tones while higher values are shown from dark green to yellow as indicated by the color bar. The 10 best values (in this case the ten lowest values) are highlighted by red borders and the overall minimum is indicated by an additional cross.

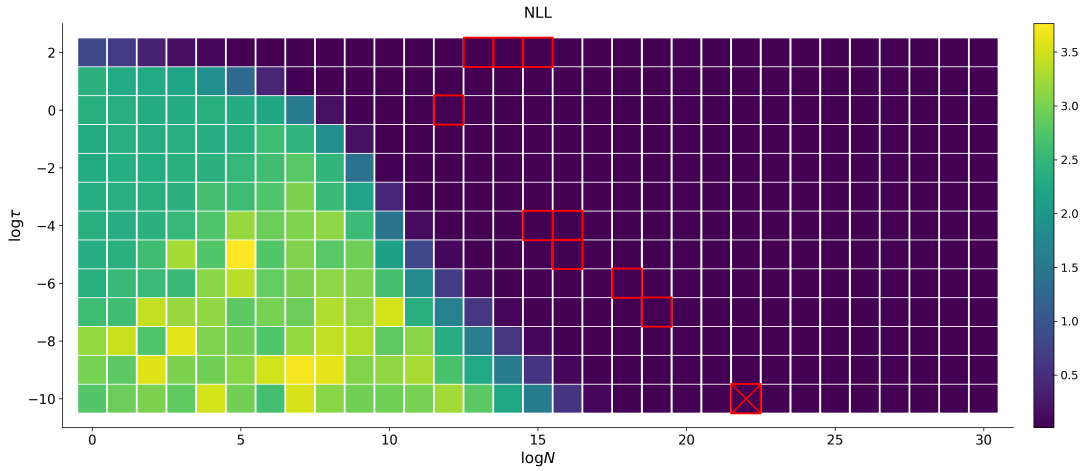


Figure 5.2: Grid search in log-space over the NLL as a function of the two hyperparameters τ and N of the Laplace approximation method using the LeNet5 network architecture and the MNIST dataset. Lower is better.

Clearly, sufficient damping is needed to obtain a low NLL (remember that the terms NLL, cross-entropy and loss have the same meaning in this context and are used interchangeably). Interestingly, increasing the value of one parameter often allows to decrease the other without loss of performance, which will be discussed further in Section 6. While the NLL provides information about the classification performance of the model, we are also interested in its calibration and its capability to discriminate between known and unknown data.

Looking at the results of the JSD as a function of both damping parameters we can observe that is it not sufficient to just apply strong damping to the curvature factors, as we lose the valuable information about the classifier's uncertainty they provide and return to the deterministic setting.

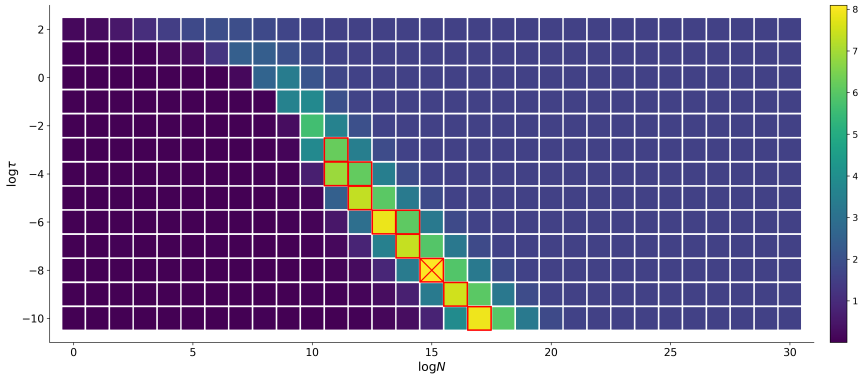


Figure 5.3: JSD between in- and out-of-distribution data as a function of τ and N. Higher is better.

This type of hyperparameter search is unfortunately not feasible for the larger network architectures and datasets, especially as it is still too coarse to yield good results. While a single cell in the above figures is computed in less than a minute for LeNet5 on both small datasets, it takes around one hour for even the least complex deep network and a state of the art GPU. We therefore use the insights gained, to pick suitable ranges of parameter values and apply them to BO. Figure 5.4 shows the parameter sampling behavior of the TPE, clearly focusing on a specific range of parameters, also showing some initial and intermittent random sampling behavior. The size of each circle indicates when it was sampled where early samples are smaller than later ones, which is interesting in the context of BO, because we would hope to see the optimizer converge on better values as the search progresses.

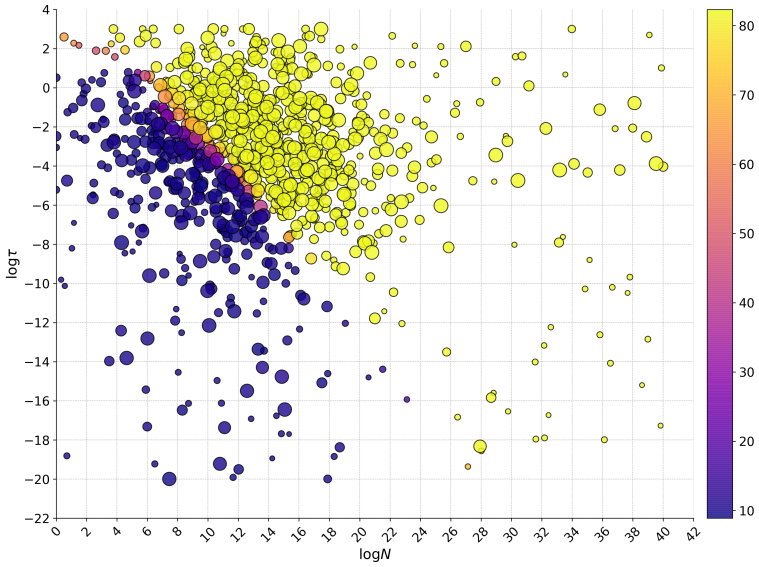
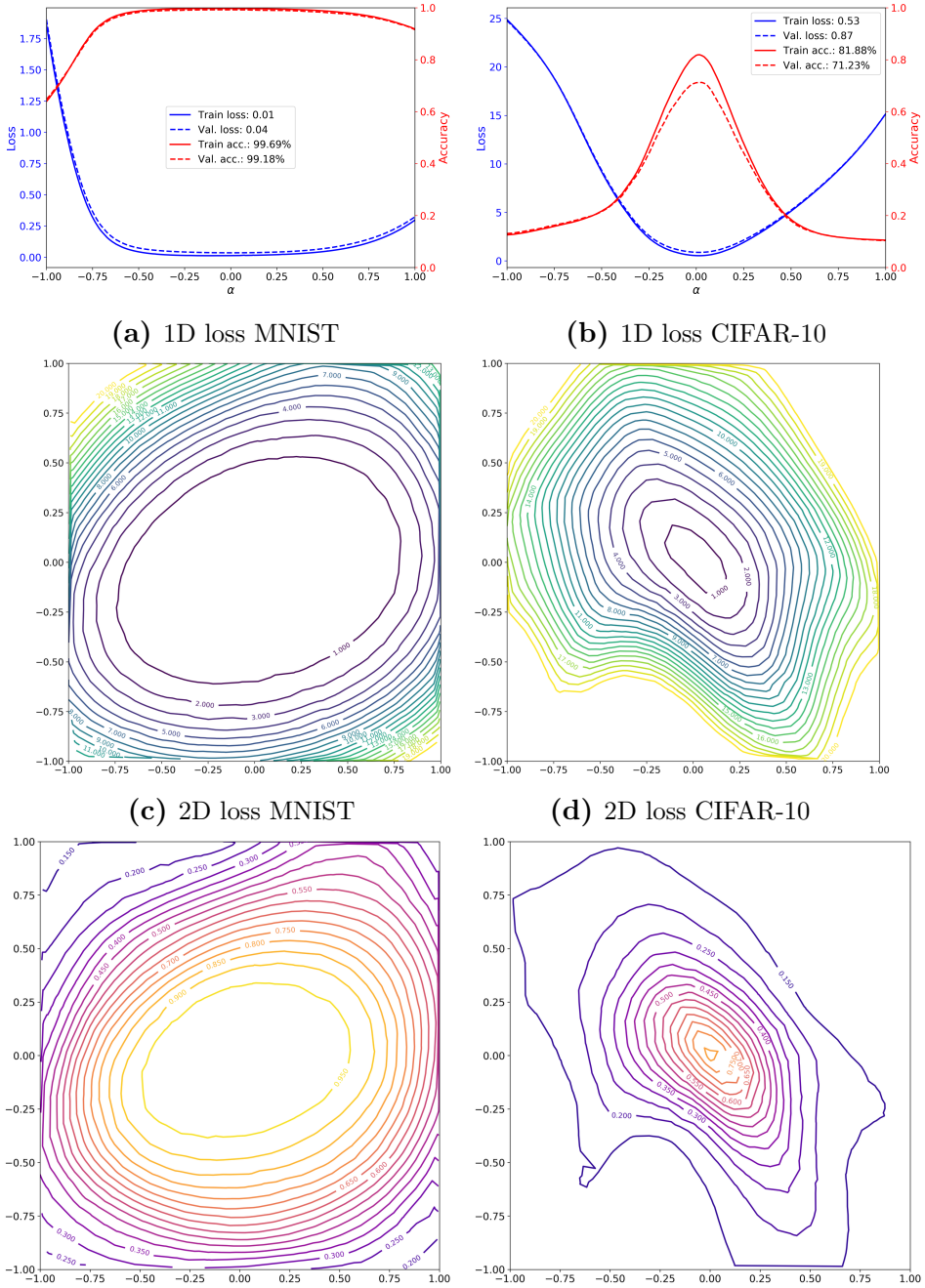


Figure 5.4: Accuracy as a function of τ and N obtained using BO.

5.2.4 On eigenvalues and loss landscapes: Part I

To understand and explain why different networks and datasets lend themselves to Laplace approximation to different degrees, the loss and accuracy as a function of the networks' parameters can be visualized and compared to the eigenvalue histogram of the curvature factors as explained in Section 4.2.



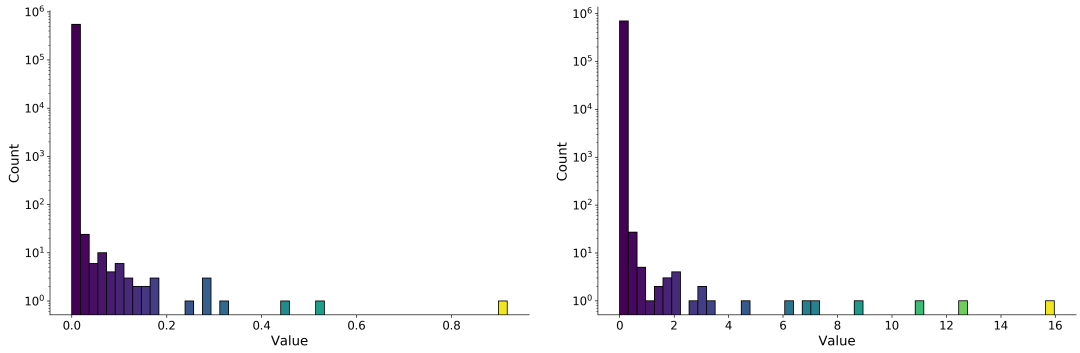
(a) 1D loss MNIST **(b)** 1D loss CIFAR-10

(c) 2D loss MNIST **(d)** 2D loss CIFAR-10

(e) Accuracy MNIST **(f)** Accuracy CIFAR-10

Figure 5.5: 1D (5.5a, 5.5b) and 2D (5.5c, 5.5d) loss as well as 2D accuracy surfaces (5.5e, 5.5f) for LeNet5 trained on MNIST and CIFAR-10.

While both loss landscapes seem to be convex, the one obtained from the MNIST dataset is less steep and generally slightly less chaotic than the one obtained from CIFAR-10. Especially further away from the minimum, the loss surface of MNIST – contrary to that of CIFAR-10 – maintains its elliptic shape, which makes the quadratic approximation applicable. We can now compare those intuitions to the eigenvalue histograms of the curvature factors for both datasets.



(a) MNIST **(b)** CIFAR-10

Figure 5.6: Eigenvalue histograms of the curvature factors from LeNet5 trained on MNIST and CIFAR-10.

Very small eigenvalues are clearly dominant for both datasets[9] but the eigenvalues obtained from MNIST are by and large about one order of magnitude smaller than those from CIFAR-10, as expected. This means we can gain insights about the shape of the loss landscape from the eigenvalue histogram of the curvature matrices and vice versa, but if that extends to predictability of the performance of Laplace approximation needs further analysis.

5.2.5 Calibration

We will now take a look at the calibration of LeNet5 both for a deterministic forward pass and the averaged probabilities of multiple probabilistic forward passes, which was found to improve the predicted class probabilities (Gal and Ghahramani, 2016b; Guo et al., 2017). To generate the results of multiple probabilistic forward passes, we use the weights sampled from the approximate posterior distribution that we obtained using Laplace approximation. To visualize ACE we can use confidence histograms, also showing the average accuracy and confidence as dashed black lines. The ACE is the distance between the two lines, showing that LeNet5 trained on CIFAR-10 is around 6% overconfident.

[9]Note the log-scale on the ordinate, which allows us to observe the very rare larger eigenvalues.

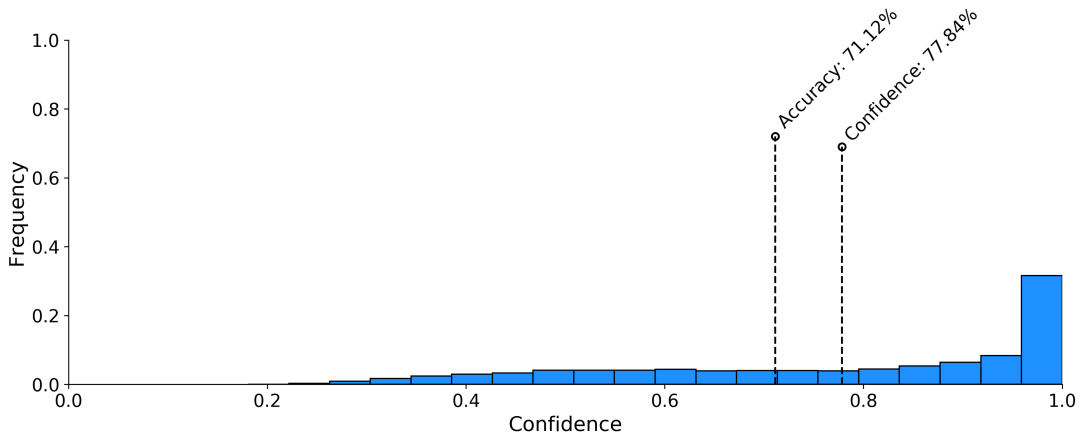


Figure 5.7: Confidence histogram for LeNet5 trained on CIFAR-10.

To directly compare the calibration of different setups, a visualization inspired by Maddox et al. (2019) is more suitable. Here we compare the calibration of LeNet5 trained on MNIST and CIFAR-10. The network is overconfident where the graph is above the dashed black horizontal line and underconfident where it is below.

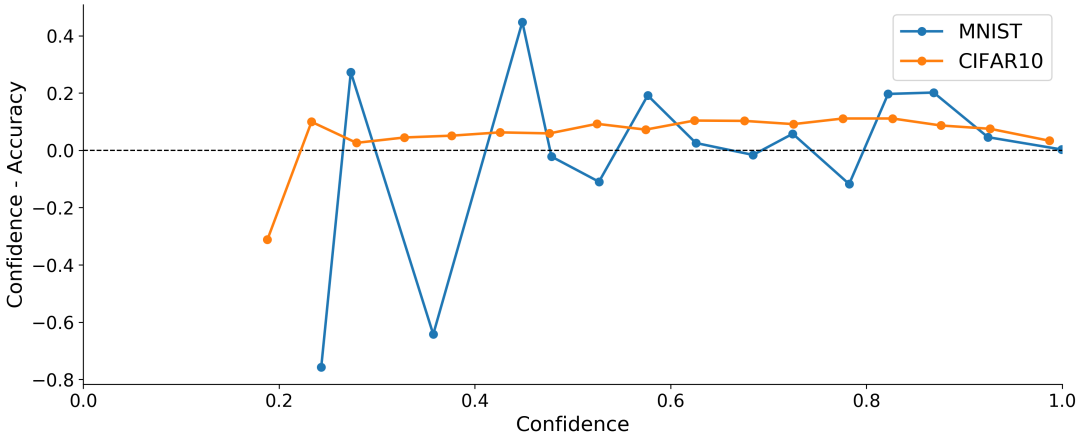


Figure 5.8: Calibration comparison of LeNet5 trained on MNIST and CIFAR-10.

While LeNet5 trained on MNIST seems to exhibit much poorer calibration than the same network trained on CFIAR-10, this can be misleading, especially on small datasets, because calibration as well as reliability diagrams, which we will introduce next, are not sensitive to the number of samples in any given bin[10]. Visually significant miscalibration like the one seen above can therefore potentially be caused by a single poorly predicted data point.

[10]To the contrary, the ECE metric weighs the miscalibration of each bin by the number of samples in it.

To thoroughly analyze the calibration of a single network, reliability diagrams similar to Guo et al. (2017) can be used, which visualize accuracy as a function of confidence, by showing the average accuracy per confidence bin as blue bars, and their difference (the calibration gap) as red bars on top. A perfectly calibrated network exhibits the same accuracy in each bin as the corresponding confidence dictates, resulting in the identity function, indicated by the dashed black line. LeNet5 trained on CIFAR-10 shown below is not well calibrated, as it is underconfident in the region from 10% to 20% confidence and overconfident in all remaining regions.

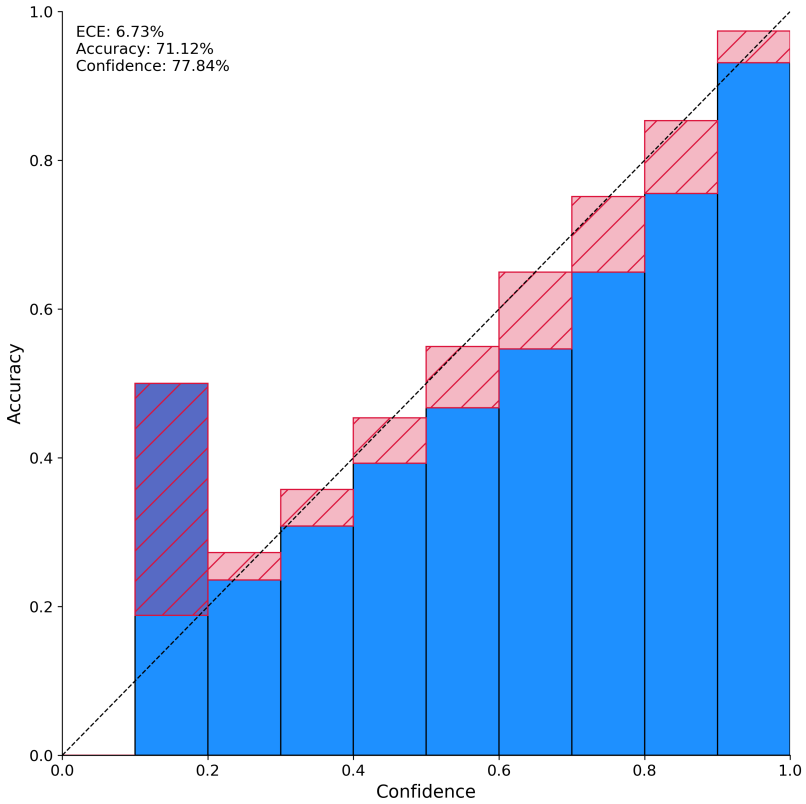


Figure 5.9: Reliability diagram of LeNet5 (CIFAR-10)

To calibrate the network, the average class probabilities obtained thorough Laplace approximation can be utilized. To do so, we sample 100 weight configurations using $\log \tau = -8$ and $\log N = 17$ as damping parameters obtained through the previous grid search. We then provide the network with each set of weights and compute a forward pass on the test set of CIFAR-10, collecting the probability vectors of the softmax layer. Using the averaged predicted class probabilities we obtain the following reliability diagram. The ECE has improved by more than 5% with an additional slight increase in accuracy.

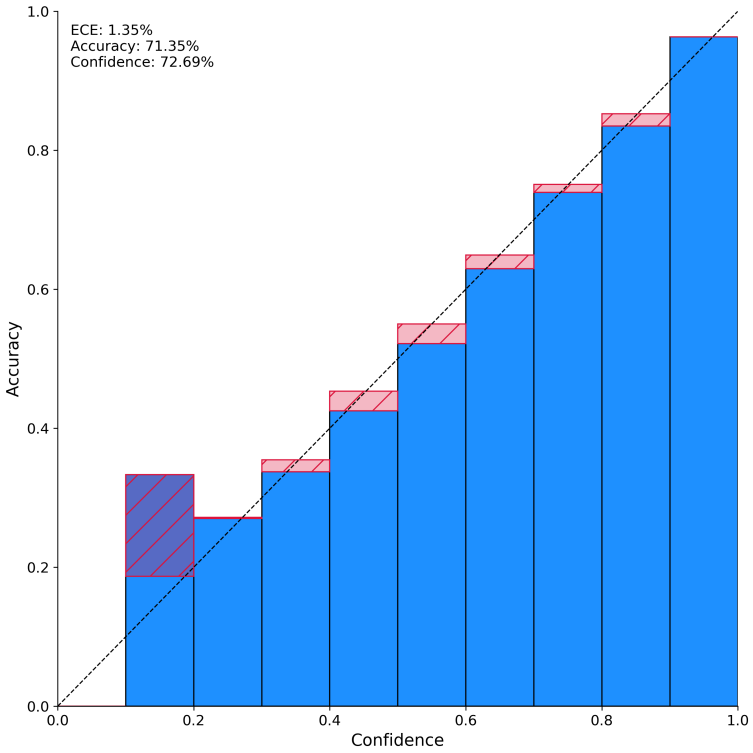


Figure 5.10: Reliability diagram for LeNet5 on CIFAR-10 using 100 weight posterior samples ($\tau = \log -8$, $N = \log 17$).

5.2.6 Out-of-distribution detection

To visualize the uncertainty of a NN on a given dataset, we can use the entropy of its predictions as introduced in Section 4.4.5. The most straight forward way to do so is to produce an entropy histogram as shown below[11].

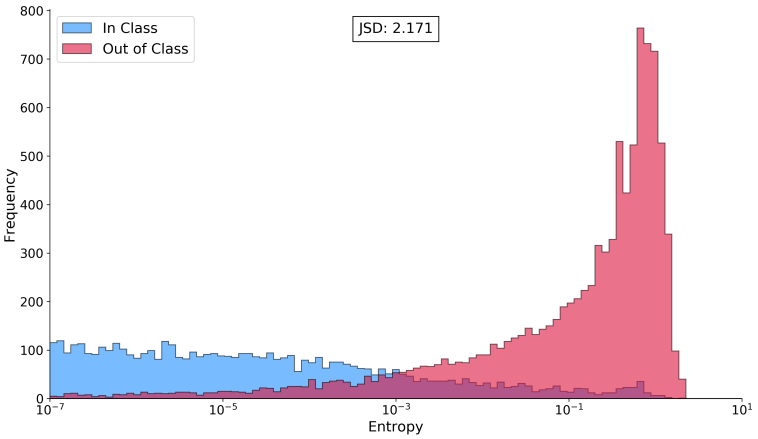


Figure 5.11: Entropy histogram of LeNet5 on MNIST (blue) and notMNIST (red).

[11] We use a logarithmic scale for the abscissa to increase the resolution of the low entropy region, which is useful for highly confident networks.

The graph compares the uncertainty the network exhibits on both the MNIST dataset it was trained on and the notMNIST dataset. We see that even the deterministic network output produces a good separation of both data distributions but there remains an overlapping region. A perfect classifier would exhibit zero uncertainty for the known dataset, producing a clear peak on the left side of the graph, and maximum uncertainty (a value of around 2.3 for a 10 classes dataset) for all samples from unknown data resulting in another peak on the right end of the entropy spectrum. Again, Laplace's method can be used to increase the distance between both distributions. Indeed, the separation of in- and out-of-distribution data can be increased significantly, producing two clear peaks in the below histogram and a five times larger JSD compared to the deterministic network output.

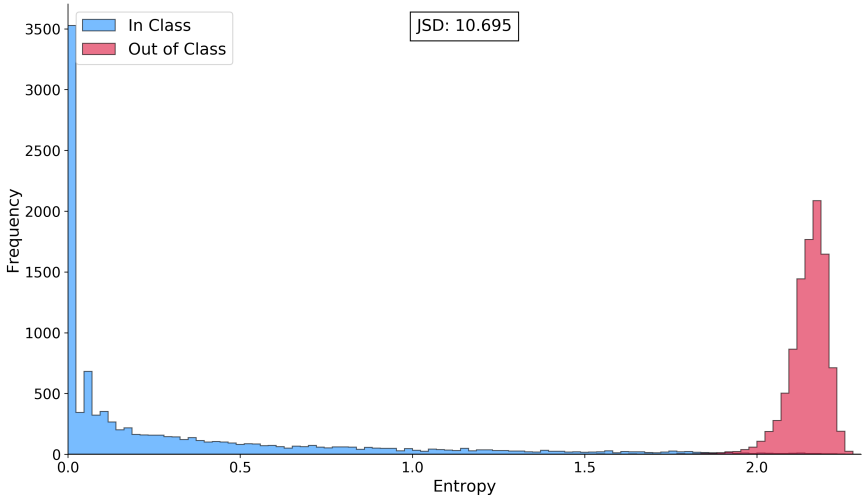


Figure 5.12: Entropy histogram of LeNet5 on MNIST and notMNIST using 100 posterior weight samples ($\tau = \log -5$, $N = \log 12$).

To directly compare the increase in performance between deterministic and probabilistic network outputs, a slightly more sophisticated visualization inspired by (Ritter et al., 2018) can be used. We keep the predictive entropy on the abscissa but use the inverse empirical cumulative distribution function (ECDF) on the ordinate. The CDF of a random variable expresses the probability (on the ordinate) that this random variable will take on a value less than or equal to any value on the abscissa. The empirical variant of the CDF provides the empirical frequency instead of the probability. We therefore see the share of data points for which the network exhibits *less than or equal* entropy than the value indicated on the abscissa. The inverse ECDF then provides insight into which data points are predicted with *equal or higher* entropy.

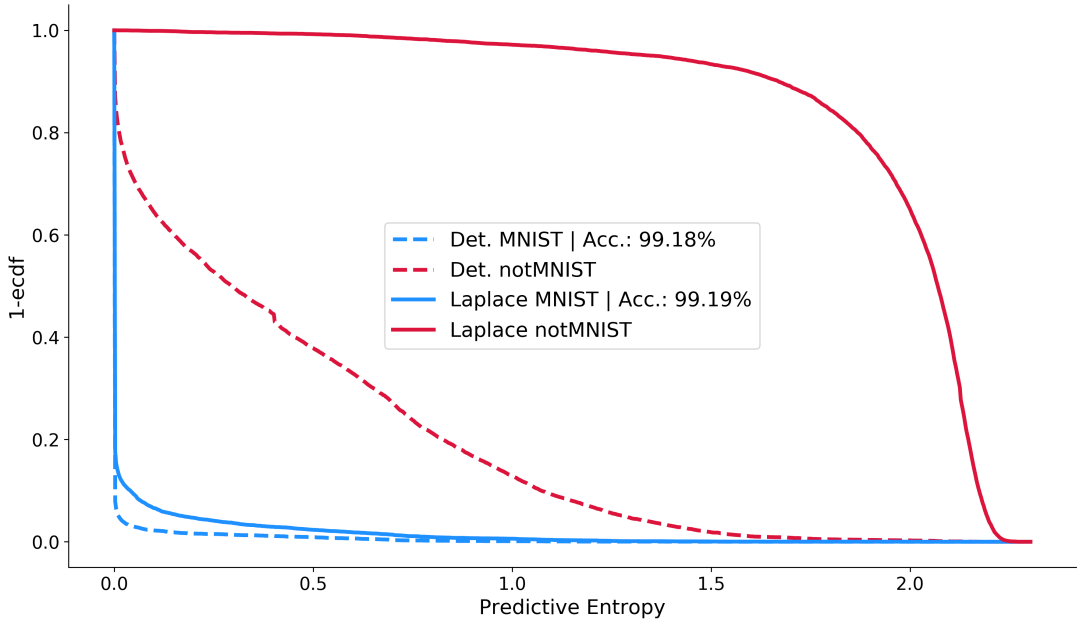
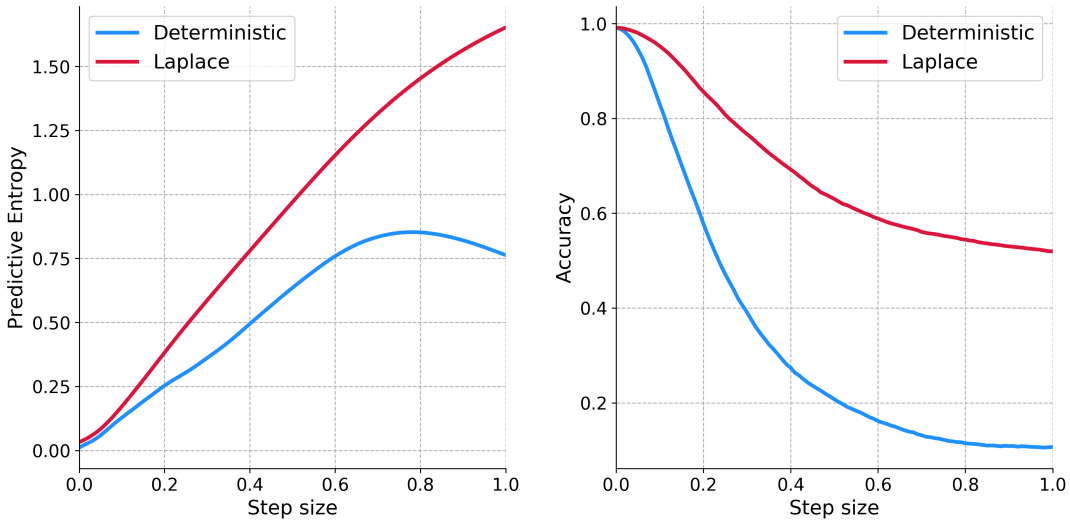


Figure 5.13: Inverse ECDF vs. predictive entropy of LeNet5 on MNIST (blue) and notMNIST (red) using 100 posterior weight samples ($\tau = 0.01$, $N = 10^{10}$).

The deterministic network output predicts almost no data points with an entropy greater 0.1 on MNIST (dashed blue line) but also almost 80% of the data points of notMNIST with an entropy near zero (dashed red line). The uncertainty using Laplace approximation increases slightly on MNIST (solid blue line) but greatly on notMNIST (solid red line), now predicting all OOD samples with above zero entropy and still even almost all with an entropy greater 1.5 while maintaining high accuracy. A perfect classifier would produce a uniform distribution at 1.0 for all OOD samples and another uniform distribution at 0.0 for the known data.

5.2.7 Adversarial examples

Detecting adversarial attacks is similar to detecting OOD data. The advantage is that we are able to precisely determine the amount of distortion we want to apply to the images, which allows us to test the behavior of a classifier close to the data distribution. Figure 5.14 shows the average predictive uncertainty and accuracy of LeNet5 on MNIST for increasing steps size of the FGSM as explained in Section 4.5.



(a) Predictive entropy **(b)** Accuracy

Figure 5.14: Adversarial attack using the FGSM with increasing step size on a LeNet5 network trained on MNIST.

Similar to Ritter et al. (2018) we observe a more rapid increase in predictive entropy when using the averaged predicted class probabilities of the Laplace approximated posterior compared to a single deterministic forward pass. Interestingly, and in contrast to the results obtained from probabilistic output, the predictive entropy of the deterministic network begins to decrease again at a step size of around 0.7, indicating an erroneous overconfidence in light of strong distortions. Additionally, but contrary to previous work, Laplace approximation seems to make the network more robust to adversarial attacks. While the accuracy of the deterministic network drops to the level of chance with a step size of around 0.8, the probabilistic network maintains an accuracy of above 50%. This could be the result of the convolutional LeNet5 architecture in contrast to the fully-connected network used by Ritter et al. (2018).

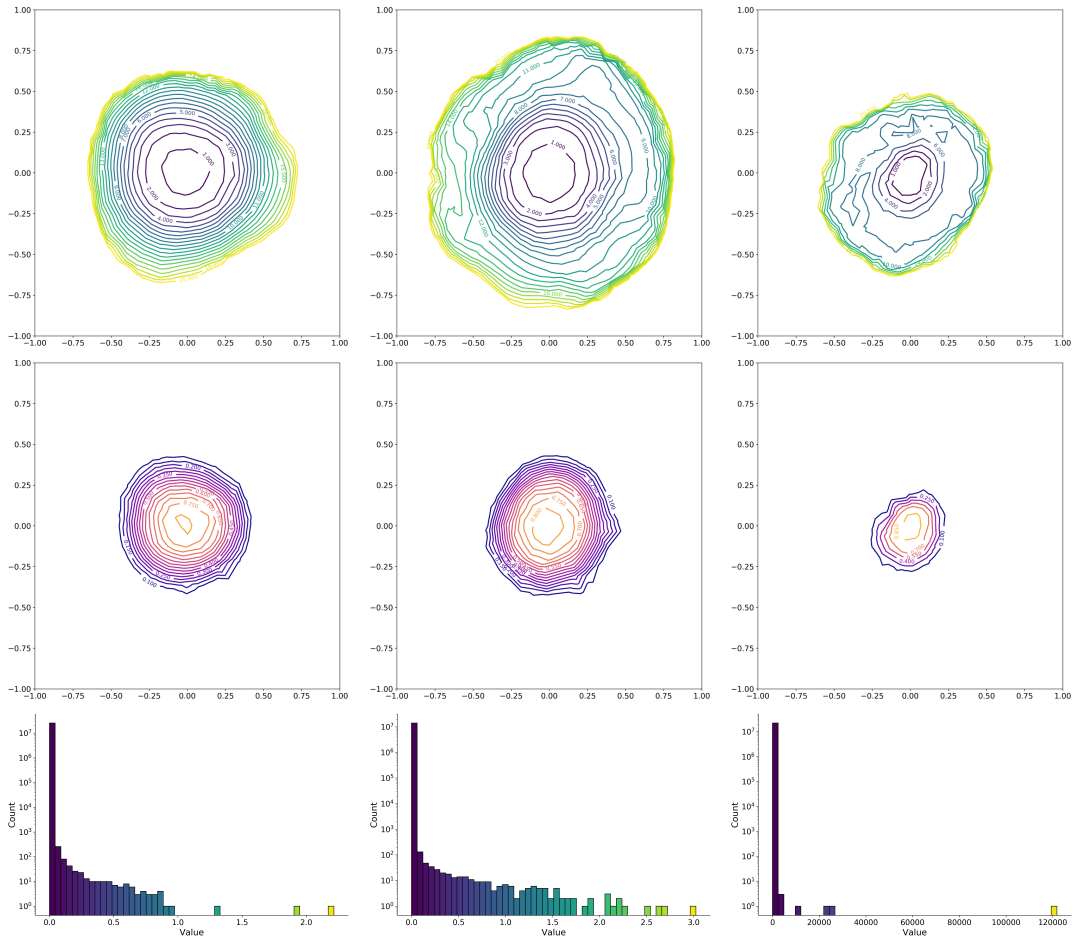
5.3 Dealing with deep nets

This section presents the main results of this thesis obtained from the DNN architectures on the ImageNet dataset[12]. Similar to the previous section, we first focus on the loss landscapes and eigenvalue histograms and then analyze calibration, OOD detection and adversarial attacks in turn.

[12]Results are presented for selected architectures. The remaining results are in the appendix.

5.3.1 On eigenvalues and loss landscapes: Part II

In the first part of the eigenvalue histogram and loss landscape comparison (5.2.4), a correlation between the smoothness of the loss surface and the frequency of small eigenvalues obtained from the curvature factors could be observed. This observation extends to the deep networks, as can be seen in the following figure. Three deep network architectures are presented next to each other, from most to least smooth, a trend also visible in the eigenvalue histograms presented at the bottom.



(a) ResNet50 **(b)** DenseNet169 **(c)** Inception v3

Figure 5.15: Loss and accuracy surface as well as eigenvalue histogram comparison for three deep network architectures.

All networks show convex loss landscapes, while deeper networks within each architecture produce smaller curvature, resulting in flatter surfaces. While small eigenvalues and smooth loss surfaces do not seem to predict good performance of Laplace's method reliably, a chaotic loss and large eigenvalues seem to prohibit a proper Gaussian posterior approximation.

The confounding influence of good and bad hyperparameter choices remains however difficult to exclude, so that further investigation is needed to tie specific architectural choices to resulting changes in loss surface and curvature eigenvalues and to predict applicability of Laplace approximation.

5.3.2 Case Study 1: Calibration

Before trying to improve the calibration of the deep networks, a baseline comparison is shown in the following figure, as introduced in Section 5.2.5. All architectures show overconfident behavior across most confidence regions, except for Inception v3, which surprisingly is slightly underconfident. As already presented in table 5.1, the ECE is not as high as Guo et al. (2017) seem to imply.

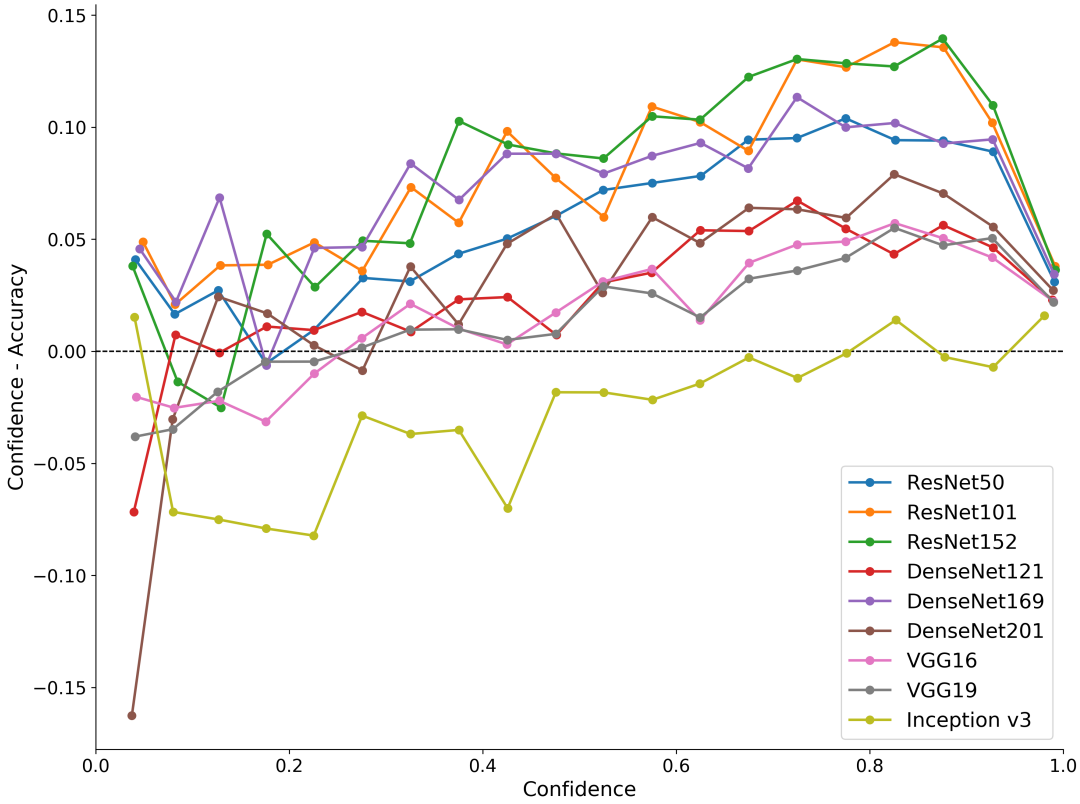
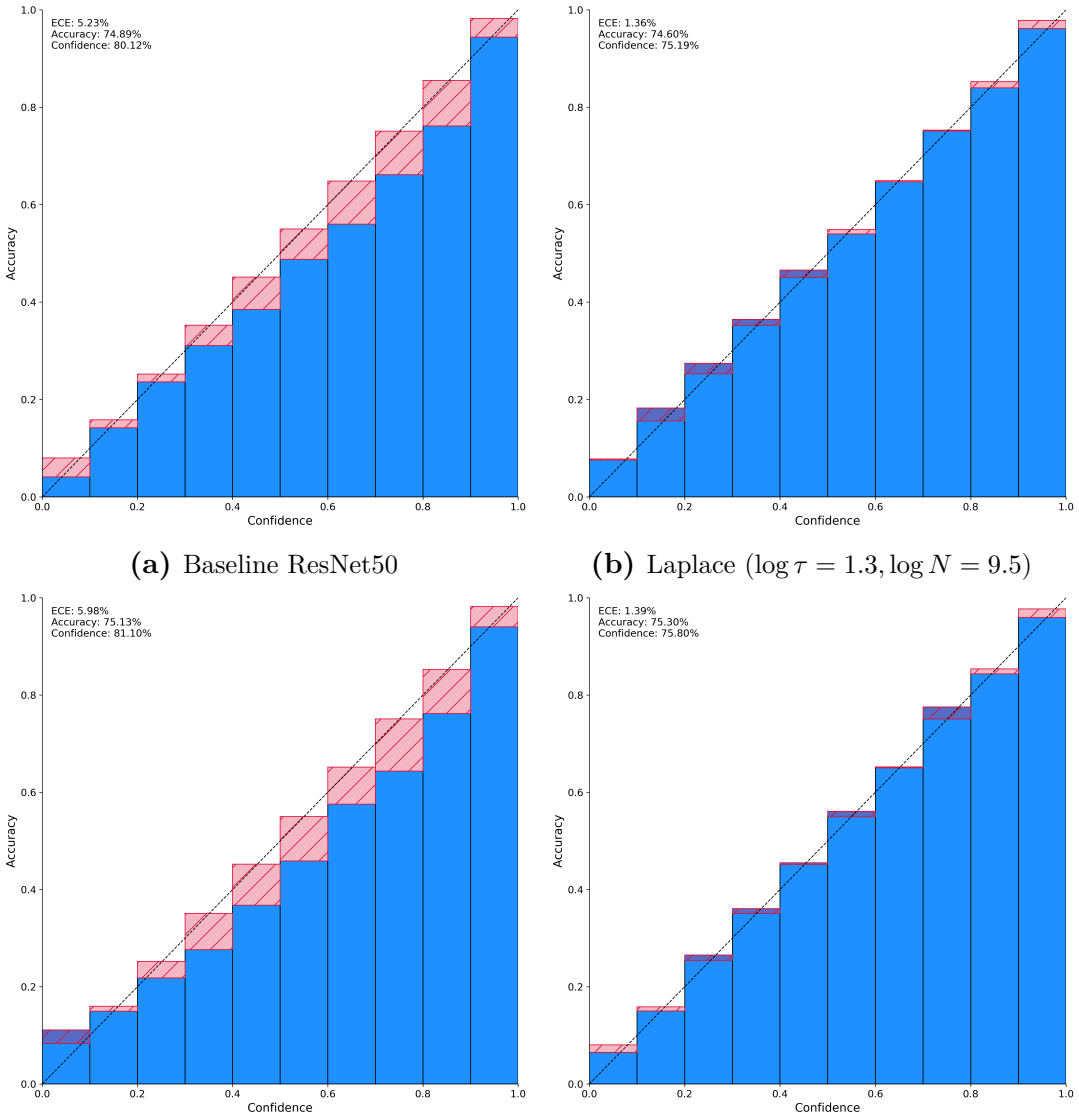


Figure 5.16: Baseline calibration of all considered deep networks.

For an in-depth comparison, the reliability diagrams of ResNet50 and DenseNet169 are show below. On both architectures, the use of Laplace's method reduces the ECE by around 4%, a five times reduction compared to the baseline calibration. Interestingly, the remaining calibration error is mostly caused by slight underconfidence, which is favorable over overconfidence in safety-critical applications.



(a) Baseline ResNet50

(b) Laplace ($\log \tau = 1.3, \log N = 9.5$)

(c) Baseline DenseNet169

(d) Laplace ($\log \tau = 1, \log N = 10$)

Figure 5.17: Calibration comparison of ResNet50 (5.17a, 5.17b) and DenseNet169 (5.17c, 5.17d).

Finally, the improvement of multiple networks from the DenseNet architecture is depicted in figure 5.18. The dashed, slightly transparent lines are baseline calibration results extracted from figure 5.16, while the solid lines show the newly obtained calibration obtained through Laplace approximation.

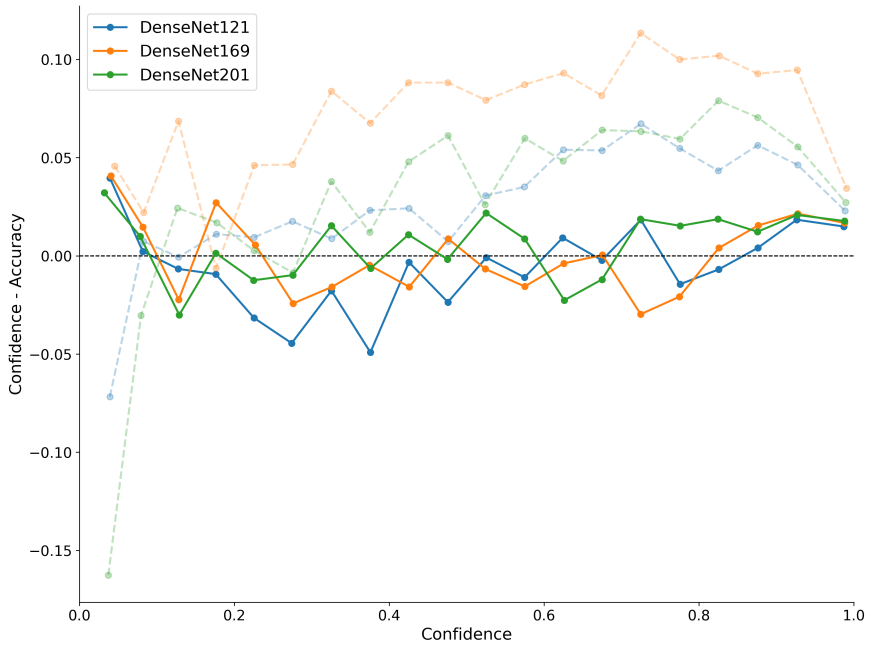
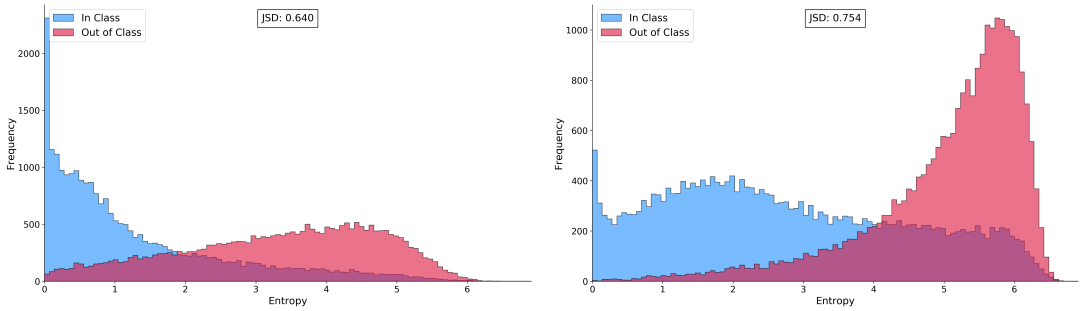


Figure 5.18: Baseline (dashed, transparent) and Laplace (solid) calibration for all DenseNet variants.

5.3.3 Case Study 2: Out-of-distribution detection

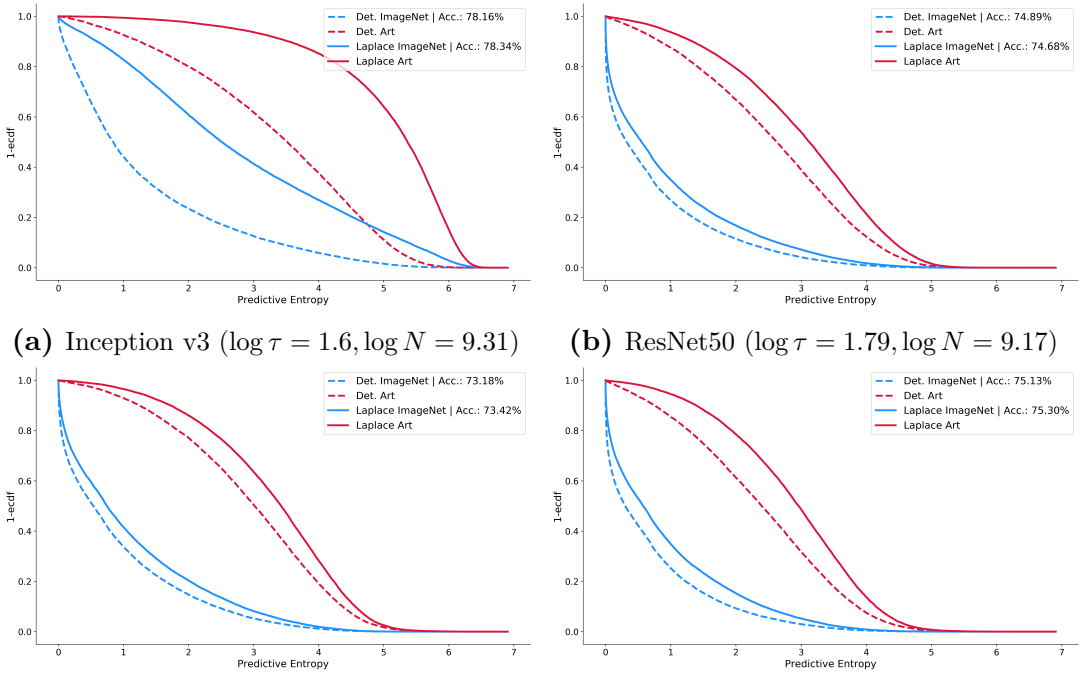
Contrary to the results obtained from the LeNet5 architecture (5.2.6), only slight improvements in in- and out-of-distribution separation could be achieved through Laplace's method on the deep networks. While the entropy of OOD samples could be increased, the entropy on the known examples increased as well, as can be seen below, leading to a moderat increase in JSD.

Using the inverse ECDF vs. predictive entropy visualization introduced in Section 5.2.6 this effect can be observed more closely for a variety of different deep network architectures.



(a) Baseline Inception v3 **(b)** Laplace ($\log \tau = 1.6, \log N = 9.31$)

Figure 5.19: Entropy histograms for in- and out-of-distribution data using the deterministic and probabilistic Inception v3 architecture.



(a) Inception v3 ($\log \tau = 1.6, \log N = 9.31$) **(b)** ResNet50 ($\log \tau = 1.79, \log N = 9.17$)

(c) DN-121 ($\log \tau = -0.48, \log N = 10.24$) **(d)** DenseNet169 ($\log \tau = 1, \log N = 10$)

Figure 5.20: Entropy vs. inverse ECDF for some DNN architectures.

One possible explanation for this behavior is the lower accuracy of the deep networks on the ImageNet dataset compared to LeNet5 on e.g. MNIST. As observed in the previous section, Laplace's method can help to improve the calibration of the predicted class probabilities. A network with an average accuracy of 70% can therefore be expected to also provide an average confidence of 70%. Lower confidence results in higher entropy on the known data, as the probability vector is more uniform. Separating known and unknown data based on entropy can therefore be expected to be more difficult.

Additionally, it might well be the case that the chosen OOD data is more similar to ImageNet than, e.g. notMNIST to MNIST, because ImageNet is such a large and diverse dataset (see Section 5.1.2).

Of course the explanation might be even simpler, i.e. bad weight posterior approximation through Laplace approximation, but the good results on calibration and, as we will see next, resilience to adversarial attacks, seem to suggest otherwise.

5.3.4 Case Study 3: Adversarial attacks

This final section of the empirical analysis focuses on the resilience of the deep network architectures to adversarial attacks both deterministically and under Laplace approximation. The setup is identical to Section 5.2.7, but due to computational constraints, the resolution of the FGSM step size had to be reduced significantly.

All network architectures show a faster and more pronounced increase of predictive entropy under attack compared to their deterministic counterparts. Just like LeNet5, most deterministic networks show a decrease of predictive entropy for larger step sizes, meaning they get increasingly confident while at the same time getting less accurate. Here, deeper variants outperform shallower ones, a trend which is not observable under Laplace approximation, which again is mainly governed by the correct choice of regularization.

Interestingly, the regularization parameters of Laplace approximation that yield the lowest ECE give better performance than those which yield the highest JSD, even though one would expect the opposite, as changing the image intuitively corresponds to creating OOD data. On the other hand, the predictive entropy is governed by the networks calibration, s.t. a network that knows what it knows and does not know can be expected to show higher uncertainty on unknown (adversarial) data.

In contrast to the results obtained from LeNet5, the accuracy does eventually drop to a random level, even when using Laplace approximation, but it does so more gracefully compared to the deterministic network, again showing stronger resilience to adversarial attacks.

It is further visible that all networks under Laplace approximation exhibit higher uncertainty from the start. As those networks are better calibrated than their deterministic counterparts, this lends further support to the hypothesis by Guo et al. (2017), presented in Section 4.3.2, that modern NNs can overfit on NLL by sacrificing calibration.

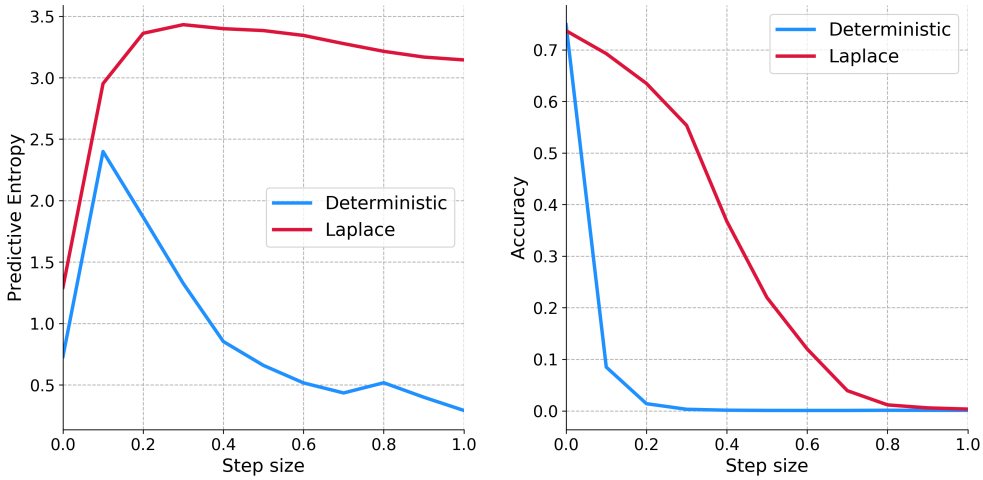


Figure 5.21: Entropy (left), accuracy (right), ResNet50 ($\log \tau = 1.3, \log N = 9$)

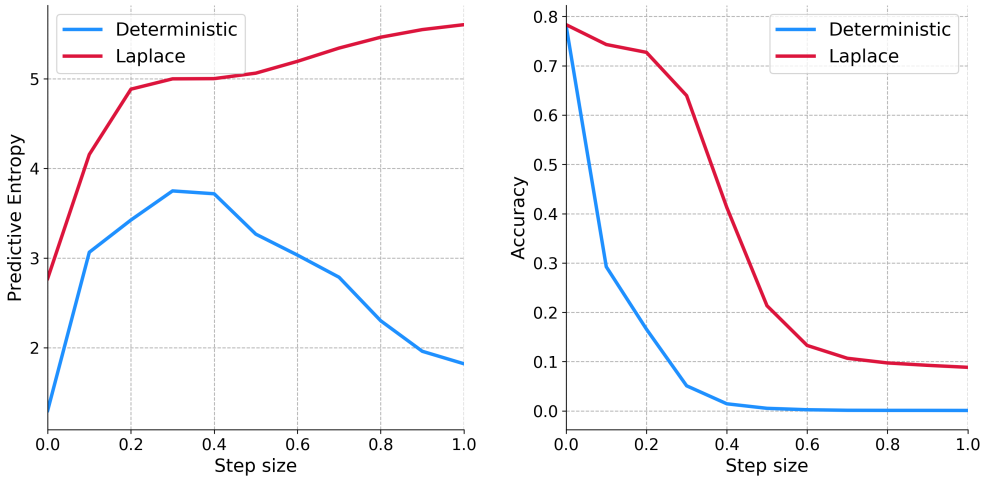


Figure 5.22: Inception v3 ($\log \tau = 1.6, \log N = 9.3$)

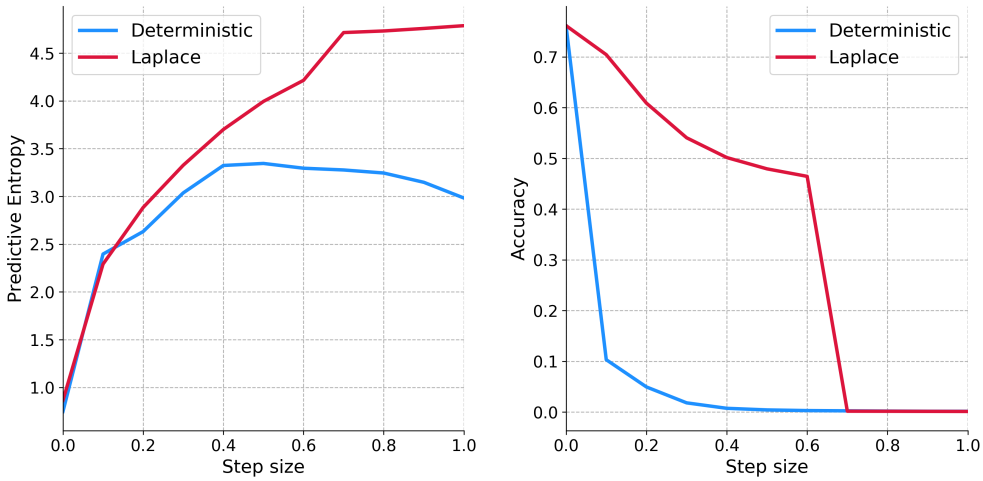


Figure 5.23: DenseNet201 ($\log \tau = -0.44, \log N = 10.75$)

6 Discussion

In this section, the results and insights gained throughout the work are briefly summarized and elaborated on where necessary. We will then take a final theoretical look at what turned out to be a topic of central importance for this thesis: How to understand the hyperparameters of Laplace approximation from a probabilistic perspective and as regularizers and how those interpretations could inform a goal directed search.

6.1 Connecting the dots

Four types of experiments have been performed to assess the suitability of multiple DNN architectures to Laplace approximation.

From characteristics of the loss landscape and by comparing it to the histogram of eigenvalues from the Kronecker factored curvature matrices (5.2.4, 5.3.1), which were inverted to serve as covariance matrices of a layer-wise factored multivariate normal distribution, approximating the posterior distribution over the weights of the NNs (4.1), we tried to predict and explain the subsequent results.

We saw that all networks exhibit a convex loss surface around the learned minimum and are roughly circular or elliptical in shape[1], both being desirable properties for a Gaussian approximation. The main differences could be found by looking at the smoothness and steep- or flatness of the landscapes, which were accurately captured by the aforementioned eigenvalue histograms. Generally, networks with fewer parameters, but not necessarily fewer layers, produced steeper surfaces. This is another important aspect, as the assumed second-order Taylor approximation only captures function properties near the mode while additionally the normal distribution is rounded at its peak, meaning it provides an inferior approximation if the region around the extremum is extremely flat.

We tentatively conjecture that smooth, non-flat, elliptic or circular loss surfaces weakly predict superior quality of uncertainty estimates obtained through Laplace approximation, but further research in this area is definitely needed. The notion is

[1]Keeping in mind that we are looking at an extremely low-dimensional visualizations of the actual surface.

additionally confounded by the importance of the right choice of hyperparameters, a theme running through all obtained results.

The quality of the obtained uncertainty estimates was analyzed by comparing the calibration (5.2.5, 5.3.2), in- and out-of-distribution separation (5.2.6, 5.3.3) and adversarial attack resilience (5.2.7, 5.3.4) between the different network architectures as well as the deterministic and probabilistic outputs of each network.

We saw that the calibration could be drastically improved on most networks, even though they were not overly miscalibrated in the first place. To the contrary, the distance between known and unknown data could only be increased slightly on the deep networks – as the uncertainty on unknown data increased in conjunction with the uncertainty of the known data – though significantly on the small validation network. Moving to the results of the adversarial attacks, Laplace approximation clearly outperformed the deterministic networks across all architectures. Generally, a network that performed well in one experiment also performed well in the others. Among those were all small networks with up to 25 million parameters (see 5.1).

Having obtained those results, we can now compare both the absolute performance of each network as well as its improvement relative to the deterministic baseline. The following table states the deterministic accuracy, ECE and JSD and the evaluation of these metrics using Laplace approximation for all networks[2].

	Accuracy		**ECE**		**JSD**	
Architecture	**Det.**	**Lapl.**	**Det.**	**Lapl.**	**Det.**	**Lapl.**
VGG16	71.24%	-	2.70%	-	0.511	-
VGG19	**71.46%**	71.38%	2.29%	**1.79%**	0.512	-
Inception v3	78.16%	78.16%	1.63%	1.63%	0.640	**0.756**
ResNet50	**74.89%**	74.60	5.23%	**1.36%**	0.440	**0.621**
ResNet101	**76.28%**	76.29%	6.75%	**4.80%**	0.305	-
ResNet152	**76.58%**	76.24%	6.78%	**4.32%**	0.278	-
DenseNet121	73.18%	**73.33%**	3.16%	**1.31%**	0.686	**0.890**
DenseNet169	75.13%	**75.30%**	5.98%	**1.39%**	0.495	**0.741**
DenseNet201	76.09%	**76.16%**	3.93%	**1.54%**	0.636	**0.764**

Table 6.1: Laplace vs. deterministic network comparison.

Numbers in bold denote better performance while a hyphen denotes that no improvement over the deterministic baseline could be achieved. Better calibration also increased the accuracy of the DenseNet variants, while an increase in JSD

[2]Accuracy and ECE obtained from the same set of hyperparameters. JSD obtained from a different set with approx. baseline accuracy

mostly decreased the calibration (except for the DenseNet variants). A possible explanation could be the overfitting on NLL with adverse effects on calibration as explained in Section 4.3.2. This overfitting could be reinforced through the right choice of hyperparameters to increase the distance between in- and out-of-distribution data where low uncertainty is beneficial, but mitigated by another set of hyperparameters, leveraging the variance introduced by averaging the predicted class probabilities of multiple weight posterior samples.

Table 6.1 can be visualized to provide quick access for choosing an appropriate architecture. Well calibrated and accurate networks are on the upper left. The size of a circle informs about the size of the network in millions of parameters while the color provides insight into the ability to separate known from unknown data[3]

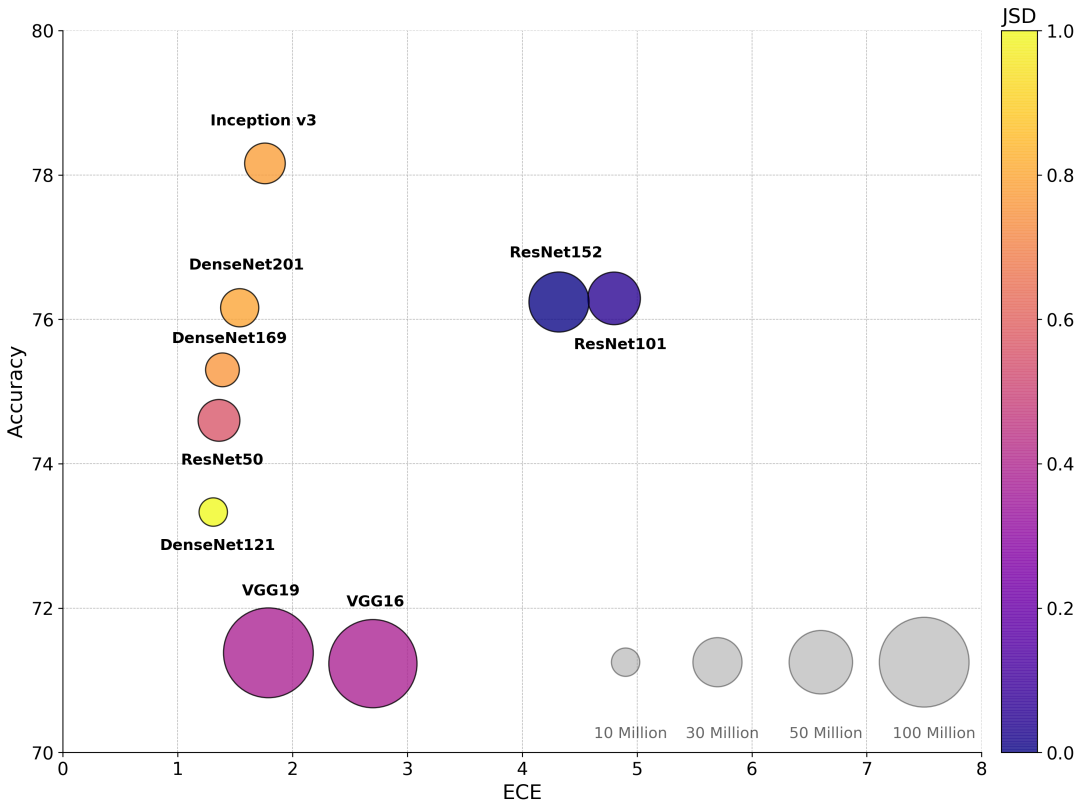


Figure 6.1: Visualization of the network comparison from table 6.1.

One potential reason for the deterioration of performance with a growing number of parameters is the sensitivity of Laplace approximation to the ration between the number of parameters to the size of the available dataset (Ritter et al., 2018). We can think about this intuitively by viewing the Hessian as a parameterized function.

[3]To increase the visual resolution, the values that determine the color are a linear interpolation between the lowest and highest JSD values.

From each data point we produce a noisy sample of this function, constraining its shape. Naturally, a more complex function with more parameters needs more samples to be tightly constrained, so the ratio of the number of parameters to the size of the dataset is important (Ogden, 2018; Ruli et al., 2016). Even more problematic is the fact that each additional parameter introduces a new curvature *dimension*, so we operate under the *curse of dimensionality* (Murphy, 2012, pp. 18, 19).

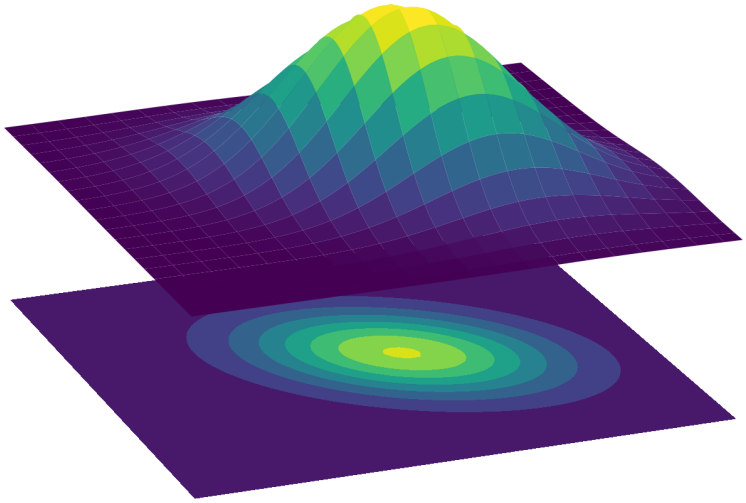
There are several ways to deal with this problem. Two trivial approaches are to acquire more data or to shrink the model size, both of which are often not possible. Another approach is to constrain the function to be approximated by making assumptions about its form. When using Kronecker factored Laplace approximation, we assumed the weight posterior distribution to be of Gaussian functional form and layer-wise factored as well as Kronecker factored within each layer. Other approaches like VI often make even more restrictive assumptions by using multivariate normal distributions with diagonal covariance. Finally, the function can be constrained further through regularization, which we explore next.

6.2 Hyperparameters: A closer look

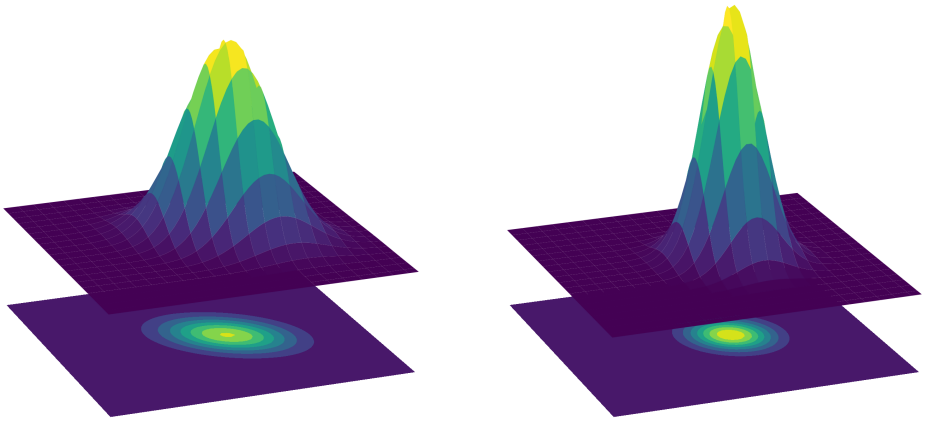
There are at least three ways to think about N and τ, the two parameters of Laplace approximation when implemented as proposed by Ritter et al. (2018). We have already briefly touched upon all of them in Section 4.3.1.

Ultimately, we are interested in high quality uncertainty estimates and therefore, setting both parameters to any value that achieves this goal is desirable. This is the hyperparameter view and has driven the approach of using BO to find such proper values. As it turned out, finding them was both crucial to the performance of Laplace approximation and very difficult. This is partly due to a lack of understanding regarding the effect of adjusting the parameters individually and jointly as well as a further lack of understanding regarding a theoretically founded explanation for what the parameters represent.

An intuitive understanding can be obtained by reducing the dimensionality. As explained in Section 4.2, the curvature matrices are inverted and become the covariance matrices of a Guassian distribution. The pdf of a two-dimensional multivariate normal distribution can be visualized in a 2D plane, connecting points of identical probability to contour lines.



If the variance on the diagonal of the covariance matrix of both parameters is identical, we see a circle and an ellipse otherwise. If the parameters are independent, the off-diagonal of the covariance matrix is zero. Otherwise, entries on the off-diagonal will rotate the ellipse. Increasing N increases the curvature in all directions, therefore shrinking the ellipse (or circle) as the pdf of the Gaussian gets more peaky, i.e. decreasing the uncertainty about the correct value of the parameters. Sampling from this distribution will yield weights closer to those that were learned during the training of the NN. To the contrary, increasing τ only affects the entries on the diagonal of the curvature matrix, i.e. the variances in the covariance matrix. Increasing τ will make the pdf more circular, because the difference between the diagonal entries decreases.



(a) $\tau = 1, N = 2$ **(b)** $\tau = 2, N = 1$

The difference between settings of N and τ however can be subtle depending on the value of both parameters, which explains the linear behavior we saw in Section 5.2.3 and which is also visible for the deep networks (see appendix A.1). Because of their regularizing effect on the uncertainty, the parameters were referred to as *regularizers* throughout this work. Apart from hyperparameters and regularizers, there is a final interpretation for our parameters, namely the probabilistic view.

It assigns τ the meaning of the precision parameter of a multivariate standard normal distribution (Bishop, 2006, pp. 279, 280), which follows the same line of reasoning as the probabilistic interpretation of L_2-regularization (weight decay). We thereby split the posterior distribution again into a data-dependent (scaled) likelihood and prior term.

Irrespective of the chosen interpretation, setting one set of parameters for all curvature factors does not seem reasonable. Not only can the curvatures be of very different magnitude, such that the same amount of damping can have a strong or weak regularizing effect, but the posterior approximation could additionally be more or less wrong in certain regions, requiring different degrees of adjustment. A small experiment was performed to see if multiple sets of hyperparameters (up to the number of curvature factors) can lead to a increased performance, but the resulting higher dimensional search space made optimization even more time consuming and difficult.

Apart from differences between curvature factors, all networks have architectural differences between the first layers, which are usually convolutional and commonly referred to as *feature extractors*, and the final layer, which is usually a fully-connected layer making the predictions. It is therefore interesting to see if any of these layers have a dominant impact on the quality of the uncertainty estimates, i.e. if a full posterior over all weights is actually needed. Tables 6.2, 6.3 and 6.4 compare the performance between those different settings, where *all* means that the posterior of all layers was approximated and sampled from, *conv* means that only the feature extractor layers where considered and *dense* means that only the final prediction layer was used probabilistically. Lastly *none* denotes the deterministic network.

Interestingly, using only the convolutional layers outperforms the usage of all layers in some cases, making Bayesian feature extraction promising. Only using the final layer was almost indistinguishable from the deterministic network results. This indicates either the need for individual regularization, or the limited contribution of the final layer which, despite being only a single layer, still features a large number of weights due to its fully-connected architecture.

DenseNet121				
	All	**Conv**	**Dense**	**None**
Accuracy	**73.33**%	73.24%	73.19%	73.18%
ECE	1.31%	**1.22**%	3.15%	3.16%
JSD	0.769	**0.778**	0.686	0.686

Table 6.2: Effects of limiting Laplace to certain layers on DenseNet121. Hyperparamters: $\tau = \log -1.99$, $N = \log 10.84$.

DenseNet169				
	All	**Conv**	**Dense**	**None**
Accuracy	75.32%	**75.36**%	75.07%	75.13%
ECE	1.47%	**1.42**%	5.98%	5.98%
JSD	**0.729**	0.718	0.496	0.495

Table 6.3: Effects of limiting Laplace to certain layers on DenseNet169. Hyperparamters: $\tau = \log -0.12$, $N = \log 10.04$.

ResNet50				
	All	**Conv**	**Dense**	**None**
Accuracy	74.54%	**74.62**%	74.88%	74.89%
ECE	**1.37**%	1.90%	5.23%	5.23%
JSD	0.452	0.452	0.440	0.440

Table 6.4: Effects of limiting Laplace to certain layers on ResNet50. Hyperparamters: $\tau = \log -3.97$, $N = \log 15.06$.

7 Conclusion and Outlook

In this work the suitability of a variety of DNN architectures to Laplace approximation was studied both empirically and, to a lesser extent, theoretically. The first important result is that Laplace approximation is indeed a practical and scalable method to obtain high quality uncertainty estimates from trained DNNs of various sizes and architectures, and that is tractable even for very large datasets without requiring any changes to the networks themselves.

High resolution loss surface plots were contrasted with an eigenvalue study of the layer-wise factored curvature matrices, providing preliminary insights into the connection between network architectures and their weight posterior distributions.

The quality of the obtained uncertainty estimates was empirically verified on three different benchmarks, showing that Laplace approximation can be used to improve calibration, OOD detection and resilience to adversarial attacks across multiple networks' sizes and architectures.

Finally, BO was introduced as an efficient way to find suitable hyperparameters for the regularization of Laplace's method, which proved to be the Achilles' heel of the algorithm. The properties and implications of those parameters were studied and first results from single- and multi-layer as well as layer-wise regularization were presented.

There remain many interesting avenues to be explored. Using the approximate posterior distribution to evaluate the evidence (marginal likelihood) of each network for the automatic selection of the regularization parameters as proposed by Ritter et al. (2018) would make the entire hyperparameter optimization superfluous. Alternatively, a proper objective function for BO could greatly increase its utility. Additionally, Laplace approximation could be solely used on the networks' convolutional layers to obtain Bayesian feature extractors e.g. for transfer learning. Furthermore, the uncertainty estimates could be used to select the most helpful data in an active learning scenario or for Bayesian model compression (Federici et al., 2017; Louizos et al., 2017). Lastly, uncertainty in image segmentation tasks could be evaluated by decomposing the predictive uncertainty into epistemic and aleatoric uncertainty, as proposed by Kendall and Gal (2017) and Kwon et al. (2018), but using Laplace approximation instead of MC dropout.

Bibliography

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283.

Azevedo-Filho, A. and Shachter, R. D. (1994). Laplace's method approximations for probabilistic inference in belief networks with continuous variables. In *Uncertainty Proceedings 1994*, pages 28–36. Elsevier.

Barber, D. (2012). *Bayesian reasoning and machine learning*. Cambridge University Press.

Barber, D. and Bishop, C. M. (1998). Ensemble learning in bayesian neural networks. *Nato ASI Series F Computer and Systems Sciences*, 168:215–238.

Benatan, M., Daresbury, S.-T., and Pyzer-Knapp, E. O. (2018). Practical considerations for probabilistic backpropagation. In *Third workshop on Bayesian Deep Learning (NeurIPS 2018), Montreal, Canada*.

Berger, J. O. (2013). *Statistical decision theory and Bayesian analysis*. Springer Science & Business Media.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305.

Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

Bishop, C. M. et al. (1995). *Neural networks for pattern recognition*. Oxford University Press.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*.

Botev, A., Ritter, H., and Barber, D. (2017). Practical gauss-newton optimisation for deep learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 557–565. JMLR. org.

Buntine, W. L. and Weigend, A. S. (1991). Bayesian back-propagation. *Complex systems*, 5(6):603–643.

Carter, S., Armstrong, Z., Schubert, L., Johnson, I., and Olah, C. (2019). Activation atlas. *Distill*. https://distill.pub/2019/activation-atlas.

Chen, S.-W., Chou, C.-N., and Chang, E. Y. (2018). Ea-cg: An approximate second-order method for training fully-connected neural networks. *arXiv preprint arXiv:1802.06502*.

Chollet, F. et al. (2015). Keras. `https://keras.io`.

Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM.

Csáji, B. C. (2001). Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24:48.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

Dangel, F. and Hennig, P. (2019). A modular approach to block-diagonal hessian approximations for second-order optimization methods. *arXiv preprint arXiv:1902.01813*.

Dawid, A. P. (1982). The well-calibrated bayesian. *Journal of the American Statistical Association*, 77(379):605–610.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

Denker, J. S. and Lecun, Y. (1991). Transforming neural-net output levels to probability distributions. In *Advances in neural information processing systems*, pages 853–859.

Der Kiureghian, A. and Ditlevsen, O. (2009). Aleatory or epistemic? Does it matter? *Structural Safety*, 31(2):105–112.

Federici, M., Ullrich, K., and Welling, M. (2017). Improved bayesian compression. *arXiv preprint arXiv:1711.06494*.

Gal, Y. (2016). *Uncertainty in deep learning*. PhD thesis, University of Cambridge.

Gal, Y. and Ghahramani, Z. (2016a). Bayesian convolutional neural networks with bernoulli approximate variational inference. *arXiv preprint arXiv:1506.02158*.

Gal, Y. and Ghahramani, Z. (2016b). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*, pages 1050–1059.

Gilks, W. R., Richardson, S., and Spiegelhalter, D. J. (1996). Introducing markov chain monte carlo. *Markov chain Monte Carlo in practice*, 1:19.

Girija, S. S. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *Software available from tensorflow. org*.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

Graves, A. (2011). Practical variational inference for neural networks. In *Advances in neural information processing systems*, pages 2348–2356.

Grimmett, H., Triebel, R., Paul, R., and Posner, I. (2016). Introspective classification for robot perception. *The International Journal of Robotics Research*, 35(7):743–762.

Grosse, R. and Martens, J. (2016). A kronecker-factored approximate fisher matrix for convolution layers. In *International Conference on Machine Learning*, pages 573–582.

Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. (2017). On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1321–1330. JMLR. org.

Gupta, A. K. and Nagar, D. K. (1991). *Matrix variate distributions*. Chapman and Hall/CRC.

Hanin, B. (2017). Universal function approximation by deep neural nets with bounded width and relu activations. *arXiv preprint arXiv:1708.02691*.

Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A., et al. (2014). Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Hernández-Lobato, J. M. and Adams, R. (2015). Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International Conference on Machine Learning*, pages 1861–1869.

Hernández-Lobato, J. M., Li, Y., Rowland, M., Hernández-Lobato, D., Bui, T., and Turner, R. (2016). Black-box α-divergence minimization.

Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Kingsbury, B., et al. (2012). Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 29.

Hinton, G. and Van Camp, D. (1993). Keeping neural networks simple by minimizing the description length of the weights. In *in Proc. of the 6th Ann. ACM Conf. on Computational Learning Theory*. Citeseer.

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

Horvitz, E., Heckerman, D., and Langlotz, C. (1986). A framework for comparing alternative formalisms for plausible reasoning. In *AAAI*, pages 210–214.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017a). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708.

Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S., et al. (2017b). Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7310–7311.

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. (2016). Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Jaynes, E. T. (1986). Bayesian methods: General background.

Jylänki, P., Nummenmaa, A., and Vehtari, A. (2014). Expectation propagation for neural networks with sparsity-promoting priors. *The Journal of Machine Learning Research*, 15(1):1849–1901.

Karpathy, A. (2019a). CS231n convolutional neural networks for visual recognition. [Online; accessed July 2, 2019].

Karpathy, A. (2019b). CS231n convolutional neural networks for visual recognition. [Online; accessed July 2, 2019].

Kendall, A. and Gal, Y. (2017). What uncertainties do we need in bayesian deep learning for computer vision? In *Advances in neural information processing systems*, pages 5574–5584.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P., Salimans, T., and Welling, M. (2015). Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pages 2575–2583.

Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images. Technical report, Citeseer.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Kwon, Y., Won, J.-H., Kim, B. J., and Paik, M. C. (2018). Uncertainty quantification using bayesian neural networks in classification: Application to ischemic stroke lesion segmentation.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, pages 6402–6413.

Lampinen, J. and Vehtari, A. (2001). Bayesian approach for neural networks—review and case studies. *Neural networks*, 14(3):257–274.

Laplace, P. S. (1774). Mémoire sur la probabilité des causes par les événements (1774). *Œuvres compl*, pages 27–65.

LeCun, Y. (1998). The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436.

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

Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T. (2018). Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems*, pages 6389–6399.

Li, Y. and Gal, Y. (2017). Dropout inference in bayesian neural networks with alpha-divergences. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2052–2061. JMLR. org.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2016). Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer.

Louizos, C., Ullrich, K., and Welling, M. (2017). Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems*, pages 3288–3298.

Lu, Z., Pu, H., Wang, F., Hu, Z., and Wang, L. (2017). The expressive power of neural networks: A view from the width. In *Advances in Neural Information Processing Systems*, pages 6231–6239.

MacKay, D. J. (1992a). Bayesian interpolation. *Neural computation*, 4(3):415–447.

MacKay, D. J. (1992b). The evidence framework applied to classification networks. *Neural computation*, 4(5):720–736.

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

MacKay, D. J. (1995). Probable networks and plausible predictions—a review of practical bayesian methods for supervised neural networks. *Network: computation in neural systems*, 6(3):469–505.

Maddox, W., Garipov, T., Izmailov, P., Vetrov, D., and Wilson, A. G. (2019). A simple baseline for bayesian uncertainty in deep learning. *arXiv preprint arXiv:1902.02476*.

Martens, J. (2014). New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193*.

Martens, J. and Grosse, R. (2015). Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Miller, D., Dayoub, F., Milford, M., and Sünderhauf, N. (2018a). Evaluating merging strategies for sampling-based uncertainty techniques in object detection. *arXiv preprint arXiv:1809.06006*.

Miller, D., Nicholson, L., Dayoub, F., and Sünderhauf, N. (2018b). Dropout sampling for robust object detection in open-set conditions. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–7. IEEE.

Minka, T. P. (2001). Expectation propagation for approximate bayesian inference. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 362–369. Morgan Kaufmann Publishers Inc.

Molchanov, D., Ashukha, A., and Vetrov, D. (2017). Variational dropout sparsifies deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2498–2507. JMLR. org.

Murphy, A. H. (1973). A new vector partition of the probability score. *Journal of applied Meteorology*, 12(4):595–600.

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

Naeini, M. P., Cooper, G., and Hauskrecht, M. (2015). Obtaining well calibrated probabilities using bayesian binning. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

Neal, R. (1992). Bayesian training of backpropagation networks by the hybrid monte carlo method (tech. rep. crg-tr-92-1). *Toronto, Canada: Department of Computer Science, University of Toronto*.

Neal, R. M. (1995). *Bayesian learning for neural networks*. University of Toronto.

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

Ogden, H. (2018). On the error in laplace approximations of high-dimensional integrals. *arXiv preprint arXiv:1808.06341*.

Olah, C., Mordvintsev, A., and Schubert, L. (2017). Feature visualization. *Distill*. https://distill.pub/2017/feature-visualization.

Olah, C., Satyanarayan, A., Johnson, I., Carter, S., Schubert, L., Ye, K., and Mordvintsev, A. (2018). The building blocks of interpretability. *Distill*. https://distill.pub/2018/building-blocks.

Papernot, N., Goodfellow, I., Sheatsley, R., Feinman, R., and McDaniel, P. (2016). cleverhans v1.0.0: an adversarial machine learning library. *arXiv preprint arXiv:1610.00768*.

Pearce, T., Zaki, M., Brintrup, A., and Neel, A. (2018). Uncertainty in neural networks: Bayesian ensembling. *arXiv preprint arXiv:1810.05546*.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788.

Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99.

Ritter, H., Botev, A., and Barber, D. (2018). A scalable laplace approximation for neural networks.

Ruli, E., Sartori, N., Ventura, L., et al. (2016). Improved laplace approximation for marginal likelihoods. *Electronic Journal of Statistics*, 10(2):3986–4009.

Rumelhart, D. E., Durbin, R., Golden, R., and Chauvin, Y. (1995). Backpropagation: The basic theory. *Backpropagation: Theory, architectures and applications*, pages 1–34.

Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. (1988). Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252.

Sheldon, R. et al. (2002). *A first course in probability*. Pearson Education India.

Shridhar, K., Laumann, F., and Liwicki, M. (2019). A comprehensive guide to bayesian convolutional neural network with variational inference. *arXiv preprint arXiv:1901.02731*.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Soudry, D., Hubara, I., and Meir, R. (2014). Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights. In *Advances in Neural Information Processing Systems*, pages 963–971.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. A. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

Wan, L., Zeiler, M., Zhang, S., Le Cun, Y., and Fergus, R. (2013). Regularization of neural networks using dropconnect. In *International conference on machine learning*, pages 1058–1066.

Wang, S. and Manning, C. (2013). Fast dropout training. In *international conference on machine learning*, pages 118–126.

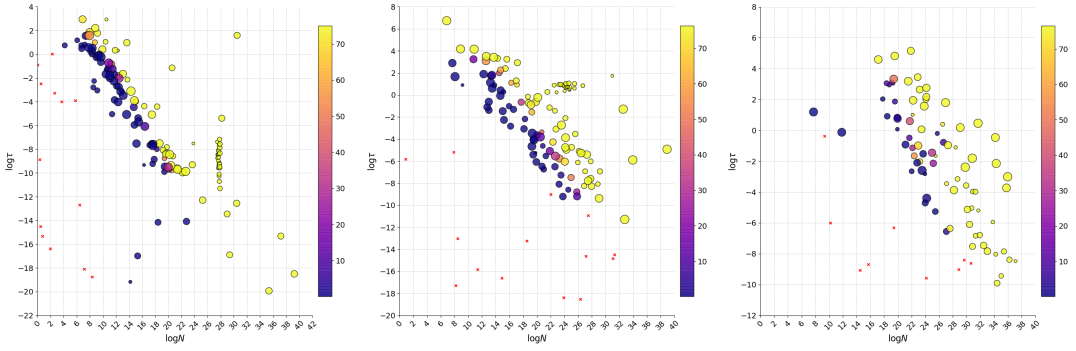
Wu, R., Yan, S., Shan, Y., Dang, Q., and Sun, G. (2015). Deep image: Scaling up image recognition. *arXiv preprint arXiv:1501.02876*.

Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. *arXiv preprint arXiv:1605.07146*.

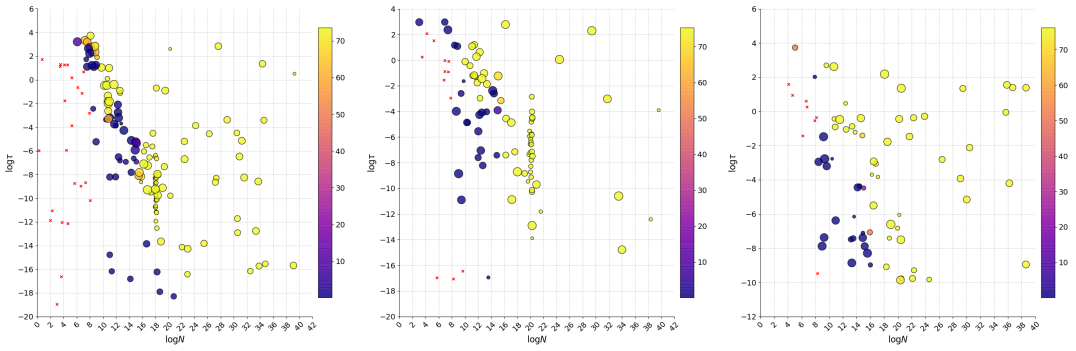
Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer.

A Appendix

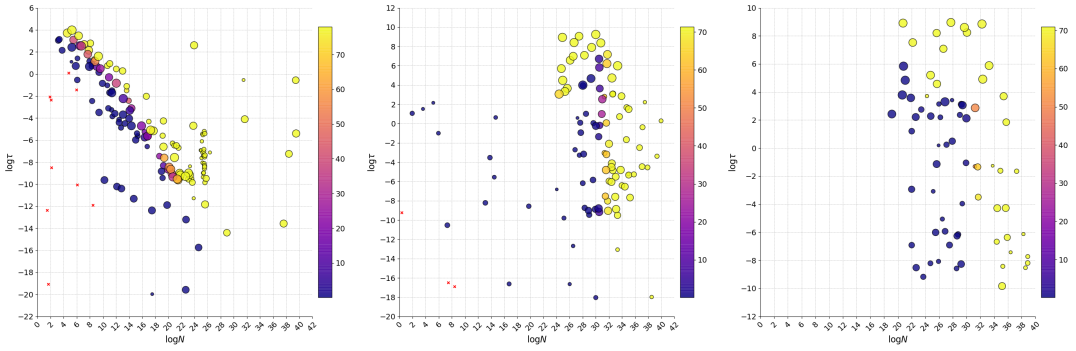
A.1 Hyperparameter search



(a) ResNet50 **(b)** ResNet101 **(c)** ResNet152

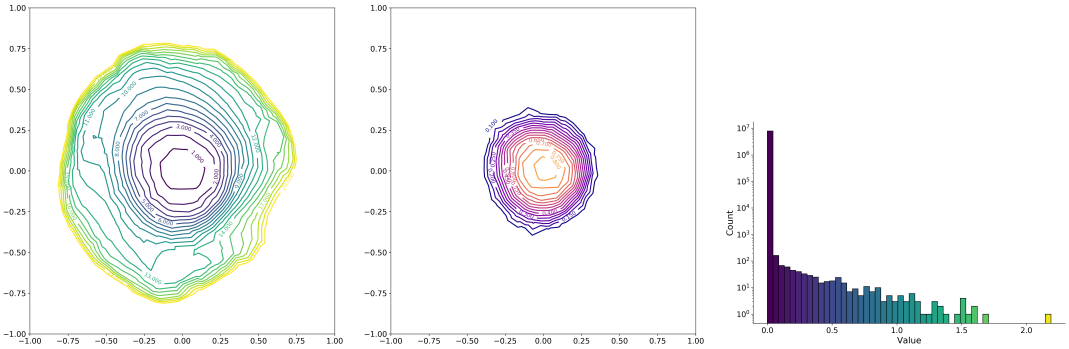


(d) DenseNet121 **(e)** DenseNet169 **(f)** DenseNet201

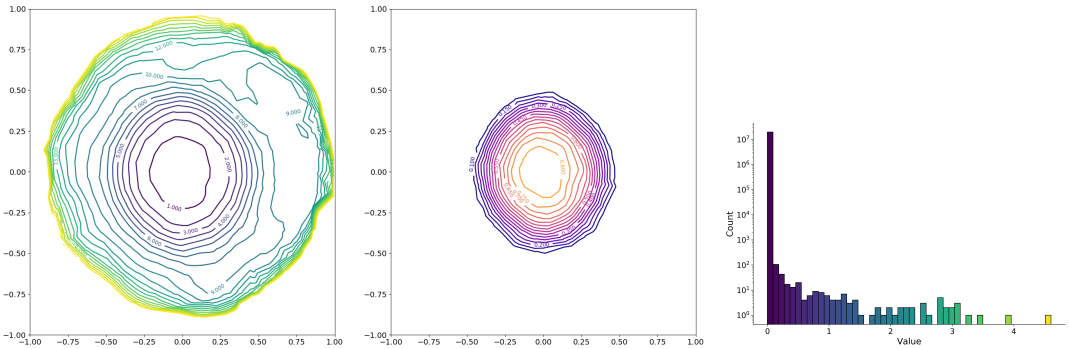


(g) Inception v3 **(h)** VGG16 **(i)** VGG19

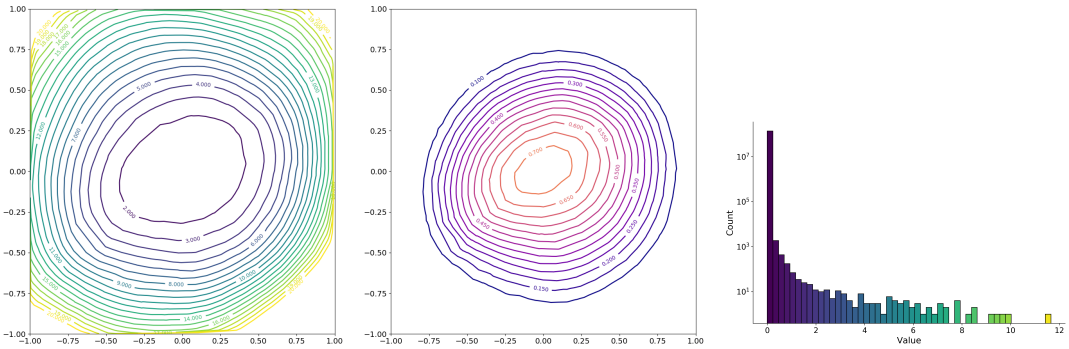
A.2 Loss landscapes and eigenvalue histograms



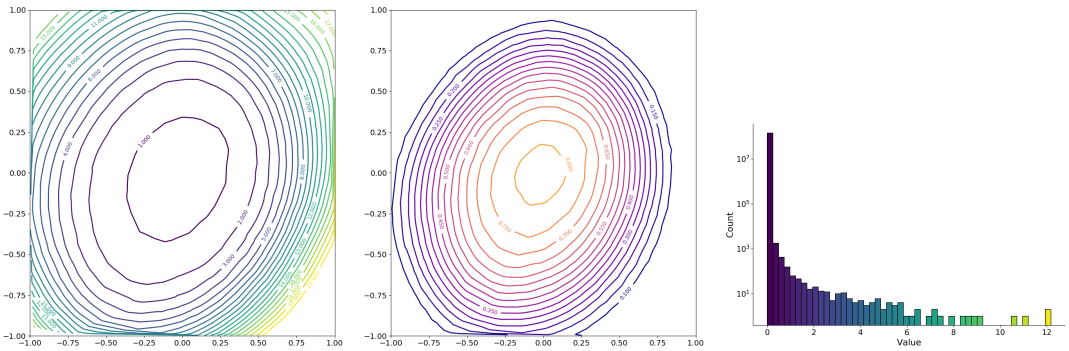
(a) DenseNet121 loss (b) Accuracy (c) Eigenvalues



(d) DenseNet201 loss (e) Accuracy (f) Eigenvalues

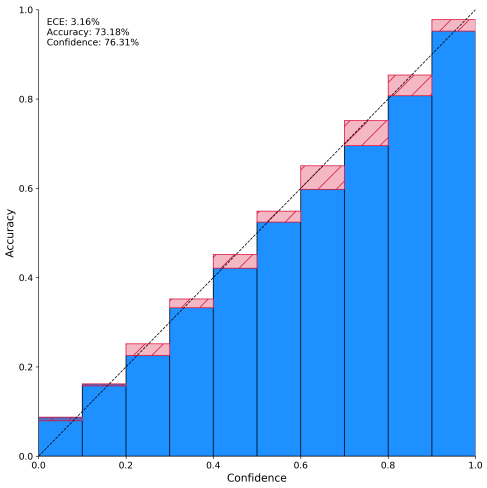


(g) VGG16 loss (h) Accuracy (i) Eigenvalues

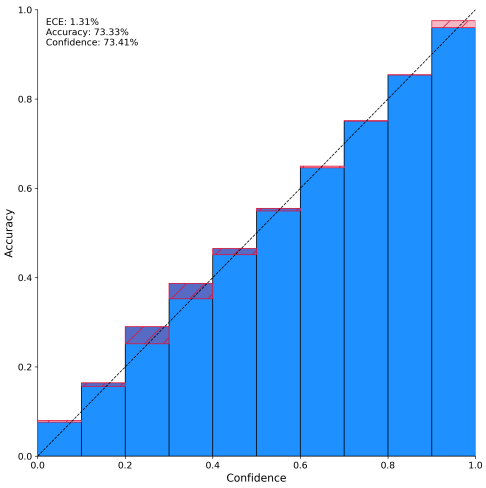


(j) VGG19 loss (k) Accuracy (l) Eigenvalues

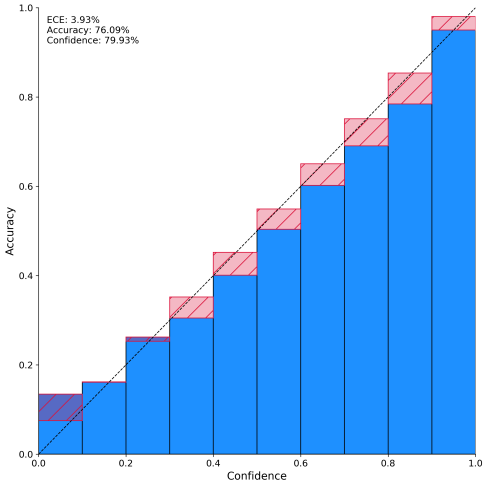
A.3 Reliability and calibration



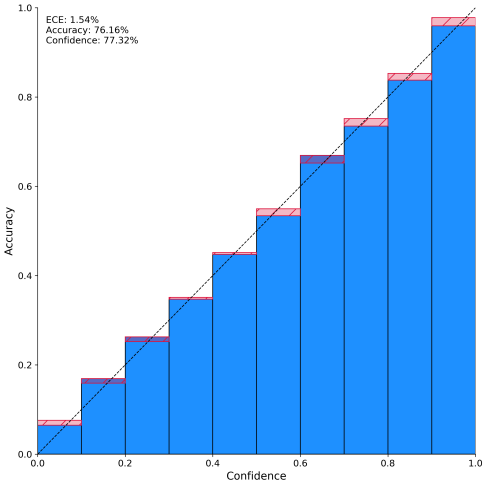
(a) DenseNet121 baseline



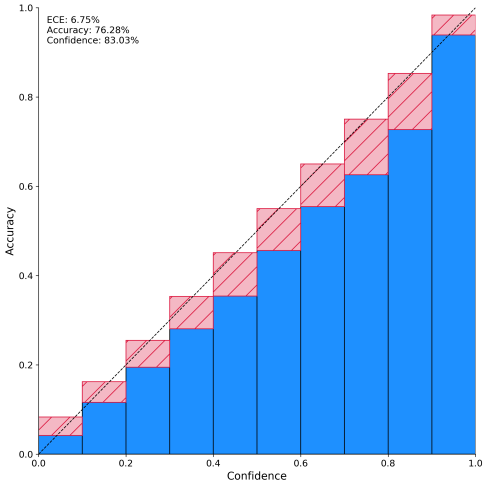
(b) Laplace ($\tau = -1.99, N = 10.84$)



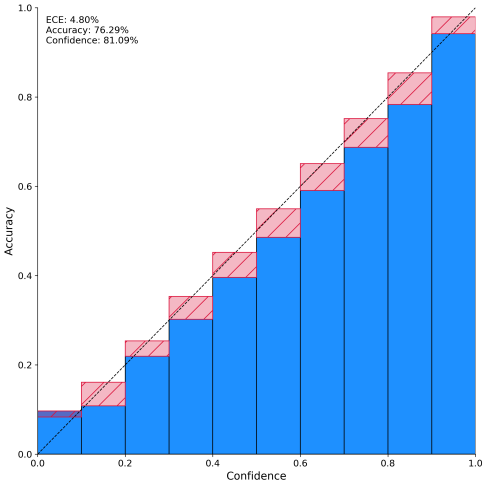
(c) DenseNet201 baseline



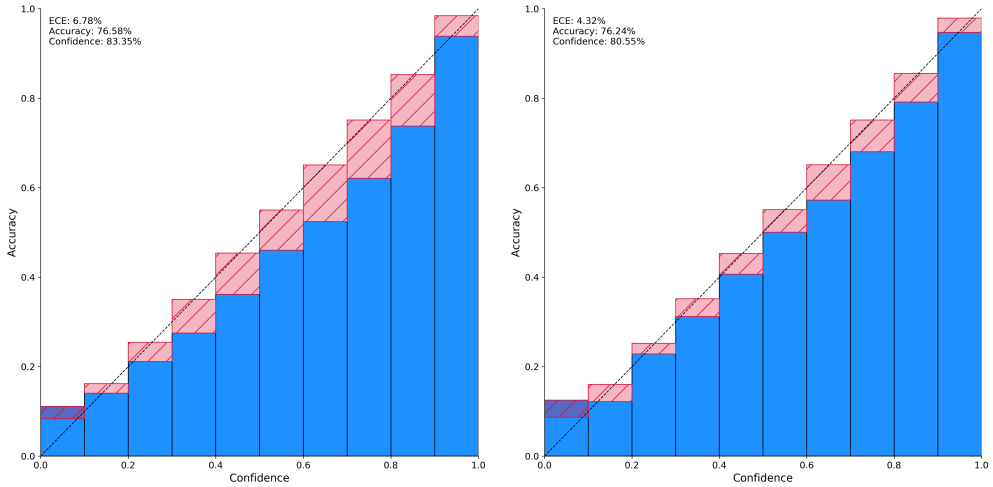
(d) Laplace ($\tau = -0.44, N = 10.75$)



(e) ResNet101 baseline

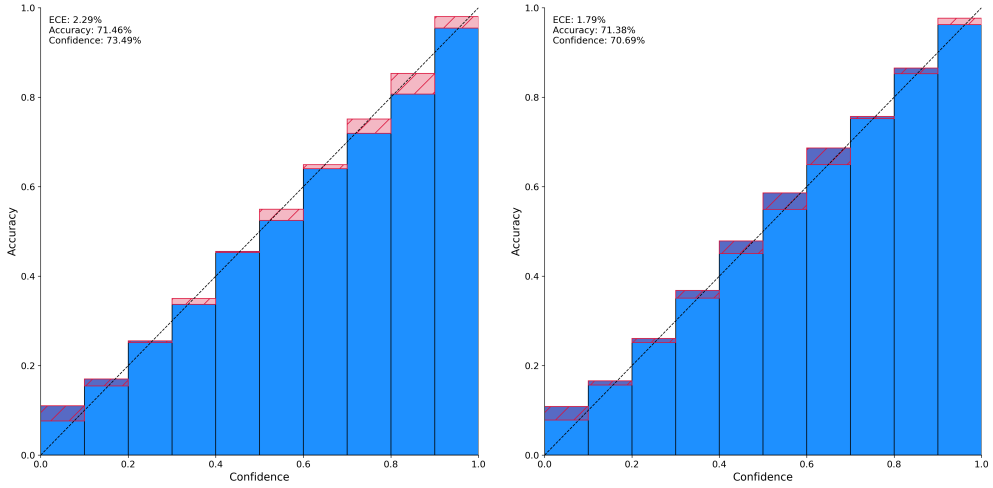


(f) Laplace ($\tau = 3.51, N = 12.64$)



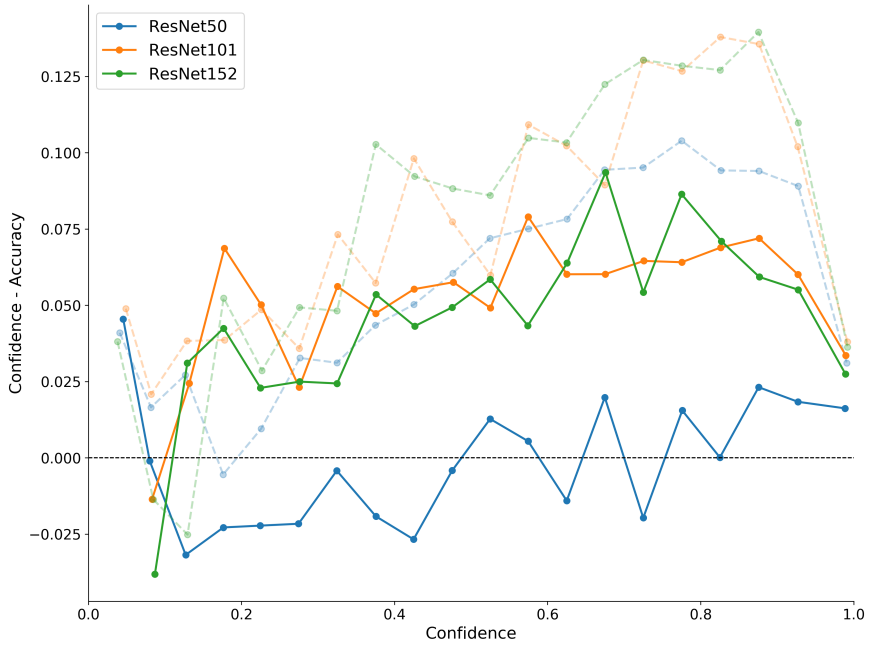
(a) ResNet152 baseline

(b) Laplace ($\tau = 0.06, N = 23.45$)



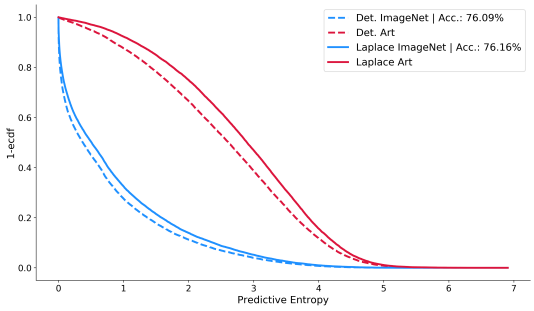
(c) VGG19 baseline

(d) Laplace ($\tau = 8.91, N = 20.76$)

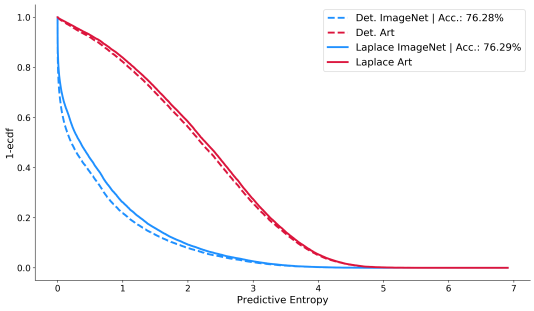


(e) Calibration comparison ResNets

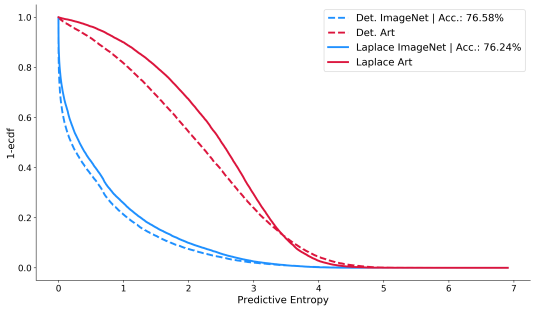
A.4 Predictive entropy



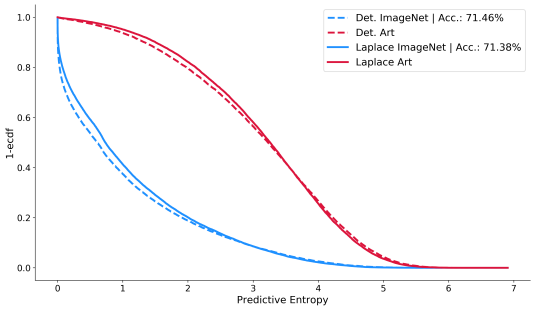
(a) DNet201 ($\tau = -0.44, N = 10.75$)



(b) ResNet101 ($\tau = 3.51, N = 12.64$)

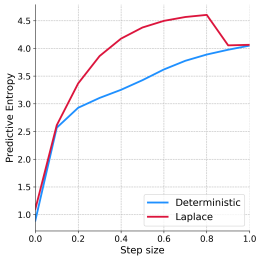


(c) ResNet152 ($\tau = 0.06, N = 23.45$)

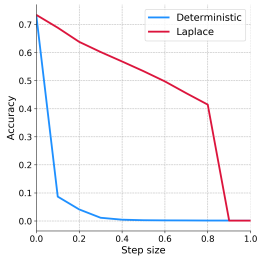


(d) VGG19 ($\tau = 8.91, N = 20.76$)

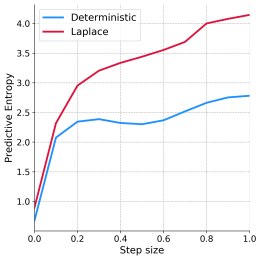
A.5 Adversarial attacks



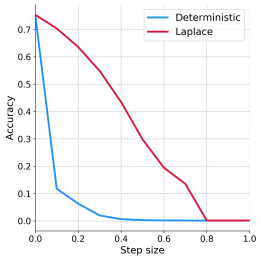
(a) DenseNet121



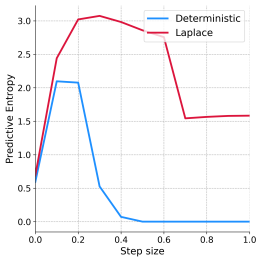
(b) Accuracy



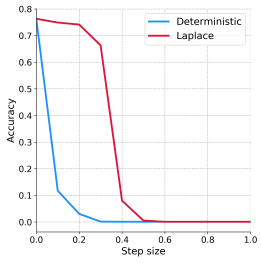
(c) DenseNet169



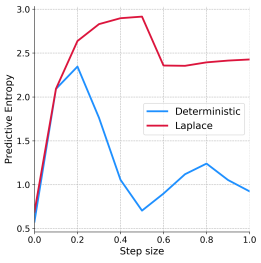
(d) Accuracy



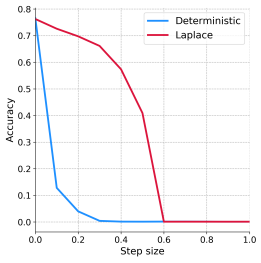
(e) ResNet101



(f) Accuracy



(g) ResNet152



(h) Accuracy