Master's Thesis

in

Computational and Data Science

Estimating Uncertainty of Deep Learning Multi-Label Classifications using Laplace Approximation

for the attainment of the degree

Master of Science (M.Sc.)

submitted by

Ferdinand Rewicki

*06/04/1986 in Friedrichroda (Germany)

ferdinand.rewicki@uni-jena.de

supervised by

M.Sc. Jakob Gawlikowski

Prof. Dr. Joachim Giesen

written at

Chair for Theoretical Computer Science II

Department of Mathematics and Computer Science

Friedrich-Schiller-Universität Jena

in cooperation with

DLR German Aerospace Center

Institute of Data Science

Machine Learning Group

29/07/2021

Abstract

With the huge successes of deep learning and its application in critical areas such as medical diagnosis or autonomous driving and in fields with noisy and very varying data such as remote sensing, the need for reliable confidence statements about such model's predictions becomes apparent. Therefore, uncertainty estimation methods for neural networks have raised rising interest in the machine learning community. While various methods for regression and multi-class classification tasks have been published, the field of multi-label classification has hardly been considered yet. In this work, we derive the *Kronecker-factored Laplace approximation* in the multi-label setting, a method to approximate the intractable posterior distribution over the parameters of neural networks. We employ this method in the remote sensing domain and estimate the model uncertainty of eight deep neural networks that have been trained on an aerial scene classification dataset. By comparing the probabilistic classifiers to their deterministic counterparts, we evaluate the potential for using the uncertainty estimates to improve the calibration of those classifiers as well as the out-of-distribution detection. We found that we can improve the calibration for overconfident classifiers whereas for underconfident ones, this method might not be beneficial. Furthermore, the ability to improve the separation from in- and out-of-distribution data seems to be depending on the depth of the neural network within one model family.

Keywords— Multi-Label Classification, Bayesian Deep Neural Networks, Uncertainty Estimation, Laplace Approximation, Remote Sensing

Contents

Acronyms

AUROC area under the receiver operating characteristic curve

BNN Bayesian neural network

BO Bayesian optimisation

CNN convolutional neutral network

DNN deep neural network

ECE expected calibration error

FIM Fisher information matrix

FPR false positive rate

GGN generalised Gauss Newton matrix

MAP maximum a posteriori

MLE maximum likelihood estimate

MLP multi-layer perceptron

MLRSNet Multi-Label Remote Sensing Dataset

MND matrix variate normal distribution

NLL negative log likelihood

OOD out-of-distribution

p.s.d. positive semi definite

ROC receiver operating characteristic

TPR true positive rate

Nomenclature

D	Dimensionality of the data
C	Number of classes
$x_i \in \mathbb{R}^D$	single observation
$y_i \in \{0,1\}^C$	single multi-label
$\hat{y}_i \in \{0,1\}^C$	predicted multi-label
$\hat{p}_i \in [0,1]^C$	class probabilities vector
$X \in \mathbb{R}^{N \times D}$	Observations
$Y \in \{0,1\}^{N \times C}$	Targets / (Multi) Labels
$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$	Dataset (training data, known by the model)
$\mathcal{D}^* = \{(x_i^*, y_i^*)\}_{i=1}^N$	Dataset (test data, unknown by the model)
f_θ	A neural network, parametrised by θ
$z = f_\theta(x)$	The logits of a neural network when presented input x
θ	The parameters of a neural network
θ^*	The MAP estimate for the parameters θ
N_θ	Number of parameters
$\mathcal{L}(y, z)$	Loss function $\mathcal{L}$
$H_{g(x)}(x_0)$	Hessian of function $g(x)$ at x_0
H_g	same as $H_{g(x)}(x_0)$ if context is clear
T	Number of Monte Carlo Samples

1 Introduction

Multi-label classification is the classification task that aims at labelling data with multiple class labels simultaneously and is a common problem in many real world applications [Wu and Zhou, 2017]. Being a challenging problem due to the consideration of class overlap in the feature space [Ghoshal et al., 2019], the exponential growing number of possible label combinations and often poorly balanced data [Alazaidah and Ahmad, 2016] it is attractive to be approached using deep learning techniques. With the enormous successes of deep learning over the last decade, deep neural networks (DNNs) have started to be applied in critical areas such as medical image analysis for disease diagnosis, decision making in autonomous driving or earth observation for understanding climate change. Due to their black-box nature and often observed over- or under-confidence [Guo et al., 2017, Ritter et al., 2018, Gawlikowski et al., 2021], the development of methods for understanding and quantifying uncertainty of DNNs became a vibrant research field in the machine learning community.

While regression and multi-class tasks have been addressed by many publications such as [Gal and Ghahramani, 2015, Blundell et al., 2015, Ritter et al., 2018, Humt, 2019], the field of multi-label classification has rarely been in focus yet. With this work we aim to reduce this gap and therefore evaluate Bayesian neural networks (BNNs) based on *Laplace approximation* for quantifying uncertainties on real-world multi-label classification problems. The contribution of this thesis consists of:

- the derivation of the *Kronecker-factored Laplace approximation* in the multi-label classification setting of neural networks,
- `tf-laplace`, a Python package for using the Kronecker-factored Laplace approximation with TensorFlow [Abadi et al., 2015] 2.x compatible models[1] which is available on GitHub at `https://github.com/ferewi/tf-laplace`,
- a synthetic dataset that allows to visualise the uncertainties and their distribution, especially along class-overlap regions and

[1] The implementation is currently limited to models with dense and convolutional layers.

- the application to the domain of remote sensing by comparing eight modern DNNs trained on the aerial scene classification dataset MLRSNet.

We start by introducing some fundamental concepts from probability theory and neural networks as well as giving an overview over different approaches for building multi-label classifiers in Chapter 2. After presenting related literature in Chapter 3 we derive the Kronecker-factored Laplace approximation in the multi-label classification setting in Chapter 4. We describe the approximation of the unknown posterior distribution while focusing on the characteristics of the multi-label setting. In Chapter 5, we use the derived method to visualise the obtained uncertainties and evaluate the potential of using the uncertainty estimates to improve (a) the calibration of the multi-label classifier and (b) the out-of-distribution (OOD) detection for a synthetic dataset we created and for the large-scale remote sensing dataset MLRSNet. After discussing the experimental results in Chapter 6 we conclude by giving a summary as well as an outlook to future work.

For the best of our knowledge, this is the first work applying Laplace approximation on multi-label classification as well as to the remote sensing domain.

2 Background

In this Chapter we will introduce the basic concepts of probability theory and artificial neural networks that are relevant for this work as well as their application for multi-label classification in general and remote sensing in particular. As describing these topics in full depth would by far exceed the scope of this thesis, we will refer to the appropriate sources where needed. We start presenting some probability theory basics in Section 2.1 and introduce artificial neural networks in Section 2.2. After describing the different types of uncertainties in Section 2.3 and defining the multi-label classification problem in Section 2.4, we will describe the remote sensing domain in Section 2.5 which is the field of application of this work.

2.1 Probability Theory

Probability theory is the mathematical framework to work with in the presence of uncertainty. Probability theory provides the tools for quantifying uncertainty and to reason from uncertain statements. In this section we will introduce some basic concepts which are relevant for this thesis. For a more thorough introduction please refer to a machine learning textbook such as [Goodfellow et al., 2016], [Bishop, 2006] or [Murphy, 2021].

The need for dealing with uncertainty in the field of machine learning comes from (a) the system to be modelled is inherent stochastic (e.g. in quantum mechanics), (b) the system being modelled can not be fully observed, (c) the model built for a system being incomplete [Goodfellow et al., 2016, p. 52]. To deal with these uncertainties we adopt the Bayesian interpretation of probability in which probability represents a *degree of belief* and allows us to quantify the certainty about an event to occur [Murphy, 2021, p.31]; [Goodfellow et al., 2016, p.53]. As opposed to the Bayesian interpretation of probability, the *frequentist* sees probabilities as the relative frequencies with that events occur when repeated multiple times [Murphy, 2021, p.31].

When training a machine learning model that is parametrised by parameters θ, we are interested in inferring the posterior distribution ever these parameters. Given some observed data $\mathcal{D}$ and some prior belief $p(\theta)$ about the values for θ, we can use *Bayes' Rule* to update that belief in the light of the observations [Barber, 2012, p.18]:

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \ , \tag{2.1}$$

where $p(\mathcal{D}|\theta)$ represents the *likelihood* for $\mathcal{D}$ being generated using parameters θ and $p(\theta)$ the prior distribution over θ. $p(\mathcal{D}) = \int_\theta p(\mathcal{D}|\theta)p(\theta)$ is called the evidence and works as a normalisation constant to ensure $p(\theta|\mathcal{D})$ being a valid probability density [Bishop, 2006, p.22]. Thus, the posterior is proportional to the likelihood times the prior:

$$p(\theta|\mathcal{D}) \propto p(\mathcal{D}|\theta)p(\theta) \ . \tag{2.2}$$

The *maximum a posteriori (MAP)* estimate θ^* is given by the set of parameters that maximises the posterior: $\theta^* = \arg\max_\theta p(\theta|\mathcal{D})$. For a uniform prior $p(\theta) = c$, the MAP is equivalent to the *maximum likelihood estimate (MLE)* which is the θ that maximises the likelihood $p(\mathcal{D}|\theta)$, as the prior in 2.1 is not changing with θ anymore [Barber, 2012, p.18]. However, the MAP is a point estimate. We obtain the *posterior predictive distribution* for predicting the output y^* on new data inputs x^*, by marginalising w.r.t. the posterior distribution [Bishop, 2006, p.279]

$$p(y^*|x^*, \mathcal{D}) = \int p(y^*|x^*, \theta)p(\theta|\mathcal{D})d\theta \ , \tag{2.3}$$

which is known as *Full Bayesian Inference*. As this integral is intractable in most cases, we will use a technique called *Monte Carlo Approximation* for approximating the posterior predictive distribution in Section 4.1.

2.2 Neural Networks

Feed forward neural networks are the essential machine learning models used in this work. We are going to introduce their basic structure as well as how neural networks are trained in Section 2.2.1 and describe a variant that is mainly used in image analysis called *convolutional neutral networks* in Section 2.2.2. In Section 2.2.3 we will present *Bayesian neural networks*, a probabilistic version of neural networks.

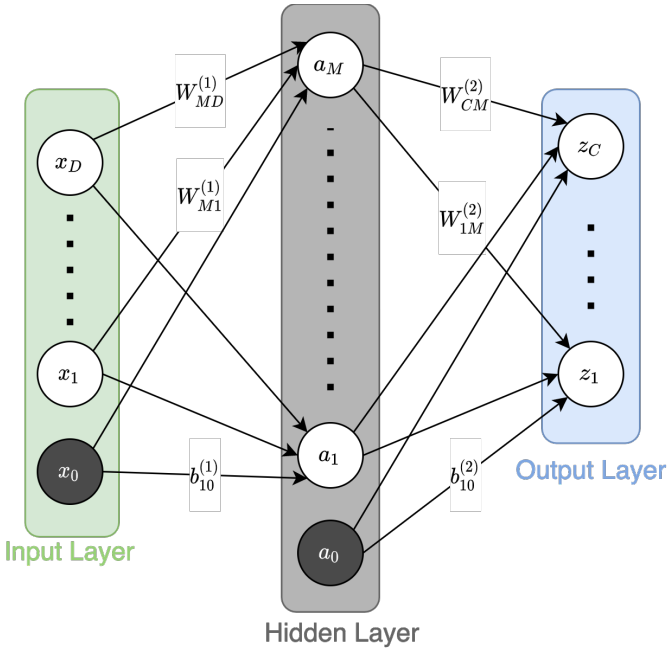


Figure 2.1: Network diagram of a multi-layer perceptron (MLP) with one hidden layer. The vertices represent the input, hidden and output variables, whereas the parameters θ are represented by the edges between the vertices. The arrows denote the direction of information flow from left to right. The parameters $\theta^{(\lambda)}_{i0}, i \in \{1, \ldots, max(M, C)\}, \lambda \in \{1, 2\}$ are called *biases* while all other parameters are called *weights*.

2.2.1 Feed Forward Neural Networks

Feed forward neural networks[1], are used to approximate some unknown function f^* for mapping an input x to a target $y = f^*(x)$. The mapping of a neural network is defined by $\hat{y} = f_\theta(x)$ where θ are the parameters of the function f that should be learned to obtain the best approximation to f^*. The parameters θ are also called the *weights* of the neural network. In a feed-forward network, the function $f_\theta(x)$ is composed by a number of functions $f^{(L)}(f^{(L-1)}(...(f^{(1)}(x))...))$ which are organized in a chain. $f^{(\lambda)}(x) = f_{\theta_\lambda}(x)$ denotes the function at *layer* λ. [Goodfellow et al., 2016, p.164]; [Murphy, 2021, p.413]

Figure 2.1 shows a schematic representation of a neural network with on hidden layer as a network diagram. The middle layer is called *hidden* as the variables $a_0, \ldots, a_M$

[1] We will use the term "neural network" synonym to "feed forward neural network" as those are the only class of neural networks considered in this work.

are not observed. Each layer λ consists of one or several *neurons* which are the basic building blocks of neural networks. Each neuron i in layer λ calculates an activation

$$a_i^{(\lambda)} = \phi^{(\lambda)}(h_i^{(\lambda)}), \text{ with } h_i^{(\lambda)} = (\sum_{j=1}^{D} W_{ij}^{(\lambda)} a_j^{(\lambda-1)}) + b_i^{\lambda} \ , \tag{2.4}$$

by applying an activation function $\phi^{(\lambda)}$ to the linear pre-activations $h^{(\lambda)}$ in layer λ. The pre-activations are the linear combination of the activations in the previous layer $a_i^{(\lambda-1)}$ and the weights $W_i^{(\lambda)}$ of layer λ plus a bias term $b_i^{(\lambda)}$. We can express this for an entire layer using matrix notation

$$a^{(\lambda)} = \phi^{(\lambda)}(\theta^{(\lambda)} a^{(\lambda-1)}) \ , \tag{2.5}$$

with the bias terms being absorbed into the parameter matrix $\theta^{(\lambda)} = [W^{(\lambda)} b^{(\lambda)}]$ by appending a 1 to the vector $a^{(\lambda-1)}$. $\theta^{(\lambda)}$ are called the *parameters* of layer λ. The variables of the first layer $x_i = a_i^{(0)}$ are called *inputs* while those in the final layer $z_i = h_i^{(L)}$ are denoted as *outputs* or *logits*. The activation functions ϕ are usually nonlinear, as neural networks with linear activations reduce to a single neuron[2] [Goodfellow et al., 2016, p.168]. Depending on the application, the logits of the neural network z_i are transformed according to the needs of the applications. For standard regression problems this is done by using the identity $\hat{y}_i = z_i$ whereas in multi class classification the softmax function

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}} \tag{2.6}$$

is applied to normalize the logits z_i to a categorical distribution over the C possible classes. For binary and multi-label classification problems, the logits[3] z_i are passed through the logistic sigmoid function

$$\sigma(z_i) = \frac{1}{1 + e^{-z_i}} \tag{2.7}$$

to obtain a probability for the i^{th} class being present in the input.

When training a neural network on a dataset $\mathcal{D}$, we want to learn the parameters θ of f_θ to obtain the best approximation f_θ for f^*. The data points $(x, y) \in \mathcal{D}$ are

[2] Given two linear functions $f^{(1)}(x) = W^\intercal x$ and $f^{(2)}(h) = h^\intercal w$, the composed function $f(x) = f^{(2)}(f^{(1)}(x)) = f^{(2)}(W^\intercal x) = x^\intercal W w$, which we can rewrite as $f(x) = x^\intercal w'$ with $w' = Ww$.

[3] In the binary classification case this would be a single logit, thus $i = 1$.

noisy, approximate examples $y \approx f^*(x)$ of f^* evaluated at point x and the goal of the training process is to minimise the error between $f_\theta(x)$ and y. For this purpose, a loss function $\mathcal{L}$ is minimised which is depending on the problem to be solved. A common probabilistic interpretation of this procedure is that by minimising the error between the true label y and the neural network output $f_\theta(x)$ for data points (x, y) of a dataset $\mathcal{D}$, we are maximising the likelihood that the data given in $\mathcal{D}$ was produced using the parameters θ for f_θ. This is done by minimising the negative log likelihood (NLL)

$$\mathcal{L}(\theta) = -\log p(\mathcal{D}|\theta) = -\sum_{i=1}^{N} \log p(y_i|x_i;\theta) \ . \tag{2.8}$$

The training is usually done iteratively using first-order optimisation procedures, e.g. the well-known *stochastic gradient descent* algorithm. The training examples are passed through the neural network and the gradient $\nabla_\theta \mathcal{L}$ of the NLL w.r.t. the parameters θ is determined and used to update to parameters according to the learning rule

$$\theta \leftarrow \theta - \epsilon \nabla_\theta \mathcal{L} \ , \tag{2.9}$$

with ϵ being the learning rate. The gradient of the NLL w.r.t. the parameters θ can be computed by using the *backpropagation* algorithm. For further details on the training process and an in-depth introduction on neural networks, please refer to a machine learning textbook such as [Goodfellow et al., 2016], [Bishop, 2006] or [Murphy, 2021]. We will come back to the internals of neural networks and the relationship between the binary cross entropy loss function and the negative log likelihood in Chapter 4.

2.2.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) [LeCun et al., 1999] are a special form of neural networks that employ a *convolution* operation instead of the matrix multiplication (2.5) in at least one of their layers [Goodfellow et al., 2016, p.326]. They are used to process data with a grid-like topology like timeseries or images, which are the number one use case for CNNs.

The 1-dimensional continuous convolution is defined as an operation between two real-valued functions $f, g : \mathbb{R}^D \to \mathbb{R}$ as [Murphy, 2021, p.454]

$$[f * g](z) = \int_{\mathbb{R}^D} f(u)g(z-u)du \tag{2.10}$$

and can be though of as "blending" the two functions. In CNN terminology, f is often called the *filter* or *kernel* and g the *input* of the convolution [Goodfellow et al., 2016, p.327]; [Murphy, 2021, p.454]. As in the field of image analysis we usually use images having two spatial dimensions I as inputs, we also want to use two-dimensional kernels K for the convolution. Furthermore, we are dealing with a finite set of discrete data so that the integral in (2.10) becomes a finite sum [Goodfellow et al., 2016, p.328]

$$[K * I](i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} K(i,j)I(i-m, j-n) \ , \tag{2.11}$$

with $M \times N$ being the dimension of the kernel. The values of the kernel are the parameters that should be learned from training.

Convolution convey three properties that help to improve machine learning systems:

1. *Sparse interactions* refers to reducing the number of parameters compared to the MLP by using kernels that are smaller than the inputs to be processed.
2. *Parameter sharing* describes that the same parameter is used in multiple multiplications as opposed to the MLP case, where every parameter is only used once in the matrix multiplication. This reduces the number or parameters further which is beneficial as it improves the memory requirements of the model as well as its statistical efficiency [Goodfellow et al., 2016, p.330].
3. *Equivariance* to translations that asrises from the form of parameter sharing, helps to preserve information about the location of input features [Murphy, 2021, p.461].

CNNs consist of several convolutional layers which are built mostly of three parts. First the input is convoluted with several kernels that produce a set of linear pre-activations which are then run through a non-linear activation function such as the Rectified linear activation function [Goodfellow et al., 2016, p.355]. Finally, these activation are passed through a pooling function that reduces the size of the output (often called *feature maps*) further. The early layers in a deep convolutional neural network are considered to detect low-level features such as edges and corners while

more complex features are extracted by later layers. All of the eight DNNs that we use for our experiments in Chapter 5 are CNNs as we apply multi-label classification to aerial image data.

2.2.3 Bayesian Neural Networks

BNNs [Buntine and Weigend, 1991] extend classical neural networks by combining them with the Bayesian learning approach [Gawlikowski et al., 2021] and are a method for quantifying model uncertainty of neural networks. Instead of training the neural network in a maximum likelihood manner on a dataset $\mathcal{D}$, a probability distribution $p(\theta|\mathcal{D})$ over the network parameters θ is inferred by assuming a prior distribution $p(\theta)$ and applying Bayes' Rule (2.1). [Gawlikowski et al., 2021] differentiate between three types of BNNs, depending on how the posterior distribution is inferred.

Variational Inference [Hinton and Van Camp, 1993, Barber and Bishop, 1998] methods approximate the posterior distribution $p(\theta|\mathcal{D})$ by finding a distribution $q(\theta)$ from a pre-defined parametrised family of distributions that is close to the true posterior distribution. The distance between two distributions p and q is given by the Kullback-Leibler divergence $KL(q||p)$:

$$KL(q||p) = \mathbb{E}_q\Big[\log \frac{q(\theta)}{p(\theta|\mathcal{D})}\Big] \,, \tag{2.12}$$

which can not be optimised directly due to the posterior distribution in the denominator of (2.12). Instead, the *evidence lower bound*

$$L = \mathbb{E}_q\Big[\log \frac{p(\mathcal{D}|\theta)}{q(\theta)}\Big] \tag{2.13}$$

is minimised which is equal to the Kullback-Leibler divergence up to a constant: $KL(q||p) = -L + \log p(y|x)$ [Gawlikowski et al., 2021].

Sampling Approaches, also called *Monte Carlo methods*, represent uncertainty without a parametric model by yielding a representation of the posterior distribution from which samples can be drawn [Gawlikowski et al., 2021]. An example for a sampling method is the Markov Chain Monte Carlo sampling (MCMC) algorithm, which allows to draw samples from the desired distribution by recording states from a Markov chain.

Laplace Approximation [Denker and LeCun, 1990, MacKay, 1992] aims at approximating the true posterior distribution over the parameters $p(\theta|\mathcal{D})$ with a multivariate Gaussian around a local mode of that distribution. This method has the advantage that it can be applied post-hoc to already trained neural networks that use an exponential family loss function (e.g. cross entropy or mean squared error) and piece-wise linear activation functions such as ReLU. We will present this method in-depth in Chapter 4 and derive a scalable version that can be used with modern DNNs.

2.3 Epistemic & Aleatoric Uncertainty

When speaking about uncertainty in neural networks, we mainly distinguish between two forms of uncertainty:

1. *Epistemic* or model uncertainty comes from a lack of domain-specific knowledge about what type of model or parameters suit(s) the underlying problem best [Gawlikowski et al., 2021].

2. *Aleatoric* or data uncertainty refers to the intrinsic, irreducible randomness or partial observability of the data generating process [Murphy, 2021, p.7]; [Depeweg, 2019].

While we mostly can not avoid the latter, we have full influence on how we create or train our model and can reduce the model uncertainty e.g. by increasing the amount of data used for training. The *predictive* uncertainty subsumes both uncertainty types and describes the uncertainty in a prediction. As the model uncertainty represents what a model is missing in terms of domain-specific knowledge it covers in-domain uncertainty as well as out-of-domain uncertainty, where the former describes the uncertainty about inputs drawn from the same distribution as the data the model was trained on [Gawlikowski et al., 2021]. Out-of-distribution uncertainty on the other hand refers to inputs being drawn from a distribution far away from the training distribution[4] [Gawlikowski et al., 2021]. We will apply BNNs to model the epistemic uncertainty of DNNs and employ those among others to separate in-distribution from out-of-distribution data in Chapter 5.

[4]We will refer to those inputs as *out-of-distribution data*.

2.4 Multi-Label Classification

Multi-label classification is the classification task that, as opposed to the multi-class or binary classification, aims at labelling input data with *multiple* labels simultaneously [Wu and Zhou, 2017]. The applications include many real-world problems such as *text categorisation* where articles should be tagged with a list of topics [Schapire and Singer, 2000], *music information retrieval* where a piece of music is associated with different moods or genres [Turnbull et al., 2008] or *image classification* where images are labelled according to their content e.g. with "beach" and "sunset" [Boutell et al., 2004]. Before we give an overview of different approaches for building multi-label classification models, we will formulate the general problem.

Given a dataset $\mathcal{D} \subseteq \mathcal{X} \times \mathcal{Y}$ with $\mathcal{X}$ being a set of examples to be classified and $\mathcal{Y} \subseteq \{0,1\}^C$ a set of C-dimensional binary vectors with $y = (y_1, \ldots, y_c) \in Y$ being called a *multi-label* and the components y_i *micro-label* [BakIr et al., 2007, p.107]; [Boutell et al., 2004]. For a pair $(x^{(j)}, y^{(j)}) \in \mathcal{D}$ the micro-label $y_i^{(j)} = 1$ indicates, that the i^{th} class is assigned to the j^{th} example. Let further $\mathcal{M}$ be the set off classifiers for labelling $x \in \mathcal{X}$ with multi-labels $y \in \mathcal{Y}$. The multi-label classification task aims at finding that classifier $M \in \mathcal{M}$, that minimises a distance (e.g. the hamming distance) between $h(x^*)$ and y^* when predicting on an unseen example (x^*, y^*) [Boutell et al., 2004]. Existing methods for multi-label classification can be grouped into two main categories.

Problem transformation methods solve the multi-label problem by turning it into multiple single-class problems. While *Binary Relevance* breaks down the problem of assigning $k \leq C$ labels into solving C independent binary classification problems [Tomás et al., 2014], the *Label Powerset* method considers every label combination as a separate class in a multi-class classification problem with 2^C classes [Tsoumakas and Katakis, 2007].

Adapted Algorithms on the other hand extend or change existing single-class classification algorithms in a way to solve the multi-label case. For example is the *ML-kNN* algorithm [Zhang and Zhou, 2005] a multi-label version of the k-nearest-neighbours (kNN) algorithm, which uses the kNN for each of the C labels independently [Tsoumakas and Katakis, 2007].

The multi-label setup for neural networks also falls into the latter category. In this setting, the network is trained on a dataset $\mathcal{D} = (x_i, y_i)_{i=1}^n \subseteq \mathcal{X} \times \mathcal{Y}$ providing a multi-label $y^{(i)}$ for each training example $x^{(i)}$. For a multi-label problem with C

possible classes, the output layer of the neural network consists of C neurons. We obtain the probabilities for each class $c \in C$ by applying the sigmoid function (2.7) to the output z_c of the neural network, also known as the logits:

$$\hat{p}_c = \sigma(z_c) = \frac{1}{1 + e^{-z_c}} \ . \tag{2.14}$$

The network is trained by using the binary cross entropy loss $\mathcal{L}_{BCE}$ which is defined as

$$\mathcal{L}_{BCE}(y, z) = \sum_{c=1}^{C} -y_c \log \sigma(z_c) - (1 - y_c) \log(1 - \sigma(z_c)) \ , \tag{2.15}$$

where $y \in \{0, 1\}^C$ is the true multi-label and $z \in \mathbb{R}^C$ is the logits vector at the output of the neural network. As the logits z_i of the neural network f_θ are conditionally independent given the data x and parameters θ [Monteiro et al., 2020], the model assumes independence between the classes and is therefore comparable to the setting in *Binary Relevance*. In this work, we will present the Laplace approximation Method to estimate the posterior distribution of neural networks in the multi-label classification setting in Chapter 4 and will apply that method to synthetic and real-world datasets in Chapter 5. To the best of our knowledge, this is the first work addressing the uncertainty estimation for multi-label classification by employing Laplace approximation.

2.5 Remote Sensing

Remote sensing deals with the detection and monitoring of physical characteristics of the earth using sensors on satellites or aircrafts. Such sensors include optical sensors (multi- and hyperspectral), Lidar or synthetic aperture radar (SAR) [Zhu et al., 2017]. Over the past years, deep learning techniques have heavily been adopted by the remote sensing community for tasks including hyperspectral image analysis, multi-modal data fusion or 3-D reconstruction [Zhu et al., 2017]. Remote sensing meets different challenges and specifics such as the combination of multi-modal data sources including those with differing geometries and content, the importance of the time component of those measurements as well as those measurements often being physical or biochemical quantities [Zhu et al., 2017]. Also, remote sensing data is geo-located, which means that each pixel in an image corresponds to a spatial coordinate which allows the fusion with other data from other information systems or sensors [Zhu et al., 2017].

In this work, we address the task of *aerial scene classification* which aims to automatically label aerial imagery with semantic categories to promote the understanding of high resolution remote sensing data [Xia et al., 2017]. We apply the Laplace approximation method that we present in Chapter 4 to a remote sensing scene classification problem in Section 5.4. We will use the uncertainty estimates obtained from applying a BNN to investigate the potential for improving (a) the calibration of the multi-label classifier and (b) the detection and separation from in- and out-of-distribution data.

3 Related Work

In this section we will present an overview over related research in the field of Bayesian neural networks as uncertainty estimation methods and multi-label classification. After introducing some key works on Laplace approximation in Section 3.1 we describe developments in the related areas of variational inference and Monte Carlo methods in Section 3.2. We conclude this overview by looking at existing works on uncertainty estimation for multi-label classification in Section 3.3.

3.1 Laplace Approximation

The idea of Laplace's method to approximate integrals of the form $\int_a^b f(t)e^{-\lambda g(t)}$ was first described in [Laplace, 1774]. In the field of machine learning [Denker and LeCun, 1990] and [MacKay, 1992] pioneered using Laplace's method to approximate the true posterior distribution over the parameters of a neural network. The basic idea is to approximate the unknown distribution with a multivariate Gaussian around a known mode of that distribution, which is found by point-wise optimisation. The core of the method is the estimation of the Hessian of the negative log-likelihood function. While [MacKay, 1992] and [Denker and LeCun, 1990] used networks with a relatively small number of parameters and could therefore compute the Hessian for their network directly, this approach fails for modern scale DNNs due to their huge number of parameters. Several approximations of the Hessian have been published, such as diagonal approximations (e.g. [Becker and Lecun, 1989]) as well as those including off-diagonal elements ([Liu and Nocedal, 1989, Roux and Fitzgibbon, 2010]).

[Martens and Grosse, 2015] and [Botev et al., 2017] proposed a block-wise approximation of the Hessian by showing that under certain conditions the diagonal blocks of the generalised Gauss Newton matrix (GGN) and Fisher information matrix (FIM) are equal to those of the Hessian. They further showed, that for a single data point these diagonal blocks are Kronecker factored. [Ritter et al.,

2018] transferred this approach for approximating the Hessian from the second-order optimisation field to the field of uncertainty estimation in neural networks. They presented the *Kronecker-factored Laplace approximation*, a version of the Laplace approximation that scales to the size of modern deep neural networks and which is the workhorse of this thesis. [Ritter et al., 2018] applied the method to a wide-residual network [Zagoruyko and Komodakis, 2016] trained on the *CIFAR-100* dataset [Krizhevsky, 2009] and showed that this method can be used to improve the detection of out-of-distribution examples in multi-class classification.

[Humt, 2019] applied the Kronecker-factored Laplace approximation two nine modern DNN trained on *ImageNet* [Russakovsky et al., 2015] and showed that the calibration of the classifier as well as the detection of out-of-distribution examples could be improved by this method.

3.2 Bayesian Neural Networks

Bayesian neural networks, as described in Section 2.2.3, combine classical, deterministic neural networks with Bayesian learning and can be divided into the three main types *Variational Inference*, *Sampling Methods* and *Laplace Approximation*. While we already gave an overview about the latter in the previous section, we will now introduce some related works for the former two.

[Hinton and Van Camp, 1993] pioneered in variational inference methods for BNNs who approximated the posterior distribution of a small neural network with four units and one hidden layer by a multivariate Gaussian having a diagonal covariance matrix. [Barber and Bishop, 1998] extended the idea to full covariance matrices and demonstrated how the *evidence lower bound* can be optimised for neural networks.

[Graves, 2011] proposed a stochastic method for variational inference as the first scalable approach using Gaussian priors. The method is stochastic as the evidence lower bound is optimised over mini-batches of data. [Blundell et al., 2015] extended this stochastic approach to non-Gaussian priors with their *Bayes by Backprop* algorithm. The idea is to learn the shape of variational distributions over the parameters during the training of the network.

The works of [Gal and Ghahramani, 2016, Gal and Ghahramani, 2015] can be seen as at the border of variational inference and sampling methods as they cast existing stochastic elements in neural networks as variational inference. Their *Monte Carlo Dropout* method formulates the dropout [Srivastava et al., 2014] layers, used in

most modern neural networks for regularisation, as Bernoulli distributed random variables. Training a neural network with dropout layers means that in each iteration a randomly selected subset of weights is set to zero, which can be can be linked to performing variational inference [Gal and Ghahramani, 2016]. The predictive uncertainty can be obtained during test time by keeping dropout active. As many modern neural networks are already trained using dropout, this method is highly attractive for practitioners to obtain uncertainty estimates without the need for extensive expert knowledge.

In the area of Monte Carlo methods for BNNs, the pioneering work was published by [Neal, 1992] who proposed the *Hybrid Monte Carlo*[1] (HMC) method which corresponds to an instance of the *Metropolis-Hastings* algorithm and allows to approximate the true posterior distribution arbitrarily close [Humt, 2019]. HMC does not scale well to large datasets as it requires processing the whole dataset in each iteration.

3.3 Uncertainty for Multi-Label Classification

The evaluation of uncertainty estimation methods like Bayesian neural networks for multi-label classification problems is a rather new research field. The works that have been published so far approach this topic mainly from an application point.

[Chen et al., 2020] use Monte Carlo Dropout to model epistemic uncertainty in their proposed *Uncertainty Quantification for Multilabel Text Classification (UC-MLTC)* framework.

[Ghoshal et al., 2019] apply *Monte Carlo Dropweights* (or *Monte Carlo drop connect*), a variant of Monte Carlo dropout where the weights on the incoming activations are dropped instead the outgoing activation for the following neurons, for classifying cell types in immunohistochemically stained images. According to the authors, this is the first work on quantifying uncertainty in multi-label image classification.

[1] Also known as *Hamiltonian Monte Carlo*

4 Kronecker Factored Laplace Approximation

In this section we will derive the Kronecker-factored Laplace approximation, a method for approximating the intractable posterior distribution over the parameters of neural networks. After presenting the general theory behind the Laplace approximation in Section 4.1, we will have a detailed look on how to approximate the curvature of neural networks and the compatibility with the binary cross entropy loss that is used for training the multi-label classifier in Section 4.2. In Section 4.3 we describe the Kronecker factorisation of the curvature factors that allows to scale that method to the size of modern DNNs and discuss the proposed regularisation scheme in Section 4.4.

4.1 Laplace Approximation

The goal of Laplace's method is to approximate the intractable posterior distribution with a multivariate gaussian around a known mode of that distribution. The Laplace approximation of the posterior $p(\theta|\mathcal{D})$ of a neural network can be obtained by taking the second order Taylor expansion of the log-posterior over the weights around the MAP estimate θ^*, which we get from training the network on data $\mathcal{D}$, e.g. by using gradient based techniques [Ritter et al., 2018, Gawlikowski et al., 2021]:

$$\begin{aligned}\log\, p(\theta|\mathcal{D}) &\approx \log p(\theta^*|\mathcal{D}) + (\theta-\theta^*)^\intercal \nabla \log p(\theta^*|\mathcal{D}) + \frac{1}{2}(\theta-\theta^*)^\intercal H_{\log p(\theta|\mathcal{D})}(\theta^*)(\theta-\theta^*) \\ &= \log p(\theta^*|\mathcal{D}) + \frac{1}{2}(\theta-\theta^*)^\intercal H_{\log p(\theta|\mathcal{D})}(\theta^*)(\theta-\theta^*)\ .\end{aligned} \tag{4.1}$$

As the Taylor series is evaluated at a maximum θ^* where the gradient is zero, the first order term can be omitted. $H_{\log p(\theta|\mathcal{D})}(\theta^*)$ is the Hessian of the log posterior which describes the local curvature around θ^*. As θ^* is a maximum, $H_{\log p(\theta|\mathcal{D})}(\theta^*)$ is nega-

tive definite. If we use the Hessian of the negative log posterior $H = -H_{\log p(\theta|\mathcal{D})}(\theta^*)$ instead - which is in turn positive definite - we get

$$\log p(\theta|\mathcal{D}) \approx \log p(\theta^*|\mathcal{D}) - \frac{1}{2}(\theta - \theta^*)^\intercal H(\theta - \theta^*) \ . \tag{4.2}$$

From now on, H will refer to the Hessian of the negative log posterior $= -H_{\log p(\theta|\mathcal{D})}(\theta^*)$ if not stated otherwise. Taking the exponential on both sides of (4.2) leads to

$$p(\theta|\mathcal{D}) \approx p(\theta^*|\mathcal{D}) \ \exp\left\{-\frac{1}{2}(\theta - \theta^*)^\intercal H(\theta - \theta^*)\right\} \ . \tag{4.3}$$

The right hand side of (4.3) is the unnormlised probability density function of a gaussian distribution and can be normalised with a normalisation coefficient Z [Bishop, 2006, pp. 215] [Humt, 2019]

$$\frac{1}{Z}p(\theta|\mathcal{D}) \approx \frac{1}{Z}p(\theta^*|\mathcal{D}) \ \exp\left\{-\frac{1}{2}(\theta - \theta^*)^\intercal H(\theta - \theta^*)\right\} \ , \tag{4.4}$$

which can also be written as

$$\frac{1}{Z}p(\theta|\mathcal{D}) \approx \mathcal{N}(\theta^*, H^{-1}) \ . \tag{4.5}$$

The normalisation coefficient Z should be choosen accordingly to the normalisation coefficient $\frac{\sqrt{det(H)}}{(2\pi)^{\frac{N_\theta}{2}}}$ of a multivariate normal distribution with covariance H^{-1}:

$$\begin{aligned} \frac{1}{Z}p(\theta^*|\mathcal{D}) &\overset{!}{=} \frac{\sqrt{det(H)}}{(2\pi)^{\frac{N_\theta}{2}}} \\ \Leftrightarrow \frac{1}{Z} &= \frac{\sqrt{det(H)}}{(2\pi)^{\frac{N_\theta}{2}}} \frac{1}{p(\theta^*|\mathcal{D})} \\ \Leftrightarrow Z &= p(\theta^*|\mathcal{D}) \frac{(2\pi)^{\frac{N_\theta}{2}}}{\sqrt{det(H)}} \ . \end{aligned} \tag{4.6}$$

However, Z does not need to be evaluated in practice as we are aiming for sampling from that distribution rather then computing densities. Furthermore, following from the central limit theorem, the posterior distribution is asymptotically normal distributed, making the approximation of the posterior with a Gaussian becoming increasingly better as more data points are observed [Bishop, 2006, pp. 215, 216] [Humt, 2019].

Wrapping this up, after training the neural network on a sufficient amount of data, we can assume the network's parameters to be normal distributed

$$\theta \sim \mathcal{N}(\theta^*, H^{-1}) \ , \tag{4.7}$$

with the MAP estimate θ^* as mean and the inverse of the Hessian of the negative log posterior as covariance.

To approximate the posterior mean on unseen data $x^* \in X$, we use Monte Carlo integration to approximate the intractable integral in (4.8). We draw T[1] weight configurations from the posterior distribution and average over T predictions, using a different set of weights $\theta^{(t)}$ on each forward pass [Ritter et al., 2018, Gawlikowski et al., 2021]:

$$p(y^*|x^*, \mathcal{D}) = \int p(y^*|x^*, \theta)p(\theta|\mathcal{D})d\theta \approx \frac{1}{T}\sum_{t=1}^{T} p(y^*|x^*, \theta^{(t)}) \ . \tag{4.8}$$

4.2 Approximating the Hessian

Working with the Hessian comes with three problems:

1. The size of the Hessian is quadratic in the number of network parameters ($N_\theta \times N_\theta$) and therefore way too large for probably every real-life-scale neural network.[2]

2. Computing the Hessian involves computing second derivatives which is computationally expensive

3. The covariance of the posterior distribution is approximated by the inverse of the Hessian of the negative log posterior, though the posterior is the unknown distribution we want to approximate. How can we actually compute the Hessian of a function we don't know?

Given those three problems, we need to come up with an alternative to the Hessian of the log posterior that has reasonable memory requirements and that we actually know how to compute in an efficient way. Let's start with the third problem.

[1] We will use $T = 50$ Monte Carlo samples throughout this work.

[2] The storage requirement for the Hessian of a neural network with 10 million parameters would be more then 2.9 peta byte when using a 32 bit floating point representation.

4.2.1 Decomposing the Log Posterior Hessian

The covariance of the posterior distribution is given as the inverse of the Hessian of the negative log posterior (see (4.7)). By applying the logarithm to Bayes' Theorem and taking the second derivative, we can express the Hessian of the negative log posterior $-H_{\log p(\theta|\mathcal{D})}$ with the Hessian of the negative log likelihood $-H_{logp(\mathcal{D}|\theta)}$ minus the Hessian of the prior $H_{\log p(\theta)}$ which is $-\tau I$ if we assume a Gaussian prior with precision τ:

$$-H_{\log p(\theta|\mathcal{D})} = -H_{\log p(\mathcal{D}|\theta)} + \tau I \ . \tag{4.9}$$

The full derivation can be found in Appendix A.1. As we will see at the end of this section, the negative log likelihood is actually the loss function (or objective function) we are minimising when training the neural network, making the Hessian of the negative log likelihood identical with the Hessian of the loss w.r.t. the parameters θ. While we might know how to compute the Hessian of the negative log likelihood, it still leaves us with first two problems: its size and the cost of computing second derivatives.

Fortunately we can approximate the Hessian with matrices that are both, smaller regarding their storage requirements and easier to compute. Under certain conditions, the GGN and the FIM can be used as approximations to the Hessian of the loss function.

4.2.2 The Generalised Gauss Newton Matrix

The GGN is often used as a positive semi definite (p.s.d.) approximation of the Hessian of the loss function [Martens and Grosse, 2015] in second order optimisation and can be defined as

$$GGN = \frac{1}{|\mathcal{D}|} \sum_{(x,y)\in D} J_f^\intercal H_{\mathcal{L}} J_f \ , \tag{4.10}$$

with J_f denoting the Jacobian of the networks outputs $z = f_\theta(x)$ w.r.t. the parameters θ and $H_{\mathcal{L}}$ is the Hessian of a convex loss function $\mathcal{L}(y, z)$ w.r.t. the network outputs $z = f_\theta(x)$ (which should not be confused with the Hessian of the negative log likelihood that we want to approximate) [Martens, 2020,Botev et al., 2017]. The GGN needs less storage as it is enough to store the Jacobian J_f, which has size $N_\theta \times C$, where C is the number of outputs of the neural network, and the Hessian of the loss $H_{\mathcal{L}}$, which is an $C \times C$ matrix. Looking at the computational effort, we still need to compute seconds derivatives for $H_{\mathcal{L}}$, but much less then for the Hessian of

the negative log likelihood. [Botev et al., 2017] showed, that the layer-wise diagonal blocks of the Hessian and GGN are equivalent for neural networks with piece-wise linear activation functions (e.g. ReLU).

4.2.3 The Fisher Information Matrix

The FIM of the model's distribution $P_{\mathcal{D}}(\theta)$ w.r.t. the parameters θ is given by the expectation over the outer vector product of the gradient of the log likelihood w.r.t. the parameters θ

$$FIM = \mathbb{E}\Big[\nabla_\theta \log p(\mathcal{D}|\theta) \nabla_\theta \log p(\mathcal{D}|\theta)^\intercal\Big] \ . \tag{4.11}$$

The FIM is modest in its storage requirements as it is enough to store this gradient which's size is the number of parameters N_θ and it is easy to compute, as this is the gradient we use for backpropagation when training the neural network. [Martens, 2020] proved the equivalence of the GGN and the FIM, given that the network's predictive distribution is an exponential family model, parametrised by the network parameters θ.

4.2.4 Exponential Family Loss

To be able to use the FIM as an approximation, we need to verify, that the predictive distribution of our neural network is an exponential family model, i.e. that the loss function can be interpreted as an exponential family negative log likelihood [Ritter et al., 2018]. In the following section we will have a look at the loss function we use for training neural networks on Multi-Label Classification tasks.

In the multi-label classification setting, we obtain the probabilities for each class by applying the sigmoid function to the output of the neural network (2.14). The network is trained by using the binary cross entropy loss (2.15). Additionally to the logits being conditionally independent given the data and parameters [Monteiro et al., 2020], the class-wise predictions are just depending on their respective logit. Under those conditions we can write the joint predictive probability as a product of class-wise probabilities

$$p(y|x,\theta) = \prod_{c=1}^{C} p(y_c|z_c) = \prod_{c=1}^{C} \sigma(z_c)^{y_c}(1-\sigma(z_c))^{1-y_c} \ , \text{ with } z_c = f_\theta(x)_c \ . \tag{4.12}$$

As we want to look at the negative log likelihood, we take the logarithm on both sides and multiply with -1. Secondly, by applying the product and power rule for the logarithm, we arrive at the expression for the binary cross entropy loss as in (2.15):

$$\begin{aligned} -\log p(y|x,\theta) &= -\log \prod_{c=1}^{C} \sigma(z_c)^{y_c}(1-\sigma(z_c))^{1-y_c} \\ &= \sum_{c=1}^{C} -y_c \log \sigma(z_c) - (1-y_c)\log(1-\sigma(z_c)) \end{aligned} \tag{4.13}$$

with $z_c = f_\theta(x)_c$. Hence, the condition for the equivalence of the GGN and FIM are met and we can use the FIM as an approximation of the Hessian.

The approximation of the Hessian with the FIM is guaranteed to be p.s.d. as the FIM is the expectation over an outer product of the gradient[3]. Though, to use it as the covariance of the posterior distribution we would need it to be positive definite as the the covariance is the inverse of the Hessian. As we are using the FIM as an approximation to the Hessian of the log likelihood, it is enough to have it p.s.d. as $(H + \tau I)$ becomes positive definite[4] for every $\tau > 0$.

4.3 Kronecker Factorisation

As we have seen in Section 4.2, we can approximate the Hessian of the negative log posterior by a block-diagonal matrix, where each block corresponds to the parameters of the respective layer of the neural network and is equivalent to the same block of the FIM. By employing this approximation, we already assumed the layer-wise independence of the networks parameters. This assumption is acceptable, as the block-diagonal approximation preserves enough information about the neural network's curvature [Ritter et al., 2018] by considering the covariances within layers. To further improve computational efficiency we exploit that - for a single data

[3] Let $a \in \mathbb{R}^n \implies x^\intercal a a^\intercal x = (x^\intercal a)^2 \geq 0, \forall x \neq 0 \implies A = aa^\intercal is$ p.s.d. $\square$

[4] Let $A \in \mathbb{R}^{n\times n}$ be p.s.d. and $D \in \mathbb{R}^{n\times n}$ a positive diagonal matrix. $\implies \forall v \neq 0 : v^\intercal(A+D)v = v^\intercal A v + v^\intercal D v > 0 \implies (A+D)$ is positive definite. $\square$

point - we can express each diagonal block λ of the curvature matrix as a Kronecker Product [Botev et al., 2017, Martens and Grosse, 2015, Ritter et al., 2018][5]:

$$H_\lambda = \frac{\partial^2 \mathcal{L}}{\partial \text{vec}(W_\lambda) \partial \text{vec}(W_\lambda)} = \mathcal{Q}_\lambda \otimes \mathcal{H}_\lambda. \tag{4.14}$$

In (4.14), H_λ denotes the Hessian w.r.t. the parameters in layer λ, $\mathcal{L}$ the loss function, $\mathcal{Q}_\lambda = a_{\lambda-1} a_{\lambda-1}^\intercal$ the covariance of the incoming activations from the previous layer $\lambda - 1$ and $\mathcal{H}_\lambda = \frac{\partial^2}{\partial h_\lambda \partial h_\lambda}$ the Hessian of the loss function w.r.t. the linear pre-activations in layer λ [Ritter et al., 2018]. The Kronecker Product $A \otimes B$ of two matrices A and B is defined as $\{A \otimes B\}_{ij} = a_{ij} B$ and has the property, that the inverse of a Kronecker Product equals the Kronecker Product of its inverted factors[6]. This is fortunate, as we can invert two relatively small matrices for each layer. So far we gained from the independence assumption, that the per-sample Hessian of our neural network decomposes into $2L$ much smaller matrices where L is the number of layers in the neural network [Martens and Grosse, 2015].

As a second approximation, we'll assume independence of $\mathcal{Q}_\lambda$ and $\mathcal{H}_\lambda$ for all layers λ. By doing this, we keep the Kronecker factorisation in expectation and can express the expected Hessian for each layer λ as [Ritter et al., 2018]:

$$\mathbb{E}[H_\lambda] = \mathbb{E}[\mathcal{Q}_\lambda \otimes \mathcal{H}_\lambda] \approx \mathbb{E}[\mathcal{Q}_\lambda] \otimes \mathbb{E}[\mathcal{H}_\lambda] \ . \tag{4.15}$$

Even though this second approximation is likely to be major, it is considered to be "fairly accurate in practice" according to [Martens and Grosse, 2015].

Instead of sampling the parameters form a multivariate normal distribution having the inverted curvature matrix as its covariance (like we saw in Section 4.1), we now need to migrate to the matrix variate normal distribution (MND) [Gupta and Nagar, 2010] to draw parameters using the Kronecker factors. The MND is defined over a matrix $X : p \times n$ of random variables instead of just a vector. It is parametrised by a mean matrix $M : n \times p$ and covariance matrix $U \otimes V$. $U : n \times n$ and $V : p \times p$ are both p.s.d. and relate to the covariances of the rows and columns in X [Gupta

[5] Here we assume a uniform prior to have the Hessian of the log posterior equal the Hessian of the log likelihood and allow for a more compact way of presentation. In Section 4.2.1, we already assumed a Gaussian prior and will incorporate this in the following section.

[6] $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1} \Leftrightarrow (A \otimes B)(A^{-1} \otimes B^{-1}) = I : (A \otimes B)(A^{-1} \otimes B^{-1}) = (AA^{-1}) \otimes (BB^{-1}) = I \otimes I = I \ \square$

and Nagar, 2010, Ritter et al., 2018]. Hence, our resulting posterior for parameters in layer λ becomes

$$\theta \sim \mathcal{MN}(\theta_\lambda^*, \mathcal{Q}_\lambda^{-1}, \mathcal{H}_\lambda^{-1}). \tag{4.16}$$

While drawing a vectorized sample from $\mathcal{MN}(\theta_\lambda^*, \mathcal{Q}_\lambda^{-1}, \mathcal{H}_\lambda^{-1})$ corresponds to drawing a sample from the normal distribution $\mathcal{N}(vec(\theta_\lambda^*), \mathcal{Q}_\lambda^{-1} \otimes \mathcal{H}_\lambda^{-1})$, this sampling can be done in a more efficient way. For $QQ^\intercal = \mathcal{Q}_\lambda^{-1} \in \mathbb{R}^{q \times q}$ and $HH^\intercal = \mathcal{H}_\lambda^{-1} \in \mathbb{R}^{h \times h}$ being the Cholesky decompositions of the covariance matrices for layer λ and θ_λ^* the MAP estimate for the parameters θ in the same layer, reshaped into shape $q \times h$, we can obtain a new set of parameters from

$$\theta_\lambda = \theta_\lambda^* + QSH^\intercal \ , \tag{4.17}$$

with S being qh samples drawn from a standard normal distribution and reshaped into a $q \times h$ matrix [Humt, 2019, Ritter et al., 2018]. Thereby we spare the evaluation of the Kronecker product as well as a matrix vector product with a large matrix in favour of two matrix products of small matrices [Ritter et al., 2018].

4.4 Regularisation

The need for regularising the curvature factors comes from three reasons. First, while the FIM is always p.s.d. and symmetric in theory, that might not be the case in practice due to numerical issues [Humt, 2019]. Second, the Laplace approximation might place probability mass in low probability regions of the true posterior and third, the approximations made for making the Laplace approximation tractable (layer- and factor independence) might cause an overestimation of the variance in certain directions [Ritter et al., 2018]

As we have seen in Section 4.2.1, the Hessian of the negative log posterior decomposes into the Hessian of the negative log likelihood minus the Hessian of the prior. For a Gaussian prior, corresponding to the L_2 regularisation, the Hessian is $-\tau I$ where τ is the precision of the prior and I is the identity matrix. To incorporate that prior into (4.15) we add a multiple of the identity to each Kronecker factor which results in following approximation of the expected Hessian for layer λ [Ritter et al., 2018]:

$$\mathbb{E}[H_\lambda] = N\mathbb{E}\left[-\frac{\partial^2 \log p(\mathcal{D}|\theta)}{\partial \theta_\lambda^2}\right] + \tau I \approx (\sqrt{N}\mathbb{E}[\mathcal{Q}_\lambda] + \sqrt{\tau} I) \otimes (\sqrt{N}\mathbb{E}[\mathcal{H}_\lambda] + \sqrt{\tau} I). \tag{4.18}$$

Despite τ referring to the precision of the Gaussian prior on the parameters and N to the size of the dataset, they can also be treated as hyperparameters and optimized w.r.t. a performance measure. Choosing values for N that exceed the actual dataset size can be seen as including duplicates of data points as pseudo-observations [Ritter et al., 2018].

5 Experiments

In this chapter we present the experiments that we conducted to evaluate the method derived in Chapter 4, as well as our results. We examined the calibration and OOD detection performance in a multi-label setting of the presented method on (a) a synthetic multi-label dataset we created and (b) a large multi-label remote sensing dataset called MLRSNet. Due to the motivation of Bayesian methods, one would expect a better calibration and an improved OOD detection. For the cases of multi-class classification and regression [Humt, 2019] and [Ritter et al., 2018] already showed that this holds, but the multi-label setting adds multiple significant differences:

1. The multi-label classifier returns conditional independent probabilities for each of the C classes by passing the logits individually through the sigmoid function.
2. We can not differentiate clearly between a correct and an incorrect prediction anymore.
3. The training optimises a binary cross-entropy loss function instead of mean squared error or categorical cross-entropy, which we already discussed in Section 4.2.4.

Before we describe the experimental setup and results, we will shortly introduce our implementation in Section 5.1 and define the evaluation measures to be used in Section 5.2.

5.1 Implementation

We implemented the Kronecker-factored Laplace approximation method in Python for the TensorFlow [Abadi et al., 2015] framework, version 2.x[1]. We choose TensorFlow over PyTorch [Paszke et al., 2019] mainly for two reasons: Firstly, because [Qi et al., 2020] are providing pre-trained TensorFlow models for the Multi-Label Earth

[1] We used Python v3.6 and TensorFlow v2.4.1

Observation dataset we are using in our experiments. Secondly, [Lee et al., 2020] have already implemented a PyTorch compatible version[2]. [Martens and Grosse, 2015] published an implementation of their "K-FAC" method[3], which could have been used to calculate the curvature matrix, though it does only work with TensorFlow version 1.x.

The `tf-laplace` package consists mainly of two modules: `Curvature` and `Sampler`. `Curvature` provides an interface for computing and inverting the curvature matrix of a neural network, given as a sequential TensorFlow model. Our implementation contains three different approximations to the real curvature matrix, namely the diagonal of the real FIM (`DiagFisher`), a block diagonal approximation that contains the layer-wise diagonal blocks of the real FIM (`BlockDiagFisher`) and finally the Kronecker-factored approximation described in chapter 4 (`KFAC`). So far, the curvature is only calculated for dense and convolutional layers, as those are the only layer types, relevant for the models used in this work (see Sections 5.3.1.2 and 5.4.1.2).

The `Sampler` module provides an interface for sampling weights from the estimated posterior distribution and setting them as weights of the provided model. As the way that weights are sampled depends on the curvature approximation, each of the three curvature implementations has a corresponding sampler implementation.

As TensorFlow does not provide a mechanism to obtain the pre-activation of neurons, which we need to calculate the factors Q_λ of the Kronecker-factored curvature matrix (see Section 4.3), we needed to monkey-patch TensorFlow's `Layer` class. This patch can be found in the `hooks` module.

Our code is available on GitHub at: `https://github.com/ferewi/tf-laplace`.

5.2 Evaluation Measures

In this section we will introduce the performance measures used to quantify the quality of the obtained uncertainty estimates in the field of multi-label classification.

5.2.1 Multi-Label Measures

Unlike in binary or multi-class classification where a prediction can either be correct or false, we need to consider partially correct predictions in the multi-label setting.

[2] `https://github.com/DLR-RM/curvature`
[3] `https://github.com/tensorflow/kfac`

For dealing with partially correct predictions we distinguish between different forms of averaging.

For **instance-based** measures, a score (e.g. Accuracy, Precision, Recall or F1) is computed for each example. The final score is obtained by aggregating over all instances.

In the case of **class-based** evaluation measures, the C binary predictions $\hat{y}_c$ for each class $c \in \{1, \ldots, C\}$ are evaluated using a binary measure. The class-wise results are then averaged over all labels, where either micro- or macro averaging can be used to compute the global means [Yang, 1999]. Let $B(TP_c, FP_c, TN_c, FN_c)$ be a binary evaluation measure with TP_c being the number of true positives for class c, FP_c the number of false positives and TN_c and FN_c the numbers of true negatives and false negatives likewise. The macro-averaged score is obtained by computing the per-class scores B_c and then averaging over all classes [Yang, 1999, Tomás et al., 2014]

$$B_{macro} = \frac{1}{C} \sum_{c=1}^{C} B(TP_c, FP_c, TN_c, FN_c), \tag{5.1}$$

For the micro-averaged score, the true and false positives and Negatives for each class are summed up and the binary measure is computed based on these aggregated values [Yang, 1999, Tomás et al., 2014]:

$$B_{micro} = B(\sum_{c=1}^{C} TP_c, \sum_{c=1}^{C} TN_c, \sum_{c=1}^{C} FP_c, \sum_{c=1}^{C} FN_c). \tag{5.2}$$

While micro-averaging assigns equal weight to each class, the macro-averaging method weights classes according to their relative frequency [Dendamrongvit et al., 2011] which can be favourable when dealing with imbalanced datasets [Tomás et al., 2014].

5.2.2 Precision

The precision of a binary classifier M_b is the fraction of instances that have been correctly given the positive class label among all predicted instances:

$$\text{Prec}(M_b) = \frac{\text{TP}}{\text{TP} + \text{FP}} \ . \tag{5.3}$$

The macro- and micro-averaged versions are defined according to (5.1) and (5.2).

The instance-based precision of a multi-label classifier M_{ml} is defined as:

$$\text{Prec}(M_{ml}, \mathcal{D}^*) = \frac{1}{N} \sum_{i=1}^{N} \frac{|y_i \cap \hat{y}_i|}{|\hat{y}_i|} \ , \tag{5.4}$$

with $\mathcal{D}^*$ being the evaluated dataset of size N. $|y_i \cap \hat{y}_i|$ and $\hat{y}_i$ denote the number of correctly predicted and all predicted classes.

5.2.3 Confidence

The confidence of a binary classifier M_b in its prediction is given by the difference of the predicted value to the threshold[4] that is used mapping a returned class probability to the positive or negative result. Given a threshold t and the predicted class probability $\hat{p}_i$, the confidence, normalised to the interval $[0, 1]$, is defined as:

$$\text{Conf}(\hat{p}_i, t) = \frac{|\hat{p}_i - t|}{t} \ . \tag{5.5}$$

5.2.4 Expected Calibration Error (ECE)

The expected calibration error (ECE) is a measure for the reliability of a classifier. A binary classifier M_b is said to be reliable or well-calibrated if its mean predicted value matches its relative frequency of positive class predictions [DeGroot and Fienberg, 1983] - also known as precision - on a test set. The ECE of a binary classifier M_b on a test set $\mathcal{D}^*$ is computed by discretising the predictions of the classifier into a fixed number of K bins[5] and evaluating the average error between each bin's mean predicted value and precision, weighted by the bin size [Niculescu-Mizil and Caruana, 2005, Naeini et al., 2015, Guo et al., 2017]

$$\text{ECE}(M_b, \mathcal{D}^*) = \sum_{k=1}^{K} \frac{|B_k|}{N} \Big| Prec(M_b, \mathcal{D}^*)_{B_k} - \frac{1}{|B_k|} \sum_{i=1}^{|B_k|} \mathbf{1}_{B_k}(\hat{p}^{(i)}) \hat{p}^{(i)} \Big| \ , \tag{5.6}$$

with $Prec(M_b, \mathcal{D}^*)_{B_k}$ being the precision within bin k and $\mathbf{1}_{B_k}(\hat{p}^{(i)})$ the indicator function for the prediction $\hat{p}$ of example x_i falling into that bin.

[4] We used a threshold of 0.5 throughout this work.
[5] We use $K = 10$ bins throughout our experiments.

This approach can be easily adapted to the multi-label classification context by using an averaged precision value as described in Section 5.2.2. Thus, the macro-averaged ECE of a multi-label classifier M_{ml} is given by

$$\mathrm{ECE}_{macro}(M_{ml}, \mathcal{D}^*) = \sum_{k=1}^{K} \frac{|B_k|}{N} \left| Prec_{macro}(M_{ml}, \mathcal{D}^*)_{B_k} - \frac{1}{C|B_k|} \sum_{c=1}^{C} \sum_{i=1}^{|B_k|} \mathbf{1}_{B_k}(\hat{p}_c^{(i)}) \hat{p}_c^{(i)} \right| . \tag{5.7}$$

The micro-averaged and instance-based versions are calculated likewise. We will use the macro-averaged ECE throughout our experiments to account for the imbalance of the earth observation dataset which we use in the experiments described in Section 5.4.

The (mis-)calibration of a classifier can be visualised by reliability diagrams [Guo et al., 2017, Niculescu-Mizil and Caruana, 2005, Naeini et al., 2015]. For a binary classifier, the mean predicted value is plotted against the relative frequency of positive class predictions within each bin [Niculescu-Mizil and Caruana, 2005]. A well calibrated model would have those points near the diagonal, an overconfident model below and an underconfident model above it. Again, we can adapt this to the multi-label case by plotting the mean precision against the mean predicted value within each bin.

5.2.5 Area Under ROC-Curve (AUROC)

The area under the receiver operating characteristic curve (AUROC) is a measure for the ability of a binary classifier to separate two classes and can be seen as a single-number summary of a receiver operating characteristic (ROC) plot [Bradley, 1997]. In a ROC plot, the true positive rate (TPR) is plotted against the false positive rate (FPR) at increasing threshold levels for thresholding the output of the classifier. The higher the AUROC, the better the classifier can separate both classes from another. A perfect classifier would have an AUROC score of 1 and ranks all examples of the positive class higher then all examples of the negative class.

We use the AUROC score to describe, how well a model is able to separate in-distribution from out-of distribution examples based on either the predicted values or the predictive standard deviation of the BNN being thresholded.

5.3 Synthetic Multi-Label Example

Before we apply the method described in Chapter 4 to a real-world earth observation dataset, we will asses it on a synthetic multi-label dataset we created and called F^3. We chose the creation of a synthetic dataset over using multi-label extensions of popular classification benchmark datasets (like multi-label MNIST) to have more control over the dataset's properties and allow for a better visualisation. We based our dataset on the toy regression example from [Hernández-Lobato and Adams, 2015] and [Ritter et al., 2018] and rebuilt it as a multi-label classification problem. After describing the experimental setup in Section 5.3.1, we will evaluate the uncertainties obtained from the Kronecker-factored Laplace approximation by visualising the predictive standard deviation on a test set (5.3.2). In Section 5.4.2 we will analyse the potential of improving the model's calibration. Finally, we will use the uncertainty estimates to detect out-of-distribution examples in Section 5.3.4.

5.3.1 Setup

In the following Section we describe our experimental setup. We start by introducing the multi-label dataset F^3. Afterwards, the neural network that we used is described in Section 5.3.1.2 before we present our approach for finding good values for the hyperparameters τ and N in Section 5.3.1.3.

5.3.1.1 Dataset

The training set F^3_{train} contains 50 points, having their x_1 coordinates sampled uniformly from the interval $[-5.5, 5.5]$ and their x_2 coordinate sampled from normal distributions $\mathcal{N}(x_1^3, 9)$. Figure 5.1 shows an example of a sampled training set. Each point belongs to at least one of two classes, red and blue, with the red class containing points having $x_2 > x_1^3 - 10$ while the blue class contains points with $x_2 < x_1^3 + 10$. Points in the overlapping region around $x_2 = f(x_1) = x_1^3$ with $x_1^3 - 10 \leq x_2 \leq x_1^3 + 10$ belong to both classes, indicated by their multi-label $(1, 1)$. In addition to that 50-points dataset we used for training the neural network, we created a validation set F^3_{val} for optimising the hyperparameters, a test set F^3_{test} to use for the uncertainty visualisation and calibration experiment and an out-of-distribution set F^3_{ood} that we used in the out-of-distribution detection experiment. For the validation and test set, we samples 1000 and 2000 points respectively uniformly from $[-5.5, 5.5] \times [-5.5^3, 5.5^3]$. For the out-of-distribution set, we

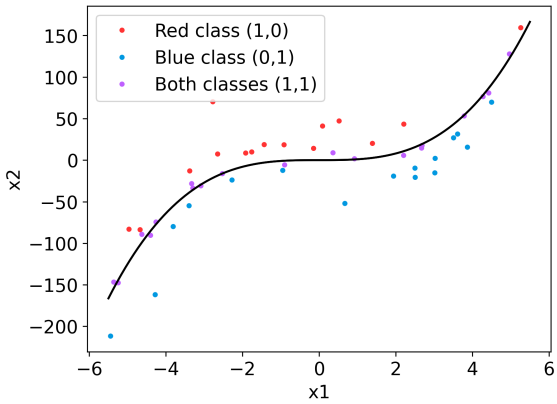


Figure 5.1: The training set F^3_{train} consists of two classes, along the graph from x^3 in the interval $[-5.5, 5.5]$, with 50 data points in total.

sampled 1000 points $(x_1, x_2) \in [-10, 5.5] \times [-400, -167]$ and another 1000 points $(x_1, x_2) \in [-10, 5.5] \times [167, 400]$.

In addition to the F^3 dataset, we also created another 4 datasets, based on the idea of the "Mldatagen" framework presented by [Tomás et al., 2014], though the F^3 dataset seemed more suitable for the experiments we intended to run. A more detailed description of those hypersphere based datasets can be found in Appendix A.2.

5.3.1.2 Network

For the dataset described in Section 5.3.1.1 we trained a MLP with two hidden layers and 10 neurons in each layer as our baseline model. We used ReLU activations and L2 regularisation in the hidden layers. On the output neurons we applied Sigmoid activations to transform the logits into probabilities. We trained the network for 1000 epochs, using binary cross-entropy loss and Adam [Kingma and Ba, 2014] as optimiser. After training the neural network we approximated its curvature using our `KFAC` implementation that we described in Section 5.1.

5.3.1.3 Hyperparameter Tuning

The two hyperparameters τ and N have been optimised w.r.t. the loss function, i.e. the negative log likelihood. As we had limited initial knowledge about the size of those parameters, we used a grid search over log-space with $log_{10}(\tau) \in [-10, 10]$ and $log_{10}(N) \in [-10, 10]$. Figure 5.2 shows the result for the F^3 dataset. The best

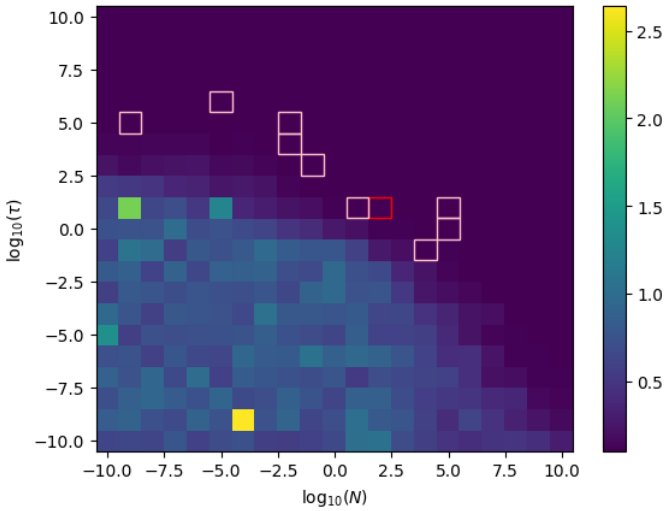
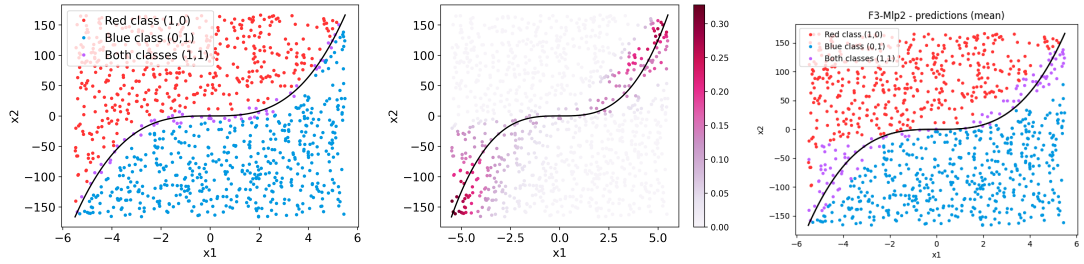


Figure 5.2: Result of the grid search for the two hyperparameters τ and N on the F^3 dataset, showing the NLL as a function of the two parameters. The 10 best parameter pairs a marked with a colored border. The parameter pair for the lowest NLL value ($\tau = 10, N = 100$) is shown with a dark red border.

value pair $\tau = 10$, $N = 100$ is marked with a red border. The next 9 best pairs are marked with a light-red border. The color of the squares encodes the value of the NLL, ranging from dark violet color for low values to to high values shown in yellow.

As we can see, the choice of τ and N is crucial for obtaining a low NLL. While low values for τ and N lead to large NLLs, we could increase one and decrease the other parameter without loosing performance in a certain range. It should be noted that while choosing a value for N larger than the dataset size can be interpreted as including pseudo-observations (see Section 4.4), we are lacking this kind of interpretation for smaller values. As values $N < 1$ scale down the curvature factors and therefore increase the covariance in each layer, the uncertainty regarding the correct value of the model parameters might be overestimated. Choosing very high values for N on the other hand leads to very small covariances and thus very low variability in the sampled weights. The parameter pair that yields the lowest NLL is $\tau = 10$ and $N = 100$ with N being twice as large as the training set.



(a) (b) (c)

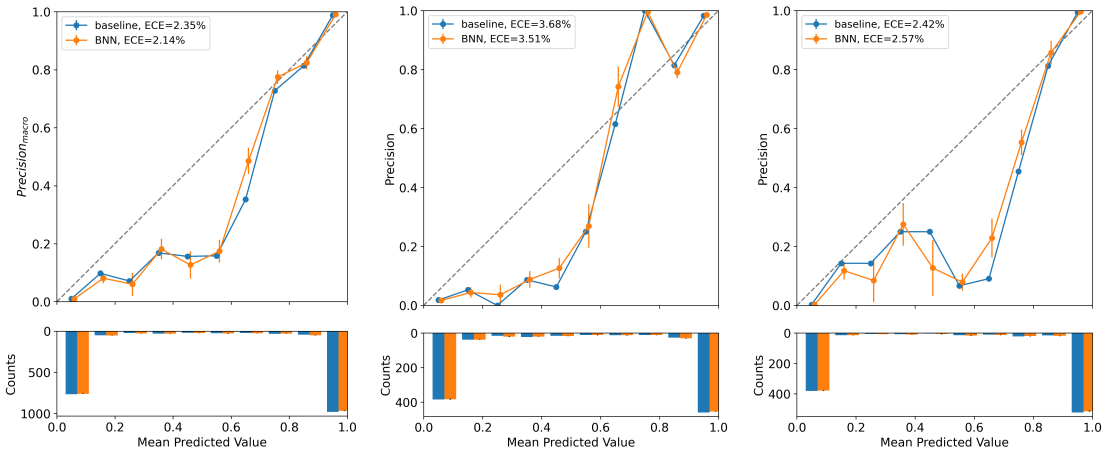
Figure 5.3: Model uncertainty on the test set as captured by the Kronecker-factored Laplace approximation. The plots show: the ground truth labels (5.3a), the accumulated predictive standard deviation over both classes (5.3b) and the mean predictions (5.3c) of 50 probabilistic forward passes with hyperparameters $\tau = 100$ and $N = 10$.

5.3.2 Uncertainty Visualisation

To visualise the model uncertainty, we ran 50 forward passes of the test set through the Bayesian version of the neural network, each time with a new set of weights sampled from the approximated posterior distribution. The results are shown in Figure 5.3b. Each point of the test set in Figure 5.3b encodes the accumulated standard deviation of the predictions over both classes by its color. We see a higher model uncertainty along the area, where both classes overlap, which gets higher and more spread out near the borders of the training interval, especially where the model fails to classify correctly. This matches our expectation about model uncertainty as this area is only covered with very few training points (see Figure 5.1). On the other hand, the model is very certain in its prediction for the points in the upper left and lower right corner as the low standard deviation of the predictions indicates. These results also match those produced by [Ritter et al., 2018] for their toy regression example, which also shows the model uncertainty as predictive standard deviation spreading out fan-shaped on the borders of the training interval.

5.3.3 Calibration

To asses the calibration of the multi-label classifier we compare the calibration error of the deterministic baseline model with that of 50 forward passes through the BNN, with a new set of weights sampled from the approximated posterior distribution on each forward pass. A positive effect on the calibration of multi-class classifications was previously shown by [Humt, 2019].



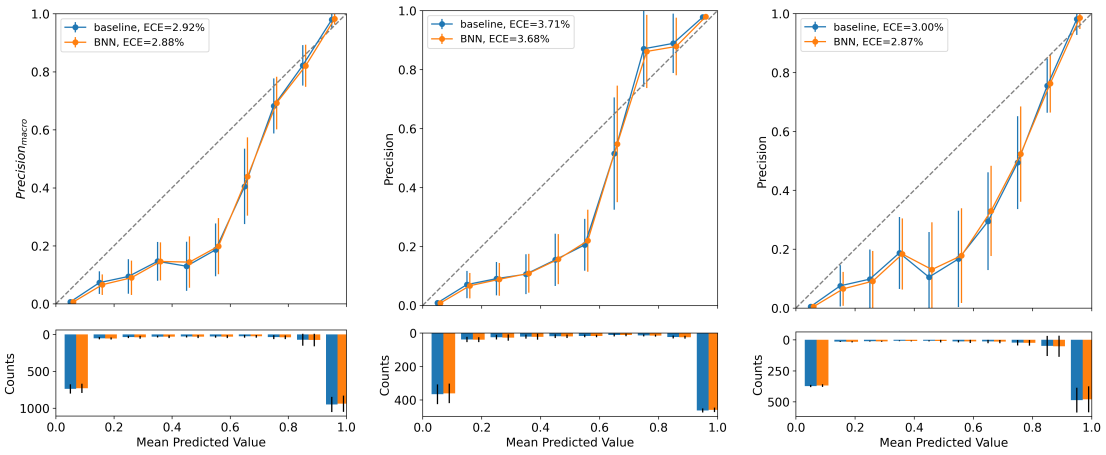
(a) both classes (b) red class (c) blue class

Figure 5.4: Reliability diagrams for the deterministic and probabilistic multi-label classifiers trained on the F^3 dataset (5.4a). (5.4b) and (5.4c) show the reliability diagrams for the red and blue class individually. For the BNNs, the error bars show the standard deviation of the precision over 100 repetitions, while keeping the baseline model fixed.

To account for random effects - e.g. when approximating the Hessian or when sampling weights from the posterior distribution - we repeated the experiment 100 times, meaning that we kept the trained model fixed and approximated 100 posterior distributions and averaged over 100 BNNs.

The results are shown in Figure 5.4a in form of a reliability diagram. It compares the mean calibration curve of the baseline model to that of the BNNs. The error bars indicate the standard deviation of the precision within each bin. We extended the reliability diagrams by adding the probability histograms which show the number of elements in each probability bin and should help with the interpretation of those plots. The BNN shows a very slight improvement in the calibration of the overconfident baseline model by reducing the ECE from 2.35% to 2.14%. Additionally to that aggregated view, we evaluated the calibration of the two classes individually, by treating the class-wise outputs as binary classification results. Figures 5.4b and 5.4c show the reliability diagrams for the red and blue class, revealing that the (very little) overall calibration improvement is caused by the improvement in the red class while the ECE for the blue class increases.

As we can see in the probability histograms, the bins $[0.0, 0.1]$ and $(0.90, 1]$ contain the vast majority of samples. The baseline model classifies more than 87% of the



(a) both classes (b) red class (c) blue class

Figure 5.5: Reliability diagrams comparing the average calibration over 100 models trained on the $F3$ dataset to their BNN versions. The error bars show the standard deviation in the precision.

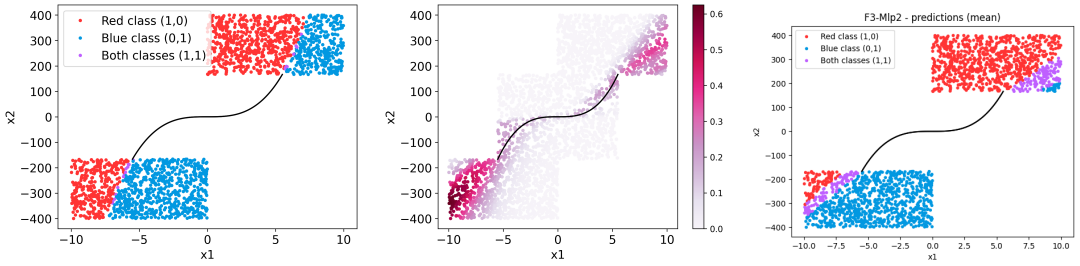
points with a predicted value of below 0.1 or above 0.9, while the BNN reduce that number to approximately 86%.

In a second experiment we generalised the setting considering *any* model trained on the F^3 dataset instead of a specific one. We did this by executing the full cycle (approximate Hessian, find hyperparameters, run 50 forward passes through the BNN), for 100 different models trained on the $F3$ dataset.

As shown in Figure 5.5a, if we generalise our experiment we can't observe a calibration improvement anymore. While the calibration for the red class gets slightly better, the ECE for the blue class is increased. We will come to back to the question whether the Laplace approximation can improve the calibration of a neural network multi-label classifier or not by evaluating the calibration of eight deep neural networks trained on earth observation image data in Section 5.4.2 and in the discussion in Chapter 6.

5.3.4 Out-of-Distribution Detection

Another application of the uncertainty estimates produced by the BNN is the detection of out-of-distribution examples. When classifying examples drawn from a sufficiently different distribution than the training data, the multi-label classifier should either assign the multi-label $\hat{y}_i = 0^C$ or raise a high uncertainty about its result. [Ritter et al., 2018] found, that the uncertainties obtained from the Kronecker-



(a) (b) (c)

Figure 5.6: Model uncertainty on OOD data obtained from running 50 probabilistic forward passes with hyperparameters $\tau = 10$ and $N = 100$. Shown are: The ground truth labels (5.6a), the accumulated standard deviation over both classes (5.6b) and the mean predictions (5.6c) on the OOD data. Figure 5.6b also shows the standard deviation on the in-distribution test set to allow a better comparison.

factored Laplace approximation lead to a higher predictive entropy when passing *notMNIST* images through a neural network trained on *MNIST* compared to a deterministic multi-class classifier. [Humt, 2019] ran similar experiments on *ImageNet* [Russakovsky et al., 2015] for different DNN architectures[6].

Again, we ran 50 forward passes through the BNN with sampling new weights on each run from the approximated posterior distribution. The result is shown in Figure 5.6. The predictions on the OOD data show a higher degree of uncertainty than those on the in-distribution set, with the maximum standard deviation for the OOD data (0.6) being more then twice as high as for the in-distribution points (0.26). Nonetheless, the uncertainties are not equally distributed among the OOD points. Predictions on points where the model fails to extrapolate correctly show a high standard deviation. Those in turn that are correctly classified are predicted quite certain. It is also notable, that the uncertainty is higher in $[-10, 0] \times [-400, -167]$ then in $[0, 10] \times [167, 400]$ which is most probably caused by the training set not being point-symmetric around (0,0) due to the randomised sampling process. Therefore, the model extrapolates differently on both ends of the training interval which is also visible in the mean predictions shown in Figure 5.6c. Those differences in the standard deviations on in-distribution compared to OOD data are promising for being able to detect those OOD examples that can't be classified correctly.

[6]The DNNs used by [Humt, 2019] include those we use in our earth observation experiments in Section 5.4: DenseNet121, DenseNet169, DenseNet201, InceptionV3, ResNet50, ResNet101, VGG16 and VGG19

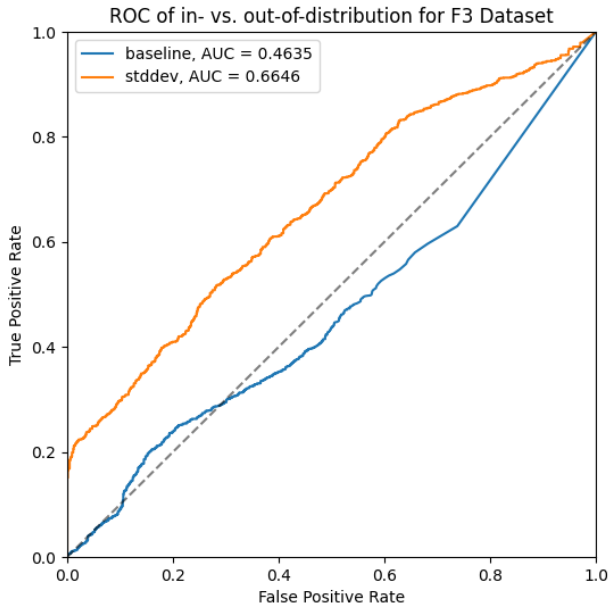


Figure 5.7: ROC curves for the deterministic baseline model using the predicted values and that for the BNN, using the accumulated standard deviations of the predictions.

To analyse the separability of in- and out-of-distribution data we plotted the ROC curves (see Section 5.2.5) for the deterministic baseline model and for the BNN which are shown in Figure 5.7. The ROC curve for the BNN shows the thresholded minimal predictive standard deviation for positive class predictions as a function of the TPR and FPR at increasing threshold levels, labelling in-distribution examples with 0 and OOD examples with 1. We ignore those for the negative class predictions here, as predicting the multi-label $\hat{y}_i = (0, 0)$ on an OOD example i would be a valid outcome if the prediction can't be made correctly with certainty. For the deterministic baseline model we used the negative minimal predicted value[7] instead of the standard deviation as this would be zero. Here, the uncertainties obtained from the BNN clearly improve the ability to separate in-distribution from OOD examples. While the performance of the baseline model with an AUROC score of 0.46 is slightly worse than guessing on random, the AUROC score for the BNN improves by 0.2 to 0.66. We also evaluated the negative minimal predicted value for the BNNs. As this was not notably different to the deterministic baseline model, we

[7]The negative predicted value is needed here as positive in-distribution predictions should have a high probability while that on OOD should be low. For the standard deviation it is the other way round.

left it out of the plots because it did not add any value and would have decreased the readability.

5.4 Earth-Observation Dataset MLRSNet

After evaluating the uncertainties estimated by the Laplace approximation and their potential on improving calibration and OOD detection on a synthetic dataset, we will now take this one step further an examine the method on a real-world multi-label classification problem. The semantic annotation of aerial images by assigning multiple labels that represent the content of the images plays an important role in the remote sensing field [Hua et al., 2020,Li et al., 2020]. In the following section we will build on the previous experiments and evaluate whether the Laplace approximation can be applied to improve (a) the calibration of the classifiers (5.4.2) and/or (b) the detection of out-of-distribution examples (5.4.3). For this purpose we will compare eight DNN models that have been trained on MLRSNet, a multi-label remote sensing dataset.

5.4.1 Setup

Like for the synthetic example, we will start by describing the dataset (5.4.1.1), introducing the deep neural networks we worked with (5.4.1.2) and how we tuned the hyperparameters τ and N (5.4.1.3).

5.4.1.1 Dataset

The Multi-Label Remote Sensing Dataset (MLRSNet)[8] [Qi et al., 2020] is an earth observation dataset for multi-label classification. It consists of 109,161 high resolution remote sensing images with a size of 256×256 pixels, that have been organized in 46 categories, describing the main object in the image. Each image is tagged with 1 to 13 labels from 60 different classes, such as "Car" and "Road" depending on the content of the image [Qi et al., 2020]. The spatial resolution varies from 10 - 0.1 meters. The images have been extracted from Google Earth, which in turn gets its image data from a number of remote imaging sensors. The dataset covers a large variety of spatial locations, seasonal and weather conditions or viewpoints [Qi et al., 2020].

[8]The dataset is available on this GitHub repository: `https://github.com/cugbrs/MLRSNet`

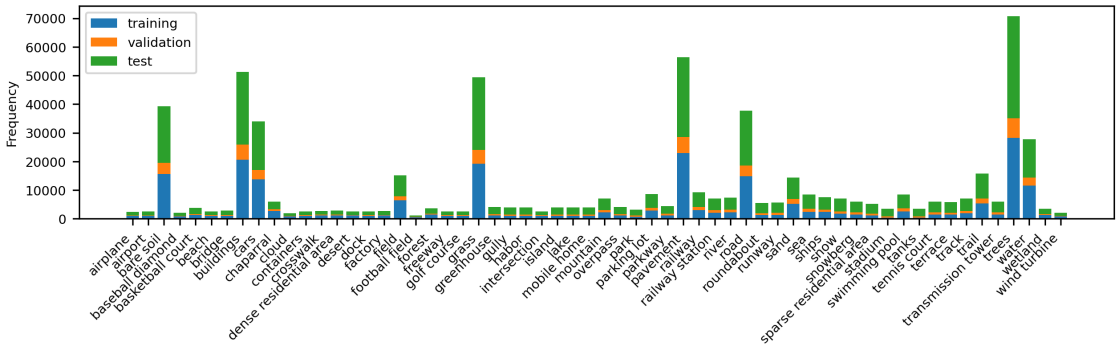


Figure 5.8: Label frequency for the MLRSNet dataset, also showing the training / validation / test split for each class.

Figure 5.8 shows how often a label is assigned in the ground truth of the dataset, revealing a high class imbalance between more general and very specific classes. While the ”Tree” class is assigned to more than 70000 images, the smallest class "Football Field" contains just 1108 instances. Class-imbalance can lead to misleading performance measures, e.g. for accuracy [Buda et al., 2018], as low-frequency classes achieve high accuracy due to a large number of true negatives. We are addressing this problem by (a) using precision as performance metric, which does not account for the negatives and (b) by focusing on the eight best balanced classes (Trees, Pavement, Grass, Buildings, Bar Soil, Road, Car and Field) when evaluating classes individually, which we will refer to as the *top8-frequency* classes.

5.4.1.2 Networks

We will now describe the eight different DNN architectures we used for the earth observation experiments. We choose those eight architectures as [Qi et al., 2020] provided pre-trained models for MLRSNet and to be able to compare our results to those of e.g. [Humt, 2019]. We used the train/validation/test ratio of 0.4/0.1/0.5 that has been used when training the models on MLRSNet [Qi et al., 2020].

VGG16/19 The VGG [Simonyan and Zisserman, 2015] architecture was published in 2015 and won the ILSVRC-2014 localisation challenge [Simonyan and Zisserman, 2015,Russakovsky et al., 2015]. It aimed at understanding the effect, network depth has on the performance in large scale image recognition tasks. It consists of 16 (VGG16) or 19 (VGG19) consecutive convolutional layers with very small (3×3) receptive fields followed by max pooling layers. The classifier is build with three

fully connected layers. The models use weight decay and dropout in the first two fully connected layers for regularisation [Simonyan and Zisserman, 2015]. In our experiments we use both versions, VGG16 and VGG19, which are by far those with the most parameters.

ResNet50/101 ResNet [He et al., 2016] was published in 2016 by Microsoft Research and won the ILSVRC-2015 classification task. It increases depth up to 152 layers which is eight times the size of VGG19. At the same time it reduces complexity and eases the training by introducing residual or shortcut connections, that pass information from previous layers directly to later layers, while skipping those in between. The resulting residual blocks are then stacked to networks with 50, 101 or 152 layers. The classifier consists of one fully connected layer. Weight decay and momentum are used for regularisation [He et al., 2016]. In our experiments we use the versions with 50 (ResNet50) and 101 (ResNet101) layers.

InceptionV3 The Inception [Szegedy et al., 2016] network family has been developed by Google and builds on top of GoogleLeNet. The main difference to other architectures is the introduction of the inception module that increases the networks width and applies several different kernel sizes to its input in parallel instead of stacking them sequentially. This approach addresses the problem of great computational overhead of e.g. VGG nets as well as overfitting. The classifier is formed by an average pooling layer followed by a fully connected layer [Szegedy et al., 2016] . In this work, we use version InceptionV3.

DenseNet121/169/201 The DenseNet [Huang et al., 2017] architecture adopts the idea of shortcut connection from ResNet but goes further by connecting each layer with every other layer within one block in a fully connected manner. The resulting dense blocks are connected to a network with having convolutional, pooling and batch normalisation layers between two dense blocks. The classifier is made of an average pooling and a fully connected layer. DenseNet allows an easier training e.g. by mitigating the vanishing gradient problem while again improving the performance compared to the other networks presented before [Huang et al., 2017]. We use the version with 121 (DenseNet121), 169 (DenseNet169) and 201 (DenseNet201) layers.

An overview over the models we used as a baseline in our experiments and that we extended to BNNs is shown in Table 5.1. We found that the F1 scores for the

Table 5.1: The eight baseline models DNNs used in this work. The table shows the instance based $F1$ and macro averaged precision score as well as the ECE that we achieved when running the models on the test set. The last column shows the size of the approximated curvature matrix when stored on disk.

Model	**Year**	**# Layers**	**# Parameters**	$F1_{instance}$	$Precision_{marco}$	**ECE**	**KFAC Size**
DenseNet121	2018	121	8M	0.6857	0.8116	4.28%	2.1 GB
DenseNet169	2018	169	14M	0.6757	0.8282	4.80%	3.8 GB
DenseNet201	2018	201	20M	0.7029	0.8311	4.41%	5,6 GB
InceptionV3	2015	47	23M	0.6348	0.7799	5.38%	2.7 GB
ResNet50	2015	50	25M	0.6673	0.8190	6.41%	2.9 GB
ResNet101	2015	101	44M	0.5780	0.7818	8.25%	5.3 GB
VGG16	2014	16	138M	0.5012	0.5679	4.65%	2.4 GB
VGG19	2014	19	143M	0.4046	0.6448	5.56%	3.3 GB

baseline models are much lower than published in [Qi et al., 2020], e.g. the paper mentions an F1 score of 0.8538 for DenseNet201 while we got 0.7029 for the same model.

We have not been able to find out where that discrepancy comes from. But we noticed that the order of the classes in the ground truth label files of the Multi-Label Remote Sensing Dataset (MLRSNet) dataset is not consistent across all categories[9]. Even though the authors of MLRSNet fixed those errors in the ground truth files, we can not rule out the possibility, that the pre-trained models we used have been affected by that problem[10]. Nonetheless, as this might only affect a few categories (desert, golf course, intersection, island, parkway, railway station, storage tank and wetland) we retained using those models.

We ran our experiments on a NVIDIA Tesla V100 GPU with 32GB Memory.

5.4.1.3 Hyperparameters

While it has been possible to conduct a grid search on the hyperparameters τ and N for the F^3 dataset, we had to take an alternative approach for the large models used here. The time needed to evaluate the BNN once on the validation set with 50 posterior samples takes from about 22:30 min for VGG16 up more then 50 minutes for DenseNet201. Bayesian optimisation (BO) has shown to be a useful tool for hyperparameter tuning, especially when the evaluation of the optimisation objective

[9]The dataset provides one ground truth label file per category.

[10]For the conversation about this issue, please refer to the thread on this GitHub repository: `https://github.com/cugbrs/MLRSNet/issues/4`

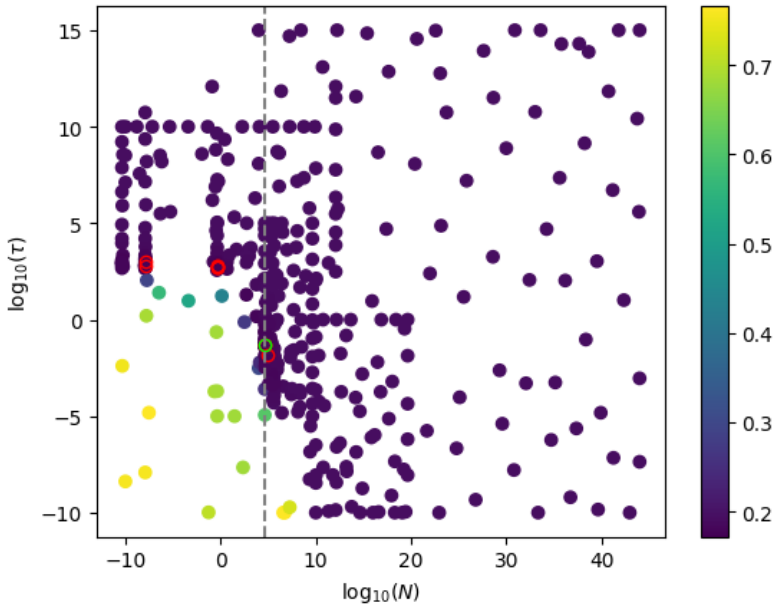


Figure 5.9: Result of multiple rounds of hyperparameter optimisation for τ and N using Bayesian Optimisation for the DenseNet121 model. The 10 best parameter pairs are marked by a colored border with the pair we used in our experiments shown in green. The dashed line marks the training set size of 43645.

is expensive [Joy et al., 2016, Mockus, 1994]. In BO, the unknown function space is modeled using a surrogate model which can be e.g. a Gaussian process[11]. The expensive objective is optimized using a cheap acquisition function [Joy et al., 2016], such as "Expected Improvement" or "Lower Confidence Bound". The acquisition function is used to decide which parameters configuration should be evaluated next, based on the knowledge gained in previous iterations [Joy et al., 2016].

We used the BO implementation in scikit-optimize [Head et al., 2020], which uses a Gaussian Process as surrogate model, with the default setting for the acquisition function argument, which makes the implementation choose randomly between "Expected Improvement", "Probability of Improvement" and "Lower Confidence Bound" as the acquisition function on each iteration. As in the previous experiments, we optimised the NLL as a function of τ and N. We did several optimisation rounds for different areas of the parameters. The result for DenseNet121 is shown in Figure 5.9 while all results are available in appendix A.3.

[11] For a comparison and in-depth discussion of different surrogate models please refer to [Bergstra et al., 2011]

To explore the parameter space, we included values for N smaller than the dataset size. For the experiments in Section 5.4.2 and 5.4.3 we used that parameter pair providing the minimal NLL while having N greater or equal but still close to the dataset size.

5.4.2 Calibration

We valuated the calibration of the eight deep neural networks in a top-down manner, meaning that we will first look at the overall calibration and will focus on individual classes and differences between the architectures afterwards.

Looking at the overall calibration of the multi-label classifiers (Figure 5.10, Table 5.2), we see a slight improvement in all of the eight networks considered, ranging from 0.41% for ResNet50 up to 1.8% for InceptionV3. All networks appear to be underconfident in the lower bins (i.e. for negative class predictions) and slightly overconfident for the towards 1.0. The VGG models do not achieve a precision higher than 0.8, even in the high probability bins which might be caused by their overall very poor performance (see Table 5.1).

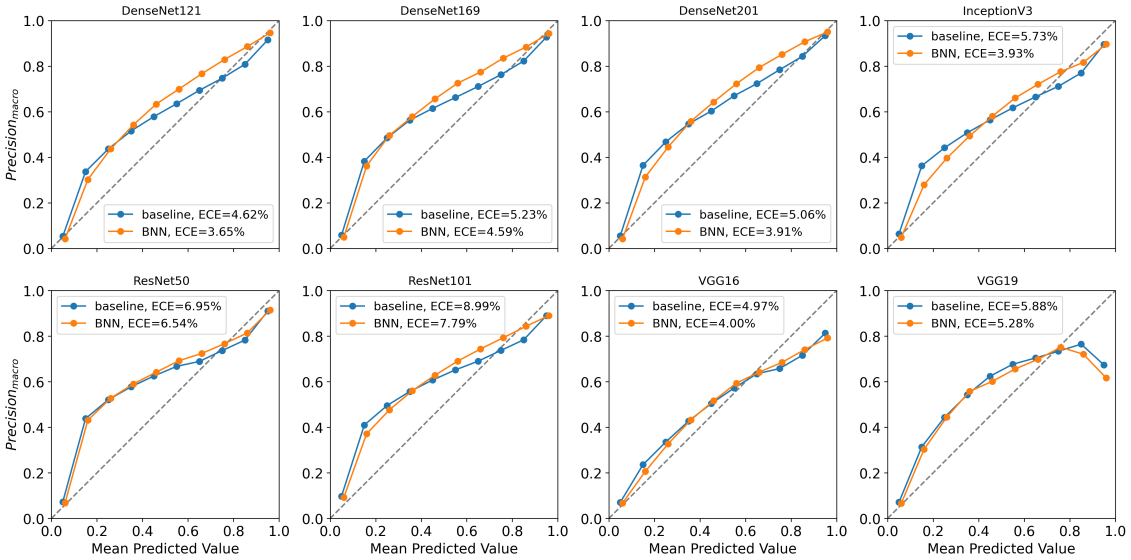


Figure 5.10: Reliability diagrams for the eight networks comparing the baseline calibration to that of the BNNs.

Let's have closer look at individual categories. Figure 5.12 shows the reliability diagrams for the classes *Water* (5.12a) and *Buildings* (5.12b). Both classes are among the top8-frequency classes and while we get overconfident predictions for the *Water*

Table 5.2: Comparing the ECE improvement between the baseline models and the BNNs.

ECE (in %)	DenseNet121	DenseNet169	DenseNet201	InceptionV3	ResNet50	ResNet101	VGG16	VGG19
baseline	4.62	5.23	5.06	5.73	6.95	8.99	4.97	5.88
BNN	**3.65**	**4.59**	**3.91**	**3.93**	**6.54**	**7.79**	**4.00**	**5.28**
Improvement	0.97	0.64	1.15	1.8	0.41	1.2	0.97	0.6

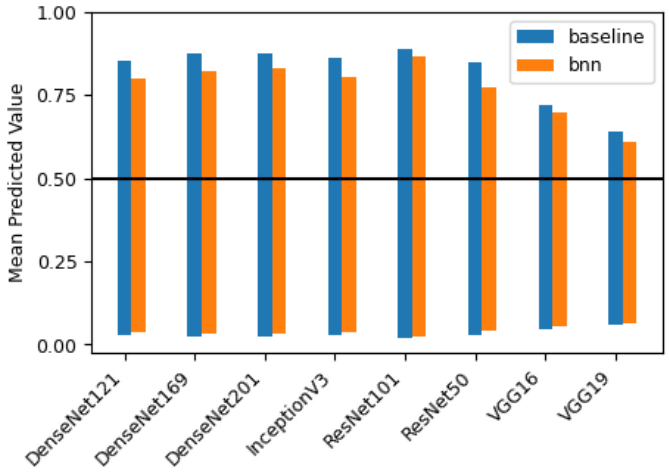
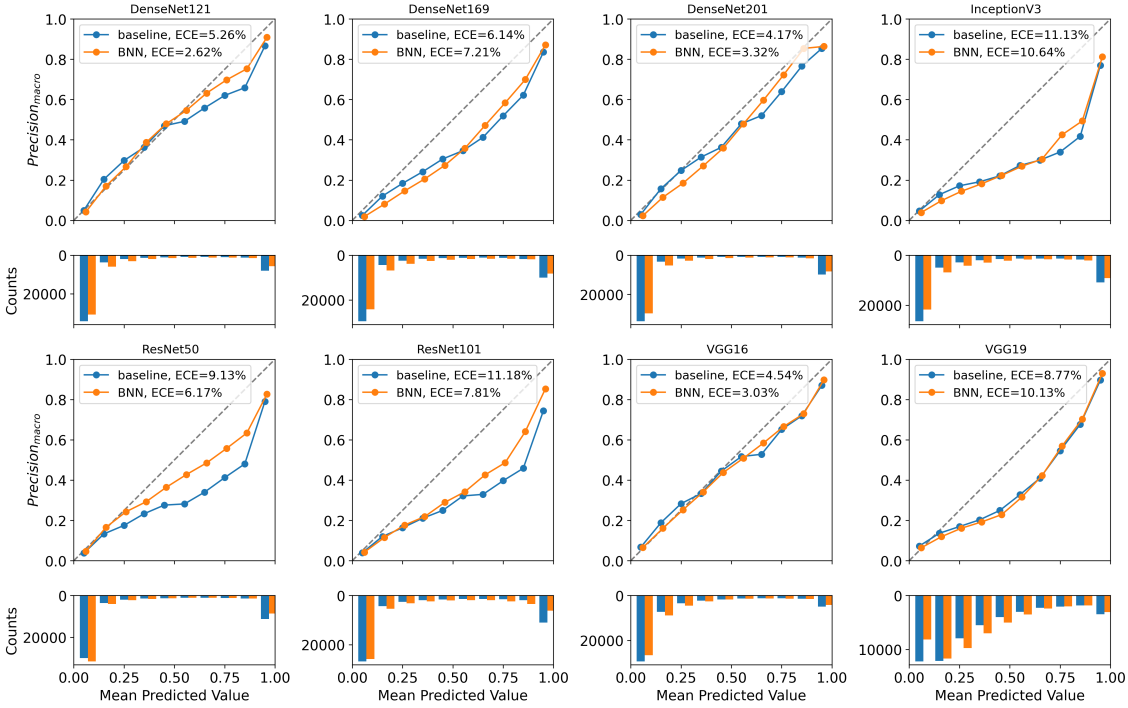


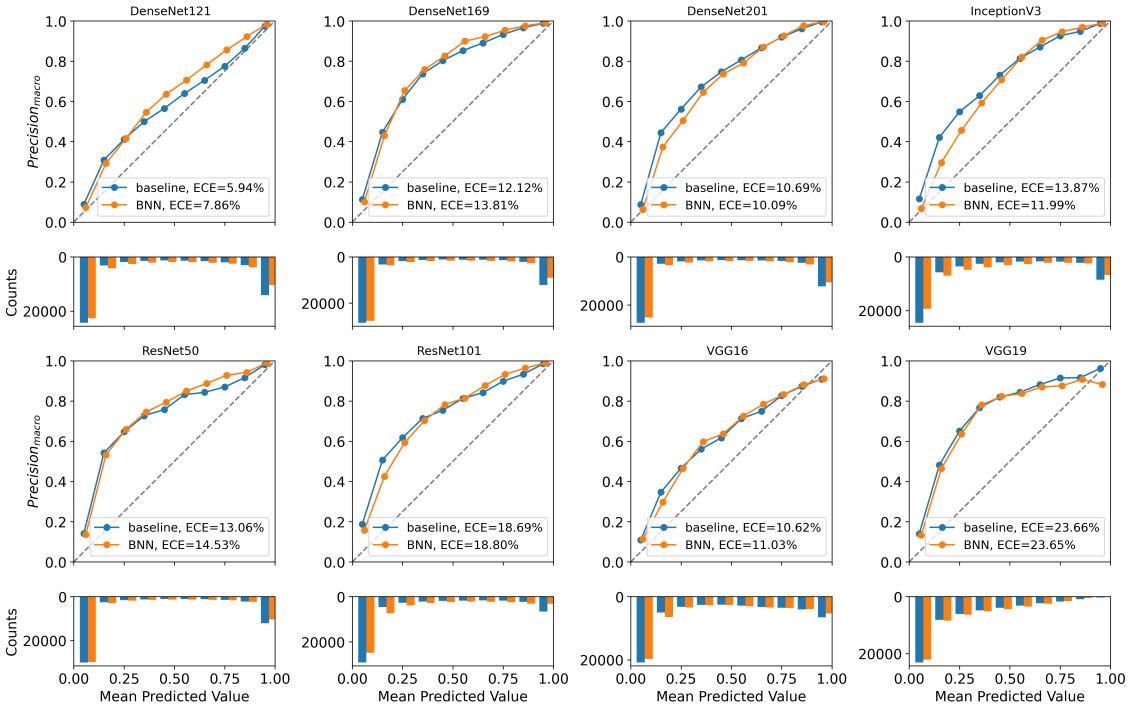
Figure 5.11: Average predicted value for the positive class prediction (upward bars) and negative class prediction (downward bars) for the baseline models and their respective BNN versions.

class, the predictions for class *Buildings* are underconfident. This behaviour is consistent across all eight architectures. For the overconfident *Water* class, the ECE has been improved in six of eight cases, for the underconfident *Buildings* class just in three. Looking at the probability histograms we notice that the baseline models put more instances in the high-confidence bins ($[0.0, 0.1]$ and $(0.9, 1]$) whereas the probabilities for the BNN are shifted towards the lower confidence regions, i.e. closer to 0.5. This observation persists throughout the other classes. Figure 5.11 summarises this, as it compares the average predicted values for positive and negative class predictions between the deterministic baseline models and their respective BNN versions across all eight network architectures. The shift towards 0.5 is more obvious for the positive class predictions than for the negatives. This raises the suspicion, that the Laplace approximation might not improve the calibration in all cases, but reduces the confidence which has an improving effect on overconfident classifiers, but degrades calibration for underconfident ones. We will present an explanation for this observation in the discussion.

When focusing on the results for the top-8-frequency classes as presented in Table 5.3, we notice that the only remarkable improvement in the ECE was achieved for



(a) *Water*



(b) *Buildings*

Figure 5.12: Calibration diagrams for the two classes *Water* and *Buildings*

Table 5.3: Differences in the ECE between the baseline model and the BNN for the top8-frequency classes. The last row shows the average improvement of the ece across those 8 classes.

ECE (in %)		DenseNet121	DenseNet169	DenseNet201	InceptionV3	ResNet50	ResNet101	VGG16	VGG19
Bare Soil	baseline	**7.41**	7.56	**8.12**	7.37	**10.49**	9.12	5.26	**5.98**
	BNN	8.38	**5.62**	10.78	**6.32**	11.59	**6.69**	**3.68**	6.03
Buildings	baseline	**5.94**	**12.12**	10.69	13.87	**13.06**	**18.69**	**10.62**	23.66
	BNN	7.86	13.31	**10.09**	**11.99**	14.53	18.8	11.03	**23.65**
Cars	baseline	2.51	**5.96**	3.24	6.28	**14.04**	**11.44**	8.80	**15.78**
	BNN	**2.02**	8.02	**2.86**	**4.06**	14.21	11.51	**8.64**	16.56
Grass	baseline	**6.41**	**4.72**	16.01	11.92	**9.42**	**21.52**	11.65	5.30
	BNN	8.27	5.18	**15.63**	**8.10**	10.13	24.04	**10.78**	**4.88**
Pavement	baseline	**4.78**	3.34	**4.47**	**8.45**	7.81	5.83	**11.6**	**19.99**
	BNN	5.30	**3.26**	5.63	9.20	**7.80**	**1.46**	11.96	22.57
Road	baseline	**1.54**	4.61	1.76	6.00	**10.00**	10.63	**13.42**	**14.92**
	BNN	2.74	**2.30**	**1.46**	**4.02**	10.75	**9.87**	13.37	16.91
Trees	baseline	**7.92**	**11.51**	15.02	5.31	8.09	**12.95**	**3.84**	4.04
	BNN	8.19	11.99	**13.41**	**2.94**	**7.50**	16.67	4.53	**3.99**
Water	baseline	5.26	**6.14**	4.17	11.13	9.13	11.18	4.54	**8.77**
	BNN	**2.62**	7.21	**3.32**	**10.64**	**6.17**	**7.81**	**3.03**	10.13
% Improvement		-0.45	-0.26	0.04	1.63	-0.08	0.56	0.34	-0.79

InceptionV3. For this network, the calibration is improved for seven in eight cases. While DenseNet121, ResNet101 and VGG16 still show a slight improvement, the ECE for DenseNet121, DenseNet169, ResNet50 and VGG19 increases. Hence for the latter, the overall improvement is caused by the lower frequency classes. To summarize the results we state, that using a BNN with the posterior distribution being approximated using Laplace approximation might not improve the Calibration of the baseline models in all cases, but reduces the overall confidence which has an improving effect on overconfident classifiers.

5.4.3 Out-of-Distribution Detection

As the results from the experiments on the F^3 dataset in Section 5.3.4 have been promising that the Laplace approximation BNN could be useful to separate in-distribution from OOD examples, we conducted a similar experiment for the eight DNN models and their Bayesian extensions. As OOD data, we used a 100 images subset of the *VOC2007* [Everingham et al., 2007] scene classification dataset. By evaluating the predictions the models made on the *VOC2007* images we found, that almost always the top8-frequency classes get predicted, with the baseline models producing slightly more positive class predictions than the BNNs. Figure A.3a and A.4a in appendix A.3 illustrate this observation, which might be caused by a strong bias towards these classes due to their over-presence in the ground-truth. Another reason could be, that elements or areas in those images are either correctly detected or misinterpreted by the neural networks.

(a) baseline (b) baseline (c) baseline

(d) BNN (e) BNN (f) BNN

Figure 5.13: Three examples for predictions on the OOD dataset *VOC2007* as made by the DenseNet121 model. The upper row contains the predictions of the baseline model while those of the BNN are shown below.

Figure 5.13 shows three examples for predictions on the *VOC2007* dataset as made by the DenseNet121 model. While the prediction of *grass* and *base soil* in image 5.13a might be reasonable misinterpretations for a model that has been trained to see the world from bird's eye perspective, the predictions in image 5.13b appear rather to be caused by a bias towards the high-frequency classes. In the third picture, the water the labels *Water* and *Lake* seem to be assigned correctly. However, the BNN reduces the amount of predicted labels in all three cases. This fewer positive class predictions on *VOC2007* made by the BNN compared to the baseline models is observed for all network architectures (see Figure A.4a in appendix A.3).

Our experiment on the synthetic dataset in Section 5.3.4 intended, that using the standard deviations in the predictions of the BNN can improve the separability of in-distribution and OOD data, compared to using the predicted values of the baseline model. When evaluating the ROC curves shown in Figure 5.14 for separating MLRSNet from *VOC2007* data, we see the majority of BNNs fails to do so. A larger AUROC score is only observed for the BNN versions of DenseNet201 and ResNet50. While for DenseNet201 the AUROC improved by 8%, the performance gain of more than 22% for ResNet50 was notably higher.

We reran the experiment with *MNIST* being used as the OOD dataset. Regarding the positive class predictions (see Figure A.3b in appendix A.3), we get a similar picture as for *VOC2007*. The predicted labels concentrate on a few classes, though

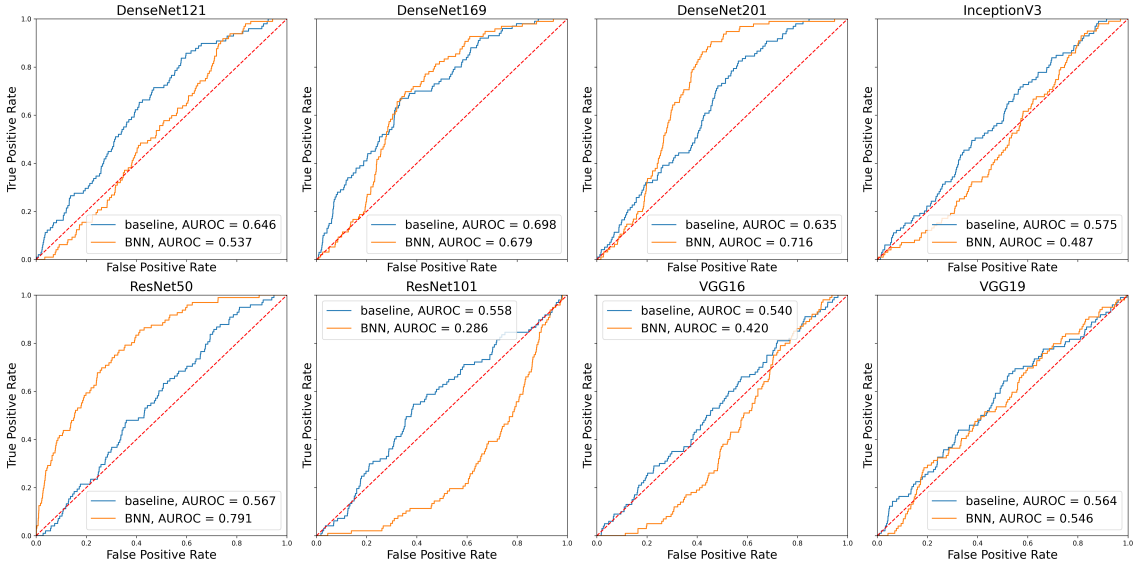


Figure 5.14: ROC curves describing the ability of the baseline models and their Bayesian extensions to separate in-distribution earth observation images from *VOC2007* data.

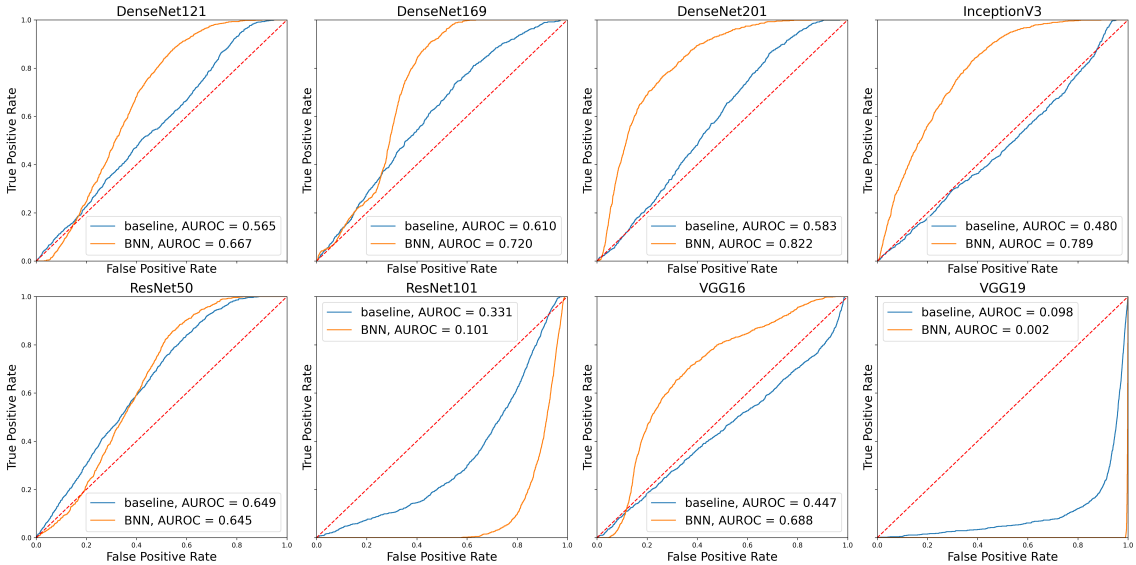


Figure 5.15: ROC curves describing the ability of the baseline models and their Bayesian extensions to separate in-distribution earth observation images from *MNIST* data.

the differences between the networks are greater. For example, almost all pictures are labelled with *Greenhouse*, *Transmission Tower*, *Trees* and *Water* by the VGG19 model, while ResNet101 "sees" all the digits as *Pavement* and *Water*.

Looking at the ROC curves in Figure 5.15 for separating in-distribution from OOD images, we see this time an improvement for 5 out of 8 architectures of up to 30%

for InveptionV3. While there could no improvement be obtained for ResNet50, the Bayesian version of ResNet101 clearly underperforms compared to its deterministic baseline model. The BNN for VGG19 misclassifies "perfectly" in-distribution images as OOD and vice versa. For the DenseNets, the intensity of the improvement seems to be coupled to the network depth - as does it for ResNet and VGG but here in the opposite direction.

Although the results for *VOC2007* as OOD dataset might look rather daunting, those for *MNIST* show that in the majority of cases we can improve the detection of OOD examples by using the standard deviation of the predictions obtained from the BNNs. Reasons for the underperformance of the BNNs on *voc2007* as well as that for VGG and ResNet101 on *MNIST* will be addressed in the following discussion.

6 Discussion

In the following discussion, we will summarise the results from the experiments we described in the previous chapter. We ran our experiments with the expectation, that using a Bayesian neural network based on the posterior distribution over the parameters being approximated by the Kronecker-factored Laplace approximation will improve the calibration of our classifier and the separability from in- and out-of-distribution data. We got that expectations from the findings of [Ritter et al., 2018] and [Humt, 2019], who showed that the method can improve both for regression and multi-class classification problems.

In the calibration experiments we saw slight improvements up to 1.8% in the calibration of the networks when comparing the baseline- and BNN classifiers on a macro-averaged ECE basis. However, when focusing on individual classes we got mixed results, depending on whether the classifier is over- or underconfident on the respective class, which can be summarised as a confidence shift towards the decision boundary of 0.5, meaning that the BNN produced probabilities closer to 0.5 compared to the deterministic baseline models.

To get an idea where that confidence decline might come from let's have a closer look at what happens to the network outputs when we change from deterministic to probabilistic multi-label classification models. The Laplace approximation approximates the posterior distribution over the parameters θ of a neural network with a multivariate Gaussian around the point estimates θ^* which are obtained by training the model on a dataset $\mathcal{D}$. The class probability outputs of our multi-label classification are received by applying the sigmoid function σ to the logits z of the neural network, i.e. the result in the output neurons. Those logits in turn are computed as a linear combination of the parameters of that last layer and the incoming activations from the previous one, giving the class probability vector as

$$\hat{p} = \sigma(z) = \sigma(\theta_L a_{L-1}) \tag{6.1}$$

for a network with L layers.

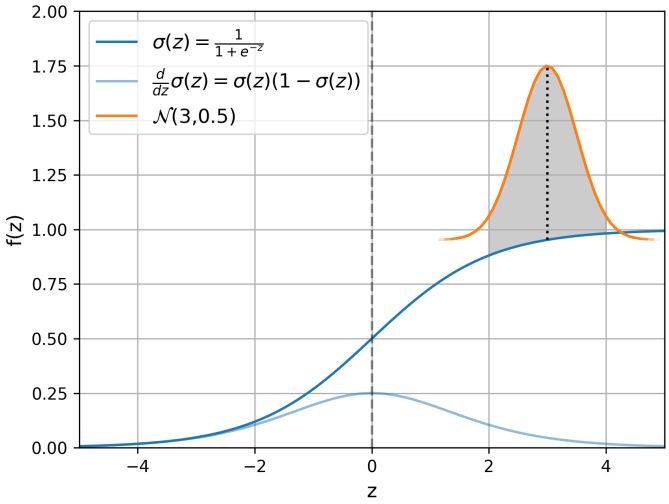


Figure 6.1: Sampling a parameter from a univariate Gaussian $\mathcal{N}(3, 0.5)$. The light blue curve represents the slope of the sigmoid function shown in dark blue.

Figure 6.1 illustrates this for the simplistic case of one output neuron with a single parameter drawn from an univariate normal distribution with mean 3 and standard deviation 0.5 and an incoming activation of 1. The grey area marks the 95% quantile. As the slope of the sigmoid function is higher on the left-hand side of the mean than to the right, the differences in the resulting class probabilities are higher as well. The average slope of the sigmoid function in the left-hand part of the 95% quantile in Figure 6.1 is ~0.0718 while it is just ~0.0294 on the right-hand side. This leads to a decrease in the average value of the sigmoid function when drawing a number of samples compared to the sigmoid of the mean and could be one reason for the shift towards 0.5 we saw in our experiments. As the sigmoid function is point symmetric around $(0, 0.5)$, the same applies for logits lower than zero. So far, we have just focused on the parameters in the final layer. The influence of that mechanism to the parameters in earlier layers as well as the interaction with e.g. batch normalisation or pooling is not considered here yet and could be subject to future work.

In a second series of experiments we investigated the application of the uncertainties that we get from applying the Laplace approximation to improve the separation from in- and OOD data. We compared the ability of the baseline model to separate in- from out-of-distribution to that of the BNN by comparing their receiver operating characteristics. For the baseline model we used the minimal predicted probabilities of the positive class predictions as a separator, for the BNN we separated by their minimal standard deviation. We have been able to increase the AUROC for the BNN

on the F^3 dataset by 20% compared to the baseline model. For the DNNs we got heterogeneous results for the two different OOD sets we used. With *MNIST* as OOD dataset, we have been able to increase the AUROC up to 30% for the majority of networks, though two of the remaining DNNs - ResNet101 and VGG19 - drastically underperformed. This is interesting, as the differences in the AUROC compared to their shallower versions are immense. We can't explain yet where those differences come from, but the depth of a network seems to play an important role here. It is also those two networks, who label almost all OOD examples with the same two or four labels respectively. While the prediction of *Water* on black-and-white images by networks who have been trained on earth observation data seems reasonable, it might be instructive to understand which features led to the the labelling of almost all images with *Transmission Tower*, as done by ResNet50, VGG16 and VGG19. In general, the predicted labels concentrate only on a few of the 60 available classes but vary between the different networks, which might be caused by the varying network specific features extractions.

Besides *MNIST*, we ran the experiment with using a subset of the *VOC2007* dataset as OOD examples. Here, we also observed the concentration of positive class predictions on just very few classes but it seems to be more evenly distributed among the different networks. In case of *VOC2007* the predicted classes match those with the highest frequency in the ground truth. While the separability of in- and OOD data could not have been improved for the majority of networks, the tendency between the different versions of the neural networks looks comparable to the results we got with *MNIST*. For the DenseNets, the separability improves with increasing depth, while for ResNet the deeper BNN performs much worse than its shallower version. Altogether, *VOC2007* might have been a problematic choice for an OOD dataset here. While it contains images from scenes and objects taken from a ground-floor perspective and could therefore be seen as out-of-distribution to aerial images, the *VOC2007* images often contain objects that the networks also have been trained to detect, like trees or water and might therefore be able to assign the correct labels. Another problem might be caused by the models misinterpreting elements or structures in the *VOC2007* images as something they learned, e.g. trees as grass or linear structures as roads. This effect might be enforced as [Qi et al., 2020] used transfer learning in ImageNet [Russakovsky et al., 2015] to train their models. An alternative to using *VOC 2007* as OOD data would have been to train the networks on just a subset of MLRSNet containing e.g. just pictures of urban areas and use woodland or water images as the OOD set. As this would have exceeded the scope

of this thesis, it could be an interesting topic for ongoing research as well as understanding the predictions the networks made on the OOD datasets we used here, e.g. by using methods from the explainable AI field.

A third topic which turned out to be unexpectedly difficult was the determination of suitable values for the two parameters τ and N, which are used to regularise the curvature and thus the covariances. While τ is the precision of the Gaussian prior on the parameters and N the size of the dataset, [Ritter et al., 2018] state, that those values can be treated as hyperparameters and optimised as such. We took that hyperparameter approach and used Bayesian optimisation or grid search (where applicable) to find appropriate values, where we had to decide (a) which criterion to optimise to retrieve good uncertainty estimates and (b) which range of values to search. We could not optimise directly for calibration or good separability without sacrificing predictive performance. Assuming a balanced dataset, the uniform output of 0.5 would lead to a perfectly calibrated model but with a probably undesirable predictive performance. Good separability of in- and OOD data does neither guarantee high quality predictions. We decided to optimise the negative log likelihood which is the probabilistic interpretation of the binary cross entropy loss that has already been minimised when training the network. The second question and especially that about the range for N appeared to be much harder to answer. While we noticed that when allowing values $N < 1$, those often yield the lowest NLLs, we hesitated to use those small values for three reasons: First, we could not interpret those values as we could for values for N larger than the dataset size. Second, choosing $N < 1$ would downscale the curvature factors and thus increase the covariances in each layer which bears the risk of overestimating the uncertainties. And last, we did not find any other work, using values for N smaller than the dataset or even $N < 1$. Finally we choose values for N that are larger but close to the dataset size to retain enough variability when sampling parameters, even though we included smaller values in the hyperparameter search to develop a better understanding. For the second parameter τ, which has only an effect on the diagonal elements of the curvature factors, we considered values between 10^{-10} and 10^{10}. We found that small values for both parameters yield large NLLs which is not desirable. When plotting the NLL as a function of the two parameters, the lowest NLLs are located on a curve around the area of high NLLs. The open question, whether or not choosing $N < 1$ would be appropriate, would also be an interesting topic to investigate further.

7 Conclusion & Outlook

In this work we derived and evaluated the estimation of model uncertainty for multi-label classification tasks. We approximated the posterior distribution over the parameters of neural networks using Laplace approximation, which does so by placing a multivariate Gaussian around the MAP estimate obtained from training the network. We thereby focused on the characteristics of the multi-label setting such as the compatibility of the binary cross entropy loss with the requirements for using the Fisher information matrix as a block-diagonal approximation to the Hessian of a neural network. With experiments on a synthetic and a large-scale remote sensing dataset, we visualised and evaluated the obtained uncertainty estimates. While we gained slight improvements on the overall calibration of a given multi-label classifier, we saw mixed outcomes when looking at individual classes. While we expected to see a general improvement on the calibration from using a probabilistic instead of a deterministic model, we noticed that this only holds for overconfident classifiers as the class probabilities shift towards 0.5 for the BNN. We consider that this is caused by the slope of the sigmoid function being steeper towards 0.5. In a second experiment we used the uncertainties in form of predictive standard deviation to separate in-distribution from out-of-distribution data. While we have been able to improve the separability for most of the considered DNN architectures when using *MNIST* [LeCun et al., 2010] as OOD data, this was not achieved when running the experiment with *VOC2007* [Everingham et al., 2007] as OOD examples. Although, we found that using *VOC2007* here might not have been the optimal choice for an OOD dataset, we still found that the ability to improve the separability seems to be depending on the network depth.

Finally, the optimisation of the hyperparameters that are used to regularise the curvature factors has shown to need far more thought then assumed initially. While the choice is crucial for obtaining high quality uncertainty estimates, the way to come to that choice is still not totally clear. We used Bayesian optimisation as an approach to find suitable hyperparameter pairs, but the question about the appropriate range especially for the parameter N leaves room for further works in this direction.

Besides the question if and how the choice of $N < 1$ could be justified there are several more interesting directions to take from here. Extending on the idea why approximating the posterior distribution for networks using sigmoid activations on their outputs might lead to a shift of the class probabilities towards 0.5, understanding those effects on earlier and different types of layers might eventually lead to methods for improving calibration in general. Additionally, a further investigation of the prediction patterns on the OOD datasets might improve understanding the network specific feature extraction and decision making mechanisms in terms of explainable results.

Bibliography

[Abadi et al., 2015] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[Alazaidah and Ahmad, 2016] Alazaidah, R. and Ahmad, F. K. (2016). Trending challenges in multi label classification. *International Journal of Advanced Computer Science and Applications*, 7(10):127–131.

[BakIr et al., 2007] BakIr, G., Hofmann, T., Smola, A. J., Schölkopf, B., and Taskar, B. (2007). *Predicting structured data*. MIT press.

[Barber, 2012] Barber, D. (2012). *Bayesian Reasoning and Machine Learning*. Cambridge University Press.

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

[Becker and Lecun, 1989] Becker, S. and Lecun, Y. (1989). Improving the convergence of back-propagation learning with second-order methods. In Touretzky, D., Hinton, G., and Sejnowski, T., editors, *Proceedings of the 1988 Connectionist Models Summer School, San Mateo*, pages 29–37. Morgan Kaufmann.

[Bergstra et al., 2011] Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24.

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

[Blundell et al., 2015] Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight uncertainty in neural network. In *International Conference on Machine Learning*, pages 1613–1622. PMLR.

[Botev et al., 2017] Botev, A., Ritter, H., and Barber, D. (2017). Practical Gauss-Newton optimisation for deep learning. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 557–565, International Convention Centre, Sydney, Australia. PMLR.

[Boutell et al., 2004] Boutell, M. R., Luo, J., Shen, X., and Brown, C. M. (2004). Learning multi-label scene classification. *Pattern recognition*, 37(9):1757–1771.

[Bradley, 1997] Bradley, A. P. (1997). The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern recognition*, 30(7):1145–1159.

[Buda et al., 2018] Buda, M., Maki, A., and Mazurowski, M. A. (2018). A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks*, 106:249–259.

[Buntine and Weigend, 1991] Buntine, W. L. and Weigend, A. (1991). Bayesian back-propagation. *Complex Syst.*, 5.

[Chen et al., 2020] Chen, W., Zhang, B., and Lu, M. (2020). Uncertainty quantification for multilabel text classification. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 10(6):e1384.

[DeGroot and Fienberg, 1983] DeGroot, M. H. and Fienberg, S. E. (1983). The comparison and evaluation of forecasters. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 32(1-2):12–22.

[Dendamrongvit et al., 2011] Dendamrongvit, S., Vateekul, P., and Kubat, M. (2011). Irrelevant attributes and imbalanced classes in multi-label text-categorization domains. *Intelligent Data Analysis*, 15(6):843–859.

[Denker and LeCun, 1990] Denker, J. S. and LeCun, Y. (1990). Transforming neural-net output levels to probability distributions. In *Proceedings of the 3rd International Conference on Neural Information Processing Systems*, pages 853–859.

[Depeweg, 2019] Depeweg, S. (2019). *Modeling epistemic and aleatoric uncertainty with bayesian neural networks and latent variables*. PhD thesis, Technische Universität München.

[Everingham et al., 2007] Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2007). The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[Gal and Ghahramani, 2015] Gal, Y. and Ghahramani, Z. (2015). Bayesian convolutional neural networks with bernoulli approximate variational inference. *arXiv preprint arXiv:1506.02158*.

[Gal and Ghahramani, 2016] Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR.

[Gawlikowski et al., 2021] Gawlikowski, J., Tassi, C. R. N., Ali, M., Lee, J., Humt, M., Feng, J., Kruspe, A., Triebel, R., Jung, P., Roscher, R., et al. (2021). A survey of uncertainty in deep neural networks. *arXiv preprint arXiv:2107.03342*.

[Ghoshal et al., 2019] Ghoshal, B., Tucker, A., Sanghera, B., and Wong, W. L. (2019). Estimating uncertainty in deep learning for reporting confidence to clinicians when segmenting nuclei image data. In *2019 IEEE 32nd International Symposium on Computer-Based Medical Systems (CBMS)*, pages 318–324.

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

[Graves, 2011] Graves, A. (2011). Practical variational inference for neural networks. In Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc.

[Guo et al., 2017] Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. (2017). On calibration of modern neural networks. In *International Conference on Machine Learning*, pages 1321–1330. PMLR.

[Gupta and Nagar, 2010] Gupta, A. K. and Nagar, D. K. (2010). *Matrix variate distributions*, volume 104. Chapman & Hall/CRC.

[He et al., 2016] 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.

[Head et al., 2020] Head, T., Kumar, M., Nahrstaedt, H., Louppe, G., and Shcherbatyi, I. (2020). scikit-optimize/scikit-optimize.

[Hernández-Lobato and Adams, 2015] 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. PMLR.

[Hinton and Van Camp, 1993] Hinton, G. E. and Van Camp, D. (1993). Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13.

[Hua et al., 2020] Hua, Y., Mou, L., and Zhu, X. X. (2020). Relation network for multilabel aerial image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 58(7):4558–4572.

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

[Humt, 2019] Humt, M. (2019). Laplace approximation for uncertainty estimation of deep neural networks. Master's thesis, TUM.

[Joy et al., 2016] Joy, T. T., Rana, S., Gupta, S., and Venkatesh, S. (2016). Hyperparameter tuning for big data using bayesian optimisation. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2574–2579. IEEE.

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

[Krizhevsky, 2009] Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. Technical report.

[Laplace, 1774] Laplace, P. S. (1774). Memoire sur la probabilite de causes par les evenements. *Memoire de l'Academie Royale des Sciences*, pages 27–65.

[LeCun et al., 2010] LeCun, Y., Cortes, C., and Burges, C. (2010). Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2.

[LeCun et al., 1999] LeCun, Y., Haffner, P., Bottou, L., and Bengio, Y. (1999). Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*, pages 319–345. Springer.

[Lee et al., 2020] Lee, J., Humt, M., Feng, J., and Triebel, R. (2020). Estimating model uncertainty of neural networks in sparse information form. In *International Conference on Machine Learning (ICML)*. Proceedings of Machine Learning Research.

[Li et al., 2020] Li, Y., Chen, R., Zhang, Y., Zhang, M., and Chen, L. (2020). Multi-label remote sensing image scene classification by combining a convolutional neural network and a graph neural network. *Remote Sensing*, 12(23):4003.

[Liu and Nocedal, 1989] Liu, D. C. and Nocedal, J. (1989). On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528.

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

[Martens, 2020] Martens, J. (2020). New insights and perspectives on the natural gradient method. *J. Mach. Learn. Res.*, 21:146:1–146:76.

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

[Mockus, 1994] Mockus, J. (1994). Application of bayesian approach to numerical methods of global and stochastic optimization. *Journal of Global Optimization*, 4(4):347–365.

[Monteiro et al., 2020] Monteiro, M., Le Folgoc, L., Coelho de Castro, D., Pawlowski, N., Marques, B., Kamnitsas, K., van der Wilk, M., and Glocker, B. (2020). Stochastic segmentation networks: Modelling spatially correlated aleatoric uncertainty. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 12756–12767. Curran Associates, Inc.

[Murphy, 2021] Murphy, K. P. (2021). *Probabilistic Machine Learning: An introduction*. MIT Press.

[Naeini et al., 2015] Naeini, M. P., Cooper, G., and Hauskrecht, M. (2015). Obtaining well calibrated probabilities using bayesian binning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29.

[Neal, 1992] Neal, R. M. (1992). Bayesian training of backpropagation networks by the hybrid monte carlo method. Technical report, Citeseer.

[Niculescu-Mizil and Caruana, 2005] Niculescu-Mizil, A. and Caruana, R. (2005). Predicting good probabilities with supervised learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 625–632.

[Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

[Qi et al., 2020] Qi, X., Zhu, P., Wang, Y., Zhang, L., Peng, J., Wu, M., Chen, J., Zhao, X., Zang, N., and Mathiopoulos, P. T. (2020). Mlrsnet: A multi-label high spatial resolution remote sensing dataset for semantic scene understanding. *ISPRS Journal of Photogrammetry and Remote Sensing*, 169:337–350.

[Ritter et al., 2018] Ritter, H., Botev, A., and Barber, D. (2018). A scalable laplace approximation for neural networks. In *International Conference on Learning Representations*.

[Roux and Fitzgibbon, 2010] Roux, N. L. and Fitzgibbon, A. (2010). A fast natural newton method. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, pages 623–630.

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

[Schapire and Singer, 2000] Schapire, R. E. and Singer, Y. (2000). Boostexter: A boosting-based system for text categorization. *Machine learning*, 39(2):135–168.

[Simonyan and Zisserman, 2015] Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*.

[Srivastava et al., 2014] 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.

[Szegedy et al., 2016] 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.

[Tomás et al., 2014] Tomás, J. T., Spolaôr, N., Cherman, E. A., and Monard, M. C. (2014). A framework to generate synthetic multi-label datasets. *Electronic Notes in Theoretical Computer Science*, 302:155â176.

[Tsoumakas and Katakis, 2007] Tsoumakas, G. and Katakis, I. (2007). Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)*, 3(3):1–13.

[Turnbull et al., 2008] Turnbull, D., Barrington, L., Torres, D., and Lanckriet, G. (2008). Semantic annotation and retrieval of music and sound effects. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(2):467–476.

[Wu and Zhou, 2017] Wu, X.-Z. and Zhou, Z.-H. (2017). A unified view of multi-label performance measures. In *International Conference on Machine Learning*, pages 3780–3788. PMLR.

[Xia et al., 2017] Xia, G.-S., Hu, J., Hu, F., Shi, B., Bai, X., Zhong, Y., Zhang, L., and Lu, X. (2017). Aid: A benchmark data set for performance evaluation of aerial scene classification. *IEEE Transactions on Geoscience and Remote Sensing*, 55(7):3965–3981.

[Yang, 1999] Yang, Y. (1999). An evaluation of statistical approaches to text categorization. *Information retrieval*, 1(1):69–90.

[Zagoruyko and Komodakis, 2016] Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. In *BMVC*.

[Zhang and Zhou, 2005] Zhang, M.-L. and Zhou, Z.-H. (2005). A k-nearest neighbor based algorithm for multi-label classification. In *2005 IEEE international conference on granular computing*, volume 2, pages 718–721. IEEE.

[Zhu et al., 2017] Zhu, X. X., Tuia, D., Mou, L., Xia, G.-S., Zhang, L., Xu, F., and Fraundorfer, F. (2017). Deep learning in remote sensing: A comprehensive review and list of resources. *IEEE Geoscience and Remote Sensing Magazine*, 5(4):8–36.

List of Figures

List of Tables

A Appendix

A.1 Decomposing the Hessian

We want to get an approximation of the posterior distribution $p(\theta|\mathcal{D})$ over the network parameters θ given the data $\mathcal{D}$. Following Bayes' Theorem, the posterior distribution is given as the likelihood of the data $p(\mathcal{D}|\theta)$ times the prior parameter distribution $p(\theta)$ normalized by the evidence $p(\mathcal{D})$:

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \ . \tag{A.1}$$

If we take the log on Bayes' Theorem we get:

$$\log p(\theta|\mathcal{D}) = \log p(\mathcal{D}|\theta) + \log p(\theta) - \log p(\mathcal{D}) \ . \tag{A.2}$$

Taking the second derivative on both sides w.r.t. θ shows that the Hessian of the log posterior $H_{\log p(\theta|\mathcal{D})}$ can be expressed by adding the Hessian of the log likelihood $H_{\log p(\mathcal{D}|\theta)}$ and the Hessian of the log prior $H_{\log p(\theta)}$:

$$H_{\log p(\theta|\mathcal{D})}(\theta) = H_{\log p(\mathcal{D}|\theta)}(\theta) + H_{\log p(\theta)}(\theta) - \underbrace{H_{\log p(\mathcal{D})}(\theta)}_{\substack{=0, \text{ as } \log p(\mathcal{D}) \text{ is} \\ \text{not depending on } \theta}} \tag{A.3}$$

The Hessian $H_{\log p(\theta)}(\theta)$ of the log prior, with prior mean θ_0 and precision matrix $\tau I = \Sigma^{-1}$ of size $n \times n$ is $-\tau I$, which can be seen as follows:

$$\begin{aligned} \log p(\theta) &= log\Big(\frac{1}{\sqrt{(2\pi)^n|\Sigma|}}\Big) - \frac{1}{2}(\theta-\theta_0)^\intercal \tau I(\theta-\theta_0) \\ \nabla \log p(\theta) &= 0 - 2\frac{1}{2}\tau I(\theta-\theta_0) = -\tau I(\theta-\theta_0) \\ H_{\log p(\theta)}(\theta) &= -\tau I \end{aligned} \tag{A.4}$$

Thus, we can express the Hessian of the log posterior as:

$$H_{\log p(\theta|\mathcal{D})}(\theta) = H_{\log p(\mathcal{D}|\theta)}(\theta) - \tau I \tag{A.5}$$

where τ is the precision of the gaussian prior.

Analogous, we can get the Hessian of the negative log posterior (which is identical to the negative Hessian of the log posterior) by multiplying A.2 with -1 and taking the second derivative w.r.t. θ:

$$\begin{aligned} -\log p(\theta|\mathcal{D}) &= -\log p(\mathcal{D}|\theta) - \log p(\theta) + \log p(\mathcal{D}) \\ -H_{\log p(\theta|\mathcal{D})}(\theta) &= -H_{\log p(\mathcal{D}|\theta)}(\theta) - H_{\log p(\theta)}(\theta) + \underbrace{H_{\log p(\mathcal{D})}(\theta)}_{\substack{=0, \text{ as } \log p(\mathcal{D}) \text{ is} \\ \text{not depending on } \theta^*}} \end{aligned} \tag{A.6}$$

Thus, we can express the Hessian of the negative log posterior with:

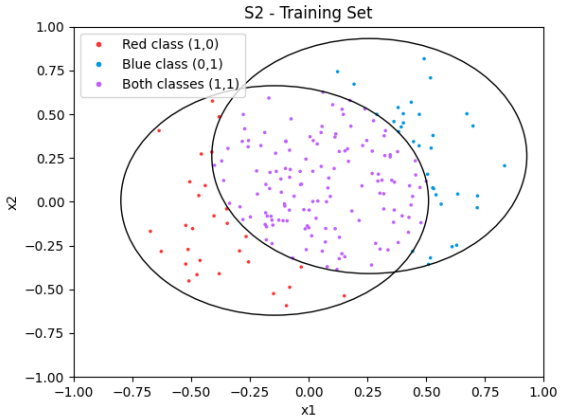
$$-H_{\log p(\theta|\mathcal{D})}(\theta) = -H_{\log p(\mathcal{D}|\theta)}(\theta) + \tau I \tag{A.7}$$

where τ is the precision of the gaussian prior. As shown in section 4.2, the negative log likelihood is just the loss function that we aim to minimize during training.

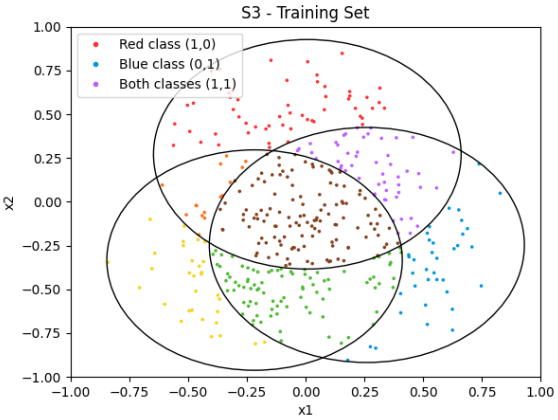
A.2 Hypersphere datasets

The hypersphere datasets are based on the idea of the "Mldatagen" framework presented by [Tomás et al., 2014]. In this framework, multi-label datasets are generated by sampling points uniformly from intersecting hyperspheres in $[-1, 1]^D$. The dimension D of the hypersphere represents the dimension of the feature space while the C hyperspheres represent the classes. We adopt this idea, but instead of sampling the points uniformly from the hyperspheres we draw them from normal distributions $\mathcal{N}(c_i, r_i^3)$ where c_i denotes the center and r_i the radius of the i-th hypersphere. The hyperspheres, a point j does or does not belong to, define the multi-label $y_j \in 0, 1^C$ of this point, i.e. if a point j is internal to hypersphere i, the i-th dimension of the multi-label y_j is 1, and 0 otherwise. The radii $r_i, i = 1 \dots C$ of the hyperspheres, with $0 < r_{\min} \leq r \leq r_{\max} < 1$, are sampled uniformly from the interval between the minimal and maximal radius which needs to be specified when creating the dataset. The centers $c_i, i = 1 \dots C$ of the hyperspheres are then sampled uniformly from $[-1 + r_i, 1 - r_i]^D$.

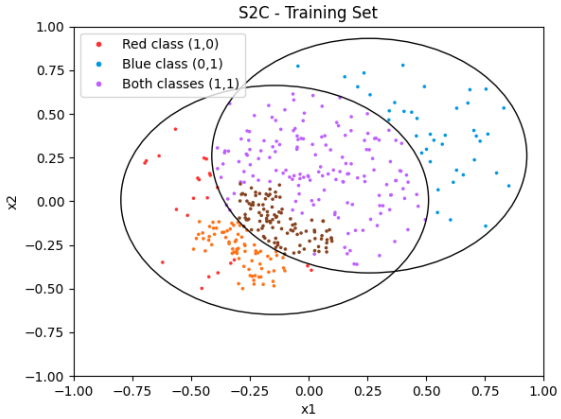
As we want to create a dataset that is easy to visualise we choose $d = 2$. Figure A.1a and A.1b show two examples of hypersphere datasets for $d = 2$ and $C \in \{2, 3\}$. The multi-label of each point is encoded by its color. The "2S" (2 Spheres) dataset shown in Figure A.1a has a red and a blue class. Points that belong to both classes have purple color. The "2S" dataset is the most basic we can create using this approach. The "3S" dataset in Figure A.1b has an additional yellow class. Based on dataset "S2", we created the datasets "S2C" (2 Spheres and Cross) and "S2L" (2 Spheres and Line), which are shown in Figures A.1c and A.1d and have two hypersphere shaped classes and an additional class with a different shape. While the "S2C" dataset has a cross shaped yellow class which forms a subclass of the red hypersphere class, the yellow class in the "S2L" dataset forms a thick line that spans across both hypersphere classes.



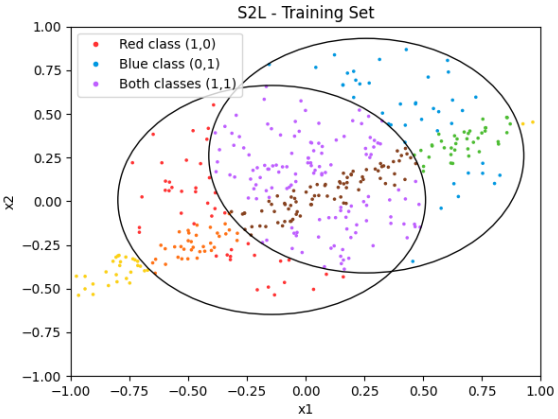
(a) "S2" dataset - the most basic dataset consisting of two hypersphere classes (red and blue).



(b) "S3" dataset - three hypersphere classes (red, blue and yellow)



(c) "S2C" dataset - S2 with an additional cross shaped yellow class.



(d) "S2L" dataset - S2 with an additional thick yellow line as third class.

Figure A.1: The hypersphere-based datasets.

A.3 Additional Plots

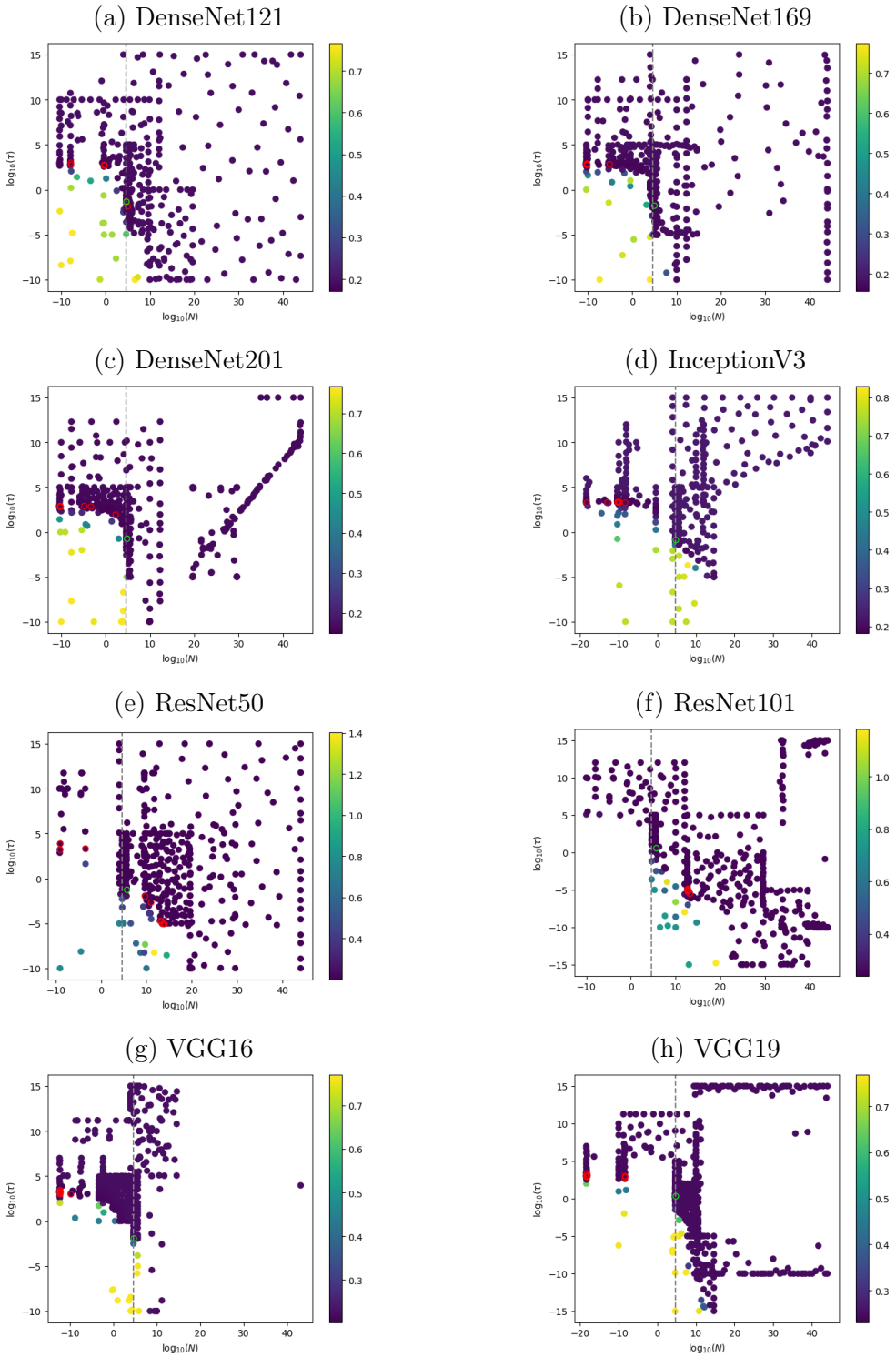
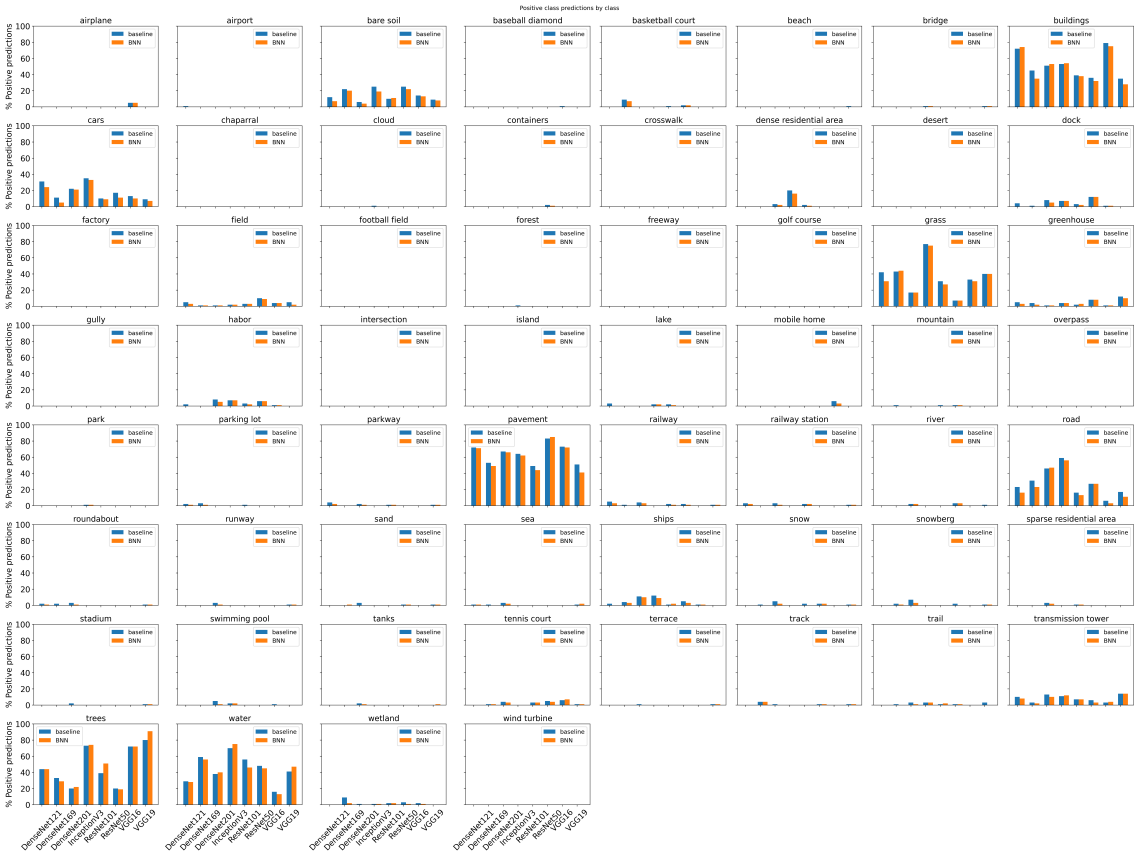


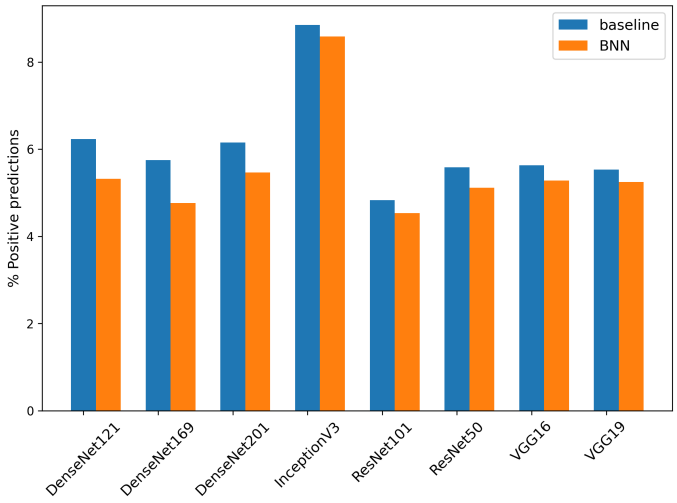
Figure A.2: Results of the hyperparameter search for τ and N using Bayesian optimization.



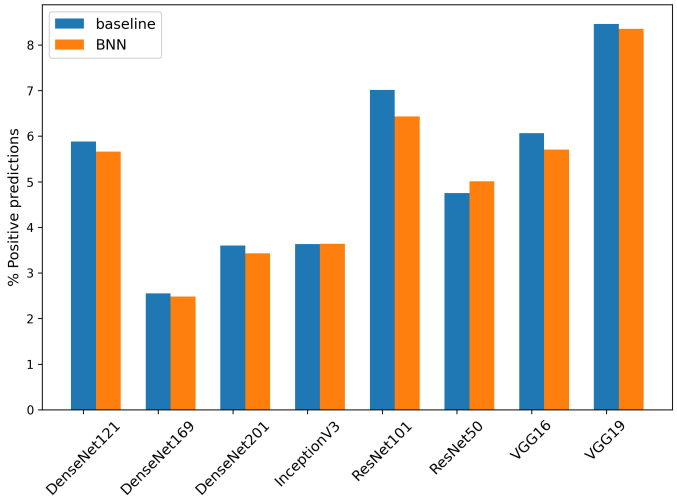
(a) VOC2007

(b) MNIST

Figure A.3: Class-wise predictions on the two OOD datasets.



(a) *VOC2007*



(b) *MNIST*

Figure A.4: Total number of positive class prediction on the OOD datasets as made by the baseline models compared to their respective BNNs.

Declaration of originality

I hereby declare that this Master's Thesis is my own work and I have documented all sources and material used.

As the author, I don't have any objections against making this work publicly usable in the archive of the Friedrich-Schiller-Universität Jena.

Jena, 29/07/2021