# Bayesian Learning and Optimization of Neural Networks

Thesis by

John Valen

In Partial Fulfillment of the Requirements for the Degree of Master of Statistics

Katholieke Universiteit Leuven

Leuven, Belgium

**Advisor:** Dr. Shell Xu Hu

**Supervisor:** Prof. Johan Suykens

June 2020

## Acknowledgements

Over the past year, I have spent a good amount of time contemplating whether Bayesian methods can adequately complement the frequentism that we take for granted in neural networks – indeed, they are almost part and parcel: few notice the general reliance on frequentist techniques and fewer still question it. It seems that to suggest the alternative Bayesian approach is to suggest going back to the drawing board, and I hope that this work demonstrates an ample amount of the benefits that may make such a revisit worth it.

I firstly wish to thank my supervisor Prof. Johan Suykens for suggesting a more formal and attentive treatment of Bayesian inference to convolutional neural networks. His guidance, and his recommendation of the seminal work on Bayesian neural networks of David MacKay opened my eyes and my mind to this topic.

I am also extremely thankful to my advisor Dr. Shell Xu Hu who directed me in this investigation, and without whom I may have settled for a less meaningful work. An expert in Bayesian inference, his feedback and his Skype conversations kept me motivated all throughout.

I am very grateful to my techy friends Indranil Balki, Mauro Mendez, and Bohdan Zaremba who helped not only proof-read this work, but questioned my intentions and made me think hard about what I was pursuing. To that end, I also wish to thank Prof. Pascal Tyrrell of the University of Toronto, in whose medical imaging lab I had volunteered in the summer of 2018 and become inspired to pursue applied machine learning.

I thank Sriram Sivaraman, a fellow Master of Statistics student at KU Leuven with whom I studied for many exams, and who kept me in the loop with things at the university while living abroad.

I thank both my parents for their continuous support and for having given me the freedom to pursue my interests since I was a child. Without their support, it would have been infinitely harder to maintain focus and creativity.

## Contents

**1** Introduction to Bayesian inference 1

**1.1** What is Bayesian statistics? 1

**1.2** The motivation behind Bayesian machine learning 3

**1.3** MacKay's two levels of Bayesian inference 4

**1.4** A brief introduction to neural networks 6

**1.5** Main contributions of this thesis 7

**2** Bayesian Learning of neural networks 9

**2.1** Why Bayesian when we have frequentist? 9

**2.2** The origins of Bayesian learning in neural networks 10

**2.3** Bayesian neural networks by MacKay 10

**2.4** Variational inference for Bayesian neural networks 11

**2.5** Bayesian uncertainties of neural networks 13

**2.5.1** Aleatoric and epistemic uncertainties 13

**2.5.2** An intuitive explanation of how the Bayesian approach gives uncertainties 15

**2.5.3** A practical approach using dropout 15

**2.5.4** A modified loss function 16

**2.6** The specifics of Bayesian training 17

**2.6.1** Backpropagation and *Bayes by backprop* 17

**2.6.2** The local reparameterization trick 18

**2.7** Experiments and applications 18

**2.7.1** A motivating example of Bayesian uncertainty using a neural network for function approximation

**2.7.2** A Bayesian neural network classifier that relies on uncertainty when making predictions 20

**3** Bayesian Deep Learning 22

**3.1** Convolutional Neural Networks 22

**3.1.1** The neuroscientific basis 22

**3.1.2** Motivation of convolutional neural networks 23

**3.2** The structure of convolutional neural networks 24

**3.2.1** Idea 24

**3.2.2** Design and hyperparameters 26

**3.2.3** The convolution and pooling operations as infinitely strong priors 27

**3.2.4** Bayesian formulation of a convolutional neural network 28

**3.3** MacKay's formulation adapted to convolutional neural networks 29

**3.3.1** The first level of inference 29

**3.3.2** The second level of inference 30

**3.4** Methods for training Bayesian convolutional neural networks 31

**3.4.1** The local reparameterization trick for convolutional layers 31

**3.4.2** Two convolutional operations for mean and variance 32

**3.4.3** An example of some assumptions and mechanics 32

**3.5** Distributional assumptions for the weights 33

**3.5.1** The assumption of normality 33

**3.5.2** A word about assumptions 34

**3.6** Experiments and applications 35

**3.6.1** A comparison between a Bayesian and frequentist CNN 35

**3.6.2** Modeling the Occam factor for a Bayesian CNN 37

**4** Bayesian optimization 40

**4.1** The idea 40

**4.1.1** The formulation 40

**4.1.2** History and previous work 41

**4.1.3** As a Bayesian model selection technique 42

**4.2** Gaussian processes 43

**4.2.1** As priors over functions 43

**4.2.2** Parameterization 43

**4.2.3** Covariance function 44

**4.2.4** Bayesian uncertainty modeling with Gaussian processes 45

**4.3** Acquisition functions 45

**4.3.1** Maximum probability of improvement 46

**4.3.2** Expected improvement 46

**4.3.3** GP upper/lower confidence bound 47

**4.4** Experiments and applications 47

**4.4.1** Bayesian optimization for a noisy function 47

**4.4.2** The importance of the acquisition function and GP parameterization in hyperparameter tuning 49

**4.4.3** Bayesian optimization as an informed alternative to random search 51

**4.5** Limitations of Bayesian optimization 52

**5** Final thoughts and conclusion 54

**5.1** Demonstrating Bayesian overfitting with an application to COVID-19 imaging data 54

**5.2** Main contributions 56

**5.3** Limitations 57

**5.4** For future study 57

**A** Appendix 59

**A.A** Formulas and derivations 59

**A.A.1.4** Neural networks as functions 59

**A.E** Experiments* 59

**A.E.2.7.1** 60

**A.E.2.7.2** 61

**A.E.3.6.1** 64

**A.E.3.6.2** 66

*Each experiment in this subsection corresponding to experiments from throughout the text

| | |
|---|---|
| **A.E.4.4.1** | 68 |
| **A.E.4.4.2** | 69 |
| **A.E.4.4.3** | 71 |
| **A.E.5.1** | 72 |
| **References** | 74 |

# Chapter 1

# Introduction to Bayesian Inference

## Abstract

In this chapter, a close look is taken at the intuition of Bayesian inference in machine learning, beginning with a general introduction and motivating examples of a simple nature. Bayesian methods are motivated as an essential addition to any analyst's toolkit, and the case for Bayesian learning dates back to David MacKay's 1992 PhD thesis [36] which formulated two levels of inference: the first for the model's parameterization, and the second for model selection in the case where several models are considered for a given task. These approaches are developed on a general level, with specific modifications to follow in the subsequent chapters.

## 1.1 What is Bayesian statistics?

Machine learning models rely on *learned parameters* to make useful predictions on data that is new to them. The choice of model (*neural networks* considered here) comes with its own structure (such as the architecture), and that structure is endowed with numerical quantities represented by $\boldsymbol{w}$ (the parameters, typically called 'weights' in neural network lingo, they may generally include additive *biases* which act as learnable constants). A simple example of parameters are the slope and intercept in traditional linear regression $y = mx + b$, so that $\boldsymbol{w} = (m, b)$. When estimated, these parameters mathematically interact with the data to produce an output $\hat{y}$, typically called a prediction. The goal in machine learning is to train a model with *optimal* weights $\hat{\boldsymbol{w}}^*$that will perform well enough on data it has never seen before.

More generally, frequentist inference finds a single *point-estimate* for $\hat{\boldsymbol{w}} = g\left(\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(n)}\right)$ as a function of the data $\left(\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(n)}\right)$ and predictions of the model rely on this being the so-called best parameterization that training could find. The frequentist assumes that this observed data is drawn from a random process, and that the estimate $\hat{\boldsymbol{w}}$ is a random variable since it is a function of this random data (the *true* $\boldsymbol{w}$ is unknown but *fixed*, and thus not random). The Bayesian perspective is the reverse: that the data is the only thing that is observed, and thus known- it *cannot* be described as random, while instead the underlying parameter $\boldsymbol{w}$ is the culprit for this randomness [19]. This way, the Bayesian places a *probability distribution* over $\boldsymbol{w}$ and before any data is observed by the model (i.e. passed through it), the analyst can incorporate a degree of belief about $\boldsymbol{w}$ in the form of its *prior* probability distribution $p(\boldsymbol{w})$. To make this more concrete, suppose there is initially reason to believe that the linear relationship between $x$ and $y$ is positive: then it is easy to incorporate a prior belief about the line's slope in the form of a probability distribution $p(m)$ such that mass is placed over positive support of $m$. If there is reason to believe that $\boldsymbol{w}$ hovers around a particular value, then the prior distribution will place more mass there.

A criticism of the Bayesian approach is that the analysis is sensitive to the *subjective* beliefs of the researcher, especially in black-box models like neural networks where the parameters have no clear interpretation other than to enable the model's arithmetic. A less confident Bayesian places *uninformative* priors [18] over $\boldsymbol{w}$: these prior distributions have high *entropy* (reflecting high uncertainty about $\boldsymbol{w}$ before data is seen) [19]. In fact, the use of prior distributions arguably circumvents the modelling issue that is parameter initialization [19] to some (often arbitrary) initial value $\boldsymbol{w}_0$ that training then abandons on its search for the optimal weights. With distributions instead of initial point-values however, a degree of both flexibility *and* uncertainty is encoded off the bat, the influence of which diminishes through training as well. Now, a closer look is taken at the mechanics of these ideas.

As data $\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(n)}$ is observed, the belief about network weights $\boldsymbol{w}$ can be updated through the data likelihood $p\left(\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(n)} \middle| \boldsymbol{w}\right)$ and the prior using Bayes' rule to produce the *posterior* probability distribution:

$$p(\boldsymbol{w}|\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(n)}) = \frac{p\left(\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(n)} \middle| \boldsymbol{w}\right) p(w)}{p(x^{(1)}, \dots, x^{(n)})} \propto p\left(\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(n)} \middle| \boldsymbol{w}\right) p(\boldsymbol{w}) \tag{1.1}$$

where the data term in the denominator (called *evidence*) can be disregarded when inference for $\boldsymbol{w}$ is performed as it does not depend on $\boldsymbol{w}$. Therefore, the data likelihood is the mechanism by which prior beliefs about $\boldsymbol{w}$ (however vague) are updated to reflect 'learning' from data $\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(n)}$. The main distinction between the Bayesian and the frequentist approach is that predictions are now made using a full distribution over $\boldsymbol{w}$, whereas the frequentist approach relies on a single point estimate $\hat{\boldsymbol{w}}$ as stated above (for example, maximum likelihood). Figure 1.1 below is a graphical representation of just how the prior distribution is transformed via the likelihood to yield the updated posterior.

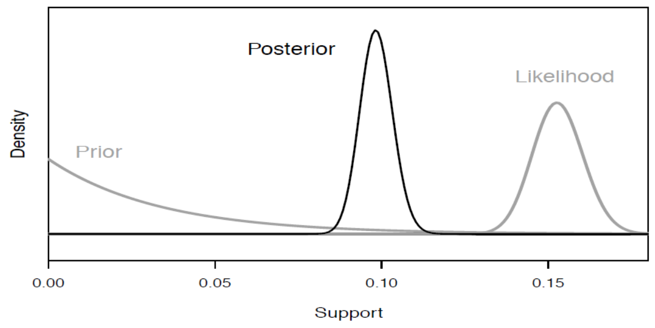


Figure 1.1: A high entropy, uninformed prior distribution is updated as data is observed through the rightmost distribution- the *likelihood*. The entropy is thus reduced, as more data reduces uncertainty about the parameter $\boldsymbol{w}$ in question. Notice that the posterior is a 'compromise' between the prior and likelihood. Image source: [18]

Finally, one may wonder just why obtaining the posterior distribution is desirable. Having updated beliefs about $\boldsymbol{w}$ in the form of a probability distribution allows one to *marginalize* over all possible values of this parameter that may produce the next observation, using what is called the *posterior predictive distribution:*

$$p\left(\boldsymbol{x}^{(n+1)}|\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(n)}\right) = \int p\left(\boldsymbol{x}^{(n+1)}|\boldsymbol{w}\right) p\left(\boldsymbol{w}|\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(n)}\right) d\boldsymbol{w} \tag{1.2}$$

where marginalization over all possible $\boldsymbol{w}$ allows each value with positive probability density to contribute to the prediction of the next example, where each contribution $p(\boldsymbol{x}^{(n+1)}|\boldsymbol{w})$ is weighted against the updated beliefs about that parameter in the form of the posterior [19].

## 1.2 The motivation behind Bayesian machine learning

Perhaps the growing interest in Bayesian inferential statistics can mainly be attributed to improvements in computing power [18] that have made it progressively more tractable. Perhaps it is a combination of this effect and the growing awareness of what a Bayesian approach can offer, as well as what it does not require. Among other things, the Bayesian modeling approach is suitable any time that one may desire to have an ability to update probability statements about quantities of interest as more data comes in, incorporate previous knowledge or beliefs into the study, and recognize that many population quantities actually may change over time (recall the frequentist supposition that $\boldsymbol{w}$ is fixed but unknown) [18]. In practice, it is often the case that almost entirely uninformative priors are placed over $\boldsymbol{w}$, in which case summary statements from the Bayesian approach are almost entirely explained by the likelihood (see above), and thus approximate the frequentist solution. Furthermore, the prior's influence is 'explained away' as more data comes in, which Figure 1.1 above conveys [18]. While this may make it seem that more data will just produce something close to the frequentist solution anyways, that is not at all an argument against the Bayesian approach. Under certain circumstances like a well-specified prior and likelihood, Goodfellow *et al.* [19] explain that Bayesian models have been shown to *generalize* (that is, how they perform on unseen data) much better when training data is limited, with the drawback that computational cost is much larger.

However, the computational cost that was once so worrying as to force Bayesians to stick to a very limited set of assumptions (such that the prior and posterior should be *conjugate*, that is, in the same family of distributions so as to create closed-form solutions that work out with pencil and paper) has recently dissipated. Traditional computational approaches relied on sampling from the posterior via Markov Chain Monte Carlo techniques [43], but even this became prohibitive for most realistic datasets, and practically entirely so for computer vision [23].

Recent approaches like in [3] explained in chapter 2 have made it less prohibitive. Bayesian methods now find application in computer vision [3][6][14][21][46], where for example neural networks trained in the 'Bayesian way' can incorporate distinct forms of uncertainty (see chapter 2, section 2.5) alongside accuracy and other traditional performance metrics [30][47]. Bayesian methods have thus been in use over the past decade in a very matter of fact way and have been shown to produce desirable results. The rest of this paper will explore these methods, starting with a brief history of Bayesian learning for *neural networks* in chapter 2, to *deep learning* in chapter 3, and then extended to their *optimization* in chapter 4. In this way, Bayesian inference over model parameters *and* hyperparameters is explored and motivated.

## 1.3 MacKay's two levels of Bayesian inference

Given the motivation of the preceding sections, it may be of particular interest to approach things in a more systematic way. To that end, David MacKay's 1992 PhD thesis [36] describes two levels of inference for any machine learning model: *model fitting* and *model comparison.* A closer look is taken in chapters 2 and 3 at how these apply to neural networks, but it is appropriate to explain these concepts in a more general and introductory setting here. Suppose that for a certain task, one may have a collection of models $H = \{H_i\}_{i=1}^{n}$each parameterized by weights $\boldsymbol{w}$ as we have seen above, and defined by two probability distributions: the prior on the weights of that model as $P(\boldsymbol{w}|H_i)$ and the likelihood $P(D|\boldsymbol{w}, H_i)$ of the data $D$ (a generalized notation over the data in (1.1)) for a given model and its weights. These are the ingredients of MacKay's two levels of inference, the first of which pertains to inferring the weights *model-wise* given the data:

Level 1. Using Bayes' rule as in (1.1), assuming that one of the models $H_*$ is true, the weights are inferred according to:

$$P(\boldsymbol{w}|D, H_*) = \frac{likelihood \times prior}{evidence} = \frac{P(D|\boldsymbol{w}, H_*) \times P(\boldsymbol{w}|H_*)}{P(D|H_*)} \tag{1.3}$$

where the evidence term in the denominator is dropped, as explained in 1.1. 'Inferring the weights' here essentially boils down to finding the most probable weights $\boldsymbol{w}_{MAP}$ where *MAP* denotes *maximum a posteriori,* the most probable weights after seeing the data for that specific model (i.e. the posterior mode of the left-hand side of (1.3)). MacKay then suggests summarizing each posterior by this most likely weight, along with an error bar that captures the dispersion of the distribution (in chapter 2, we will see concrete notions of dispersion and uncertainty). Note that in the case of completely uninformative priors, this *MAP* estimate becomes a maximum likelihood estimate and the Bayesian solution collapses onto the frequentist one (see Figure 1.1 and recall that the likelihood 'pulls' the prior towards the posterior, but if there is hardly anything to 'pull', the likelihood is pretty much already there). The predictive posterior distribution (1.2) requires inferring the posterior of (1.3), and section 2.3 will explore MacKay's application to neural networks [36].

Level 2. Now the evidence term $P(D|H_i)$ that was thus far dropped as irrelevant, becomes very much so, relevant. In this second level of inference, MacKay [36] wishes to infer *which* model among those considered in the collection $H$ is the most probable after seeing the data. Again, Bayes' rule gives the posterior probability of *each* model as:

$$P(H_i|D) \propto P(D|H_i)P(H_i) \tag{1.4}$$

The prior $P(H_i)$ encodes the subjective beliefs about what models are deemed appropriate before the data is seen (for example, the convolutional neural networks of chapter 3 are models well-suited for image data, and preferences for their structure can be encoded in this prior). Therefore, this second level of inference allows us to rank models by evaluating the evidence.

Given these two levels of inference, one may now wonder how to go about inferring the model in a *careful* way. It is surely a subtle thing, but nothing stops one from selecting the model in (1.4) that maximizes the evidence $P(D|H_i)$, which thereby corresponds precisely to maximum likelihood. The problem with this approach is that the most *overparameterized* model (that is, the most complex one) will be suggested. MacKay therefore suggests the inclusion of an *Occam factor*, a multiplicative quantity that penalizes the evidence term to encourage preference for simpler models (named after Occam's razor). The evidence term can therefore be expressed as:

$$P(D|H_i) \simeq P(D|\boldsymbol{w}_{MAP}, H_i)P(\boldsymbol{w}_{MAP}|H_i)\Delta\boldsymbol{w} \tag{1.5}$$

where the first term is the height of the posterior, and the product of the second and third term is the Occam factor, where $\Delta\boldsymbol{w}$ acts as a measure of the width of the posterior distribution over the weights, $P(\boldsymbol{w}|D, H_i)$. Note that (1.5) is MacKay's approximation of the true evidence, given by the intractable integral:

$$P(D|H_i) = \int P(D|\boldsymbol{w}, H_i)P(\boldsymbol{w}|H_i)d\boldsymbol{w} \tag{1.6}$$

so that this true integral is essentially approximated in a rectangular fashion as a product of the posterior's height and its width, shown below in Figure 1.2. The whole idea of the Occam factor is to quantify by how much the prior's entropy (which MacKay conceptualizes as the *accessible volume* of the distribution) is reduced as more data comes in. In the event that the prior is purely uninformative (thus uniform), and given that no data has yet been observed, the likelihood of the *MP* estimate is $P(\boldsymbol{w}_{MAP}|H_i) = \frac{1}{\Delta^0\boldsymbol{w}}$ since all weights are uniformly likely. In this case, Occam's factor takes the intuitive and easy-to-understand form $\frac{\Delta\boldsymbol{w}}{\Delta^0\boldsymbol{w}} \leq 1$, measuring the reduction in width (or entropy) after training, and is also sensitive to over-parameterized models with large deviation in weights, $\Delta^0\boldsymbol{w}$.

The Occam factor according to MacKay measures complexity, and favours models that minimize complexity and data misfit, through a trade-off that maximizes the evidence in (1.5) [36]. This notion is generalized for priors that are not uniform in chapter 2, and these scrutinous methods that penalize complexity while favouring good fits to the data are seen again throughout that chapter (one such example we will see is *variational inference*, a method to learn the weight posterior via a minimization problem that embodies this trade-off by automatically incorporating preference for simpler models). In the second experiment of chapter 3, we will see how we may mimic and utilize this Occam factor for a fundamentally more complicated model.

In concluding this section, it is worth mentioning that the weights $\boldsymbol{w}$ can be conceptualized as *latent* variables, and these networks as *latent variable models*. Thinking of them in this way allows for interesting interpretations, namely in the context of *sensitivity analysis* (discussed again in chapter 3). While the learned network may be sensitive to the choice of specification (such as the form of prior distribution), assessing network performance amounts to assessing this learned latent distribution over the weights (discussed also at greater length in chapter 3). With this view, distributions of continuous latent variables (like $\boldsymbol{w}$) can be approximated by discrete distributions (although this is mentioned only in passing). We will see other examples throughout this thesis of where being able to conceptualize something in more than one way will provide richer (and relevant) insight.

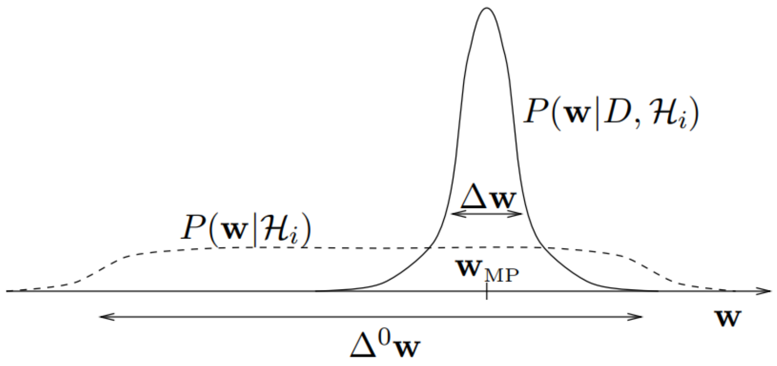

$P(\mathbf{w}|D, \mathcal{H}_i)$
$\Delta\mathbf{w}$
$P(\mathbf{w}|\mathcal{H}_i)$
$\mathbf{w}_{\mathrm{MP}}$
$\mathbf{w}$
$\Delta^0\mathbf{w}$

Figure 1.2: During training, the weight posterior for a given model $H_i$ narrows down as entropy is reduced through observing more data. That is reflected in the reduction in the width of the prior distribution, captured by $\Delta^0\boldsymbol{w}$ to the width of the posterior, $\Delta\boldsymbol{w}$. Image source: [36].

## 1.4 A brief introduction to neural networks

Until now, the discussion was rather general to the choice of model. However, as the rest of this paper is dedicated to specifically neural networks, it is worth explaining their structure (more details follow in chapters 2 and 3) in a concise mathematical way from the outset. Neural networks are functions of input data, which by their design ‘channel’ the data through layers, applying function transformations along the way (see Appendix A.1.4), and thereby learning a mapping from input $\boldsymbol{x}$ to output $y$ [19]. For a multidimensional input $\boldsymbol{x}$ (examples of which we will see in the upcoming chapters; among them are images of grid-like structure), the neural network with $L$ layers will pass $\boldsymbol{x}$ through in the following way:

$$f(\boldsymbol{x}) = f_L(\dots f_2(f_1(\boldsymbol{x}; \boldsymbol{w}_1); \boldsymbol{w}_2) \dots), \boldsymbol{w}_L) \qquad (1.7)$$

where each layer is parameterized by weights $\boldsymbol{w}_i$ (including layer-wise additive biases). As mentioned in section 1.1, the goal is to learn the optimal weights of the network as a whole that will best represent the mapping $\boldsymbol{x} \mapsto \boldsymbol{y}$ while generalizing well to unseen data (in machine learning terms, reduce *overfitting*; more on this in chapter 2). Recall that finding optimal *point-estimate* weights is the frequentist approach, and we are interested in the Bayesian treatment of these weights; this boils down to MacKay’s first level of inference from equation (1.3). Therefore, as an abstraction to the above, we may say that the neural network as a mapping is learned by inferring the posterior over the weights, so that we may write:

$$f(\boldsymbol{x}) = f_L(\dots f_2(f_1(\boldsymbol{x}; \boldsymbol{w}_1); \boldsymbol{w}_2) \dots), \boldsymbol{w}_L) | \boldsymbol{w}_i \sim P(\boldsymbol{w}|D, H_*) \qquad (1.8)$$

to emphasize that layer weights are random variables with a certain learned *global* posterior distribution (recall that there is *one* posterior, not $L$ layer-wise posteriors). With the concepts introduced in this chapter, we are now ready to move on to higher-level concepts and closely examine all the moving parts.

## 1.5 Main contributions of this thesis

While still too early to dive into what we will soon dedicate entire chapters to, here is what makes up the rest of this work (see also Figure 1.3 below as a pictorial glance). Having by now been gently introduced to the Bayesian framework, not everything should seem entirely foreign. While we may not yet have seen these concepts, it is good to be exposed to them now, and re-visit them in the conclusion with a richer background.

- First and foremost, this work is meant as an all-round introduction and explanation of Bayesian methodologies for defining, training, and testing neural networks (both traditional feedforward, fully connected ones and the *convolutional* neural networks of chapter 3 that are common in deep learning).
- In this work, we aim to extend the two levels of Bayesian inference of David MacKay that we saw in section 1.3 to these convolutional neural networks in chapter 3.
- In the first level, we explore recent methods like *variational inference* [47] for inferring the network posterior over the weights (equation (1.3)). From this learned posterior, we can derive the predictive posterior for weights in line with equation (1.2), which can be accompanied by predictive Bayesian uncertainties. We also show how these uncertainties may be approximated in the case of frequentist networks, so that performance metrics may be accompanied by a margin of error.
- In the second level, we suggest and implement an approximation to MacKay's Occam factor for convolutional neural networks and compare it to more commonly used metrics. In chapter 4, we frame the popular *Bayesian optimization* algorithm typically used for automated hyperparameter tuning as a model-selection technique, and thus suggest it as a second-level inferential tool.
- We close with an application of building a binary classifier network on publicly available COVID-19 CT scans of patients' chests, in order to demonstrate overfitting in a custom-defined Bayesian convolutional neural network.

With all of this in mind going forward, Figure 1.3 below acts as a 'mind map' of this thesis and demonstrates how the concepts are all inter-related. Note that any terms in Figure 1.3 not introduced up until this point will be mentioned and clarified in subsequent chapters, so that later reference to this mind map will become increasingly well-warranted should the reader lose track of the 'bigger picture'.

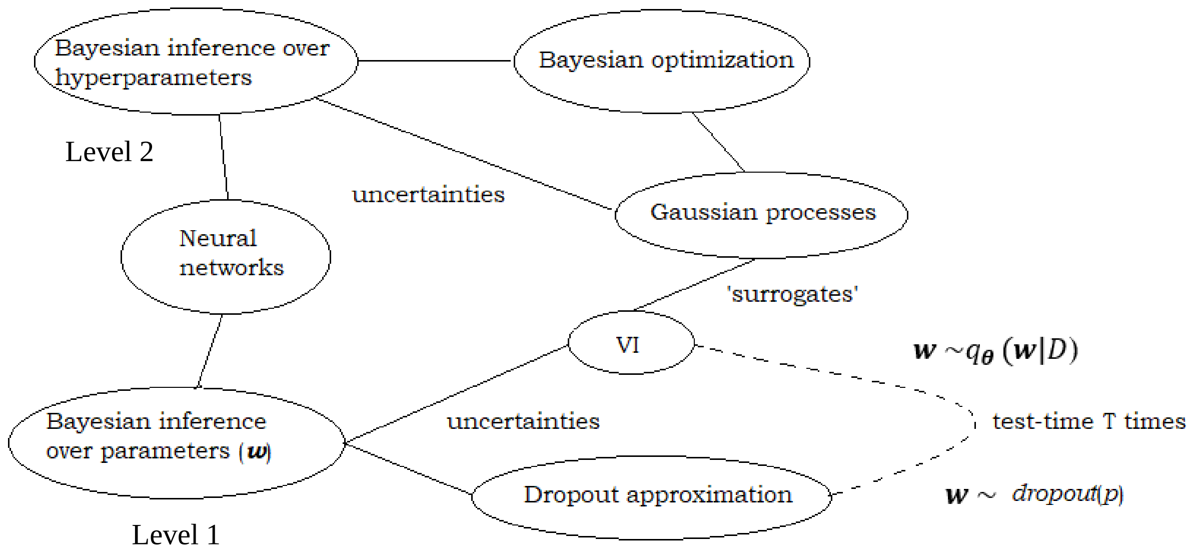

Bayesian inference over hyperparameters
Bayesian optimization
Level 2
uncertainties
Gaussian processes
Neural networks
'surrogates'
VI
$\boldsymbol{w} \sim q_{\boldsymbol{\theta}}(\boldsymbol{w}|D)$
Bayesian inference over parameters ($\boldsymbol{w}$)
uncertainties
test-time T times
Dropout approximation
$\boldsymbol{w} \sim$ *dropout*($p$)
Level 1

Figure 1.3: A high-level mind-map of how these concepts are all inter-connected, with the main focus being inference in two stages: over model parameters (level 1 of MacKay's inference) and over model hyperparameters, which effectively doubles as model selection (level 2 of MacKay's inference).

# Chapter 2

# Bayesian learning of neural networks

**Abstract**

This chapter extends on the principles of the previous one in the context of neural networks. A close look is taken at David MacKay's benchmark formulation for these models, following his 1994 work titled "Bayesian Neural Networks and Density Networks" [37]. At that time, learning the posterior distribution over the weights was still a major computational issue; Radford Neal seems to have chipped the ice a year later, with his PhD thesis titled "Bayesian Learning for Neural Networks" [43], but only slightly and mostly in theory. We will see in this chapter a more recent method called *variational inference* which has nice intuitive properties and was made tractable in 2011 by Alex Graves [21], along with model and data-specific uncertainties that play a crucial role in the decision process [30]. The uncertainties are also demonstrated in a practical way, according to the 2016 work of Gal and Ghahramani [15]. The chapter concludes with a collection of experiments that are aimed at motivating these methods in different practical contexts such as AI in medical imaging.

## 2.1 Why Bayesian when we have frequentist?

Perhaps one of the most important works on Bayesian learning of neural networks is that of David MacKay [37], whose 1994 paper *Bayesian Neural Networks and Density Networks* introduced the idea of placing a probability distribution over the unknown inputs of a network with known target outputs. MacKay develops the Bayesian framework for the multilayer perceptron, arguing that the Bayesian application of a prior density over model weights is analogous to traditional regularization with the added effect of a probability distribution over that weight space, as opposed to point estimates. One favorable consequence, MacKay argues, is that quality of predictions is improved while taking uncertainty into account.

To that end, Heek and Kalchbrenner [22] point out that unique advantages derive from sampling model instances (that is, unique networks defined by the sampled weights from the learnt weight posteriors; more on this in 2.5.2). Among other things, the Bayesian posterior over network weights allows one to aggregate several models (like an ensemble) simply by re-sampling weights, and this can give robust uncertainty estimates over the network's predictions [22]. They further espouse the method for its ability to sample, rather than optimize. The idea here is that defining a network by sampled weights rather than optimal weights (e.g. that minimize a *loss*) makes the network less prone to overfitting. It can also help generalization under more training, in the generally typical case that a discrepancy between training and test data distributions is present. Blundell *et al.* further demonstrate how such networks can have improved generalization performance from their learnt weight uncertainties [3], which are discussed in more detail in section 2.5.

## 2.2 The origins of Bayesian learning in neural networks

Having been around a long time, the study of Bayesian learning applied to neural network models perhaps finds among its major first contributions in the aforementioned work of MacKay [37], although that work did not consider deep models like convolutional neural networks (nor rich data structures seen in image recognition for example). In this work, MacKay not only develops a Bayesian framework for inferring the network's weights from the data, but also for inferring the network's hyperparameters (such as the network's architecture, studied in chapter 4). When it comes to predicting class membership, his framework marginalizes over the model weights *and* the model hyperparameters. Monte Carlo methods are suggested for approximating intractable integrals in the posterior expression for the model weights given the data, hyperparameters, and hypothesis. A year later, in 1995, Radford Neal's PhD thesis [43] explored the use of *hybrid* Monte Carlo methods as an extension of MacKay's work and found that good results are found on small datasets with large networks – a result that is very difficult to apply to large datasets (especially image datasets), and with deep and complex models like convolutional neural networks that are discussed in chapter 3.

As MacKay mentions in [36], optimization-based techniques were also tried in the context of Bayesian neural network learning, with his own previous work using MAP (maximum a-posteriori) *point* estimates as modes of the learned weight posterior distributions as in (1.1), an approach dating back to as early as 1991 [6]. In 1993, *variational* methods (see section 2.4) for approximating the posterior were already hinted at in the work of Hinton and van Camp [23]. MacKay's construction of a full covariance Gaussian approximation to the posterior around the single locally optimal point-estimate in weight space is simplified by Hinton and van Camp's use of a simpler Gaussian approximation that is updated during learning (and with strictly diagonal covariance matrix). This in effect plays the role of a variational posterior distribution, whose goal is to continually be updated during learning such that its parameterization minimizes the optimality condition (2.3) below. This variational posterior distribution in effect acts a *surrogate* of the true *unknown* posterior, a line of reasoning that is seen again in chapter 4 where a black box function is optimized by inferring a likely set of approximating functions which is continually updated with new observations.

## 2.3 Bayesian neural networks by MacKay

Supposing that the network in question is parameterized by some real-valued weights $\boldsymbol{w} \in W$, the traditional view of learning evolves $\boldsymbol{w}$ during training from some initial $\boldsymbol{w}_0$ towards an optimum $\boldsymbol{w}^*$so as to minimize a loss (the nature of which depends on the task). A regularization term may also be included to combat overfitting. This contrasts with the Bayesian view in that instead, a posterior distribution (or *ensemble,* in MacKay's paper [37]) over $\boldsymbol{w}$ is obtained as a product of likelihood and prior over $\boldsymbol{w}$, as we saw in chapter 1. Recall that the likelihood is the mechanism by which the posterior is updated as data comes in [18]. The formulation according to MacKay

[36] for the Bayesian inference problem starts by conditioning on an observed dataset $D = \{\boldsymbol{x}^{(n)}, y^{(n)}\}_{n=1}^{N}$with targets $y^{(n)}$, assumed model $H$ and its hyperparameters $\alpha$. As in (1.1), conditional inference over $\boldsymbol{w}$ can be written using Bayes' rule:

$$P(\boldsymbol{w}|D, \alpha, H) = \frac{P(D|\boldsymbol{w}, H)P(\boldsymbol{w}|\alpha, H)}{P(D|\alpha, H)} \propto p(D|\boldsymbol{w}, H)p(\boldsymbol{w}|\alpha, H) \tag{2.1}$$

Note that although this approach appears general, we are strictly considering models $H$ to be neural networks, and therefore $\alpha$ to be the set of appropriate hyperparameters like network architecture (number of hidden layers, layer sizes, choice of activation functions). In practice, it is even possible to consider aspects unrelated to the network, like choice of optimizer, learning rate, etc. [19]; we will see one such way in chapter 4. Expression (2.1) is in line with MacKay's first level of inference we saw in chapter 1, with one small extension for making predictions using the derived weight posterior in (2.1).

Predictions for a new data point $(\boldsymbol{x}^{(n+1)}, y^{(n+1)})$ are made according to the predictive distribution as in (1.2), by also marginalizing over the weights and the model's hyperparameters:

$$P\left(y^{(n+1)}\middle|\boldsymbol{x}^{(n+1)}, D, H\right) = \int P\left(y^{(n+1)}\middle|\boldsymbol{w}, H\right)P(\boldsymbol{w}|D, \alpha, H)P(\alpha|D, H)d\boldsymbol{w}d\alpha \tag{2.2}$$

Notice in (2.2) the presence of the posterior $P(\boldsymbol{w}|D, \alpha, H)$ which is explicitly defined in (2.1). MacKay [36] mentions two approaches for estimating this quantity: Gaussian distributions centered at the optimal $\boldsymbol{w}^*$, and sampling of the posterior by Monte Carlo methods. For complex models like the convolutional neural networks discussed in chapter 3, this approach quickly becomes computationally prohibitive, so alternative methods like *variational inference* have been proposed [21][23]. The second level of inference is model selection, and to that end, one could in practice compare the performances of these derived predictive posteriors through something like leave-one-out cross-validation on neural networks with different architectures [19]. This in effect also measures overfitting, encouraging the consideration of simpler models. One problem is that measuring Occam's factor becomes challenging for the real-world complex datasets and networks needed (like in chapter 3), but conceptually it is an excellent tool that reminds the practitioner to favour simpler models.

## 2.4 Variational inference for Bayesian neural networks

As discussed above, intractability is a problem that plagues the Bayesian's quest for the posterior, though variational inference is a relatively recent approach designed to overcome this. Its intuitive nature is also worthy of mention, as will be shortly shown. Assume a more general training set is given as $D = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$, and recall MacKay's first level of inference. The Bayesian neural network will be defined in terms of a prior distribution $p(\boldsymbol{w})$ over the weights $\boldsymbol{w}$ of the network, as well as a likelihood $p(D|\boldsymbol{w})$, where we supress the model $H$ and hyperparameters $\alpha$ of MacKay's formulation from this chapter, without loss of generality. Variational inference then attempts to fit an approximate posterior $q(\boldsymbol{w}) \approx p(\boldsymbol{w}|D)$ over the network's weights, which is easier to sample from and acts as a form of *surrogate* (a concept seen again in Bayesian optimization, the focus of chapter 4).

Blundell et al [3] find a variational approximation of this posterior distribution by finding a parameterization $\boldsymbol{\theta}$ that governs a distribution on the weights $q_{\boldsymbol{\theta}}(\boldsymbol{w})$, such that this $\boldsymbol{\theta}$ is optimal by minimizing the Kullback-Leibler (KL) divergence between this $q$ and the *true* Bayesian posterior on the weights, $p(\boldsymbol{w}|D)$. Intuitively, the KL-divergence is a measure of similarity between two distributions (a value of 0 implies that the distributions in question are identical). Therefore, the optimal parameterization $\boldsymbol{\theta^*}$ of the approximating *variational* distribution $q_{\boldsymbol{\theta}}(\boldsymbol{w})$ is:

$$\begin{aligned}\boldsymbol{\theta}^* &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \boldsymbol{KL}[q_{\boldsymbol{\theta}}(\boldsymbol{w})\,||p(\boldsymbol{w}|D)] = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \int q_{\boldsymbol{\theta}}(\boldsymbol{w})\, log \frac{q_{\boldsymbol{\theta}}(\boldsymbol{w})}{P(\boldsymbol{w})p(D|\boldsymbol{w})} d\boldsymbol{w} \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \boldsymbol{KL}\big[[q_{\boldsymbol{\theta}}(\boldsymbol{w})||P(\boldsymbol{w})]\big] - E_{q_{\boldsymbol{\theta}}(\boldsymbol{w}),}[logP(D|\boldsymbol{w})] \end{aligned} \tag{2.3}$$

The final expression minimizes a cost function, which embodies a trade-off between satisfying the complexity of the data $D,$ and satisfying the simplicity prior $P(\boldsymbol{w})$ over the network weights [3]. The first term depends on the definition of the prior and is called the complexity cost, and the latter on the data and is called the likelihood cost. This 'hybrid' cost function explicitly demonstrates the regularization effect that comes for free with Bayesian neural networks, as it favours simplicity in line with Occam's principle that was discussed in chapter 1.

This expression is still intractable however, so minimizing the KL divergence is typically done by (analogously) maximizing the *evidence lower bound* (ELBO) [47]:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, ELBO(\boldsymbol{\theta}) = E_{q_{\boldsymbol{\theta}}(\boldsymbol{w}),}[logP(D,\boldsymbol{w})] - E_{q_{\boldsymbol{\theta}}(\boldsymbol{w}),}[logq_{\boldsymbol{\theta}}(\boldsymbol{w})] \tag{2.4}$$

where the first term represents *energy* (which encourages $q$ to focus probability mass where appropriate) while the second term represents *entropy* of $q$- minimizing this will encourage $q$ to spread its mass and avoid concentrating on any one area [47], equivalently a form of built-in regularization. Gradient-based optimization over $\boldsymbol{\theta}$ suddenly becomes possible, turning the problem of integration into one of optimization (with *Adam*, for example).

Finally, each time a prediction is made on unseen data, weights are sampled from this variational posterior to define a network with unique weight values. By making *T* predictions on a new test example, this defines *T* different networks, in essence acting as an *ensemble*. This not only makes the Bayesian approach more robust by 'averaging' over its predictions, but naturally incorporates uncertainty (discussed below). Since it is possible to sample multiple networks (i.e. by sampling weights) for predicting on the same input, the extent of heterogeneity of these predictions expresses uncertainty. Finally, Figure 2.1 below depicts variational inference in action.

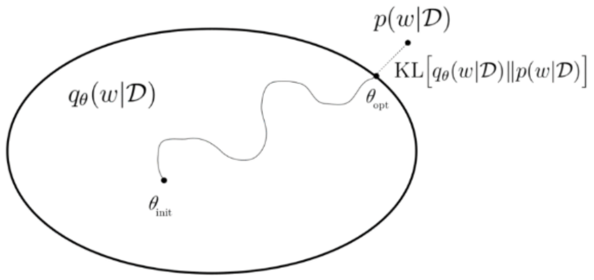

$p(w|\mathcal{D})$
$\mathrm{KL}\left[q_\theta(w|\mathcal{D})\|p(w|\mathcal{D})\right]$
$q_\theta(w|\mathcal{D})$
$\theta_{\text{opt}}$
$\theta_{\text{init}}$

Figure 2.1: The initialized parameterization $\boldsymbol{\theta}_{init}$ of the variational posterior $q$ almost certainly causes it to be dissimilar (in terms of KL-divergence) from the desired posterior *p*. As $\boldsymbol{\theta}$ is learnable during training, a path is drawn out as *q* moves "towards" *p*. The boundary represents the limits to approximation- $\boldsymbol{\theta}_{opt}$ minimizes the KL-divergence but does not necessarily make it 0, in which case the boundary would extend to $p$ itself. Image source: Graves [21].

Recall that the foundational idea of variational inference is that $p$ is approximated by $q$ whose shape and behavior is governed by $\boldsymbol{\theta}$, which is itself updated during training based on weights that are sampled from this very $q$. For example, in the case of a normal variational posterior, we might have $\boldsymbol{\theta} = (\boldsymbol{\mu},\ \sigma^{\mathbf{2}}\boldsymbol{I})$ with the sampled weights $\boldsymbol{w}$ governed by this parameterization. However, this parameterization $\boldsymbol{\theta}$ will begin to resemble that of $p$ more and more along the path towards $p$ in Figure 2.1, and movement along the path is likewise dependent on the sampled weights. Section 2.6.1 shows how the minimization problem (2.3) can be approximated in practice.

Both Graves [21] and Blundell *et al*. [3] specify this variational posterior as a product of multivariate normal distributions, making the problem *scalable* but doubling the number of parameters (having to learn a mean and variance, as opposed to a single point estimate for each of the $\boldsymbol{w}$, an issue we will revisit in 3.4.2, the solution to which is provided by Shridhar *et al.* [47]). Other assumptions are possible, as we will discuss in section 3.5.

## 2.5 Bayesian uncertainties of neural networks

### 2.5.1 Aleatoric and epistemic uncertainties

Another benefit of Bayesian learning is that uncertainty of a learned network can be decomposed into *aleatoric* and *epistemic* uncertainties [30]. The former acts as a measure for the inherent noise of observations, which cannot be reduced as more data comes in. The latter is a measure of model uncertainty and *can* be reduced with more data. As explained in [30], the aleatoric uncertainty can be further decomposed into *homoscedastic* and *heteroscedastic* uncertainties, where the latter measure depends on the nature of the inputs while the former is constant over different inputs. Capturing the heteroscedastic uncertainty is particularly important so that the model does not output overly confident decisions [47] and was part of the motivation of the aforementioned works of MacKay and Neal.

Denoting (2.2) in a more simplified fashion by suppressing inference over hyperparameters and hypotheses and denoting outputs by $y$, the predictive distribution takes the form:

$$p(y^{new}|\boldsymbol{x}^{new}, D) = \int p(y^{new}|\boldsymbol{w})p(\boldsymbol{w}|D)\, d\boldsymbol{w} \tag{2.5}$$

For classification, this predictive distribution is taken to be categorical as in [47], while the prior over the data can be taken to be normal (with varying subjective degrees of entropy governed by its variance). Shridhar *et al.* [47] show that the expectation over the predictive distribution (with corresponding unbiased estimator obtained by sampling $T$ times from the variational posterior, and in effect defining an ensemble of $T$ models as explained in section 2.4) can be given by:

$$\boldsymbol{E}_q[p(y^{new}|\boldsymbol{x}^{new}, D)] = \int p(y^{new}|\boldsymbol{w})q_{\boldsymbol{\theta}}(\boldsymbol{w})\, d\boldsymbol{w} \approx \frac{1}{T}\sum_{i=1}^{T} p_{\boldsymbol{w}_i}(y^{new}|\boldsymbol{x}^{new}) \tag{2.6}$$

Arriving finally at the expression of interest, the *predictive variance* to model uncertainty:

$$\begin{aligned} Var_q[p(y^{new}|\boldsymbol{x}^{new})] &= \boldsymbol{E}_q[y^{new}(y^{new})^T] - \boldsymbol{E}_q[y^{new}]\boldsymbol{E}_q[y^{new}]^T \\ &= \int [diag(\boldsymbol{E}_p[y^{new}]) - \boldsymbol{E}_p[y^{new}]\boldsymbol{E}_p[y^{new}]^T]q_{\boldsymbol{\theta}}(\boldsymbol{w})d\boldsymbol{w} \\ &+ \int (\boldsymbol{E}_p[y^{new}] - \boldsymbol{E}_q[y^{new}])\,(\boldsymbol{E}_p[y^{new}] - \boldsymbol{E}_q[y^{new}])^T q_{\boldsymbol{\theta}}(\boldsymbol{w})d\boldsymbol{w} \end{aligned} \tag{2.7}$$

where the first integral represents *aleatoric* uncertainty, while the second represents *epistemic*. Several unbiased estimators for these quantities have been suggested, which rely on sampled weights of the variational posterior $q_{\boldsymbol{\theta}}(\boldsymbol{w})$. One particular reason why this is useful is because the modeler can gauge whether the quality of the data is to blame (high *aleatoric* uncertainty) or if the model is to blame (high *epistemic* uncertainty) [47]. This is very likely what MacKay would have liked to see [36] and makes for an interesting extension to his Occam factor for conceptualizing reduction in entropy over weights from training. Heek and Kalchbrenner also propose this decomposition for specific domains such as medical imaging where uncertainty estimates are crucial, as incorrect predictions like false negatives in cancer screening can have catastrophic consequences [22].

Finally, besides being quantified, a *qualitative* description in terms of aleatoric and epistemic uncertainties allows for an (admittedly) primitive notion of self-awareness. That is to say, the model will exhibit higher uncertainty when it comes to examples it does not so easily recognize [30]. These uncertainties arise from unanticipated discrepancies between the training and the test data distributions, yet Kendall and Gal showed that modeling both uncertainties improves prediction performance [30] as shown in experiment 2.7.2. When it comes to say, medical imaging, professionals can step in and revise the more uncertain predictions so as to diminish the probability of making a false diagnostic.

### 2.5.2 An intuitive explanation of how the Bayesian approach gives uncertainties

When it comes to making predictions on a new example, the Bayesian neural network relies on the posterior predictive distribution given in equation (2.2). Recall however, that the predictive distribution's reliance on the true data posterior can be made tractable by solving maximization problem (2.4), finding the best possible parameterization for the more tractable (and whose form is generally known or assumed) variational posterior, $q_{\boldsymbol{\theta}}(\boldsymbol{w})$. Assuming that the analyst decides to take $T$ samples of weights (and thus obtain $T$ networks) to consider at test-time, each network $BNN_i$ is governed by weights $\boldsymbol{w}_i$. Each of these networks in turn ascribes a predictive posterior probability to the new example $\boldsymbol{x}$ in the form of a vector, each element of which represents the probability of class-membership of $\boldsymbol{x}$ for class $C_j$ (in the case of classification) as $P_{BNN_i}(C_j|\boldsymbol{x})$. The network then uses its activation layer to make the final prediction and can represent the model-inherent epistemic uncertainty of its decision by looking at the heterogeneity between outputs of each of the $BNN_i$ networks. Along these same lines, *dropout* can be used at test-time [15] to achieve the same goal and is discussed in further detail below.

Therefore, the epistemic uncertainty captures the uncertainty in the prediction due to the different models that vie for it, each of which is obtained by sampling weights from the variational posterior (other approaches may sample the posterior using MCMC as in [43]), which effectively act as weight perturbations that Shridhar *et al.* note drives this form of uncertainty [47]. The predictive variance in (2.7) is the mechanism by which these uncertainties are derived and allows modeling of uncertainties over specific test inputs which is shown in experiment 2.7.1.

### 2.5.3 A practical approach using dropout

As an example, consider how simpler statistical models like linear regression are generally reported with standard errors on estimated parameters, and how this helps give a richer context to predictions that the model makes. Such an extension to neural networks is typically impractical since there can easily be millions of parameters. In 2016 however, Gal and Ghahramani [15] showed that when viewing the network as an approximation to a *Gaussian process* (explored in chapter 4), these uncertainty estimates can be estimated by training a network with *dropout* [51], and then testing it with dropout as well. The use of dropout effectively allows Monte Carlo samples to be drawn from the posterior, in turn used to approximate this true posterior [15] by summarizing the sample.

Dropout was a method originally conceived of as a means of regularization: a mechanism shown in Figure 2.2 below, by which neurons (along with their connections) are dropped with some probability $p$ during training, thereby preventing neurons from excessively co-adapting [51]. This approach essentially defines a new network architecture each time a training example is fed forward (stochastic forward passes, called *Monte Carlo dropout* [30]), thereby training an ensemble of models and making the overall network more robust. However, by keeping dropout on during test time, predictions are no longer deterministic functions of the trained weights and selected hyperparameters. Predicting on the same input $T$ times with different networks governed by dropout rate $p$ is analogous in idea to sampling weights $\boldsymbol{w} \sim q_{\boldsymbol{\theta}}(\boldsymbol{w}|D)$ $T$ times and quantifying the heterogeneity in the predictions due to the $T$ sampled predictive networks. Such an approach allows us to report performance metrics along with confidence intervals (or *error bars* in MacKay's case), something seen in the experiments in chapter 4.

What is more, Gal *et al.* [16] also show how the hyperparameter $p$ can be learned for the task at hand, and this is used in experiment 2.7.1.

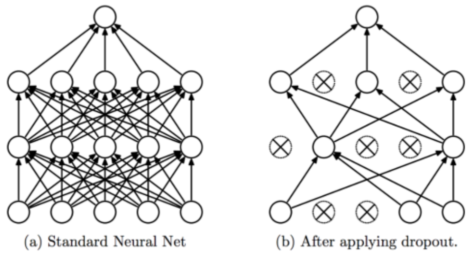


Figure 2.2: Dropout in a fully connected neural network with two hidden layers. Image source: [51].

Building on the intuition of the close connection between sampling and uncertainty as explained in this section and the previous one, a specific thought experiment for image classification is shown here. Assume a binary image classification problem (as is quite typical in medical imaging in the form of treatment vs. control, seen in chapter 5 for COVID-19 and non-COVID-19 CT scans), with two labels $l_1$ and $l_2$. Then, when faced with a new example (like an unseen MRI slice of a brain), the analyst can sample the variational posterior $T$ (or cast dropout $T$ times) times to obtain an ensemble of $T$ models vying for a prediction. The predictive variances $Var_q[p(y^{new} = l_1|\boldsymbol{x}^{new}, D)]$ and $Var_q[p(y^{new} = l_2|\boldsymbol{x}^{new}, D)]$ will measure the model's uncertainties (both epistemic and aleatoric, as shown above in equation (2.7)) for *each* of the labels by using unbiased estimates discussed above. This is shown in practice in experiments 2.7.1 and 2.7.2, where uncertainties are example-dependent (for regression) and label-specific (for classification) and carries over into the deep learning experiments of the next chapter.

### 2.5.4 A modified loss function

Since epistemic uncertainty is model-dependent, capturing this uncertainty in a neural network amounts to inferring posterior distributions over weights as in 2.3, with the variational approach from 2.4. Inducing model-dependent prediction uncertainty is done by marginalising over the (approximate) weight posterior distribution [30]. A full mathematical development is shown in this paper.

In the case of regression, the loss function can be modified to be *heteroscedastic* as in [30] and relies on a Gaussian likelihood model for the aleatoric uncertainty. This has the form:

$$\frac{1}{N}\sum_{i=1}^{N}\frac{1}{2\hat{\sigma}_i^2}(\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i)^2 + \frac{1}{2}log\hat{\sigma}_i^2 \qquad (2.8)$$

Here is the intuition. Minimization of this loss requires that $1/\hat{\sigma}_i^2$ in the first term be kept small when the prediction is far from the ground truth, encouraging the model to indicate high estimated uncertainty $\hat{\sigma}_i^2$. Additionally, it is discouraged from being too low since this would magnify the effect of points with high residual error. The inclusion of the second term discourages the model from indicating high uncertainty *everywhere* (and ignoring data). Note that this approach does not capture *epistemic* uncertainty, as that is inherent to the model itself.

In regression tasks, the aleatoric uncertainty can thus be interpreted as learned loss *attenuation* (meaning to reduce the effect of) [30], since such a loss function will attenuate the impact of unusual observations through its added ability to quantify uncertainty. This loss function is applied in experiment 2.7.1 where both measures of uncertainty are also combined, and its extension to classification is discussed in chapter 3 on deep learning.

## 2.6 The specifics of Bayesian training

### 2.6.1 Backpropagation and *Bayes by backprop*

When training a *feedforward* neural network, information is propagated from the input $\boldsymbol{x}$ to the output $\hat{\boldsymbol{y}}$ via the network architecture and parameterization $\boldsymbol{w}$ [19] as was shown in section 1.4. This *forward propagation* of information through the network culminates in the computation of a scalar cost (which is a function of the network parameterization) $J(\boldsymbol{w})$, through the discrepancy between the fitted values $\hat{\boldsymbol{y}}$ and the true target labels $\boldsymbol{y}$. In 1986, Rumelhart et al [46] suggested flowing the information from this cost backward through the network in order to compute the gradient of the loss in terms of the weights, $\boldsymbol{w}$ [19]. Finally, this two-step process updates the network's weights by use of these gradients, in an iterative process during the training phase that seeks the optimal $\boldsymbol{w}$ to minimize the cost $J(\boldsymbol{w})$.

Notice that this optimization problem gives a point-estimate solution for the optimal weights. But recall that in the Bayesian setting, data is fixed, and parameters are random. We need to fit distributions as stated earlier in the chapter, and an optimization algorithm for that is *Bayes by Backprop* [3].

Cost function (2.3) from section 2.4 frames the problem of finding the variational posterior parameterization $\boldsymbol{\theta*}$ such that the KL-divergence between the true data posterior and this variational posterior is minimized. The goal of *Bayes by Backprop* is to use unbiased estimates of gradients of the cost function to learn a distribution over the weights of the network [3] by minimizing this KL-divergence. However, notice that equation (2.3) involves the intractable term:

$$\boldsymbol{KL}[q_{\boldsymbol{\theta}}(\boldsymbol{w}|D)||P(\boldsymbol{w})] = \int q_{\boldsymbol{\theta}}(\boldsymbol{w}|D)\log\frac{q_{\boldsymbol{\theta}}(\boldsymbol{w}|D)}{P(\boldsymbol{w})}d\boldsymbol{w} \tag{2.9}$$

so that like in [3], we should follow a stochastic variational method whereby we sample $T$ weights $\boldsymbol{w}^{(i)} \sim q_{\boldsymbol{\theta}}(\boldsymbol{w}|D)$ and use these sampled weights to construct a tractable cost function $\digamma$, minimizable with respect to variational parameters $\boldsymbol{\theta}$ during training:

$$\mathfrak{F}(D,\boldsymbol{\theta}) \approx \sum_{i=1}^{T} log\, q_{\boldsymbol{\theta}}\left(\boldsymbol{w}^{(i)}\middle|D\right) - logp\left(\boldsymbol{w}^{(i)}\right) - logp\left(D\middle|\boldsymbol{w}^{(i)}\right) \tag{2.10}$$

where $\boldsymbol{w}^{(i)}$ denotes the $i^{th}$ Monte Carlo sample from the variational posterior. Blundell *et al*. explain that using the same weight sample for each additive term in (2.10) results in gradients of (2.10) that are only affected by the parts of the posterior distribution characterized by those weight samples. Graves [21] and Hinton and Van Camp [23] both suggest a variational inference over a Monte Carlo approach in approximating the posterior, for its computational speed.

### 2.6.2 The local reparameterization trick

In line with what was explained above, this section explains the essence of backpropagation through stochastic nodes for any Bayesian neural network. Recall that the goal is derivative-based optimization of the parameters $\boldsymbol{\theta}$ of the variational distribution $q_{\boldsymbol{\theta}}\,(\boldsymbol{w}|D)$. The formulation of this approach is borrowed from [32] and relies on the parameterization of the sampled weights $\boldsymbol{w} \sim q_{\boldsymbol{\theta}}\,(\boldsymbol{w}|D)$ as $\boldsymbol{w} = f(\boldsymbol{\theta},\varepsilon)$ for some differentiable $f$ and random noise $\varepsilon \sim p(\varepsilon)$ [32]. The reason for this approach is that direct random sampling of weights cannot be used here because backpropagation through such an operation is not possible. This reparameterization trick instead randomly samples an auxiliary variable $\varepsilon$, which is then shifted by the mean and scaled by the standard deviation and treated as $\boldsymbol{w}$. The intuitive explanation is that randomness now acts as input to the model, as opposed to being something that occurs inside of it, in which case differentiating would be impossible.

To make this more concrete, suppose we assume a *normal* $q_{\boldsymbol{\theta}}\,(\boldsymbol{w}|D)$ as in Shridhar *et al.* [47], so that this 'trick' will move the learnable mean and variance of $\boldsymbol{\theta} = (\boldsymbol{\mu},\ \sigma^{\mathbf{2}}\boldsymbol{I})$ out of the distribution function for any weight $\boldsymbol{w}$. Therefore, supposing that $\boldsymbol{\theta} = (\boldsymbol{\mu},\ \sigma^{\mathbf{2}}\boldsymbol{I})$ and with $\boldsymbol{\varepsilon} \sim N(\mathbf{0},\boldsymbol{I})$ we have now $\boldsymbol{w} = f(\boldsymbol{\theta},\varepsilon) = \boldsymbol{\mu} +\ \sigma\boldsymbol{\varepsilon}$**.** This representation as in [47] will allow us to propagate through a stochastic node during backprop defined as a transformation in terms of $\boldsymbol{\varepsilon}$, which is now independent from the model. Neural networks regularized with dropout seen in section 2.5.3 have this property too, although in a different form: they are designed to take stochasticity as an input [19].

## 2.7 Experiments and applications

### 2.7.1 *A motivating example of Bayesian uncertainty using a neural network for function approximation*

In this experiment, Bayesian principles of uncertainty in the context of noisy function estimation is demonstrated on a synthetic dataset. Epistemic and aleatoric uncertainties are derived and shown where the goal is to have a neural network trained with dropout approximate the function $f(x) = 3x + 7$ for 1000 noisy training examples $(x, f(x) + \varepsilon)$, where the noise is standard normal. The posterior distribution (and thus predictive posterior distribution) over the parameters is inferred via Monte Carlo (MC) sampling (done 20 times as in [16]), giving an ensemble of model predictions on the test set (of size 1000).

See appendix section A.E for more detailed explanations of the data, the model, and how uncertainties are modelled. The basic idea is that Monte Carlo sampling of the posterior returns a mean and a variance for each observation, which are used to calculate epistemic and aleatoric uncertainties. Epistemic uncertainty is calculated as the variance of the 20 MC samples (i.e. the 20 MC predictions $E(f(x)|x)$), while the aleatoric is measured as the average of the variances of these samples [31].

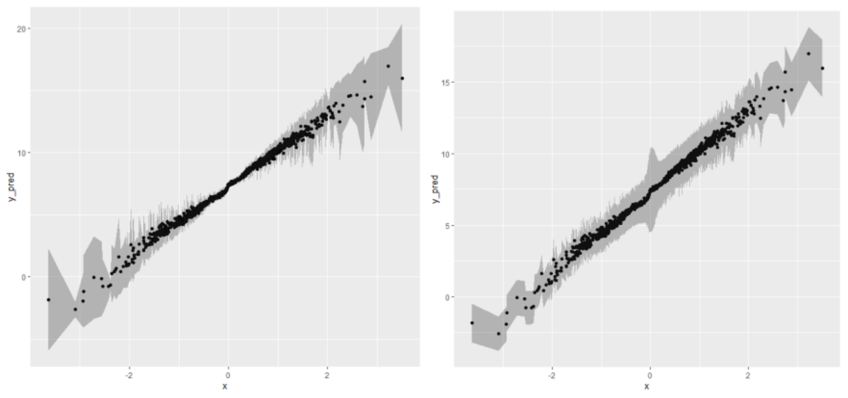


Figure 2.3: On the left: epistemic uncertainty. On the right: aleatoric. The shaded bands indicate one standard deviation above and below the predicted mean.

The epistemic uncertainty visible in the left panel of Figure 2.3 makes it clear that the model feels more confident when new examples are close to what it has seen, a concept seen again in the next experiment. This uncertainty increases for more ‘exotic’ examples and allows the model to say, “I’m not so sure about that one”. The right side of the panel shows the (irreducible) data-inherent aleatoric uncertainty, which is crucial for factoring in a safety margin. Interestingly, the aleatoric uncertainty appears greater close to the value $\mu_x = 0$, perhaps due to the mass of $x$-samples being drawn from here and the model thus having influence only through the intercept parameter (the effect of the slope disappears, and the model loses expressive power). Combining both uncertainties into the model is as simple as adding them (recall equation (2.7)), shown below in Figure 2.4.

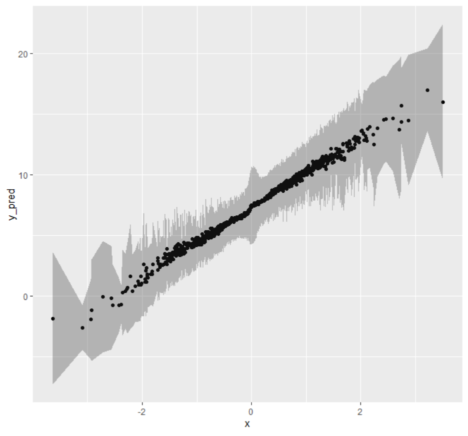


Figure 2.4: Epistemic and aleatoric uncertainties combine to demonstrate the fundamental idea that the model's uncertainty (i.e. epistemic) is driven away in regions with many examples, while the data-inherent uncertainty (i.e. aleatoric) remains in those regions. More exotic examples (and this extends to other applications in machine learning like computer vision where an image of a car may be taken from some strange angle) give rise to an uncertainty that is a sum of the model's shortcomings as well as of the data's.

### 2.7.2 *A Bayesian neural network classifier that relies on uncertainty when making predictions*

As explained in chapter 1, the main difference between the frequentist neural network and the Bayesian counterpart is the use of probability distributions over model parameters (weights and biases both), as opposed to single point estimates. In this experiment, a Bayesian neural network with a specific architecture (see appendix A.E.2.7.2) is used in the context of handwritten digit recognition (the MNIST dataset with $k = 10$ classes). Explored too, is how the Bayesian approach formulates *uncertainty,* granting the network the freedom to refuse outputting a prediction for very 'exotic' examples it may be unsure about. Since the underlying weight posterior is unknown (see equation (1.1)), a variational approach is taken here as explained in section 2.4.

As that section explains, one can decide how many times to sample the weights from this variational posterior- the more samples $T$ drawn, the more networks vie for making a prediction on that *one* new example. For the specifics of the network, as well as these manipulations, see the appendix. It is important to note that the neural network's predictions are now 'filtered' by a *threshold* (see Table A.E.2.4) such that higher values of it make the network more 'picky' (more precisely, the threshold acts as a percentile cut-off such that only those class-specific posterior distributions with enough mass beyond it will lead to a prediction for that class).

Table A.E.2.4 explores how increasing the threshold makes the network 'pickier', but ultimately improves its predictive accuracy, without evident implications to uncertainties (see Figures A.E.2.1 and A.E.2.2). One interesting finding from these figures is that we may minimize uncertainties together, for a certain threshold (0.5 in this case). In the interest of time, weights are sampled $T = 10$ times when making predictions, since Table A.E.2.3 shows a lack of significance of this variable. Manually setting this variable (and for a particular value of it, the threshold as done in Table A.E.2.4) is a form of *grid search* which can be laborious and arbitrary. In chapter 4, *Bayesian Optimization* as an automated and informed alternative will be tried. Notice that the average prediction accuracy is now as much as 4% higher, and the epistemic uncertainty has fallen. Interestingly, the aleatoric uncertainty does not seem to be much lower, despite the network passing on the more 'exotic' images like the handwritten 6 in Figure 2.5. These relationships are shown graphically in the appendix.

Figure 2.5 below shows the case when $T = 10$ posterior predictive distributions $P(C = c|\boldsymbol{x}^{new}, \boldsymbol{X}, \boldsymbol{y})$ are fit by the Bayesian neural network. It is clear that the network feels very confident about the 7, but not about the 6, and it thus refuses to classify it.

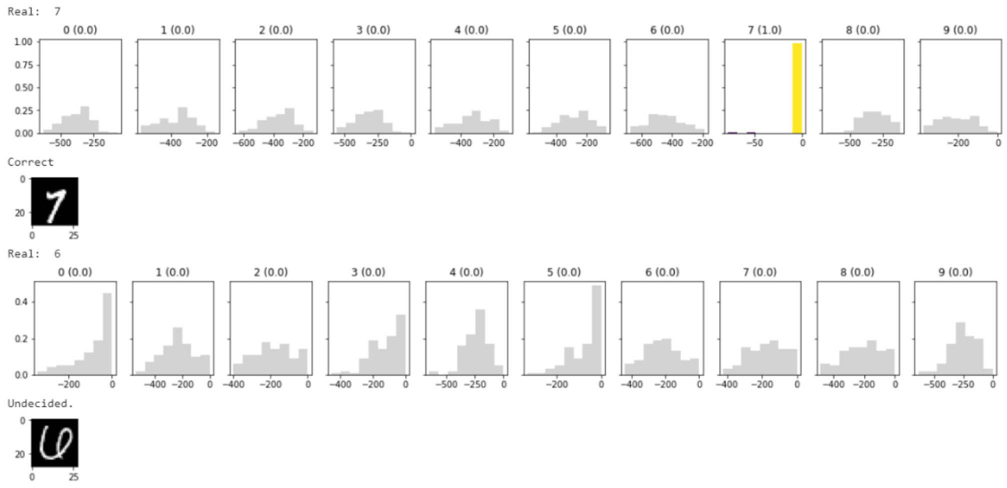


Figure 2.5: A confident prediction compared to a not-so-confident prediction. Such an application is well-warranted in medical imaging, but beyond the scope of this paper as the complexity of modeling in that context is worth its own attention.

Finally, Figure 2.6 below shows that this Bayesian network refuses to classify foreign images (here taken as a randomly generated one):

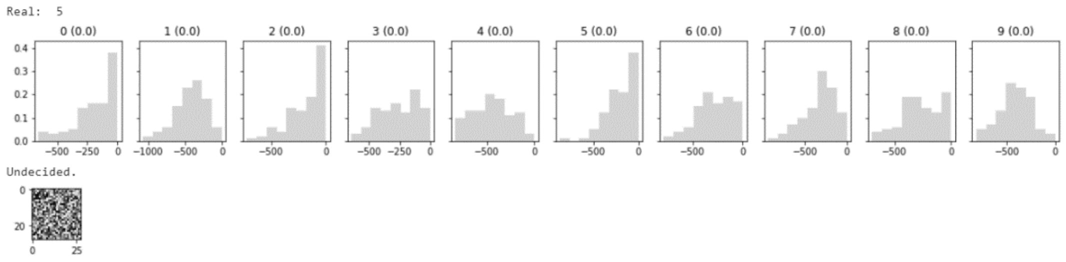


Figure 2.6: A randomly generated image leaves the network undecided. In purely statistical terms, putting this image in the test set creates too excessive of a discrepancy between the test set and training set distributions, causing the network to be too uncertain about its prediction.

# Chapter 3

# Bayesian deep learning

**Abstract**

Given what we have seen in the previous two chapters, we should now have a good grasp on the Bayesian philosophy and its methodology for complex models like neural networks. At this point, the level of complexity can yet increase, and David MacKay's two levels of inference are developed for the powerful deep learning *convolutional neural network* (shortened 'CNN'), a biologically inspired model with many features designed primarily for handling image data. These models were given the Bayesian treatment only as recently as 2018 [47], yet a rigorous Bayesian definition like that of MacKay for traditional neural networks is lacking. The chapter begins with the inspiration for these models, followed by a relatively thorough illustration of all of its quirks and features, before giving it this Bayesian treatment. Finally, a quick walkthrough is given on how to train such a model [47], concluding with experiments that illustrate (among other things) the Occam factor for Bayesian CNNs.

## 3.1 Convolutional Neural Networks

### 3.1.1 The neuroscientific basis

As an extension to the neural networks considered thus far, a *convolutional* neural network (CNN) is one that has been specialized to process data with a grid-like topology, such as imagery in the field of computer vision. In [19], Goodfellow *et al.* describe how such a network finds its roots in the V1 region (primary visual cortex) of the mammalian brain. The finding of neurophysiologists David Hubel and Torsten Wiesel [24] that neurons in the *early* visual system respond most strongly to very specific patterns of light such as oriented bars (but hardly to any other patterns) is a parallel phenomenon in convolutional neural networks [19]. To that end, the V1 region is the first area of the brain to begin advanced processing of visual input, having three properties which convolutional networks are designed to capture.

The first is the arrangement of the V1 region as a spatial map. Its two-dimensional structure mirrors the structure of the image in the retina and is captured by defining the features of a convolutional network in terms of two-dimensional maps [19]. The second and third property of the V1 region concerns itself with the *simple* and the *complex* cells that define a hierarchy structure [24]. This hierarchy assigns simple cells the task of recognizing simpler features (like edges and corners), while the complex cells not only have a tendency to respond to more complicated features (like ensembles of the simpler features learned by the simpler cells) but are more robust to small shifts in the feature's position or lighting conditions [19]. The convolutional network analogously treats cells as neurons, and has its shallow layers extract simple features while its deeper layers piece these together into more complicated patterns.

### 3.1.2 Motivation of convolutional neural networks

As explained above, the convolutional network offers a simplistic interpretation of the visual process in a mammalian brain, but there are design subtleties left to be discussed that suggest this form of learner over a traditional fully connected neural network (though we will soon see that these models still utilize fully connected layers, just in a different way). There are certain properties of convolutional networks that make them better suited for image recognition tasks. Traditional neural networks are designed such that every layer's neurons are connected to every other layer's neurons through the use of fully connected interactions, and linear functions involving the weights in each layer as in appendix A.A.1.4 [19]. For *convolutional* networks, this is not the case.

The sparse connectivity of convolutional networks enables them to instead focus on smaller regions of the image and detect meaningful features such as edges through locally applying a 'scanner' called a *filter* (or sometimes *kernel*). In short, a filter is an array that scans the image left-right and up-down, having as many channels as the image itself (3 channels for RGB, 1 channel for greyscale) and length and width less than those of the image. It operates by multiplying its values pixel-wise with those of the image at its current location (called a Hadamard product in mathematics), and effectively producing a *feature map* as a form of compressed representation of the image. The specifics of these filters are seen in section 3.2.2.

In essence, the image is analyzed locally and thus only requires local descriptors, as opposed to being analyzed globally and requiring full connectivity. This reduces the number of weights to learn, improving statistical efficiency [19]. This then causes deeper layers in the network to interact *indirectly* with the shallow layers, thereby piecing together simple features to construct complex concepts, as shown below in Figure 3.1:

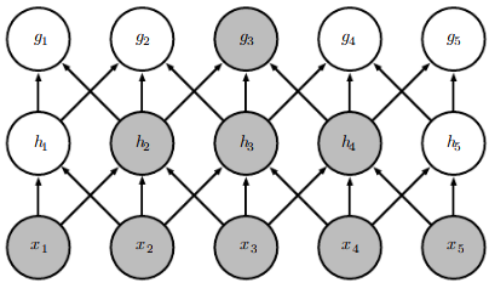


Figure 3.1: The receptive field increases with the depth of the network, but *direct* connections are very sparse. Therefore, *indirect* connections to the input image exist, and are channeled through the network's shallower layers. Image source: Goodfellow et al [19]; page 327

As a result of sparse connectivity, convolutional networks leverage *weight sharing* to reduce the connectivity between neurons in adjacent layers. This is done by learning a small set of weights and applying this set to subsets (of the same size) throughout the image, rather than having a different weight for each pixel in the input image. This allows the network to effectively learn features such as edges and corners. Finally, weight sharing gives the network a property called *equivariance* to local translation [19]. This essentially means that shifting the image's contents by up to some amount does not affect the network's predictions, and since some images contain features that more or less repeat throughout the image, weight sharing becomes practical.

Without weight sharing and sparse connectivity, the convolutional network would lose the ability to learn features with single neurons (as shown in Figure 3.1 above) *and* with far fewer weights overall. Detecting a vertical edge in a 200*200 image would be far more efficiently done using the learned feature detectors (the filters), as opposed to a search for these vertical edges over every possible distinct location, which is what fully connected neural networks would do. In essence, a convolutional neural network is simply a neural network that uses *convolution* (see next section) instead of general matrix multiplication in at least one layer [19], and loosely speaking thinks locally rather than globally.

## 3.2 The structure of convolutional neural networks

### 3.2.1 Idea

It is worth a section of this paper to illustrate the way that these networks work, as understanding their minute details is requisite to treating them with David MacKay's framework of Bayesian inference in those sections. So far, it was explained *why* these networks are well-suited for complex problems like image recognition, but just *what* they consist of is discussed in neither a Bayesian nor a frequentist manner here, since no form of inference is yet involved. As in section 1.4, a convolutional network is best conceptualized as a mapping from a tensor input $\boldsymbol{x}$ of dimensions $M \times N \times K$ (where $K$ is the number of channels; like 3 in the case of an RGB image, or 1 if greyscale) to a label $y$. There is currently a very limited rigorous definition of these networks, and most design choices remain *ad hoc* [47].

A convolutional network can consist of several layers, each of which is typically made up of three stages. In the first stage, the convolution operation is performed by sliding the filters (there can be more than one) over the current state of the image (depending on which layer we are looking at), and producing some linear activations which are then passed forward to the second stage through nonlinear activation functions (a versatile choice is the ReLU, or *rectified linear unit* activation) [19]. The mathematical formulation is omitted here for brevity but is described in much richer detail by Goodfellow *et al.*'s book [19].

In the third stage, a *pooling* function modifies the output even further. Wherever applied, the goal of pooling is to replace local outputs of the current layer (that is, pixel values over the current receptive field) with a single summary statistic like an average or a maximum (called *average pooling* and *max pooling*, respectively). It is the use of this pooling that makes the so-called representation (or *feature map* because it is no longer the original image, but some version in this compressed space) invariant to local translation, as mentioned above [19]. This property is great when one is concerned about the presence of a certain feature (such as an eye) in an image, and not so much about its precise location.

Passing the filter over the image in the first stage creates this feature map, and depending on *where* the image is in the network (recall that shallow layers extract low-level features while deeper layers piece these together to construct more complex patterns as in Figure 3.1), the feature map will 'summarize' the image by its features (or patterns). These feature maps are shown below in Figure 3.2 (although this is Bayesian, since it places distributions on the weights of the network), along with the input image that is convolved by the filter. Note that the discussion here is rather general in that it can apply to any CNN architecture (which is where the modeler can get creative: they decide the size of the filter, the number of layers, etc.).

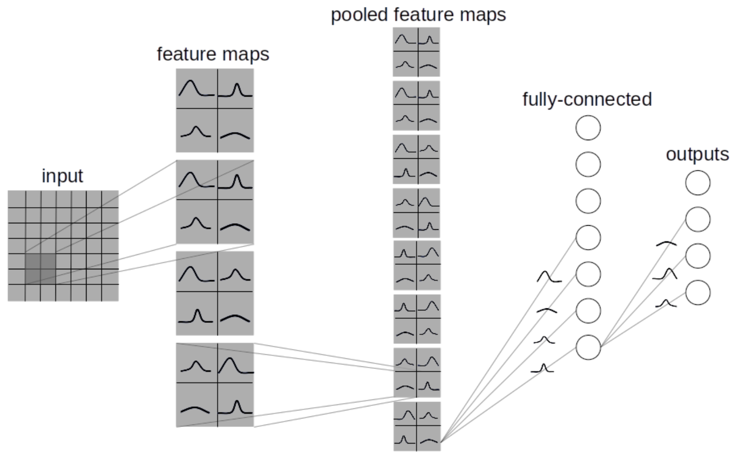


Figure 3.2: An over-simplification of a Bayesian convolutional neural network showing the use of distributions over model weights wherever possible. Image source: [47].

As a simple example, let us suppose that the goal is to identify vertical edges in an image. Recall that the filter is applied over a region of the image, such that its values are multiplied elementwise (convolved) with the image as it slides over it. In that case, we could hand-engineer the filter to be square in shape (for simplicity), and have larger numerical values in a vertical arrangement down its center (although the center is not strictly necessary, but again simple) so that as it is convolved with the image, the Hadamard product will be more or less maximized along some vertical axis, and the constructed feature map will capture this.

The novel idea of convolutional neural networks is that instead of hand-engineering filters, they are *learned*; this is done by assigning them weights (and how this is done is seen in the next section) that are updated during training. The Bayesian in typical fashion gives the weights of the filter distributions (see Figure 3.2). Learning filters makes the network more versatile and helps it generalize as it can be argued that the alternative of hand-engineering them only serves the purpose of extracting a particular pre-chosen feature like a vertical edge.

As a conceptual example, consider an $RGB$-image of dimensions $224 \times 224$ as input to the network. Such an image has 3 channels, and thus requires a filter of just as many channels to convolve over it. As we will see in Table 3.1 below, we can pick the width and length of the filter, so that if it is taken to be $10 \times 10 \times 3$ then there will be $10 * 10 * 3 + 1 = 301$ weights for this filter (including an additive bias). If there are, say, 5 filters in that layer then there will be $301 * 5 = 1{,}505$ weights in that layer to be learned (Table 3.1 explains why we may have multiple filters). Therefore, the number of weights in the network is a function of its design (through its filter) and not the dimensions of the input (of course, the filter's maximal dimensions are restricted by the image's) [19]. Notice weight sharing in action: the filter has the same weights regardless of where it passes over the image (hence why it learns the same features regardless of location), whereas in the traditional fully connected neural networks of chapter 2, there would be a weight for each pixel.

### 3.2.2 Design and hyperparameters

| Component name | Trainable? * | Hyperparameters? |
|---|---|---|
| Input layer | No, it just 'reads in' the image. | None, although dimensions of the image dictate its size. |
| Filter (as part of the convolution operation) | Yes, the filter has weights (and a bias) that are convolved with the pixel values of the output from the previous layer. Each filter can be characterized by $F_{ij,whd}$ , where $i$ is the layer, $j$ the specific filter for that layer, $w$ the width, $h$ the height, and $d$ the depth (dependent on the previous feature map). Therefore, each filter's learnable weight vector can be represented by $\boldsymbol{w}^{F}_{ij,whd}$.<br><br>The subtlety here is that weights are shared, unlike the fully connected weights in the final layers (see below). | The filter's *dimensions*: length, width, and depth (depth depending on which layer of the network we are looking at). The first layer receives the actual image, where the filter's depth will match it (3 if image is RGB, for example). For deeper layers, the filter's depth changes depending on the previous layer's output. Each layer can have an arbitrary number of filters, each of which learns what features to extract (too many filters may overfit). The *number of filters in each layer* is therefore also a hyperparameter.<br><br>The *stride* and use of *padding* ('same' and 'valid'). The former decides the pixel distance to slide the filter by (a value of 2 means that the filter slides over by 2 pixels each time it convolves over the image). Padding adds zeros to the edge of the image (if 'same') so that the filter has enough pixels to convolve everywhere, otherwise (if 'valid') the edges are not padded, and the filter will not apply to the residual pixels on the edge. |
| Pooling operation | No, pooling acts as a down sampling method, and relies on a choice of summary statistic. | The choice of *summary statistic*: average, max, median, etc. Down sampling means that the output of the convolution has its dimensions reduced via pooling. |
| Activation function | No, these functions just process outputs of each convolution | The choice of activation: ReLU, $tanh$, etc. [19]. Experiments use *Softplus* as discussed in section 3.4.3. |

| | | |
|---|---|---|
| | layer and introduce non-linear representations. | |
| Fully connected layers | Yes, these final layers have their own weights that prepare the processed image for classification. We can differentiate these weights from the weights of the filter as $\boldsymbol{w}_k^{FC}$ with $k$ denoting the $k^{th}$ layer. | The *architecture*, like the number of layers, the size of each layer, and the final output layer activation (such as *SoftMax*). |

Table 3.1: Components of a CNN with specification as to what is trainable and what is not. This way, we can see where we need to place distributions when being Bayesian.

Figure 3.2 shows us one fundamental fact about these networks. They are essentially composed of two parts: the first half is responsible for processing the image through the use of the convolutional and pooling operations discussed in 3.2.1, while the second half is reminiscent of what we have seen in chapter 2: a fully-connected feedforward neural network that takes the processed image and prepares it for prediction. Table 3.1 above characterizes the different components of CNNs, with a description of what is trainable so that the reader can go into the next section understanding where and how Bayesian inference can be applied in line with MacKay's formulation seen in chapter 1.

Note that Table 3.1 does not consider aspects related to data and training as *network* hyperparameters. These include the batch size, the number of epochs in training, the choice of data preprocessing, the choice of weight optimizer (the means by which we maximize equation (2.4) in the Bayesian case), and so on. I consider these as general considerations for any form of neural network, while Table 3.1 strictly pertains to the components of a CNN. We may still condition on these in spirit with MacKay's first level of inference, but just include them under the hyperparameter abstraction $\alpha$, despite being unrelated to the network itself.

### 3.2.3 The convolution and pooling operations as infinitely strong priors

A curious Bayesian thought experiment that views convolution and pooling as 'infinitely strong' priors is useful to understanding how they work [19]. It was stated in chapter 1 that a prior distribution over a model's parameters is considered 'strong' when it has low entropy, placing more probability mass over a certain region of values for $\boldsymbol{w}$, like a Gaussian with a small variance. An infinitely strong prior will place zero probability mass over certain values of $\boldsymbol{w}$, making them forbidden (basically like an infinitely strong bias against those values). Recall that CNNs rely on weight sharing (Figure 3.1), which places an infinitely strong prior over weights and says that the weights for one hidden unit must be identical to the weights of its neighbour but shifted in space [19]. This prior says that there are only local interactions to be learned and is equivariant to translation.

One reason that it is important to conceptualize it in this Bayesian way, is that some learning tasks that use convolutional networks may behave awkwardly. For example, an image-generation model for human faces may create an image of a face with two different eye colours, a result of not being able to model relationships between pixels at larger distances across the image due to this infinitely strong prior. In the next section, when extending David MacKay's two levels of inference to these more complicated models, it is essential to consider this idea.

### 3.2.4 Bayesian formulation of a convolutional neural network

Recall from equation (2.1) that we may even condition on the network's hyperparameters α, yet we can be more specific by explicitly distinguishing between $\alpha^{CNN}$ hyperparameters of Table 3.1 and $\alpha^{train}$ hyperparameters related to the training set-up. This distinction is especially important because a modeler in practice may tweak the network's hyperparameters (tuning $\alpha^{CNN}$) for a given experimental design setup (fixed $\alpha^{train}$). Interestingly, while Bayesian inference over the network weights conditions on these, it is also possible to infer their distributions in a Bayesian fashion as in MacKay's work [37] (treating them as random variables, this idea is explored in close detail in chapter 4). This bridges nicely over to MacKay's second level of inference, which is concerned with model selection.

If we consider that $\alpha^{conv.}$ consists of the specifics of the filter, the form of pooling, the choice of activations, and so on, then we may consider the convolutional network as two pieces, starting with the convolution half. Taking the Bayesian view and representing this first half of the network in spirit with equation (1.8) gives:

$$g(\boldsymbol{x}) = f_C\left(\dots f_2\left(f_1\left(\boldsymbol{x}; \boldsymbol{w}^F_{1J,whd}\right); \boldsymbol{w}^F_{2J,whd}\right) \dots\right), \boldsymbol{w}^F_{CJ,whd}\right) | \boldsymbol{w}^F_{iJ,whd} \sim P\left(\boldsymbol{w}^F_{iJ,whd} \,\middle|\, \boldsymbol{w}^{FC}_k, D, H_i, \alpha^{conv.}\right)$$

where the function $f_i$ represents the 3 stages explained in section 3.2.1 (convolution, activation, pooling applied $C$ times) although alternative architectures that violate this order are entirely possible. Note the use of $J$ as a subscript on the filter weights to instead denote the set of *all* filters for that respective layer, as opposed to denoting single filters via $j$ as in Table 3.1. The fully connected second half of $L$ layers can likewise be expressed, although with the function transformations we have seen in chapter 1. Now, we also condition on $\alpha^{FC}$ which consists of hyperparameters like size of layer, number of fully connected layers, and activation in the output layer:

$$f(\boldsymbol{x}) = f_L(\dots f_2(f_1(g(\boldsymbol{x});\ \boldsymbol{w}^{FC}_1);\ \boldsymbol{w}^{FC}_2) \dots),\ \boldsymbol{w}^{FC}_L) |\ \boldsymbol{w}^{FC}_i \sim P\left(\boldsymbol{w}^{FC}_k \,\middle|\boldsymbol{w}^F_{ij,whd}, D, H_i, \alpha^{FC}, \alpha^{conv.}\right)$$

where the input to the first fully connected layer is the output of the final convolution-pooling layer in the first half of the network. This input is generally flattened into vector-form before being fed into these final layers. Note that while these two functions feature weights sampled from their respective marginal distributions, the goal of Bayesian learning here is to learn the *joint* posterior $P\left(\boldsymbol{w}^F_{ij,whd}, \boldsymbol{w}^{FC}_k \middle| D, H_i\right)$ seen in the following section.

## 3.3 MacKay's formulation adapted to convolutional neural networks

Recall from the previous section that a distinction was drawn between the weights of the first half of the network, and those of the fully connected second half. Recall also that the CNN differs from the traditional neural network in that weights are shared, a fact that differentiates these networks from feedforward neural networks. While the Bayesian treatment to CNNs is as recent as 2018 [47] there has been no formal or rigorous Bayesian definition like the one we find in MacKay's 1994 work [37]. To that end, this section aims to formalize *all* features of these models that give themselves up to inference, in that Bayesian manner.

We have up until this point seen and become familiarized with the various quirks and features of these models, with detailed explanations of section 3.2.2 regarding the learnable aspects. But learnable does not immediately mean point-estimate, as has been consistently argued throughout this paper. To that end, the two levels of Bayesian inference of MacKay are extended to these models, where the reader will find the biggest difference in the first level (where the second level is concerned with model selection in a *practical* way, with comments more inspired by practice).

### 3.3.1 The first level of inference

Recall from chapter 1 that MacKay's first level of inference is concerned with *model fitting*, which amounts to inferring model weights for a given selected model, $H_i$. It was shown, however, that convolutional neural networks have components that vary dramatically in scope and complexity, where convolution layers extract features, while fully connected layers finalize classification. As Table 3.1 showed, it was necessary to distinguish weights of the filter from weights of the fully connected layers, as $\boldsymbol{w}_{ij,whd}^{F}$ and $\boldsymbol{w}_{k}^{FC}$ respectively since they fundamentally serve different purposes. Note however that these weights are still trained in conjunction with backpropagation [19] (and by *Bayes by backprop* in the Bayesian case). Therefore, when it comes to CNNs, for a certain image-recognition task consisting of a dataset $D$ and suitable collection of models $H$ (this can be a collection of different CNN architectures), we have for one such model $H_i$ the weight posterior:

$$P\left(\boldsymbol{w}_{ij,whd}^{F}, \boldsymbol{w}_{k}^{FC} \middle| D, H_i\right) = \frac{P(D|\boldsymbol{w}_{ij,whd}^{F}, \boldsymbol{w}_{k}^{FC}, H_i)P(\boldsymbol{w}_{ij,whd}^{F}, \boldsymbol{w}_{k}^{FC}|H_i)}{P(D|H_i)} \tag{3.1}$$

where weights of the filters and weights of the fully connected layers are inferred simultaneously, via *Bayes by backprop* (recall that with variational inference, the goal would be to approximate the intractable (3.1) by $q_{\boldsymbol{\theta}}(\boldsymbol{w}_{ij,whd}^{F}, \boldsymbol{w}_{k}^{FC}|D)$). It may simplify things to concatenate these weights into a single vector $\boldsymbol{w} = \left(\boldsymbol{w}_{ij,whd}^{F}, \boldsymbol{w}_{k}^{FC}\right)$ for the sake of notation.

### 3.3.2. The second level of inference

In chapter 1, we saw David MacKay's elegantly intuitive Occam factor at work in a theoretical fashion, which arguably can be practical for both small-scale models and small-scale data. However, as we know, this is not the case with CNNs: neither the model nor the data is every 'simple', so that representing the reduction in entropy of the weights $\boldsymbol{w}$ by a single ratio after training is not only a horrible oversimplification, but also rather naïve. The reason is that CNNs are extremely powerful and also extremely versatile and come with a plethora of performance metrics for a certain task, that would be neglected if only examining this.

Even the most elementary binary image classification should report (for example) test-set accuracy, test-set precision (the fraction of positive predictions that are actually positive), test-set recall (proportion of negatives correctly classified as such), F1 score (the harmonic mean between these two), uncertainties we have seen in chapter 2, and more [19]. Therefore, it may be more sensible to compare models on the basis of these metrics, which are also widely tractable. One concern may be that these metrics do not, however, embody Occam's razor. However, Occam's razor is at play in a more subtle way here: these metrics are particularly concerned with generalization, which can only be good when the model is such that it penalizes complexity in favour of performance. Furthermore, recall that the Bayesian network (convolutional or otherwise) is trained by minimization of the KL divergence (see equation (2.3)) which implicitly incorporates regularization. However, these metrics are only part of the story; on their own, they tell us very little about the network and its complexity besides raising a red flag that perhaps overfitting may be the case (i.e. violating Occam's razor). Experiment 3.5.2 examines the relationship between the Occam factor of the network and other performance metrics, while section 5.1 does so in the context of overfitting.

To that end, a more complete picture of model comparison for deep learning models like CNNs considers their architecture [19], with an eye out for overfitting which often plagues these naturally complex models. Interestingly, authors Gaier and Ha [13] show that even complex tasks like balancing a rod can be learned when all weights take the same arbitrary value (except 0, of course), but only the network structure is learned. This suggests an optimal structure for a choice of weights. Furthermore, neural networks are *degenerate*, which means that for a desired metric (like classification accuracy), there exist many weights $\boldsymbol{w}$ (or weight distributions $P(\boldsymbol{w}|D, H_i)$) to achieve it [19]. This suggests an optimal set of weights for a choice of (expressive enough) structure. Deep learning models seek an optimal combination of the two, with attention to simplicity (and in the Bayesian case, we have seen how this is achieved by incorporating regularization on the learned distribution, rather than on the weights directly).

To that end, it is worth discussing the *generalization gap*, which is ubiquitously interpreted as the difference between the network's training set accuracy and the network's test set accuracy [19]. Considering that we train for $n$ epochs (which means $n$ complete passes through the training set), the $n^{th}$ pass will give us the final trained network with learned weights $\boldsymbol{w}^*$ in the frequentist case, and learned posterior $q_{\boldsymbol{\theta}^*}(\boldsymbol{w}|D)$ in the Bayesian case. If $GG_H$ denotes the generalization gap for the choice of network $H$ and $A$ the average accuracy across all batches, then:

$$GG_H(\boldsymbol{w}^*, \alpha) = A^{training}(\boldsymbol{w}^*, \alpha) - A^{test}(\boldsymbol{w}^*, \alpha) \tag{3.2}$$

and in the Bayesian case:

$$GG_H(q_{\boldsymbol{\theta}^*}(\boldsymbol{w}|D), \alpha) = A^{training}(q_{\boldsymbol{\theta}^*}(\boldsymbol{w}|D), \alpha) - A^{test}(q_{\boldsymbol{\theta}^*}(\boldsymbol{w}|D), \alpha) \tag{3.3}$$

Note the inclusion of $\alpha$ as defined by Mackay (seen in chapter 2); the gap is certainly a function of the network's hyperparameters (its architecture, for example) which impact overfitting. We should also track the generalization gap between the training and validation sets, to see how overfitting plays out *during* training. For the currently sampled weights (from the not-yet optimal variational posterior $q_{\boldsymbol{\theta}^k}(\boldsymbol{w}|D)$) in iteration $k$ during Bayesian training, this gap can be defined as:

$$GG_H(\boldsymbol{w}^k, \alpha) = A^{training}(\boldsymbol{w}^k, \alpha) - A^{validation}(\boldsymbol{w}^k, \alpha|I_d)|\boldsymbol{w}^k \sim q_{\boldsymbol{\theta}^k}(\boldsymbol{w}|D) \tag{3.4}$$

where $I_d$ denotes an indicator of whether dropout is active during validation ($I_d = 1$) or not ($I_d = 0$). Generally speaking, if dropout is on during training, but off during validation, the generalization gap can come out negative [19] (see experiment 3.6.1), although (3.3) can stay positive. Indeed, it is possible to combine dropout with the Bayesian treatment, a mathematical development for which is given in [15] but is beyond the scope of this thesis.

In the Bayesian paradigm, the gap directly depends on the learned posterior from the first level as (3.3) shows, which directly bridges the gap between training and testing by constructing the predictive posterior from equation (2.2). Although current research on the Bayesian generalization gap is limited, it would make for interesting future research, especially in the context of *model selection*- which is what this second level is concerned with.

## 3.4 Methods for training Bayesian convolutional neural networks

### 3.4.1 The local reparameterization trick for convolutional layers

Having seen in large how convolution works in 3.2.1, this section applies the neural network mechanical details of chapter 2 to CNNs, starting with an application of the local reparameterization trick of chapter 2 to these models. Before anything else is said, it is essential to give the variational posterior $q_{\boldsymbol{\theta}}(\boldsymbol{w}|D)$ a description, in the form of a suitable distribution as taken by the authors Shridhar *et al.* [47]. This form is the normal, such that by following [47], $q_{\boldsymbol{\theta}}\left(\boldsymbol{w}_{ijhw}|D\right) = N(\mu_{ijhw}, \alpha_{ijhw}\mu_{ijhw}^2)$ with $i, j$ denoting the input-output layers, and $h, w$ denoting the height and width of the given filter, as we saw in Table 3.1. Shridhar *et al.*'s assumption that the variational posterior is normal allows straightforward application of this local reparameterization trick in convolutional layers (the consequences of which are explained in 3.4.2 below). Putting these together gives convolutional layer *activations b* (recall that activations in CNNs play the role of allowing the network to learn non-linear features):

$$b_j = A_i * \mu_i + \varepsilon_j \odot \sqrt{A_i^2 * (\alpha_i \odot \mu_i^2)} \tag{3.5}$$

In this formula, $\varepsilon_j \sim N(0,1)$, $A_i$ is the receptive field (that part of the image visible to one filter at a time), $*$ denotes convolution and $\odot$ component-wise multiplication. During training, this activation $b$ is sampled instead of $\boldsymbol{w}$ due to its computational acceleration [47], and another motivation is explained below. Note however that the normality assumption is not always appropriate, and this is discussed in more detail in section 3.5.

### 3.4.2 Two convolutional operations for mean and variance

Recall that in the frequentist view of a convolutional neural network, filters that pass over the image in each layer of the network apply one convolutional operation only. In the Bayesian view, the point-estimates turn into distributions, and when assumed normal these come with a mean and variance (two parameters instead of just one, the weight itself). Shridhar *et al.* [47] update the variational posterior $q_{\boldsymbol{\theta}}\left(\boldsymbol{w}_{ijhw}\middle|D\right)$ via backpropagation by applying two convolutional operations, one for the mean and one for the variance [47]. Since the activation *b* sampled during training is a function of both mean and variance (which is itself a function of the mean), these can be learned and used to parameterize the normal distributions. Once training is finished, the network can be pruned via L1 regularization to reduce the number of parameters [47], and the experiments of this chapter implement these techniques (along with what follows in 3.4.3 below).

According to the approach taken by Shridhar *et al.*, the first convolutional operation treats the *b* activation as the output of a CNN trained via frequentist inference [47]. This point-estimate is derived in their work (and extended in the experiments) via the Adam optimizer, and is interpreted as the mean $\mu_{ijhw}$ of the variational posterior $q_{\boldsymbol{\theta}}\left(\boldsymbol{w}_{ijhw}\middle|D\right)$ so that the first convolutional operation learns the *MAP* (since it is normal, the mean is the mode).

Recall that the variance is a function of the mean, so that the second convolutional operation just has to learn the parameter $\alpha_{ijhw}$, meaning that ultimately only one parameter *per* convolutional operation is learned, just like in frequentist training [47]. The amount by which $\boldsymbol{w}$ vary around the *MAP* estimate in the second convolutional operation finally capture the variance. To ensure positive variance $\log(\alpha_{ijhw})$ is learned and to enhance accuracy the *Softplus* activation (discussed below in 3.4.3) is used [47].

### 3.4.3 An example of some assumptions and mechanics

What remains to be discussed is how these all combine to create a viable machine learning model. This section follows the guidelines of Shridhar *et al.*'s paper on Bayesian learning of convolutional neural networks, starting with the activation function. They consider the *Softplus* activation because it ensures that the variance parameter discussed above never becomes 0. Whereas *ReLU* activations become zero for $x \rightarrow -\infty$, the *Softplus* activation is a smooth approximation to these functions that does not itself become zero. It has the form (note that $\beta$ can be taken to be 1 by default [3]):

$$Softplus(x) = \frac{1}{\beta}\log\left(1 + \exp(\beta x)\right) \tag{3.6}$$

The architectures for the convolutional neural networks can be common architectures like LeNet-5 and AlexNet and made Bayesian by placing distributions over parameters like Shridhar *et al*. do [47] (note that specific architectures like these were not alluded to in previous sections, as these are just specific examples and countless ones exist). Learning the objective function is done by *Bayes by backprop* (seen in section 2.6.1), maximizing the tractable *ELBO* cost function with respect to variational parameterization $\boldsymbol{\theta}$ of equation (2.4).

Finally, it is important to discuss just how the components of that cost function are explicitly articulated by Shridhar *et al.* [47]. Recall that the variational posterior is assumed normal, so that for sampled weights $\boldsymbol{w}^1, \ldots, \boldsymbol{w}^n$ (weights of both types as defined in section 3.2.4 for filters and for fully connected layers), the joint distribution is given by the following equation:

$$q_{\boldsymbol{\theta}}\left(\boldsymbol{w}^1, \ldots, \boldsymbol{w}^n | D\right) = \prod_{i=1}^{n} q_{\boldsymbol{\theta}}\left(\boldsymbol{w}^i | D\right) = \prod_{i=1}^{n} N(\boldsymbol{w}^i | \mu, \sigma^2) \tag{3.7}$$

where the mean and variance are discussed in the previous part, with the variance as a function of the mean. Taking the logarithm defines the posterior as:

$$log q_{\boldsymbol{\theta}}\left(\boldsymbol{w}^1, \ldots, \boldsymbol{w}^n | D\right) = \sum_{i=1}^{n} log\, N(\boldsymbol{w}^i | \mu, \sigma^2) \tag{3.8}$$

Finally, recall from chapter 1 that Bayesian inference is all about starting with a prior and updating it via the likelihood to produce the posterior. To that end, the prior over the weights is given by:

$$p(\boldsymbol{w}^1, \ldots, \boldsymbol{w}^n) = \prod_{i=1}^{n} N(\boldsymbol{w}^i | 0, \sigma_{prior}^2) \tag{3.9}$$

which is zero-centred with varying degrees of entropy given by the prior variance $\sigma_{prior}^2$, large values for which are less informative (encode fewer prior beliefs) and thus more data is needed to narrow the distribution via the likelihood (the mechanism which ‘feeds in’ the data). This likelihood is the last term $logp(D|\boldsymbol{w}^{(i)})$ from equation (2.11).

## 3.5 Distributional assumptions for the weights

### 3.5.1 The assumption of normality

The previous section is one such example among many of the form of assumptions one can make about the form of distribution over the weights. That section borrows from the work of Shridhar *et al.* [47] where the prior over the weights is taken to be normal, and the posterior remains normal.

However, one may generalize these assumptions to cover a wider variety of distributions (so long as they are appropriate for the task at hand: Shridhar *et al.* use categorical distributions to model the predictive distributions for classification). As such, it is important to dedicate a section of this chapter to discuss other approaches and motivate them in different contexts.

Non-normal distributional assumptions are not unusual. Indeed, Sun *et al.* [52] describe how other forms such as log-uniform and *horseshoe* priors [7] (a robust distribution that induces sparsity and was shown to be more applicable in Bayesian linear regression settings) can be suitable choices. These choices are motivated for model compression and model selection, and by favoring simpler models embody the philosophy of MacKay's Occam factor when properly specified. In line with favoring simplicity, I take the liberty of using normal assumptions for both the prior and the variational posterior in the implementation of variational inference in the experiments that follow this section. On the one hand, the experimental performance of the networks is not poor (although I make no claim that this is because the normal assumption is justified by any other means than *simplicity* and good previous results of Shridhar *et al.* [47]). On the other hand, the assumption of standard normal priors and a variational posterior whose variance is a function of the mean is one of diagonal covariance matrices and one of limited functional expressiveness, respectively.

### 3.5.2 A word about assumptions

Although interesting, a more rigorous exploration of prior specification and modelling assumptions would be best left for another study. In short, a sensitivity analysis as mentioned in chapter 1 [18] would check the robustness of the network to the model assumptions (i.e. how well can the network remain consistent despite different prior distributions, different assumptions on the variational posterior, and so on). Of course, if data were unlimited, this would hardly even matter: recall from chapter 1, that with more data, the prior's influence dissipates and the likelihood dominates (therefore, a proper likelihood specification would become the main focus). Unfortunately, what is currently available online for Bayesian deep learning in the way of libraries and public code repositories for flexible modeling is practically nil: the main contribution being that of Shridhar *et al.* [47], and so this too, would make for great future research and contribution.

Next, we will try our hand at these formulations (with some modifications, and some additional considerations that were omitted from these aforementioned studies). The assumptions made throughout the experiments are substantiated further in the appendix if ever the reader feels that something is lacking.

## 3.6 Experiments and applications

### 3.6.1 *A comparison between a Bayesian and frequentist CNN*

In this experiment, the common convolutional neural network architecture AlexNet is trained in both a Bayesian and frequentist way, and comparisons of the two approaches are made as in the work of Shridhar *et al.* [47], with Bayesian specifications as in section 3.4.3 above. This architecture is chosen for its inclusion of all CNN features explained in Table 3.1, as well as for its frequentist form featuring dropout (the other architecture considered in [47] being LeNet, which does not). The dataset of choice is MNIST. The two approaches are compared on the basis of their test set accuracies and generalization gaps (as in the second level of inference; see the appendix for details of training).

Note that confidence intervals on accuracy are reported instead of point-estimates, obtained by using dropout at test-time in the frequentist case with $p = 0.5$, and sampling from the variational posterior in the Bayesian case as discussed in section 2.4 (Gal & Ghahramani [15] report that using dropout 10 times is appropriate at test-time. For this reason, the variational posterior is also sampled 10 times when making predictions). Reporting in this way extends on the work of Shridhar *et al.* [47] and is reminiscent of the 'error bars' of MacKay [36], although for a different purpose. I report both traditional confidence intervals and distribution-free confidence intervals for the sake of thoroughness and to emphasize that the former requires dependence on some assumptions (like normality of accuracy) that may not be entirely satisfied [25]. Another reason is that predictions are made by sampling 10 times, which is an effectively small sample size of the metric we are interested in.

Table 3.2 below reports the *best* average performances of the frequentist and Bayesian AlexNets across 5 runs (unfortunately, averaging the runs in an automated fashion was rather impossible since both forms of AlexNet would occasionally become 'stuck' and not update, so I would need to manually restart the training). Note that since the architecture is pre-defined (and therefore so are its hyperparameters from Table 3.1), for simplicity I control the number of epochs so that MacKay's $\alpha = \{n\_epochs\}$ where $n_{epochs} \in \{5, 10, 15, 20, 25\}$. This way, I can observe marginal improvements with more training. Note that for each choice of $n\_epochs$, the network is re-trained from scratch rather than logging and comparing networks in 5-epoch intervals. Doing so creates more opportunities for different initializations, which is something to consider too.

| | | |
|---|---|---|
| Bayesian AlexNet with variational inference and *Bayes by backprop* (best result from 5 epochs) | **Average test accuracy:** 95.138 $\pm$ 3.195%<br>**Distribution-free confidence interval for test accuracy:**<br>(84.375, 1.00)<br>**Median test accuracy:**<br>96.875% | **Generalization gap** (training-validation)**:**<br>-1.043%<br>**Generalization gap** (training-testing)**:**<br>-3.160% |
| Frequentist AlexNet with dropout (best result from 15 epochs) | **Average test accuracy:** 85.838 ± 5.702%<br>**Distribution-free confidence interval for test accuracy:**<br>(83.878, 1.00)<br>**Median test accuracy:**<br>85.472% | **Generalization gap** (training-validation)**:**<br>-1.109%<br>**Generalization gap** (training-testing)**:**<br>0.658% |

Table 3.2: 95% confidence intervals on accuracy derived by using dropout at test-time for the frequentist AlexNet, and re-sampling of weights $\boldsymbol{w} \sim q_{\boldsymbol{\theta}}(\boldsymbol{w}|D)$ in the Bayesian case at test-time. Distribution-free confidence intervals are presented too, as a robustness check against violations of normality.

Note that the distribution-free confidence intervals include the value of 1.00 which might inspire erroneous enthusiasm. The reason is that these intervals define the upper bound as the 97.5th percentile, and it happened often enough that the test accuracy averaged out to 100% on the batches. The traditional confidence intervals are constructed using a t-statistic, although based on the perhaps unlikely assumption of normal accuracies. In any case, these intervals are to be taken with a grain of salt. All too often, machine learning research neglects reporting uncertainty around performance metrics. This experiment illustrates that in both a frequentist and a Bayesian scenario, it is possible to report a confidence interval alongside it, which makes for more complete reporting.

**Discussion**

One interesting phenomenon is that training accuracy is always about half the magnitude of that on the validation set in the first epoch of both Bayesian and frequentist AlexNets. This could very well be due to the use of dropout during training which becomes inactive during validation and explains the negative average generalization gaps as in equation (3.4). Similar results were obtained with dropout on during validation, however. This result is most curious, and we revisit the generalization gap in the next section, but the anomaly of a negative gap is perhaps due to the model itself and its tendency to also get 'stuck'. More time spent tweaking the model would be required, but this is satisfactory as a proof-of-concept.

As for initialization schemes, the frequentist network's weights are initialized using random draws from a $N(0,1)$ distribution, while the Bayesian network places $N(0,1)$ priors over weights, both as in [47] so as to not induce confounding factors through inconsistent weight-initialization distributions. Interestingly, using priors as opposed to sampling initial values could likely be a factor in reducing the frequency with which the network is stuck during training (additional 'robustness', perhaps?), although a more rigorous and conclusive analysis would be best left for future study.

An interesting result not to be glossed over is that frequentist testing was far more time-consuming, even more so than its own training (unusual, but not without explanation). The reason for this is that casting dropout at test-time suddenly implies processing each of the 10,000 test images through 10 different subgraphs (architectures). On the other hand, Bayesian testing is much faster perhaps because there is no need to pass the image through a unique network each time, but instead just samples different weight values which is done more efficiently from the learned normal posterior.

Finally, Figure 3.3 below shows the KL loss as defined in equation (2.3) during training for one of the training runs of the Bayesian AlexNet. It appears to exhibit a smooth exponential decay as training over 20 epochs unfolds, and in agreeing with the results of Table 3.2 above, the loss is already very low by the fifth iteration. Interestingly enough, the best results in the Bayesian training case were observed after around 5 epochs, which Figure 3.4 confirms: even when re-training the network, the accuracy would dip before begin climbing again beyond 5 epochs.

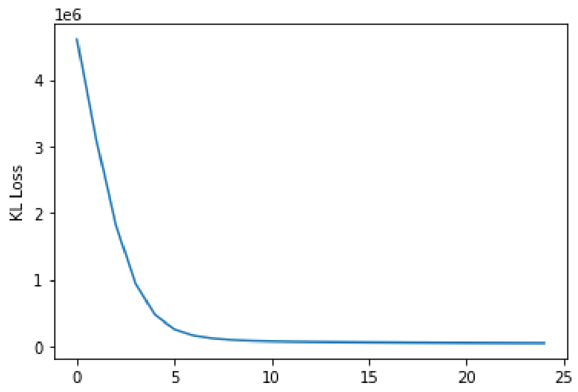


Figure 3.3: The smooth exponentially decaying nature of the KL loss used in *Bayes by backprop* for training the Bayesian AlexNet CNN in this experiment.

Figure 3.4 also captures the generalization gap during training and validation, showing that the two sets' performances seem to trade. These phenomena would make for interesting further research, but it is worth mentioning in closing that the simpler Bayesian LeNet [47] was more often 'user-friendly' (it did not, for example, get stuck during training nearly as much). There may be some interaction effect here between the network structure itself and the use of Bayesian inference, but this is beyond the scope of this thesis.

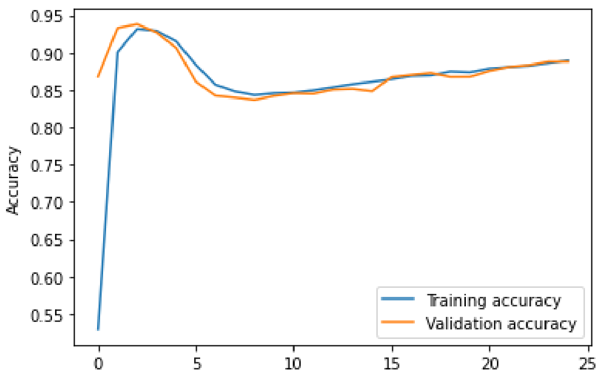


Figure 3.4: Despite a smoothly decaying KL loss, the Bayesian network displayed interesting behavior beyond 5 epochs.

### 3.6.2 *Modeling the Occam factor for a Bayesian CNN*

This experiment extends on the previous section and the mechanics of 3.4.3, where we take (this time) a Bayesian LeNet architecture and train it with variational inference and *Bayes by backprop*, while keeping track of *one* randomly selected filter weight $w_i^F$, and *one* randomly selected fully connected weight $w_j^{FC}$ during training (details for how they are chosen are found in the appendix). The marginal variational posteriors $q_{\boldsymbol{\theta}}\left(w_i^F \middle| \boldsymbol{w}_{-i}, D\right)$ and $q_{\boldsymbol{\theta}}\left(w_j^{FC} \middle| \boldsymbol{w}_{-j}, D\right)$ are tracked during training with regard to their evolution towards the unknown marginal posteriors for those weights. Note that it is possible to keep track of the average of all weights feeding into a fully connected layer's node, or the average of a certain filter's weights; this would however, retrieve the distribution of a sample mean (which is normal by the central limit theorem). We instead track one particular weight of each class to observe that respective distribution, with the assumption as in [47] of $N(0,1)$ priors.

We then conceptualize the Occam factor from chapter 1 for both classes of weights as $\Delta w/\Delta^0 w$ where the choice of entropy (width) measure is taken as the standard deviation of the respective variational posterior. Certainly, reduction in entropy as a result of training will result in $\Delta w/\Delta^0 w < 1$, but the question is whether this factor correlates well with the standard metrics for model selection that we see in practice (consider for example, accuracy or the generalization gap and whether these are well-correlated and therefore 'proxies' for this Occam factor). Recall that the Occam factor is based on Occam's razor- so that simpler models are preferred. Other metrics implicitly capture this preference, through variational inference's induced regularization effect. MacKay's original formulation of the Occam factor (recall section 1.3 and Figure 1.2) included all of $\boldsymbol{w}$- but we are well-convinced by now that this is quite impractical for such complex networks as CNNs, and this measure that I propose should serve as a suitable approximator.

Fifteen of the same networks are trained for 2, 5, 10, and 30 epochs each. The metrics that are considered are the test accuracy, the test loss, the generalization gap, the two Occam factors mentioned above, and the epistemic and aleatoric uncertainties. Training and testing across a variety of epochs ensures a broader representation of these metrics (reporting only on 30 epochs would only consider extensively trained networks, but we wish to generalize across a broader spectrum). Figure 3.4 below demonstrates that we may conceptualize the Occam factor as the ratio of the last standard deviation and the first (as a *reduction* in entropy of posterior to prior) of the weight in question.

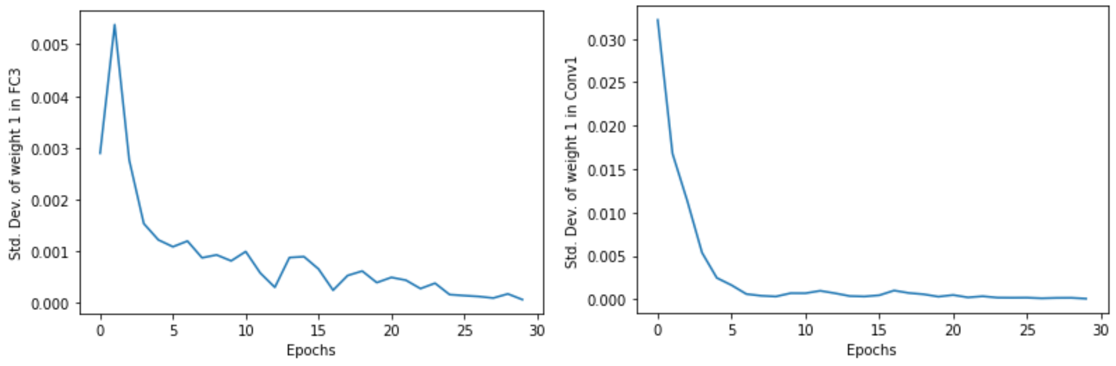


Figure 3.4: The standard deviations of both a randomly selected fully connected weight and a randomly selected filter weight, monitored over 30 epochs of training.

Based on the correlation matrix of Figure A.E.3.3, we find the correlations of Table 3.3 below. Correlations are reported with regard to Occam factors based on both classes of weights. Note that the generalization gap on average came out negative (test set performance was slightly better than in-sample performance, interestingly). With a smaller Occam factor (i.e. more confident network, which is quite supported by the moderate *positive* correlations with both Bayesian uncertainties) we observe a shrinking generalization gap (see Figure A.E.3.5) implying a closeness between performance on both sample and held-out data. Finally, we see moderate positive relationships between the Occam factors and the two Bayesian uncertainties, as well as moderate negative relationships with test accuracy, as one may expect (minus the magnitudes of the relationships). In conclusion, the use of alternative metrics on which to base *model selection* is not only justified by these findings, but recommended: the Occam factors here were only approximations (via single sampled weights, too) of the far more complicated thing, and this experiment served as a proof of concept that David MacKay [36] was already sowing the seeds for the use of uncertainty in model selection.

| Metric | Correlation ($w_j^{FC}/w_i^{F}$) |
|---|---|
| Test accuracy | -0.59/ -0.64 |
| Generalization gap | -0.67/ -0.77 |
| Epistemic uncertainty | +0.57/ +0.62 |
| Aleatoric uncertainty | +0.58/ +0.64 |

Table 3.3: Network metrics compared to the Occam factors of both classes of weights. The correlation coefficient between the two Occam factors is 0.62, which is moderate.

In closing, it is worth mentioning that a distinction was made between the two forms of weights when approximating the overall Occam factor in order to study whether the results would be robust to the choice of formulation. It seems to be the case that they are since Table 3.3 above shows consistency between correlations with the factors. Also, it is entirely possible to average the two into a single Occam factor, but I did not do that here since I did not wish to sacrifice this insight.

# Chapter 4

# Bayesian optimization

**Abstract**

We have seen up until this point how a Bayesian treats weights; that is, the network's *parameters*. Recall David MacKay's formulation of chapter 2, in which *hyperparameters* of the network were inferred (or marginalized over) depending on the goal. Previous chapters showed practical ways like variational inference to learn the posterior distribution over the weights, effectively using it at test time to make informed predictions. Although Bayesian optimization is quite a versatile algorithm as we will shortly see, one of its main applications is using Bayes' rule in an informed way to obtain optimal hyperparameters, thereby making it an inferential technique for *model selection*. As such, this chapter is dedicated to exploring a practical and efficient Bayesian approach for making design-related decisions for the complex models we have seen up until now in line with MacKay's second level of inference.

## 4.1 The idea

Bayesian optimization is a versatile derivative-free approach to global optimization of a black box objective function $f$ that leverages the principles of Bayesian inference. The approach is well-suited for all-too common functions that are expensive to evaluate (function evaluations may be limited due to time or resource constraints), lack any known structure (like concavity or linearity), and lack first- or second-order derivatives (preventing optimization by algorithms like gradient descent or Newton's method) [11]. Bayesian optimization falls under a broader approach called 'surrogate methods' as in [4] that seek to mimic this elusive $f$ via a so-called *surrogate* function, and an *acquisition* function by which inputs $\boldsymbol{x} \in X$ are suggested to evaluate $f$ and guide the search towards the global optimum, $\boldsymbol{x}^*$. Typically, $X$ is taken to be axis-aligned, like a hyperrectangle of dimension $d$ [3].

### 4.1.1 The formulation

Bayesian inference places a prior set of beliefs on the objective function $f$, and updates this set as more observations $D_{1:t} = \{x_{1:t}, f(x_{1:t})\}$ are collected according to a guided sampling of the feasible set $X$. With more of this data, the prior $P(f)$ is combined with the likelihood $\mathrm{P}(D_{1:t}|f)$ to produce the posterior $P(f|D_{1:t}) \propto P(D_{1:t}|f)P(f)$ as in chapter 1, which captures updated beliefs about the unknown objective [5]. Bayesian optimization differentiates itself from other surrogate methods in that it uses surrogates developed from Bayesian statistics (more specifically, the surrogate probabilistically summarizes the conditional distribution of the objective $f$ over data as $P(f|D_{1:t})$) and uses these surrogates to guide sampling. The entire process can thus be summarized as the updating of some prior beliefs about the objective $f$, where the updated posterior over $f$ is used in each iteration to guide the next sample according to the acquisition function. In this way, Bayesian optimization limits expensive evaluations of $f$ by selecting the next input based on those that have performed well in the past.

When it comes to optimizing $f$, there are typically two primary components, which until now were left vague. The first is the form of the surrogate function, and the second the acquisition function which suggests how to sample. With good reason that will be explained in section 4.2, a popular model for placing a prior on $f$ is the Gaussian process [58] as it provides a non-parametric inferential framework over (continuous) functions. This defines a setting known as 'GP regression', which requires that $f$ be continuous, although this is not generally troublesome in practice [11]. The second component consists of an acquisition function (forms for which are discussed in section 4.3) which depends on the surrogate and is maximized by some $\boldsymbol{x}'$, thus suggesting the next sample as $(\boldsymbol{x}', f(\boldsymbol{x}'))$. The dataset is then augmented to include this new observation, which in turn updates the posterior, and goes on iteratively as in Figure 4.1 below.

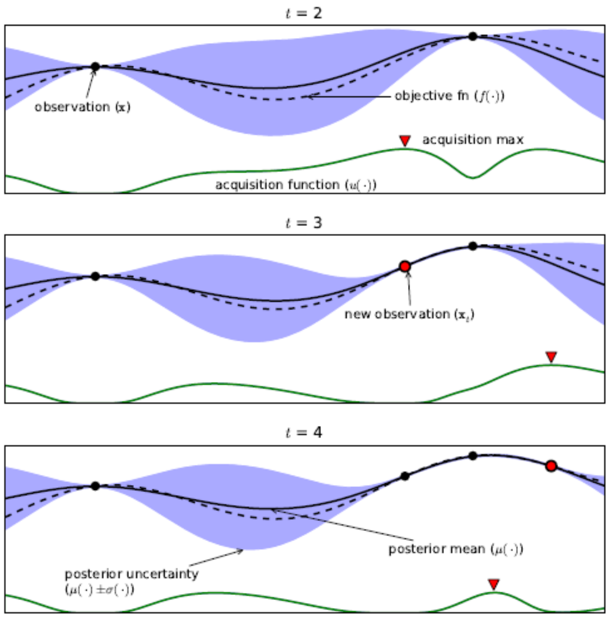


Figure 4.1: The ground truth is the dotted line. The solid line represents the mean surrogate function, with shaded regions representing Bayesian credible intervals. Over 3 iterations, as more observations are suggested by the green acquisition function, the posterior over the objective is updated, reflected by the modified mean (solid curve) and shaded Bayesian credible intervals (thinner now in areas that have been sampled). Image source [5].

### 4.1.2 History and previous work

Perhaps the earliest recognizable work on Bayesian optimization dates back to 1964 [34], when H. J. Kushner used Wiener processes for unconstrained optimization. His work even incorporated early aspects of reinforcement learning whereby he incorporated a trade-off parameter between 'more global' and 'more local' optimization, which today would be recognized as the exploration-exploitation phenomenon and discussed in further length in section 4.3**.** Later, in 1978, Mockus [41] *et al.* extended this to a multidimensional Bayesian optimization setting, using linear combinations of Wiener processes. Jones *et al.* in 1998 [29] successfully used Gaussian processes as surrogates in Bayesian optimization, which they referred to as *Efficient Global Optimization.*

Brochu *et al.* [5] summarize some interesting applications of Bayesian optimization across a variety of disciplines, including the training of a neural network to balance two different vertical poles on a moving cart [12], and sequential approaches for automatically tuning algorithm parameters using Bayesian optimization [27]. The algorithm is noted for its appeal and popularity in [4][5][58], with tutorial papers like [11][20] to make it accessible, although it is not without its faults (discussed at length in section 4.5). The model is deeply reliant on the specification of the surrogate (i.e. the parameterization of the gaussian process mean and covariance, details of which are in 4.2.2), which [5][11][58] warn can be disastrous if poorly selected. Wang *et al.* define a theoretical framework for GP hyperparameter specification using a cumulative regret bound, while others favour neural processes which they have shown to be faster and more flexible in certain applications [17].

Despite these shortcomings, the Bayesian optimization algorithm has still worked successfully in diverse applications, and when combined with the transparency of Gaussian processes (as opposed to neural networks like in [17]), it can be both interesting and powerful for small-scale experiments (like in section 4.4).

### 4.1.3 As a Bayesian model selection technique

Until now, the objective function $f$ was some general black box function. One application of Bayesian optimization is automated hyperparameter tuning, which can for instance take the objective function to be the *validation set error*. Therefore, for a given machine learning algorithm (such as a multilayer perceptron) and a given learning task, the problem consists of finding the hyperparameters that yield the lowest validation set error (which typically generalizes well to test set error when the validation data and test data distributions are alike [19]). Supposing in a simple example that a single-layer neural network model has as its hyperparameters *n* neurons in the hidden layer and *k* possible values of the learning rate for its optimizer, the problem consists of training *n*k* different models and selecting the model with the lowest validation set error. This makes for a practical and accessible inference over models as in the spirits of the second level of inference from section 1.3.

Traditional approaches like grid search and random search very quickly become prohibitive (especially for complex models like in chapter 3) and are also uninformed as they blindly test all possible specified hyperparameter combinations with no implicit bias for better combinations. In short, these methods absolutely disregard both a prior over models $P(H_i)$ and mechanism $P(D|H_i)$ by which to produce an updated search space over models as in equation (1.4). In experiment 4.4.3, random search is compared to Bayesian optimization for a multilayer neural network on handwritten digit recognition and is shown to be both *uninformative* (that is, future evaluations are independent of current ones) and also fails to outperform the latter.

The above example is meant to motivate Bayesian optimization as an informed hyperparameter search. This is arguably a model selection technique that leverages Bayesian inference by continually updating the posterior to concentrate more on promising hyperparameter combinations and perform less evaluations of poor such combinations. In section 4.4**,** experiments for finding the optimal hyperparameter set-up of a (frequentist) neural network function approximation are carried out in this manner.

## 4.2 Gaussian processes

### 4.2.1 As priors over functions

Recall from chapter 1 that when applying Bayesian inference, subjective beliefs come in the form of a prior distribution. A Gaussian process ('GP' for short) is a distribution over functions (as a Gaussian distribution is a distribution over random variables), and is entirely and precisely defined by its mean function $\boldsymbol{\mu}$(x) and positive semi-definite covariance matrix $\boldsymbol{\Sigma}$, so that functions are sampled as:

$$f(\boldsymbol{x}) \sim GP(\boldsymbol{\mu}(\boldsymbol{x}), \boldsymbol{\Sigma}(\boldsymbol{x}, \boldsymbol{x_0})) \tag{4.1}$$

GP regression therefore defines a Bayesian inference setting over functions, making samples from it suitable as surrogates [11], and is furthermore appealing for its closed-formed multivariate normal representations of prior and updated posterior distributions over this space of functions [45]. Mockus [40] laid the groundwork for GPs in 1994, with a set of 'natural and simple' conditions like continuity of the acquisition function (for more, see [5]). Finally, it is useful to think of a GP as returning a mean and covariance of a normal distribution for an arbitrary input $\boldsymbol{x}$, as opposed to returning a scalar $f(\boldsymbol{x})$ [5].

Through the updating of the posterior as in 4.1.1, it is possible to derive a closed-form posterior *predictive* distribution as well, which can be used during test time to model the unobserved function values. The following formulation is according to Rasmussen & Williams [45]. Assume that the input space $X$ gives rise to a finite collection of $N$ $d$-dimensional inputs (perhaps according to some partitioning experimental design) $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\}$ such that $\boldsymbol{x}_i \in \mathbf{R}^{\boldsymbol{d}}$ and we form $\mathbf{X} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_N] \in \mathbf{R}^{\boldsymbol{d*N}}$. Assume further that corresponding (possibly noisy) objective values $\{y_1, \dots, y_N\}$ are observed during training and vectorized as $\mathbf{y}$, where $y_i = f(\boldsymbol{x}_i) + \varepsilon_i$ with $\varepsilon_i$ normal noise according to $N(0,\ \sigma_y^2)$. As such, the (possibly noisy) training data is made up of $D = \{\mathbf{X}, \mathbf{y}\}$.

Given this set-up, before the training process begins, a prior belief can be defined over the possible function values as $P(\boldsymbol{y}\ ) = N(\mathbf{y}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$, and updated to the posterior after training on the $N$ labeled examples as $P(\boldsymbol{y}|D)$ according to Bayes' rule. Note that the prior's parameterization can often be $\boldsymbol{\mu} = \mathbf{0}$ (with impunity since the mean $\boldsymbol{\mu}$ can be added back after the prediction step [5]), but is more complicated for the covariance matrix $\boldsymbol{\Sigma}$ which governs the shape of the distribution and thus the characteristics of the function to be predicted [20]. When it comes to the test phase, $N_*$ input values are provided as $\mathbf{X}_{test} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_{N_*}] \in \mathbf{R}^{\boldsymbol{d} \times N_*}$ with *unknown* accompanying values $\boldsymbol{f}_{test}$. As such, Bayesian probabilistic inference over noiseless $\boldsymbol{f}_{test}$ is done via the posterior predictive distribution (4.2) below (which is $N_*$-dimensional multivariate normal) according to parameterization $\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*$, discussed in section 4.2.2 below.

$$p(\boldsymbol{f}_{test}|\mathbf{X}_{test}, D) = \int p(\boldsymbol{f}_{test}|\mathbf{X}_{test}, \boldsymbol{f}) p(\boldsymbol{f}|D) d\boldsymbol{f} = N(\boldsymbol{f}_{test}|\boldsymbol{\mu}_*, \Sigma_*) \tag{4.2}$$

### 4.2.2 Parameterization

For the prior distribution mentioned in the above section, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ were defined as the mean vector (constructed by evaluating a mean function $\mu_0 : X \to \mathbf{R}$ at each $\boldsymbol{x}_i$, generally taken to be zero-centred) and the positive definite covariance matrix (constructed by evaluating a covariance function or *kernel* $\kappa : X * X \to \mathbf{R}$ at each pair of points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$) [45][49].

The complete-data joint distribution is given by the multivariate normal distribution over the known training and inferred test data values, and its parameterization changes slightly. Note that this assumes $N_* \geq 1$, where [5] use $N_* = 1$ to decide a single $\boldsymbol{x}$ to sample next:

$$\begin{pmatrix} \boldsymbol{y} \\ \boldsymbol{f}_{test} \end{pmatrix} \sim N(\mathbf{0}, \begin{pmatrix} \mathbf{K}_{\boldsymbol{y}} & \mathbf{K}_{test} \\ \mathbf{K}_{test}^T & \mathbf{K}_{test,test} \end{pmatrix}) \tag{4.3}$$

This joint distribution has dimension $N + N_*$, and the components of the covariance matrix are:

$$\begin{aligned} \mathbf{K}_{\boldsymbol{y}} &= \kappa(\mathbf{X}, \mathbf{X}) + \sigma_{\boldsymbol{y}}^2 \mathbf{I} = \mathbf{K} + \sigma_{\boldsymbol{y}}^2 \mathbf{I} \\ \mathbf{K}_{test} &= \kappa(\mathbf{X}, \mathbf{X}_{test}) \\ \mathbf{K}_{test,test} &= \kappa(\mathbf{X}_{test}, \mathbf{X}_{test}) \end{aligned} \tag{4.4}$$

where κ is a measure of similarity between inputs and has the property that points closer in the input space are more strongly correlated (the choice of κ is thus a crucial determinant of the smoothness) [11] and is explored in experiment 4.4.2. Popular choices of kernel functions are the *power exponential* (also called *Gaussian kernel*), and its generalization the *Matérn* kernel (suggested in [49] to combat overly smooth regression) and are discussed in section 4.4 with the experiments. The choice of kernel function introduces its own hyperparameters (ironically), and methods for selecting them are discussed in 4.2.3 below. Finally, these components defined in (4.5) determine the sufficient statistics of the predictive posterior distribution $p(\boldsymbol{f}_{test}|\mathbf{X}_{test}, D)$ for $N_*$ new test cases:

$$\begin{aligned} \boldsymbol{\mu}_* &= \mathbf{K}_{test}^T \mathbf{K}_{\boldsymbol{y}}^{-1} \boldsymbol{y} \\ \mathbf{\Sigma}_* &= \mathbf{K}_{test,test} - \mathbf{K}_{test}^T \mathbf{K}_{\boldsymbol{y}}^{-1} \mathbf{K}_{test} \end{aligned} \tag{4.5}$$

where the expression for $\mathbf{\Sigma}_*$ incorporates a reduction in uncertainty as more training data is available, seen as the subtraction of the second term.

### 4.2.3 Covariance function

As was explained throughout section 4.2 and 4.3, the choice of GP parameterization and acquisition function are crucial to the modeling problem. When it comes to the kernel function of the Gaussian process, the assumptions of Mockus [40] have as conditions for convergence that the kernel must be positive semi-definite, and approach 1 for nearby inputs and 0 otherwise (i.e. points closer in the input space are more strongly correlated). The importance of an appropriate kernel is stressed in [5][11][58], for which a typically sensible choice is the Matérn kernel [11]. This kernel is a generalization of the popular (though sometimes overly smooth) squared exponential kernel and is the product of an exponential and polynomial of order $p \in \{0, 1, 2\}$. In its general form it is written as:

$$\mathbf{\Sigma}(\boldsymbol{x}, \boldsymbol{x}_0) = \alpha_0 \frac{2^{1-\nu}}{\Gamma(\nu)} (\sqrt{2\nu}||\boldsymbol{x} - \boldsymbol{x}_0||)^\nu K_\nu(\sqrt{2\nu}||\boldsymbol{x} - \boldsymbol{x}_0||) \tag{4.6}$$

where $K_\nu$ is the modified Bessel function and $\nu$ is the adjustable parameter that governs the smoothness of the kernel. The smaller the $\nu$ hyperparameter, the less smooth is the function (in the limit as $\nu \to \infty$, (4.6) simplifies to the squared exponential kernel). Finally, these hyperparameters have their own way of being optimized in the form of MLE or MAP estimation, while others marginalize over them via MCMC [11].

In the experiments that follow, the mean function is taken to be constant and the kernel's $\nu$ hyperparameter varied by hand, for the sake of time and simplicity according to the general guideline in [11] that $\nu = p + \frac{1}{2}$. This gives rise to three values $\nu \in \{\frac{1}{2}, \frac{3}{2}, \frac{5}{2}\}$ which are specific cases of this kernel that find common application, such as the first simplifying it to the *absolute exponential kernel*.

### 4.2.4 Bayesian uncertainty modeling with Gaussian processes

Note that for noiseless training, it is possible to set $\sigma_y^2 = 0$ although this assumption is too ideal in practice. Interestingly, in the context of Bayesian uncertainty measures for GP regression, this error noise term acts as a good measure of the *aleatoric* (data-inherent) uncertainty, while the variance $\sigma^2$ of the predictive posterior distribution for one new input $\boldsymbol{x}'$ is a good indicator of the epistemic (model-inherent) uncertainty [26] for each new sample.

In the event of noiseless training ($\sigma_y^2 = 0$), Frazier [11] builds the posterior distribution $P(f|D_{1:t})$ for inference over $f(\boldsymbol{x})$ for some new input, $\boldsymbol{x} = \boldsymbol{x}_{new}$ according to [45]:

$$\begin{aligned} &f(\boldsymbol{x})|f(\boldsymbol{x}_{1:n}) \sim N(\mu_n(\boldsymbol{x}), \sigma_n^2(\boldsymbol{x})) \\ &\mu_n(\boldsymbol{x}) = \boldsymbol{\Sigma}_0(\boldsymbol{x}, \boldsymbol{x}_{1:n})\boldsymbol{\Sigma}_0(\boldsymbol{x}_{1:n}, \boldsymbol{x}_{1:n})^{-1}\big(f(\boldsymbol{x}_{1:n}) - \mu_0(\boldsymbol{x}_{1:n})\big) + \mu_0(\boldsymbol{x}) \\ &\sigma_n^2(\boldsymbol{x}) = \boldsymbol{\Sigma}_0(\boldsymbol{x}, \boldsymbol{x}) - \boldsymbol{\Sigma}_0(\boldsymbol{x}, \boldsymbol{x}_{1:n})\boldsymbol{\Sigma}_0(\boldsymbol{x}_{1:n}, \boldsymbol{x}_{1:n})^{-1}\boldsymbol{\Sigma}_0(\boldsymbol{x}_{1:n}, \boldsymbol{x}) \end{aligned} \tag{4.7}$$

So that epistemic uncertainty can be used to construct Bayesian credible intervals around $f(\boldsymbol{x})$ (see Figure 4.3 in the experiments) as $\mu_n(\boldsymbol{x}) \pm 1.96 \cdot \sigma_n(\boldsymbol{x})$. When training is noiseless, the acquisition function will take its minimal value of 0 at previously evaluated points, because these points contribute nothing new to guide the search towards the optimum [11]. Regions that have larger credible intervals therefore create the opportunity for the acquisition to take larger values (if maximization is the goal; smaller otherwise), motivating the acquisition (i.e. the sampler) to explore new regions. Finally, another opportunity is created when the posterior mean is larger (as the mean function can be thought of as a point estimate of the objective function), motivating the sampler to exploit regions it knows. Together, these create an exploration-exploitation trade-off of the input space, discussed in the next section.

## 4.3 Acquisition functions

Since the optimization process relies on drawing successive samples $\boldsymbol{x} \in X$, acquisition functions leverage the conditional probability of locations in $X$ that are more likely to be extrema, thereby continually updating the posterior on $f$ with more and more meaningful samples [11]. Formally, the acquisition function can be written as $u(\boldsymbol{x}|D_{1:t-1})$ where $D_{1:t-1} = \{(\boldsymbol{x}_1, f(\boldsymbol{x}_1)), \dots, (\boldsymbol{x}_{t-1}, f(\boldsymbol{x}_{t-1}))\}$ is the history of samples and their values. The mapping $u : X \to \mathbf{R}^+$ allows a single subsequent sample to be proposed according to an implicit maximization of the form $\boldsymbol{x}_t = {}^{argmax}{}_{\boldsymbol{x}} u(\boldsymbol{x}|D_{i:t-1})$, making the acquisition *myopic* [5] (meaning that it does not look forward to some lengthier horizon, although current research seems promising). Finally, the acquisition function depends on the model through the predictive mean function and predictive variance function as in (2.2) [49]. The original prohibitive optimization on $f$ is thus replaced by a cheaper proxy optimization on $u$, sharing a common theme with the goal of variational inference in 2.4.

By drawing inspiration from reinforcement learning [5], an adjustable parameter $\xi \in [0,1]$ is incorporated in as $u(\boldsymbol{x}|\xi, D_{i:t-1})$ and acts as an exploration-exploitation trade-off. This parameter governs the extent to which samples search new locations for possible extrema (exploration of areas with high surrogate variance), while also staying true to promising regions of the sampling distribution (exploitation of areas with a high surrogate mean in the case of objective maximization) [5]. Its choice is crucial as too much exploration ($\xi \approx 1$) takes too many iterations, and too much exploitation ($\xi \approx 0$) finds *local* extrema, so that Kushner [34] recommends an adaptive schedule throughout training. Later and rather surprisingly, Brochu *et al.* [5] report that $\xi = 0.01$ worked well in all experiments and a cooling schedule that explores early and exploits later did not.

In practice, there are three generally used acquisition functions [5][49], each of which is described below and used later in experiments in section 4.4. For what follows, $\boldsymbol{x}_{best} = {}^{argmax}{}_{\boldsymbol{x}} f(\boldsymbol{x})$ is the current best sampled input $\boldsymbol{x}$, $\Phi(\cdot)$ is the standard normal cumulative distribution function, and

$$\gamma(\boldsymbol{x}) = \frac{f(\boldsymbol{x}_{best}) - \mu(\boldsymbol{x}; \{\boldsymbol{x}_n, \boldsymbol{y}_n\}, \boldsymbol{\theta}) - \xi}{\sigma(\boldsymbol{x}; \{\boldsymbol{x}_n, \boldsymbol{y}_n\}, \boldsymbol{\theta})} \tag{4.8}$$

acts as a standardized input to the acquisition function where $\boldsymbol{\theta}$ is the parameterization of the Gaussian process (i.e. the choice of mean and kernel along with their hyperparameters), which gives acquisitions of the form $u(\boldsymbol{x}|\xi, D_{i:t-1}, \boldsymbol{\theta})$ to incorporate dependence on the GP set-up.

### 4.3.1 Maximum probability of improvement

Perhaps the most intuitive approach [34] is to propose the next sample by maximizing $u(\boldsymbol{x}|\xi, D_{i:t-1}, \boldsymbol{\theta}) = P(f(\boldsymbol{x}) \geq f(\boldsymbol{x}_{best})) = \Phi(\gamma(\boldsymbol{x}))$. Incorporation of $\xi$ prevents the otherwise purely exploitative nature, whereby points that provide infinitesimal improvement over $f(\boldsymbol{x}_{best})$ with a great probability will be preferred to points with greater improvement, but less certainty [5].

### 4.3.2 Expected improvement

As an extension to the above acquisition function, this approach not only captures the probability of improvement, but the magnitude of that improvement that a suggested point can yield [5]. The goal thus becomes to minimize the expected deviation from the true maximum $f(\boldsymbol{x}^*)$ when sampling according to:

$$\boldsymbol{x}_{t+1} = {}^{argmin}{}_{\boldsymbol{x}} \mathbf{E}(\|f_{t+1}(\boldsymbol{x}) - f(\boldsymbol{x}^*)\| | D_{1:t})$$
$$= {}^{argmin}{}_{\boldsymbol{x}} \int \|f_{t+1}(\boldsymbol{x}) - f(\boldsymbol{x}^*)\| \, P(\boldsymbol{f}_{t+1}|D_{1:t}) df_{t+1} \tag{4.9}$$

with the added bonus of being able to look ahead by applying it recursively (as a fix to the myopic problem mentioned in 4.3), but this becomes expensive. Mockus *et al.* [41] thus suggest maximizing the expected improvement with respect to the current best value $f(\boldsymbol{x}^{+})$:

$$\boldsymbol{x}_{t+1} = {}^{argmax}_{\quad x}\mathbf{E}(\{0, f_{t+1}(\boldsymbol{x}) - f(\boldsymbol{x}^{+})\}|D_{1:t}) \tag{4.10}$$

where the argument of the expectation is the improvement $I$ and is positive only if the new sampled value is larger than the current best one (in the case of maximization). Mockus *et al.* make use of the parameterization of the GP process (its posterior mean and variance) to express the likelihood of improvement $I$ function as a normal distribution which is integrated over to yield the analytic solution (full details in [5]). Unlike in [5] however, the application of this acquisition function will incorporate the exploration-exploitation trade-off $\xi$ through $\Phi(\gamma(\boldsymbol{x}))$:

$$\begin{aligned} EI(\boldsymbol{x}) &= (\mu(\boldsymbol{x};\{\boldsymbol{x}_n,\boldsymbol{y}_n\},\boldsymbol{\theta}) - f(\boldsymbol{x}^{+}))\Phi(\gamma(\boldsymbol{x})) \\ &\qquad + \sigma(\boldsymbol{x};\{\boldsymbol{x}_n,\boldsymbol{y}_n\},\boldsymbol{\theta})\phi(\gamma(\boldsymbol{x}))\ if\ \ \sigma(\boldsymbol{x};\{\boldsymbol{x}_n,\boldsymbol{y}_n\},\boldsymbol{\theta}) > 0, \\ EI(\boldsymbol{x}) &= 0\ otherwise \end{aligned} \tag{4.11}$$

### 4.3.3 GP upper/lower confidence bound

A method with another mechanism that incorporates reinforcement learning in the form of *regret* is by Srivinas *et al.* [50]. This exploits lower confidence bounds in the case of objective function minimization (and upper in the case of maximization) and defines the acquisition:

$$\alpha_{LCB}(\boldsymbol{x};\{\boldsymbol{x}_n,y_n\},\boldsymbol{\theta}) = \mu(\boldsymbol{x};\{\boldsymbol{x}_n,y_n\},\boldsymbol{\theta}) - \alpha\sigma(\boldsymbol{x};\{\boldsymbol{x}_n,y_n\},\boldsymbol{\theta}) \tag{4.12}$$

with the second term added in case of the *upper* confidence bound for maximization. Here, α plays the role of $\xi$ but is a function of the instantaneous regret $f(\boldsymbol{x}^{*}) - f(\boldsymbol{x})$, with a full development in [50] and summary in [5].

Now we have seen all of the components of Bayesian optimization as an informed model selection technique (again, through its applications in hyperparameter tuning). In the experiments that follow in section 4.4, the acquisition function is maximized using the BFGS algorithm, the default with the libraries used here (see the appendix for these experiments).

## 4.4 Experiments and applications

### 4.4.1 *Bayesian optimization for a noisy function*

As a first demonstration of the capabilities of the Bayesian optimization algorithm, values of a 1-dimensional function are observed with noise and the goal is to find its global maximum on $[-1,1]$ which is $(0.666, 1.426)$. Though the objective function $f$ is known to be $f(x) = -\cos(5x) + 0.4x^2 + 0.4x$ in this experiment, it is treated as a black box since only the noisy observations $(x, f(x) + \varepsilon)$ are known, where $\varepsilon \sim N(0, \delta)$ taken with varying variance $\delta$ independently of $x$. Note that it is very possible to simulate a heteroskedastic setting in which $\varepsilon \sim N(0, \delta_x)$ however this introduces a distribution on $\delta_x$ which would be decided ad hoc (quite possibly, arbitrarily!) and complicate the process even more.

For full details of the implementation, see this experiment in the appendix. The exploration-exploitation trade-off parameter is taken as 0.01 as per suggestions of [5] (higher values did tend to settle in local extrema, as expected), and Figure 4.4 interestingly captures the relationship between epistemic (reducible with more data) uncertainty and exploration. Recall that the aleatoric uncertainty is data-dependent and is well approximated by the variance of noise infused into the observations [26], which remains indifferent to each new input throughout this problem (and is thus homoscedastic). The following plots demonstrate one go at maximizing $f(x)$, with details under Figure 4.2.

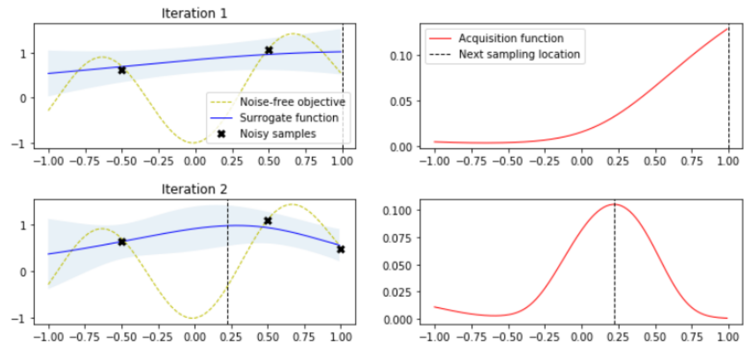


Figure 4.2: Each sample is selected to maximize the acquisition function, which is continually updated (see Figure 4.1). The shaded credible intervals capture aleatoric and epistemic uncertainties. The GP regression uses the Matérn kernel with $\nu = 2.5$, expected improvement acquisition function with exploration-exploitation trade-off of 0.01, $\varepsilon \sim 0.1 * N(0,1)$ and is run for 20 iterations. Average epistemic uncertainty is 0.116 and 0.093 in these two iterations.

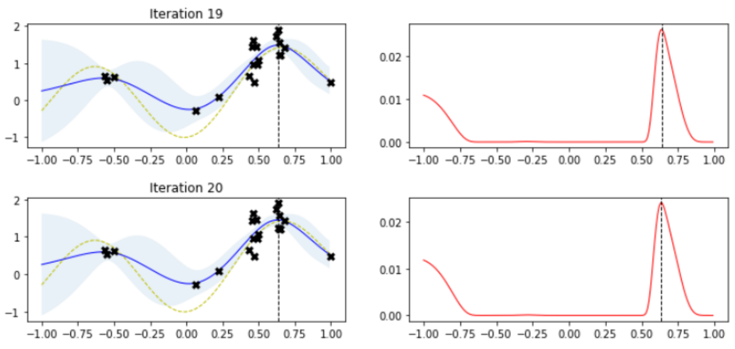


Figure 4.3: After 20 samples are taken, the acquisition function shows a reasonable extent of convergence, seen by the mass of samples close to this value. On occasional re-runs of the experiment, convergence would not be reached in 20 iterations, as it tended to exhibit more exploration of the domain space. The sampler is ‘self-aware’ in the primitive sense that it exhibits high epistemic uncertainty in areas rarely explored (i.e. it quantifies what it does not know). Average epistemic uncertainty decreases to 0.016 and 0.019 in these last iterations, and the optimum is returned as (0.686, 1.621). Quite close!

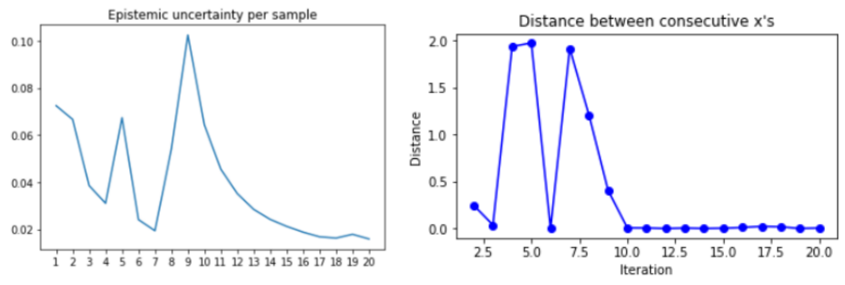


Figure 4.4: The epistemic uncertainty is higher in regions that have not yet been exploited (like in experiment 2.7.1). The two plots exhibit similarities as Figure 4.3 more succinctly explains: when $x$ values are sampled close to each other, data-inherent aleatoric uncertainty begins to dominate the uncertainty decomposition, until epistemic uncertainty is minimal and tapers off as convergence is reached.

This experiment thus reinforces the importance of guided exploration, where epistemic uncertainty affirms the notion that although more data is always good, more *relevant* data is better. As the additive noise term increased, so did the average epistemic uncertainty. The aleatoric uncertainty did not, although it consistently underestimated the variance of the additive noise term (see Table A.E.4.2).

### 4.4.2 *The importance of the acquisition function and GP parameterization in hyperparameter tuning*

Sometimes, a neural network may achieve excellent generalization, but the modeler may wish to not settle and try finer adjustments that may provide a marginal improvement in performance (oftentimes desirable in fields such as medical imaging). The network can still be tuned by applying Bayesian optimization as an informed search over the space close to the believed optimal values, and this experiment does exactly that, for the three most common choices of acquisition function presented in section 4.3. The appendix section A.4.4.2 gives the details of training and optimizing, including the bounds of the hyperparameters.

The most interesting result of Table A.E.4.4 is that the performance metrics appear to be relatively robust to choice of acquisition, but the optimized hyperparameters themselves vary vastly (although there is arguably more similarity as suggested by the EI and the MPI acquisitions, which are formulated quite similarly and differently from the LCB acquisition). During the Bayesian Optimization search, the best performance is achieved by the EI acquisition (also supported by Figure 4.5 below), where the accuracy is 0.2% higher than the others (arguably negligible, but in some domains like medical imaging, everything matters) and the loss is 0.7% lower; so the EI is chosen to continue experimenting with the parameterization of the GP regression, through modifying $\nu$ in the Matérn kernel. Note however that at test-time, casting dropout 10 times for each prediction gives us 95% confidence intervals on the accuracy (as in experiment 3.6.1) and these are reported in the tables as well.

This distinction in accuracies is an interesting one, because in the former case, we are using the accuracy as the metric by which to select the network's architecture. Once that is selected, predictions are made using dropout to estimate uncertainties as well as to report an average accuracy (more robust when averaging) for these hyperparameters. The former accuracy seems to overestimate the network's true accuracy, because in all cases in Table A.E.4.4, this single point estimate exceeds the upper bound of the 95% confidence interval.

The two figures below make it clear that Bayesian optimization is indeed a *guided* search, as there is a (slight) negative trend in the loss, and a (slight) positive trend in the accuracy (the next experiment statistically tests for these). In the case of random search or grid search, there could hardly be a justification for on-average improvement with more iterations: every iteration is independent of the previous (see the next experiment for this too). The uncertainty measures are reported too, and a negative correlation between uncertainty and accuracy can be observed.

Using just the expected improvement acquisition function (for simplicity, but also for its seemingly better performance in this and in the following experiment), the GP parameterization is now varied through the kernel's smoothness hyperparameter $\nu$ and results shown in Table A.E.4.5.

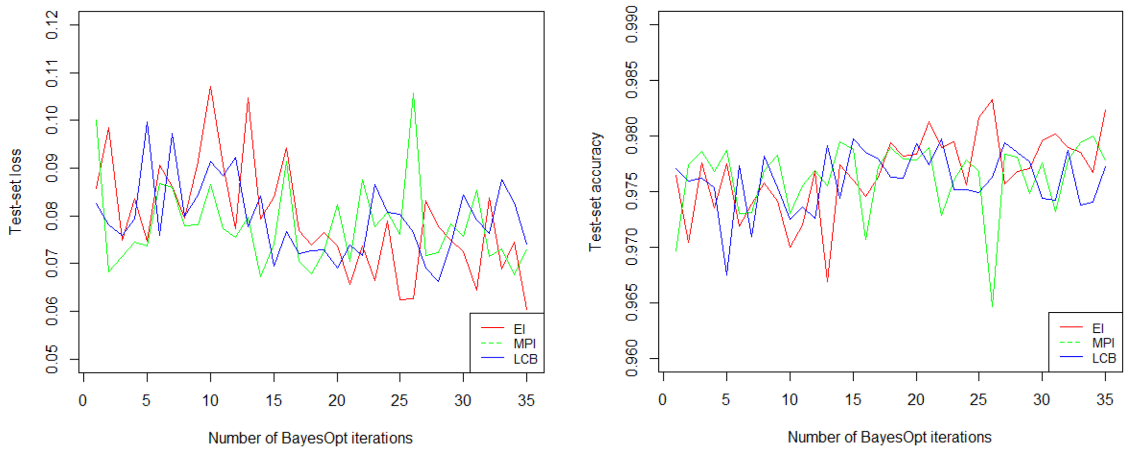


Figure 4.5: The choice of acquisition function has an effect on the model's performance. 35 is a rather ample amount of iterations in practice as it implies 35 different models and is involved. All acquisition functions are encouraged to explore by setting the exploration-exploitation parameter $\xi = 0.01$. The LCB was designed to incorporate exploration even without this $\xi$ parameter, although it appears to be the least volatile. Over the last several iterations, it appears that the EI acquisition leads the pack; amongst the three, it has the smallest loss and the greatest accuracy.

The best Bayesian Optimization performance resulted from $\nu = 1.5$ for the EI acquisition function. Amongst all rule-of-thumb values for $\nu$ (suggested in section 4.2.3), this value gives surrogate functions that are least smooth— perhaps this allows the sampling of more 'complex' functions as surrogates for the naturally bumpy and irregular categorical cross entropy loss function. At test time, however, a choice of $\nu = 3.5$ gives the confidence interval with highest average accuracy and lowest uncertainties. This corresponds to the smoothest surrogates, perhaps suggesting better generalization across the ensemble.

As a final check, Table A.E.4.6 gives the minimal and maximal times taken for Bayesian optimization to run for 35 iterations. Some scenarios led to wider variation in optimization times such as using an EI acquisition with $\nu = 2.5$, (from 17 minutes 23 seconds to 32 minutes 58 seconds), or the MPI acquisition with $\nu = 3.5$ (from 14 minutes 33 seconds to 26 minutes 57 seconds). Performance metrics were not seen to be significantly better despite the extra time taken to tune hyperparameters, and results were shown to be consistent in tables A.E.4.4 and A.E.4.5 above for the different approaches. This suggests some form of experimental design in which time is saved by early stopping when some desired target performance is reached.

### 4.4.3 *Bayesian optimization as an informed alternative to random search*

Continuing from experiment 4.4.2, a closer (and more rigorous) look is taken at the merits of Bayesian optimization over simpler approaches like random search. At its core, random search is attractive because of its trivial application (coded here from scratch— the same cannot be said for Bayesian optimization), and typically much faster. However, as was mentioned in the previous experiment, an *informed* approach should demonstrate some desirable trend as more iterations are carried out, which is theoretically impossible with random search (all iterations are independent since hyperparameter values are sampled arbitrarily and thus *blindly*). To that end, Figure 4.6 shows the test-set performance metrics with the same network and same dataset from the previous experiment. Random search is run for the same number of iterations here.

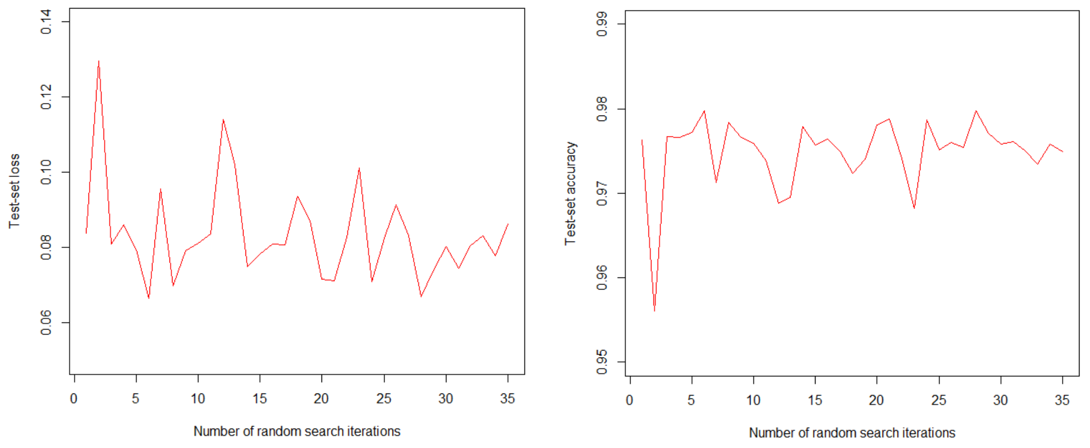


Figure 4.6: A randomly selected run of random search is presented here. Treated as time series, these two graphs reflect the sampling theory of random search: iterations are independent, and thus the time series should resemble 'white noise' (serially uncorrelated random variables, taken to be the loss and accuracy) albeit with non-zero mean. It should also be 'stationary', which in simple terms means that it must exhibit time-indifferent mean and variance.

As a rigorous check, the loss and accuracy metrics are treated as time series (for both Bayesian optimization and random search), and the hypothesis is that the former will exhibit appropriate trends depending on the metric, while the latter will not. See the appendix for the full details on the models and the hypotheses.

According to Table A.E.4.8, the respective tests for white noise and stationarity show that the hypothesis for random search is substantiated— that is, the time series of Figure 4.6 is indeed stationary (thus uninformed) for both metrics. However, Bayesian optimization is considered non-stationary at the 1% level for all choices of acquisition, with expected improvement much more strongly rejecting the null of stationarity.

This is a most curious result, as it implies that the choice of acquisition is crucial, and this experiment acts as a proof of concept that time spent designing and trying out different parameterizations of Bayesian optimization will be well warranted. Not only the time taken to implement Bayesian optimization is longer, but the time to design it is time that random search can use in trying out hyperparameter combinations to actually solve the problem. While random search suggested hyperparameters that came short in terms of accuracy (arguably by a negligible 0.22% in the worst case), it beat all in terms of loss except for Bayesian optimization with expected improvement. Of course, this result is dataset-specific, and might change with more iterations. More iterations can help Bayesian optimization converge towards a solution as seen in experiment 4.4.1, but more iterations for random search will offer no such guarantee; at best, it will provide it with more opportunities to blindly stumble close to the global optimum.

**4.5 Limitations of Bayesian optimization**

In closing of this chapter on Bayesian optimization, it is important to discuss limitations of these techniques, otherwise this would seem like a sales pitch. To begin with, since Bayesian optimization optimizes the objective function with respect to input $\boldsymbol{x} \in \mathbf{R}^d$**,** generally proving a successful approach when $d \leq 20$ [11] and has been found to perform comparatively worse otherwise. What is worse, Bayesian optimization has been difficult to parallelize (unlike random search, which is trivial) though recent research looks promising [5][11]. The core of the GP regression that is in turn the core of Bayesian optimization is the choice of parameterization. Throughout the experiments, the mean function was taken to be constant (typical in practice [5][11]) and the Matérn kernel was used as it is more general than the commonly-used squared exponential kernel, allowing for a varying scale parameter $\nu$ that adjusts the smoothness and thus quality of fit.

As an extension of the work here, it may be worthwhile to try automated GP parameterization techniques (one cannot escape from hyperparameter tuning, it seems) as in [6], or MLE/MAP estimates over the hyperparameters as in [11]. This approach extends beyond the scope of this work, however. Other acquisition functions like *knowledge gradient* exist too [5], but the ones used here are most typical, and performed rather well in experiments 4.4.1, 4.4.2, and 4.4.3, despite being *myopic* (acquisition functions that samples sequences as opposed to single values are possible, though also beyond the scope of this work). Furthermore, the acquisitions can be maximized by different means (like *DIRECT* [5]), but the BFGS algorithm fared well. While on the subject of acquisition functions, experiment 4.4.3 showed that the expected improvement acquisition fared best against random search, but the time taken to discover that could have been used running random search and actually obtaining results.

Finally, the normality assumption of Gaussian processes (i.e. that observed values $[f(\boldsymbol{x}_1), \dots, f(\boldsymbol{x}_k)]$ are multivariate normal) is modified, with some authors [46] opting for student's t processes as generalizations of GPs, while other authors like Garnelo *et al.* [17] opt for distribution-free approaches, introducing *neural* processes- finding a more flexible and efficient model. These modeling techniques are also beyond the scope of this work, which is aimed at exploring the more transparent approaches passed on over time. Overall, this chapter's main concern was in motivating and illustrating Bayesian optimization as a practical way to perform

MacKay’s second level of inference for model selection, as a tractable alternative to using the Occam factor (which itself only applies to Bayesian models like in experiment 3.5.2- however, we have seen how this approach leverages Bayesian inference even for frequentist models).

# Chapter 5

# Final thoughts and conclusion

## 5.1 Demonstrating Bayesian overfitting with an application to COVID-19 imaging data

Before any formal conclusion of this thesis, it would be a good idea to apply these tools to a real-world (and contemporary!) problem. Publicly available COVID-19 chest CT-scan imaging data is exceptionally rare (although few public repos like the one I use here by Cohen *et al.* [8] seem to be collecting and updating a variety of different images, and by variety I mean varying 'views' and pathologies, which aside from COVID-19 are categorized into the 'other' class). Any machine learning model is therefore immediately handicapped by this shortage of data, and what is worse, such datasets come with severe class imbalances (for example, in classifying 'COVID-19' vs. '*not* COVID-19', we are more than likely to find significant deviation from a 50-50 split). Such situations will undoubtedly see the complex convolutional networks overfit: far too many parameters for the few data.

Indeed, in the dataset used here [8], we train on 148 images (112 of which are 'COVID-19' and 33 are of type 'other') and test on a balanced set of 60 images. Note that this implies a baseline accuracy of 75.67% to beat, since the network can achieve this just by outputting a positive COVID-19 prediction every time. This statistical discrepancy between training and test set distributions is not without justification- indeed class imbalance is common in medical imaging [2] but the test set's distribution is largely elusive, so the assumption of balance is a practical one. No validation set is used here as data is already scarce enough.

For an example of the type of imaging data used for this application, see Figure 5.1 below which shows the CT scans of both 'COVID-19' and 'other'. Note that before the images are fed into the classifier network, they are transformed and pre-processed (see the appendix for more details).

The network of choice is a Bayesian CNN with a custom architecture (specifics of which are all in the appendix), and the performance metrics are reported in Table 5.1. By training the network 10 times for 30 epochs, we can construct distribution-free 95% confidence intervals on accuracies, uncertainties, and losses. An approximate Occam factor is also reported, as was done in chapter 3 in order to model the reduction in entropy of the prior. Immediately obvious is the poor performance that may leave one wondering why this is even reported. Well, the *frequentist* CNN of identical architecture hardly fared any better even with several different considerations and model selection techniques seen in chapter 4. The networks are certainly overfitting on all 10 reruns (indicated by the combination of high generalization gap and very low Occam factor) despite the architecture arguably not being overly complex (even relative to the ones used on MNIST in previous sections). A simpler network with less convolutions in my view would not fare well in learning rich features, while a more complex one would only increase the generalization gap.

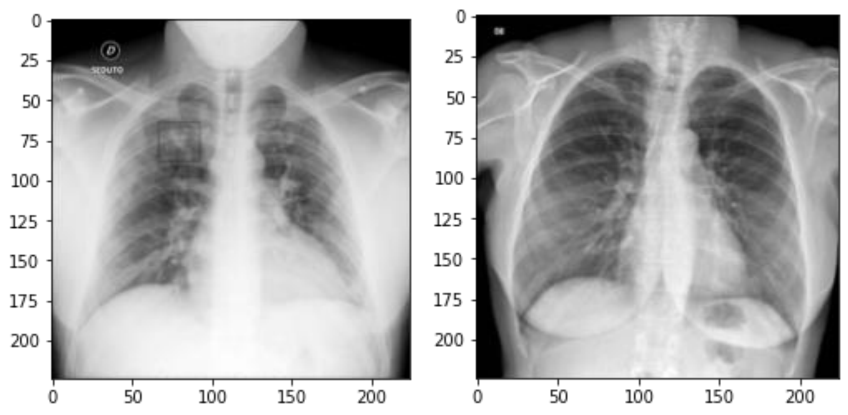


Figure 5.1: Grayscale image of COVID-19 on the left; image of ‘other’ on the right. Note the heterogeneity that exists between the classes here (in other words, the image on the right is structurally unlike the one of COVID-19 on the left, with certain features and lighting conditions unrelated to the pathology that the network needs to discern). Such heterogeneity often remains a problem even with large datasets [19], often ‘confounding’ the true pattern to be learned. Interestingly, *adversarial* training (that is adding noise, here in the form of rotating and flipping) injected excessive noise that ruined both the Bayesian and frequentist networks’ performances. An interesting result, which I suspect is simply due to the small data set size and over-parameterized networks, or perhaps an interaction of the two. Image credit: [8].

Therefore, if neither the model nor the approach is the problem, then both of the data’s shortage and imbalance could very well be the culprit. This is also consistent with the much-higher aleatoric uncertainty (compared to epistemic) which recall, is data-driven. In line with model selection techniques, this data uncertainty is irreducible, but the epistemic uncertainty can disappear with more data and this effect can be explored in order to guide the design of the network. In technical terms, the evidence term $P(D|H_i)$ of chapter 1 ought to be inversely related to epistemic uncertainty, and positively related to the size of the training set given an appropriate network $H_i$.

| Metric | Dist’n-free 95% conf. interval | Median |
|---|---|---|
| Train accuracy | (66.87%, 74.52%) | 72.54% |
| Test accuracy | N/A | 46.88%* |
| Avg. Epistemic | (0.0312, 0.03915) | 0.0348 |
| Avg. Aleatoric | (0.2316, 0.2955) | 0.2755 |
| Approx. Occam factor | (0.0010, 0.0559) | 0.0056 |

Table 5.1: Distribution-free confidence intervals on performance metrics of the custom-built Bayesian convolutional neural network on the public COVID-19 CT scan imaging data averaged across 10 models. *Test accuracy is a constant across all 10 Bayesian CNNs.

Note that the baseline is not beaten even with the training set, and Table A.E.5.4 shows that training loss and test loss (recall, these are measured by KL divergence) are still very high, while the median Occam factor in Table 5.1 above is small. This indicates that the $N(0,1)$ prior has been updated to a high degree, but not in the desired way. As mentioned earlier, the frequentist counterpart to the network used here did not fare much better- training across 30 epochs as well, (and for several re-runs), we observe a generalization gap of 31.25% (see Table A.E.5.3) with a test accuracy of 50%. This is no better than always outputting the same class, so the network is drastically overfitting. The frequentist network does, however, achieve higher accuracies on both training and testing sets, for what it is worth.

As an interesting application of the Bayesian network, we may track input-specific epistemic uncertainty and set those images aside that create a higher such uncertainty for the radiologist to examine. Additionally, *confusion matrices* and sensitivity/specificity metrics may be considered as the problem is one of non-equally perilous misclassifications (a false negative for COVID-19 may likely be far more serious than a false negative for the other class). To that end, see the appendix, however I have omitted these measures here because of the network's overfitting making them rather useless (sensitivity 1, specificity 0 together implying that the network always detects positive examples and never negative ones).

Nonetheless, this application is still an interesting proof-of-concept that if more publicly available data were compiled, an actually useful network may be possible in the first level of inference (i.e. one with existence of viable weights $\boldsymbol{w}$), and suddenly hyperparameter tuning strategies like Bayesian optimization seen in chapter 4 can guide us in practical and informed model selection like in the second level of Bayesian inference. In closing, a case could be made that machine learning is of help in times of pandemic, and looking forward, it could be well-poised to play a role in diagnostics when healthcare systems are overwhelmed.

## 5.2 Main contributions

In section 1.5, a quick overview of the contributions of this thesis was introduced on a surface level. Now that we have reached nearer to the end, we may re-state them in richer detail. The bigger picture of this work is an overview of Bayesian learning that takes the reader from the basic ideas to their development and application in more complex deep learning models like the CNNs of chapter 3, and automated hyperparameter tuning (i.e. model selection) strategies of chapter 4. Specifically, these Bayesian inference methods are framed in two levels as in David MacKay's work [36] where the first level infers the weight posterior (using variational inference) while the second concerns itself with model selection (where I suggest Bayesian optimization). Additionally, this thesis also suggests an approximation to MacKay's Occam factor (which he proposes as a global measure) in the all-too-common scenario of prohibitive deep networks by sampling and monitoring the reduction in entropy of a network weight.

Interestingly, we have seen also how Bayesian uncertainties correlate with this Occam factor approximation in experiment 3.6.2 and shown that a Bayesian network may still overfit despite a low Occam factor and even a relatively low epistemic uncertainty. In the case of small (and more complex) datasets such as the COVID-19 imaging one seen in this chapter, we have explored and demonstrated that while Bayesian techniques may be touted for working better on smaller datasets [18], this was not the case. Overfitting may plague these models just as often as it does the frequentist ones, though initialization schemes may be more robust by grace of their distributional, rather than point-estimate, assumptions. When it comes to model uncertainty, we have also shown the merits of reporting with confidence intervals (and in general, distribution-free ones based on percentiles should be used so as to avoid further assumptions like normality of accuracies across runs or weight-resampling), as well as showing that even frequentist networks may be equipped with uncertainties at test-time by casting dropout as in [15].

Finally, by framing MacKay's second level of Bayesian inference in more practical terms (such as with respect to CNN architecture, generalization gap, and Bayesian optimization as a model-selection technique) we have materialized these more abstract notions and nuances, and practically demonstrated the importance of thinking 'in stages'. We saw in chapter 4 that the automated and informed Bayesian optimization comes with its own hyperparameters (Gaussian process mean and kernel, choice of acquisition function, etc.) and the experiments demonstrated the importance of each in turn. These considerations are also a crucial ingredient in the second level of inference.

## 5.3 Limitations

Every study comes with its limitations and seeing that there are a series of studies here, a discussion is well-warranted. A word on the programs used for experiments: the construction, training, testing, and comparing of neural networks are all done in Python using PyTorch on Google Colab's cloud-based GPU services. These are free and come with time and resource constraints making only small-scale experiments feasible. As a result, most of the networks here were trained on MNIST imaging data since it is easily accessible and served well for proof-of-concept trials. Note that *R* was used for all statistical analyses (like correlation plots, regressions, time series models, summary statistics). Unfortunately, a fully Bayesian approach (Bayesian CNN with Bayesian optimization for hyperparameter selection) is very costly, and while standardized libraries exist for *BayesOpt*, implementation still remains very *ad hoc*. Even the first level of inference (training of a single network to learn the weight distribution) often timed out in Google Colab, so that simpler specification or manual supervision to prevent timing out was necessary.

As to content, many concepts were explored in passing (like Bayesian generalization gaps in chapter 2 and neural processes in chapter 4), either because they are still being researched or are beyond the scope of this paper. Nonetheless, they are included as very relevant to keep in mind and to provide a richer context to the paper.

Finally, some technical limitations are worth mentioning as well. Experiment 3.6.1 saw a comparison of a Bayesian CNN and frequentist CNN- where the frequentist AlexNet CNN architecture used out of the box [47] would much more frequently become stuck and not update the weights, while its Bayesian counterpart did so less often, while the LeNet architecture fared quite well on both accounts. An intuitive explanation was provided in that experiment, but this was only 'educated guessing' as to what was going on. Bayesian CNNs are still black boxes, but this thesis aimed at demystifying them a little bit (see sections 3.2 and 3.3) but more work is still necessary especially in making them more accessible on a grander scale. Of course, before libraries are created for them (as are easily available through PyTorch for frequentist CNNs for example) people must appreciate their worth, which this thesis aims to do.

## 5.4 For future study

While this work covers a rather vast body of concepts and draws on lots of literature, each topic could be extended for further research. For example, larger-scale experiments may likely provide a more practical and contemporary demonstration of whether these Bayesian methods are really all that. A fully-Bayesian approach that seamlessly combines level 1 and level 2 of inference [36] will be an interesting follow-up, and so too will experiments with alternate considerations beyond the scope of this work (such as neural processes instead of Gaussian processes [17]). More serious hardware would also be well-warranted, or a subscription to cloud-based services for the necessary boost in computing power.

Finally, as section 5.1 shows, the benefits of the Bayesian approach (like Bayesian uncertainties) are well-warranted in medical imaging where each classification can be accompanied by decomposable prediction uncertainty (model- and data-driven). To that end, larger public repositories of imaging data (especially in times of pandemic) would invite a larger host of machine learning enthusiasts and researchers to contribute creative solutions (note that medical imaging is naturally difficult to release to the public, so this may be a stretch). Also, diagnostic performance is unknown until these technologies are implemented in clinical practice [2].

*In closing*, this thesis is a self-contained overview that goes into detail where appropriate on the more recent and growing interest in Bayesian techniques for neural networks. From two levels of inference that consider both the network in isolation and relative to others [36], to variational inference for learning posterior distributions [3], to modelling Bayesian network uncertainties [47] (with approximations for frequentist networks [15]), and finally to Bayesian hyperparameter tuning [5], this work has displayed the merits of these techniques through a collection of small-scale experiments and applications to contemporary problems.

# Appendix

## A.A Formulas and derivations

### A.A.1.4 *Neural networks as functions*

Based on the simple formula:

$$f(\boldsymbol{x}) = f_L(\dots f_2(f_1(\boldsymbol{x}; \boldsymbol{w}_1); \boldsymbol{w}_2) \dots), \boldsymbol{w}_L) \tag{A.1.1}$$

we see that a function $f_i$ is applied at each layer of the network. This merits explanation because these functions were left rather abstract in the first chapter. When the multi-dimensional input $\boldsymbol{x}$ first enters the model, its matrix product is taken with $\boldsymbol{w}_1$ and an additive learnable bias term $b_1$ is included to produce:

$$f_1(\boldsymbol{x}; \boldsymbol{w}_1) = \Phi^{(1)}(\boldsymbol{x} \cdot \boldsymbol{w}_1 + b_1) \tag{A.1.2}$$

where $\Phi$ denotes a non-linear *activation* function that is chosen as a hyperparameter and meant to help learn non-linearly spaces. This output then passes to the second layer where it is greeted by the next round of weights $\boldsymbol{w}_2$, second additive bias $b_2$, and second activation $\Phi^{(2)}$:

$$f_2(f_1(\boldsymbol{x}; \boldsymbol{w}_1); \boldsymbol{w}_2) = \Phi^{(2)}(f_1(\boldsymbol{x}; \boldsymbol{w}_1) \cdot \boldsymbol{w}_2 + b_2) \tag{A.1.3}$$

continuing on $L$ times. Note that if we opt for a linear network, then the activations $\Phi^{(i)}$ become the identity mapping to preserve linearity. A generalized notation that turns weight vectors into weight matrices $\boldsymbol{W}_i$is also common, though weight vectors conceptually work as well.

A Bayesian would see these learned weights as samples from the posterior, so that we may write layer 1 as:

$$f_1(\boldsymbol{x}; \boldsymbol{w}_1 \sim P(\boldsymbol{w}|D)) = \Phi^{(1)}(\boldsymbol{x} \cdot \boldsymbol{w}_1 + b_1) \tag{A.1.4}$$

and so on. Note that a *global* posterior is learned, and not a layer-wise posterior, as the network is trained as a whole during *Bayes by Backprop* (in the Bayesian case) and traditional backpropagation (in the frequentist). This is an important distinction.

## A.E Experiments

Note that the code for these experiments is available on my GitHub repo:
https://github.com/johnvalen1/Bayesian-Learning-and-Optimization-Master-Thesis.git and is ordered by section (see the *ReadMe*).

### A.E.2.7.1 *A motivating example of Bayesian uncertainty using a neural network for function approximation*

The function to be learned is $f(x) = 3x + 7$ using a neural network regressor that learns the functional mapping $x \to f(x)$ from $x \to f(x) + \varepsilon$, where both $x, \varepsilon \sim N(0,1)$. In this way, the input is a random variable that clusters around $\mu_x = 0$, so that ordered pairs $(x, f(x))$ further from the point (0, 7) are observed less and modeling uncertainty in these regions is more illustrative. See Table A.E.2.1 below for the details of the data used in this experiment. Note that both training and test sets come from the same distributions, admittedly not always the case in practice.

Note also that $\varepsilon \sim N(0, \sigma^2)$ for $\sigma^2 > 1$ was tried too, with the intuitive result that the shaded uncertainties from Figures 2.3 and 2.4 grew wider. More noise in observations correlating with higher model *and* data uncertainties is no surprise.

| Ground truth | Samples | How much data? | Batches and epochs? |
|---|---|---|---|
| $f(x) = 3x + 7$ | $(x, f(x) + \varepsilon)$ | Training size | Batch size |
| | $x, \varepsilon \sim N(0,1)$ | $\{(x_i, f(x_i) + \varepsilon_i)\}_{i=1}^{1000}$ | 10 |
| | | Test size | Number of epochs |
| | | $\{(x_i, f(x_i) + \varepsilon_i)\}_{i=1001}^{2000}$ | 30 |

Table A.E.2.1: The specifics of the data used for this experiment.

For this network, the dropout rate is learned like in [16], and the weight regularizer and dropout regularizer values are learned according to a prior *length scale* (which encodes *a priori* beliefs about the frequency characteristics of the data). Gal uses a value of 0.0001 for this length scale to base the two regularizers off of, and these are left unchanged. See Table A.E.2.2 below for more details on the neural network's architecture. Note the use of a custom loss function that leverages aleatoric uncertainty, as shown in section 2.5.4 equation (2.8), such that the effect of 'exotic' examples is attenuated.

| Layers | Size | Followed by activation? | Dropout? |
|---|---|---|---|
| Input | 1 | -- | -- |
| Fully connected | 1024 | *ReLU* | Learnable weight and dropout regularizers |
| Fully connected | 1024 | *ReLU* | " |
| Fully connected | 1024 | *ReLU* | " |
| Output | 1 | -- | -- |

Table A.E.2.2: The architecture of the neural network regressor model.

The model outputs predictive conditional mean $E(f(x)|x)$ and predictive variance $var(f(x)|x)$ from which we can model the uncertainties by taking 20 MC samples. These 'ensemble' predictions (in the parlance of MacKay [36]) on the test set can be averaged to produce a single prediction, where the epistemic uncertainty is the variance across these predictions. On the other hand, the aleatoric uncertainty is the average over the output predictive variances. This agrees with the notion that the former measures *model* uncertainty, while the latter is concerned with *data* uncertainty.

This experiment relies on a *TensorFlow* implementation in *R*, adapted from the Python code for Gal *et al.*'s paper 'Concrete Dropout' by Sigrid Keydana [31].

### A.E.2.7.2 *A Bayesian neural network classifier that relies on uncertainty when making predictions*

This experiment uses a single-layer Bayesian neural network (or directed graphical model, in Bayesian parlance) to classify MNIST handwritten digit images. The details of the model and the data are given below in Tables A.E.2.5 and A.E.2.6.

Here, $T$ is varied (see leftmost column of Table A.E.2.3 below). The test image will then have ascribed to it a vector consisting of 10 digit-specific probabilities from each of the $T$ models (the first element of which represents the approximate posterior class probability $P(C = 0|\boldsymbol{x}^{new}, \boldsymbol{X}, \boldsymbol{y})$ that the test image is a 0, and so on). Thus, for each of the 10 digits, the model will have $T$ posterior probabilities (defining a distribution, as in Figure 2.5 below), and its final classification will be whichever digit(s) have their median posterior probability exceeding a pre-defined threshold (see leftmost column of Table A.E.2.4). This allows the network to output zero, one, or several labels- a very useful feature when it comes to medical imaging as it can raise attention for further inspection from a radiologist.

Table A.E.2.3 reports the model's predictive performance metrics (accuracy, epistemic and aleatoric uncertainties) averaged across 5 runs, before being allowed to 'refuse' making predictions. By imposing the pre-defined threshold of acceptance, the network will be forced to 'think harder' about its predictions, thereby rejecting some images, but increasing the quality of its predictions made on the remaining images. Without the ability to 'refuse' classifying, the network can still report its uncertainties, without indication of a clear relationship to the number of samples taken from the variational posterior at test time. This is seen below in Table A.E.2.3 which reports the network's average performance *before* leveraging the uncertainties encoded in the posteriors to decide whether or not to classify. Table A.E.2.4 reports its average performance *after*.

| Number of times weights sampled | Avg. accuracy | Avg. epistemic uncertainty | Avg. aleatoric uncertainty |
|---|---|---|---|
| 10 | 8,967 / 10,000 = 90% | 0.00201 | 0.00464 |
| 15 | 8,915 / 10,000 = 89% | 0.00380 | 0.00652 |
| 20 | 8, 971 / 10,000 = 90% | 0.00075 | 0.00292 |
| 25 | 8,987 / 10,000 = 90% | 0.00095 | 0.00375 |
| 30 | 8,966 / 10,000 = 90% | 0.00176 | 0.00377 |

Table A.E.2.3: Varying *T* does not seem to improve model accuracy, nor significantly reduce uncertainties. The network's predictive performance before being able to 'refuse' classifying more exotic examples. There is no clear connection between column 1 and the uncertainties. Note that the accuracy is also relatively unaffected.

| Threshold of acceptance | Images not classified | Avg.accuracy when able to refuse | Avg. epistemic uncertainty | Avg. aleatoric uncertainty |
|---|---|---|---|---|
| 0.1 | 864 / 10,000 | 8,372 / 9,135 = 91% | 0.00110 | 0.00264 |
| 0.2 | 953 / 10,000 | 8,390 / 9,046 = 92% | 0.00142 | 0.00466 |
| 0.3 | 974 / 10,000 | 8,359 / 9,025 = 92% | 0.00209 | 0.00536 |
| 0.4 | 965 / 10,000 | 8,314 / 9,034 = 92% | 0.00095 | 0.00287 |
| 0.5 | 1,331 / 10,000 | 8,115 / 8,668 = 93% | 0.00057 | 0.00221 |
| 0.6 | 1,757 / 10,000 | 7,817 / 8,243 = 94% | 0.00097 | 0.00336 |

Table A.E.2.4: Varying the threshold of so-called 'acceptance' is more effective in increasing average accuracy and reducing epistemic uncertainty. The network's predictive performance as a function of a threshold that determines its decision to classify or not, based on how uncertain it is about the new example. Naturally, the higher the threshold, the more it refuses to classify because it becomes 'picky'.

Now, Table A.E.2.5 gives the specifics of the data and the architecture of the Bayesian neural network (or directed graphical model, in the Bayesian parlance). The last column demonstrates the difference in architecture, as prior distributions over the fully connected weights and biases must be stated as part of the model design. Note the use of multivariate normal priors, inspired by Shridhar *et al.*'s work [47] (see chapter 3 section 3.4.3). Arguably, the prior would not pass as 'uninformative' (since it would need to spread its probability mass out much more via some $\sigma^2 \gg 1$ so that its variance $\sigma^2 \boldsymbol{I}$ would represent higher entropy). However, the results obtained were still acceptable, and running a *sensitivity analysis* (i.e. seeing what happens under different priors) is better warranted for more complex models where model specification is more crucial.

| Layers | Size | Followed by activation? | Prior distributions? |
|---|---|---|---|
| Input | 28*28 | -- | -- |
| Fully connected | 1024 | *ReLU* | Multivariate $N(\mathbf{0}, \boldsymbol{I})$ |
| Output | 10 | -- | Multivariate $N(\mathbf{0}, \boldsymbol{I})$ |

Table A.E.2.5: Architecture of the Bayesian network in this experiment.

The network is trained in the Bayesian way by maximizing the ELBO in cost function (2.4) via the Adam optimizer with a learning rate of 0.01.

Uncertainty measures are calculated as in the work of Shridhar *et al.* [47] and Kwon *et al.* [35]. Predictions with the trained network are made across 5 runs and all performance metrics are therefore averaged across the runs, as well as being averaged across batches for each run.

Figure A.E.2.1 below shows the lack of relationship between number of samples taken at test time, and the uncertainty measures. Figure A.E.2.2 shows an analogous finding but for the threshold of acceptance. These figures show that uncertainties can be minimized together, on the basis of number of samples $T$ or on the basis of a threshold that changes the prediction behaviour of the network.

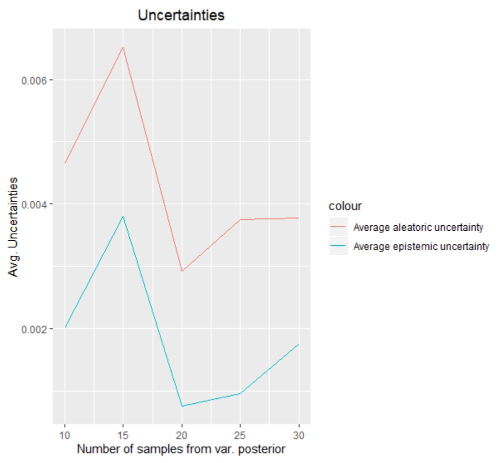


Figure A.E.2.1: Uncertainties as we vary the number of samples taken from the variational posterior.

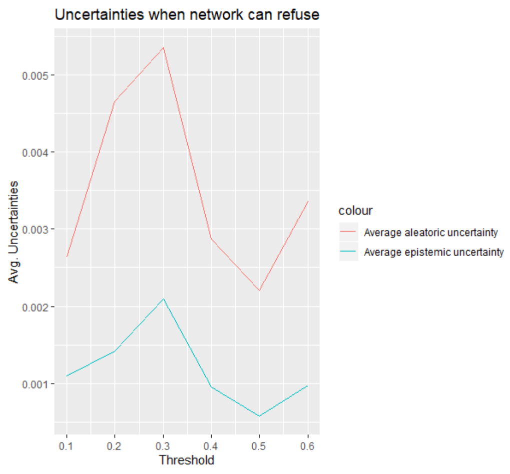


Figure A.E.2.2: Uncertainties when we vary the threshold at which the network makes predictions, for $T = 10$ samples of weights and biases from the variational posterior.

### A.3.6.1 *A comparison between Bayesian and frequentist CNNs*

*Data*

The dataset is MNIST, with 50,000 training/validation images, and 10,000 test images. An 80-20 training-validation split is taken. Note that images are resized to 32×32 but this is optional. They are also transformed via random horizontal flipping and random rotation of 10° as in [47] which inspired this experiment. Batch sizes for training are 32, while for testing the batch size is 1 for the frequentist AlexNet (to run drop-out *T* times across *each* image, not batch. Conceptually, reporting batch-wise *averaged* accuracy is equivalent to image-wise *averaged* accuracy).

*Architectures*

Table A.E.3.1 below illustrates the architecture of the AlexNet CNN with custom hyperparameter values throughout as in [47] (note that the same overall structure is used for both frequentist and Bayesian approaches, only the inferential procedure changes).

For more information, see the PyTorch documentation on each layer component.

| What? | PyTorch name | Description |
|---|---|---|
| Input layer | -- | • $32 \times 32$ image |
| Convolution | Conv2d(*in-channels*=1, *out-channels*=64, *kernel-size*=11, *stride*=4, *padding*=5) | • 1 input channel (MNIST is greyscale)<br>• 64 output channels (feature maps)<br>• Kernel is $11 \times 11 \times 1$<br>• Stride is 4<br>• Padding is 5 (same) |
| Activation | ReLU(*inplace*=True) | • Activation applied in place (i.e. locally) |
| Dropout | Dropout(*p*=0.5) | • Dropout following activation, with probability 0.5 |
| Max pooling | MaxPool2d(*kernel-size*=2, *stride*=2) | • Kernel size is the window over which the max is taken, so it is $2 \times 2$<br>• Stride |
| Conv2d | Conv2d(*in-channels*=64, *out-channels*=192, *kernel-size*=5, *stride*=1, *padding*=2) | • Input channel must match output channel from previous convolution |
| Activation | ReLU(*inplace*=True) | -- |
| Max pooling | MaxPool2d(*kernel-size*=2, *stride*=2) | • Same as above |
| Conv2d | Conv2d(*in-channels*=192, *out-channels*=384, *kernel-size*=3, *stride*=1, *padding*=1) | -- |
| Activation | ReLU(*inplace*=True) | -- |
| Dropout | Dropout(*p*=0.5) | -- |
| Conv2d | Conv2d(*in-channels*=384, *out-channels*=256, *kernel-size*=3, *stride*=1, *padding*=1) | -- |
| Activation | ReLU(*inplace*=True) | -- |

| | | |
|---|---|---|
| Conv2d | Conv2d(*in-channels*=256, *out-channels*=256, *kernel-size*=3, *stride*=1, *padding*=1) | -- |
| Activation | ReLU(*inplace*=True) | -- |
| Dropout | Dropout($p$=0.5) | -- |
| Max pooling | MaxPool2d(*kernel-size*=2, *stride*=2) | -- |
| Linear classifier | Linear(*in-features*=256, *out-features*=10, *bias*=True) | • One fully connected layer for making the classification |

Table A.E.3.1: AlexNet architecture according to PyTorch documentation.

*Aspects related to training*

Note that both use Adam optimizer (for different purposes; the frequentist optimizes the *categorical cross entropy* here, while the Bayesian the *ELBO* from chapter 2). Note that no hyperparameters related to the architecture itself are modified at any point, since these come pre-defined (as Table A.E.3.1 above shows). The learning rate for the Bayesian CNN is *adaptable* and starts at 0.001 while the frequentist CNN has learning rate 0.01. The confidence interval uses a t-statistic with $t = 2.262\ (10 - 1\ df, \alpha = 0.05)$ while the distribution-free confidence interval uses order statistics and sets the lower and upper bound as the 2.5th percentile and 97.5th percentile respectively.

Interestingly, in the Bayesian case, the initial epoch gives a training set accuracy that is less than half that of the validation set, so this initial epoch is discarded as a 'burn-in' value, the discarding of which does not affect the average generalization gap between training and testing (see below). The frequentist network more often had noticeably poorer initial performance on certain runs, often getting 'stuck' and not improving enough. In that case, it was executed and restarted.

Note that the train-validation generalization gap is calculated as the average difference *during* training, while the train-test generalization gap is calculated as the difference between the trained network's training accuracy and its test accuracy. Times for training and testing are both tracked and reported, for both types of networks.

| $\alpha = \{n_{epochs}\}$ | Frequentist AlexNet with dropout | Bayesian AlexNet with VI | Freq. Generalization Gap | Bayesian Generalization Gap |
|---|---|---|---|---|
| 5 | **Accuracy:**<br>76.251 $\pm$ 4.207%<br>**Distribution free C.I.:**<br>(73.889, 1.000)<br>**Median**: 75.990% | 95.138 $\pm$ 3.195%<br>(84.375, 1.00)<br>96.875% | **Avg. train-validation**:<br>-8.554%<br>**Avg. train-test**:<br>1.178% | -1.043%<br>-3.160% |
| 10 | 82.199 ± 3.854%<br>(80.819, 1.00)<br>82.251% | 90.625 ± 4.916%<br>(74.375, 1.00)<br>90.625% | 0.379%<br>1.919% | 0.380%<br>-4.154% |
| 15 | 85.838 ± 5.702%<br>(83.878, 1.00) | 85.573 ± 6.178%<br>(65.625, 1.00)<br>87.500% | -1.109%<br>0.658% | -0.704%<br>1.606% |

| | | | | |
|---|---|---|---|---|
| | 85.472% | | | |
| 20 | 80.544 ± 5.412%<br>(78.914, 1.00)<br>80.391% | 89.956 ± 4.956%<br>(75.000, 1.00),<br>90.625% | -0.844%<br><br>1.468% | 0.181%<br><br>-1.494% |
| 25 | 84.472 ± 3.872%<br>(83.192, 1.00)<br>84.362% | 88.778 ± 5.433%<br>(71.875, 1.00),<br>90.625% | -0.147%<br><br>1.284% | 0.258%<br><br>0.193% |

Table A.E.3.2: Comparing Bayesian and frequentist AlexNet performance metrics for different choices of epochs.

### A.E.3.6.2 *Modeling the Occam factor for a Bayesian CNN*

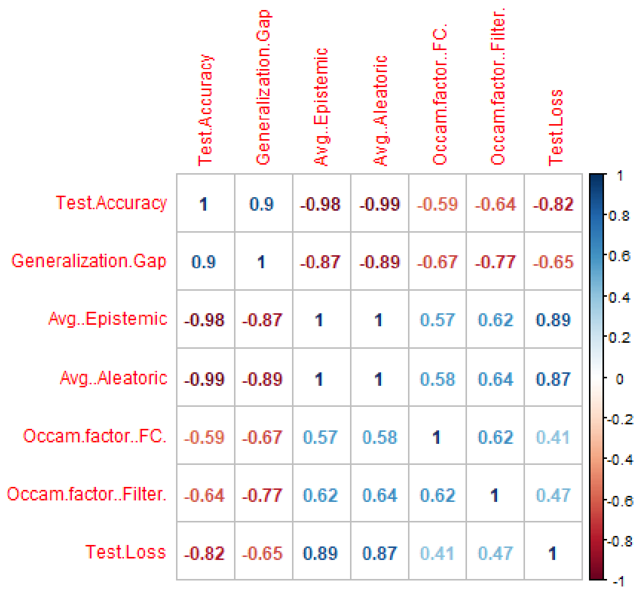
Note: more figures and plots (along with the *R* code to produce them) can be found on my GitHub repo.

This experiment relies on a LeNet-5 Bayesian architecture as in Shridhar *et al.*'s work [47], the details of which are left untouched except the number of epochs to train for (more epochs, more convergence, and thus a more accurate Occam factor). The architecture can be found on their GitHub repo at https://github.com/kumar-shridhar/PyTorch-BayesianCNN.git.

The randomly selected fully connected weight connects the first nodes of the second and third fully connected layers, while the filter weight is the top-left weight of the first filter in the first convolutional layer (there is only one channel since the data is greyscale). Other weights exhibited similar behavior, so we settle on these.

The Bayesian CNN is trained 15 times across 2, 5, 10, and 30 epochs, where the idea is that with each epoch, the standard deviations of both weights will decrease as the variational posterior converges. By training 15 networks, we are not relying on a single trained network for interpretation. Each network is trained and tested, keeping track of test accuracy, generalization gap between train and test accuracies, two Occam factors (one approximated by the fully connected weight and the other by the filter weight), and uncertainty measures (epistemic and aleatoric).

These are reported in the correlation matrix in Figure A.E.3.3 below, and the scatter plot matrix of Figure A.E.3.4 displays pairwise relationships to determine non-linear relationships not captured by the correlation matrix (which is based on linearity). Finally, Figure A.E.3.5 features two panels which relate the generalization gap to both classes of Occam factor.



| | Test.Accuracy | Generalization.Gap | Avg..Epistemic | Avg..Aleatoric | Occam.factor..FC. | Occam.factor..Filter. | Test.Loss |
|---|---|---|---|---|---|---|---|
| Test.Accuracy | 1 | 0.9 | -0.98 | -0.99 | -0.59 | -0.64 | -0.82 |
| Generalization.Gap | 0.9 | 1 | -0.87 | -0.89 | -0.67 | -0.77 | -0.65 |
| Avg..Epistemic | -0.98 | -0.87 | 1 | 1 | 0.57 | 0.62 | 0.89 |
| Avg..Aleatoric | -0.99 | -0.89 | 1 | 1 | 0.58 | 0.64 | 0.87 |
| Occam.factor..FC. | -0.59 | -0.67 | 0.57 | 0.58 | 1 | 0.62 | 0.41 |
| Occam.factor..Filter. | -0.64 | -0.77 | 0.62 | 0.64 | 0.62 | 1 | 0.47 |
| Test.Loss | -0.82 | -0.65 | 0.89 | 0.87 | 0.41 | 0.47 | 1 |

Figure A.E.3.3: The correlation matrix with color gradients. Red means negative correlation

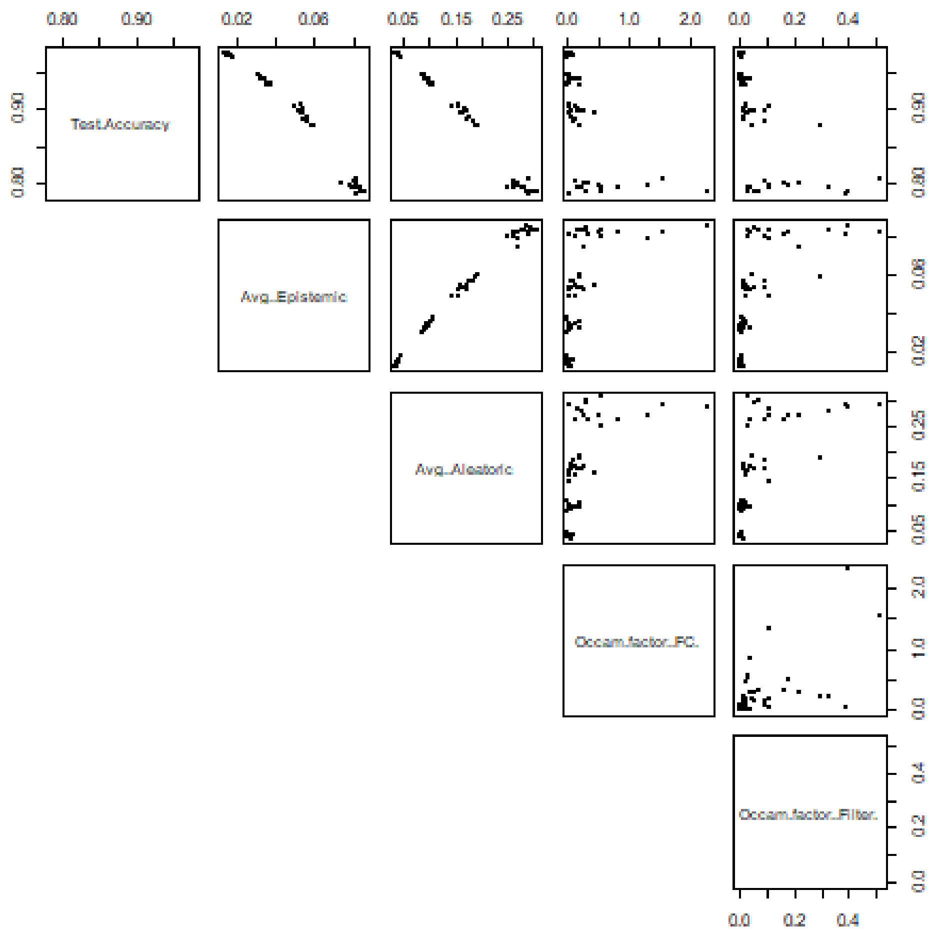


Figure A.E.3.4: The scatter plot matrix between model metrics to determine existence of linear relationships.

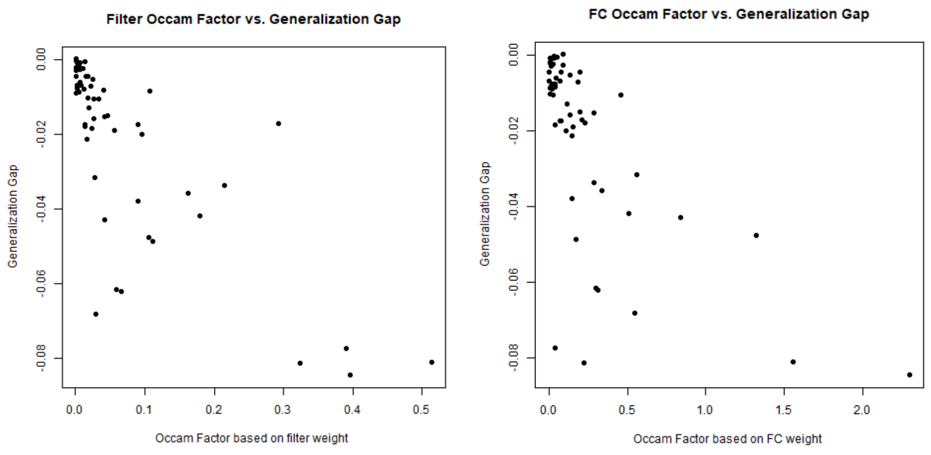


Figure A.E.3.5: The generalization gap and the two forms of Occam factor.

#### A.E.4.4.1 *Bayesian optimization for a noisy function*

This experiment uses the package *GPyOpt*, and its code can be found on the GitHub repository.

The acquisition function, GP parameterization, and $\varepsilon$ are each experimented with, with observations in Table A.E.4.1 below. 20 iterations generally seemed to work well enough, a judgment that is typically subjective in practice. The exploration-exploitation parameter $\xi$ is taken to be 0.01. The acquisition is maximized by the BFGS algorithm. Table A.E.4.1 summarizes the experiment design.

| Search space | Initial samples | Number of iterations | GP parameterization for Matérn kernel | $\varepsilon \sim N(0, \delta)$ | Best performing acquisition function |
|---|---|---|---|---|---|
| $x \in [-1,1]$ | $x = -0.5$<br>$x = 0.5$ | 20 | $\nu \in \{\frac{3}{2}, \frac{5}{2}, \frac{7}{2}\}$ | $\delta \in \{0.1, 0.2, 0.5\}$ | Expected improvement |

Table A.E.4.1: The choice of varying GP parameterization, varying additive noise, and best-performing acquisition function (based on observation).

The following are the different scenarios in which the data-inherent additive noise and GP parameterization are modified. Note the varying scale parameter $\nu$ (automated kernel parameter selection methods exist but rely on MLE or MCMC [18] and are beyond the scope of this work). The number of iterations was generally kept to 20 due to time and hardware constraints. Note that only the expected improvement acquisition showed adequate performance; the other two acquisition functions performed poorly and are not included here. Run time is an average of 11.76 seconds.

| $\nu$ (of GP kernel) | Additive noise | Average epistemic uncertainty | Average aleatoric uncertainty | Convergence? |
|---|---|---|---|---|
| 1.5 | $\varepsilon \sim N(0,0.1)$ | 0.005424 | 0.099268 | Yes |
| 2.5 | | 0.005499 | | Yes |
| 3.5 | | 0.004259 | | Yes |
| 1.5 | $\varepsilon \sim N(0,0.2)$ | 0.012451 | 0.198537 | Yes* |
| 2.5 | | 0.011434 | | Yes* |
| 3.5 | | 0.011244 | | Yes |
| 1.5 | $\varepsilon \sim N(0,0.5)$ | 0.034696 | 0.496342 | No** |
| 2.5 | | 0.041670 | | Yes |
| 3.5 | | 0.042178 | | Yes* |

Table A.E.4.2: Showing the positive relationship between additive noise and epistemic uncertainty; the higher the data-inherent noise, the more uncertain is the model. This is also true of the aleatoric uncertainty, but by default since this is modeled as the variance of the additive noise term.

*Yes, but the convergence is now visibly worse than the convergence of the preceding row

**More iterations would have likely helped it converge. Too much noise is present.

#### A.E.4.4.2 *The importance of the acquisition function and GP parameterization in hyperparameter tuning*

The neural network model is defined in *PyTorch*, and the Bayesian Optimization algorithm is run via *GPyOpt*.

The model of choice here is a neural network with two dropout hidden layers that achieves good test-set accuracy and loss by default (~96% and 0.07 respectively) on MNIST, with categorical cross entropy loss. Its architecture is then tuned via Bayesian optimization on the hyperparameters shown below in Table A.E.4.3. The loss and accuracy are tracked in each of the 35 iterations of the algorithm (by default taking an initial 5 samples), and the goal is to track the network's performance for each choice of acquisition function (see Figure 4.5) throughout the hyperparameter tuning process. The reason for this is to check if these results are robust against choice of acquisition function; inspired by [11], where Frazier discusses the importance of the experimental design. In each cell of Table A.E.4.4, the hyperparameter's optimal value is given, organized according to one of the three acquisition functions considered here.

| Hyperparameter | Bounds |
|---|---|
| $p_1$-dropout rate in FC-layer 1 | [0, 0.5] |
| $p_2$-dropout rate in FC-layer 2 | [0, 0.5] |
| $l_1$-size of hidden FC-layer 1 | {64, 128, 256, 512, 1024} |
| $l_2$-size of hidden FC-layer 2 | {64, 128, 256, 512, 1024} |
| Batch size | {32, 64} |
| Epochs | {5, 10, 20} |

Table A.E.4.3: Hyperparameter bounds for the neural network's architecture.

In the tables that follow, a distinction is made between accuracy and test-time accuracy. The former is the accuracy value obtained during Bayesian Optimization that led to the selected hyperparameters, while the latter is reported with a 95% confidence interval because it is the result of applying dropout 10 times at test-time (as in section 2.5.3).

| **Hyperparameter** | **EI acq.** | **MPI acq.** | **LCB acq.** |
|---|---|---|---|

| | | | |
|---|---|---|---|
| $p_1$-dropout rate in layer 1 | 0.416 | 0.306 | 0.279 |
| $p_2$-dropout rate in layer 2 | 0.198 | 0.280 | 0.413 |
| $l_1$-size of hidden layer 1 | 1024 | 1024 | 256 |
| $l_2$-size of hidden layer 2 | 64 | 256 | 128 |
| Batch size | 64 | 64 | 32 |
| Epochs | 20 | 20 | 10 |
| **Overall performance** | | | |
| Accuracy: | 98.2% | 98.0% | 98.0% |
| Loss: | 0.060 | 0.067 | 0.0701 |
| Test-time accuracy: (*dropout 10 times*) | $97.53 \pm 0.005$ | $97.76 \pm 0.013$ | $96.99 \pm 0.007$ |
| Aleatoric uncertainty: | 0.0060 | 0.0047 | 0.0070 |
| Epistemic uncertainty: | 0.0074 | 0.0044 | 0.0119 |

Table A.E.4.4: Retrieved hyperparameters from Bayesian Optimization along with performance metrics for each acquisition function (with $\nu = 1.5$). Reported with best performance metrics from amongst 5 runs (to account for chance results). Note that this is not a Bayesian CNN/neural network; uncertainty measures are still reported, obtained by casting drop-out at test time like in [14].

| **Hyperparameter** | $\nu = 1.5$ | $\nu = 2.5$ | $\nu = 3.5$ |
|---|---|---|---|
| $p_1$-dropout rate in layer 1 | 0.416 | 0.193 | 0.0002 |
| $p_2$-dropout rate in layer 2 | 0.198 | 0.238 | 0.295 |
| $l_1$-size of hidden layer 1 | 1024 | 512 | 512 |
| $l_2$-size of hidden layer 2 | 64 | 512 | 128 |
| Batch size | 64 | 32 | 64 |
| Epochs | 20 | 5 | 10 |
| **Overall performance** | | | |
| Accuracy: | 98.2% | 98.1% | 98.1% |
| Loss: | 0.060 | 0.067 | 0.063 |
| Test-time accuracy: (*dropout 10 times*) | $97.53 \pm 0.005$ | $97.04 \pm 0.007$ | $97.90 \pm 0.008$ |
| Aleatoric uncertainty: | 0.0060 | 0.0042 | 0.0015 |
| Epistemic uncertainty: | 0.0074 | 0.0108 | 0.0064 |

Table A.E.4.5: Retrieved hyperparameters from Bayesian Optimization along with performance metrics for the EI acquisition function and different Matérn kernel parameterizations, reported with best performance metrics across 5 runs (to account for chance results).

| Acquisition | $\nu = 1.5$ | $\nu = 2.5$ | $\nu = 3.5$ |
|---|---|---|---|
| EI | 17m28s – 27m47s | 17m23s – 32m58s | 25m19s – 30m36s |
| MPI | 14m56s – 27m57s | 15m20s – 16m14s | 14m33s – 26m57s |

| LCB | 15m19s – 25m38s | 22m5s – 23m23s | 20m36s – 21m16s |
|---|---|---|---|

Table A.E.4.6: Range of times taken for each combination of parameterization of the GP regressor, and choice of acquisition function

This experiment uses the *GPyOpt* library, and once the optimizer runs on this data, a *.csv* file is constructed for each choice of acquisition where each row represents the results of one iteration of the algorithm. The neural network is trained using the *adam* optimizer, and the cost function is categorical cross entropy since this is a typical classification task with 10 classes.

### A.E.4.4.3 *Bayesian optimization as an informed alternative to random search*

This experiment relies on the ADF (Augmented Dickey-Fuller) and Ljung-Box test for time series data.

The ADF tests whether the data is non-stationary by testing $\gamma = 0$ in the higher-order autoregressive process:

$$\Delta y_t = \alpha + \beta t + \gamma y_{t-1} + \delta_1 \Delta y_{t-1} + \delta_2 \Delta y_{t-2} + \cdots$$

The Ljung-Box test tests the null hypothesis that the data are independently distributed and is commonly applied to the residuals of a fitted ARIMA model. In this case, we are testing whether the residuals from the model are autocorrelated, and if the model is white noise, then these autocorrelations should be zero. Table A.E.4.7 below gives the best ARIMA model (on the basis of AIC; see full code on GitHub repository) for each of the time series considered here, and Table A.E.4.7 reports p-values of the Ljung-Box test on the residuals of these models. Interestingly, almost all models favoured simplicity as the parameterization of the ARIMA models was minimal.

| Algorithm / Time Series | ARIMA$(p,d,q)$ |
|---|---|
| BayesOpt with EI acquisition | Loss: $(p,d,q) = (1,1,1)$<br>Accuracy: (1,1,1) |
| BayesOpt with MPI acquisition | Loss: (1,1,1)<br>Accuracy:(2,1,1) |
| BayesOpt with LCB acquisition | Loss: (1,1,1)<br>Accuracy: (1,1,1) |
| Random search | Loss: (1,1,1)<br>Accuracy: (1,1,1) |

Table A.E.4.7: ARIMA models for the ‘time series’ to obtain residuals and perform the Ljung-Box test.

Since all ARIMA models are almost the same (with the exception of one), it makes sense to compare them on the basis of whiteness of residuals to establish whether autocorrelation is present. This helps us examine whether the trend that should exist with an informed model exists.

Table A.E.4.7 below presents the results of the Ljung-Box statistical test, where the null hypothesis is that the process is white noise, and of the augmented Dickey-Fuller (ADF) test for stationarity. The GP parameterization for Bayesian optimization takes constant mean, and Matérn kernel with $\nu = 1.5$.

|  | Ljung-Box p-value | ADF test p-value | Time taken | Metrics |
|---|---|---|---|---|

| Algorithm / time series | Loss | Accuracy | Loss | Accuracy | | |
|---|---|---|---|---|---|---|
| BayesOpt with EI acquisition | 0.8030 | 0.9789 | 0.7454 | 0.6093 | 27m 47s | 98.2% acc, 0.060 loss |
| BayesOpt with MPI acquisition | 0.9715 | 0.9991 | <0.0001* | <0.0001* | 27m 57s | 98.0% 0.067 |
| BayesOpt with LCB acquisition | 0.8214 | 0.6225 | 0.0073* | 0.0002* | 25m 38s | 98.0% 0.0701 |
| Random search | 0.5921 | 0.7171 | 0.0014* | 0.0007* | 18m 46s | 97.98% 0.0665 |

Table A.E.4.8: The use of * denotes statistical significance.

### A.E.5.1

This experiment considers both a Bayesian CNN and a frequentist CNN. The custom architecture is described below in Table A.E.5.1.

The imaging data is resized to 224*224, normalized, and turned into greyscale (so as to retain only one channel). Note that transformations like flipping and randomly rotating were also tried, but performance was worse (too much additional noise and not enough data, perhaps).

The Bayesian convolutional networks are trained for 30 epochs, and 10 times. Therefore, a .csv file is created with 10 rows (one row per model) where each column tracks a performance metric for that model. The metrics that are tracked are training accuracy, testing accuracy, generalization gap (the difference of these two), average epistemic and aleatoric uncertainties, Occam factors based on both a fully connected and filter weight, and train/test losses. Tables A.E.5.1 and A.E.5.2 below give the architecture along with the hyperparameters used. Note the use of a ‘Beta’ hyperparameter- this corresponds to the Beta type used in the Softplus activation and is taken as 0.2, as in [47].

| What? | PyTorch name | Description |
|---|---|---|
| Input layer | -- | • $224 \times 224$ image, 1 channel |
| Convolution | Conv2d(*in-channels*=1, *out-channels*=6, *kernel-size*=5, *stride*=1, *padding*=0) | • 1 input channel (images made greyscale)<br>• 6 output channels (feature maps)<br>• Kernel is $5 \times 5 \times 1$<br>• Stride is 1<br>• Padding is 0 (same) |
| Activation | Softplus(*inplace*=True) | • Activation applied in place (i.e. locally) |
| Max pooling | MaxPool2d(*kernel-size*=2, *stride*=2) | • Kernel size is the window over which the max is taken, so it is $2 \times 2$ |
| Conv2d | Conv2d(*in-channels*=6, *out-channels*=16, *kernel-size*=5, *stride*=1, *padding*=0) | • Input channel matches output channel from previous convolution |
| Activation | Softplus(*inplace*=True) | -- |

| | | |
|---|---|---|
| Max pooling | MaxPool2d(*kernel-size*=2, *stride*=2) | • Same as above |
| Conv2d | Conv2d(*in-channels*=16, *out-channels*=16, *kernel-size*=5, *stride*=1, *padding*=1) | -- |
| Activation | Softplus(*inplace*=True) | -- |
| Max pooling | MaxPool2d(*kernel-size*=3, *stride*=2) | -- |
| Linear classifier | Linear(*in-features*=25*25*16, *out-features*=1000, *bias*=True) | • Three fully connected layers for making the classification |
| Activation | Softplus(*inplace*=True) | • -- |
| Linear classifier | Linear(*in-features*=1000, *out-features*=500, *bias*=True) | • -- |
| Activation | Softplus(*inplace*=True) | • -- |
| Linear classifier | Linear(*in-features*=500, *out-features*=2, *bias*=True) | • Final classification output layer |

Table A.E.5.1: Custom architecture for the COVID-19 CT scan medical imaging experiment.

| Hyperparameter | Bounds |
|---|---|
| Batch size | 32 |
| Epochs | 30 |
| Optimizer | Adam |
| Learning rate | 0.01 |
| Validation split | 0 (no validation set, only train/test) |
| Prior initialization | $N(0,1)$ priors for Bayesian (and draws for frequentist) |
| Beta | 0.2 |

Table A.E.5.2: Hyperparameters for the Bayesian CNN (same used for frequentist counterpart where applicable).

| Metric | Value |
|---|---|
| Train accuracy | 81.25% |
| Test accuracy | 50%* |
| Generalization gap | 31.25% |

Table A.E.5.3: Training and testing results of custom-defined frequentist CNN from one of several re-runs. *Test accuracy a constant too; network overfits dramatically.

| Metric | Dist'n-free 95% conf. interval | Median |
|---|---|---|
| Train accuracy | (66.87%, 74.52%) | 72.54% |
| Test accuracy | N/A | 46.88%* |
| Avg. Epistemic | (0.0312, 0.03915) | 0.0348 |
| Avg. Aleatoric | (0.2316, 0.2955) | 0.2755 |
| Approx. Occam factor | (0.0010, 0.0559) | 0.0056 |
| Train Loss | (410511.6, 548040.1) | 502561.8 |
| Test Loss | (4044328, 5420186) | 4967411 |

Table A.E.5.4: Training and testing results of custom-defined Bayesian CNN averaged across 10 re-runs. *Test accuracy a constant too; network overfits dramatically.

| | Label (+) | Label (-) |
|---|---|---|
| Predict (+) | 30 | 30 |
| Predict (-) | 0 | 0 |

Figure A.E.5.1: Confusion matrix for Bayesian CNN, along with sensitivity/specificity.

## References

[1] Anonymous. "Empirical Bayes Transductive Meta-Learning With Synthetic Gradients." *ICLR Conference 2020*, 2019.

[2] Balki, Indranil, Afsaneh Amirabadi, Jacob Levman, Anne L. Martel, Ziga Emersic, Blaz Meden, Angel Garcia-Pedrero, Saul C. Ramirez, Dehan Kong, Alan R. Moody, and Pascal N. Tyrrell. "Sample-Size Determination Methodologies for Machine Learning in Medical Imaging Research: A Systematic Review." *Canadian Association of Radiologists Journal*, 2019.

[3] Blundell, Charles, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. "Weight Uncertainty in Neural Networks." *arXiv preprint arXiv:1505.05424*, May 21, 2015.

[4] Booker, A., J. Dennis, D. Serafini, V. Torczon, and M. Trosset. "A Rigorous Framework for Optimization of Expensive Functions by Surrogates." *Structural and Multidisciplinary Optimization*, February 1999.

[5] Brochu, Eric, Nando de Freitas, and Vlad M. Cora. "A Tutorial on Bayesian Optimization of Expensive Cost Functions," *arXiv preprint arXiv:1012.2599*, December 10, 2014.

[6] Buntine, Wray L., and Andreas S. Weigend. "Bayesian Back-Propagation." *Complex Systems*, 1991.

[7] Carvalho, Carlos M., Nicholas G. James G. Scott Polson, and James G. Scott. "Handling Sparsity via the Horseshoe." *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics*, 2009.

[8] Cohen, Joseph Paul, Paul Morrison, and Lan Dao. "COVID-19 Image Data Collection." *arXiv preprint arXiv:2003.11597*, March 25, 2020.

[9] Cribari-Neto, Francisco, and Achim Zeileis. "Beta Regression in R." *CRAN*.

[10] Duvenaud, David Kristjanson. "Automatic Model Construction with Gaussian Processes." PhD diss., University of Cambridge, 2014.

[11] Frazier, Peter I. "A Tutorial on Bayesian Optimization." *arXiv preprint arXiv:1807.02811*, July 10, 2018.

[12] Frean, M., and P. Boyle. "Using Gaussian Processes to Optimize Expensive Functions." *AI 2008: Advances in Artificial Intelligence* 5360 (2008): 258-67.

[13] Gaier, Adam, and David Ha. "Weight Agnostic Neural Networks." *arXiv preprint arXiv:1906.04358*, September 5, 2019.

[14] Gal, Yarin, and Zoubin Ghahramani. "Bayesian Convolutional Neural Networks with Bernoulli Approximate Variational Inference." *arXiv preprint arXiv:1506.02158*, January 2016.

[15] ———. "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning." *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

[16] Gal, Yarin, Jiri Hron, and Alex Kendall. "Concrete Dropout." *arXiv preprint arXiv:1705.07832*, May 22, 2017.

[17] Garnelo, Marta, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J. Rezende, S. M. Ali Eslami, and Yee Whye Teh. "Neural Processes." *arXiv preprint arXiv:1807.01622*, July 4, 2018.

[18] Gill, Jeff. *Bayesian Methods A Social and Behavioral Sciences Approach*. 3rd ed. Boca Raton, FL: Taylor & Francis Group, 2015.

[19] Goodfellow, Ian, Yushua Bengio, and Aaron Courville. *Deep Learning*. Cambridge, MA: MIT Press, n.d.

[20] Görtler, Jochen, Rebecca Kehlbeck, and Oliver Deussen. "A Visual Exploration of Gaussian Processes." *Distill*, April 2, 2019. https://doi.org/10.23915/distill.00017.

[21] Graves, Alex. "Practical Variational Inference for Neural Networks." *Advances in neural information processing systems*, 2011, 2348-56.

[22] Heek, Jonathan, and Nal Kalchbrenner. "Bayesian Inference for Large Scale Image Classification." *arXiv preprint arXiv:1908.03491*, August 9, 2019.

[23] Hinton, Geoffrey E., and Drew van Camp. "Keeping Neural Networks Simple by Minimizing the Description Length of the Weights." *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, 1993.

[24] Hubel, David H., and Torsten N. Wiesel. "Effects of Monocular Deprivation in Kittens." *Naunyn-Schmiedebergs Archiv for Experimentelle Pathologie und Pharmakologie*, 1964.

[25] Huber, Peter J. *Robust Statistics*. Cambridge, MA: John Wiley & Sons, 1981.

[26] Hüllermeier, Eyke, and Willem Waegeman. "Aleatoric and Epistemic Uncertainty in Machine Learning: A Tutorial Introduction." *arXiv preprint arXiv:1910.09457*, October 21, 2019.

[27] Hutter, F. "Automating the Conguration of Algorithms for Solving Hard Computational Problems." PhD diss., University of British Columbia, 2009.

[28] Hutter, Frank, Holger Hoos, and Kevin Leyton-Brown. "An Efficient Approach for Assessing Hyperparameter Importance." *Proceedings of the 31 st International Conference on Machine Learning*, 2014.

[29] Jones, D. R., M. Schonlau, and W. J. Welch. "Efficient Global Optimization of Expensive Black-box Functions." *J. Global Optimization*, 1998, 455-92.

[30] Kendall, Alex, and Yarin Gal. "What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?" *arXiv preprint arXiv:1703.04977*, October 2017.

[31] Keydana, Sigrid. "You Sure? A Bayesian Approach to Obtaining Uncertainty Estimates from Neural Networks." *TensorFlow for R Blog*. Entry posted November 11, 2018. https://blogs.rstudio.com/tensorflow/posts/2018-11-12-uncertainty_estimates_dropout/.

[32] Kingma, Diederik P., Tim Salimans, and Max Welling. "Variational Dropout and the Local Reparameterization Trick." *arXiv preprint arXiv:1506.02557*, December 2015.

[33] Kingma, D. P., and M. Welling. "Auto-encoding Variational Bayes." *Proceedings of the 2nd International Conference on Learning Representations*, 2014.

[34] Kushner, H. J. "A New Method of Locating the Maximum Point of an Arbitrary Multipeak Curve in the Presence of Noise." *J. Basic Engineering* 86 (March 1, 1964): 97-106.

[35] Kwon, Yongchan, Joong-Ho Won, Beom Joon Kim, and Myunghee Cho Paik. "Uncertainty Quantification Using Bayesian Neural Networks in Classification: Application to Ischemic Stroke Lesion Segmentation." *1st Conference on Medical Imaging with Deep Learning*, 2018.

[36] MacKay, David. "Bayesian Methods for Adaptive Models." PhD diss., California Institute of Technology, 1992.

[37] ———. "Bayesian Neural Networks and Density Networks." *Proceedings of Workshop on Neutron Scattering Data Analysis*, 1994.

[38] Masters, Dominic, and Carlo Luschi. "Revisiting Small Batch Training for Deep Neural Networks." *arXiv preprint arXiv:1804.07612*, April 2018.

[40] Mockus, J. "Application of Bayesian Approach to Numerical Methods of Global and Stochastic Optimization." *J. Global Optimization* 4 (1994): 347-65.

[41] Mockus, J., V. Tiesis, and A. Zilinskas. "The Application of Bayesian Methods for Seeking the Extremum." *Toward Global Optimization* Volume 2 (1978): 117-28.

[42] Munkhdalai, Tsendsuren, and Hong Yu. "Meta Networks." *arXiv preprint arXiv:1703.00837*, June 2017.

[43] Neal, Radford. "Bayesian Learning for Neural Networks." PhD diss., University of Toronto, 1995.

[44] Nichol, Alex, Joshua Achiam, and John Schulman. "On First-Order Meta-Learning Algorithms." *arXiv preprint arXiv:1803.02999*, October 2018.

[45] Rasmussen, Carl Edward, and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[46] Rumelhart, D., G. Hinton, and R. Williams. "Learning Representations by Back-propagating Errors." *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*, 1986.

[47] Shridhar, Kumar, Felix Laumann, and Marcus Liwicki. "A Comprehensive Guide to Bayesian Convolutional Neural Network with Variational Inference." *arXiv preprint arXiv:1901.02731*, 2019.

[48] ———. "Uncertainty Estimations by Softplus Normalization in Bayesian Convolutional Neural Networks with Variational Inference." *arXiv preprint arXiv:1806.05978*, May 2019.

[49] Snoek, Jasper, Hugo Larochelle, and Ryan P. Adams. "Practical Bayesian Optimization of Machine Learning Algorithms." *arXiv preprint arXiv:1206.2944*, August 29, 2012.

[50] Srinivas, Niranjan, Andreas Krause, Sham M. Kakade, and Matthias Seeger. "Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design." *arXiv preprint arXiv:0912.3995*, June 9, 2010.

[51] Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." *Journal of Machine Learning Research 15*, 2014.

[52] Sun, Shengyang, Guodong Zhang, Jiaxin Shi, and Roger Grosse. "Functional Variational Bayesian Neural Networks." *arXiv preprint arXiv:1903.05779*, March 14, 2019.

[53] Sutton, Richard S., and Andrew G. Barto. *Reinforcement Learning: An Introduction*. N.p.: MIT Press, 2018.

[54] Tomczak, Marcin B., Siddharth Swaroop, and Richard E. Turner. "Neural Network Ensembles and Variational Inference Revisited." *1st Symposium on Advances in Approximate Bayesian Inference*, 2018.

[55] Tracey, Brendan D., and David H. Wolpert. "Upgrading from Gaussian Processes to Student's-T Processes." *arXiv preprint arXiv:1801.06147*, January 18, 2018.

[56] Tran, Dustin, Danijar Hafner, Mark van der Wilk, and Michael W. Dusenberry. "Bayesian Layers: A Module for Neural Network Uncertainty." *arXiv preprint arXiv:1812.03973*, March 5, 2019.

[57] Vilalta, Ricardo, and Youssef Drissi. "A Perspective View and Survey of Meta-Learning." *Kluwer Academic Publishers*, 2002.

[58] Wang, Ziyu, and Nando de Freitas. "Theoretical Analysis of Bayesian Optimisation with Unknown Gaussian Process Hyper-Parameters." *arXiv preprint arXiv:1406.7758*, June 30, 2014.

[59] Wen, Yeming, Paul Vicol, Jimmy Ba, Dustin Tran, and Roger Grosse. "Flipout: Efficient Pseudo-Independent Weight Perturbations on Mini-Batches." *arXiv preprint arXiv:1803.04386*, April 2, 2018.