Bayesian Convolutional Neural Network

Kumar Shridhar

Supervisor: Prof. Dr. Marcus Liwicki, Felix Laumann

Department of Computer Science
University of Kaiserslautern

This thesis is submitted in fulfillment for the degree of
Master of Science

January 2019

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this thesis are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This thesis is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and in acknowledgements.

Kumar Shridhar
January 2019

Acknowledgements

I would first like to thank my thesis advisor **Prof. Marcus Liwicki** who was always present physically or virtually whenever I ran into some trouble or had a crazy thought. He consistently steered me in the right direction whenever he thought I needed it.

I would also like to thank my second thesis advisor **Felix Laumann** with whom I worked in Copenhagen and in Kaiserslautern. He also made sure Imperial College, London is never far away with our continuous calls and Skype discussion. He contributed in all possible ways in the thesis and made sure we are on the right path.

I would also like to acknowledge **University of Kaiserlautern** for providing me with the opportunity for writing the thesis and **MindGarage** for providing me with the computation power.

I must express my very profound gratitude to my parents (**Vivek** and **Anita**), to my sister (**Mineshi**) and to **Purvanshi Mehta** for providing me with unfailing support and continuous encouragement throughout the thesis.

Finally, I am thankful to **Ashutosh Mishra**, **Saurabh Varshneya**, and **Ayushman Dash** for their invaluable comments and endless discussion. I am also thankful to **Sadique Adnan Siddiqui** for his late night tea and delicious food. This accomplishment would not have been possible without them.

I would also like to mention **BotSupply** for providing me with the funds to go and meet new people to discuss ideas and to get some inspiration.

Abstract

Artificial Neural Networks are connectionist systems that perform a given task by learning on examples without having prior knowledge about the task. This is done by finding an optimal point estimate for the weights in every node. Generally, the network using point estimates as weights perform well with large datasets, but they fail to express uncertainty in regions with little or no data, leading to overconfident decisions.

In this thesis, Bayesian Convolutional Neural Network (BayesCNN) using Variational Inference is proposed, that introduces probability distribution over the weights. Furthermore, the proposed BayesCNN architecture is applied to tasks like Image Classification, Image Super-Resolution and Generative Adversarial Networks.

BayesCNN is based on Bayes by Backprop which derives a variational approximation to the true posterior. Our proposed method not only achieves performances equivalent to frequentist inference in identical architectures but also incorporate a measurement for uncertainties and regularisation. It further eliminates the use of dropout in the model. Moreover, we predict how certain the model prediction is based on the epistemic and aleatoric uncertainties and finally, we propose ways to prune the Bayesian architecture and to make it more computational and time effective.

In the first part of the thesis, the Bayesian Neural Network is explained and it is applied to an Image Classification task. The results are compared to point-estimates based architectures on MNIST, CIFAR-10, and CIFAR-100 datasets. Moreover, uncertainties are calculated and the architecture is pruned and a comparison between the results is drawn.

In the second part of the thesis, the concept is further applied to other computer vision tasks namely, Image Super-Resolution and Generative Adversarial Networks. The concept of BayesCNN is tested and compared against other concepts in a similar domain.

Table of contents

List of figures xiii

List of tables xv

Nomenclature xvii

1 Introduction 1

1.1 Problem Statement 1

1.2 Current Situation 2

1.3 Our Hypothesis 3

1.4 Our Contribution 4

2 Background 5

2.1 Neural Networks 6

2.1.1 Brain Analogies 6

2.1.2 Neural Network 6

2.1.3 Convolutional Neural Network 7

2.2 Probabilistic Machine Learning 8

2.2.1 Variational Inference 8

2.2.2 Local Reparametrisation Trick 10

2.3 Uncertainties in Bayesian Learning 10

2.4 Backpropagation 12

2.4.1 Bayes by Backprop 12

2.5 Model Weights Pruning 13

3 Related Work 15

3.1 Related Work 16

3.1.1 Bayesian Training 16

3.1.2 Uncertainty Estimation 17

3.1.3 Model Pruning 17

4 Concept **19**

4.1 Bayesian Convolutional Neural Networks with Variational Inference 20

4.1.1 Local Reparameterization Trick for Convolutional Layers 20

4.1.2 Applying two Sequential Convolutional Operations (Mean and Variance) 21

4.2 Uncertainty Estimation in CNN 22

4.3 Model Pruning 24

5 Empirical Analysis **27**

5.1 Experimentation Methodology 28

5.2 Case Study 1: Small Datasets (MNIST, CIFAR-10) 30

5.2.1 Datasets 31

5.2.2 Results 31

5.3 Case Study 2: Large Dataset (CIFAR-100) 35

5.3.1 Dataset 35

5.3.2 Results 36

5.4 Uncertainity Estimation 38

5.5 Model Pruning 39

5.6 Training Time 41

6 Applications **43**

6.1 BayesCNN for Image Super Resolution 44

6.1.1 Our Approach 44

6.1.2 Empirical Analysis 46

6.2 BayesCNN for Generative Adversarial Networks 47

6.2.1 Our Approach 47

6.2.2 Empirical Analysis 49

7 Conclusion and Outlook **53**

References **57**

Appendix A Experiment Specifications **63**

A.0.1 Image Recognition 63

A.0.2 Image Super Resolution 64

A.0.3 Generative Adversarial Network 65

A.0.4 Image Recognition 66

A.0.5 LeNet-5 67
A.0.6 AlexNet 68
A.0.7 AlexNetHalf 69

Appendix B How to replicate results **71**
B.0.1 Image Recognition 71

List of figures

1.1 Top: Each filter weight has a fixed value, as in the case of frequentist Convolutional Networks. Bottom: Each filter weight has a distribution, as in the case of Bayesian Convolutional Networks. [16] 3

2.1 Biologically inspired Neural Network [29] 6
2.2 Neural Network with two hidden layers 7

4.1 Input image with exemplary pixel values, filters, and corresponding output with point-estimates (top) and probability distributions (bottom) over weights.[55] 20
4.2 Fully Bayesian perspective of an exemplary CNN. Weights in filters of convolutional layers, and weights in fully-connected layers have the form of a probability distribution. [55] 21
4.3 Predictive distributions is estimated for a low-dimensional active learning task. The predictive distributions are visualized as mean and two standard deviations shaded. 🟦 shows the epistemic uncertainty and 🟥 shows the aleatoric noise. Data points are shown in ■. **(Left)** A deterministic network conflates uncertainty as part of the noise and is overconfident outside of the data distribution. **(Right)** A variational Bayesian neural network with standard normal prior represents uncertainty and noise separately but is overconfident outside of the training distribution as defined by [22] 24

5.1 Comparison of Validation Accuracies of Bayesian AlexNet and LeNet-5 with frequentist approach on MNIST and CIFAR-10 datasets 33
5.2 Convergence of the standard deviation of the Gaussian variational posterior probability distribution $q_\theta(w|\mathscr{D})$ of a random model parameter at epochs 1, 5, 20, 50, and 100. MNIST is trained on Bayesian LeNet-5. 34

5.3 Convergence of the Gaussian variational posterior probability distribution $q_\theta(w|\mathscr{D})$ of a random model parameter at epochs 1, 5, 20, 50, and 100. CIFAR-10 is trained on Bayesian LeNet-5. 35
5.4 Comparison of Training and Validation Accuracies of Bayesian AlexNet and LeNet-5 with frequentist approach with and without dropouts on CIFAR-100 datasets 37

6.1 The proposed efficient sub-pixel convolutional neural network (ESPCN) [53], with two convolution layers for feature maps extraction, and a sub-pixel convolution layer that aggregates the feature maps from Low Resolution space and builds the Super Resolution image in a single step. 45
6.2 Sample image in Low Resolution image space taken randomly from BSD 300 [46] dataset. 46
6.3 Generated Super Resolution Image scaled to 40 percent to fit 46
6.4 Fake Samples generated from the Bayesian DCGAN model trained on CIFAR10 dataset 50
6.5 Real Samples taken from CIFAR10 dataset 51

List of tables

5.1 Comparison of validation accuracies (in percentage) for different architectures with variational inference (VI), frequentist inference and Dropout as a Bayesian approximation as proposed by Gal and Ghahramani [13] for MNIST, and CIFAR-10. 32

5.2 Comparison of validation accuracies (in percentage) for different architectures with variational inference (VI), frequentist inference and Dropout as a Bayesian approximation as proposed by Gal and Ghahramani [13] for MNIST, CIFAR-10, and CIFAR-100. 36

5.3 Comparison of training and validation accuracies (in percentage) for AlexNet architecture with variational inference (VI) and frequentist inference for CIFAR-100. 38

5.4 Aleatoric and epistemic uncertainty for Bayesian LeNet-5 calculated for MNIST and CIFAR-10, computed as proposed by Kwon et al. [36]. 38

5.5 AlexNetHalf with number of filters halved compared to the original architecture. 39

5.6 Comparison of validation accuracies (in percentage) for AlexNet with variational inference (VI), AlexNet with frequentist inference and AlexNet with half number of filters halved for MNIST, CIFAR-10 and CIFAR-100 datasets. . 40

5.7 Comparison of validation accuracies (in percentage) for AlexNet with variational inference (VI), AlexNet with frequentist inference and BayesianAlexNet with L1 norm applied for MNIST and CIFAR-10 datasets. 41

6.1 Network Architecture for Bayesian Super Resolution 45

6.2 Generator architecture as defined in the paper. [52] 48

6.3 Discriminator architecture with Bayesian Convolutional layers 48

A.1 LeNet architecture with original configurations as defined in the paper. [39] 67

A.2 AlexNet architecture with original configurations as defined in the paper. [34] 68

A.3 AlexNetHalf with number of filters halved compared to the original architecture. 69

Nomenclature

Acronyms / Abbreviations

ANN	Artificial Neural Networks
CIFAR	Canadian Institute for Advanced Research
CNN	Convolution Neural Networks
Cov	Covariance
DCGAN	Deep Convolutional Generative Adversarial Networks
DNN	Deep Neural Networks
GAN	Generative Adversarial Networks
GRU	Gated Recurrent Units
HR	High Resolution
KL	Kullback–Leibler
LGB	Lateral Geniculate Body
LR	Low Resolution
MAP	maximum a posteriori
μ_{ijhw}	Mean
MLE	Maximum Likelihood Estimation
MNIST	Modified National Institute of Standards and Technology
NN	Neural Networks

$ReLU$	Rectified Linear Unit
RNN	Recurrent Neural Networks
$SGVB$	Stochastic Gradient Variational Bayes
SR	Super Resolution
$\alpha_{ijhw}\mu^2_{ijhw}$	Variance
VI	Variational Inference

Chapter 1

Introduction

Deep Neural Networks (DNNs), are connectionist systems that learn to perform tasks by learning on examples without having prior knowledge about the tasks. They easily scale to millions of data points and yet remain tractable to optimize with stochastic gradient descent.

Convolutional neural networks (CNNs), a variant of DNNs, have already surpassed human accuracy in the realm of image classification (e.g. [24, 56, 34]). Due to the capacity of CNNs to fit on a wide diversity of non-linear data points, they require a large amount of training data. This often makes CNNs and Neural Networks in general, prone to overfitting on small datasets. The model tends to fit well to the training data, but are not predictive for new data. This often makes the Neural Networks incapable of correctly assessing the uncertainty in the training data and hence leads to overly confident decisions about the correct class, prediction or action.

Various regularization techniques for controlling over-fitting are used in practice namely early stopping, weight decay, L1, L2 regularizations and currently the most popular and empirically effective technique being *dropout* [25].

1.1 Problem Statement

Despite Neural Networks architectures achieving state-of-the-art results in almost all classification tasks, Neural Networks still make over-confident decisions. A measure of uncertainty in the prediction is missing from the current Neural Networks architectures. Very careful training, weight control measures like regularization of weights and similar techniques are needed to make the models susceptible to over-fitting issues.

We address both of these concerns by introducing Bayesian learning to Convolutional Neural Networks that adds a measure of uncertainty and regularization in their predictions.

1.2 Current Situation

Deep Neural Networks have been successfully applied to many domains, including very sensitive domains like health-care, security, fraudulent transactions and many more. However, from a probability theory perspective, it is unjustifiable to use single point-estimates as weights to base any classification on. On the other hand, Bayesian neural networks are more robust to over-fitting, and can easily learn from small datasets. The Bayesian approach further offers uncertainty estimates via its parameters in form of probability distributions (see Figure 1.1). At the same time, by using a prior probability distribution to integrate out the parameters, the average is computed across many models during training, which gives a regularization effect to the network, thus preventing overfitting.

Bayesian posterior inference over the neural network parameters is a theoretically attractive method for controlling overfitting; however, modelling a distribution over the kernels (also known as filters) of a CNNs has never been attempted successfully before, perhaps because of the vast number of parameters and extremely large models commonly used in practical applications.

Even with a small number of parameters, inferring model posterior in a Bayesian NN is a difficult task. Approximations to the model posterior are often used instead, with the variational inference being a popular approach. In this approach one would model the posterior using a simple *variational* distribution such as a Gaussian, and try to fit the distribution's parameters to be as close as possible to the true posterior. This is done by minimising the *Kullback-Leibler divergence* from the true posterior. Many have followed this approach in the past for standard NN models [26, 3, 19, 4]. But the variational approach used to approximate the posterior in Bayesian NNs can be fairly computationally expensive – the use of Gaussian approximating distributions increases the number of model parameters considerably, without increasing model capacity by much. Blundell et al. [4] for example used Gaussian distributions for Bayesian NN posterior approximation and have doubled the number of model parameters, yet report the same predictive performance as traditional approaches using dropout. This makes the approach unsuitable in practice to use with CNNs as the increase in the number of parameters is too costly.

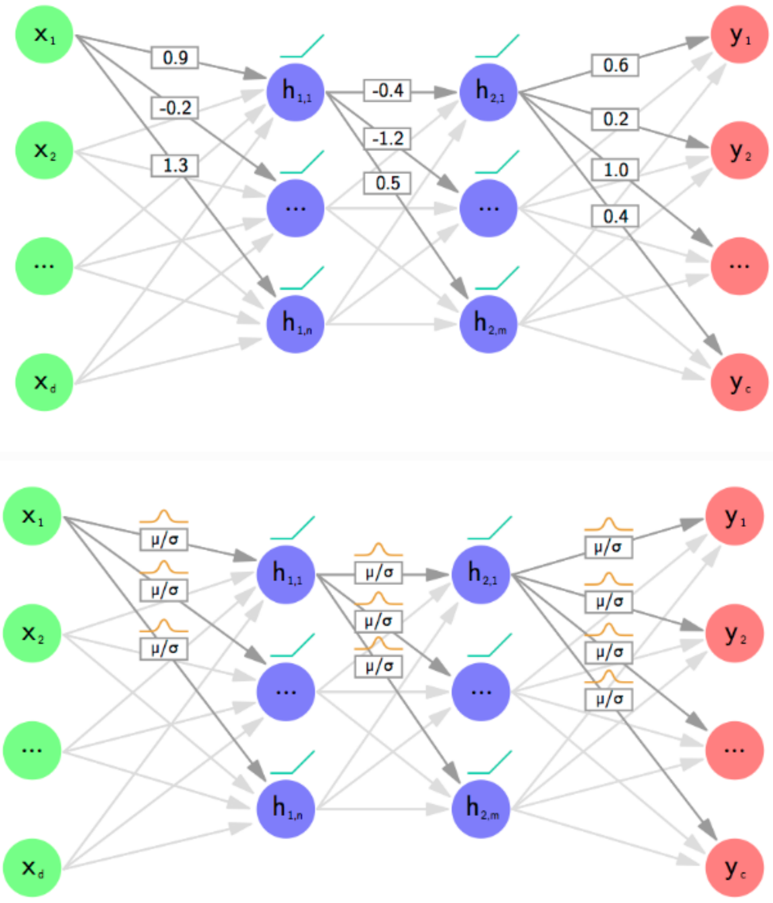

Fig. 1.1 Top: Each filter weight has a fixed value, as in the case of frequentist Convolutional Networks. Bottom: Each filter weight has a distribution, as in the case of Bayesian Convolutional Networks. [16]

1.3 Our Hypothesis

We build our Bayesian CNN upon *Bayes by Backprop* [19, 4]. The exact Bayesian inference on the weights of a neural network is intractable as the number of parameters is very large and the functional form of a neural network does not lend itself to exact integration. So, we approximate the intractable true posterior probability distributions $p(w|\mathcal{D})$ with variational probability distributions $q_\theta(w|\mathcal{D})$, which comprise the properties of Gaussian distributions $\mu \in \mathbb{R}^d$ and $\sigma \in \mathbb{R}^d$, denoted by $\mathcal{N}(\theta|\mu, \sigma^2)$, where d is the total number of parameters defining a probability distribution. The shape of these Gaussian variational posterior probability distributions, determined by their variance σ^2, expresses an uncertainty estimation of every model parameter.

1.4 Our Contribution

The main contributions of our work are as follows:

1. We present how *Bayes by Backprop* can be efficiently applied to CNNs. We, therefore, introduce the idea of applying two convolutional operations, one for the mean and one for the variance.
2. We show how the model learns richer representations and predictions from cheap model averaging.
3. We empirically show that our proposed generic and reliable variational inference method for Bayesian CNNs can be applied to various CNN architectures without any limitations on their performances.
4. We examine how to estimate the aleatoric and epistemic uncertainties and empirically show how the uncertainty can decrease, allowing the decisions made by the network to become more deterministic as the training accuracy increases.
5. We also empirically show how our method typically only doubles the number of parameters yet trains an infinite ensemble using unbiased Monte Carlo estimates of the gradients.
6. We also apply L1 norm to the trained model parameters and prune the number of non zero values and further, fine-tune the model to reduce the number of model parameters without a reduction in the model prediction accuracy.
7. Finally, we will apply the concept of Bayesian CNN to tasks like Image Super-Resolution and Generative Adversarial Networks and we will compare the results with other prominent architectures in the respective domain.

This work builds on the foundations laid out by Blundell et al. [4], who introduced *Bayes by Backprop* for feedforward neural networks. Together with the extension to recurrent neural networks, introduced by Fortunato et al. [11], *Bayes by Backprop* is now applicable on the three most frequently used types of neural networks, i.e., feedforward, recurrent, and convolutional neural networks.

Chapter 2

Background

Chapter Overview

- Neural Networks and Convolutional Neural Networks.
- Concepts overview of Variational Inference, and local reparameterization trick in Bayesian Neural Network.
- Backpropagation in Bayesian Networks using Bayes by Backprop.
- Estimation of Uncertainties in a network.
- Pruning a network to reduce the number of overall parameters without affecting its performance.

2.1 Neural Networks

2.1.1 Brain Analogies

A perceptron is conceived as a mathematical model of how the neurons function in our brain by a famous psychologist Rosenblatt. According to Rosenblatt, a neuron takes a set of binary inputs (nearby neurons), multiplies each input by a continuous-valued weight (the synapse strength to each nearby neuron), and thresholds the sum of these weighted inputs to output a 1 if the sum is big enough and otherwise a 0 (the same way neurons either fire or do not fire).

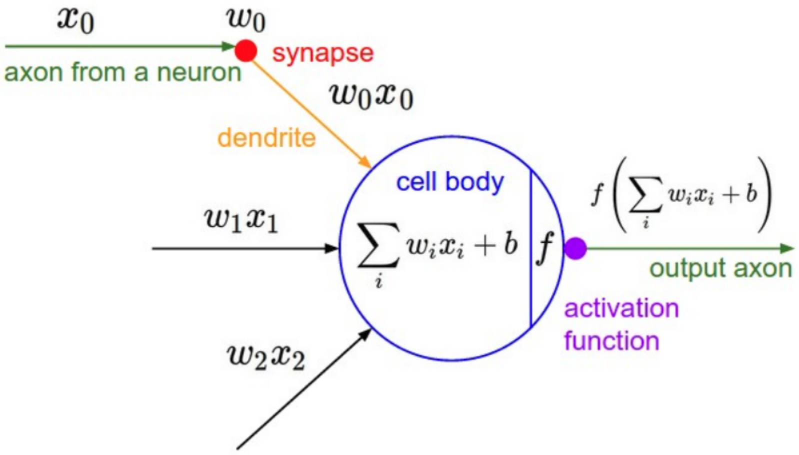

Fig. 2.1 Biologically inspired Neural Network [29]

2.1.2 Neural Network

Inspired by the biological nervous system, the structure of an Artificial Neural Network (ANN) was developed to process information similar to how brain process information. A large number of highly interconnected processing elements (neurons) working together makes a Neural Network solve complex problems. Just like humans learn by example, so does a Neural Network. Learning in biological systems involves adjustments to the synaptic connections which is similar to weight updates in a Neural Network.

A Neural Network consists of three layers: input layer to feed the data to the model to learn representation, hidden layer that learns the representation and the output layer that outputs the results or predictions. Neural Networks can be thought of an end to end system that finds patterns in data which are too complex to be recognized by a human to teach to a machine.

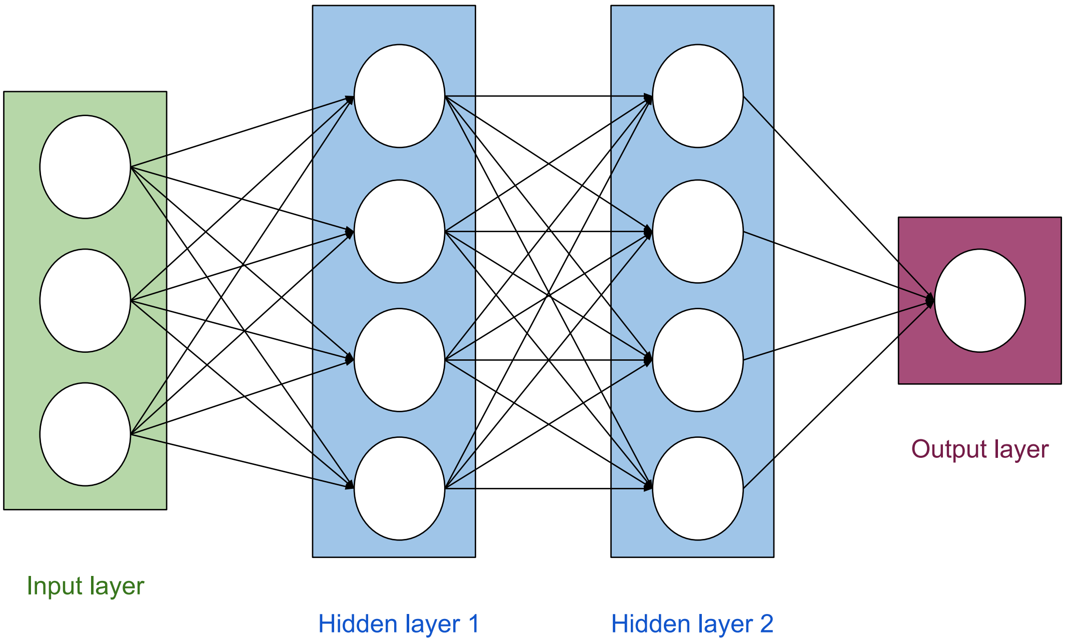

Fig. 2.2 Neural Network with two hidden layers

2.1.3 Convolutional Neural Network

Hubel and Wiesel in their hierarchy model mentioned a neural network to have a hierarchy structure in the visual cortex. LGB (lateral geniculate body) forms the simple cells that form the complex cells which form the lower order hypercomplex cells that finally form the higher order hypercomplex cells. Also, the network between the lower order hypercomplex cells and the higher order hypercomplex cells are structurally similar to the network between simple cells and the complex cells. In this hierarchy, a cell in a higher stage generally has a tendency to respond selectively to a more complicated feature of the stimulus pattern, and the cell at the lower stage responds to simpler features. Also, higher stage cells possess a larger receptive field and are more insensitive to the shift in the position of the stimulus pattern.

Similar to a hierarchy model, a neural network starting layers learns simpler features like edges and corners and subsequent layers learn complex features like colours, textures and so on. Also, higher neural units possess a larger receptive field which builds over the initial layers. However, unlike in multilayer perceptron where all neurons from one layer are connected with all the neurons in the next layer, weight sharing is the main idea behind a convolutional neural network. Example: instead of each neuron having a different weight for each pixel of the input image (28*28 weights), the neurons only have a small set of weights (5*5) that is applied to a whole bunch of small subsets of the image of the same size. Layers past the first layer work in a similar way by taking in the ‘local’ features found in the previously hidden layer rather than pixel images, and successively see larger portions of the image since they are combining information about increasingly larger subsets of the image. Finally, the final layer makes the correct prediction for the output class.

The reason for why this is helpful is intuitive if not mathematically clear: without such constraints, the network would have to learn the same simple things (such as detecting edges, corners, etc) a whole bunch of times for each portion of the image. But with the constraint there, only one neuron would need to learn each simple feature - and with far fewer weights overall, it could do so much faster! Moreover, since the pixel-exact locations of such features do not matter the neuron could basically skip neighbouring subsets of the image - subsampling, now known as a type of pooling - when applying the weights, further reducing the training time. The addition of these two types of layers - convolutional and pooling layers - are the primary distinctions of Convolutional Neural Nets (CNNs/ConvNets) from plain old neural nets.

2.2 Probabilistic Machine Learning

2.2.1 Variational Inference

We define a function $y = f(\mathbf{x})$ that estimates the given inputs $\{x_1, \dots, x_N\}$ and their corresponding outputs $\{y_1, \dots, y_N\}$ and produces an predictive output. Using Bayesian inference, a prior distribution is used over the space of functions $p(f)$. This distribution represents our prior belief as to which functions are likely to have generated our data.

A likelihood is defined as $p(Y|f,X)$ to capture the process in which given a function observation is generated. We use the Bayes rule to find the posterior distribution given our dataset: $p(f|X,Y)$.

The new output can be predicted for a new input point x^* by integrating over all possible functions f,

$$p(y^*|x^*,X,Y)=\int p(y^*|f^*)p(f^*|x^*,X,Y)df^* \tag{2.1}$$

The equation (2.1) is intractable due to the integration sign. We can approximate it by taking a finite set of random variables w and conditioning the model on it. However, it is based on a modelling assumption that the model depends on these variables alone, and making them into sufficient statistics in our approximate model.

The predictive distribution for a new input point x^* is then given by

$$p(y^*|x^*,X,Y)=\int p(y^*|f^*)p(f^*|x^*,w)p(w|X,Y)df^*dw.$$

However, the distribution $p(w|X,Y)$ still remains intractable and we need to approximate it with a variational distribution $q(w)$, that can be computed. The approximate distribution needs to be as close as possible to the posterior distribution obtained from the original model. We thus minimise the Kullback–Leibler (KL) divergence, intuitively a measure of similarity between two distributions: $KL(q(w)\ ||\ p(w|X,Y))$, resulting in the approximate predictive distribution

$$q(y^*|x^*)=\int p(y^*|f^*)p(f^*|x^*,w)q(w)df^*dw. \tag{2.2}$$

Minimising the Kullback–Leibler divergence is equivalent to maximising the *log evidence lower bound,*

$$KL_{\text{VI}}:=\int q(w)p(F|X,w)\log p(Y|F)dFdw-KL(q(w)||p(w)) \tag{2.3}$$

with respect to the variational parameters defining $q(w)$. This is known as *variational inference,* a standard technique in Bayesian modelling.

Maximizing the KL divergence between the posterior and the prior over w will result in a variational distribution that learns a good representation from the data (obtained from log likelihood) and is closer to the prior distribution. In other words, it can prevent overfitting.

2.2.2 Local Reparametrisation Trick

The ability to rewrite statistical problems in an equivalent but different form, to reparameterise them, is one of the most general-purpose tools used in mathematical statistics. The type of *reparameterization* when the global uncertainty in the weights is translated into a form of local uncertainty which is independent across examples is known as the *local reparameterization trick*. An alternative estimator is deployed for which $\mathrm{Cov}\left[L_i, L_j\right] = 0$, so that the variance of the stochastic gradients scales as $1/M$. The new estimator is made computationally efficient by sampling the intermediate variables and not sampling ϵ directly, but only $f(\epsilon)$ through which ϵ influences $L_{\mathscr{D}}^{\mathrm{SGVB}}(\phi)$. Hence, the source of global noise can be translated to local noise ($\epsilon \rightarrow f(\epsilon)$), a local reparameterization can be applied so as to obtain a statistically efficient gradient estimator.

The technique can be explained through a simple example: We consider an input(X) of random uniform function ranging from -1 to +1 and an output(Y) as a random normal distribution around mean X and standard deviation δ. The Mean Squared Loss would be defined as $(Y - X)^2$. The problem here is during the backpropagation from the random normal distribution function. As we are trying to propagate through a stochastic node we reparameterize by adding X to the random normal function output and multiplying by δ. The movement of parameters out of the normal distribution does not change the behaviour of the model.

2.3 Uncertainties in Bayesian Learning

Uncertainties in a network is a measure of how certain the model is with its prediction. In Bayesian modelling, there are two main types of uncertainty one can model [9]: *Aleatoric* uncertainty and *Epistemic* uncertainty.

Aleatoric uncertainty measures the noise inherent in the observations. This type of uncertainty is present in the data collection method like the sensor noise or motion noise which is uniform along the dataset. This cannot be reduced if more data is collected. *Epistemic* uncertainty, on the other hand, represents the uncertainty caused by the model. This uncertainty can be reduced given more data and is often referred to as *model uncertainty*. Aleatoric uncertainty can further be categorized into *homoscedastic* uncertainty, uncertainty which stays constant for different inputs, and *heteroscedastic* uncertainty. Heteroscedastic uncertainty depends on the inputs to the model, with some inputs potentially having more noisy outputs than others. Heteroscedastic uncertainty is in particular important so that model prevents from outputting very confident decisions.

Current work measures uncertainties by placing a probability distributions over either the model parameters or model outputs. Epistemic uncertainty is modelled by placing a prior distribution over a model's weights and then trying to capture how much these weights vary given some data. Aleatoric uncertainty, on the other hand, is modelled by placing a distribution over the output of the model.

Sources of Uncertainities

The following can be the source of uncertainty as mentioned by Kiureghian [9]:

1. Uncertainty inherent in the basic random variables X, such as the uncertainty inherent in material property constants and load values, which can be directly measured.

2. Uncertain model error resulting from the selection of the form of the probabilistic sub-model $f_X(x, H_f)$ used to describe the distribution of basic variables.

3. Uncertain modeling errors resulting from selection of the physical sub-models $g_i(x, H_g), i = 1, 2, ..., m$, used to describe the derived variables.

4. Statistical uncertainty in the estimation of the parameters H_f of the probabilistic sub-model.

5. Statistical uncertainty in the estimation of the parameters H_g of the physical sub-models.

6. Uncertain errors involved in measuring of observations, based on which the parameters H_f and H_g are estimated. These include errors involved in indirect measurement, e.g., the measurement of a quantity through a proxy, as in non-destructive testing of material strength.

7. Uncertainty modelled by the random variables Y corresponding to the derived variables y, which may include, in addition to all the above uncertainties, uncertain errors resulting from computational errors, numerical approximations or truncations. For example, the computation of load effects in a nonlinear structure by a finite element procedure employs iterative calculations, which invariably involve convergence tolerances and truncation errors.

2.4 Backpropagation

Backpropagation in a Neural Networks was proposed by Rumelhart [10] in 1986 and it is the most commonly used method for training neural networks. Backpropagation is a technique to compute the gradient of the loss in terms of the network weights. It operates in two phases: firstly, the input features through the network propagates in the forward direction to compute the function output and thereby the loss associated with the parameters. Secondly, the derivatives of the training loss with respect to the weights are propagated back from the output layer towards the input layers. These computed derivatives are further used to update the weights of the network. This is a continuous process and updating of the weight occurs continuously over every iteration.

Despite the popularity of backpropagation, there are many hyperparameters in backpropagation based stochastic optimization that requires specific tuning, e.g., learning rate, momentum, weight decay, etc. The time required for finding the optimal values is proportional to the data size. For a network trained with backpropagation, only point estimates of the weights are achieved in the network. As a result, these networks make overconfident predictions and do not account for uncertainty in the parameters. Lack of uncertainty measure makes the network prone to overfitting and a need for regularization.

A Bayesian approach to Neural Networks provides the shortcomings with the backpropagation approach [44] as Bayesian methods naturally account for uncertainty in parameter estimates and can propagate this uncertainty into predictions. Also, averaging over parameter values instead of just choosing single point estimates makes the model robust to overfitting.

Sevreal approaches has been proposed in the past for learning in Bayesian Networks: Laplace approximation [45], MC Dropout [13],and Variational Inference [26] [19] [4]. We used Bayes by Backprop [4] for our work and is explained next.

2.4.1 Bayes by Backprop

Bayes by Backprop [19, 4] is a variational inference method to learn the posterior distribution on the weights $w \sim q_\theta(w|\mathscr{D})$ of a neural network from which weights w can be sampled in backpropagation. It regularises the weights by minimising a compression cost, known as the variational free energy or the expected lower bound on the marginal likelihood.

Since the true posterior is typically intractable, an approximate distribution $q_\theta(w|\mathscr{D})$ is defined that is aimed to be as similar as possible to the true posterior $p(w|\mathscr{D})$, measured by the Kullback-Leibler (KL) divergence [35]. Hence, we define the optimal parameters

θ^{opt} as

$$\begin{aligned}\theta^{opt} &= \arg\min_{\theta} \text{KL}\left[q_\theta(w|\mathscr{D}) \,\|\, p(w|\mathscr{D})\right] \\ &= \arg\min_{\theta} \text{KL}\left[q_\theta(w|\mathscr{D}) \,\|\, p(w)\right] \\ &\quad - \mathbb{E}_{q(w|\theta)}[\log p(\mathscr{D}|w)] + \log p(\mathscr{D})\end{aligned} \tag{2.4}$$

where

$$\text{KL}\left[q_\theta(w|\mathscr{D}) \,\|\, p(w)\right] = \int q_\theta(w|\mathscr{D}) \log \frac{q_\theta(w|\mathscr{D})}{p(w)} dw. \tag{2.5}$$

This derivation forms an optimisation problem with a resulting cost function widely known as *variational free energy* [50, 63, 12] which is built upon two terms: the former, $\text{KL}\left[q_\theta(w|\mathscr{D}) \,\|\, p(w)\right]$, is dependent on the definition of the prior $p(w)$, thus called complexity cost, whereas the latter, $\mathbb{E}_{q(w|\theta)}[\log p(\mathscr{D}|w)]$, is dependent on the data $p(\mathscr{D}|w)$, thus called likelihood cost. The term $\log p(\mathscr{D})$ can be omitted in the optimisation because it is constant.

Since the KL-divergence is also intractable to compute exactly, we follow a stochastic variational method [19, 4]. We sample the weights w from the variational distribution $q_\theta(w|\mathscr{D})$ since it is much more probable to draw samples which are appropriate for numerical methods from the variational posterior $q_\theta(w|\mathscr{D})$ than from the true posterior $p(w|\mathscr{D})$. Consequently, we arrive at the tractable cost function (5.2) which is aimed to be optimized, i.e. minimised w.r.t. θ, during training:

$$\mathscr{F}(\mathscr{D},\theta) \approx \sum_{i=1}^{n} \log q_\theta(w^{(i)}|\mathscr{D}) - \log p(w^{(i)}) - \log p(\mathscr{D}|w^{(i)}) \tag{2.6}$$

where n is the number of draws.

We sample $w^{(i)}$ from $q_\theta(w|\mathscr{D})$. The uncertainty afforded by *Bayes by Backprop* trained neural networks has been used successfully for training feedforward neural networks in both supervised and reinforcement learning environments [4, 42, 28], for training recurrent neural networks [11], but has not been applied to convolutional neural networks to-date.

2.5 Model Weights Pruning

Model pruning reduces the sparsity in a deep neural network's various connection matrices, thereby reducing the number of valued parameters in the model. The whole idea of model pruning is to reduce the number of parameters without much loss in the accuracy of the model. This reduces the use of a large parameterized model with regularization and promotes the use of dense connected smaller models. Some recent work suggests [23] [48] that the network can achieve a sizable reduction in model size, yet achieving comparable accuracy. Model pruning possesses several advantages in terms of reduction in computational cost, inference time and in energy-efficiency. The resulting pruned model typically has sparse connection matrices, so efficient inference using these sparse models requires purpose-built hardware capable of loading sparse matrices and/or performing sparse matrix-vector operations. Thus the overall memory usage is reduced with the new pruned model.

There are several ways of achieving the pruned model, the most popular one is to map the low contributing weights to zero and reducing the number of overall non-zero valued weights. This can be achieved by training a large sparse model and pruning it further which makes it comparable to training a small dense model.

Assigning weights zero to most features and non-zero weights to only important features can be formalized by applying the L_0 norm, where $L_0 = ||\theta||_0 = \sum_j \delta(\theta_j \neq 0)$, and it applies a constant penalty to all non-zero weights. L_0 norm can be thought of a feature selector norm that only assigns non-zero values to feature that are important. However, the L_0 norm is non-convex and hence, non-differentiable that makes it a NP-hard problem and can be only efficiently solved when $P = NP$. The alternative that we use in our work is the L_1 norm, which is equal to the sum of the absolute weight values, $||\theta||_1 = \sum_j |\theta_j|$. L_1 norm is convex and hence differentiable and can be used as an approximation to L_0 norm [59]. L_1 norm works as a sparsity inducing regularizer by making large number of coefficients equal to zero, working as a great feature selector in our case. Only thing to keep in mind is that the L_1 norm do not have a gradient at $\theta_j = 0$ and we need to keep that in mind. Pruning away the less salient features to zero has been used in this thesis and is explained in details in Chapter 4.

Chapter 3

Related Work

Chapter Overview

- How Bayesian Methods were applied to Neural Networks for the intractable true posterior distribution.
- Various ways of training Neural Networks posterior probability distributions: Laplace approximations, Monte Carlo and Variational Inference.
- Proposals on Dropout and Gaussian Dropout as Variational Inference schemes.
- Work done in the past for uncertainty estimation in Neural Network.
- Ways to reduce the number of parameters in a model.

3.1 Related Work

3.1.1 Bayesian Training

Applying Bayesian methods to neural networks has been studied in the past with various approximation methods for the intractable true posterior probability distribution $p(w|\mathscr{D})$. Buntine and Weigend [5] started to propose various *maximum-a-posteriori* (MAP) schemes for neural networks. They were also the first who suggested second order derivatives in the prior probability distribution $p(w)$ to encourage smoothness of the resulting approximate posterior probability distribution. In subsequent work by Hinton and Van Camp [26], the first variational methods were proposed which naturally served as a regularizer in neural networks. He also mentioned that the amount of information in weight can be controlled by adding Gaussian noise. When optimizing the trade-off between the expected error and the information in the weights, the noise level can be adapted during learning.

Hochreiter and Schmidhuber [27] suggested taking an information theory perspective into account and utilising a minimum description length (MDL) loss. This penalises non-robust weights by means of an approximate penalty based upon perturbations of the weights on the outputs. Denker and LeCun [7], and MacKay [43] investigated the posterior probability distributions of neural networks and treated the search in the model space (the space of architectures, weight decay, regularizers, etc..) as an inference problem and tried to solve it using Laplace approximations. As a response to the limitations of Laplace approximations, Neal [49] investigated the use of hybrid Monte Carlo for training neural networks, although it has so far been difficult to apply these to the large sizes of neural networks built in modern applications. Also, these approaches lacked scalability in terms of both the network architecture and the data size.

More recently, Graves [19] derived a variational inference scheme for neural networks and Blundell et al. [4] extended this with an update for the variance that is unbiased and simpler to compute. Graves [20] derives a similar algorithm in the case of a mixture posterior probability distribution. A more scalable solution based on expectation propagation was proposed by Soudry [57] in 2014. While this method works for networks with binary weights, its extension to continuous weights is unsatisfying as it does not produce estimates of posterior variance.

Several authors have claimed that Dropout [58] and Gaussian Dropout [61] can be viewed as approximate variational inference schemes [13, 32]. We compare our results to Gal's & Ghahramani's [13] and discuss the methodological differences in detail.

3.1.2 Uncertainty Estimation

Neural Networks can predict uncertainty when Bayesian methods are introduced in it. An attempt to model uncertainty has been studied from 1990s [49] but has not been applied successfully until 2015. Gal and Ghahramani [14] in 2015 provided a theoretical framework for modelling Bayesian uncertainty. Gal and Ghahramani [13] obtained the uncertainty estimates by casting dropout training in conventional deep networks as a Bayesian approximation of a Gaussian Process. They showed that any network trained with dropout is an approximate Bayesian model, and uncertainty estimates can be obtained by computing the variance on multiple predictions with different dropout masks.

3.1.3 Model Pruning

Some early work in the model pruning domain used a second-order Taylor approximation of the increase in the loss function of the network when weight is set to zero [41]. A diagonal Hessian approximation was used to calculate the saliency for each parameter [41] and the low-saliency parameters were pruned from the network and the network was retrained.

Narang [48] showed that a pruned RNN and GRU model performed better for the task of speech recognition compared to a dense network of the original size. This result is very similar to the results obtained in our case where a pruned model achieved better results than a normal network. However, no comparisons can be drawn as the model architecture (CNN vs RNN) used and the tasks (Computer Vision vs Speech Recognition) are completely different. Narang [48] in his work introduced a gradual pruning scheme based on pruning all the weights in a layer less than some threshold (manually chosen) which is linear with some slope in phase 1 and linear with some slope in phase 2 followed by normal training. However, we reduced the number of filters to half for one case and in the other case, we induced a sparsity-based on L1 regularization to remove the less contributing weights and reduced the parameters.

Other similar work [2, 38, 6] to our work that reduces or removed the redundant connections or induces sparsity are motivated by the desire to speed up computation. The techniques used are highly convolutional layer dependent and is not applicable to other architectures like Recurrent Neural Networks. One another interesting method of pruning is to represent each parameter with a smaller floating point number like 16-bits instead of 64 bits. This way there is a speed up in the training and inference time and the model is less computationally expensive.

Another viewpoint for model compression was presented by Gong [17]. He proposed vector quantization to achieve different compression ratios and different accuracies and depending on the use case, the compression and accuracies can be chosen. However, it requires a different hardware architecture to support the inference at runtime. Besides quantization, other potentially complementary approaches to reducing model size include low-rank matrix factorization [8, 38] and group sparsity regularization to arrive at an optimal layer size [1].

Chapter 4

Concept

Chapter Overview

- Bayesian CNNs with Variational Inference based on Bayes by Backprop.
- Bayesian convolutional operations with mean and variance
- Local reparameterization trick for Bayesian CNNs.
- Uncertainty estimation in a Bayesian network.
- Using L1 norm for reducing the number of parameters in a Bayesian network.

4.1 Bayesian Convolutional Neural Networks with Variational Inference

In this section, we explain our algorithm of building a CNN with probability distributions over its weights in each filter, as seen in Figure 4.3, and apply variational inference, i.e. *Bayes by Backprop*, to compute the intractable true posterior probability distribution, as described in the last Chapter. Notably, a fully Bayesian perspective on a CNN is for most CNN architectures not accomplished by merely placing probability distributions over weights in convolutional layers; it also requires probability distributions over weights in fully-connected layers (see Figure 4.2).

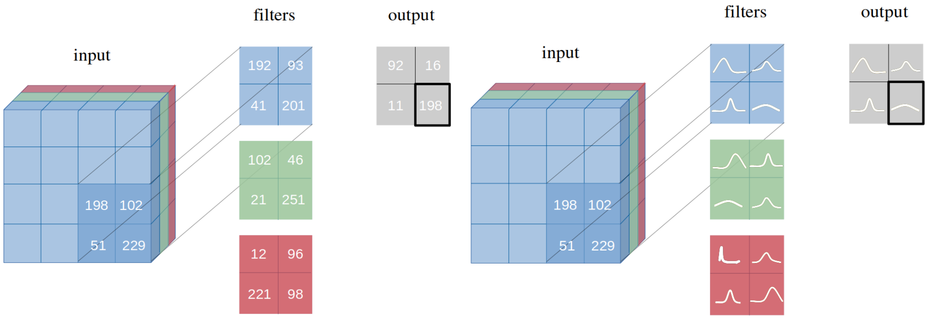

Fig. 4.1 Input image with exemplary pixel values, filters, and corresponding output with point-estimates (top) and probability distributions (bottom) over weights.[55]

4.1.1 Local Reparameterization Trick for Convolutional Layers

We utilise the local reparameterization trick [32] and apply it to CNNs. Following [32, 51], we do not sample the weights w, but we sample instead layer activations b due to its consequent computational acceleration. The variational posterior probability distribution $q_\theta(w_{ijhw}|\mathcal{D}) = \mathcal{N}(\mu_{ijhw}, \alpha_{ijhw}\mu^2_{ijhw})$ (where i and j are the input, respectively output layers, h and w the height, respectively width of any given filter) allows to implement the local reparamerization trick in convolutional layers. This results in the subsequent equation for convolutional layer activations b:

$$b_j = A_i * \mu_i + \epsilon_j \odot \sqrt{A_i^2 * (\alpha_i \odot \mu_i^2)} \tag{4.1}$$

where $\epsilon_j \sim \mathcal{N}(0,1)$, A_i is the receptive field, $*$ signalises the convolutional operation, and $\odot$ the component-wise multiplication.

4.1.2 Applying two Sequential Convolutional Operations (Mean and Variance)

The crux of equipping a CNN with probability distributions over weights instead of single point-estimates and being able to update the variational posterior probability distribution $q_\theta(w|\mathscr{D})$ by backpropagation lies in applying *two* convolutional operations whereas filters with single point-estimates apply *one*. As explained in the last chapter, we deploy the local reparametrization trick and sample from the output b. Since the output b is a function of mean μ_{ijwh} and variance $\alpha_{ijhw}\mu^2_{ijhw}$ among others, we are then able to compute the two variables determining a Gaussian probability distribution, namely mean μ_{ijhw} and variance $\alpha_{ijhw}\mu^2_{ijhw}$, separately.

We do this in two convolutional operations: in the first, we treat the output b as an output of a CNN updated by frequentist inference. We optimize with Adam [31] towards a single point-estimate which makes the validation accuracy of classifications increasing. We interpret this single point-estimate as the mean μ_{ijwh} of the variational posterior

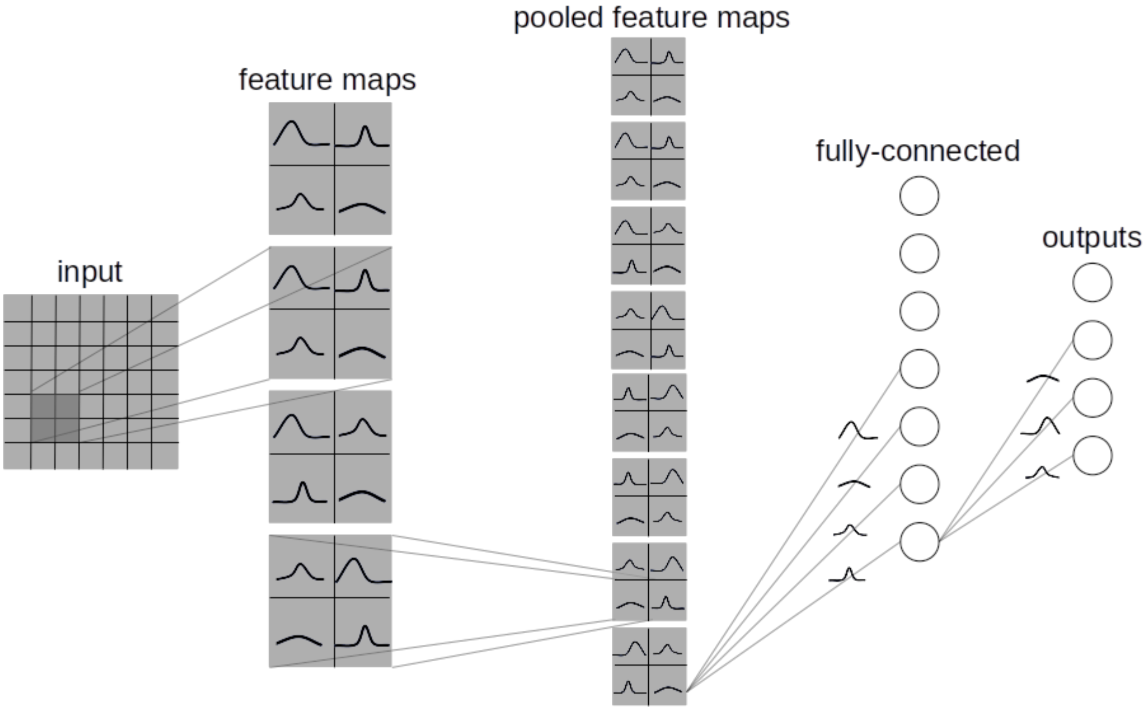

Fig. 4.2 Fully Bayesian perspective of an exemplary CNN. Weights in filters of convolutional layers, and weights in fully-connected layers have the form of a probability distribution. [55]

probability distributions $q_\theta(w|\mathcal{D})$. In the second convolutional operation, we learn the variance $\alpha_{ijhw}\mu^2_{ijhw}$. As this formulation of the variance includes the mean μ_{ijwh}, only α_{ijhw} needs to be learned in the second convolutional operation [47]. In this way, we ensure that only one parameter is updated per convolutional operation, exactly how it would have been with a CNN updated by frequentist inference.

In other words, while we learn in the first convolutional operation the MAP of the variational posterior probability distribution $q_\theta(w|\mathcal{D})$, we observe in the second convolutional operation how much values for weights w deviate from this MAP. This procedure is repeated in the fully-connected layers. In addition, to accelerate computation, to ensure a positive non-zero variance $\alpha_{ijhw}\mu^2_{ijhw}$, and to enhance accuracy, we learn $\log\alpha_{ijhw}$ and use the *Softplus* activation function as further described in the Experiments section.

4.2 Uncertainty Estimation in CNN

In classification tasks, we are interested in the predictive distribution $p_\mathcal{D}(y^*|x^*)$, where x^* is an unseen data example and y^* its predicted class. For a Bayesian neural network, this quantity is given by:

$$p_\mathcal{D}(y^*|x^*) = \int p_w(y^*|x^*)\ p_\mathcal{D}(w)\ dw \tag{4.2}$$

In *Bayes by Backprop*, Gaussian distributions $q_\theta(w|\mathcal{D}) \sim \mathcal{N}(w|\mu,\sigma^2)$, where $\theta = \{\mu,\sigma\}$ are learned with some dataset $\mathcal{D} = \{x_i, y_i\}_{i=1}^n$ as we explained previously. Due to the discrete and finite nature of most classification tasks, the predictive distribution is commonly assumed to be a categorical. Incorporating this aspect into the predictive distribution gives us

$$p_\mathcal{D}(y^*|x^*) = \int \mathrm{Cat}(y^*|f_w(x^*))\mathcal{N}(w|\mu,\sigma^2)\ dw \tag{4.3}$$

$$= \int \prod_{c=1}^{C} f(x_c^*|w)^{y_c^*}\ \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(w-\mu)^2}{2\sigma^2}}\ dw \tag{4.4}$$

where C is the total number of classes and $\sum_c f(x_c^*|w) = 1$.

As there is no closed-form solution due to the lack of conjugacy between categorical and Gaussian distributions, we cannot recover this distribution. However, we can construct an unbiased estimator of the expectation by sampling from $q_\theta(w|\mathscr{D})$:

$$\begin{aligned}\mathbb{E}_q[p_{\mathscr{D}}(y^*|x^*)] &= \int q_\theta(w|\mathscr{D})\; p_w(y|x)\; dw &(4.5)\\ &\approx \frac{1}{T}\sum_{t=1}^{T} p_{w_t}(y^*|x^*) &(4.6)\end{aligned}$$

where T is the pre-defined number of samples. This estimator allows us to evaluate the uncertainty of our predictions by the definition of variance, hence called *predictive variance* and denoted as Var_q:

$$\mathrm{Var}_q\big(p(y^*|x^*)\big) = \mathbb{E}_q[yy^T] - \mathbb{E}_q[y]\mathbb{E}_q[y]^T \tag{4.7}$$

This quantity can be decomposed into the aleatoric and epistemic uncertainty [30, 36].

$$\begin{aligned}\mathrm{Var}_q\big(p(y^*|x^*)\big) &= \underbrace{\frac{1}{T}\sum_{t=1}^{T} \mathrm{diag}(\hat{p}_t) - \hat{p}_t\; \hat{p}_t^T}_{\text{aleatoric}}\\ &+ \underbrace{\frac{1}{T}\sum_{t=1}^{T} (\hat{p}_t - \bar{p})(\hat{p}_t - \bar{p})^T}_{\text{epistemic}}\end{aligned} \tag{4.8}$$

where $\bar{p} = \frac{1}{T}\sum_{t=1}^{T}\hat{p}_t$ and $\hat{p}_t = \mathrm{Softmax}\big(f_{w_t}(x^*)\big)$.

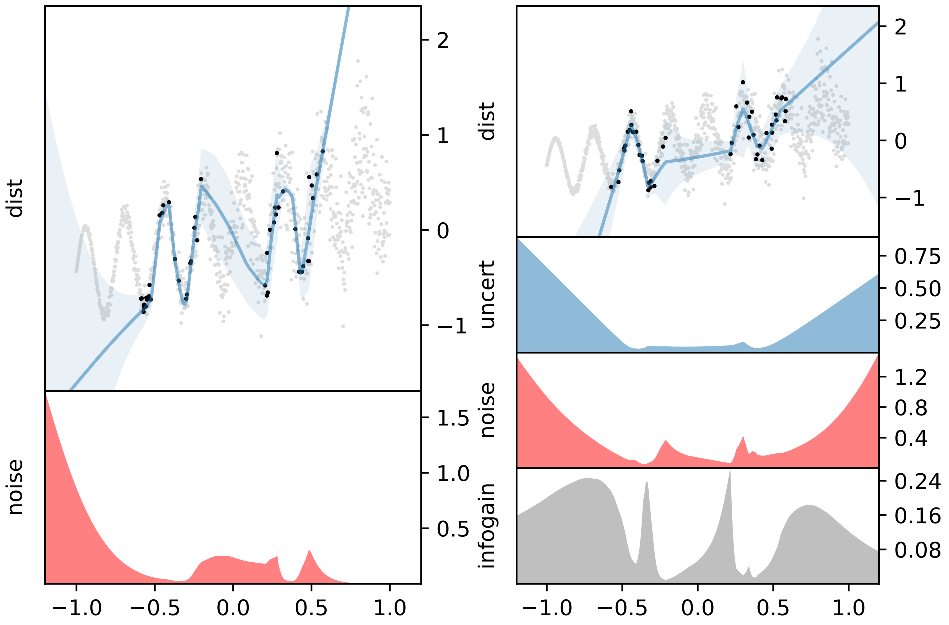

Fig. 4.3 Predictive distributions is estimated for a low-dimensional active learning task. The predictive distributions are visualized as mean and two standard deviations shaded. 🟦 shows the epistemic uncertainty and 🟥 shows the aleatoric noise. Data points are shown in ■. **(Left)** A deterministic network conflates uncertainty as part of the noise and is overconfident outside of the data distribution. **(Right)** A variational Bayesian neural network with standard normal prior represents uncertainty and noise separately but is overconfident outside of the training distribution as defined by [22]

It is of paramount importance that uncertainty is split into aleatoric and epistemic quantities since it allows the modeller to evaluate the room for improvements: while aleatoric uncertainty (also known as statistical uncertainty) is merely a measure for the variation of ("noisy") data, epistemic uncertainty is caused by the model. Hence, a modeller can see whether the quality of the data is low (i.e. high aleatoric uncertainty), or the model itself is the cause for poor performances (i.e. high epistemic uncertainty). The former cannot be improved by gathering more data, whereas the latter can be done so. [9] [30].

4.3 Model Pruning

Model pruning means the reduction in the model weights parameters to reduce the model overall non-zero weights, inference time and computation cost. In our work, a Bayesian Convolutional Network learns two weights, i.e: the mean and the variance compared to point estimate learning one single weight. This makes the overall number of parameters of a Bayesian Network twice as compared to the parameters of a point estimate similar architecture.

To make the Bayesian CNNs parameters equivalent to point estimate architecture, the number of filters in the Bayesian architectures is reduced to half. This makes up for the double learned parameters (mean and variance) against one in point estimates and makes the overall parameters equal for both networks.

Another technique used is the usage of the *L1 normalization* on the learned weights of every layer. By L1 norm, we make the vector of learned weight in various model layers very sparse, as most of its components become close to zero, and at the same time, the remaining non-zero components capture the most important features of the data. We put a threshold and make the weights to be zero if the value falls below the threshold. We only keep the non zero weights and this way the model number of parameters is reduced without affecting the overall performance of the model.

Chapter 5

Empirical Analysis

Chapter Overview

- Applying Bayesian CNNs for the task of Image Recognition on MNIST, CIFAR-10, CIFAR-100 and STL-10 datasets.
- Comparison of results of Bayesian CNNs with Normal CNN architectures on similar datasets.
- Regularization effect of Bayesian Network with dropouts.
- Distribution of mean and variance in Bayesian CNN over time.
- Parameters comparison before and after model pruning.

5.1 Experimentation Methodology

Activation Function

The originally chosen activation functions in all architectures are *ReLU*, but we must introduce another, called *Softplus*, see (5.1), because of our method to apply two convolutional or fully-connected operations. As aforementioned, one of these is determining the mean μ, and the other the variance $\alpha\mu^2$. Specifically, we apply the *Softplus* function because we want to ensure that the variance $\alpha\mu^2$ never becomes zero. This would be equivalent to merely calculating the MAP, which can be interpreted as equivalent to a maximum likelihood estimation (MLE), which is further equivalent to utilising single point-estimates, hence frequentist inference. The *Softplus* activation function is a smooth approximation of *ReLU*. Although it is practically not influential, it has the subtle and analytically important advantage that it never becomes zero for $x \to -\infty$, whereas *ReLU* becomes zero for $x \to -\infty$.

$$\text{Softplus}(x) = \frac{1}{\beta} \cdot \log\big(1 + \exp(\beta \cdot x)\big) \tag{5.1}$$

where β is by default set to 1.
All experiments are performed with the same hyper-parameters settings as stated in the Appendix.

Network Architecture

For all conducted experiments, we implement the foregoing description of Bayesian CNNs with variational inference in LeNet-5 [39] and AlexNet [34]. The exact architecture specifications can be found in the Appendix and in our GitHub repository[1].

Objective Function

To learn the objective function, we use *Bayes by Backprop* [19, 4], which is a variational inference method to learn the posterior distribution on the weights $w \sim q_\theta(w|\mathcal{D})$ of a neural network from which weights w can be sampled in backpropagation. It regularises the weights by minimising a compression cost, known as the variational free energy or the expected lower bound on the marginal likelihood.

We tackled the problem of intractability in Chapter 2 and consequently, we arrive at the tractable cost function (5.2) which is aimed to be optimized, i.e. minimised w.r.t. θ, during training:

---

[1] https://github.com/kumar-shridhar/PyTorch-BayesianCNN

$$\mathscr{F}(\mathscr{D},\theta) \approx \sum_{i=1}^{n} \log q_\theta(w^{(i)}|\mathscr{D}) - \log p(w^{(i)}) - \log p(\mathscr{D}|w^{(i)}) \tag{5.2}$$

where n is the number of draws.

Let's break the Objective Function (5.2) and discuss in more details.

Variational Posterior

The first term in the equation (5.2) is the variational posterior. The variational posterior is taken as Gaussian distribution centred around mean μ and variance as σ^2.

$$q_\theta(w^{(i)}|\mathscr{D}) = \prod_i \mathscr{N}(w_i|\mu,\sigma^2) \tag{5.3}$$

We will take the log and the log posterior is defined as :

$$log(q_\theta(w^{(i)}|\mathscr{D})) = \sum_i log\mathscr{N}(w_i|\mu,\sigma^2) \tag{5.4}$$

Prior

The second term in the equation (5.2) is the prior over the weights and we define the prior over the weights as a product of individual Gaussians :

$$p(w^{(i)}) = \prod_i \mathscr{N}(w_i|0,\sigma_p^2) \tag{5.5}$$

We will take the log and the log prior is defined as:

$$log(p(w^{(i)})) = \sum_i log\mathscr{N}(w_i|0,\sigma_p^2) \tag{5.6}$$

Likelihood

The final term of the equation (5.2) $\log p(\mathscr{D}|w^{(i)})$ is the likelihood term and is computed using the softmax function.

Parameter Initialization

We use a Gaussian distribution and we store mean and variance values instead of just one weight. The way mean μ and variance σ is computed is defined in the previous chapter. Variance cannot be negative and it is ensured by using *softplus* as the activation function.

We express variance σ as $\sigma_i = softplus(\rho_i)$ where ρ is an unconstrained parameter.

We take the Gaussian distribution and initialize mean μ as 0 and variance σ (and hence ρ) randomly. We observed mean centred around 0 and a variance starting with a big number and gradually decreasing over time. A good initialization can also be to put a restriction on variance and initialize it small. However, it might be data dependent and a good method for variance initialization is still to be discovered. We perform gradient descent over $\theta = (\mu, \rho)$, and individual weight $w_i \sim \mathcal{N}(w_i|\mu_i, \sigma_i)$.

Optimizer

For all our tasks, we take Adam optimizer [31] to optimize the parameters. We also perform the local reparameterization trick as mentioned in the previous section and take the gradient of the combined loss function with respect to the variational parameters (μ, ρ).

Model Pruning

We take the weights of all the layers of the network, apply an L1 norm over it and for all the weights value as zero or below a defined threshold are removed and the model is pruned. Also, since the Bayesian CNNs has twice the number of parameters (μ, σ) compared to a frequentist network (only 1 weight), we reduce the size of our network to half (AlexNet and LeNet- 5) by reducing the number of filters to half. The architecture used is mentioned in the Appendix.

Please note that it can be argued that reducing the number of filters to be half is a method for pruning or not. It can be seen as a method that reduces the number of overall parameters and hence can be thought of a pruning method in some sense. However, it is a subject to argument.

5.2 Case Study 1: Small Datasets (MNIST, CIFAR-10)

We train the networks with the MNIST dataset of handwritten digits [39], and CIFAR-10 dataset [33] since these datasets serve widely as benchmarks for CNNs' performances.

5.2.1 Datasets

MNIST

The MNIST database [40] of handwritten digits have a training set of 60,000 examples and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centred in a fixed-size image of 28 by 28 pixels. Each image is grayscaled and is labelled with its corresponding class that ranges from zero to nine.

CIFAR-10

The CIFAR-10 are labelled subsets of the 80 million tiny images dataset [60]. The CIFAR-10 dataset has a training dataset of 50,000 colour images in 10 classes, with 5,000 training images per class, each image 32 by 32 pixels large. There are 10000 images for testing.

5.2.2 Results

First, we evaluate the performance of our proposed method, Bayesian CNNs with variational inference. Table 5.1 shows a comparison of validation accuracies (in percentage) for architectures trained by two disparate Bayesian approaches, namely variational inference, i.e. *Bayes by Backprop* and Dropout as proposed by Gal and Ghahramani [13].

We compare the results of these two approaches to frequentist inference approach for both the datasets. Bayesian CNNs trained by variational inference achieve validation accuracies comparable to their counter-architectures trained by frequentist inference. On MNIST, validation accuracies of the two disparate Bayesian approaches are comparable, but a Bayesian LeNet-5 with Dropout achieves a considerable higher validation accuracy on CIFAR-10, although we were not able to reproduce these reported results.

	MNIST	CIFAR-10
Bayesian AlexNet (with VI)	99	73
Frequentist AlexNet	99	73
Bayesian LeNet-5 (with VI)	98	69
Frequentist LeNet-5	98	68
Bayesian LeNet-5 (with Dropout)	99.5	83

Table 5.1 Comparison of validation accuracies (in percentage) for different architectures with variational inference (VI), frequentist inference and Dropout as a Bayesian approximation as proposed by Gal and Ghahramani [13] for MNIST, and CIFAR-10.

Figure 5.1 shows the validation accuracies of Bayesian vs Non-Bayesian CNNs. One thing to observe is that in initial epochs, Bayesian CNNs trained by variational inference start with a low validation accuracy compared to architectures trained by frequentist inference. This must deduce from the initialization of the variational posterior probability distributions $q_\theta(w|\mathscr{D})$ as uniform distributions, while initial point-estimates in architectures trained by frequentist inference are randomly drawn from a standard Gaussian distribution. (For uniformity, we changed the initialization of frequentist architectures from Xavier initialization to standard Gaussian). The latter initialization method ensures the initialized weights are neither too small nor too large. In other words, the motivation of the latter initialization is to start with weights such that the activation functions do not let them begin in saturated or dead regions. This is not true in case of uniform distributions and hence, Bayesian CNNs' starting validation accuracies can be comparably low.

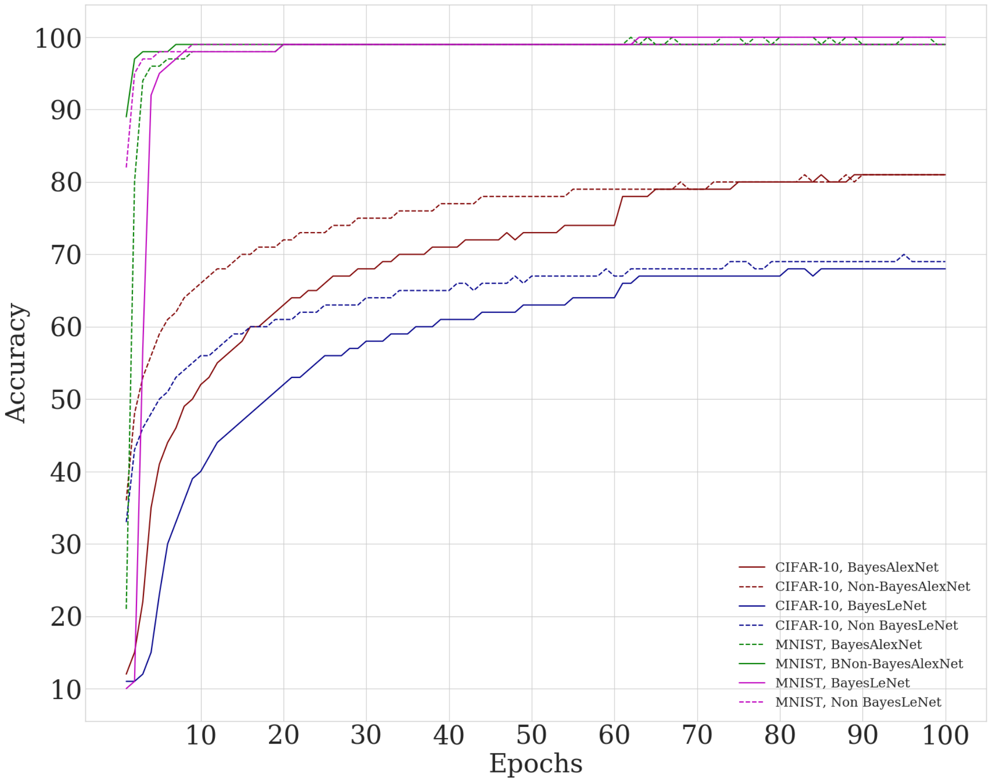

Fig. 5.1 Comparison of Validation Accuracies of Bayesian AlexNet and LeNet-5 with frequentist approach on MNIST and CIFAR-10 datasets

Figure 5.2 displays the convergence of the standard deviation σ of the variational posterior probability distribution $q_\theta(w|\mathcal{D})$ of a random model parameter over epochs. As aforementioned, all prior probability distributions $p(w)$ are initialized as uniform distributions. The variational posterior probability distributions $q_\theta(w|\mathcal{D})$ are approximated as Gaussian distributions which become more confident as more data is processed - observable by the decreasing standard deviation over epochs in Figure 5.2. Although the validation accuracy for MNIST on Bayesian LeNet-5 has already reached 99%, we can still see a fairly steep decrease in the parameter's standard deviation. In Figure 5.3, we plot the actual Gaussian variational posterior probability distributions $q_\theta(w|\mathcal{D})$ of a random parameter of LeNet-5 trained on CIFAR-10 at some epochs.

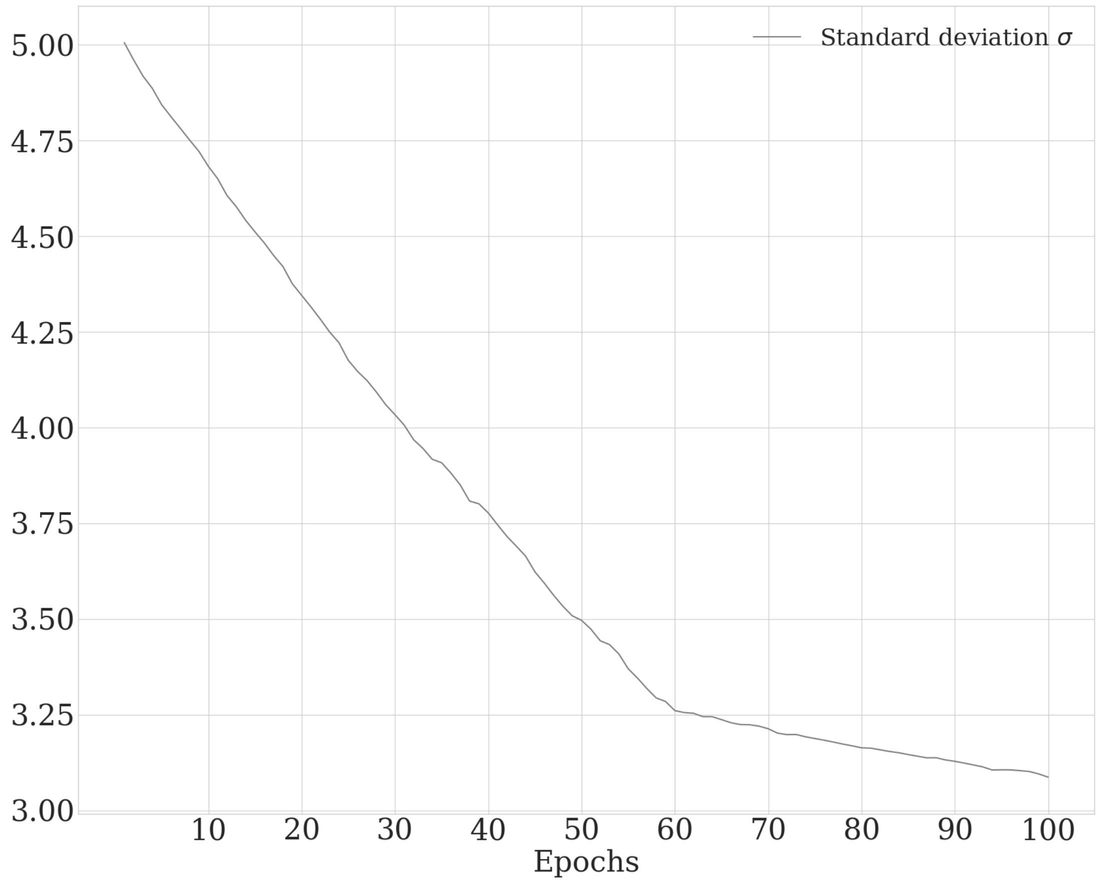

Fig. 5.2 Convergence of the standard deviation of the Gaussian variational posterior probability distribution $q_\theta(w|\mathscr{D})$ of a random model parameter at epochs 1, 5, 20, 50, and 100. MNIST is trained on Bayesian LeNet-5.

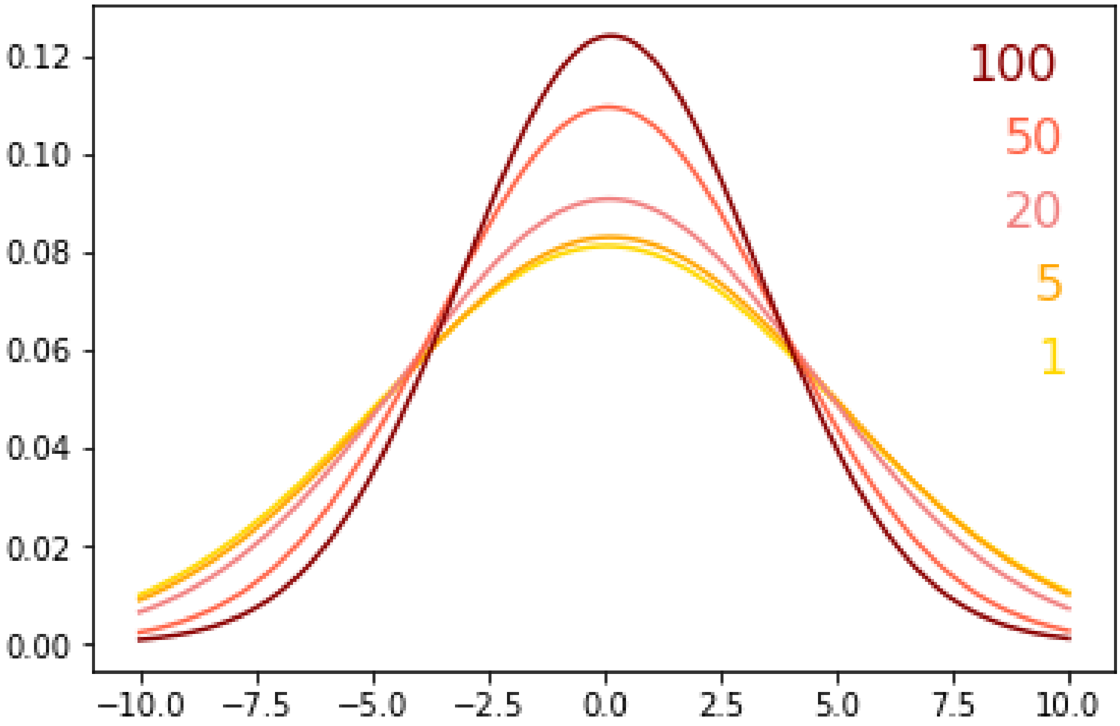

Fig. 5.3 Convergence of the Gaussian variational posterior probability distribution $q_\theta(w|\mathcal{D})$ of a random model parameter at epochs 1, 5, 20, 50, and 100. CIFAR-10 is trained on Bayesian LeNet-5.

Figure 5.3 displays the convergence of the Gaussian variational probability distribution of a weight taken randomly from the first layer of LeNet-5 architecture. The architecture is trained on CIFAR-10 dataset with uniform initialization.

5.3 Case Study 2: Large Dataset (CIFAR-100)

5.3.1 Dataset

CIFAR-100

This dataset is similar to the CIFAR-10 and is a labelled subset of the 80 million tiny images dataset [60]. The dataset has 100 classes containing 600 images each. There are 500 training images and 100 validation images per class. The images are coloured with a resolution of 32 by 32 pixels.

5.3.2 Results

	CIFAR-100
Bayesian AlexNet (with VI)	36
Frequentist AlexNet	38
Bayesian LeNet-5 (with VI)	31
Frequentist LeNet-5	33

Table 5.2 Comparison of validation accuracies (in percentage) for different architectures with variational inference (VI), frequentist inference and Dropout as a Bayesian approximation as proposed by Gal and Ghahramani [13] for MNIST, CIFAR-10, and CIFAR-100.

In Figure 5.4, we show how Bayesian networks incorporate naturally effects of regularization, exemplified on AlexNet. While an AlexNet trained by frequentist inference without any regularization overfits greatly on CIFAR-100, an AlexNet trained by Bayesian inference on CIFAR-100 does not. This is evident from the high value of training accuracy for frequentist approach with no dropout or 1 layer dropout. Bayesian CNN performs equivalently to an AlexNet trained by frequentist inference with three layers of Dropout after the first, fourth, and sixth layers in the architecture. Another thing to note here is that the Bayesian CNN with 100 samples overfits slightly lesser compared to Bayesian CNN with 25 samples. However, a higher sampling number on a smaller dataset didn't prove useful and we stuck with 25 as the number of samples for all other experiments.

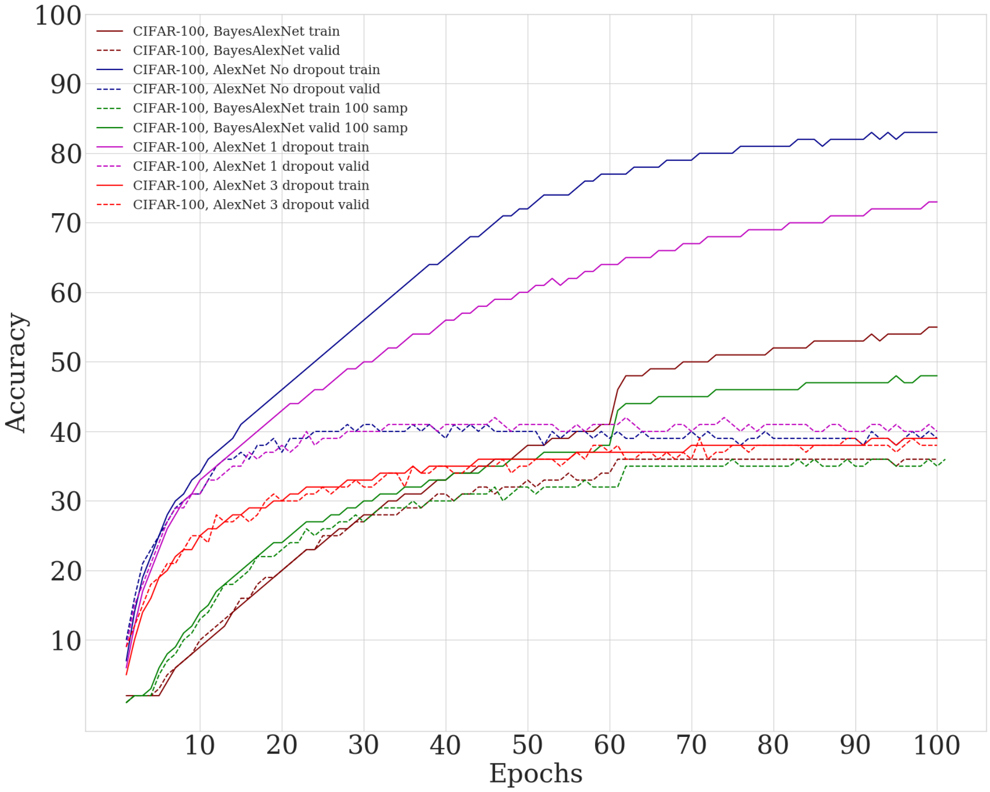

Fig. 5.4 Comparison of Training and Validation Accuracies of Bayesian AlexNet and LeNet-5 with frequentist approach with and without dropouts on CIFAR-100 datasets

Table 5.3 shows a comparison of the training and validation accuracies for AlexNet with Bayesian approach and frequentist approach. The low gap between the training and validation accuracies shows the robustness of Bayesian approach towards overfitting and shows how Bayesian approach without being regularized overfits lesser as compared to frequentist architecture with no or one dropout layer. The results are comparable with AlexNet architecture with 3 dropout layers.

	Training Accuracy	Validation Accuracy
Frequentist AlexNet (No dropout)	83	38
Frequentist AlexNet (1 dropout layer)	72	40
Frequentist AlexNet (3 dropout layer)	39	38
Bayesian AlexNet (25 num of samples)	54	37
Bayesian AlexNet (100 num of samples)	48	37

Table 5.3 Comparison of training and validation accuracies (in percentage) for AlexNet architecture with variational inference (VI) and frequentist inference for CIFAR-100.

5.4 Uncertainity Estimation

Finally, Table 5.4 compares the means of aleatoric and epistemic uncertainties for a Bayesian LeNet-5 with variational inference on MNIST and CIFAR-10. The aleatoric uncertainty of CIFAR-10 is about twenty times as large as that of MNIST. Considering that the aleatoric uncertainty measures the irreducible variability and depends on the predicted values, a larger aleatoric uncertainty for CIFAR-10 can be directly deduced from its lower validation accuracy and may be further due to the smaller number of training examples. The epistemic uncertainty of CIFAR-10 is about fifteen times larger than that of MNIST, which we anticipated since epistemic uncertainty decreases proportionally to validation accuracy.

	Aleatoric uncertainty	Epistemic uncertainty
Bayesian LeNet-5 (MNIST)	0.0096	0.0026
Bayesian LeNet-5 (CIFAR-10)	0.1920	0.0404

Table 5.4 Aleatoric and epistemic uncertainty for Bayesian LeNet-5 calculated for MNIST and CIFAR-10, computed as proposed by Kwon et al. [36].

5.5 Model Pruning

Halving the Number of Filters

For every parameter for a frequentist inference network, Bayesian CNNs has two parameters (μ, σ). Halving the number of parameters of Bayesian AlexNet ensures the number of parameters of it is comparable with a frequentist inference network. The number of filters of ALexNet is halved and a new architecture called AlexNetHalf is defined in Figure 5.4.

layer type	width	stride	padding	input shape	nonlinearity
convolution (11 × 11)	32	4	5	$M \times 3 \times 32 \times 32$	Softplus
max-pooling (2 × 2)		2	0	$M \times 32 \times 32 \times 32$	
convolution (5 × 5)	96	1	2	$M \times 32 \times 15 \times 15$	Softplus
max-pooling (2 × 2)		2	0	$M \times 96 \times 15 \times 15$	
convolution (3 × 3)	192	1	1	$M \times 96 \times 7 \times 7$	Softplus
convolution (3 × 3)	128	1	1	$M \times 192 \times 7 \times 7$	Softplus
convolution (3 × 3)	64	1	1	$M \times 128 \times 7 \times 7$	Softplus
max-pooling (2 × 2)		2	0	$M \times 64 \times 7 \times 7$	
fully-connected	64			$M \times 64$	

Table 5.5 AlexNetHalf with number of filters halved compared to the original architecture.

The AlexNetHalf architecture was trained and validated on the MNIST, CIFAR10 and CIFAR100 dataset and the results are shown in Table 5.6. The accuracy of pruned AlexNet with only half the number of filters compared to the normal architecture shows an accuracy gain of 6 per cent in case of CIFAR10 and equivalent performance for MNIST and CIFAR100 datasets. A lesser number of filters learn the most important features which proved better at inter-class classification could be one of the explanations for the rise in accuracy. However, upon visualization of the filters, no distinct clarification can be made to prove the previous statement.

Another possible explanation could be the model is generalizing better after the reduction in the number of filters ensuring the model is not overfitting and validation accuracy is comparatively higher. CIFAR-100 higher validation accuracy on ALexNetHalf and a lower training accuracy than Bayesian AlexNet proves the theory. Using a lesser number of filters further enhances the regularization effect and makes the model more robust against overfitting. Similar results have been achieved by Narang [48] in his work where a pruned model achieved better accuracy compared to the original architecture in a speech recognition task. Suppressing or removing the weights that have lesser or no contribution to the prediction makes the model rely its prediction on the most prominent and unique features and hence improves the prediction accuracy.

	MNIST	CIFAR-10	CIFAR-100
Bayesian AlexNet (with VI)	99	73	36
Frequentist AlexNet	99	73	38
Bayesian AlexNetHalf (with VI)	99	79	38

Table 5.6 Comparison of validation accuracies (in percentage) for AlexNet with variational inference (VI), AlexNet with frequentist inference and AlexNet with half number of filters halved for MNIST, CIFAR-10 and CIFAR-100 datasets.

Applying L1 Norm

L1 norm induces sparsity in the trained model parameters and sets some values to zero. We trained a model to some epochs (number of epochs differs across datasets as we applied early stopping when validation accuracy remains unchanged for 5 epochs). We removed the zero-valued parameters of the learned weights and keep the non-zero parameters for a trained Bayesian AlexNet on MNIST and CIFAR-10 datasets. We pruned the model to make the number of parameters in a Bayesian Network comparable to the number of parameters in the point-estimate architecture.

Table 5.7 shows the comparison of validation accuracies of the applied L1 Norm AlexNet Bayesian architecture with Bayesian AlexNet architecture and with AlexNet frequentist architecture. We got comparable results on MNIST and CIFAR10 with the experiments and the results are shown in Table 5.7

	MNIST	CIFAR-10
Bayesian AlexNet (with VI)	99	73
Frequentist AlexNet	99	73
Bayesian AlexNet with L1 Norm (with VI)	99	71

Table 5.7 Comparison of validation accuracies (in percentage) for AlexNet with variational inference (VI), AlexNet with frequentist inference and BayesianAlexNet with L1 norm applied for MNIST and CIFAR-10 datasets.

One thing to note here is that the numbers of parameters of Bayesian Network after applying L1 norm is not necessarily equal to the number of parameters in the frequentist AlexNet architecture. It depends on the data size and the number of classes. However, the number of parameters in the case of MNIST and CIFAR-10 are pretty comparable and there is not much reduction in the accuracy either. Also, the early stopping was applied when there is no change in the validation accuracy for 5 epochs and the model was saved and later pruned with the application of L1 norm.

5.6 Training Time

Training time of a Bayesian CNNs is twice of a frequentist network with similar architecture when the number of samples is equal to one. In general, the training time of a Bayesian CNNs, T is defined as:

$$T = 2 * number\_of\_samples * t \tag{5.7}$$

where t is the training time of a frequentist network. The factor of 2 is present due to the double learnable parameters in a Bayesian CNN network i.e. mean and variance for every single point estimate weight in the frequentist network.

However, there is no difference in the inference time for both the networks.

Chapter 6

Applications

Chapter Overview

- Empirical analysis of BayesCNN with normal architecture for Image Super Resolution.
- Empirical analysis of BayesCNN with normal architecture for Generative Adversarial Networks.

6.1 BayesCNN for Image Super Resolution

The task referred to as Super Resolution (SR) is the recovery of a High-Resolution (HR) image from a given Low-Resolution (LR) image. It is applicable to many areas like medical imaging Shi et al. [54], face recognition Gunturk et al. [21] and so on.

There are many ways to do a single image super-resolution and detailed benchmarks of the methods are provided by Yang Yang et al. [62]. Following are the major ways to do a single image super-resolution:

Prediction Models: These models generate High-Resolution images from Low-Resolution inputs through a predefined mathematical formula. No training data is needed for such models. Interpolation-based methods (bilinear, bicubic, and Lanczos) generate HR pixel intensities by weighted averaging neighbouring LR pixel values are good examples of this method.

Edge Based Methods: Edges are one of the most important features for any computer vision task. The High-Resolution images learned from the edge features high-quality edges and good sharpness. However, these models lack good colour and texture information.

Patch Based Methods: Cropped patches from Low-Resolution images and High-Resolution images are taken from training dataset to learn some mapping function. The overlapped patches are averaged or some other techniques like Conditional Random Fields [37] can be used for better mapping of the patches.

6.1.1 Our Approach

We build our work upon Shi et al. [53] work that shows that performing Super Resolution work in High-Resolution space is not the optimal solution and it adds the computation complexity. We used a Bayesian Convolutional Neural Network to extract features in the Low-Resolution space. We use an efficient sub-pixel convolution layer, as proposed by Shi et al. [53], which learns an array of upscaling filters to upscale the final Low-Resolution feature maps into the High-Resolution output. This replaces the handcrafted bicubic filter in the Super Resolution pipeline with more complex upscaling filters specifically trained for each feature map, and also reduces the computational complexity of the overall Super Resolution operation.

The hyperparameters used in the experiments are mentioned in the Appendix A section in details.

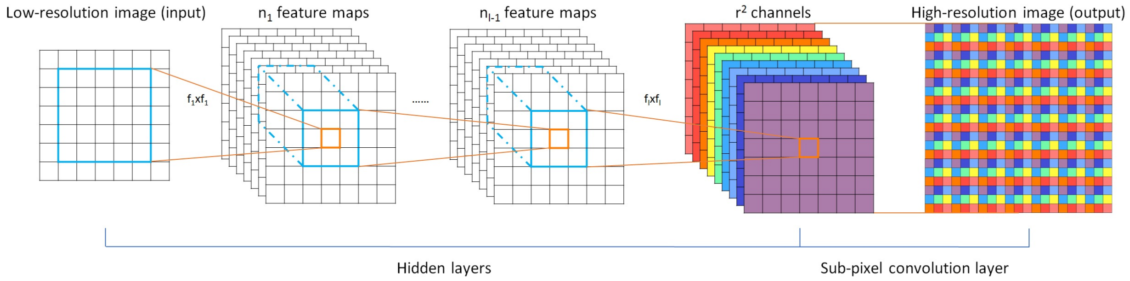
Low-resolution image (input)
n_1 feature maps
n_{l-1} feature maps
r^2 channels
High-resolution image (output)
$f_1 x f_1$
......
$f_l x f_l$
Hidden layers
Sub-pixel convolution layer

Fig. 6.1 The proposed efficient sub-pixel convolutional neural network (ESPCN) [53], with two convolution layers for feature maps extraction, and a sub-pixel convolution layer that aggregates the feature maps from Low Resolution space and builds the Super Resolution image in a single step.

We used a four-layer convolutional model as mentioned in the paper [53]. We replaced the convolution layer by Bayesian convolution layer and changed the forward pass that now computes the mean, variance and KL divergence. The PixelShuffle layer is kept same as provided by PyTorch and no changes have been made there.

layer type	width	stride	padding
convolution (5 × 5)	64	1	2
convolution (3 × 3)	64	1	1
convolution (3 × 3)	32	1	1
convolution (3 × 3)	upscale factor * * 2	1	1

Table 6.1 Network Architecture for Bayesian Super Resolution

Where *upscale factor* is defined as a parameter. For our experiments, we take upscale factor = 3.

6.1.2 Empirical Analysis

The Network architecture was trained on BSD300 dataset [46] provided by the Berkeley Computer Vision Department. The dataset is very popular for Image Super-Resolution task and thus the dataset is used to compare the results with other work in the domain.

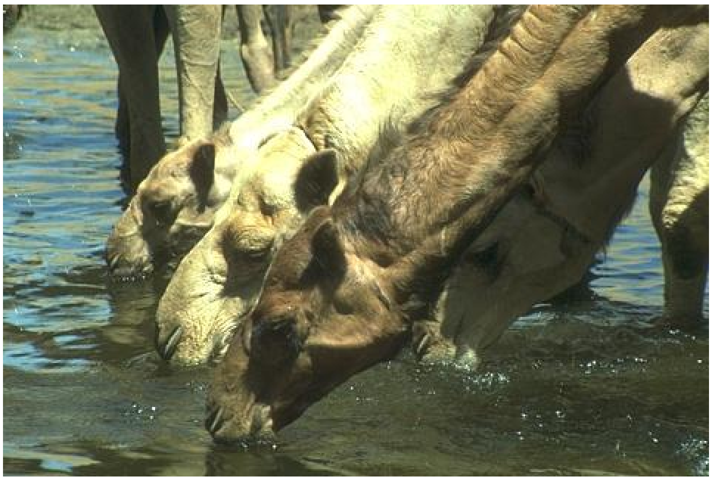

Fig. 6.2 Sample image in Low Resolution image space taken randomly from BSD 300 [46] dataset.

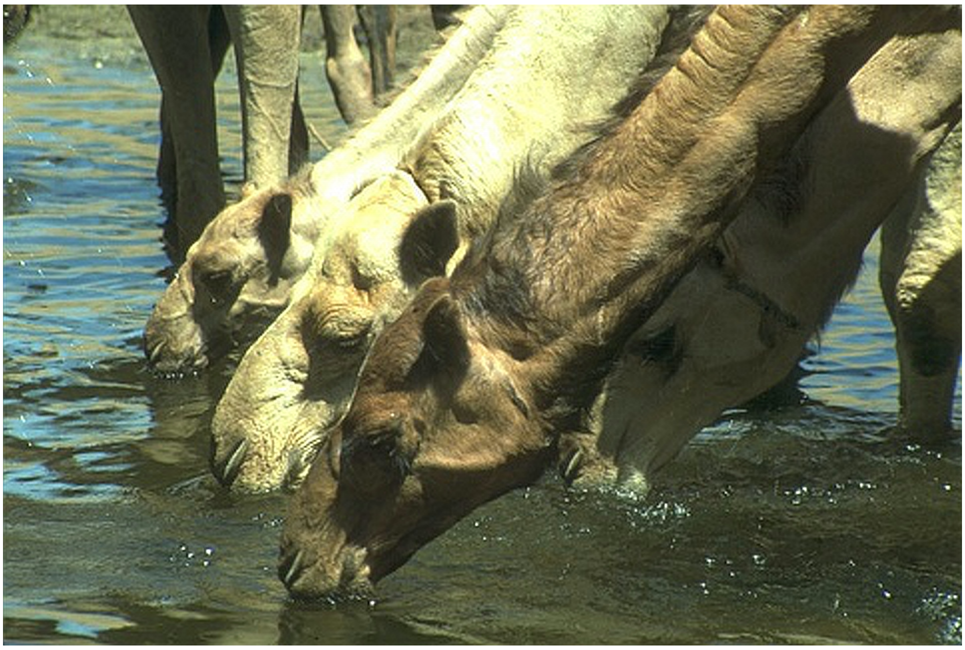

Fig. 6.3 Generated Super Resolution Image scaled to 40 percent to fit

The generated results with Bayesian Network is compared with the original paper and the results are comparable in terms of the number and the quality of the image generated. This application was to prove the concept that the Bayesian Networks can be used for the task of Image Super Resolution. Furthermore, the results are pretty good.

Some more research is needed in the future to achieve state-of-the-art results in this domain which is out of the scope of this thesis work.

6.2 BayesCNN for Generative Adversarial Networks

Generative Adversarial Networks (GANs) [18] can be used for two major tasks: to learn good feature representations by using the generator and discriminator networks as feature extractors and to generate natural images. The learned feature representation or generated images can reduce the number of images substantially for a computer vision supervised task. However, GANs were quite unstable to train in the past and that is why we base our work on the stable GAN architecture namely Deep Convolutional GANs (DCGAN) [52]. We use the trained Bayesian discriminators for image classification tasks, showing competitive performance with the normal DCGAN architecture.

6.2.1 Our Approach

We based our work on the paper: Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks by Radford et al. [52]. We used the architecture of a deep convolutional generative adversarial networks (DCGANs) that learns a hierarchy of representations from object parts to scenes in both the generator and discriminator. The generator used in the Network is shown in Table 6.2. The architecture is kept similar to the architecture used in DCGAN paper [52]. Table 6.3 shows the discriminator network with Bayesian Convolutional Layers.

layer type	width	stride	padding	nonlinearity
ConvolutionTranspose (4 × 4)	ngf * 8	1	0	ReLU
ConvolutionTranspose (4 × 4)	ngf * 4	2	1	ReLU
ConvolutionTranspose (4 × 4)	ngf * 2	2	1	ReLU
ConvolutionTranspose (4 × 4)	ngf	2	1	ReLU
ConvolutionTranspose (4 × 4)	nc	2	1	TanH

Table 6.2 Generator architecture as defined in the paper. [52]

where *ngf* is the number of generator filters which is chosen to be 64 in our work and *nc* is the number of output channels which is set to 3.

layer type	width	stride	padding	nonlinearity
Convolution (4 × 4)	ndf	2	1	LeakyReLU
Convolution(4 × 4)	ndf * 2	2	1	LeakyReLU
Convolution (4 × 4)	ndf * 4	2	1	LeakyReLU
Convolution (4 × 4)	ndf * 8	2	1	leakyReLU
ConvolutionTranspose (4 × 4)	1	1	0	Sigmoid

Table 6.3 Discriminator architecture with Bayesian Convolutional layers

where *ndf* is the number of discriminator filters and is set to 64 as default for all our experiments.

6.2.2 Empirical Analysis

The images were taken directly and no pre-processing was applied to any of the images. Normalization was applied with value 0.5 to make the data mean centred. A batch size of 64 was used along with Adam [31] as an optimizer to speed up the training. All weights were initialized from a zero-centred Normal distribution with standard deviation equal to 1. We also used LeakyReLU as mentioned in the original DCGAN paper [52]. The slope of the leak in LeakyReLU was set to 0.2 in all models. We used the learning rate of 0.0001, whereas in paper 0.0002 was used instead. Additionally, we found leaving the momentum term β_1 at the suggested value of 0.9 resulted in training oscillation and instability while reducing it to 0.5 helped stabilize training (also taken from original paper [52]).

The hyperparameters used in the experiments are mentioned in the Appendix A section in details. The fake results of the generator after 100 epochs of training is shown in Figure 6.4. To compare the results, real samples are shown in Figure 6.5. The loss in case of a Bayesian network is higher as compared to the DCGAN architecture originally described by the authors. However, upon looking at the results, there is no comparison that can be drawn from the results of the two networks. Since GANs are difficult to anticipate just by the loss number, the comparison cannot be made. The results are pretty comparable for the Bayesian models and the original DCGAN architecture.

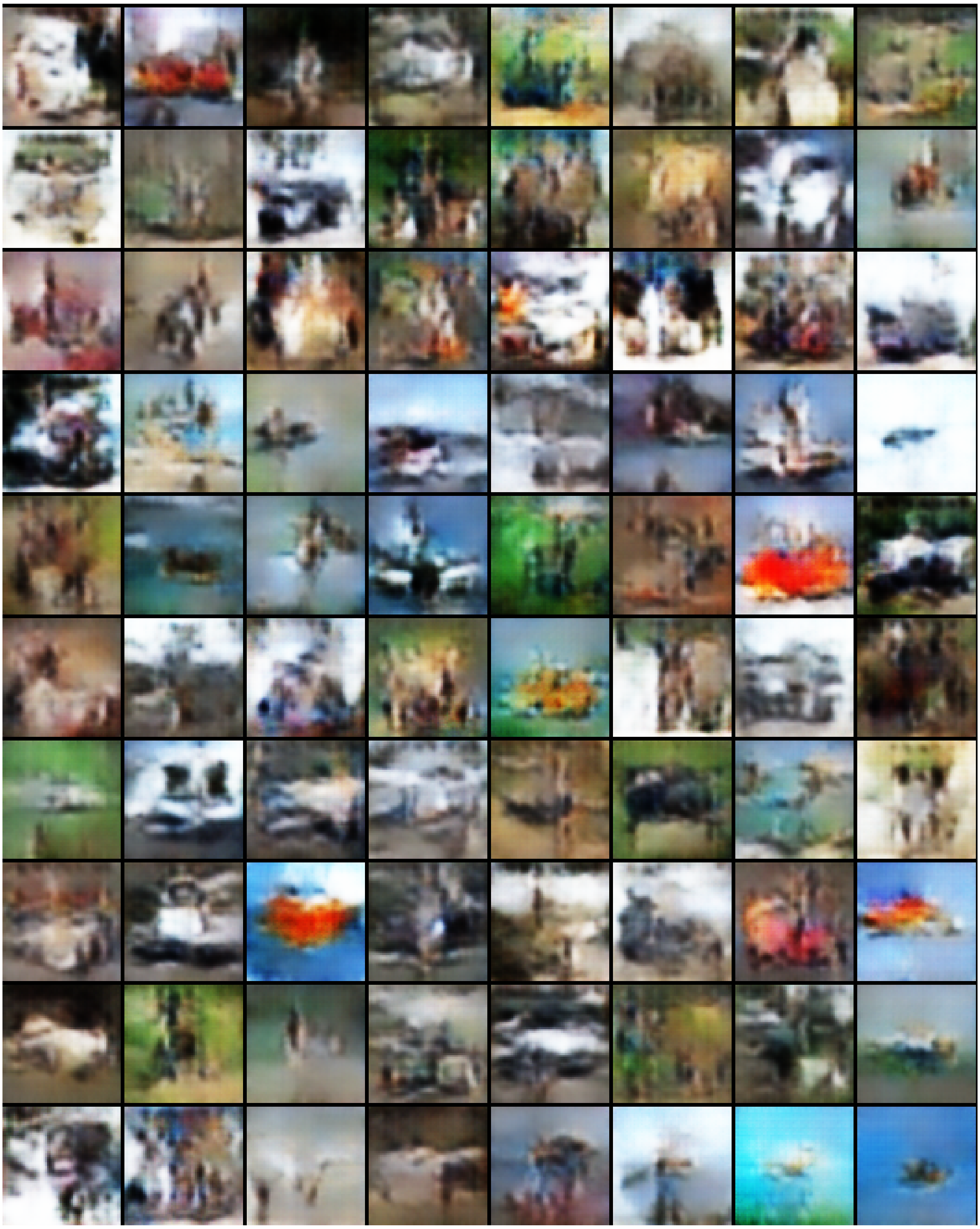

Fig. 6.4 Fake Samples generated from the Bayesian DCGAN model trained on CIFAR10 dataset

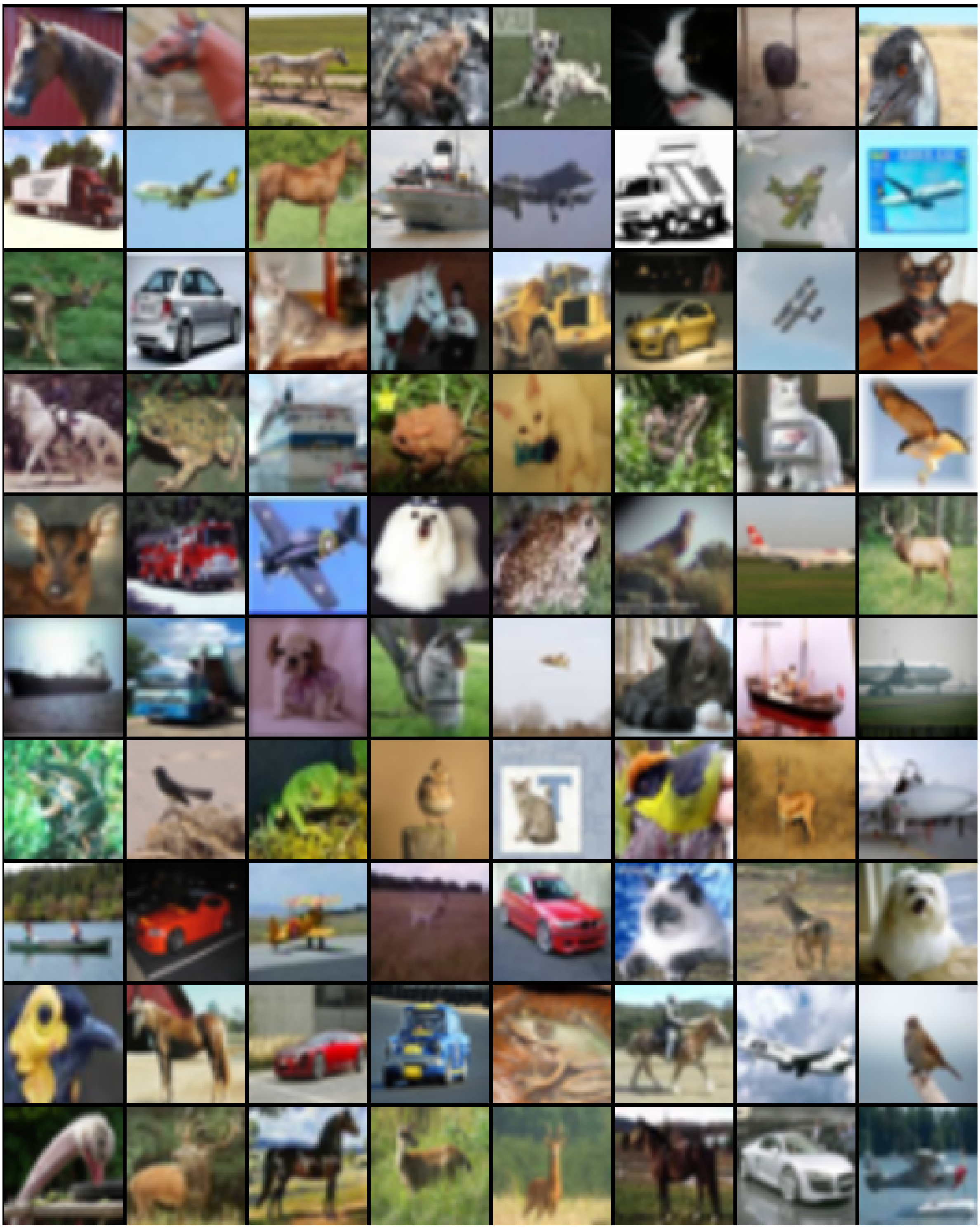

Fig. 6.5 Real Samples taken from CIFAR10 dataset

Chapter 7

Conclusion and Outlook

We propose Bayesian CNNs utilizing *Bayes by Backprop* as a reliable, variational inference method for CNNs which has not been studied to-date, and estimate the models' aleatoric and epistemic uncertainties for prediction. Furthermore, we apply different ways to pruning the Bayesian CNN and compare its results with frequentist architectures.

There has been previous work by Gal and Ghahramani [13] who utilized the various outputs of a Dropout function to define a distribution, and concluded that one can then speak of a Bayesian CNN. This approach finds, perhaps also due to its ease, a large confirming audience. However, we argue against this approach and claim deficiencies. Specifically, in Gal's and Ghahramani's [13] approach, no prior probability distributions $p(w)$ are placed on the CNN's parameters. But, these are a substantial part of a Bayesian interpretation for the simple reason that Bayes' theorem includes them. Thus we argue, starting with prior probability distributions $p(w)$ is essential in Bayesian methods. In comparison, we place prior probability distributions over all model parameters and update them according to Bayes' theorem with variational inference, precisely *Bayes by Backprop*. We show that these neural networks achieve state-of-the-art results as those achieved by the same network architectures trained by frequentist inference.

Furthermore, we examine how uncertainties (both aleatoric and epistemic uncertainties) can be computed for our proposed method and we show how epistemic uncertainties can be reduced upon more training data. We also compare the effect of dropout in a frequentist network to the proposed Bayesian CNN and show the natural regularization effect of Bayesian methods. To counter the twice number of parameters (mean and variance) in a Bayesian CNN compared to a single point estimate weight in a frequentist method, we apply methods of network pruning and show that the Bayesian CNN performs equally good or better even when the network is pruned and the number of parameters is made comparable to a frequentist method.

Finally, we show the applications of Bayesian CNNs in various domains like Image recognition, Image Super-Resolution and Generative Adversarial Networks (GANs). The results are compared with other popular approaches in the field and a comparison of results are drawn. Bayesian CNNs in general, proved to be a good idea to be applied on GANs as prior knowledge for discriminator network helps in better identification of real vs fake images.

As an add-on method to further enhance the stability of the optimization, *posterior sharpening* [11] could be applied to Bayesian CNNs in future work. There, the variational posterior distribution $q_\theta(w|\mathcal{D})$ is conditioned on the training data of a batch $\mathcal{D}^{(i)}$. We can see $q_\theta(w|\mathcal{D}^{(i)})$ as a proposal distribution, or *hyper-prior* when we rethink it as a hierarchical model, to improve the gradient estimates of the intractable likelihood function $p(\mathcal{D}|w)$. For the initialization of the mean and variance, a zero mean and one as standard deviation was used as the normal distribution seems to be the most intuitive distribution to start with. However, with the results drawn in the thesis from several experimentations, a zero-centred mean and very small standard deviation initialization seemed to be performing equally well but training faster. Xavier initialization [15] converges faster in a frequentist network compared to a normal initialization and a similar distribution space needs to be explored with Bayesian networks for initializing the distribution. Other properties like periodicity or spatial invariance are also captured by the priors in data space, and based on these properties an alternative to Gaussian process priors can be found.

Using normal distribution as prior for uncertainty estimation was also explored by Danijar et al. [22] and it was observed that standard normal prior causes the function posterior to generalize in unforeseen ways on inputs outside of the training distribution. Addition of some noise in the normal distribution as prior can help in better uncertainty estimation by the model. However, no such cases were found in our experiments but can be an interesting area to explore in future.

The network is pruned with simple methods like L1 norm and more compression tricks like vector quantization [17] and group sparsity regularization [1] can be applied. In our work, we show that reducing the number of model parameters results in a better generalization of the Bayesian architecture and even leads to improvement in the overall model accuracy on the test dataset. Upon further analysis of the model, there is no concrete learning about the change in the behaviour. A more detailed analysis by visualizing the pattern learned by each neuron and grouping them together and removing the redundant neurons which learns similar behaviour is a good way to prune the model.

The concept of Bayesian CNN is applied to the discriminative network of a GAN in our work and it has shown good initial results. However, the area of Bayesian generative networks in a GAN is still to be investigated.

References

[1] Alvarez, J. M. and Salzmann, M. (2016). Learning the number of neurons in deep networks. In *NIPS*, pages 2262–2270.

[2] Anwar, S., Hwang, K., and Sung, W. (2015). Structured pruning of deep convolutional neural networks. *CoRR*, abs/1512.08571.

[3] Barber, D. and Bishop, C. M. (1998). Ensemble learning in bayesian neural networks. *NATO ASI SERIES F COMPUTER AND SYSTEMS SCIENCES*, 168:215–238.

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

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

[6] Changpinyo, S., Sandler, M., and Zhmoginov, A. (2017). The power of sparsity in convolutional neural networks. *CoRR*, abs/1702.06257.

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

[8] Denton, E. L., Zaremba, W., Bruna, J., LeCun, Y., and Fergus, R. (2014). Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*, pages 1269–1277.

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

[10] E. Rumelhart, D., E. Hinton, G., and J. Williams, R. (1986). Learning representations by back propagating errors. *Nature*, 323:533–536.

[11] Fortunato, M., Blundell, C., and Vinyals, O. (2017). Bayesian recurrent neural networks. *arXiv preprint arXiv:1704.02798*.

[12] Friston, K., Mattout, J., Trujillo-Barreto, N., Ashburner, J., and Penny, W. (2007). Variational free energy and the laplace approximation. *Neuroimage*, 34(1):220–234.

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

[14] Gal, Y. and Ghahramani, Z. (2015b). Dropout as a Bayesian approximation: Insights and applications. In *Deep Learning Workshop, ICML*.

[15] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256.

[16] Gluon MXnet (2017). chapter18_variational-methods-and-uncertainty. https://gluon.mxnet.io/chapter18_variational-methods-and-uncertainty/bayes-by-backprop.html. Online.

[17] Gong, Y., Liu, L., Yang, M., and Bourdev, L. D. (2014). Compressing deep convolutional networks using vector quantization. *CoRR*, abs/1412.6115.

[18] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

[19] Graves, A. (2011). Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, pages 2348–2356.

[20] Graves, A. (2016). Stochastic backpropagation through mixture density distributions. *arXiv preprint arXiv:1607.05690*.

[21] Gunturk, B. K., Batur, A. U., Altunbasak, Y., Hayes, M. H., and Mersereau, R. M. (2003). Eigenface-domain super-resolution for face recognition. *IEEE Transactions on Image Processing*, 12(5):597–606.

[22] Hafner, D., Tran, D., Irpan, A., Lillicrap, T., and Davidson, J. (2018). Reliable uncertainty estimates in deep neural networks using noise contrastive priors. *arXiv preprint arXiv:1807.09289*.

[23] Han, S., Mao, H., and Dally, W. J. (2015). Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149.

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

[25] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

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

[27] Hochreiter, S. and Schmidhuber, J. (1995). Simplifying neural nets by discovering flat minima. In *Advances in neural information processing systems*, pages 529–536.

[28] Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. (2016). Curiosity-driven exploration in deep reinforcement learning via bayesian neural networks. *arXiv preprint arxiv.1605.09674*.

[29] Karparthy, Andrej (2016). Neural Networks 1. http://cs231n.github.io/neural-networks-1/. Online.

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

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

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

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

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

[35] Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86.

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

[37] Lafferty, J., McCallum, A., and Pereira, F. C. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data.

[38] Lebedev, V., Ganin, Y., Rakhuba, M., Oseledets, I. V., and Lempitsky, V. S. (2014). Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *CoRR*, abs/1412.6553.

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

[40] LeCun, Y. and Cortes, C. (2010). MNIST handwritten digit database.

[41] LeCun, Y., Denker, J. S., and Solla, S. A. (1990). Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605.

[42] Lipton, Z. C., Gao, J., Li, L., Li, X., Ahmed, F., and Deng, L. (2016). Efficient exploration for dialogue policy learning with bbq networks & replay buffer spiking. *arXiv preprint arXiv:1608.05081.*

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

[44] MacKay, D. J. (1996). Hyperparameters: optimize, or integrate out? In *Maximum entropy and bayesian methods*, pages 43–59. Springer.

[45] Mackay, D. J. C. (1991). A practical bayesian framework for backprop networks.

[46] Martin, D., Fowlkes, C., Tal, D., and Malik, J. (2001). A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. 8th Int'l Conf. Computer Vision*, volume 2, pages 416–423.

[47] Molchanov, D., Ashukha, A., and Vetrov, D. (2017). Variational dropout sparsifies deep neural networks. *arXiv preprint arXiv:1701.05369*.

[48] Narang, S., Diamos, G. F., Sengupta, S., and Elsen, E. (2017). Exploring sparsity in recurrent neural networks. *CoRR*, abs/1704.05119.

[49] Neal, R. M. (2012). *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media.

[50] Neal, R. M. and Hinton, G. E. (1998). A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pages 355–368. Springer.

[51] Neklyudov, K., Molchanov, D., Ashukha, A., and Vetrov, D. (2018). Variance networks: When expectation does not meet your expectations. *arXiv preprint arXiv:1803.03764*.

[52] Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434.

[53] Shi, W., Caballero, J., Huszár, F., Totz, J., Aitken, A. P., Bishop, R., Rueckert, D., and Wang, Z. (2016). Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. *CoRR*, abs/1609.05158.

[54] Shi, W., Caballero, J., Ledig, C., Zhuang, X., Bai, W., Bhatia, K., de Marvao, A. M. S. M., Dawes, T., O'Regan, D., and Rueckert, D. (2013). Cardiac image super-resolution with global correspondence using multi-atlas patchmatch. In Mori, K., Sakuma, I., Sato, Y., Barillot, C., and Navab, N., editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2013*. Springer Berlin Heidelberg.

[55] Shridhar, K., Laumann, F., Llopart Maurin, A., Olsen, M., and Liwicki, M. (2018). Bayesian convolutional neural networks with variational inference. *arXiv preprint arXiv:1806.05978*.

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

[57] Soudry, D., Hubara, I., and Meir, R. (2014). Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 963–971. Curran Associates, Inc.

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

[59] Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288.

[60] Torralba, A., Fergus, R., and Freeman, W. T. (2008). 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(11).

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

[62] Yang, C.-Y., Ma, C., and Yang, M.-H. (2014). Single-image super-resolution: A benchmark. In *ECCV*.

[63] Yedidia, J. S., Freeman, W. T., and Weiss, Y. (2005). Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on information theory*, 51(7):2282–2312.

Appendix A

Experiment Specifications

Bayesian Settings

A.0.1 Image Recognition

variable	value
learning rate	0.001
epochs	100
batch size	256
sample size	10-25
loss	cross-entropy
$(\alpha\mu^2)_{init}$ of approximate posterior $q_\theta(w\|\mathscr{D})$	-10
optimizer	Adam [31]
λ in ℓ-2 normalisation	0.0005
β_i	$\frac{2^{M-i}}{2^M-1}$ [4]

Sample size can vary from 10 to 25 as this range provided the best results. However, it can be played around with. For most of our experiments, it is either 10 or 25 unless specified otherwise.

A.0.2 Image Super Resolution

variable	value
learning rate	0.01
epochs	200
batch size	64
upscale factor	3
loss	Mean Squared Error
seed	123
$(\alpha\mu^2)_{init}$ of approximate posterior $q_\theta(w\|\mathscr{D})$	-10
optimizer	Adam [31]
λ in ℓ-2 normalisation	0.0005
β_i	$\frac{2^{M-i}}{2^M-1}$ [4]

A.0.3 Generative Adversarial Network

variable	value
learning rate	0.001
epochs	100
batch size	64
image size	64
latent vector (nz)	100
number of generator factor (ndf)	64
number of discriminator factor (ndf)	64
upscale factor	3
loss	Mean Squared Error
number of channels (nc)	3
$(\alpha\mu^2)_{init}$ of approximate posterior $q_\theta(w\|\mathscr{D})$	-10
optimizer	Adam [31]
λ in ℓ-2 normalisation	0.0005
β_i	$\frac{2^{M-i}}{2^M-1}$ [4]

Non Bayesian Settings

A.0.4 Image Recognition

variable	value
learning rate	0.001
epochs	100
batch size	256
loss	cross-entropy
initializer	Xavier [15] or Normal
optimizer	Adam [31]

The weights were initialized with Xavier initialization [15] at first, but to make it consistent with the Bayesian networks where initialization was Normal initialization (mean = 0 and variance = 1), the initializer was changed to Normal initialization.

Architectures

A.0.5 LeNet-5

layer type	width	stride	padding	input shape	nonlinearity
convolution (5×5)	6	1	0	$M \times 1 \times 32 \times 32$	Softplus
Mmax-pooling (2×2)		2	0	$M \times 6 \times 28 \times 28$	
convolution (5×5)	16	1	0	$M \times 1 \times 14 \times 14$	Softplus
max-pooling (2×2)		2	0	$M \times 16 \times 10 \times 10$	
fully-connected	120			$M \times 400$	Softplus
fully-connected	84			$M \times 120$	Softplus
fully-connected	10			$M \times 84$	

Table A.1 LeNet architecture with original configurations as defined in the paper. [39]

A.0.6 AlexNet

layer type	width	stride	padding	input shape	nonlinearity
convolution (11×11)	64	4	5	$M \times 3 \times 32 \times 32$	Softplus
max-pooling (2×2)		2	0	$M \times 64 \times 32 \times 32$	
convolution (5×5)	192	1	2	$M \times 64 \times 15 \times 15$	Softplus
max-pooling (2×2)		2	0	$M \times 192 \times 15 \times 15$	
convolution (3×3)	384	1	1	$M \times 192 \times 7 \times 7$	Softplus
convolution (3×3)	256	1	1	$M \times 384 \times 7 \times 7$	Softplus
convolution (3×3)	128	1	1	$M \times 256 \times 7 \times 7$	Softplus
max-pooling (2×2)		2	0	$M \times 128 \times 7 \times 7$	
fully-connected	128			$M \times 128$	

Table A.2 AlexNet architecture with original configurations as defined in the paper. [34]

A.0.7 AlexNetHalf

layer type	width	stride	padding	input shape	nonlinearity
convolution (11 × 11)	32	4	5	$M \times 3 \times 32 \times 32$	Softplus
max-pooling (2 × 2)		2	0	$M \times 32 \times 32 \times 32$	
convolution (5 × 5)	96	1	2	$M \times 32 \times 15 \times 15$	Softplus
max-pooling (2 × 2)		2	0	$M \times 96 \times 15 \times 15$	
convolution (3 × 3)	192	1	1	$M \times 96 \times 7 \times 7$	Softplus
convolution (3 × 3)	128	1	1	$M \times 192 \times 7 \times 7$	Softplus
convolution (3 × 3)	64	1	1	$M \times 128 \times 7 \times 7$	Softplus
max-pooling (2 × 2)		2	0	$M \times 64 \times 7 \times 7$	
fully-connected	64			$M \times 64$	

Table A.3 AlexNetHalf with number of filters halved compared to the original architecture.

Appendix B

How to replicate results

Install PyTorch from the official website (https://pytorch.org/)

```
git clone https://github.com/kumar-shridhar/PyTorch-BayesianCNN
pip install -r requirements.txt
```

cd into respective folder/ task to replicate (Image Recognition, Super Resolution or GAN)

B.0.1 Image Recognition

```
python main_Bayesian.py
```

to replicate the Bayesian CNNs results.

```
python main_nonBayesian.py
```

to replicate the Frequentist CNNs results.

For more details, read the README sections of the repo : https://github.com/kumar-shridhar/PyTorch-BayesianCNN