A Comparison Study of Bayesian Approximation Techniques for Neural Networks on the CIFAR-10 Dataset

Muftiqur Rahman
muftiqurrahman@iut-dhaka.edu

Abstract

Neural Networks have achieved significant success in various domains such as computer vision applications or natural language processing etc. However, some challenges like overfitting and overconfidence limit their widespread applicability. Bayesian Neural Networks address these limitations by introducing the probabilistic framework for modeling uncertainty in deep learning, where the integration of uncertainty estimation is needed. In this research project, I implemented Bayesian Convolutional Neural Networks using the Bayes-by-Backprop approximation across three architectures—LeNet, AlexNet, and ResNet-50—on the CIFAR-10 dataset. The model is optimized by utilizing the ELBO loss function to evaluate their performance. Additionally, I implemented the Monte Carlo Dropout method into the same architectures for the comparison between two Bayesian approximation methods. Each method was implemented and assessed in terms of classification accuracy. Code and experiments are available at **GitHub link**.

1 Introduction

Image classification is one of the most fundamental tasks in computer vision with an extensive research is being conducted on various datasets. Traditional image classification methods primarily rely on standard deep learning techniques with deterministic neural networks. In contrast, Bayesian Neural Networks provide a probabilistic approach by incorporating uncertainty estimation not only during training but also in inference time. This enhances model robustness and reliability, which is particularly crucial in safety-critical applications such as autonomous driving and medical imaging, where incorrect predictions are specifically crucial. By quantifying uncertainty, BNNs enable more informed decision-making, ultimately improving trustworthiness and adaptability in real-world deployments.

In this report, I explored the performance of Bayesian Neural Networks (BNNs) using Bayes by Backprop(1) approach across various convolutional models using the CIFAR-10 dataset. For the experiment, I implemented the Bayesian Convolutional Layer and Bayesian Linear Layer from scratch at first and these layers were integrated into to build AlexNet(2), LeNet(3), and ResNet-50(4) architectures. To evaluate the model's performance, I implemented KL Divergence and ELBO loss functions. This part of the experiment is done by PyTorch. Subsequently, I extended this study by implementing another Bayesian Approximation technique: Monte Carlo Dropout(5) into AlexNet, LeNet, and ResNet-50 architectures by utilizing Keras's built-in function for this implementation. .

2 Background

2.1 Bayesian Neural Network

A Bayesian Neural Network (BNN) is a type of neural network that incorporates Bayesian inference into the traditional neural network. Unlike conventional neural networks, which provide point estimates for weights and do not quantify uncertainty in predictions, BNNs on the other hand, treat weights as probability distributions that enable them to quantify uncertainty in both model parameters and predictions. This is achieved by placing prior distributions over the weights and updating these priors using Bayes' theorem based on observed data.

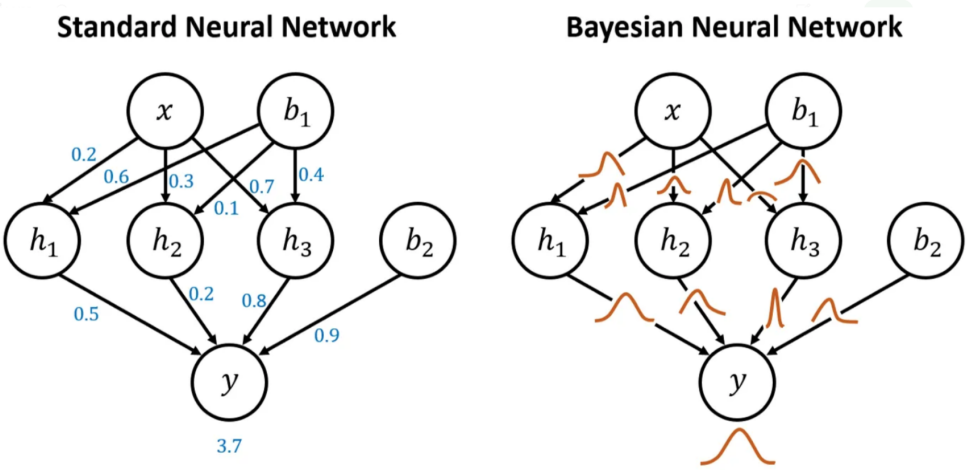

Figure 1: After training, the weights in a Standard Neural Network are fixed, while in a Bayesian Neural Network, the weights and biases are sampled from their respective probabilistic distributions.

In this project, Bayesian Neural Networks were implemented using two approximation inference techniques: Monte Carlo Dropout and Bayes by Backprop. MC Dropout applies dropout during both training and inference, allowing us to estimate uncertainty without altering the model's architecture. During inference, the model is run multiple times (n simulations) on the same input, with dropout applied each time. As different neurons are activated or deactivated in each run, we get different predictions for the same input. After running the model n times, we average the results to obtain the final classification prediction. On the other hand, the Bayes by Backprop approximates the true posterior distribution over weights using the Gaussian probabilistic distribution. During training, the goal is to minimize the Kullback-Leibler (KL) divergence between the true and approximate posteriors, which is equivalent to maximizing the Evidence Lower Bound (ELBO). At the inference time, weights are sampled differently from the learned variational distribution, causing its output to vary for the same input, resulting in the level of uncertainty.

2.2 KL Divergence

Kullback-Leibler (KL) divergence quantifies the difference between two probability distributions, serving as a non-symmetric measure of relative divergence. In machine learning, KL divergence is widely used in variational inference to compare an approximate posterior distribution with a prior distribution. The formula is

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_i P(i) \log \left(\frac{P(i)}{Q(i)} \right)$$

In Bayesian Neural Networks (BNNs), KL divergence plays a crucial role, as BNNs treat weight values as probability distributions rather than fixed parameters. The primary objective during training is to learn or approximate the posterior distribution of the weights given to the data. To achieve this,

the BNN minimizes KL divergence, ensuring that the learned weight distribution closely aligns with the true posterior. This regularization introduces a degree of uncertainty in predictions, enhancing the model's robustness and generalization capability.

2.3 ELBO Loss function

Evidence Lower Bound (ELBO) is usually used in variational inference. It consists of two terms: expected log-likelihood and KL divergence. The formula for Evidence Lower Bound is

$$\mathcal{L}_{\text{ELBO}} = \mathbb{E}_{q(z)}[\log p(x|z)] - D_{\text{KL}}(q(z) \parallel p(z))$$

The expected log-likelihood measures how good the model is at predicting the data. KL Divergence measures the difference between the approximate posterior distribution over the weights and a predefined prior Gaussian distribution. In the ELBO Loss function, KL Divergence acts as a regularizer in order to ensure that the learned weight distributions do not deviate too much from the prior distributions. In the Bayesian Neural Network, the ELBO loss function is used to approximate true posterior distribution from given training data D and initial weight W. During the training phase we have to maximize the ELBO loss function while keeping the learned weight distribution close to the prior distribution.

3 Methodology

3.1 Dataset

The CIFAR-10(6) dataset is a widely used benchmark in machine learning and computer vision, comprising 60,000 color images across 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Each image is a 32×32 RGB pixel representation. In this project, we will use CIFAR-10 for evaluation purposes.

3.2 Bayes by Backprop

This method introduces uncertainty in convolutional neural networks by learning a distribution over the weights instead of fixed-point values. In traditional neural networks, we usually optimize fixed-point weight values in such a way that they best fit the training data. In contrast, the Bayes-by-backprop layer represents weights as probability distributions parameterized by mean and standard deviation. The main goal of training is to optimize these distribution parameters. The Kullback-Leibler (KL) divergence measures how much the learned distributions W_μ, W_σ differ from the prior distribution $\text{prior}_\mu, \text{prior}_\sigma$. Throughout the process, the model attempts to minimize the KL loss. The overall process for the Bayes-by-Backprop method applied to the Linear Layer and Convolutional Layer are presented in Algorithm 1 and Algorithm 2, respectively.

Algorithm 1 Bayesian Linear Layer (BBBLinear)

Input: $input\_vector$, $in\_features$, $out\_features$, $bias$, $priors$.
Initialize learnable parameters:
    W_μ and W_ρ for weights, $bias_\mu$ and $bias_\rho$ for bias.
Reset parameters using normal distributions based on $posterior_{\mu\_initial}$ and $posterior_{\rho\_initial}$.
Sample weights and bias with noise:
    Generate random noise:
        $\epsilon \sim \mathcal{N}(0, 1)$
    $weight = W_\mu + \epsilon \cdot W_\sigma$,
    $bias = bias_\mu + \epsilon \cdot bias_\sigma$.
Perform linear transformation: $output = input\_vector \cdot weight^T + bias$.
=0

Algorithm 2 Bayesian Convolutional Layer (BBBConv2d)

Input: $input\_image$, $in\_channels$, $out\_channels$, $kernel\_size$, $stride$, $padding$, $dilation$, $bias$, $priors$, $groups$.
Initialize learnable parameters:
    W_μ and W_ρ for weights, $bias_\mu$ and $bias_\rho$ for bias.
Reset parameters using normal distributions based on $posterior_{\mu\_initial}$ and $posterior_{\rho\_initial}$.
Sample weights and bias with noise:
    Generate random noise:
        $\epsilon \sim \mathcal{N}(0, 1)$
    $weight = W_\mu + \epsilon \cdot W_\sigma$,
    $bias = bias_\mu + \epsilon \cdot bias_\sigma$.
Perform 2D convolution: $output =$ conv2d$(input_image, weight, bias)$.
=0

3.3 Convolutional Neural Network

For this research project, I have implemented four different convolutional architectures: LeNet, AlexNet, and ResNet-50 architecture. These architectures were implemented to evaluate the effectiveness of different convolutional designs on the CIFAR-10 dataset.

3.3.1 LeNet Architecture

The LeNet network consists of two convolutional layers, each followed by an average pooling layer. The first convolutional layer applies 6 filters of size 5x5 with sigmoid activation, and the second convolutional layer applies 16 filters of the same kernel size. The average pooling layer of size 2x2 is used after each convolution operation. After that. the output is flattened and passed through two fully connected layers with 120 and 84 dimensions, respectively, both using sigmoid activation. Finally, as the CIFAR-10 dataset has 10 classes, the final layer has 10 units each unit indicates a specific class.

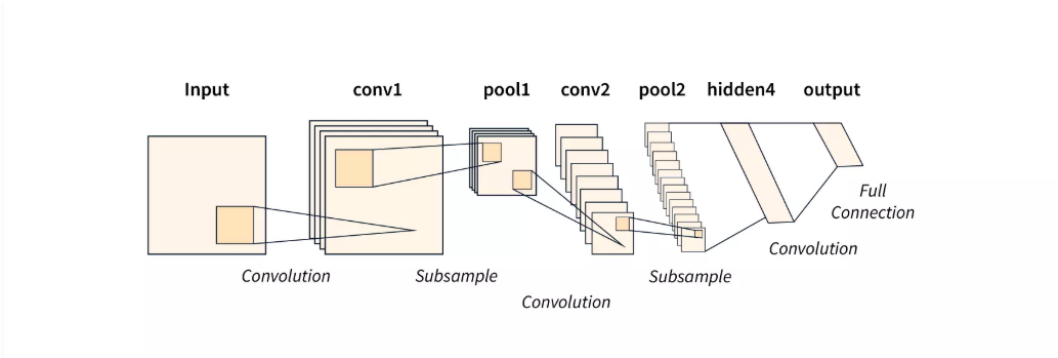

Figure 2: LeNet Architecture

3.3.2 AlexNet Architecture

The AlexNet architecture consists of five convolutional layers, some followed by max-pooling layers or ReLU activation functions. The first convolutional layer has 96 filters of size 11x11 with ReLU activation and a stride of 4, followed by max-pooling. The second layer uses 256 filters of dimension 5x5 that use ReLU for nonlinearity and is followed by a max-pooling layer. The next three convolutional layers apply 384, 384, and 256 filters of size 3x3 respectively and all of these layers use ReLU after the convolutional operations. The output is flattened and passed through two fully connected layers, each with 4,096 units. Finally, for the CIFAR-10 dataset, the last layer is set to 10 dimensions.

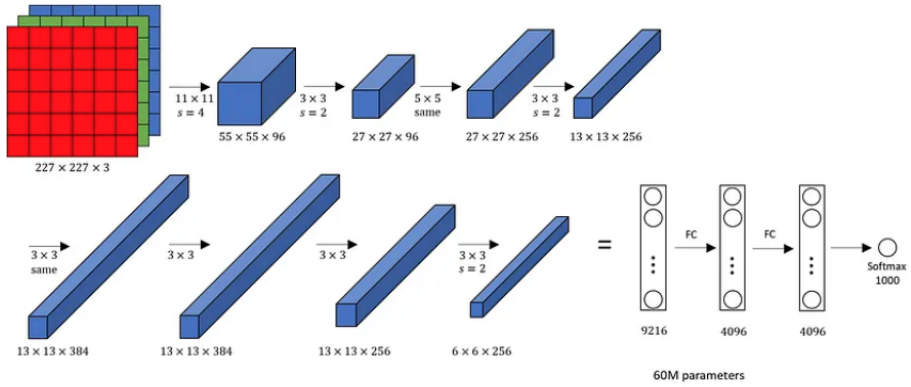

Figure 3: AlexNet Architecture

3.3.3 ResNet-50 Architecture

The ResNet-50 architecture, introduced by Kaiming He in 2015, was originally designed to address the vanishing gradient problem in deep neural networks. It has convolutional layers with additional residual blocks. Each residual block uses bottleneck design with 3 convolutional layers: 1x1, 3x3, and 1x1 filters, all with batch normalization and ReLU activation. Skip connections are incorporated to bypass these layers, allowing the network to train deeper models without encountering gradient vanishing issues. The network begins with a 7x7 convolutional layer and max-pooling, followed by four stages of residual blocks with increasing filter sizes (64, 128, 256, 512). The final output is passed through global average pooling and a fully connected layer with a softmax activation for classification.

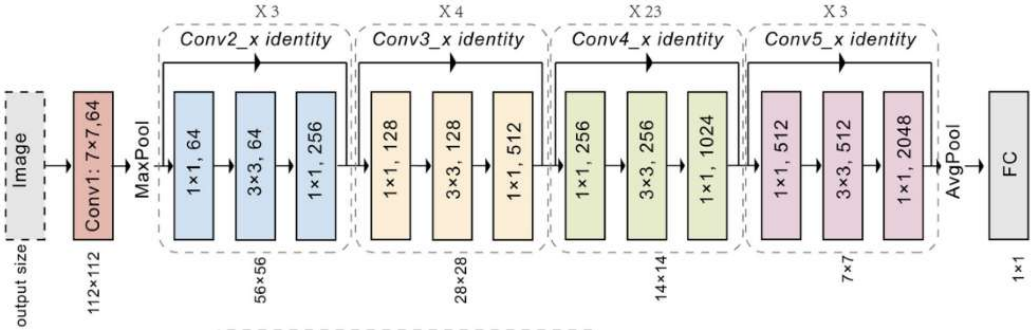

Figure 4: Resnet-50 Architecture

4 Experimental Setup

This project was entirely developed on Google Colab using a T4 GPU. I trained each convolutional layer for at least 100 epochs, during which the ELBO loss and accuracy were measured. To evaluate their performance, I incorporated Bayesian convolutional and Bayesian linear layers into LeNet, AlexNet, ResNet-50, and ConvNeXt architectures. Model accuracy was used as the primary evaluation metric. Several hyperparameters were considered, including the Adam optimizer with a learning rate of 0.001. For the ELBO loss function, I set the Beta value to 0.1 which acts as a regularization parameter in the ELBO Loss function. A higher Beta could lead to underfitting by overly punishing the model, whereas a lower Beta might cause overfitting by placing too much weight on cross-entropy

loss. Additionally, I experimented with the Monte Carlo Dropout Bayesian Approximation method on LeNet, AlexNet, and ResNet-50. Each model was trained for 100 epochs with a batch size of 64. In this setup, I used Stochastic Gradient Descent (SGD) as the optimizer with a learning rate of 0.01 and used categorical cross-entropy as the loss function. The dropout rate For Monte Carlo Dropout was set to 0.3.

5 Experimental Result

Table 1 reports the overall performance in terms of accuracy on the CIFAR-10 dataset using the Bayes by Backprop approach. In this approach, among four convolutional architectures, I found AlexNet to have the highest test accuracy.

Model Name	**Bayes by Backprop**			**Monte Carlo Dropout**		
	Train Acc	**Validation Acc**	**Test Acc**	**Train Acc**	**Validation Acc**	**Test Acc**
AlexNet	58%	57%	55.68%	**98%**	**79%**	**76.33%**
LeNet	53%	51%	51.13%	**70%**	**65%**	**67.18%**
ResNet-50	70%	68%	53.23%	**66%**	**64%**	**65.39%**

Table 1: Model Accuracy Comparison for Variational Inference BNN and Monte Carlo Dropout BNN

In this section, we compared the performance of two Bayesian Neural Network (BNN) approaches—Bayes by Backprop and Monte Carlo Dropout—across three architectures: AlexNet, LeNet, and ResNet-50, using the CIFAR-10 dataset. Among the models, AlexNet consistently achieved the highest test accuracy in both approaches, with 76.33% for Monte Carlo Dropout and 55.68% for Bayes by Backprop approach. Notably, Monte Carlo Dropout models maintained a closer alignment between training, validation, and test accuracy, indicating better regularization and robustness compared to the other method.

6 Conclusion

In this study, I conducted experiments on the CIFAR-10 dataset using two Bayesian approximation approaches: Monte Carlo Dropout and Bayes by Backprop. The results of this experiment showed that Monte Carlo Dropout consistently outperformed Bayes by Backprop across all models, achieving higher accuracy and better generalization on the test datasets. However, due to time constraints on Google Colab, the experiments were conducted with a limited number of epochs, which may have impacted the performance of the latter approach, leading to results that were lower than expected.

References

[1] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural network," in *International conference on machine learning*. PMLR, 2015, pp. 1613–1622.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[3] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed., vol. 2. Morgan-Kaufmann, 1989. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

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

[6] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)," *URL http://www. cs. toronto. edu/kriz/cifar. html*, vol. 5, no. 4, p. 1, 2010.

A Additional Figure and Plots

This appendix section presents figures demonstrating the training and validation loss, as well as accuracy, across the epochs. The section is organized into two parts, each containing three subsections. Each subsection includes the loss and accuracy curves for AlexNet, LeNet, and ResNet-50.

A.1 Bayes by Backprop

A.1.1 AlexNet

Figure 5 illustrates the loss curve for the AlexNet model, which shows a gradual decrease over the epochs. Throughout the training process, the training and validation accuracies remained closely aligned. Additionally, as shown in Table 1, the test accuracy closely mirrors both the training and validation accuracies.

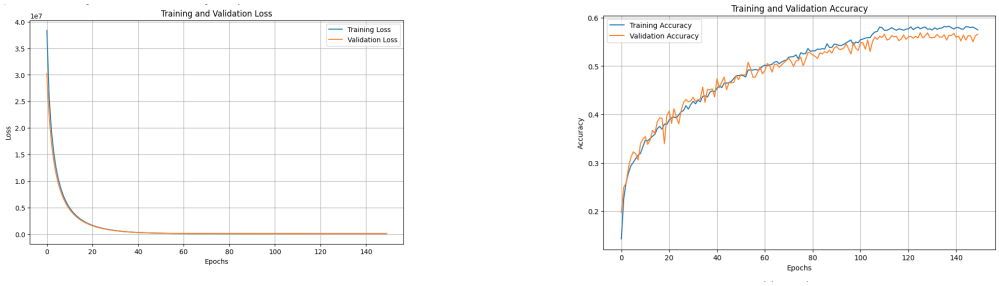

Figure 5: Fig: Training and validation loss and accuracy curves of AlexNet during the training process using Bayes by Backprop.

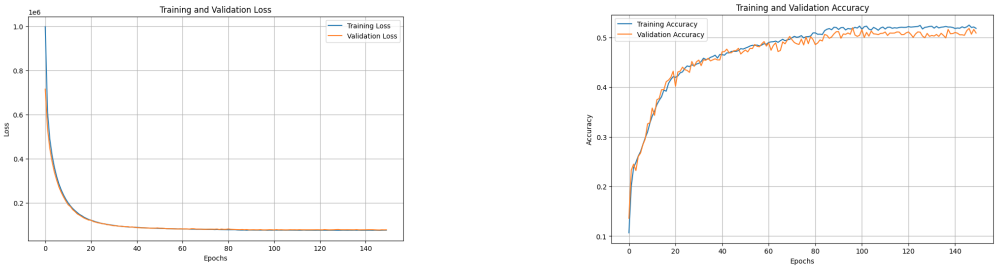

Figure 6: Fig: Training and validation loss and accuracy curves of LeNet during the training process using Bayes by Backprop.

A.1.2 LeNet

Like AlexNet, the LeNet model exhibits a similar pattern. Its loss value decreased steadily and both training and validation accuracies progressively increased over the epoch.

A.1.3 ResNet-50

In the case of ResNet, the loss values for both training and validation decreased gradually. However, the training and validation accuracies fluctuated throughout the process. Moreover, the test accuracy

(53.23%) differed significantly from both the training and validation accuracies (70% and 68%, respectively).

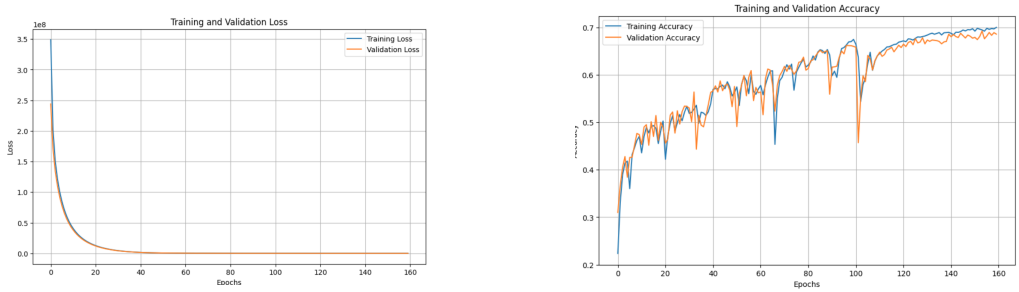

Figure 7: Fig: Training and validation loss and accuracy curves of ResNet-50 during the training process using Bayes by Backprop.

A.2 Monte Carlo Dropout

A.2.1 AlexNet

In the Monte Carlo Dropout approach for AlexNet, I set the dropout rate to 0.5. While the training loss decreased gradually, the validation loss began rising consistently after the 40th epoch. Up to this point, the validation accuracy closely followed the training accuracy. After that, it started to deviate, and by the end of training, the training accuracy reached approximately 98%, while the validation accuracy was around 79%.

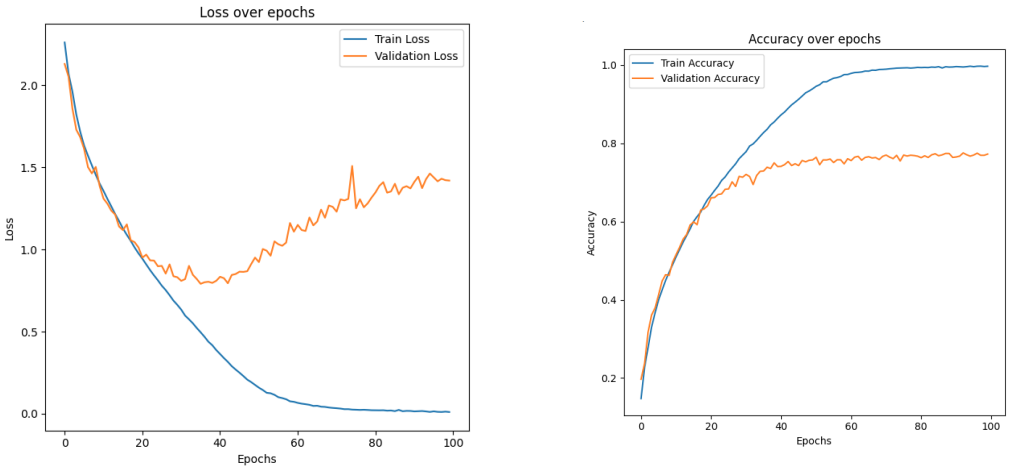

Figure 8: Fig: Training and validation loss and accuracy curves of AlexNet during the training process using Monte Carlo Dropout.

A.2.2 LeNet

As shown in Fig. 9, LeNet model, using Monte Carlo Dropout techniques causes both training and validation loss to decrease steadily. The training accuracy continues to improve over time, whereas the validation accuracy plateaus at 65%, indicating limited generalization. The test accuracy was 67%, suggesting that Monte Carlo Dropout helps reduce overfitting but does not fully eliminate it.

A.2.3 ResNet-50

For the ResNet-50 architecture in this method, the training loss decreases steadily, while the validation loss fluctuates significantly, which is also reflected in the validation accuracy curve. Despite the fluctuations, the validation accuracy shows an overall upward trend

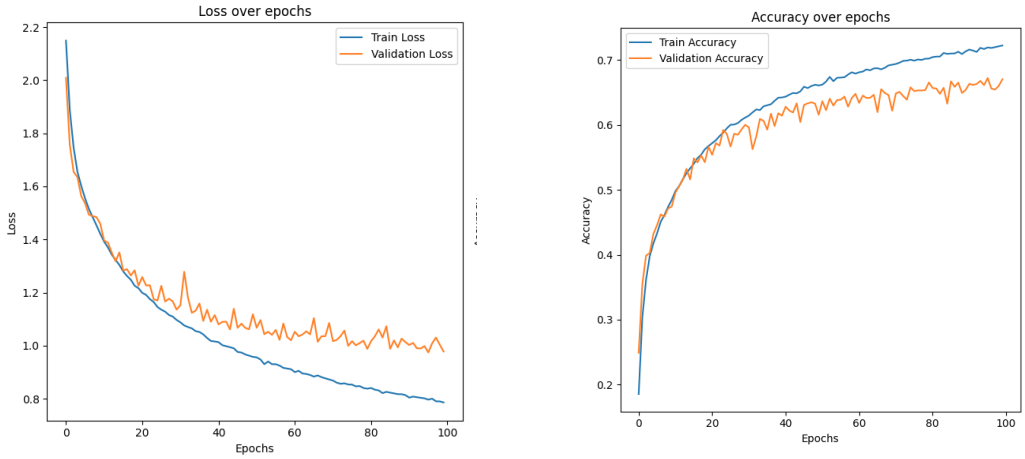

Figure 9: Fig: Training and validation loss and accuracy curves of LeNet during the training process using Monte Carlo Dropout.

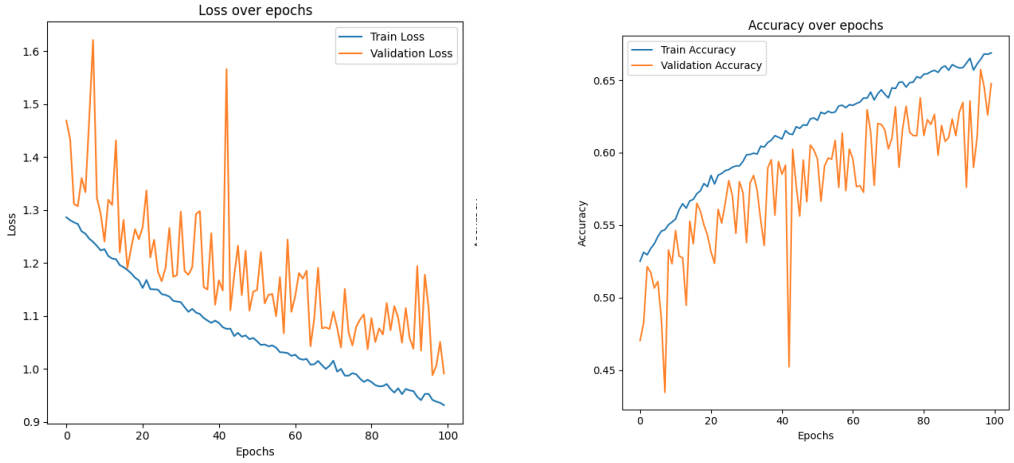

Figure 10: Fig: Training and validation loss and accuracy curves of ResNet-50 during the training process using Monte Carlo Dropout.