Bayesian Convolutional Neural Network

Alex Kendrick (ajk4yq), Brook Assefa (rnc3mm), Grant (yep8eq)
DS6040: Bayesian Machine Learning
Semester: Summer 2023

1. Problem Statement

Image classification is a fundamental task in computer vision that involves categorizing images into predefined classes. Convolutional Neural Networks (CNNs) have achieved remarkable success in image classification tasks. However, traditional CNNs often lack the ability to quantify uncertainty in their predictions, which is crucial for certain applications. In this project, we propose to explore the use of a Bayesian Convolutional Neural Network (Bayesian CNN) for image classification, leveraging the power of Bayesian inference to provide probabilistic predictions and uncertainty estimates.

The objective of this project is to modify existing model architectures to add a Bayesian twist to a CNN model for image classification. A Bayesian approach could enable insight into the confidence associated with each prediction based on a distribution of inferences.

2. Data Description

We used the CIFAR-10 dataset. CIFAR-10 is a commonly used dataset and a benchmark for image classification training tasks. CIFAR-10 consists of 60,000 labeled 32x32 images across 10 classes. There are 6,000 images per class with 5,000 specified for training and 1,000 for testing. The classes represent: airplane, car, bird, cat, deer, dog, frog, horse, ship, and truck. This implies that the CIFAR-10 dataset is a balanced dataset. Being a balanced dataset can significantly simplify the structure required to train a model. Accordingly, we will not need to worry about applying weights to a random sampler or misleading and erroneous classifications due solely to the distribution of classes from the training data.

We chose to implement the task in PyTorch due to the readily available examples and tutorials on how to enable data flow into the model. PyTorch models rely on a dataset and data loader to enable feeding training data into the model. Normally, for custom datasets, an entire data class is required to facilitate the data structure required by the torch data loaders, but because CIFAR-10 is such a common dataset for the image classification task, PyTorch has a package, torchvision, which enables seamless construction and download of the dataset to a user specified file structure. Additionally, and prior to constructing a dataset, image augmentations are common in computer vision tasks. Some common augmentations include Vertical image flipping, horizontal image flipping, random cropping. We apply these three image augmentations and normalize the image pixels and modify data types to tensors to facilitate integration with PyTorch.

After having a training dataset and test dataset, we can instantiate a data loader. The data loader divides the dataset into batches; this is necessary because training a neural network often involves feeding the data in batches. You can specify the batch size, and whether the data loader should shuffle the data at every epoch.

3. Model Description

We trained 7 different CNNs. All the CNNs were trained with 3 hidden layers. The first 2 are both convolution layers while the final hidden layer is a linear layer. One we trained as a traditional CNN to use as a baseline for comparison and the other 6 were trained with a Bayesian linear layer, but with different prior distributions specified for the weights. The models were all trained over 100 epochs using all the 50,000 training observations. Theoretically all layers could use Bayesian weights, but we found that slowed down the training of the model significantly. When using the Data Camp environments to host the notebooks we found that models training for a significant amount of time could cause the environment to timeout. As a compromise to account for those training times and DataCamp's time limits for running notebooks we restricted the use of Bayesian weights to the final linear layer.

For the Bayesian CNNs we chose 4 different distribution families for the priors:

- Normal/Gaussian
- Studentized T
- Laplace
- Cauchy

The hyperparameters for the models are shown in table 1.

Table 1 Prior Distributions for Bayesian Models

Prior Distribution Family	Location	Scale	DF
Normal/Gaussian	0	1	
Normal/Gaussian	0	3	
Normal/Gaussian	0	10	
Studentized T	0	1	10
Laplace	0.5	1	
Cauchy	0	1	

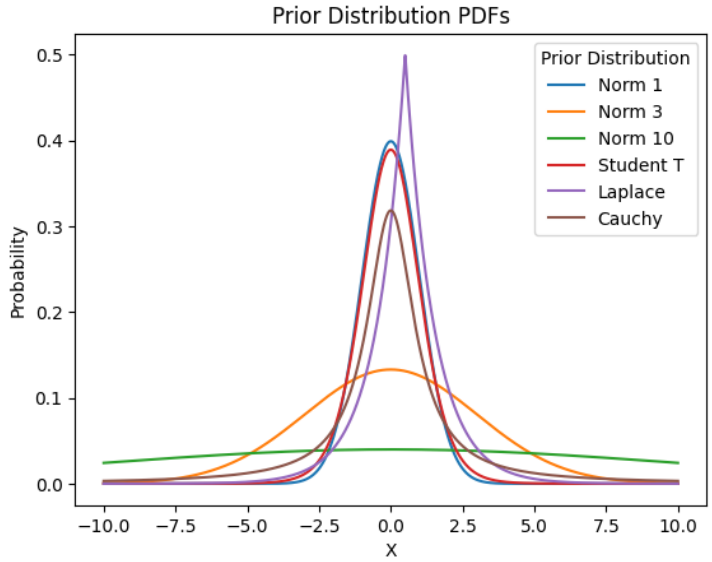

Figure 1 Prior Distribution PDFs

The models were created using the PyTorch package. The Bayesian layers were implemented using the Bayesian Layers in Torch Zoo(BLiTZ) package. The posterior weight distributions were estimated using variational inference, this approach is built into the BLiTZ package.

For training the models we relied upon cloud CPU provided in the DataCamp workspaces. Use enabled enabling scheduled, and unattended training. The Bayesian nature of the selected convolutional neural networks came at the cost of training time. For a single model to train one hundred epochs of 50,000 image observations, it took upwards of eight hours.

The Gaussian/Normal distribution was commonly found in literature we reviewed on Bayesian CNNs. It was considered due to its symmetric, bell-shaped nature which is well-suited for capturing moderate uncertainty in predictions and serving as a baseline for comparison. Multiple scale parameters were selected for the Gaussian/normal distributions to potentially observe the effect on model accuracy of more non-informational priors.

The Student's t-distribution, with its heavier tails, is another noteworthy candidate, especially when dealing with a moderate degree of uncertainty and handling situations where extreme values could occur. Additional literature indicated that ‘heavier tailed’ distributions for the weights of a neural network could have favorable impacts on the performance. We theorize that this relates to non-informational priors, but also unobserved gains during optimization.

Among these distributions, the Cauchy and Laplace distributions were chosen due to their shared heavy-tailed nature, which accommodates extreme values. In contrast to the Gaussian distribution, these alternatives demonstrate greater robustness to outliers, assigning higher likelihoods to extreme observations. By leveraging these distributions, the goal was to strengthen the Bayesian CNN's capability to capture uncertainty and generate robust predictions.

The Cauchy distribution was used as a prior distribution for weights, promoting robustness against outliers and facilitating modeling of heavy-tailed distributions. Moreover, the Cauchy distribution contributes to the estimation of the posterior distribution during training, thus enhancing the quantification of uncertainty. This innate resilience to outliers significantly improves the model's overall performance and the robustness of its predictions

Similarly, the Laplace distribution primary advantage lies in its ability to induce sparsity through the prior distribution. This trait favors solutions with fewer non-zero weights, allowing the model to focus on relevant image features while ignoring irrelevant ones. Like the Cauchy distribution, the Laplace distribution supports uncertainty estimation and outlier handling, leading to more accurate predictions.

During training we explored the use of KL (Kullback-Leibler) loss for optimization within the network. This measures the difference between two probability distributions and is commonly used in Bayesian inference. However, we encountered some convergence issues with KL loss in comparison to the traditional cross-entropy loss. In our research, we found indications that KL loss is proportional and related to cross-entropy loss, so we moved forward with cross-entropy loss as the loss function for the models.

4. Results

The Bayesian models all outperformed the non-Bayesian model at 100 epochs. The accuracy of the Bayesian models all came out very similarly as well. The accuracy for the Bayesian models is clustered right around 0.62, while the accuracy for the non-Bayesian model came out to 0.5296.

Table 2 Model Accuracies

Model	Accuracy at 100 Epochs
Gaussian Std Dev 1	0.6256
Gaussian Std Dev 3	0.6270
Gaussian Std Dev 10	0.6130
Studentized T	0.6188
Laplace	0.6199
Cauchy	0.6277
Non-Bayes	0.5296

Based on this it does not look like the choice priors have much of an influence on testing result accuracy. Though treating the linear layer as Bayesian, so having the weights represented as a distribution rather than a point estimate, does result in an increased accuracy. Overall, our model performance does not come close to state-of-the-art performance. We theorize that a better model specification would enhance overall accuracy to a greater degree than tuning the Bayesian prior hyperparameters or continuing to explore additional distributions.

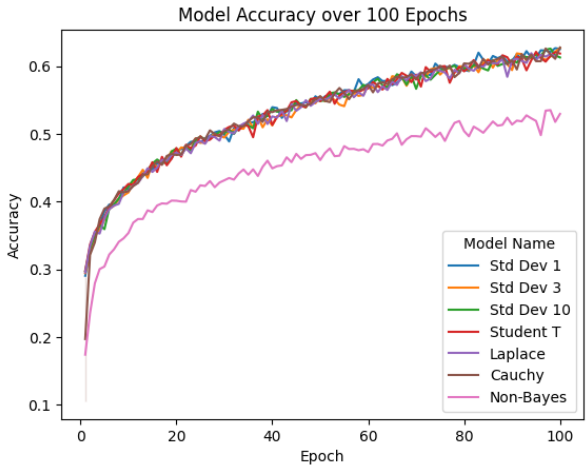

Figure 2 Model Accuracy

Additionally, from spot-checking the some of the testing results, such as in Figure 3, we found that the images that have been misclassified or have low probabilities of assigned to the correct class are images that even visually can be difficult to classify. This makes sense conceptually as the further the image is from a "typical" image of that class the less information the network will have on it.

One thing to consider is that our prior distributions all had similar shapes. A topic for further exploration in the future could be exploring more radically different prior distributions to gauge their impact on the model. Such as studying non-symmetrical or discrete distributions.

Figure 3 Model Predictions

5. Discussion

Bayesian Convolutional Neural Networks are not currently state of the art in terms of accuracy prediction on their own on the CIFAR-10 dataset. Current state of the art models have accuracies over 99% and implement entirely different model architectures. The key added benefit from the Bayesian approach to convolutional neural networks is the ability to quantify and explain how and when the model is uncertain. Our custom, BCNN, achieves non-deterministic or stochastic modeling outputs through the incorporation of both dropout layers

and a single Bayesian linear layer. Applying Bayesian methods to state of the art model architectures could be interesting but was out of scope for this project.

We explored the use of a variety of priors for the weights on the single Bayesian linear layer and found similar accuracies among the models. Our best explanation for similar performance across our different models is that the training data overwhelmed the selected priors. In other words, the distribution of the priors for the weights are transformed through the iterations and epochs of training on the convolutional neural network.

To enable inference and quantification and visualization of the uncertainty we implemented a Monte Carlo sampler. This takes one-hundred samples from the final weight distributions when predicting what an image will be, showing variation in the outcome. It allows for better interpretation in the uncertainty of the prediction. This is especially helpful when the predictive score for an image is low, as it shows what other classes have similar probabilities. We report similar performance for all our trained Bayesian models. Additionally, we report enhanced performance of each Bayesian model over the deterministic, yet effectively equivalent non-Bayesian convolutional neural network. In respect to solving the problem we set out to identify, we identified and implemented applied techniques such as implementation of Bayesian Linear layer, drop out layers, and Monte Carlo sampling to quantify the uncertainty associated with the classification of any given image from the test dataset.

Future topics for research that arose from our investigation could involve the following: more diverse prior selection. Selecting non-symmetric distributions or even discrete distributions could lead to interesting results, but we would need to make sure they encapsulate the domain of the weight values well. Adding Bayesian weights to more than just the final linear layer could be interesting as it would add more uncertainty to the model and potentially result in models with varied performance due varied optimization surface search. A better computing environment would be needed to run those models in a reasonable amount of time. With a better computing environment, we could explore extending training beyond one-hundred epochs. Using more complex network architectures more closely related to the current state-of-the-art models could be an interesting topic for further investigation as well.

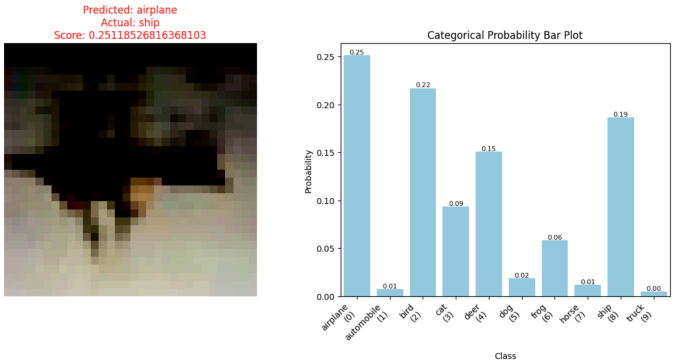

Figure 4 Prediction with Probabilities

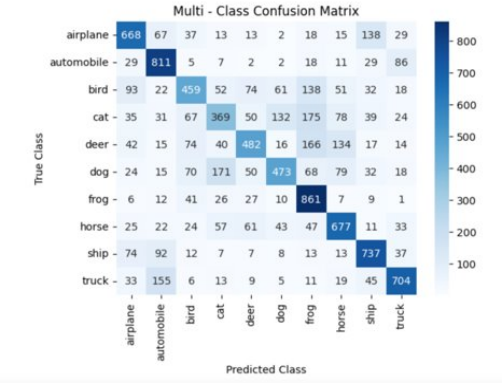

Figure 5 Multi-Class Confusion Matrix

6. Appendix

Link to Notebooks on DataCamp:
https://app.datacamp.com/workspace/w/ac33909f-20cc-4e59-909b-fb39bcbefdc6/edit

References:

Bayesian Deep Learning Workshop NIPS 2016. (2017, March 8). *History of Bayesian Neural Networks (Keynote talk)* [Video]. YouTube. https://www.youtube.com/watch?v=FD8l2vPU5FY

Gal, Y., & Ghahramani, Z. (2016). Dropout as a Bayesian approximation: representing model uncertainty in deep learning. *International Conference on Machine Learning*, 1050–1059. http://proceedings.mlr.press/v48/gal16.pdf

Johnson, M. (2022, December 17). Understanding a Bayesian neural Network: a tutorial. *nnart*. https://nnart.org/understanding-a-bayesian-neural-network-a-tutorial/#What_is_a_Bayesian_Neural_Network

Kumar-Shridhar. (n.d.). *GitHub - kumar-shridhar/PyTorch-BayesianCNN: Bayesian Convolutional Neural Network with Variational Inference based on Bayes by Backprop in PyTorch.* GitHub. https://github.com/kumar-shridhar/PyTorch-BayesianCNN

Nalisnick, E. (2018). On Priors for Bayesian Neural Networks. *UC Irvine Electronic Theses and Dissertations*. https://escholarship.org/content/qt1jq6z904/qt1jq6z904.pdf?t=pdak4f

Papers with Code - CIFAR-10 Benchmark (Image Classification). (n.d.). https://paperswithcode.com/sota/image-classification-on-cifar-10

piEsposito. (n.d.-a). *blitz-bayesian-deep-learning/doc/layers.md at master · piEsposito/blitz-bayesian-deep-learning*. GitHub. https://github.com/piEsposito/blitz-bayesian-deep-learning/blob/master/doc/layers.md#class-BayesianLinear

piEsposito. (n.d.). *GitHub - piEsposito/blitz-bayesian-deep-learning: A simple and extensible library to create Bayesian Neural Network layers on PyTorch.* GitHub. https://github.com/piEsposito/blitz-bayesian-deep-learning#The-purpose-of-Bayesian-Layers

Probability distributions - torch.distributions — PyTorch 2.0 documentation. (n.d.). https://pytorch.org/docs/stable/distributions.html#studentt

Shridhar, K., Laumann, F., & Liwicki, M. (2019). A Comprehensive guide to Bayesian Convolutional Neural Network with Variational Inference. *arXiv (Cornell University)*. https://arxiv.org/pdf/1901.02731.pdf

Training a Classifier — PyTorch Tutorials 2.0.1+cu117 documentation. (n.d.). https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

TwinEd Productions. (2021, April 3). *Bayesian Neural Network | Deep Learning* [Video]. YouTube. https://www.youtube.com/watch?v=OVne8jDKGUI