Fashion Data Classification via Bayesian Convolutional Neural Networks

Siddarth Madala
Department of Statistical Sciences
Duke University
siddarth.madala@duke.edu

1 Introduction

Image classification remains at the forefront of machine learning (ML) research with its wide applications in autonomous driving, medical imaging, augmented reality, etc. However, current literature focuses primarily on increasing test accuracy on common benchmark datasets. As state-of-the-art (SOTA) models are released at reduced intervals, it is important to uncover the black box of neural networks and understand the how confident they are in their decisions. Interpretability is critical for AI adoption in sensitive government, business, and medical applications. In this paper, we employ Bayesian Convolutional Neural Networks (BCNN) – a probabilistic CNN architecture – to gain insights into the confidence of neural network predictions on the Fashion MNIST dataset. Bayesian neural networks differ from DNNs in that their weights are assigned a probability distribution instead of a single value/point estimate. These probability distributions describe the uncertainty in weights and can thus be used to estimate uncertainty in predictions. Training a Bayesian neural network via variational inference learns the parameters of these distributions instead of the weights directly via *Bayes by Backprop*. We finally explore the BCNN's efficacy in the tasks of image classification on familiar examples, as well as its diffidence in classifying out-of-dataset images.

2 Data

The MNIST dataset is corpus of handwritten digits that is commonly used for training and benchmarking computer vision algorithms. While most models achieve very high accuracy on MNIST, their performance does not generalize well to other datasets. This is because MNIST's simplicity often fails to effectively represent model performance on modern computer vision tasks (i.e. detection, segmentation) [1]. Thus, in this project, we use the Fashion-MNIST dataset produced by e-commerce company Zalando to alleviate some of the common issues stemming from the traditional MNIST dataset.

The Fashion-MNIST dataset consists of 70,000 28x28 grayscale images. 60,000 of these images belong to the training set, while the remaining 10,000 belong to the test set. There are 10 labels, with integer labels of 0-9 as follows: $\{$T-shirts/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot$\}$. Hence, the data type is nominal discrete. This dataset is highly heterogeneous yet also extensive, making it a highly promising for studying classification with BCNNs.

We use Uniform Manifold Approximation and Projection (UMAP) [2] to project and visualize our raw data. This embedding method is a successor to t-Distributed Stochastic Neighbor Embedding (t-SNE). UMAP uses graph layout algorithms to arrange data in low-dimensional space. The algorithm constructs a high dimensional graph representation of our data – called a fuzzy simplicial complex – and then optimizes the low-dimensional graph to be as structurally similar as possible. This is advantageous because while t-SNE does not preserve global data structure, UMAP's inter-cluster distances are just as interpretable as intra-cluster distances. Thus, UMAP ensures that local structure is preserved in balance with global structure.

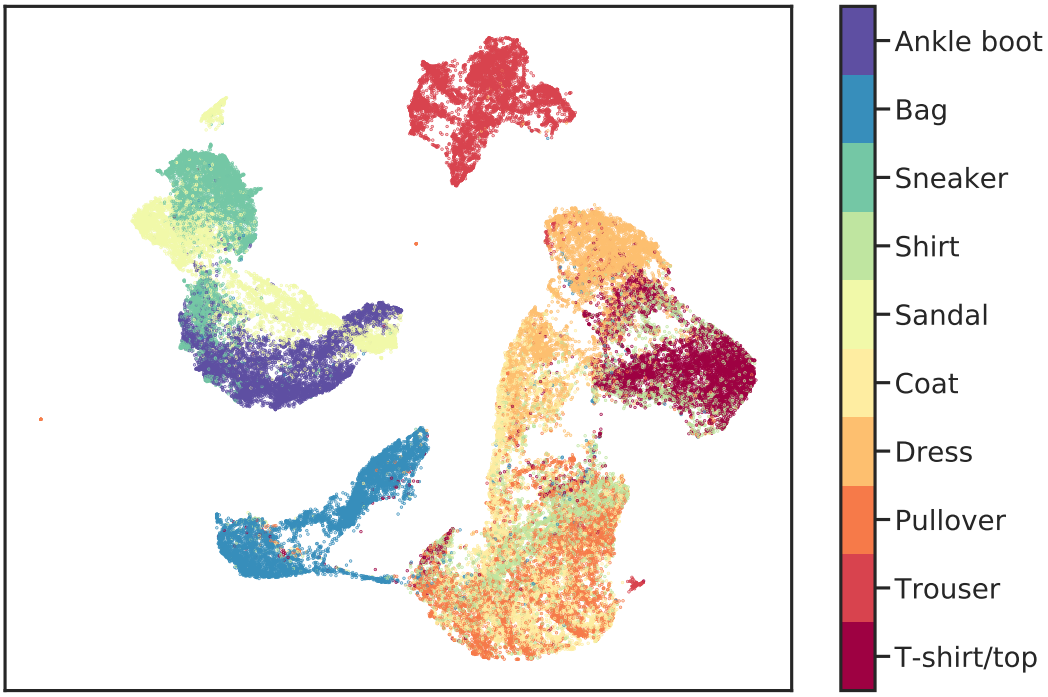


Figure 1: UMAP Projection of Fashion MNIST data Based on RAW Pixel Values

Our projection separates the classes while preserving the global structure (Figure 1). In general, the projection separated pants and footwear from shirts, coats and dresses, which are significantly different types of clothing. Unlike easier datasets (i.e. MNIST), there were classes that did not separate cleanly. Specifically, T-shirts, shirts, dresses, pullovers, and coats overlap in the 2D projection. More ideal separation can be realized by using label information with UMAP; however, the goal is to predict classes without labels.

3 Model

3.1 Architecture Selection

Deep Neural Networks (DNNs) are systems that learn to perform tasks from examples without prior knowledge of the task. Their capacity is controlled by varying their layers' depth and breadth, and they make assumptions that are strong and mostly correct [3]. Compared to traditional feed-forward neural networks, Convolutional Neural Networks (CNNs) have much fewer connections/parameters while retaining a similar upper bound in performance as DNNs. CNNs operate by employing layers of feature maps that filter for representative pixel intensities found through stochastic gradient descent (SGD) (Figure 2). Feature signals are then boosted through transformations [4] such as filtering and pooling which involve convolutional and statistical (i.e. average, max, norm, etc.) operations respectively. This interleaving of convolutions, pooling, and dense layers constructs a feature hierarchy that is invariant to image translation and exponentially more computationally efficient.

In practice, traditional neural network architectures are over-confident predictors. This is because the expressiveness of CNNs makes them overfit smaller datasets, thereby performing well during training but poorly during testing. Various regularization techniques [5] exist for preventing overfitting (i.e. early stopping, weight decay, dropout, etc.). However, using single-point estimates for real world decisions is unjustifiable. Thus, by introducing Bayesian learning to the weights and final prediction of the network (Figure 2) we both counter overfitting and add a measure of uncertainty and regularization at inference time. [6].

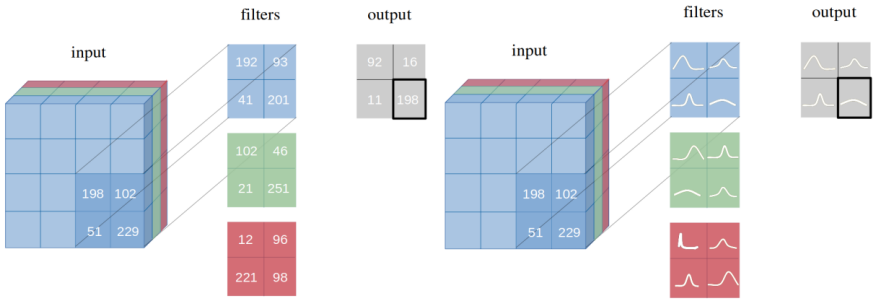


Figure 2: Traditional Convolutional Layer vs. Bayesian Convolutional Layer Architecture

3.2 Bayesian Derivation

3.2.1 Probabilistic Model

Let our neural network be represented as a probabilistic model $p(y|x, f)$, where y is the set of classes and $p(y|x, f)$ is a categorical distribution. We assign a prior distribution over the space of functions/weights $p(f)$ that could have generated our data. Given a training set $T = \{x_i, y_i\}$, we construct the likelihood/sampling function parameterized by f, $p(T|f) = \prod_i p(y_i|x_i, f)$. The maximum likelihood estimate (MLE) of f is thus given by maximizing the likelihood function. By Bayes theorem, multiplying our prior by the likelihood is proportional to the posterior distribution, $p(f|T) \propto p(T|f)p(f)$. Maximizing $p(T|f)p(f)$ gives the maximum a posteriori (MAP) estimate of f [7]. Because negative log likelihood is used during training for normal neural nets, the optimization objective is the same as for the point estimate MLE in additional to the regularization term from the log prior. However, both MLE and MAP give point estimates of parameters. The full posterior distribution over parameters allows us make predictions that take weight uncertainty into account. This is given by marginalizing out f to get the the posterior predictive distribution:

$$p(y|x, T) = \int p(y|x, f)p(f|T)df \tag{1}$$

This is equivalent to averaging predictions from an ensemble of neural networks weighted by the posterior probabilities of their parameters [8].

3.2.2 Variational Inference

Even with a small number of parameters, inferring the model posterior $p(f|T)$ in a Bayesian NN is intractable [9]. Instead, Variational inference is widely used to approximate posterior densities for Bayesian models as an alternative strategy to Markov chain Monte Carlo (MCMC) sampling. Compared to MCMC, variational inference tends to be faster and easier to scale to large data [10]. Moreover, while MCMC sampling uses sampling to converge to a stationary posterior distribution, variational inferences uses optimization techniques to achieve similar results in much less time. However, MCMC is provides guarantees of producing (asymptotically) exact samples from the target density, while variational inference can only find a density close to the target through stochastic means [11]. Therefore, we approximate the posterior with a variational distribution $q(f|\theta)$ whose form and parameters we know. Approximation is achieved by minimizing the Kullback-Leibler (KL) divergence – a measure of similarity between two distributions – between $q(f|\theta)$ and $p(f|T)$. From Blei et al. [10], we note:

$$\arg\min_{\theta} \ \mathrm{KL}(q(f|\theta) \parallel p(f|T)) = \arg\min_{\theta} \mathbb{E}_{q(f|\theta)}[\log q(f|\theta)] - \mathbb{E}_{q(f|\theta)}[\log p(f|T)] \tag{2}$$

$$\mathrm{KL}(q(f|\theta) \parallel p(f|T)) = \mathbb{E}_{q(f|\theta)}[\log q(f|\theta)] - \mathbb{E}_{q(f|\theta)}[\log p(f, T)] + \mathbb{E}_{q(f|\theta)}[\log p(T|f)] \tag{3}$$

This objective is not computable because of its dependence on $\log p(T)$ which is unknown. Therefore, we optimize an alternative objective that is equivalent to the KL divergence up to an added constant. This new function is called the evidence lower bound (ELBO) and is defined as:

$$ELBO(q) = \mathbb{E}[\log p(f, T)] - \mathbb{E}[\log q(f|\theta)] \tag{4}$$

Maximizing the ELBO is equivalent to minimizing the KL divergence. The final loss function is approximated by drawing samples f_i from $q(f|\theta)$ and given by [8]:

$$\mathrm{KL}(q(f|\theta) \parallel p(f|T)) \approx \frac{1}{N} \sum_{i=1}^{N} [\log q(f_i|\theta) - \log p(f_i) - \log p(T|f_i)] \tag{5}$$

We employ a multivariate Gaussian with diagonal co-variances. While this implies that latent dimensions are uncorrelated, it it a necessary trade-off to ensure ease of implementation. By parameterizing f with μ, Σ, we double the number of parameters in the BCNN compared to a traditional CNN. This is a necessary sacrifice in space complexity to derive confidence of predictions after experimentation.

3.2.3 Training

Training the BCNN utilizes *Bayes by Backprop* described in Blundell et al. [7] and expanded to CNNs in Shridar et al. [6]. In summary, a single forwards and backwards pass through the BCNN encompasses one cycle. In the forward pass of the cycle, a single sample is drawn from the variational posterior and is used to evaluate the approximate loss function defined in (5). To ensure the backwards pass terminates, we utilize the *reparameterization trick* described in Xu et al. [12] where we sample $\epsilon \sim \mathcal{N}(0, I)$ and transform it via mean shifting and Hadamard product $t = \mu + \Sigma \odot \epsilon$. We let the function/weight prior be a mixture of two Gaussians with respective weights and standard deviations set as hyper-parameters before training. During a backwards pass, gradients of μ, Σ are accumulated via backpropagation/gradient descent to update their values via optimizer (Adam).

3.2.4 Uncertainty

In Bayesian modeling, uncertainty comes in two flavors: aleatoric and epistemic uncertainty [13]. Aleatoric uncertainty measures the noise inherent in observations and is uniform along the dataset. Epistemic uncertainty, or model uncertainty, is variation given by the choice and training of the model, which can be reduced with enough data [14]. Traditional neural networks are difficult to quantify epistemic uncertainty for, but BCNNs are built with uncertainty in mind. Thus, we quantify epistemic uncertainty in our results.

3.3 Implementation Details

We refer to `Bayesian-CNN.py` for the implementation of this model on the Fashion-MNIST dataset. We utilize Google's Tensorflow and Keras deep learning frameworks along with Tensorflow Probability for implementations of BCNN layers. Our BCNN consists of two convolutional layers with kernel sizes of 3 and strides of 2. These layers are fed to two dense layers that map to a 10 dimensional vector. The network outputs a single integer corresponding the class labels described above. We manually implement the loss function described above and attach it as a callback to the network to track loss throughout training.

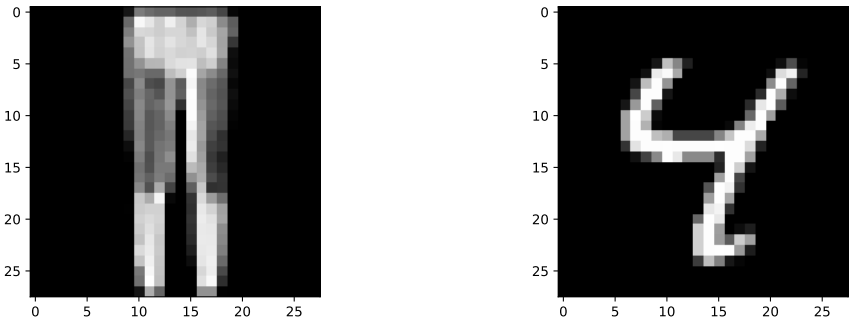
4 Results

4.1 Classification

Our network converges after training for 20 epochs. To evaluate performance, we observe that the test accuracy on recognizable samples is 0.88, which is very high for a small proof-of-concept network. A traditional CNN of similar architecture was trained alongside the BCNN to compare performance between both models. Test accuracies were comparable with the CNN outperforming the BCNN by 2 percentage points. This difference is negligible in practice. More important is the fact that the former is a complete black box for which there is no method to derive prediction confidence from in production. Thus, even with a slightly lower test performance, the BCNN can be considered more interpretable, and therefore, more useful in practice.

4.2 Thresholded Prediction

To demonstrate the confidence of a BCNN, we only perform prediction on examples that we are confident about. Thus, we set a threshold of 0.2 representing the minimum belief required by the network to make a prediction about a clothing item. Any confidence of an item's class below this value is deemed unrecognizable and is not given a label during testing. Figure 3a shows an easily identifiable pair of pants while Figure 4 displays the final dense layer's distributions of confidence for each class. Notice that the mean probability of the Trousers class is both above 0.2 and is the maximally likely estimate for the given data. Therefore the network correctly predicts that the image belongs to the Trousers class. Similarly, on unfamiliar data (Figure 3b), the BCNN fails to provide a prediction for the out-of-dataset number as no class reaches the median confidence threshold required to output a label (Figure 5).



(a) Familiar Example

(b) Unfamiliar Example

Figure 3: Examples of Familiar and Unfamiliar Data

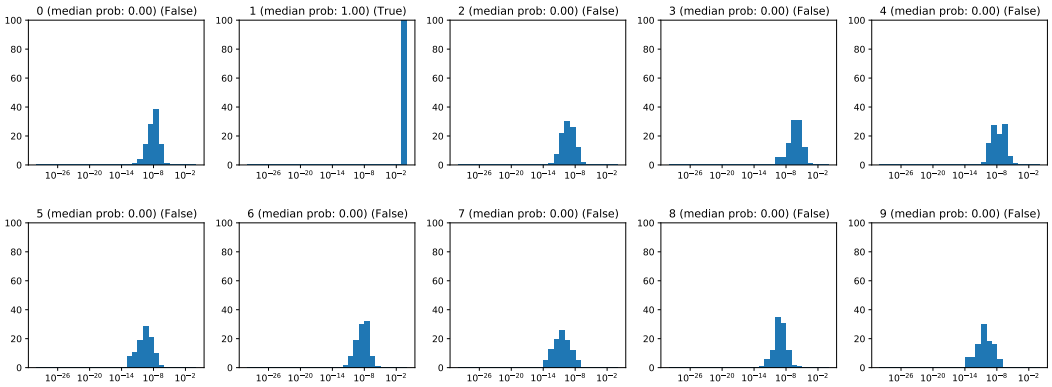


Figure 4: Familiar Example Confidence Estimates

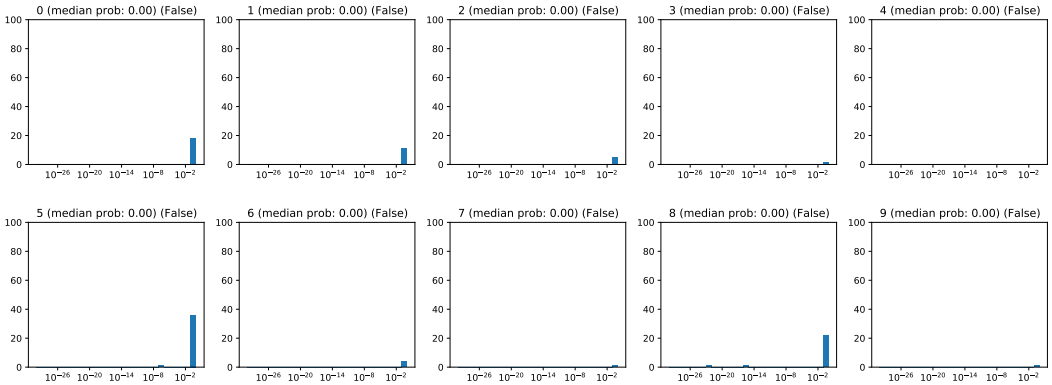


Figure 5: Unfamiliar Example Confidence Estimates

5 Conclusion

We have successfully applied a probabilistic, Bayesian deep learning model, the BCNN, to the fashion MNIST dataset. Our model learned latent features of the products in latent space, allowing us to classify test images with high accuracy. Ultimately, we classified data with performances on par with that of frequentist neural networks with identical architectures, while ensuring that our model can also incorporate a measurement for uncertainties and regularisation. We expected to face more difficulty with gaining insights into the confidence values of the BCNN, but found that the network inherently stores these values along with its weight distributions to enable predictions that meet the confidence threshold set in our experiments. Future work includes implementing pruning methods to reduce the number of parameters in the BCNN. Further extensions include replacing traditional layers of Generative Adversarial Networks (GAN) and Variational Autoencoders (VAE) with BCNNs to reconstruct test images and generate hybrid products as inspiration for fashion designers while observing belief of reconstruction.

Supplemental Materials

Bayesian-CNN.ipynb: Main code that runs our algorithm for model training and visualization. (Jupyter Notebook)

UMAP.ipynb: Code for visualizing Fashion-MNIST data with UMAP (Jupyter Notebook)

References

[1] H. Xiao, K. Rasul, and R. Vollgraf. (2017) Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

[2] L. McInnes, J. Healy, N. Saul, and L. Grossberger, "Umap: Uniform manifold approximation and projection," *The Journal of Open Source Software*, vol. 3, no. 29, p. 861, 2018.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12. Red Hook, NY, USA: Curran Associates Inc., 2012, p. 1097–1105.

[4] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1717–1724.

[5] M. Dialameh, A. Hamzeh, and H. Rahmani, "Dl-reg: A deep learning regularization technique using linear regression," 2020.

[6] K. Shridhar, F. Laumann, and M. Liwicki, "A comprehensive guide to bayesian convolutional neural network with variational inference," 2019.

[7] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks," 2015.

[8] M. Krasser, "Variational inference in bayesian neural networks," Mar 2019. [Online]. Available: http://krasserm.github.io/2019/03/14/bayesian-neural-networks/

[9] Y. Gal and Z. Ghahramani, "Bayesian convolutional neural networks with bernoulli approximate variational inference," 2016.

[10] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational inference: A review for statisticians," *Journal of the American Statistical Association*, vol. 112, no. 518, p. 859–877, Apr 2017. [Online]. Available: http://dx.doi.org/10.1080/01621459.2017.1285773

[11] C. P. Robert and G. Casella, *Monte Carlo Statistical Methods*, 2nd ed. Springer, 2004. [Online]. Available: https://link.springer.com/book/10.1007/978-1-4757-4145-2

[12] M. Xu, M. Quiroz, R. Kohn, and S. A. Sisson, "Variance reduction properties of the reparameterization trick," 2018.

[13] A. D. Kiureghian and O. D. Ditlevsen, "Aleatoric or epistemic? does it matter?" *Structural Safety*, vol. 31, no. 2, pp. 105–112, 2009.

[14] A. Kendall and Y. Gal, "What uncertainties do we need in bayesian deep learning for computer vision?" 2017.