AMATH 582: HOME WORK 5

SUKHJIT KAUR

Department of Mathematics, University of Washington, Seattle, WA
sukhjitk@uw.edu

Abstract. This project explores the classification of images from the FashionMNIST dataset using Fully Connected, and Convolutional Neural Networks (FCN and CNN). The dataset consists of 60,000 training images and 10,000 validation images, with each sample represented by a 28px by 28px greyscale image. This project investigates the classification of FashionMNIST images using Fully Connected Neural Networks (FCN) and Convolutional Neural Networks (CNN). The study compares various model configurations, including variations in architecture, optimizer choice, and regularization techniques. Results show that while FCNs require more epochs to achieve similar performance, CNNs reach a test accuracy of 88% within just 4 epochs. The effectiveness of different hyperparameters and model structures is discussed.

1. Introduction and Overview

The FashionMNIST dataset, an extended version of the original MNIST dataset, contains grayscale images of clothing items, which presents a more complex classification task. This study aims to explore two popular neural network architectures—Fully Connected Neural Networks (FCN) and Convolutional Neural Networks (CNN)—to classify these images. The impact of various hyperparameters, such as the number of hidden layers, learning rate, dropout rate, and weight initialization techniques, on model performance are examined. The objective is to determine the most efficient model configuration for achieving high classification accuracy with minimal computational cost.

The dataset is split - consisting of 60,000 images in a training set used to train the classifier, and 10,000 images in a test set used for validation of the model. The training and test sets, X_{train} and X_{test} are made up of 28px x 28px sized greyscale images. Each pixel in the 28x28 image (784 total pixels) represents a *feature* - this corresponds to the 784 neuron input layer of the model. The clothing type assigned to each image in the training or test set, y_{train}, y_{test} is referred to as a *label* - this corresponds to the output layer of 10 classes.

By adjusting the number of hidden layers, and neurons 100,000 weighs can be achieved in the FCN model. Through varying other parameters like learning rate, dropout rate, and epochs, and by using Cross-entropy loss, the model can be optimized. Including more layers and neurons than necessary increases computational complexity, without much increased benefit. This is important to take into account when designing the FCN model. By iteratively experimenting with hyperparameter tuning, the model can be shaped for reasonably high accuracy within a reasonable training time.

2. Theoretical Background

Classification is achieved using a Fully Connected Neural Network model (FCN) and using Convolutional Neural Networks (CNN). FCNs utilize a fully connected model, in which each neuron is connected to every neuron in the previous layer. Alternatively, CNNs take advantage of spatial hierarchies through convolutional and pooling layers, and shared weights. This architecture greatly reduces the number of parameters, and thus computational complexity.

The FCN models are constructed using artificial neurons, based on *sigmoid neurons*. Each neuron takes an input, x_j, with an associated weight, w_j, and a bias, b, forming the weighted sum $z = x \cdot w + b$. The sigmoid activation function, defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{1}$$

produces an output between 0 and 1, which can be interpreted as a probability. However, due to the limitations of sigmoid activations, such as vanishing gradients, we adopted the ReLU activation function for hidden layers. [2]

Date: March 19, 2025.

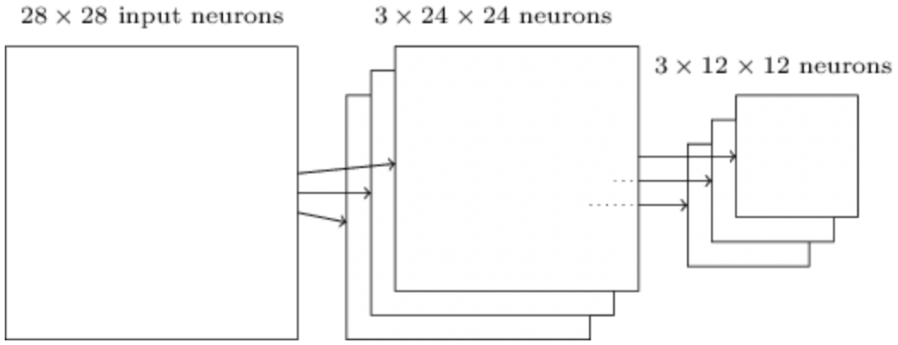

FIGURE 1. The convolutional neural network uses hidden neurons to create feature maps that are then consolidated into pooling layers.
Image credit: Nielsen [2]

CNN models use local receptive fields, shared weights and biases, and pooling to reduce the number of parameters and improve computational efficiency, leading to faster training times. The local receptive field maps the input pixels to a hidden neuron, which learns the weight and bias. The local receptive field is slid across the entire 28x28 input image, connecting to different hidden neurons. This map from the input layer to the hidden layer is called a feature map, and stacking many feature maps allows for image classification,(Figure 1). The concept of shared weights 2 allows the neurons in the hidden layer to detect the same feature of the image, regardless of its orientation, thus increasing computational efficiency. In addition, CNNs use pooling layers to simplify the output from the convolutional layers, condensing the output into a feature map. For example, a region of the previous layer can be max-pooled, where the maximum activation from the region is selected [2].

$$\sigma(b + \Sigma_{l=0}^{4}\Sigma_{m=0}^{4} w_{l,m} a_{j+l,k+m}) \tag{2}$$

The Rectified Linear Unit (ReLU) activation function was chosen for the hidden layers due to its computational efficiency and ability to mitigate the vanishing gradient problem. ReLU introduces non-linearity by outputting the input directly if positive, and zero otherwise:

$$f(z) = \max(0, z) \tag{3}$$

This allows the network to learn complex mappings between inputs and outputs. [2]

Cross-Entropy loss was used as the objective function for training, as it is well-suited for multi-class classification tasks. It measures the difference between the true label distribution and the predicted probability distribution, giving a value for training loss.

$$L = -\sum_{i=1}^{n} y_i \log(\hat{y}_i) \tag{4}$$

where y_i is the true label (1 for correct class, 0 otherwise) and $\hat{y}_i$ is the predicted probability for class i. This loss pushes the model to output probabilities close to 1 for the correct class and 0 for others. [2]

Three optimization algorithms are explored — Stochastic Gradient Descent (SGD), Adam, and RMSProp. Each optimizer uses different strategies:

- **SGD:** Updates weights using gradients computed from randomly selected mini-batches, often combined with momentum to accelerate convergence.
- **Adam:** Combines the benefits of momentum and adaptive learning rates by maintaining first and second moment estimates of gradients, leading to more efficient updates.
- **RMSProp:** Uses an adaptive learning rate that scales with the moving average of squared gradients, preventing vanishing or exploding gradients.

All three methods iteratively adjust weights and biases to minimize the loss function. The update rules for weights w_k and biases b_l in gradient descent are given by:

$$w_k \to w_k' = w_k - \eta \frac{\partial C}{\partial w_k} \tag{5}$$

$$b_l \to b_l' = b_l - \eta \frac{\partial C}{\partial b_l} \tag{6}$$

where η is the learning rate. To reduce computational complexity, Stochastic Gradient Descent (SGD) estimates the gradient using a small random sample (mini-batch) of training data, making updates more efficient.

Iterating over all training data using mini-batches completes one *epoch*, and the process is repeated for a set number of epochs to optimize the model's parameters. This effectively reduces time needed to find the gradient, and subsequently, perform learning [2].

3. Algorithm Implementation and Development

The architecture of the FCN was built with an input layer of size 784, corresponding to the 28x28 pixel image and a 10 neuron output layer, representing the 10 distinct clothing categories. The complexity of the model was adjusted by changing the number and size of the hidden layers, until approximately 100,000 weights were included. ReLU activation was used to introduce non-linearity, thereby enabling the model to capture more complex relationships within the data. Hyperparameters, including learning rate, dropout rate, and optimizer type were varied until a satisfactory performance was achieved. The model was then adjusted to achieve 50,000 and 200,000 weights.

The CNN model, using convolutional layers with shared weights, was implemented with the goal of reducing parameter count and computational complexity. Various weight initialization methods, including Kaiming initialization, were tested to stabilize training. The CNN models were trained on the FashionMNIST training data with 100,000, 50,000, 20,000 and 10,000 weights and test accuracy was compared with the FCN variants.

For training, PyTorch's built-in functions were utilized [1]. CrossEntropy loss was selected as the loss function, given its suitability for multi-class classification problems. Various optimizers — including SGD, Adam, and RMSProp — were implemented to compare their impact on the model's performance. The learning rate and dropout rate were tuned, as they directly influenced the final accuracy of the model. Batch Normalization was integrated to stabilize and accelerate training.

To monitor the model's progress, training loss, validation accuracy, and testing accuracy were recorded at each epoch. Number of weights and time to train were also noted in a .CSV file for each training run. Additional techniques such as dropout regularization were incorporated to mitigate overfitting, while different weight initialization methods — including Random Normal, Xavier Normal, and Kaiming — were explored to assess their influence on model convergence.

4. Computational Results

For model comparison, experiments were started using SGD, but then pivoted quickly to Adam, due to previous results showing Adam optimizer was superior. This optimizer was paired with a learning rate of 0.001 to observe how final accuracy was affected depending on the choice of model architecture. The training loss and validation accuracy were visualized over all epochs, and the test accuracy was also recorded.

The effect of weight initialization was previously explored by testing Random Normal, Xavier Normal, and Kaiming (He) Uniform initializations. Kaiming initialization yielded the most stable training curves and the highest final accuracy, and for this reason was chosen as the initialization throughout this classification.

To achieve the required 88% accuracy on the testing set, hyperparameter tuning was performed. The learning rate was gradually reduced and dropout rates were tested from 0.1 to 0.5, with 0.2 providing the best model performance. The Adam optimizer was used with an initial learning rate of 0.001. Early stopping was employed to prevent unnecessary training epochs after 88% test accuracy was achieved. Batch normalization was also included to provide regularization.

The combination of Adam optimizer with Kaiming initialization, and the inclusion of both dropout and batch normalization produced the most promising results for the FCN model. However, the CNN model with 100,000 weights was able to achieve a test accuracy of $\geq 88\%$ in just 4 epochs, whereas 15 epochs were required by the FCN model with the same constraints (Table 1). This rapid convergence can be attributed to the spatial structure of CNNs, which exploit local image features more effectively than FCNs. Despite the faster training, the CNN model required more computational resources due to the complexity of convolutions and pooling layers (97 seconds to achieve training with the FCN model, versus 147 seconds using CNN).

The FCN model with 50,000 weights underperformed relative to the 100,000 model, with a slight drop in test accuracy to 87.16%. The decrease in accuracy is likely due to the reduced capacity of the model to capture complex patterns. On the other hand, the FCN model with 200,000 weights achieved higher validation and test accuracies, at 90.0% and 89.3%, respectively. However, this model took longer to train and was computationally more expensive. This suggests that increasing the number of parameters beyond a point may not result in significant improvements, highlighting the diminishing returns in performance relative to the added computational cost. (Figures 7, 6)

model	weights	learn. rate	epochs	drop. rate	val. acc. %	test acc. %	training time (s)
FCN	100k	0.001	15	0.2	88.1	88.1	97
FCN	100k	0.001	20	0.2	89.17	88.19	149
FCN	200k	0.001	20	0.2	90.0	89.3	149
FCN	50k	0.001	20	0.2	88.15	87.16	128
CNN	100k	0.001	4	0.2	88.32	88.35	147
CNN	100k	0.001	20	0.2	91.65	91.20	742
CNN	50k	0.001	20	0.2	91.0	89.78	467
CNN	20k	0.001	20	0.2	89.83	88.31	779
CNN	10k	0.001	20	0.2	86.17	85.31	406

TABLE 1. Comparison of various configurations of the neural network model using Adam + Kaiming, showing the effects of dropout rate, weights, and model architecture.

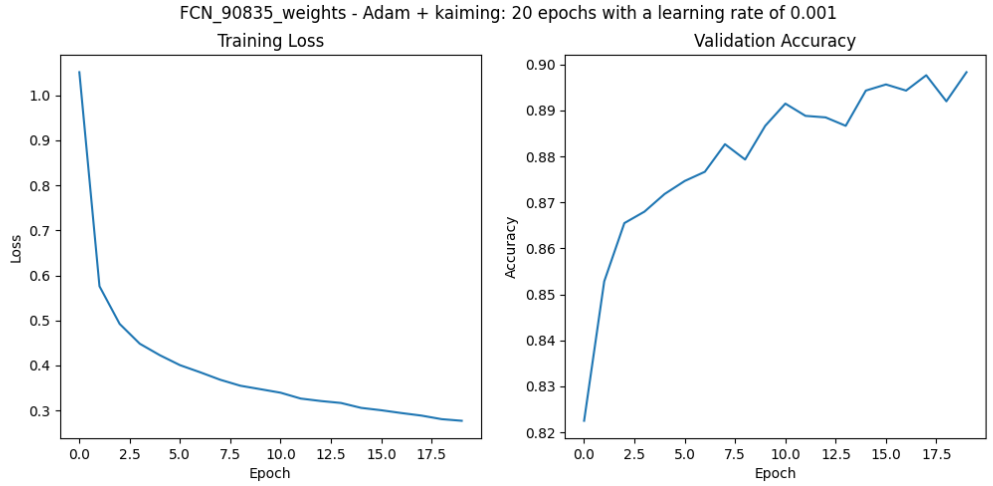

FIGURE 2. FCN model with approximately 100,000 weights was trained with Adam optimizer, Kaiming initialization, learning rate of 0.001, dropout rate of 0.2, achieving 88% test accuracy in 15 epochs.

The CNN model with 50,000 weights showed a minor reduction in test accuracy to 89.78% compared to the 100,000 model, but the training time was significantly reduced (467 seconds compared to 742 seconds). The CNN models with 20,000 and 10,000 weights, demonstrated performance drops - achieving test accuracies of 88.31% and 85.31%, respectively. The 10,000 model was computationally the least expensive but performed poorly, suggesting that further reductions in the number of weights may lead to underfitting. (Figures 4, 5)

5. Summary and Conclusions

A Fully Connected Neural network (FCN) was developed and tuned to classify the FashionMNIST dataset. The model's flexibility allowed for hyperparameter tuning, enabling investigation of the effects of various optimizers, learning rates, weight initializations, and regularization techniques. Accuracy greater than 85% was achieved using specific configurations, and surprisingly, some configurations performed poorly.

To achieve optimal performance, it was important to select appropriate hyperparameters. Adam optimizer consistently outperformed others in terms of convergence speed and final accuracy, while Kaiming initialization proved to be the most effective for stabilizing training with ReLU activation functions. Regularization through Dropout successfully reduced overfitting, and Batch Normalization further improved training efficiency and accuracy. The best performing configuration consisted of the Adam optimizer, a learning rate of 0.01, initialization of the Kaiming weight, and inclusion of Dropout and Batch Normalization. This combination resulted in high validation and test accuracy, showing the model's ability to generalize well to unseen data.

This project provided valuable insights into the design and optimization of Fully Connected and Convolutional Neural networks. The systematic exploration of hyperparameters not only improved model performance but also deepened the understanding of how different components interact during training. These findings can inform future

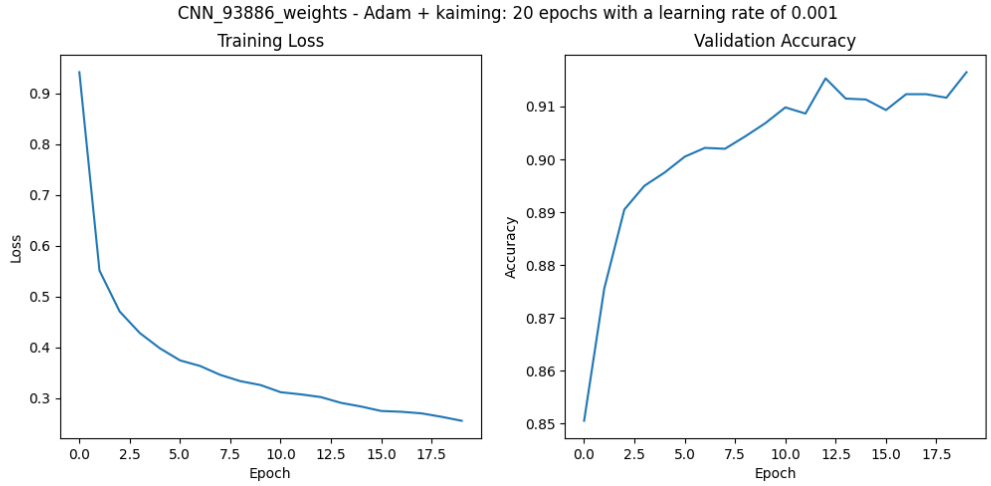

FIGURE 3. CNN model with approximately 100,000 weights was trained with Adam optimizer, Kaiming initialization, learning rate of 0.001, dropout rate of 0.2, accomplishing 88% test accuracy in 4 epochs.

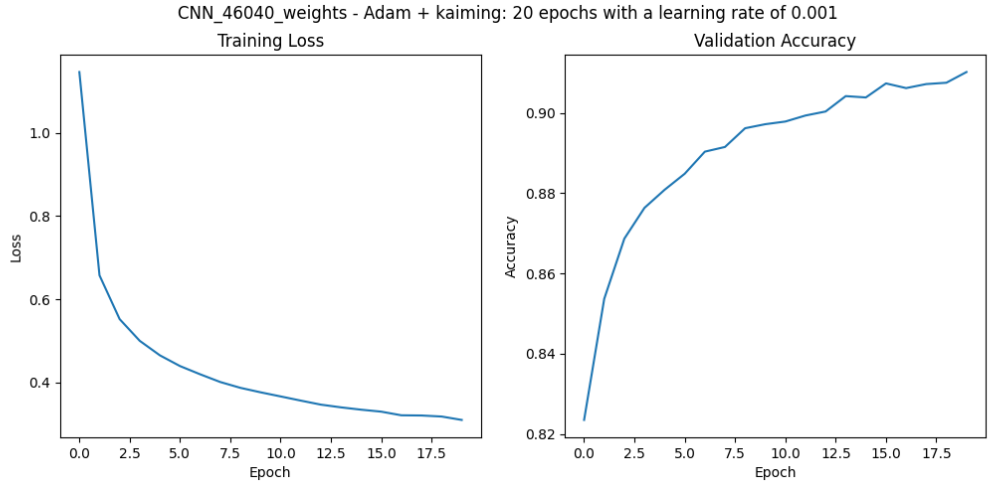

FIGURE 4. CNN model with approximately 50,000 weights.

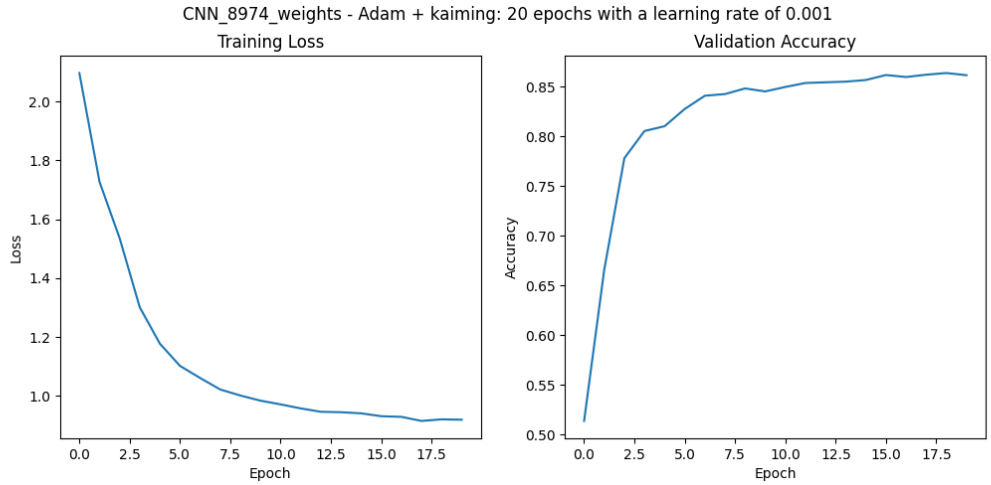

FIGURE 5. CNN model with approximately 10,000 weights.

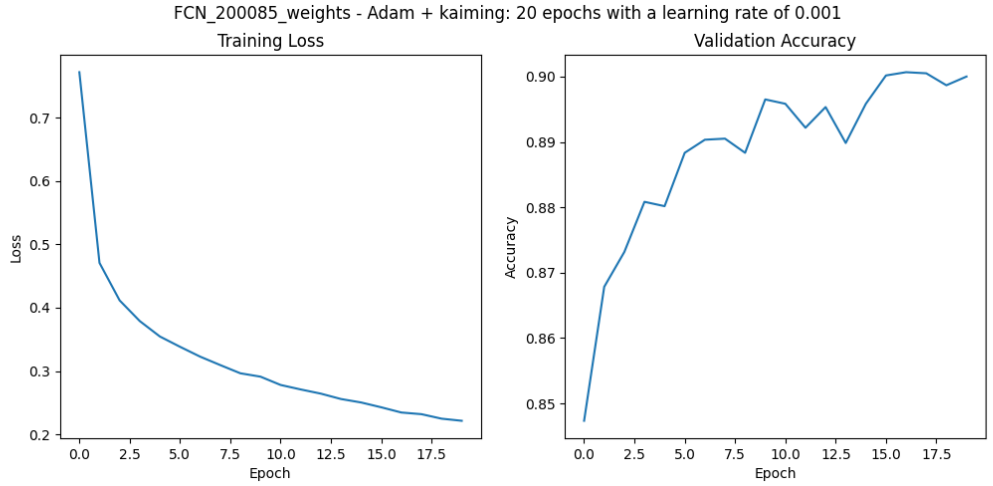

FIGURE 6. FCN model with approximately 200,000 weights.

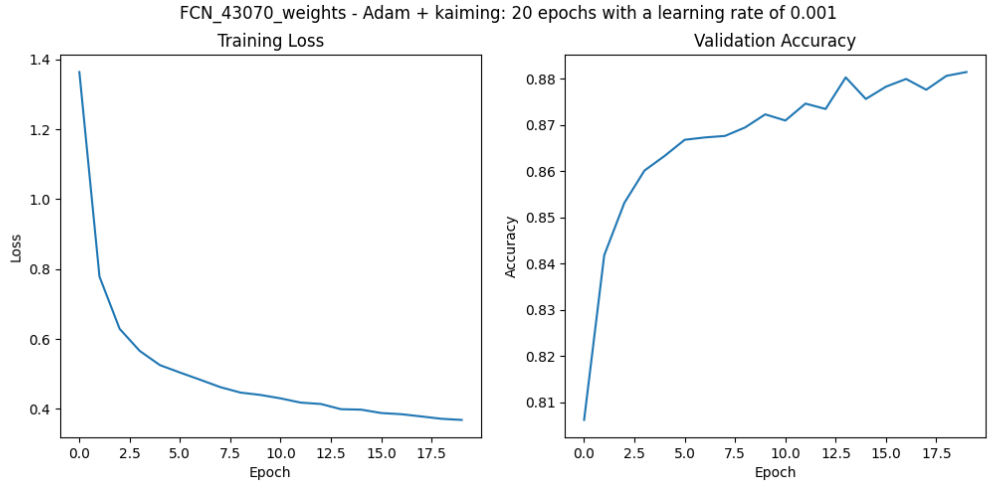

FIGURE 7. FCN model with approximately 50,000 weights.

work on neural network design such as using L2-pooling to optimize a model, or using early-stopping to get proper epochs values. It would also be interesting to build off this project and incorporate code that allows the user to visually verify the model's classification decision.

ACKNOWLEDGEMENTS

The author acknowledges the use of the Modified National Institute of Standards and Technology 'Fashion' dataset. The author also appreciates the use of Large Language Models for code debugging resolution and further explanations regarding neural networks.

REFERENCES

[1] T. L. Foundation. Pytorch documentation.
[2] M. Nielsen. Neural networks and deep learning.