AMATH 582: HOMEWORK 5

KATIE WOOD

Applied Math Department, University of Washington, Seattle, WA
kvo24@uw.edu

Abstract. We train Fully Connected and Convolutional Neural Networks to classify images in the FashionMNIST data set. This data set contains thousands of images of different types of clothing items, which due to their greater visual complexity are more challenging to classify than the images of digits in the MNIST data set. We utilize the PyTorch deep learning platform to build our neural network. We evaluate the performance of FCNs and CNNs and compare the trade-off between computational cost and accuracy for large networks.

1. Introduction and Overview

Compared to Fully Connected Neural Networks, Convolutional Neural Networks allow for some layers to operate on 2D images directly, convolving each image with each of several filters to extract meaningful information. Recently, a computational data analysis class at the University of Washington was built a Fully Connected neural network to correctly classify at least 88% of the images in the FashionMNIST data set (upon testing). The scientists on the project were then tasked with upgrading their FCN to a CNN to attain potentially higher accuracy at a higher computational cost. As before, the scientists divided the FashionMNIST data set into training, validation, and testing sets. They then performed hyper-parameter tuning on their FCN given a budget of 100K weights to achieve at least 88% testing accuracy. From there, they experimented with adjusting the number of weights up and down, to 200K and 50K. Against these FCN variants they then compared several CNN variants, again adjusting the number of weights between 10K, 20K, 50K, and 100K. They compared all variants of both types of networks to understand which gave the highest accuracy and also which was most efficient in terms of computational cost.

2. Theoretical Background

Convolutional neural networks (CNNs) arise out of the need to reduce the huge number of weights associated with Fully Connected neural networks containing many neurons. Convolution offers a different approach to introducing non-linearity into the network: rather than flatten images into vectors, work directly with the images to elucidate salient features. To be specific, in a convolutional layer one convolves small local portions of the input image with each of several filters. The content of these filters are the weights that the neural network learns over the course of training.

The process works as follows. To compute each entry of the feature map $F[m, n]$ (which is input to the next layer), calculate the following sum:

$$F[m,n] = \sum_i \sum_j I[m,n]w[m-i, n-j]$$

where I is the input image and w is the filter kernel. This amounts to the element-wise product of a small local region of the image (equal in size to the filter) with the filter kernel, and then the sum over all elements of this product. This sum then becomes a single entry in the feature map. One point worth noting is that in general this convolution process will reduce the size of the feature map

compared to the input image. This is sometimes desirable and other times not–when it is common practice to pad the resulting feature map with zeroes so as to preserve size. Different filters will detect different meaningful aspects of a given image, such as the edges of an object. The purpose of the CNN is to learn the weights that make up each filter so as to maximize image classification accuracy at the end of the network.

It is important to observe the following constraint for a valid convolution:

$$n^l = n^{l-1} - f + 1 \tag{1}$$

Where n^i denotes the length of one dimension in layer i and f denotes the length of one dimension of the filter. Lastly, to allow the filter to pass more quickly over the input image, we allow for a stride parameter s which tells the filter to skip some number of entries of the input image in between convolutions. This modifies our constraint for a valid convolution as follows:

$$n^l = (n^{l-1} - f)/s + 1$$

In this assignment, we use ReLU activation for all neurons. ReLU's wide usage in image recognition problems makes it an appropriate choice for our FashionMNIST data set [1] [2]:

$$\text{Rectified Linear Unit (ReLU):} \quad f = \max(x, 0)$$
$$\text{Leaky (parametric) ReLU:} \quad f = \max(x, ax) \quad \text{where} \quad a > 0$$

3. Algorithm Implementation and Development

Our computational approach relied primarily on Python's deep learning platform PyTorch. In addition, we utilized NumPy for scientific computing, tqdm to see our training progress bar, and the train_test_split method from Sci-Kit Learn to help divide our data into training, validation, and testing sets. We used Google's GPU hardware accelerator to speed up computation time. Visualization was provided by Matplotlib.

4. Computational Results

We used our hyper-parameter tuning from the previous assignment to start our search for an FCN with up to 100K weights that would deliver testing classification accuracy of over 88%. We found the following parameters to achieve this FCN:

$$\begin{aligned} \text{number of hidden layers} &= 2 \\ \text{neurons per hidden layer} &= 223 \\ \text{learning rate} &= 0.001 \\ \text{number of epochs} &= 40 \\ \text{number of training batches} &= 106 \end{aligned}$$

The FCN 100K model trained in approximately 9.1 minutes using Google's TPU, and achieved a testing accuracy of 89.98% (±1.97%). Below, we present the training loss curve and the validation accuracy curve for the FCN 100K model:

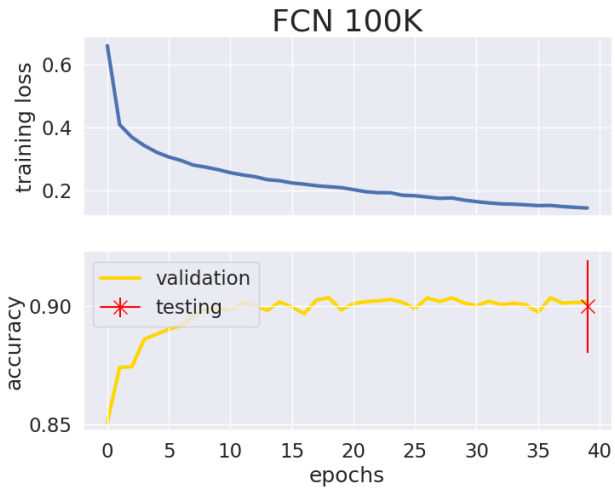

Figure 1. FCN 100K model

We then reduced the number of weights in the FCN model to 50K, obtaining a testing accuracy of 89.44%(±2.0%), and found the following training loss and validation accuracy curves:

Next we increased the number of weights in the FCN model to 200K, obtaining a testing accuracy of 89.72%(±1.8%). One important thing to note about the validation accuracy curve we present below for FCN 200K is that we observe that the validation accuracy by the end of model training exceeds the testing accuracy we were able to obtain. This is a likely indicator of over-fitting in the model, which is plausible given the larger number of neurons in FCN 200K.

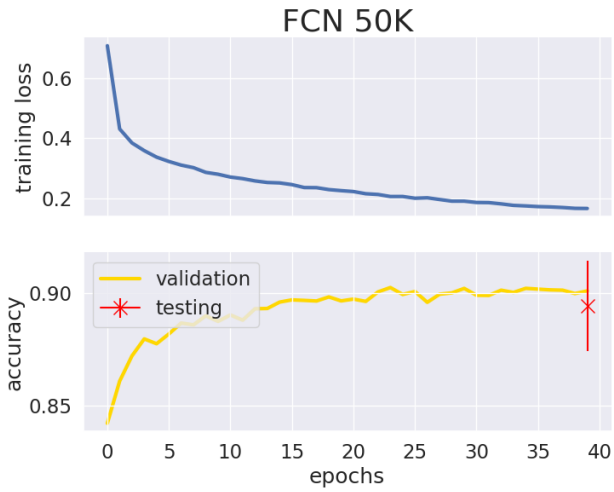

FIGURE 2. FCN 50K model

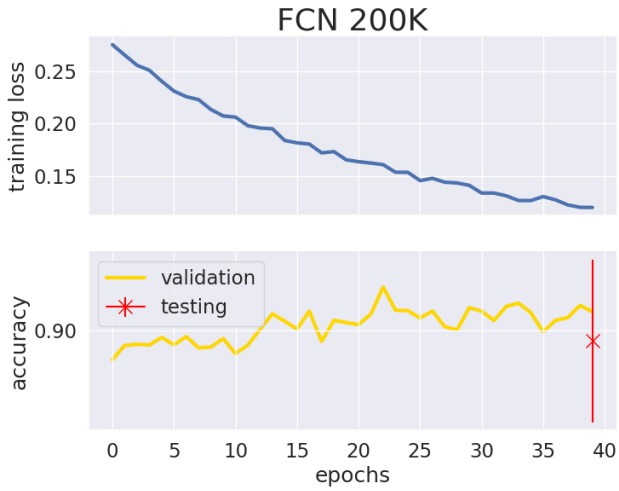

FIGURE 3. FCN 200K model

Having studied these three FCN variants, we then turned to the CNN variants. First, CNN 100K delivered a testing accuracy of 75.55%(±3.3%), and the following training and validation accuracy curves:

Next, CNN 50K delivered a testing accuracy of 76.55%(±3.68%), and the following training and validation curves:

After that, CNN 20K obtained a testing accuracy of 71.39%(±2.58%), a notable decrease from CNN 50K and CNN 100K.

Lastly, CNN 10K delivered a testing accuracy of 70.96%(±3.86%), on par with CNN 20K.

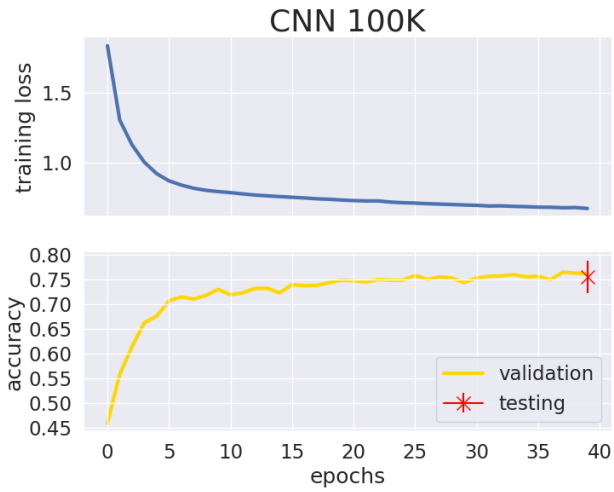

FIGURE 4. CNN 100K model

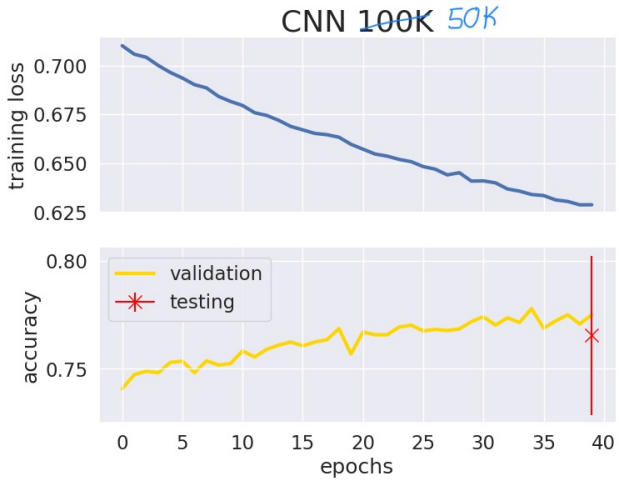

FIGURE 5. CNN 50K model

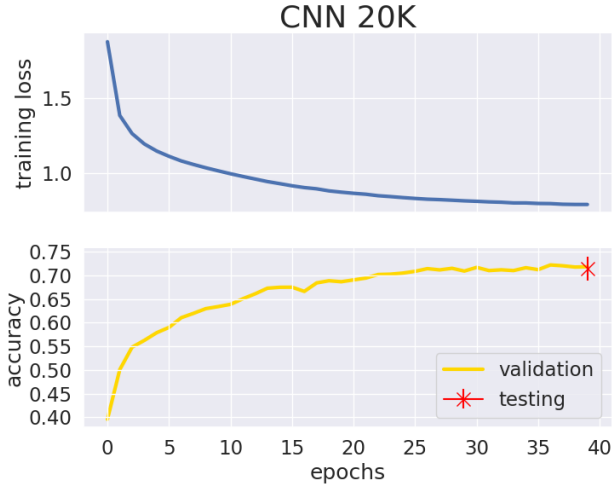

FIGURE 6. CNN 20K model

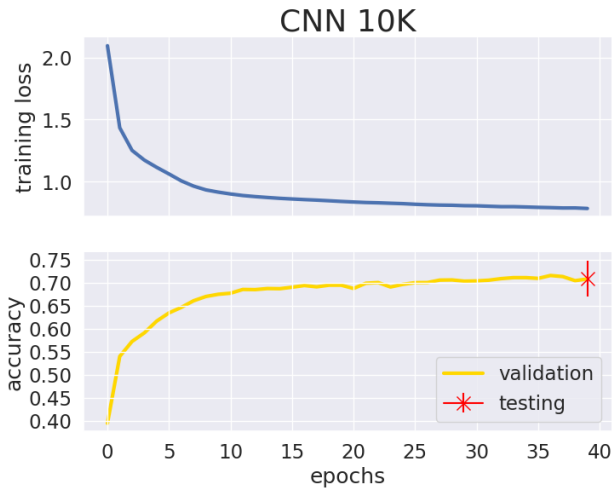

FIGURE 7. CNN 10K model

5. Summary and Conclusions

In conclusion, the highest accuracy we were able to obtain was 89.98%(±1.97%) with the FCN 100K model variant. Our inability to improve the performance of any of the CNNs over the FCNs could be due to a bug in the code, or a sub-optimal design of the CNN. All FCN variants ran in between 9 to 10 minutes, while the CNN variants ran in between 9 to 14 minutes.

In a future project, we could pick one of our CNN variants and visualize feature maps of the convolutional layers for various input samples.

Acknowledgements

The author is thankful to everyone who contributed to the class, including TAs, students, and professors.

References

[1] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[2] M. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.