Student Number: 215709

1. Abstract

This report presents the construction and evaluation of a series of deep neural network models using PyTorch on the Fashion MNIST dataset. The models were designed with varying numbers of hidden layers and explored different hyperparameters, including layer quantity, kernel size, and learning rate. The models were trained and evaluated using cross-validation techniques, and the results were analyzed to determine their performance. The findings indicated that smaller learning rates generally improved accuracy, medium kernel sizes performed optimally, and the impact of layer quantity varied. The best model achieved an accuracy of 95.6% and a loss of 0.00125 during validation. A comparison with ResNet-18 was also conducted, revealing differences in the correlation coefficient matrices generated by the two models. Suggestions for future work include increasing the number of epochs and exploring a wider range of hyperparameters to further optimize model performance.

2. Methodology

For the purpose of the project, I constructed a series of deep neural network models that contained varying amounts of hidden layers using PyTorch [4] to collect the datasets and to build the models.

Within each model, the first and second layers are almost identical. There is a convolutional 2D layer that is followed up with batch normalisation to increase training speed, increase stability and regularise to try reducing overfitting. The final step in each layer is to activate it with ReLU. For most of the models, the size of the filter and kernel size is kept consistent. The exception to this rule is where I test using different kernel sizes to see what effect that has on the evaluation of different models. In every model, the first two layers have max pool 2D function with a kernel size of two and a stride of two. In the models that feature three or more layers, the last two layers are identical and are similar to the two-layer model but instead layers three and four do not feature any max pooling.

2.1. Hyperparameters

The different models that I use have varying hyperparameters to explore how it would affect the outcome. While the values discussed previously stay the same in every model, I change one of three different hyperparameters per model. The first model I chose to change the number of layers per model. I decided that a two-layer model, a three-layer model, and a four-layer model would give me a variety of results that I could analyse.

The next three models change a different hyperparameter while keeping the number of layers to three. This was because it was the middle value between the two other layer quantities. For these three models, I opted to change the size of the kernel. For the first model I chose to use a kernel size of three. The second model I opted to increase the kernel size up to five, and the third model I opted to increase the kernel size by the same amount up to seven.

The final three models kept the same layers as the previous three models but for these models I chose to change the learning rate. The three values I chose started at 0.00025 and had increments of 0.00075, up to 0.001 and 0.00175.

2.2. Training / Evaluating

Before I began to train my models, I wanted to have three different variations on each model so that I could see how each model would perform on average. So I create my 27 models (3 versions of the 3 different model types with the 3 different hyperparameters), trained them and evaluated their performance. To ensure that the models were at their best, I used cross-validation [3] on my dataset so that each epoch used a different collection of images as the validation during the evaluation phase This meant that before training, the data is split into five equal sized sets (for the number of epochs it will run through during training).

For optimisation, I used the Adam optimiser [2] in all twenty-seven models

I normalise the images before they are trained inside each of the models prior to being placed into the input layer of the convolutional neural networks. I calculate the loss of the model to graph later on for comparisons. Once the model has run through all of the layers once, it begins to navigate backwards through the model and based on the loss, the model will be optimised. Each fold from my cross validation that is not being used for evaluating is sent through the training and then the final fold not used in training is used to validate the model. The accuracy of this epoch and the loss from the epoch is recorded to be graphed and analysed later. This happens for five epochs.

Results

Since I had created three different variants of each of my models, I created a bar chart that gave me the average performance of each of my models.

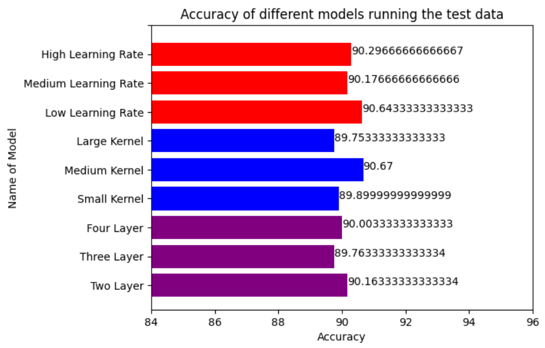


Figure 1: The average accuracy of each model running the test dataset.

Each time the models were trained, the results came out slightly differently, but the patterns were consistent. In most cases, having a smaller learning rate improved the overall accuracy of the model. The smallest learning rate had an accuracy 0.35% better than the next highest.

In regard to kernel sizes, the medium kernel seemed to perform more optimally than the large kernel size or the small kernel size. The difference between a medium kernel and the small / large kernel is about a 1% difference, suggesting that each model can still be very accurate.

The different layer quantities seemed to have varying effects for each time that it was run. In the graph shown in figure 1, it could be said that a two-layer model would outperform a four-layer model. This was not always the case and in most of the other cases, the four-layer model had the highest accuracy of the three but the difference in accuracy was not very large. There is less than a 0.5% change in the best and worst average accuracy.

Overall, the models that I have created have very close values. This could be due to the number of epochs that I chose to use but this was not a hyperparameter I chose to explore for the purpose of this project but could explain the accuracy closeness.
One of the other parameters I chose to not explore further was the batch size. I opted to use a batch size of one hundred for all the models which is just due to optimising training time.

2.3. Best Model

I created a final model to take advantage of all the best hyperparameters that I found while training and evaluating different models. For this best model, I chose to use a four-layer convolutional neural network, a kernel that is 5x5, and opted to train the model with a learning rate of 0.00025.

This best model had an accuracy of 95.6% and a loss of 0.00125 during validation as shown in figure 2 and figure 3.

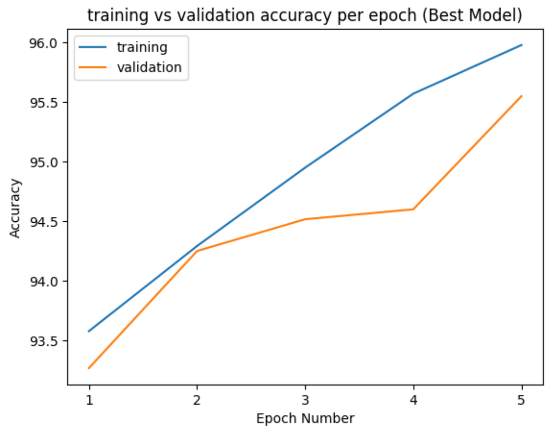


Figure 2: Accuracy per epoch of the best model in training and validation

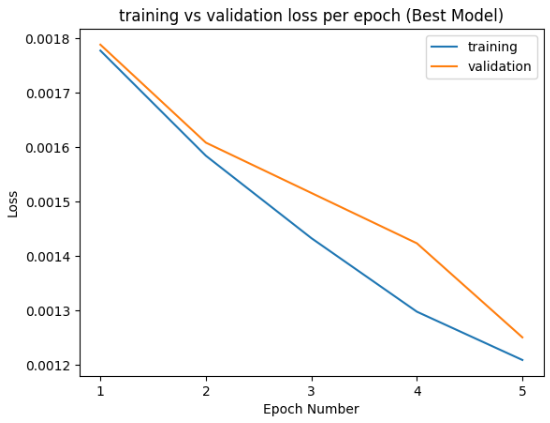
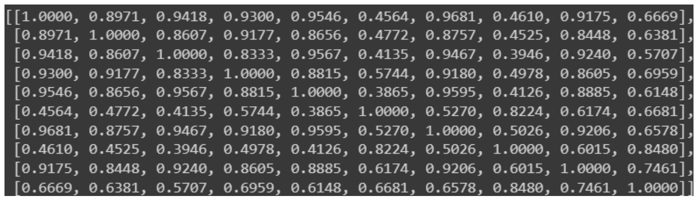


Figure 3: Loss per epoch of the best model in training and validation

The accuracy of this model during the evaluation phase was 90.38% and was ready for comparison against ResNet-18.

3. Discussion

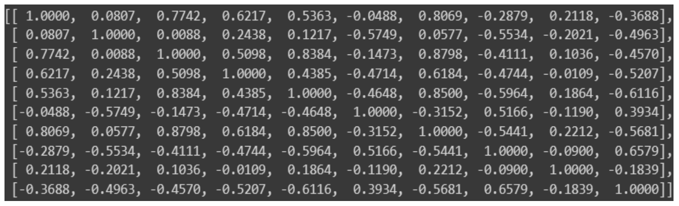
When I created the model for ResNet-18, I needed to create a ResNet-18 [1] loader for the test data and created a tuple of ten arrays that each held 100 images for each of the different clothing classifications. I then collected the average representations of all of the labels, formatting the image to work with ResNet by concatenating the images into three channels as they currently are only greyscale and ResNet requires three channels. One I have the average representations of each classification's average representation, I find the correlation coefficient matrix as shown in figure 4.



```
[[1.0000, 0.8971, 0.9418, 0.9300, 0.9546, 0.4564, 0.9681, 0.4610, 0.9175, 0.6669],
 [0.8971, 1.0000, 0.8607, 0.9177, 0.8656, 0.4772, 0.8757, 0.4525, 0.8448, 0.6381],
 [0.9418, 0.8607, 1.0000, 0.8333, 0.9567, 0.4135, 0.9467, 0.3946, 0.9240, 0.5707],
 [0.9300, 0.9177, 0.8333, 1.0000, 0.8815, 0.5744, 0.9180, 0.4978, 0.8605, 0.6959],
 [0.9546, 0.8656, 0.9567, 0.8815, 1.0000, 0.3865, 0.9595, 0.4126, 0.8885, 0.6148],
 [0.4564, 0.4772, 0.4135, 0.5744, 0.3865, 1.0000, 0.5270, 0.8224, 0.6174, 0.6681],
 [0.9681, 0.8757, 0.9467, 0.9180, 0.9595, 0.5270, 1.0000, 0.5026, 0.9206, 0.6578],
 [0.4610, 0.4525, 0.3946, 0.4978, 0.4126, 0.8224, 0.5026, 1.0000, 0.6015, 0.8480],
 [0.9175, 0.8448, 0.9240, 0.8605, 0.8885, 0.6174, 0.9206, 0.6015, 1.0000, 0.7461],
 [0.6669, 0.6381, 0.5707, 0.6959, 0.6148, 0.6681, 0.6578, 0.8480, 0.7461, 1.0000]]
```

Figure 4: Correlation coefficient matrix generated from the ResNet-18 model.

After finding the correlation coefficient matrix from the ResNet-18 model, I did the same process of finding the average representations of each classification for my best model and generated a correlation coefficient matrix for it, seen in figure 5.



```
[[ 1.0000,  0.0807,  0.7742,  0.6217,  0.5363, -0.0488,  0.8069, -0.2879,  0.2118, -0.3688],
 [ 0.0807,  1.0000,  0.0088,  0.2438,  0.1217, -0.5749,  0.0577, -0.5534, -0.2021, -0.4963],
 [ 0.7742,  0.0088,  1.0000,  0.5098,  0.8384, -0.1473,  0.8798, -0.4111,  0.1036, -0.4570],
 [ 0.6217,  0.2438,  0.5098,  1.0000,  0.4385, -0.4714,  0.6184, -0.4744, -0.0109, -0.5207],
 [ 0.5363,  0.1217,  0.8384,  0.4385,  1.0000, -0.4648,  0.8500, -0.5964,  0.1864, -0.6116],
 [-0.0488, -0.5749, -0.1473, -0.4714, -0.4648,  1.0000, -0.3152,  0.5166, -0.1190,  0.3934],
 [ 0.8069,  0.0577,  0.8798,  0.6184,  0.8500, -0.3152,  1.0000, -0.5441,  0.2212, -0.5681],
 [-0.2879, -0.5534, -0.4111, -0.4744, -0.5964,  0.5166, -0.5441,  1.0000, -0.0900,  0.6579],
 [ 0.2118, -0.2021,  0.1036, -0.0109,  0.1864, -0.1190,  0.2212, -0.0900,  1.0000, -0.1839],
 [-0.3688, -0.4963, -0.4570, -0.5207, -0.6116,  0.3934, -0.5681,  0.6579, -0.1839,  1.0000]]
```

Figure 5: Correlation coefficient matrix generated from my best model.

The matrix that was generated by ResNet-18 did not feature any numbers that went below zero, where my matrix had figures that were as low as -0.6116 and highest of 1.0000 where the categories between the row and columns were the same. While my model met my expectation that the full range of values could be used, it was strange that the ResNet-18 model did not use the full range and only used positive values.

3.1. Final Thoughts

If I had done the project differently next time, I would have used a larger quantity of epochs and explored how this would have affected the model during the training process and how the testing would have changed. This could also mean that the best model I generated for this project and how the comparison with ResNet-18 turned out could be different. I could have also used a larger range of hyperparameters such as using five or seven instead of my chosen quantity of three. I could have also explored how changing the other static hyperparameters in my model could have impacted the learning on the models.

4. References

[1] Kaiming He, X. Z. (2016). Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 770-778). IEE.

[2] Kingma, D. P. (2014). Adam: A Method for Stoachastic Optimization. *arXiv preprint arXiv:1412.6980*.

[3] Liu, Y. (2006). Create Stable Neural Networks by Cross-Validation. *The 2006 IEEE International Joint Conference on Neural Network Proceedings*. Vancouver: IEEE.

[4] The Linux Foundation. (2023). *PyTorch*. Retrieved 05 15, 2023, from PyTorch: https://pytorch.org