Joe Shymanski 5/11/21

Project 5: Classification with Deep Convolutional Neural Networks

Objective

The objective of this project is to complete the provided code for five different Convolutional Neural Network (CNN) models and analyze their performance on the Fashion-MNIST dataset.

Implementation

The first task the provided code carries out is preprocessing the MNIST data, which consists of 60,000 training examples and 10,000 testing examples, each with 10 possible output classifications. Every example in each set is divided by 255, presumably to normalize the inputs between 0 and 1. Then the data is reshaped from a 2-dimensional list with 28 values in each dimension to a 3-dimensional list with 28 values in two dimensions and a single value in the new third dimension.

Next, I had to implement the layers of Model 1A and Model 1B. Both models consisted of a sequential group of layers. The input layer, already provided, was a 2-dimensional convolution (Conv2D) layer, followed by another Conv2D layer. Then a MaxPooling2D layer followed, along with two more Conv2D layers, and then another MaxPooling2D layer. After this came a Flatten layer, which simply flattens its inputs, followed by a Dense layer with 1,024 nodes. Finally came the output layer, another Dense layer with the number of nodes matching the number of classes and the softmax function passed as the activation function. Every layer with an activation function other than the output layer was set to use the ReLU function. All the Conv2D layers set padding equal to “same” and gave a kernel size of (3, 3). Both of the MaxPooling2D layers had a pool size of (2, 2). The only difference between 1A and 1B was the number of filters

on each Conv2D layer. For 1A, the first two Conv2D layers had 16 filters, and the next two had 32. For 1B, the first two Conv2d layers had 64 filters, and the next two had 128.

The next model utilized the concept of Dropout layers. Model 2 looked identical to Model 1B, except a Dropout layer was inserted right after each MaxPooling2D layer. A value of 0.5 was passed as these layers' only argument, meaning 50% of the connections between the MaxPooling2D layers and the subsequent Conv2D layers will be dropped.

Model 3 took advantage of BatchNormalization layers. The code for this model was again copied from Model 1B. Then a BatchNormalization layer was inserted after every Conv2D layer, leaving four in total. These layers benefit the performance and training time of the model, requiring less epochs to train to a high degree of accuracy.

The last model combines all of the concepts from the previous ones and even adds some more layers. The first block of layers follows this sequence: Conv2D, BatchNormalization, Conv2D, BatchNormalization, Conv2D, BatchNormalization, MaxPooling2D, Dropout. These Conv2D layers utilized 64 filters. The next block followed this same format, only the Conv2D layers used 128 filters, instead. A third block was added after that, whose Conv2D layers had 256 filters. Lastly, a Flatten layer, followed by a Dense layer with 1024 nodes, followed by another Dense layer with 512 nodes, preceded the final Dense output layer.

Results & Analysis

Model 1A's performance is displayed in Figure 1. As you can see, the loss of the model exponentially decays towards 0 as the epoch number increases. Conversely, the accuracy of the model on the training data increases logarithmically towards 1. The returns diminish dramatically by the 5^{th} epoch, though they still increase at a gradual and almost constant rate.

Thus, a good number of epochs to train on this data would be anything after 5, depending on how much time you can spend training the model.

Figure 1

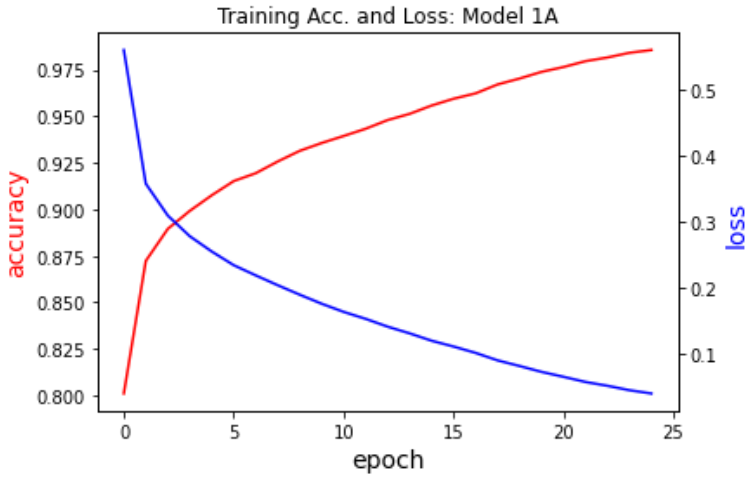
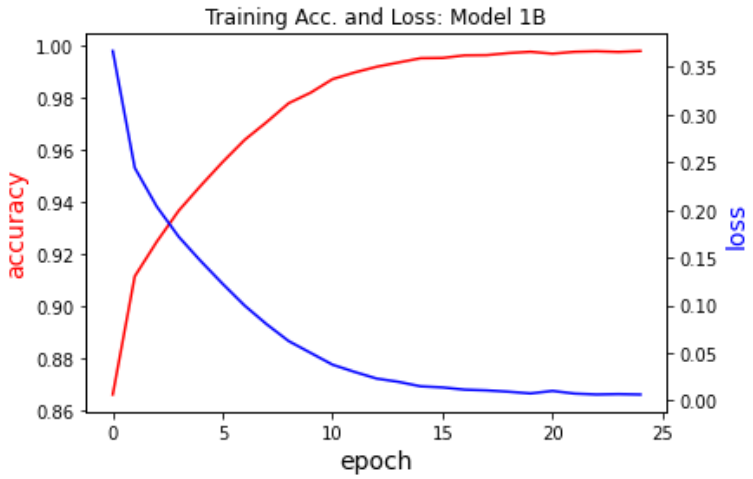


Figure 2 demonstrates the performance of Model 1B. While it took markedly longer to train, it reached a high level of accuracy much more quickly than Model 1A.

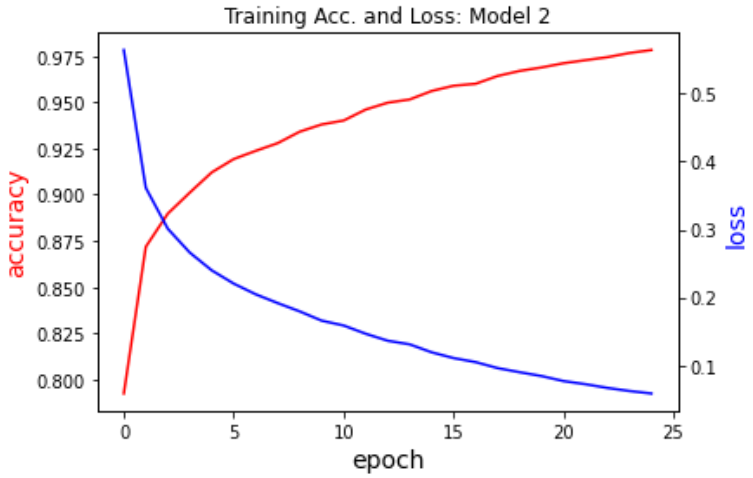
Figure 2



The returns for this model diminish significantly after about the 10th epoch, become nearly flat by the 15th, and actually worsen on the 20th epoch. Thus, anything between 10 and 20 epochs would be a perfectly suitable choice to train Model 1B on this data.

The performance graph for Model 2 is given by Figure 3. It closely resembles the shape attained by performance of Model 1A, though approaching lower loss and higher accuracy a bit more quickly.

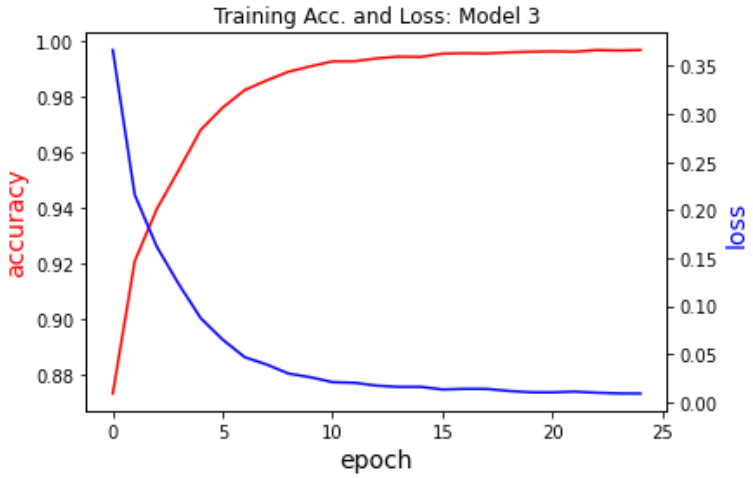
Figure 3



The model's returns diminish by the 5th epoch and remain at a fairly constant upward trajectory after that. Thus, exactly like Model 1A, any choice of epochs after 5 would be appropriate.

Figure 4 shows the performance of Model 3. This model is the top performer on training data by far, approaching near perfection by the 10th epoch.

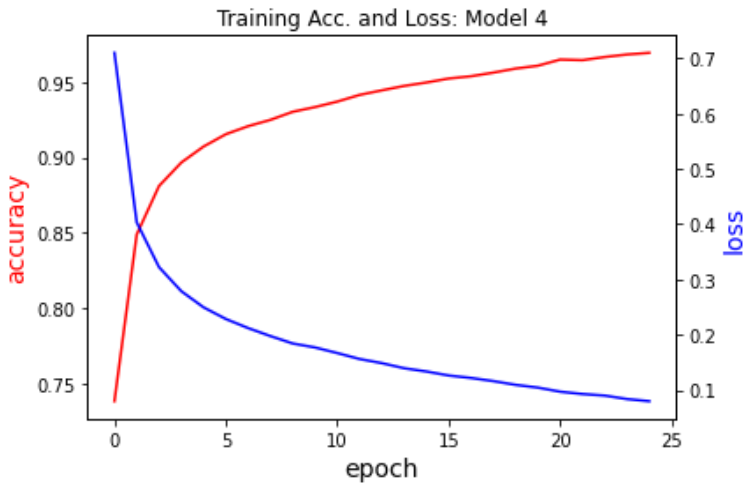
Figure 4



Since the model performs so well but takes a good amount of time to do so, it would smart to stop training around epoch 10, since doubling the work after that only yields a small fraction of a percentage increase in accuracy.

Finally, the last model, Model 4, is given in Figure 5. Given its high complexity and lengthy computation time, its performance was vastly underwhelming.

Figure 5



If you look carefully at the y-axes on the left and right sides of the graph, the model performs terribly at the start, worse than any other model before. Its loss and accuracy follow perhaps the smoothest paths of all the models, but do not converge to 0 or 1 very quickly. It would take many more epochs to reach an accuracy of 0.99. The returns diminish greatly by the 5th epoch, but stopping then wouldn't even yield a desirable prediction accuracy. Thus, this model would be great to use when computation time is not the issue, and instead obtaining a model with incredible accuracy by, say, the 50th epoch is preferred.

The models increased in training time according to the order in which they have been discussed. These training times, along with loss and accuracy values for each model's test data, are given in Table 1.

Table 1

Summary of Training Time and Performance for Each Model

Model	Training Time (s)	Loss	Accuracy
1A	185.64	0.3146	0.9214
1B	329.58	0.4776	0.9264
2	335.52	0.2288	0.9378
3	393.94	0.4851	0.9249
4	625.47	0.2178	0.9371

Note. 25 epochs used for each model.

Models 2 and 4 are the best performers on testing data, though Model 2 takes about half the time to train. Model 3 performed the worst given its high training time, terrible loss value, and mediocre prediction accuracy. This is important to note, given its incredible performance on the test data. Thus, it did not produce a generalized model and vastly overfit the data, more so than any of the other models.

Conclusion

The CNNs produced in this project put on varied performances when making predictions using the Fashion-MNIST dataset. However, compared to the basic ANNs created in Project 4, these CNNs performed exceptionally well and made classifications with incredible accuracy. It appears that the models with the best performance times and accuracies on the training data produced the worst predictions on the testing data. They approached their final models too quickly and settled on ones which did not generalize to the test data. Those that appeared to underperform in the training stage actually performed much better on the test data than the other models. Thus, it's important to utilize BatchNormalization layers, which improve convergence time, alongside Dropout layers, which improve generalizability, for the best results. More layers inevitably produce more reliable models, though they add a significant amount of time to the model's training.