Project 5

Jordan White

May 13th 2021

1 Part A

1.1 Overview

In Part A, I created two convolutional neural networks (CNNs) with slightly different structures. They both consisted of the following layer structure:

2D convolution layer
2D convolution layer
*(Max pooling operation for 2D spatial data)
2D convolution layer
2D convolution layer
*(Max pooling operation for 2D spatial data)
*(Input flatten operation)
Dense layer
Dense layer

The difference between the two neural networks is that in the first network, Model 1A, the first two convolution layers have 16 filters, and the last two have 32 filters. Meanwhile, in the second network, Model 1B, the first two convolution networks have 64 filters and the last two have 128 filters.

I will compare the performance differences between these two networks in the next section in order to make conclusions about the effect of adding filters to the network layers.

1.2 Results

Figure 1 shows that the accuracy of model 1A is 92.15% after 15 epochs. I chose the value of 15 epochs because that is right before the accuracy improvements started to noticeably taper off. Training took about 175 seconds in total.

Figure 2 shows that the accuracy of model 1B is 93.25% after 15 epochs. Training took about 450 seconds in total.

Model 1A was significantly faster to train than 1B, over 150% faster. However, Model 1B was over 1% more accurate, a nontrivial difference especially after 15 epochs. Both of these results make sense in light of the fact that model 1B had many more filters than model 1A. The greater training cost of Model 1B compared to 1A is due to the greater amount of calculation necessary to apply all of its extra filters to the inputs. The improved accuracy of Model 1B over 1A is due to its superior ability to recognize features in the input, thanks to its extra filters.

1.3 Reference

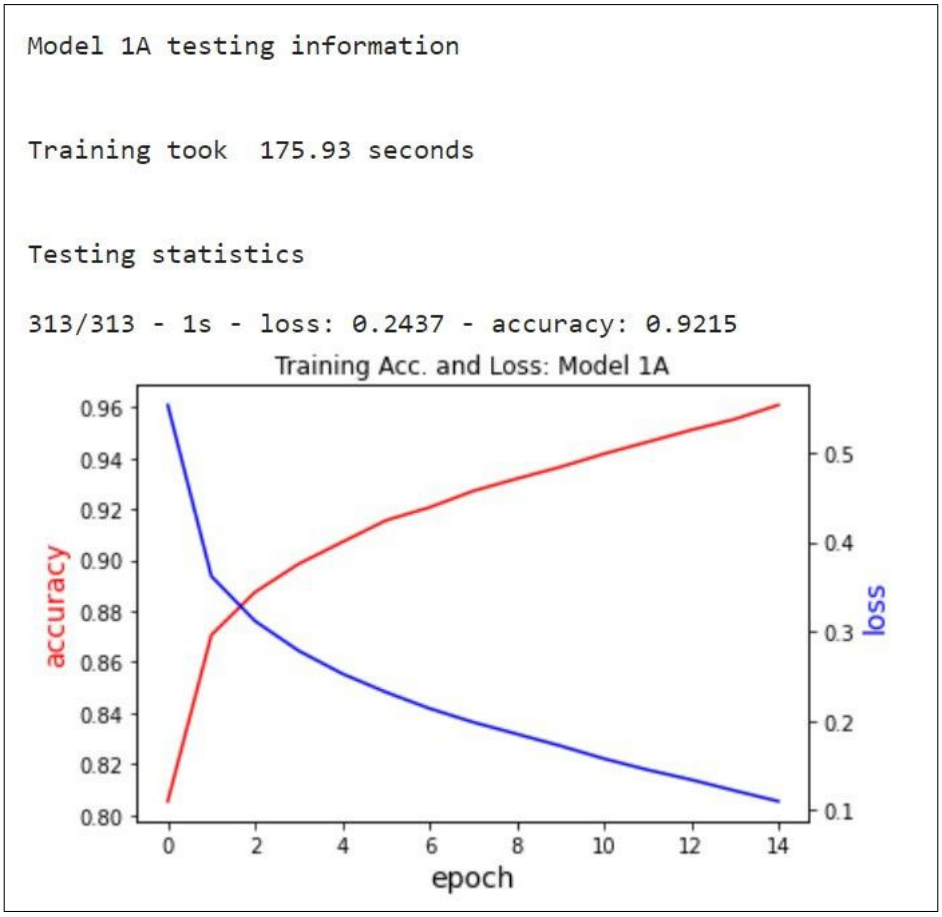

Figure 1: Performance of model 1A.

```
Model 1B testing information


Training took  450.07 seconds


Testing statistics

313/313 - 2s - loss: 0.3332 - accuracy: 0.9325
```

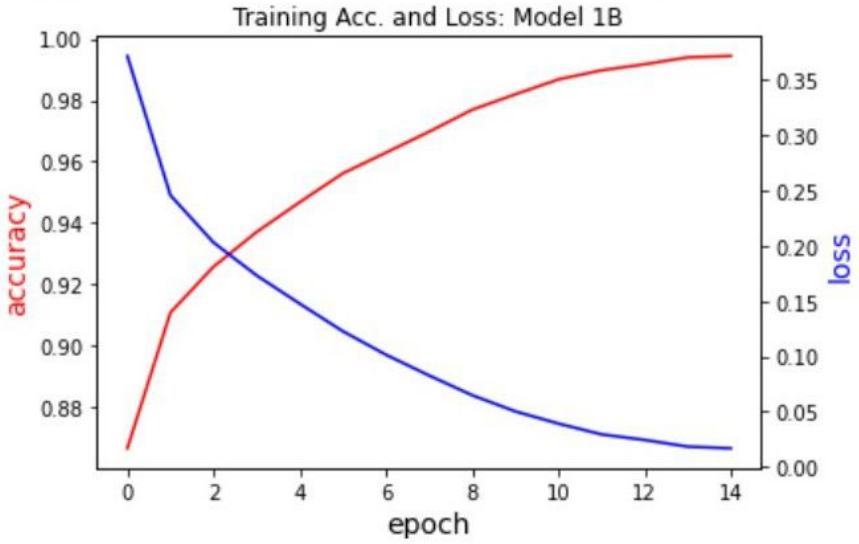

Figure 2: Performance of model 1B.

2 Part B

2.1 Overview

In this part of the project, I added two dropout layers to the network, one after each max-pooling layer. The new layer structure is as shown:

2D convolution layer
2D convolution layer
*(Max pooling operation for 2D spatial data)
Dropout(50% dropout rate)
2D convolution layer
2D convolution layer
*(Max pooling operation for 2D spatial data)
Dropout(50% dropout rate)
*(Input flatten operation)
Dense layer
Dense layer

The goal of adding these dropout layers is to improve the generalizability of the network by probabilistically dropping out random nodes. This can help prevent over-fitting the data when there is few data points to consider. I added these dropout layers to the model 1B from Part A, so I will compare the performance of the dropout network to the network shown in Figure 2.

2.2 Results

As you can see in Figure 3, the new neural network with two dropout layers performed at an accuracy of 93.54% after training for 15 epochs. The training time took about 462 seconds.

Comparing this with model 1B from Figure 2, the accuracy seems to have improved a slight amount ($\tilde{0}.29\%$), but the loss decreased significantly (by 44.4%). For this reason we can conclude that loss does improve accuracy and decrease loss in some cases.

2.3 Reference

```
Model 2 testing information


Training took  462.01 seconds


Testing statistics

313/313 - 2s - loss: 0.1853 - accuracy: 0.9354
```

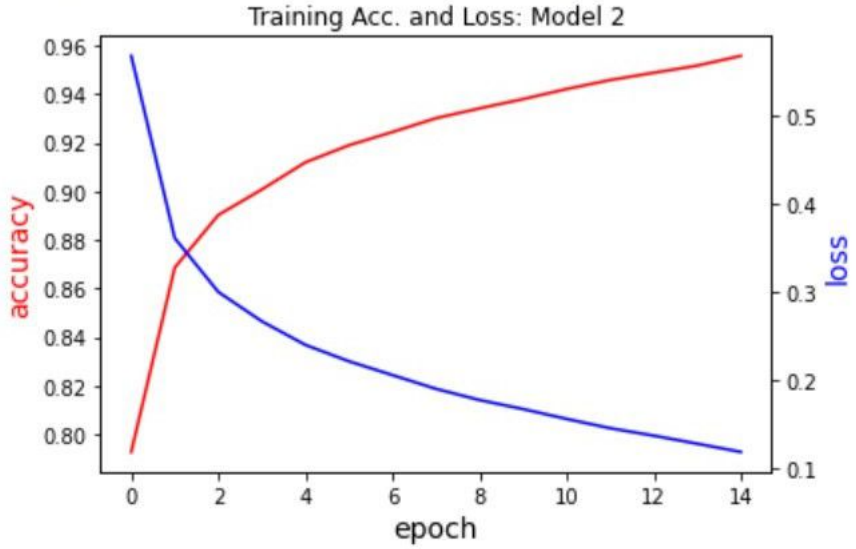

Figure 3: Performance of model with dropout layers.

3 Part C

3.1 Overview

In Part C, I experimented with adding four batch-normalization layers to the model 1B, each layer after one of the 2D convolution layer. The new layer structure is as shown:

2D convolution layer
Batch Normalization layer
2D convolution layer
Batch Normalization layer
*(Max pooling operation for 2D spatial data)
2D convolution layer
Batch Normalization layer
2D convolution layer
Batch Normalization layer
*(Max pooling operation for 2D spatial data)
*(Input flatten operation)
Dense layer
Dense layer

The purpose of batch normalization is to improve generalization and decrease training time. It does this by normalizing the data to have a mean of zero and a standard deviation of one, and then rescaling and shifting the data by some optimal amount.

I once again altered model 1B so I had a benchmark with which to compare the performance of the network with batch normalization.

3.2 Results

Figure 4 shows the accuracy of the neural network with batch normalization, after 15 epochs, to be 93.2%, and the training time to be about 506 seconds. The loss is 0.3751.

Comparing this with model 1B from Figure 2, the performance of the network with batch normalization seems to be slightly worse in every way. However, I ran another test with only 5 epochs instead of 15, as shown in Figure 5. As you can see, the network is trained to a comparable accuracy (91.82%) in less than a fourth of the time (about 104 seconds). This is an substantial time improvement, even if it comes with a slight accuracy trade-off.

3.3 Reference

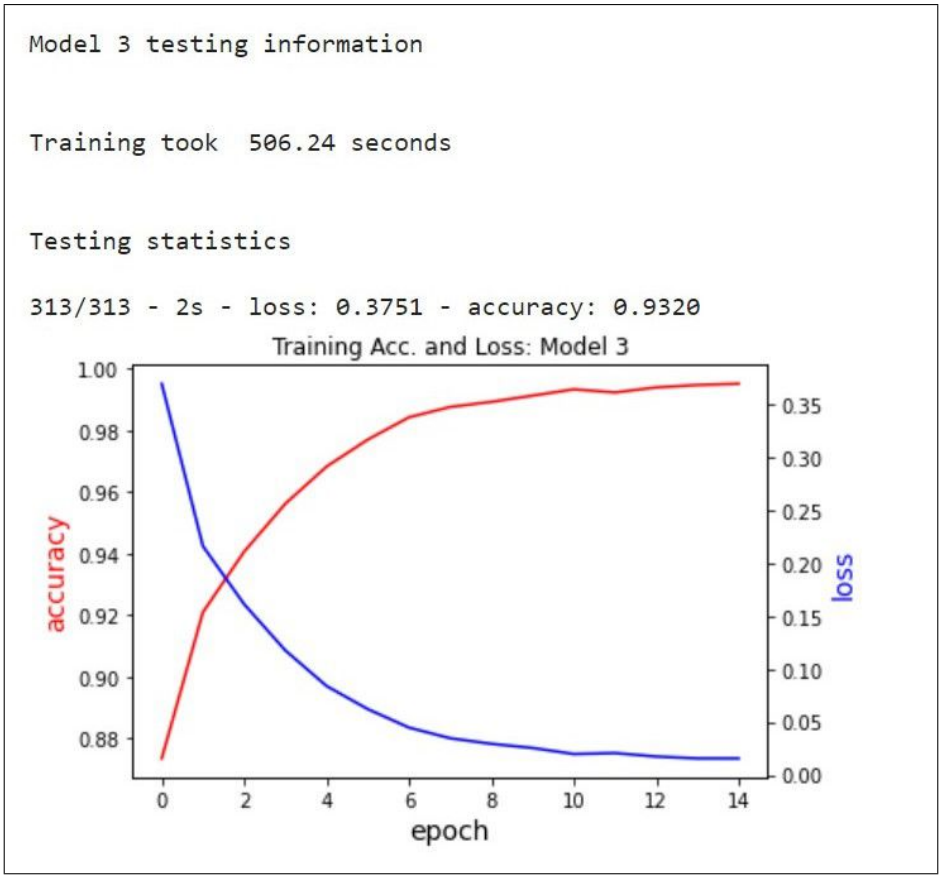

Figure 4: Performance of network with batch normalization after 15 epochs.

```
Model 3 testing information


Training took  104.37 seconds


Testing statistics

313/313 - 1s - loss: 0.2532 - accuracy: 0.9182
```

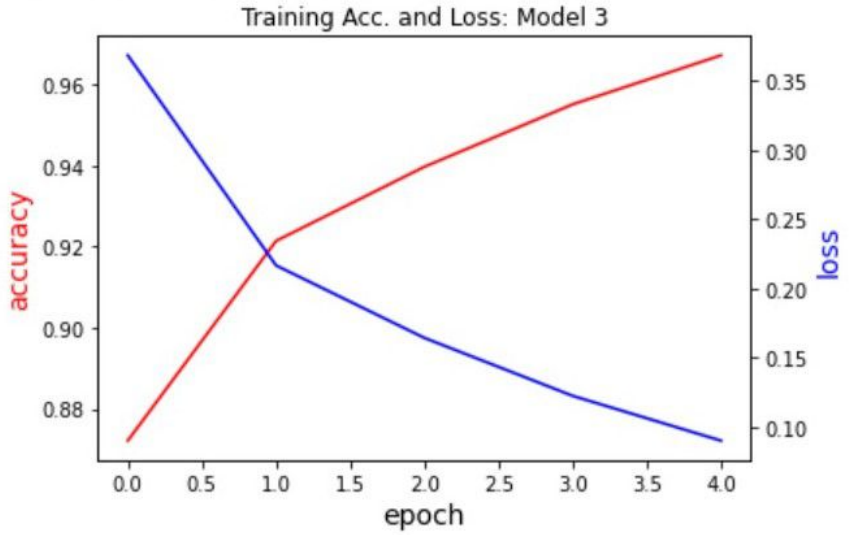

Figure 5: Performance of network with batch normalization after 5 epochs.

4 Part D

4.1 Overview

For Part D, I created a larger neural network with three main blocks of layers. Each block contained three 2D convolution layers, three Batch Normalization layers, a max-pooling layer, and a dropout layer. Afterwards, I added a flatten layer and three dense layers. The structure is shown below:

2D convolution layer
Batch Normalization layer
2D convolution layer
Batch Normalization layer
2D convolution layer
Batch Normalization layer
*(Max pooling operation for 2D spatial data)
Dropout(50% rate)
2D convolution layer
Batch Normalization layer
2D convolution layer
Batch Normalization layer
2D convolution layer
Batch Normalization layer
*(Max pooling operation for 2D spatial data)
Dropout(50% rate)
2D convolution layer
Batch Normalization layer
2D convolution layer
Batch Normalization layer
2D convolution layer
Batch Normalization layer
*(Max pooling operation for 2D spatial data)
Dropout(50% rate)
*(Input flatten operation)
Dense layer
Dense layer
Dense layer

The goal of this test is to see what the effect is of number of layers on training time. The results are outlined below.

4.2 Results

Since this is a larger network, I allowed 25 epochs for training. I also experimented with three different learning rates: 0.0001%, 0.0005%, and 0.0025%.

In Figure 6, you can see the accuracy of the model with a learning rate of 0.0001% was 93.84%. This model took about 1594 seconds to train.

In Figure 7, you can see the accuracy of the model with a learning rate of 0.0005% was 93.89%. This model took about 1600 seconds to train.

In Figure 8, you can see the accuracy of the model with a learning rate of 0.0025% was 93.29%. This model took about 1594 seconds to train.

All in all, it seems the training time was considerably longer than that of the other networks, surely due to the far greater number of layers in the network. The execution time seems to rise more-or-less linearly with the number of nodes/layers in the network.

It also seems that varying the learning rate from between 0.0001% to 0.0025% had little to no effect on the performance of the network, at least in this case.

4.3 Reference

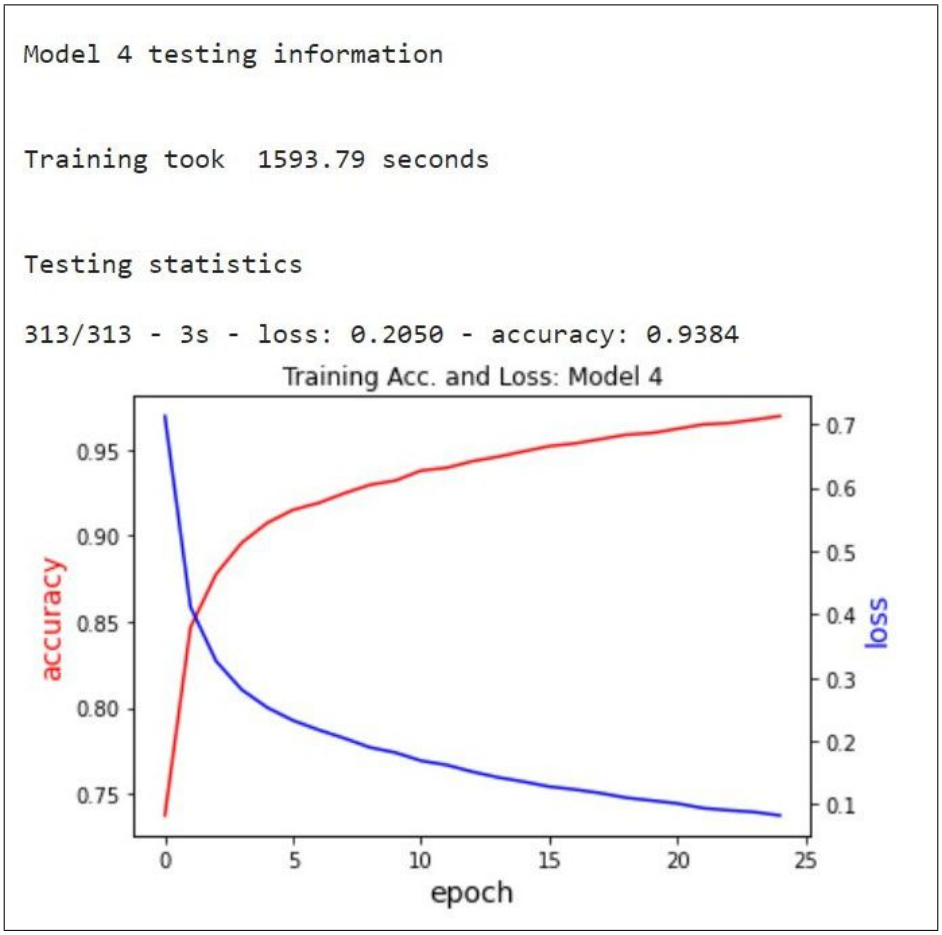

Figure 6

```
Model 4 testing information


Training took  1600.91 seconds


Testing statistics

313/313 - 3s - loss: 0.2268 - accuracy: 0.9389
```

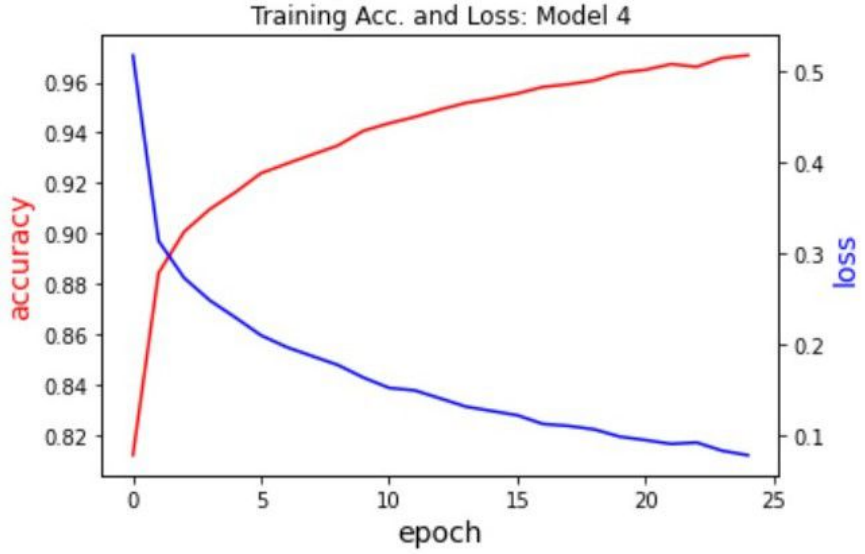

Figure 7

```
Training took  1594.97 seconds


Testing statistics

313/313 - 3s - loss: 0.2289 - accuracy: 0.9329
```

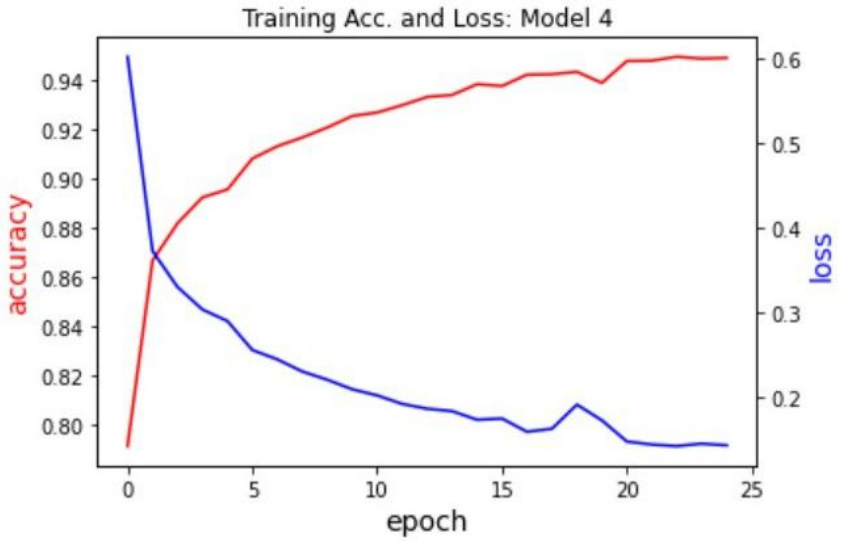

Figure 8