Machine Learning

Coursework

By Sehra Elahi

Q1(a)

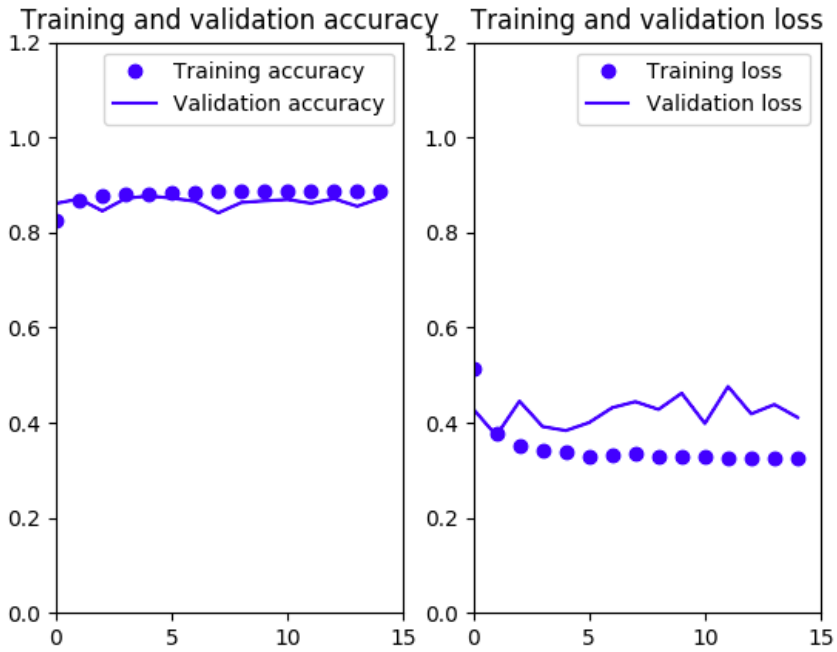


Figure 1 – The plot of the accuracy and loss over time

The plot produced above in Figure 1 shows the loss over time (on the right) for training and validation data. The validation data over time does not seem to be converging, in fact the validation set is giving a greater error loss than training data over time, which indicates that even though training is being done, the model is becoming too specialised on solving for the training data, and doesn't perform well on the validated data. This could be due to the model memorising answers from the training dataset and not generalising well when exposed to new unseen data. While the training loss is gradually decreasing, validation seem to be steady/increasing. Hence the neural network is (slightly) overfitting in this instance. To further confirm that the network is overfitting, the graph on the left for accuracy displays that the validation accuracy is slightly lower than training accuracy, for each instance of the epoch (over time).

Q1(b)

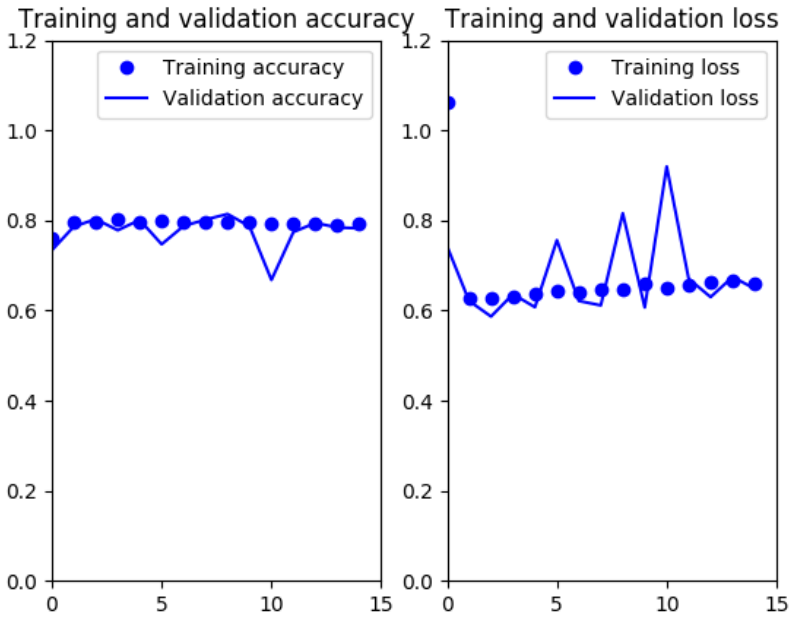


Figure 2: Learning rate set too high: 0.1

Figure 2 displays the case where the learning rate is set too high at 0.1. The loss for validation data tends to oscillate between a higher and lower error loss over the number of epochs, indicating that the patterns are being learned too quick (due to the larger learning rate) and old patterns are being forgotten, hence the zigzag shape of the validation loss. The larger learning rate doesn't improve for the training loss either, with the error loss increases gradually, indicating that its diverging away from the solution/preferred loss value.

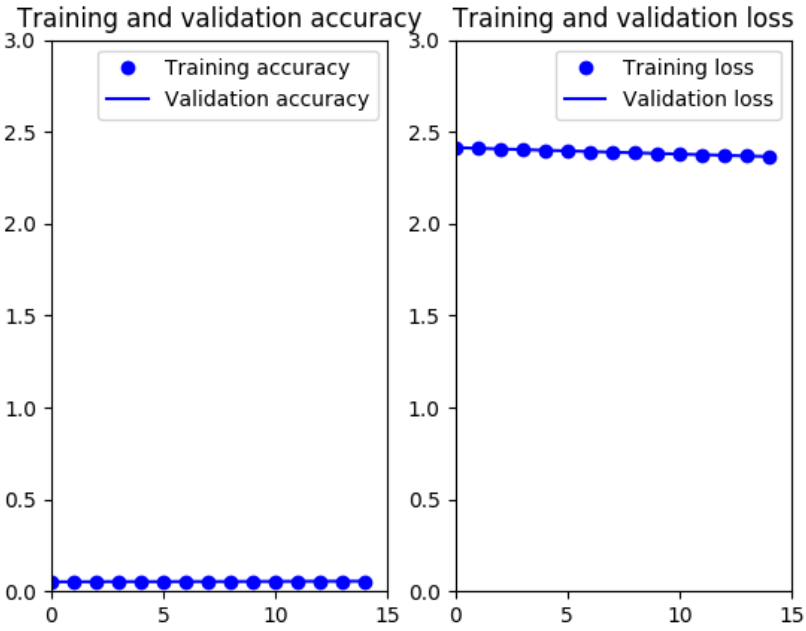


Figure 3: Learning rate set too low: 0.0000001

Figure 3 displays the case where the learning rate is set too low at 0.0000001. The error loss is greater and the steps are too small to converge for both training and validation loss. This indicates that a lot of time (epochs) will be required to reach down to an optimum solution/preferred error loss value, or perhaps it will never converge down lower due to getting stuck on a higher error value, all due to the small learning rate.

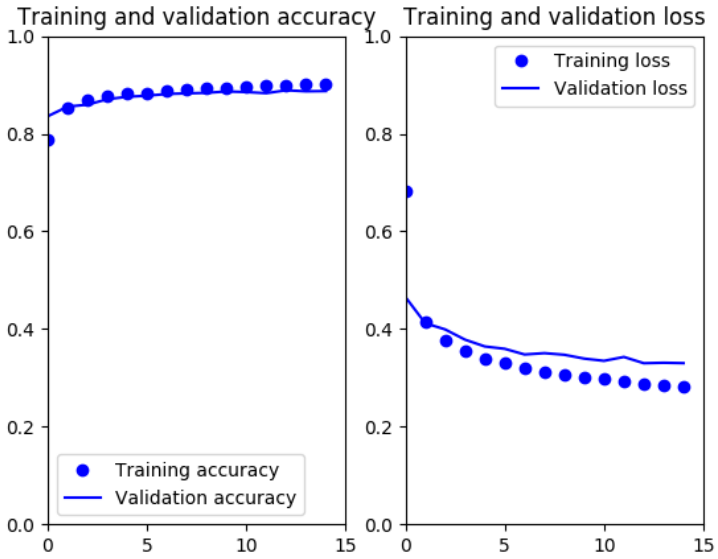


Figure 4: Appropriate Learning rate: 0.001

The learning rate is small enough allowing for the network to converge for both training and validation loss to a preferred lower error loss but still high enough, as it doesn't require an exceedinlgy long time to reach an optimum error

Q2(a)

The learning rate found to be most appropriate from the previous question is used.

```
if True:
    #----------------------------Hyper Parameters-----------------------------------
    batch_size  = 128   # how many images with their corresponding categories to use per
    # per NN weights update step
    epochs      = 15    # how many times to loop over
    # example: for a batch_size=64 and training datas
    # then each epoch will consist of 48000/64=750 up
    learning_rate = 0.001

    model = Sequential()

    #-------------------------------------Architecture-----------------------------------
    model.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=(28,28,1),padding='same'))

    model.add(MaxPooling2D((2, 2),padding='same'))

    model.add(Flatten())
    #-----------------------------------------------------------------------------------
    model.add(Dense(100, activation='relu'))

    model.add(Dense(nClasses, activation='softmax'))

    model.summary()
```

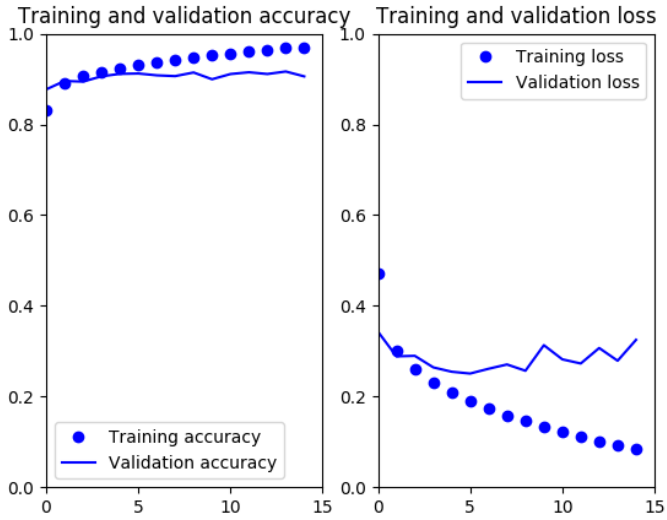
The number of filters is increased from 6 to 32 for the CNN layer (Convo2D) and the dimensions are kept at 3x3.

A dense layer is added with 100 neurons.

Figure 5: Architecture design to achieve training loss < 0.28

The learning rate was set to an appropriate value (0.001) based on previous experimentation for finding the appropriate value for it. The convolutional filters were increased to 32 to capture the data better and detect a greater number of abstractions and patterns in the clothing images. The

dimension of the matrix is passed to maxpooling, and a dense layer of 100 neurons, to classify the patterns obtained and interpret them. Although this allows for the training loss to reach a value well below 0.28 (0.08 < 0.28, see Figure 6), it causes an increase in validation loss; therefore resulting in overfitting. This is problematic and will be addressed in the next part of the question.




```
Training time = 457.29573583602905
Test loss: 0.32753762000203135
Test accuracy: 0.9046
train loss  [0.4711414589484533, 0.30169025413195294, 0.25995531328519184, 0.23104636297623315,
0.2096438065568606, 0.19024166133006415, 0.17402622077862423, 0.15818603444099427, 0.14532939781
745274, 0.13224660245577494, 0.12284486331542332, 0.11153680518269539, 0.10185096559921901, 0.09
255616270005702, 0.08546467526753744]
```

Figure 6: The training loss value achieved (0.08) from architecture developed (Figure 5)

Q2(b)

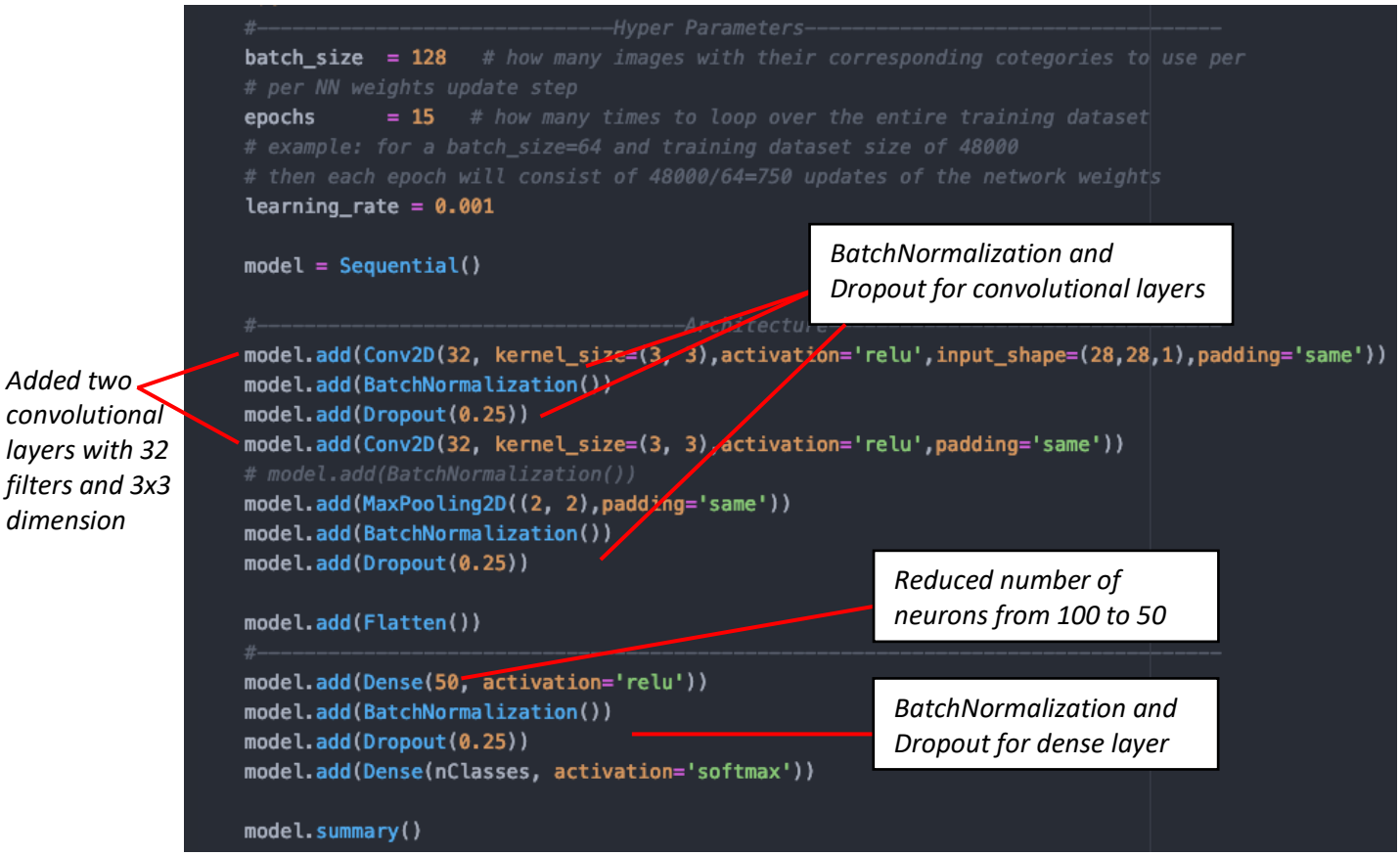
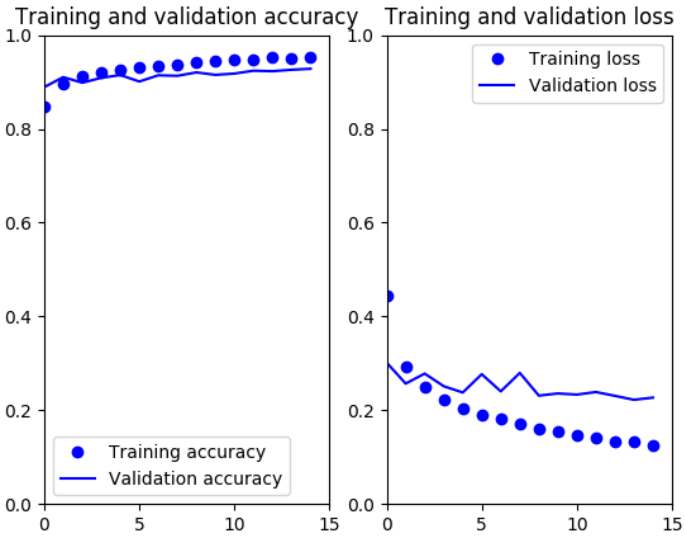


Figure 7: Architecture design to achieve validation loss < 0.27

Two convolutional layers were added to the architecture design of 32 filters each with 3 by 3 dimensions (Figure 7). The reason for adding another layer was due to the network being able to learn more filters for deeper layers. This allows for greater and more sophisticated patterns to be detected and improving overall generalisation of data. It would be better to increase filter size as we move into the deeper convolutional layers, but for our architecture it was kept at 32 filters for each, as increasing the filter (ie 64) resulted in a very slow training process for our personal machine (CPU Macbook). Regardless, maxpooling was added to the last convolutional layer, to downsize the dimensions of the network accordingly. Besides this, the neurons in the dense layer were decreased to 50 from 100. The solution from Q2a (Figure 6) was causing large overfitting of validation data, due to the information obtained from convolutional layer was not enough to train such a large number of neurons. By adding 2 layers, we have greater information, and by downscaling the number of neurons, we can make sure to train them all.

Furthermore, dropout and batch normalisation was applied to reduce overfitting experienced in previous question for validation loss. Batch normalisation is another layer that helps in generalising data better by normalising the training values. Dropout layers were added (0.25 ~ 25%), and contributed to improving the overfitting of validation. The dropout layers helped by switching off some units in the network to allow other units to train better and thereby preventing them to adapt to the data (overfitting) and improve layers to learn the fashion images better.

This allowed for the validation data to classify the clothing items with a significantly lower loss of 0.22, without increasing the training loss either. (Figure 8).




```
valid_loss  [0.30153216914335884, 0.2567805967330933, 0.2781049116055171, 0.2512397635380427, 0.
2377400920788447, 0.2768435111045837, 0.24024785419305167, 0.27957362786928813, 0.23115043946107
228, 0.23573559804757435, 0.2333199036916097, 0.23881709945201873, 0.2306908539533615, 0.2222692
3712094626, 0.22686137807369233]
```

Figure 8: The validation loss value achieved (0.22) from architecture developed (Figure 7)

Q2(c)

```
Test loss: 0.32753762000203135
Test accuracy: 0.9046
```

Figure 9: Test data results from architecture developed in Q2(a)

```
Test loss: 0.2391810121655464
Test accuracy: 0.9223
```

Figure 10: Test data results from architecture developed in Q2(b)

The architecture developed in Q2(b) (see Figure 10)performs better, with the test loss < 0.27 as well as greater test accuracy at 0.92, compared to 0.90 (Figure 9).

Q2(d)

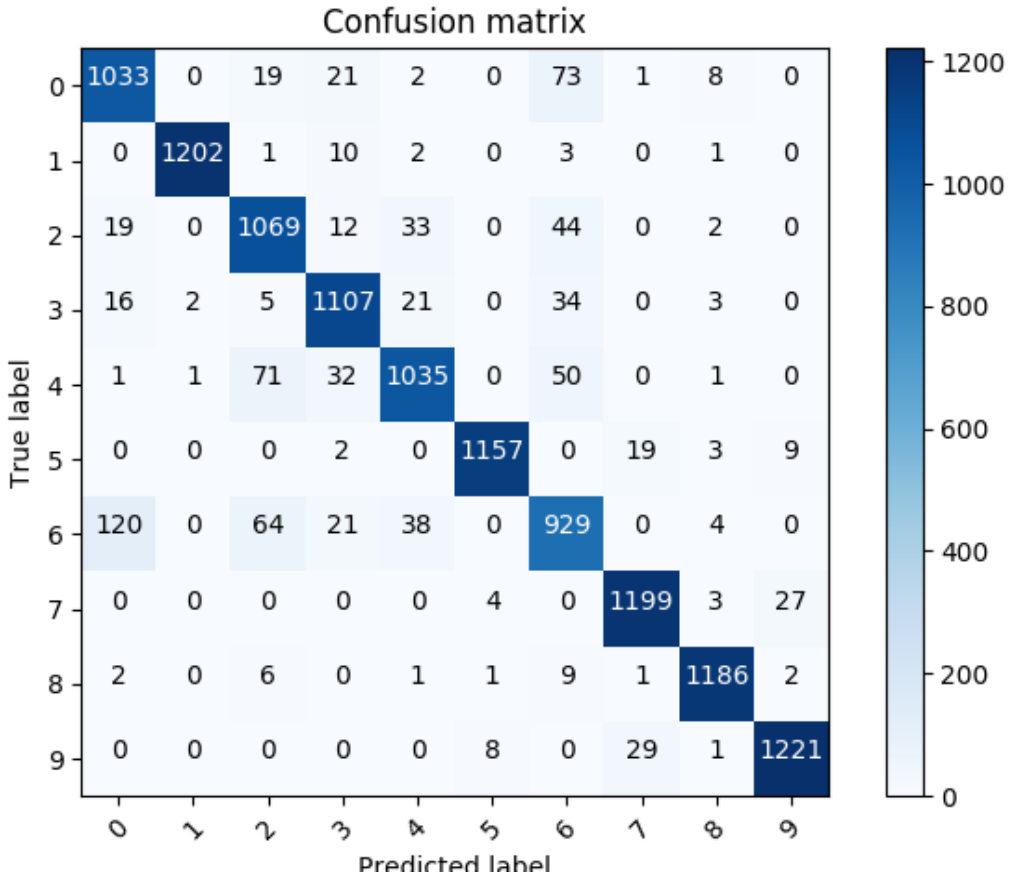


Figure 11: Confusion matrix of best architecture design

Class 6 is most difficult to classify according to the confusion matrix (Figure 11), confusing it with class 0, 2, 4 the most. Class 9 is easiest to classify according to the confusion matrix (Figure 11), with the most images being classified and labelled correctly.