Machine Learning

COMP3611

Coursework: Supervised Learning and Neural Networks

Deadline: 05/12/2019 (Extended)

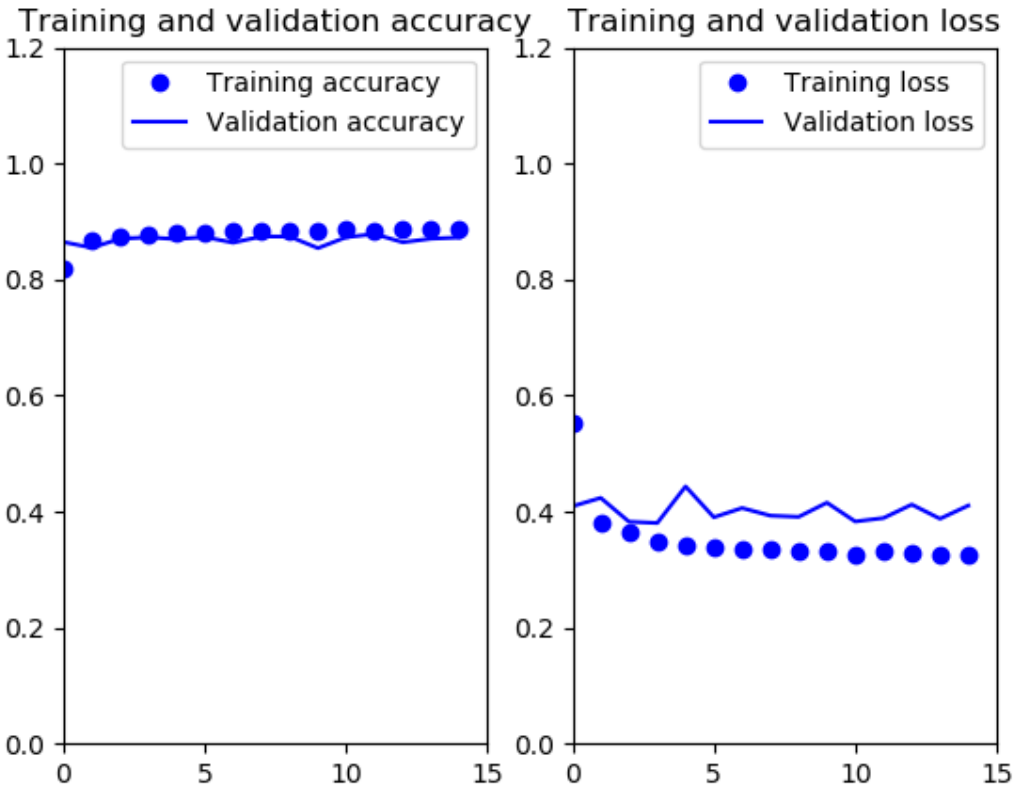
Author: James Dean, ll14j3d, 200954085

Question 1

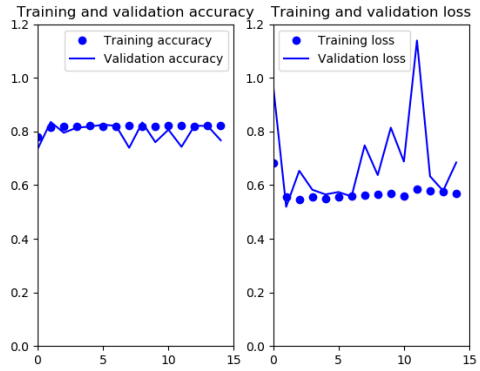
(a) ***Add to the report the plot of the accuracy and loss over time that the code generates, and comment on the result. Is the loss converging on training and validation? Is the network overfitting?***

The default learning rate is 0.03. The error rate is reasonably low, but after each epoch it's not really decreasing all that much. Hence the network is not actually 'learning' anything.

The loss is not converging to 0 on training or validation



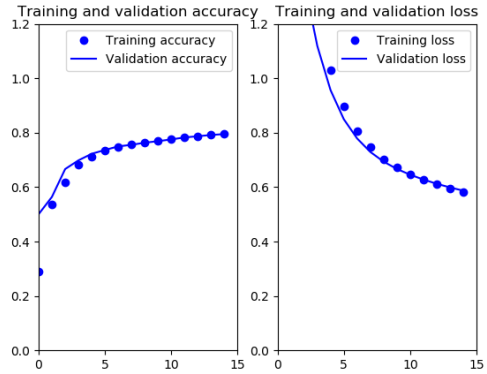
(b) ***Experiment with different learning rates and report on their effect on the loss over the training and validation data. Identify three values of the learning rate, such that one is too high, one is too law, and the third one is the one that you consider most appropriate. Include the corresponding plots in the report, with a comment on the effect of the learning rate on the graphs.***



Learning rate = 0.09

This learning rate is far too high.

The network appears to have over fitted to the training data. Meaning that that network is 'memorising' the training data rather than 'learning' to tell the differences between classes. So it would perform poorly on a general test case.

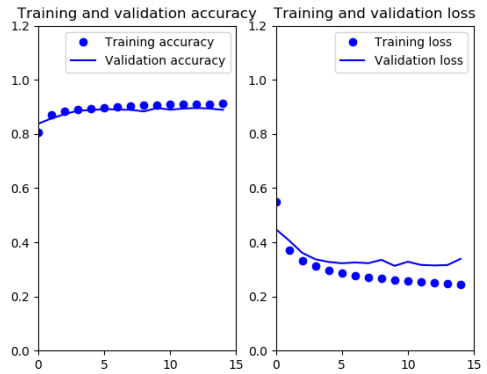


Learning rate = 0.000003

This learning rate is too low.

This is evidenced by it's initial high error rate. For the first few epochs, it does quickly reduces the error. But the rate of reduction decreases every epoch (similar to a negative logarithm function). And by 15 epochs the error is still way too high.

With this learning rate the network is not taking large enough steps to 'learn' anything meaningful per step.



Learning rate = 0.003

This appears to be a more appropriate learning rate. It starts with a reasonably low error rate, and after 15 epochs has managed to get the error near to 0.2 without any major fluctuations.

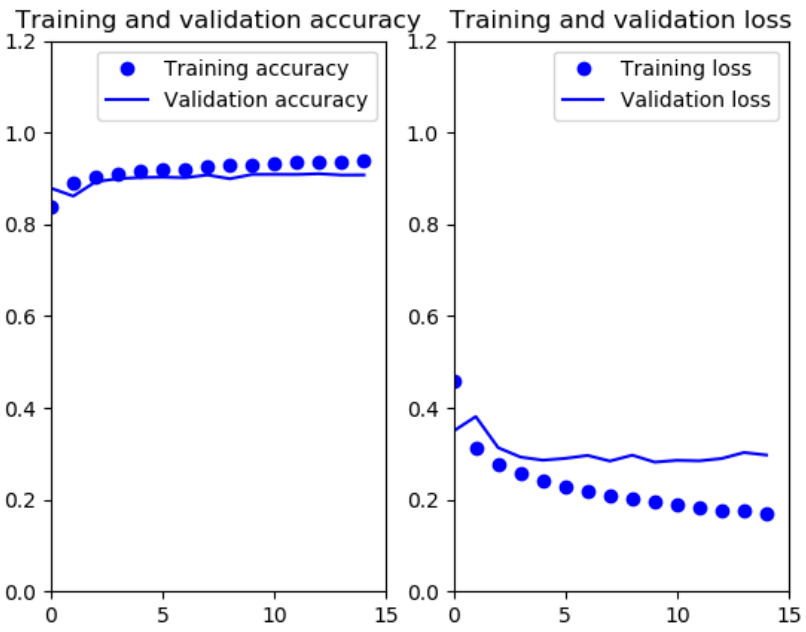
This seems to be the 'sweet spot' learning rate. Because the steps are large enough so that the network is learning enough per step, but not so much that it is overshooting

Question 2

(a) ***Design an architecture that achieves a loss on the training data that is as far below 0.28 as you can. Add the corresponding plots to the report, and describe the architecture, and the reasoning behind your changes.***

To achieve a loss below 0.28 only three changes need to be made to the original architecture. Firstly, as mentioned in 1b, in the hyper parameters reduce the learning rate to 0.03. Secondly, increase the number of filters in the `Conv2D` layer from 6 to 32. Finally Add a `Dropout` layer after the `pooling` layer. Dropout consists in randomly setting a fraction `rate` of input units to 0 at each update during training time, which helps prevent overfitting.

```
Epoch 1/15
48000/48000 [==============================] - 39s 810us/step - loss: 0.4591 - acc: 0.8374 - val_loss: 0.3492 - val_acc: 0.8788
Epoch 2/15
48000/48000 [==============================] - 37s 781us/step - loss: 0.3120 - acc: 0.8907 - val_loss: 0.3809 - val_acc: 0.8615
Epoch 3/15
48000/48000 [==============================] - 37s 780us/step - loss: 0.2773 - acc: 0.9019 - val_loss: 0.3131 - val_acc: 0.8928
Epoch 4/15
48000/48000 [==============================] - 37s 781us/step - loss: 0.2556 - acc: 0.9090 - val_loss: 0.2927 - val_acc: 0.8995
Epoch 5/15
48000/48000 [==============================] - 37s 777us/step - loss: 0.2408 - acc: 0.9148 - val_loss: 0.2859 - val_acc: 0.9020
Epoch 6/15
48000/48000 [==============================] - 39s 803us/step - loss: 0.2278 - acc: 0.9190 - val_loss: 0.2898 - val_acc: 0.9028
Epoch 7/15
48000/48000 [==============================] - 43s 891us/step - loss: 0.2180 - acc: 0.9207 - val_loss: 0.2963 - val_acc: 0.9016
Epoch 8/15
48000/48000 [==============================] - 37s 781us/step - loss: 0.2086 - acc: 0.9255 - val_loss: 0.2840 - val_acc: 0.9076
Epoch 9/15
48000/48000 [==============================] - 38s 789us/step - loss: 0.2010 - acc: 0.9290 - val_loss: 0.2968 - val_acc: 0.8993
Epoch 10/15
48000/48000 [==============================] - 38s 784us/step - loss: 0.1950 - acc: 0.9306 - val_loss: 0.2816 - val_acc: 0.9088
Epoch 11/15
48000/48000 [==============================] - 37s 773us/step - loss: 0.1895 - acc: 0.9325 - val_loss: 0.2856 - val_acc: 0.9088
Epoch 12/15
48000/48000 [==============================] - 37s 774us/step - loss: 0.1821 - acc: 0.9346 - val_loss: 0.2845 - val_acc: 0.9087
Epoch 13/15
48000/48000 [==============================] - 37s 776us/step - loss: 0.1775 - acc: 0.9368 - val_loss: 0.2895 - val_acc: 0.9103
Epoch 14/15
48000/48000 [==============================] - 38s 783us/step - loss: 0.1752 - acc: 0.9373 - val_loss: 0.3026 - val_acc: 0.9074
Epoch 15/15
48000/48000 [==============================] - 37s 780us/step - loss: 0.1697 - acc: 0.9396 - val_loss: 0.2972 - val_acc: 0.9076
Training time = 569.7754237651825
Test loss: 0.2873556327879429
Test accuracy: 0.905
Found 9014 correct labels
Found 986 incorrect labels
```



(b) ***Design an architecture (different from the one above) that achieves a loss on the validation data that is as far below 0.27 as you can. Add the corresponding plots to the report, describe the architecture, and the reasoning behind your changes.***

To reduce the error on the validation step we need to change the architecture more significantly.

Firstly I added three new `Conv2D` layers, each with 32 filters and `activation = 'relu'`. This seemed to decrease the validation loss, but as a consequence significantly increased the time per step. In an effort to combat this time issue I decreased the number of filters in the first `Conv2D` layer to 8. This decreased the time it takes per step and had minimal effect on the validation loss so I kept it in. A second edition to combat the speed issue was halving the batch size from 128 to 64. This did in fact decrease the time per step without compromising the validation loss, so again I kept this change also. SIDE NOTE: reducing the batch size reduces the amount of memory required from the computer, if I had a more powerful computer and more free time I might want to increase the batch size.

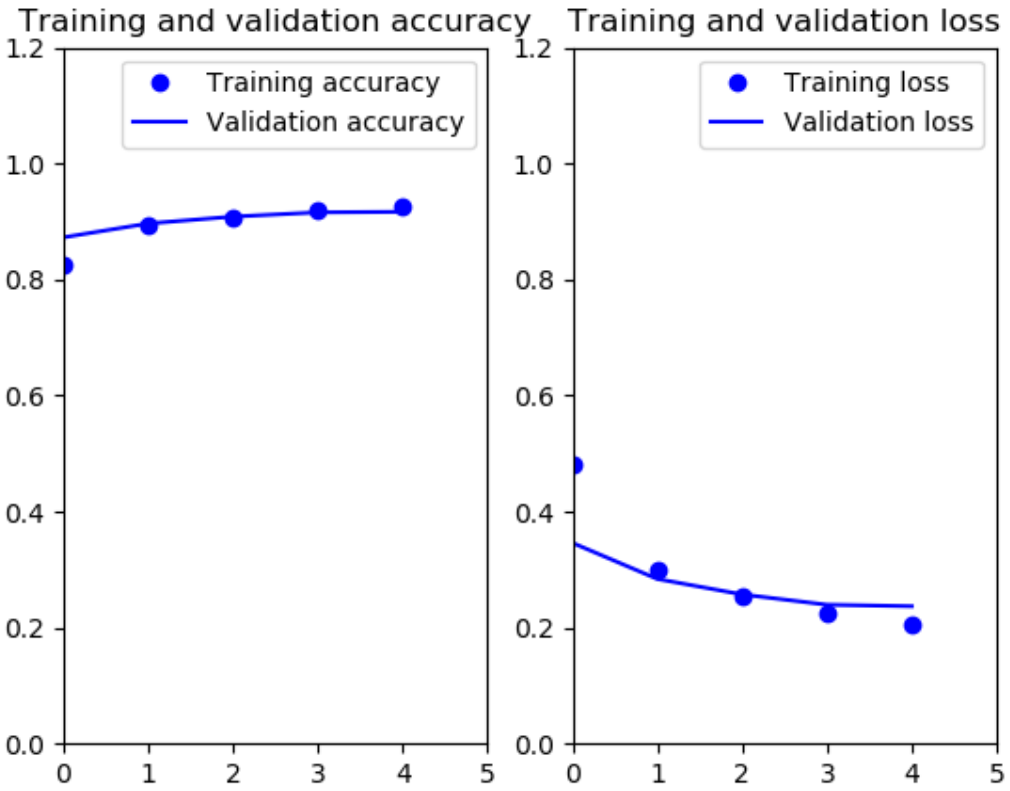
Next, decrease the number of `epochs` from 15 to 5. After testing with a large number of `epochs` I found that the optimal range for minimising validation loss is around 5

In an effort to prevent overfitting I added a `dropout` layer after the first `Conv2D` layer with a rate of 0.1, slightly higher than the rate we used for the previous section

After every two `Conv2D` layers (technically four layers counting the `relu` layers) I added a `MaxPooling` layer. My reasoning behind doing this is so that the network would look at the higher level features as a definition of a given class, rather than focus on the small similar details.

After identifying features in the images with convolutional layers, all of these get flattened into an array and further dense layers have been added. The first one having 256 neurons, the second and final is required to have the same number of neurons as there are classes, so 10.

```
Found 1000 correct labels using the untrained model
Found 9000 incorrect labels using the untrained model
Train on 48000 samples, validate on 12000 samples
Epoch 1/5
48000/48000 [==============================] - 190s 4ms/step - loss: 0.4870 - acc: 0.8226 - val_loss: 0.3175 - val_acc: 0.8876
Epoch 2/5
48000/48000 [==============================] - 150s 3ms/step - loss: 0.2942 - acc: 0.8933 - val_loss: 0.2703 - val_acc: 0.9002
Epoch 3/5
48000/48000 [==============================] - 173s 4ms/step - loss: 0.2498 - acc: 0.9098 - val_loss: 0.2597 - val_acc: 0.9061
Epoch 4/5
48000/48000 [==============================] - 163s 3ms/step - loss: 0.2234 - acc: 0.9183 - val_loss: 0.2247 - val_acc: 0.9173
Epoch 5/5
48000/48000 [==============================] - 163s 3ms/step - loss: 0.2049 - acc: 0.9260 - val_loss: 0.2296 - val_acc: 0.9182
Training time = 839.0813446044922
Test loss: 0.2368456066608429
Test accuracy: 0.915
Found 9081 correct labels
Found 919 incorrect labels
```



As evidenced by the output after running this architecture, this architecture performs quite well. The loss on the validation data gets as low as 0.2247 without significant fluctuations.

(c) ***Use both architectures from the points above on the test data, and comment on which one performs best. Hint: the results on the test set are currently printed in the terminal at the end of training.***

```
Training time = 569.7754237651825
Test loss: 0.2873556327879429
Test accuracy: 0.905
Found 9014 correct labels
Found 986 incorrect labels
```

Results from architecture (a)

While the first architecture performed well at learning and training, the test loss is worse than the second architecture. The combination of high accuracy and low test loss suggests the network was not 'learning' for general cases, but rather 'memorising' the training data.

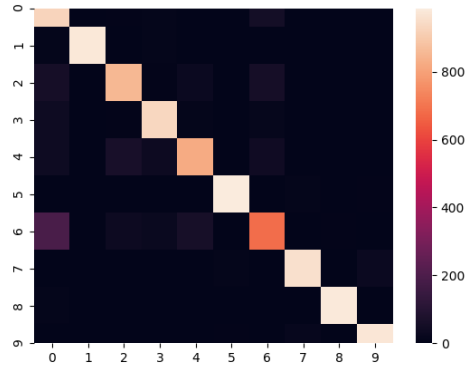
```
Training time = 839.0813446044922
Test loss: 0.2368456066608429
Test accuracy: 0.915
Found 9081 correct labels
Found 919 incorrect labels
```

Results from architecture (b)

The second architecture performed better on the test set, as evidenced by the lower test loss. This suggests that this architecture is more likely to correctly classify general data, hence would perform better than the first architecture if practice.

(d) ***Build the confusion matrix of your best architecture over the test set, and add it to the report. Which class is the easiest and which one the most difficult to classify?***

```
Confusion matrix:
 [[900   0  18  25   3   1  48   0   5   0]
 [  3 984   0   9   2   0   1   0   1   0]
 [ 38   1 887   8  41   0  24   0   1   0]
 [ 26   7   7 918  31   0  11   0   0   0]
 [ 14   1  72  18 867   0  28   0   0   0]
 [  1   0   0   0   0 987   0  10   0   2]
 [160   2  78  31  69   0 654   0   6   0]
 [  1   0   0   0   0  13   0 974   0  12]
 [  7   0   0   3   4   2   0   2 982   0]
 [  0   0   1   0   0   5   0  46   0 948]]
```



	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class7	Class 8	Class 9	Class 10	
Class 1	900	0	18	25	3	1	48	0	5	0	1000
Class 2	3	984	0	9	2	0	1	0	1	0	1000
Class 3	38	1	887	8	41	0	24	0	1	0	1000
Class 4	26	7	7	918	31	0	11	0	0	0	1000
Class 5	14	1	72	18	867	0	28	0	0	0	1000
Class 6	1	0	0	0	0	987	0	10	0	2	1000
Class 7	160	2	78	31	69	0	654	0	6	0	1000
Class 8	1	0	0	0	0	13	0	974	0	12	1000
Class 9	7	0	0	3	4	2	0	2	982	0	1000
Class 10	0	0	1	0	0	5	0	46	0	948	1000
Classification total	1150	995	1063	1012	1017	1008	766	1032	995	962	

I have output the confusion matrix in several forms for your viewing pleasure. The first is the raw print output. The second a heat map of that same matrix. Finally a spread sheet of the data with additional annotations. As you can see from the heat map, the majority of the classifications are along the diagonal of the matrix. This is good because it means that items are being classified into the correct classes in the general test case.

According to the confusion matrix Class 6 was the easiest to classify. Though there are several close runners up, notably Class 2 and Class 9. 987 images were correctly classified for Class 6, only incorrectly classifying 13 images in this test.

The hardest class for the architecture to classify, based on this test, is Class 7. The architecture only correctly classified 654 images for this class. This is a noticeably lower success rate than all other classes.