National and Kapodistrian University of Athens
Department of Informatics and Telecommunications
MSc in Data Science & Information Technologies

Machine Learning Assignment 2 – Neural Networks

Stefania Patsou

DS2190014

ATHENS
FEBRUARY 2020

Contents

Exercise 1

A)

Equations where referenced by the lecture of the course for multilayered perceptrons, in page 22. The general equations are:

General equations:

Forward calculations:

Output of every neuron:

$$y_{i\mu}^{(r)} = f(\sum_k wij^{(r)} * y_{j\mu}^{(r-1)})$$

Cost function:

$$E = \frac{1}{2} * \sum_{k\mu}(e_{k\mu})^2 \ , \ e_{k\mu} = T_{k\mu} - y_{k\mu}^{(R)}$$

Backward calculations:

Calculation of all deltas for each layer:

$$\delta_{i\mu}^{(R)} = f'(\nu_{i\mu}^{(R)}) * (y_{i\mu}^{(R)} - T_{i\mu}) \ , \ \delta_{i\mu}^{(r)} = \sum_k \delta_{k\mu}^{(r+1)} * w_{ki}^{(r+1)} * f'(\nu_{i\mu}^{(r)})$$

Weight updates:

$$w_{ij}^{(r)}(new) = w_{ij}^{(r)}(old) + \delta w_{ij}^{(r)} \ , \ \delta w_{ij}^{(r)} = -\varepsilon \sum_\mu \delta_{i\mu}^{(r)} * y_{j\mu}^{(r-1)}$$

Activation functions:

i) ReLU

$f(x) = 0$, for x < 0 , $f(x) = x$, for x $>= 0$, with range (0 to max(x)) , $f'(x) = 0$, for x < 0, $f'(x) = 1$, for x $>= 0$

ii) Hyperbolic tangent

$f(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$, with range (-1 to 1), $f'(x) = (\frac{2}{e^x+e^{-x}})^2$

iii) sigmoid

$f(x) = \frac{1}{1+e^{-x}}$, with range (0 to 1), $f'(x) = (e^{\frac{x}{2}} + e^{-\frac{x}{2}})^2$, with range [0, +infinity)

B)

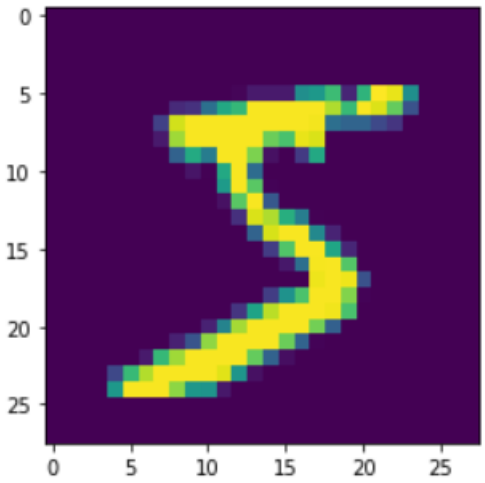
For classifying the handwritten numbers, I implemented a class named NeuralNetwork, which takes as initialization arguments the activation function and the number of layers the network must have. The layers for the exercise were 5, 20 and 40. However, my model gives +2 layers, which are the input layer and the flatten layer additionally. So, my model has 7, 22 and

42 layers. For each layer, the initialization is the same, with 32 units and the activation function for each layer is the same, either relu, tanh or sigmoid. Example of logging the creation of a model:

```
Model: "NeuralNetworkRELU5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
InputLayer (InputLayer)      (None, 28, 28)            0
_________________________________________________________________
flatten_1 (Flatten)          (None, 784)               0
_________________________________________________________________
IntermediateLayer0 (Dense)   (None, 32)                25120
_________________________________________________________________
IntermediateLayer1 (Dense)   (None, 32)                1056
_________________________________________________________________
IntermediateLayer2 (Dense)   (None, 32)                1056
_________________________________________________________________
IntermediateLayer3 (Dense)   (None, 32)                1056
_________________________________________________________________
OutputLayer (Dense)          (None, 10)                330
=================================================================
Total params: 28,618
Trainable params: 28,618
Non-trainable params: 0
_________________________________________________________________
Number of layers created: 7
```

Before compiling the model, the images of the dataset should be preprocessed, regarding their pixels and the class label type of array needs to be altered. Example of image is:

```
Out[2]: <matplotlib.image.AxesImage at 0x1fbf43ea588>
```



After fitting the model for each case, for the training data and validating on the test data, I used same number of epochs = 3. The cases which are 9, have the below test scores:

1. Hidden layers = 5 with activation function ReLU.

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/3
60000/60000 [==============================] - 10s 164us/step - loss: 0.9191 - accuracy: 0.7130 - val_loss: 0.3766 - val_accura
cy: 0.8848
Epoch 2/3
60000/60000 [==============================] - 10s 170us/step - loss: 0.3208 - accuracy: 0.9058 - val_loss: 0.2691 - val_accura
cy: 0.9192
Epoch 3/3
60000/60000 [==============================] - 8s 129us/step - loss: 0.2507 - accuracy: 0.9268 - val_loss: 0.2464 - val_accurac
y: 0.9302
```

Figure 1: Test Scores with 5 layers and ReLU

2. Hidden layers = 5 with activation function Tanh.

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/3
60000/60000 [==============================] - 14s 238us/step - loss: 0.7227 - accuracy: 0.8195 - val_loss: 0.3576 - val_accura
cy: 0.9037
Epoch 2/3
60000/60000 [==============================] - 29s 488us/step - loss: 0.3175 - accuracy: 0.9113 - val_loss: 0.2769 - val_accura
cy: 0.9191
Epoch 3/3
60000/60000 [==============================] - 46s 763us/step - loss: 0.2553 - accuracy: 0.9281 - val_loss: 0.2361 - val_accura
cy: 0.9323
```

Figure 2: Test Scores with 5 layers and Tanh

3. Hidden layers = 5 with activation function Sigmoid.

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/3
60000/60000 [==============================] - 10s 167us/step - loss: 2.3051 - accuracy: 0.1109 - val_loss: 2.3009 - val_accura
cy: 0.1135
Epoch 2/3
60000/60000 [==============================] - 111s 2ms/step - loss: 2.3012 - accuracy: 0.1114 - val_loss: 2.3011 - val_accurac
y: 0.1135
Epoch 3/3
60000/60000 [==============================] - 51s 851us/step - loss: 2.3008 - accuracy: 0.1122 - val_loss: 2.3007 - val_accura
cy: 0.1135
```

Figure 3: Test Scores with 5 layers and Sigmoid

4. Hidden layers = 20 with activation function ReLU.

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/3
60000/60000 [==============================] - 84s 1ms/step - loss: 2.2943 - accuracy: 0.1296 - val_loss: 2.2565 - val_accura
cy: 0.2009
Epoch 2/3
60000/60000 [==============================] - 69s 1ms/step - loss: 1.8652 - accuracy: 0.2599 - val_loss: 1.4297 - val_accura
cy: 0.4038
Epoch 3/3
60000/60000 [==============================] - 19s 318us/step - loss: 1.2581 - accuracy: 0.5083 - val_loss: 0.9546 - val_accu
racy: 0.6609
```

Figure 4: Test Scores with 20 layers and ReLU

5. Hidden layers = 20 with activation function Tanh.

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/3
60000/60000 [==============================] - 66s 1ms/step - loss: 0.8834 - accuracy: 0.7368 - val_loss: 0.4466 - val_accura
cy: 0.8889
Epoch 2/3
60000/60000 [==============================] - 60s 999us/step - loss: 0.3685 - accuracy: 0.9041 - val_loss: 0.2954 - val_accu
racy: 0.9245
Epoch 3/3
60000/60000 [==============================] - 18s 305us/step - loss: 0.2749 - accuracy: 0.9291 - val_loss: 0.2461 - val_accu
racy: 0.9348
```

Figure 5: Test Scores with 20 layers and Tanh

6. Hidden layers = 20 with activation function Sigmoid.

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/3
60000/60000 [==============================] - 77s 1ms/step - loss: 2.3056 - accuracy: 0.1096 - val_loss: 2.3027 - val_accura
cy: 0.1135
Epoch 2/3
60000/60000 [==============================] - 73s 1ms/step - loss: 2.3019 - accuracy: 0.1112 - val_loss: 2.3014 - val_accura
cy: 0.1135
Epoch 3/3
60000/60000 [==============================] - 56s 939us/step - loss: 2.3020 - accuracy: 0.1108 - val_loss: 2.3017 - val_accu
racy: 0.1135
```

Figure 6: Test Scores with 20 layers and Sigmoid

7. Hidden layers = 40 with activation function ReLU.

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/3
60000/60000 [==============================] - 15s 252us/step - loss: 2.3016 - accuracy: 0.1123 - val_loss: 2.3012 - val_accu
racy: 0.1135
Epoch 2/3
60000/60000 [==============================] - 74s 1ms/step - loss: 2.3013 - accuracy: 0.1124 - val_loss: 2.3011 - val_accura
cy: 0.1135
Epoch 3/3
60000/60000 [==============================] - 76s 1ms/step - loss: 2.3013 - accuracy: 0.1124 - val_loss: 2.3011 - val_accura
cy: 0.1135
```

Figure 7: Test Scores with 40 layers and ReLU

8. Hidden layers = 40 with activation function Tanh.

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/3
60000/60000 [==============================] - 13s 218us/step - loss: 1.0731 - accuracy: 0.6593 - val_loss: 0.6363 - val_accu
racy: 0.8373
Epoch 2/3
60000/60000 [==============================] - 12s 197us/step - loss: 0.4914 - accuracy: 0.8824 - val_loss: 0.4394 - val_accu
racy: 0.8989
Epoch 3/3
60000/60000 [==============================] - 39s 655us/step - loss: 0.3681 - accuracy: 0.9140 - val_loss: 0.3382 - val_accu
racy: 0.9221
```

Figure 8: Test Scores with 40 layers and Tanh

9. Hidden layers = 40 with activation function Sigmoid.

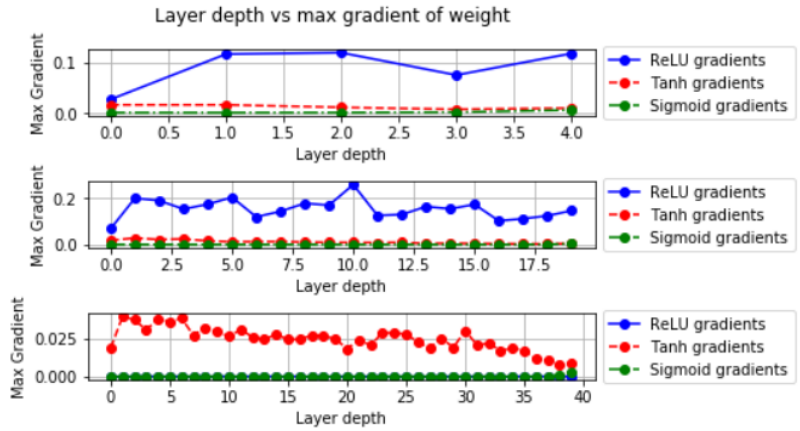
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/3
60000/60000 [==============================] - 77s 1ms/step - loss: 2.3056 - accuracy: 0.1096 - val_loss: 2.3027 - val_accura
cy: 0.1135
Epoch 2/3
60000/60000 [==============================] - 73s 1ms/step - loss: 2.3019 - accuracy: 0.1112 - val_loss: 2.3014 - val_accura
cy: 0.1135
Epoch 3/3
60000/60000 [==============================] - 56s 939us/step - loss: 2.3020 - accuracy: 0.1108 - val_loss: 2.3017 - val_accu
racy: 0.1135
```

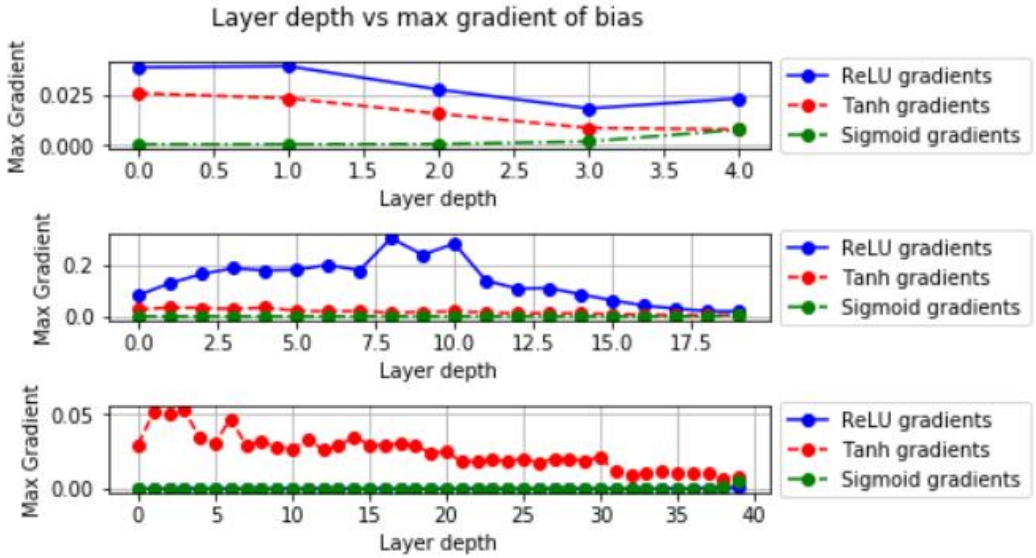
Figure 9: Test Scores with 40 layers and Sigmoid

We can see that, the activation function tanh has the best accuracies across all implementations and different number of layers. ReLU is best with small number of layers. Sigmoid function is the worst for our implementations. Best accuracy with 5 layers has tanh with 0.9281 and val_accuracy 0.9302. For models with 20 layers, best accuracy has tanh with 0.9291 and val_accuracy 0.9348. For models with 40 layers, best accuracy has tanh with 0.9140 and val_accuracy 0.9221.

C)

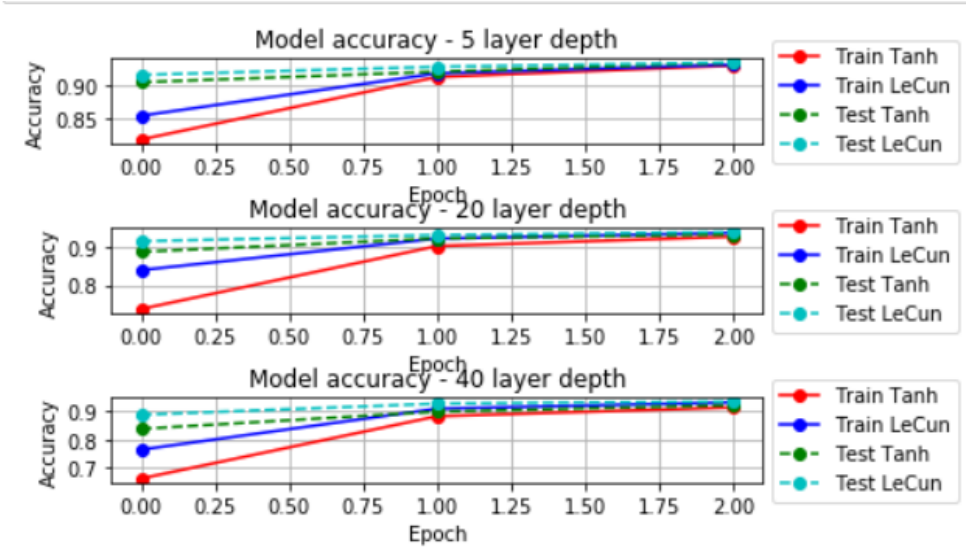
With 3 epochs, number of iterations of the training set, and the calculation of the gradients from the help pdf provided, gradients are divided to gradients of the weights and the gradients of the biases. From the plots, we can derive that ReLU function produces the maximum gradient values for layers 5 and 20. For a number of 40 layers, tanh function produces higher values. Large accumulations like the ones below, show that the network is unstable. Each peak in the plots, show how the accuracy is changed in the test scores calculated in B. Large peaks of the gradients, check the error of the network and recalculate the weights. The sigmoid function is the worst one, as we can see from the test scores and the gradient plots. The gradients of sigmoid show no large or even small change at each layer. The calculation of the gradients show the optimization of the network at each layer.

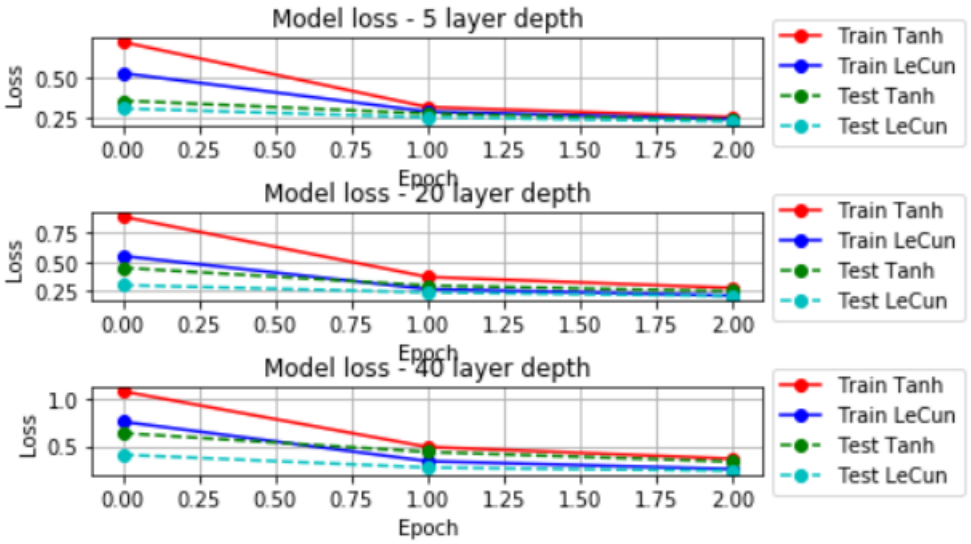




D)

After training the model with LeCun activation function, we come up with these plots:





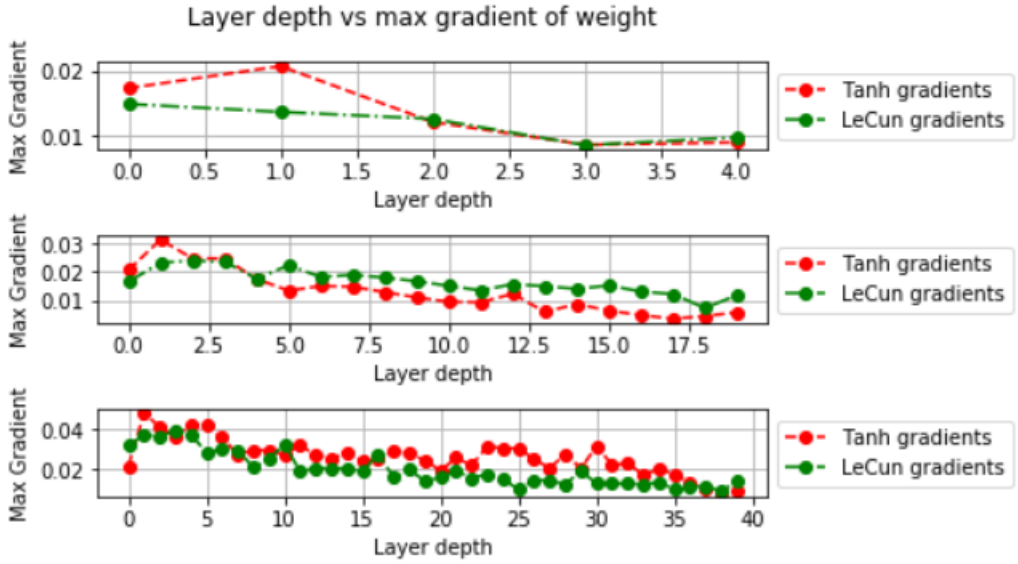
The above learning curves show that, tanh starts with a larger loss than LeCun. In addition, LeCun's accuracy is better in the beginning and at the end of the epochs.

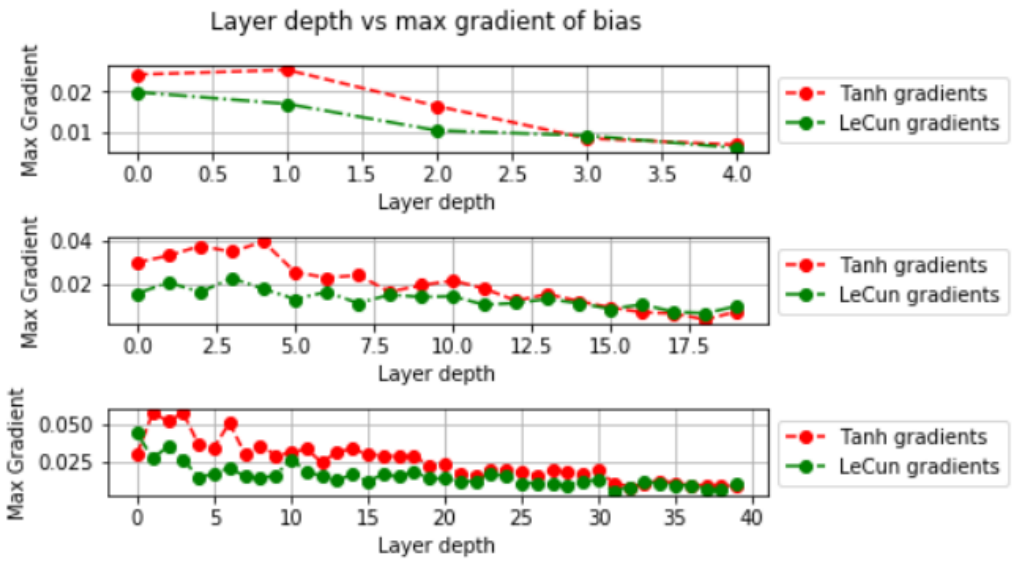
All equations are similar as A) subproblem. The only thing that changes is the activation function.

Activation function:

$f(x) = 1.7159 * tanh(\frac{2}{3}x) + 0.01 * x$, with range (-2 to 2), $f'(x) = 1.14393 * sech^2(\frac{2}{3}x) + 0.01$

The gradients below show that LeCun does not have large transitions after each layer. Also, the transitions have somewhat the same values, indicating that they are not so different with each other.

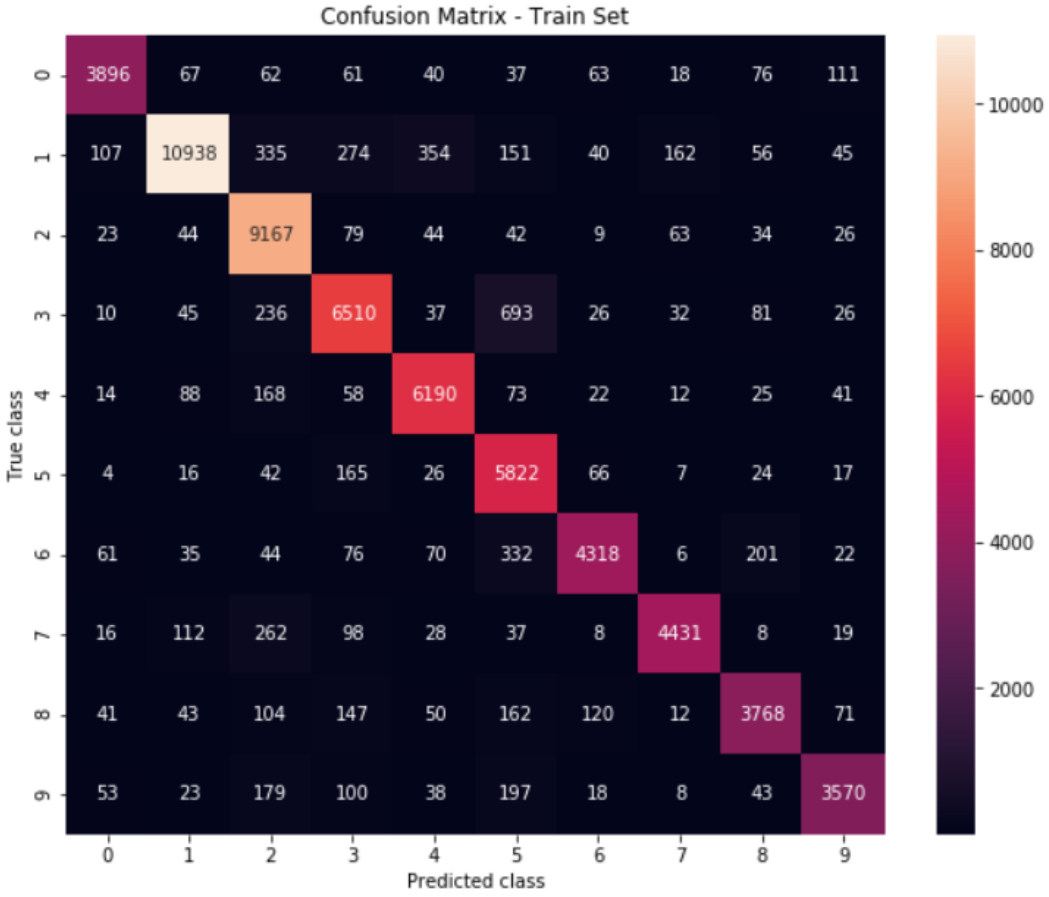




Exercise 2

A)

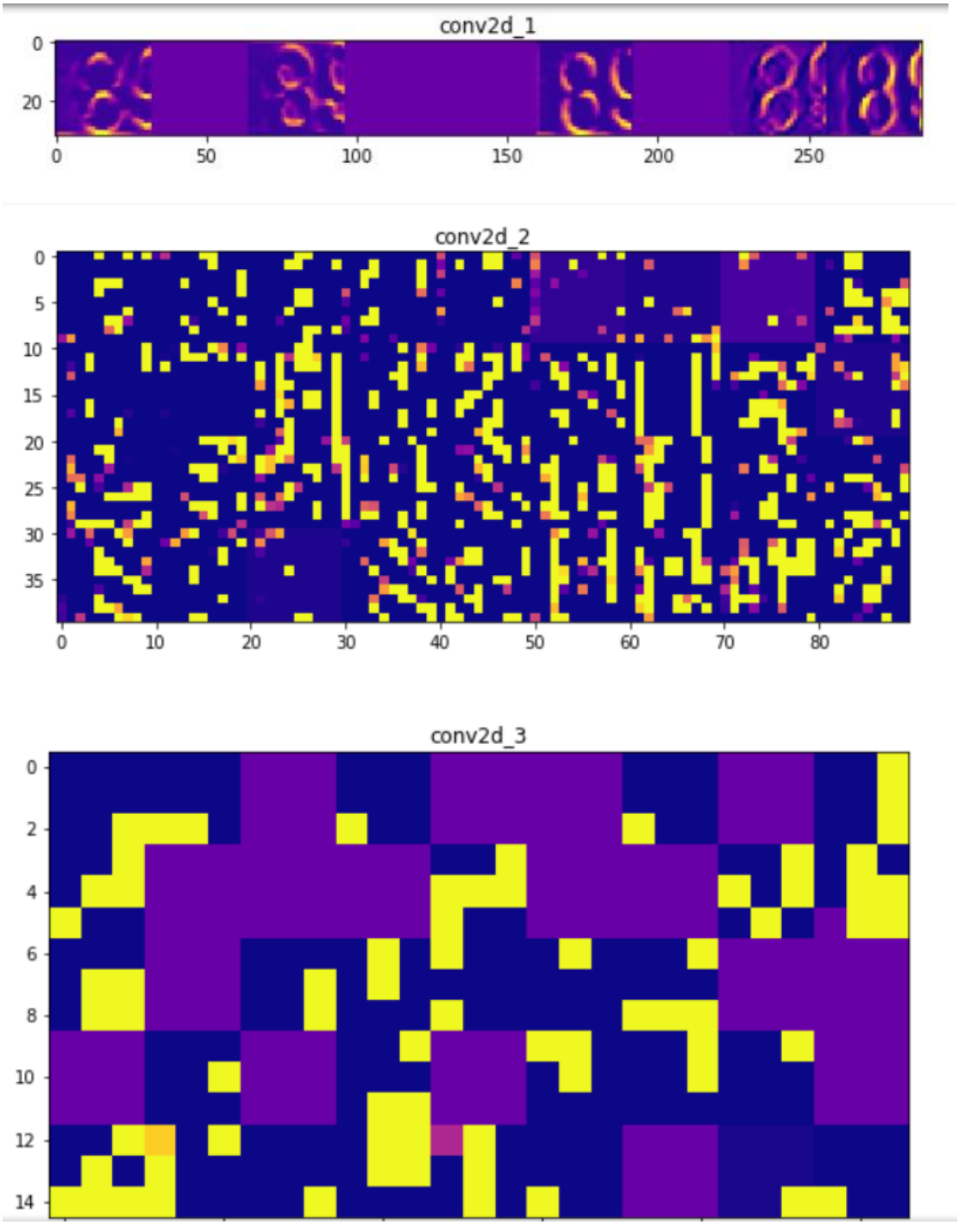
The model of Convolutional Neural Network has an accuracy of approximately 0.8762. The confusion matrix is:



The confusion matrix above shows which labels were predicted correctly. The 1s of our training set are more. Furthermore, we can see that cases predicted wrong can also depicted here, e.g. 3 and 5 labels have the most interchangeable labels, 6 and 5 too.

B)

The plots produced from the convolutional filters show that every time convolutional layers are enlarged and deepen, the characteristics of the image are not clear. This shows how CNN works. As we go deep into the layers, the more higher-level features are learned, and we cannot interpret them by our eyes. An example of



Exercise 3

This exercise takes a handwritten number from myself and giving it as input to the model created from exercise 2. The model from exercise 2 checks for numbers from the street. However, it is odd some kind of odd that it classified my handwritten number correctly.

System specs

System used for data obtaining, data pre-processing and data manipulation has the below specs:

1. Windows 10 Home 64-bit
2. 8GB RAM
3. 500GB Hard disk
4. 4 CPUs (~ 2.6 GHz)

Complete Jupyter Notebook script and files

For this project, the complete Jupyter notebook script can be found on github link:

https://github.com/PiStefania/Machine-Learning-Assignment2

References

[1] http://neuralnetworksanddeeplearning.com/ [Accessed 20/2/20]
[2] https://machinelearningmastery.com/keras-functional-api-deep-learning/ [Accessed 22/2/20]
[3] https://stackoverflow.com/questions/37664783/how-to-plot-a-learning-curve-for-a-keras-experiment [Accessed 26/2/20]
[4] https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/ [Accessed 26/2/20]
[5] https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5 [Accessed 26/2/20]
[6] https://github.com/tohinz/SVHN-Classifier/blob/master/preprocess_svhn.py [Accessed 26/2/20]
[7] https://datascience.stackexchange.com/questions/34444/what-is-the-difference-between-fit-and-fit-generator-in-keras [Accessed 26/2/20]
[8] https://keras.io/preprocessing/image/ [Accessed 25/2/20]
[9] https://keras.io/models/sequential/ [Accessed 27/2/20]
[10] https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/ [Accessed 27/2/20]
[11] https://jjallaire.github.io/deep-learning-with-r-notebooks/notebooks/5.4-visualizing-what-convnets-learn.nb.html [Accessed 27/2/20]