ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν Πανεπιστήμιον Αθηνών
ΙΔΡΥΘΕΝ ΤΟ 1837

MACHINE LEARNING

Autoencoder

Authors:

Vasiliki Koumarela VasiaKoum - 1115201600074

Charalampos Katimertzis chariskms - 1115201600062

Parts

Part 1 : Convolutional Autoencoder
Part 2 : Neural Network Classifier (CNN + FC)

Introduction

This project is about training and evaluation of neural network autocoding of numerical images digits. Encoder is used to create a neural network of image classification. First, we construct the autoencoder N1 with these hyperparameters: Number_of_Layers, Number_of_Filter_Size, Number_of_Filters_by_Layer, Number_of_Epochs & Number_of_Batch_Size, we split the dataset in training set & validation set and after the execution, the model is saved. Then, for the second part, we construct a NN for image classification (N2). The N1 model is used to form a fully connected NN and an output layer. The training of N2 is done in two stages to achieve the reduction of convergence time.

Requirements

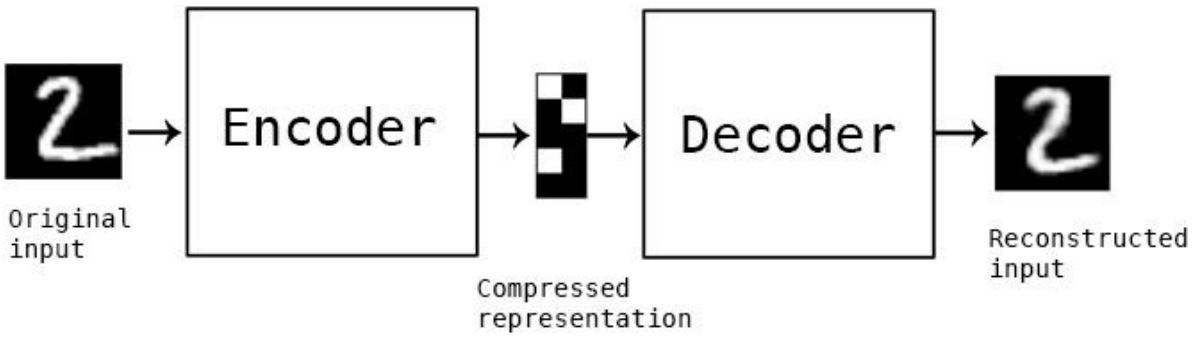
- Python 3.8
- Keras
- Numpy
- Matplotlib
- Tensorflow
- Pandas

- Sklearn

Files

- ❖ autoencoder.py
- ❖ classification.py
- ❖ functions.py /has functions for both programs

Part 1: CNN - Autoencoder



The model was trained with a subset of 60k of black and white images of digits. The encoder for one layer uses **Conv2D**, **BatchNormalization** and **MaxPooling2D** (for the first 2 layers). Respectively, the decoder uses Conv2D, BatchNormalization and **UpSampling** (for the last 2 layers). The program is using the mirrored architecture, that is symmetric layers in the encoder & decoder, which is more efficient. **Mean_Squared_error** (for loss → `mean_of_square(y_true - y_pred)`) & **RMSprop** (for optimizer → maintaining a moving average of the square of gradients, dividing the gradient by the root of this average) were used to compile the model. Also, for each hidden layer, **REctified Linear Unit** activation function is used except from the last layer in

the decoder that **Sigmoid** activation function is called. No Dropout filter is used because from the experiments we concluded that the val_loss is much lower than loss, which is not a normal training and most importantly no overfitting was observed.

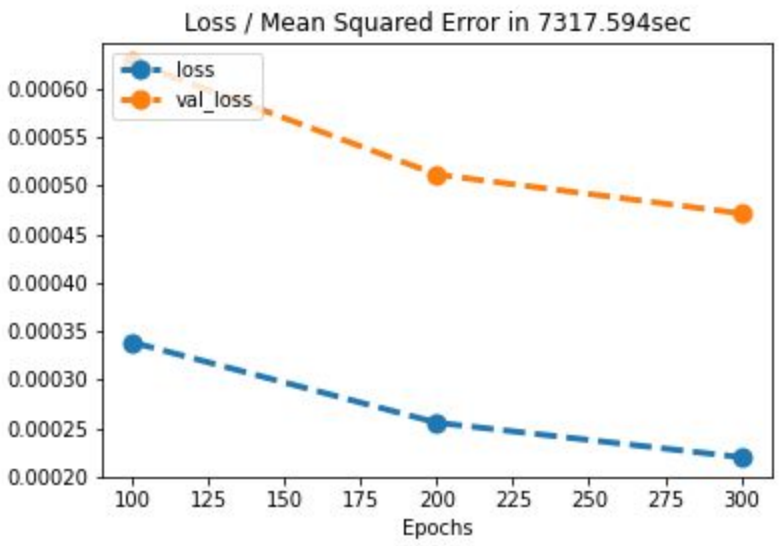
The autoencoder is executed by: python autoencoder.py -d dataset. The user choices are: 1) Execution with different hyperparameters, 2) Plot of the error-graphs, 3) Save of the existing model, 4) Exit. If the user chooses (1), then the user choices are: 1) Layers, 2) Filter_Size, 3) Filters/Layer, 4) Epochs, 5) Batch_Size. For the graph plots, the program shows a graph with the losses & the changed hyperparameter. The user, executing the program once, can change multiple hyperparameters and the graph plots will include all the changes of the hyperparameters, keeping the initial hyperparameters constant and each time the selected hyperparameter will be changed.

Metrics and times were measured at Google Colab with specs: CPU x2 @ 2.2GHz, RAM:13 GB, GPU: Nvidia Tesla K80

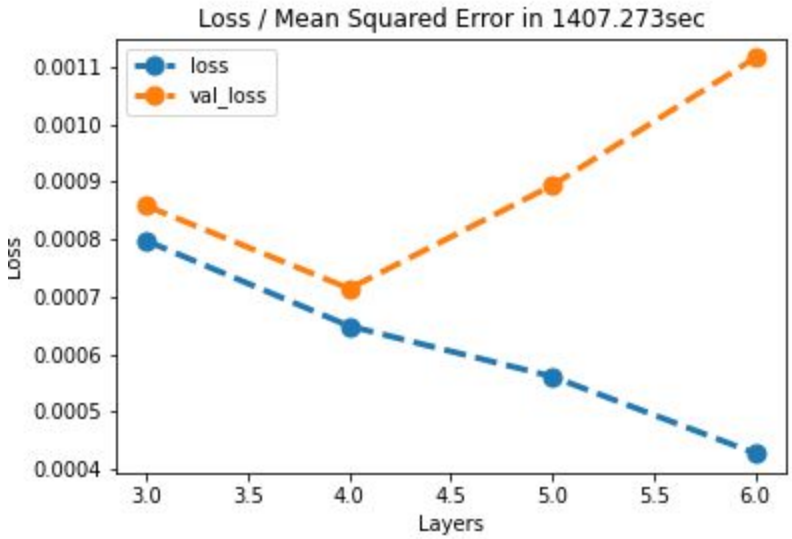
1. EXPERIMENTS

We executed the autoencoder.py with different values for hyperparameters and these are the results:

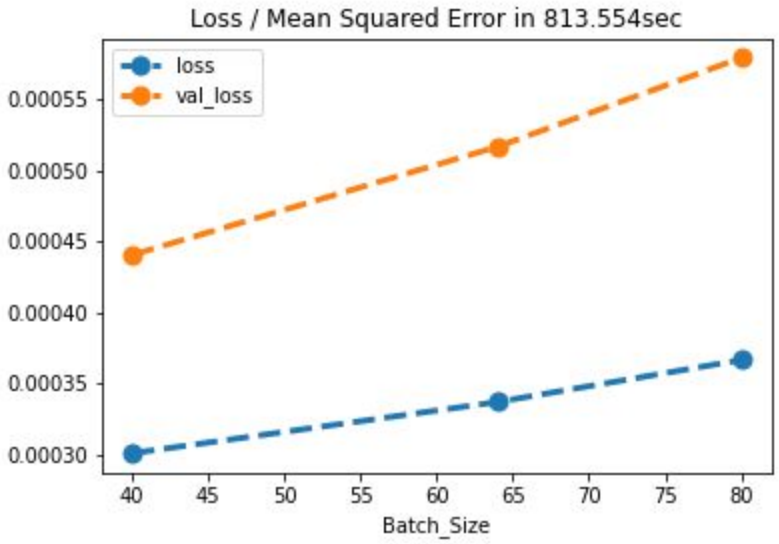
- **L4_FS3_FL32_Ex_B64:**



- **Lx_FS3_FL16_E100_B64:**

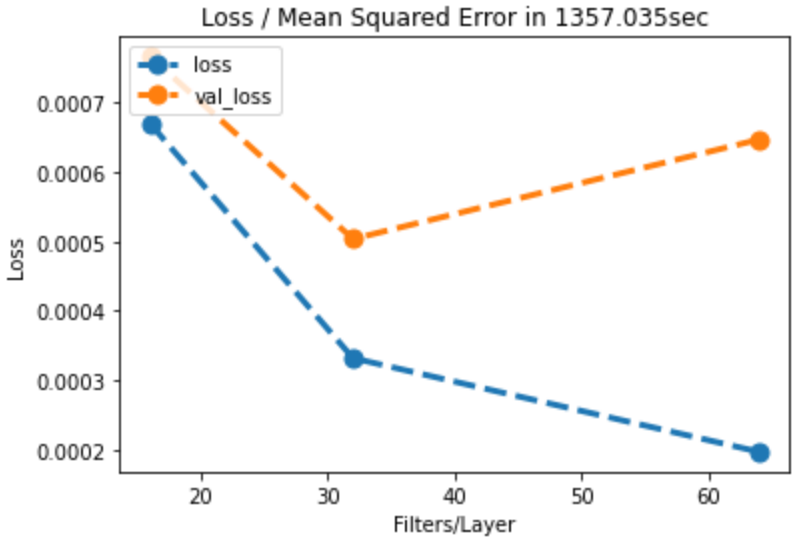


- **L4_FS3_FL32_E100_Bx:**



- **L4_FSx_FL32_E100_B64:**

- **L4_FS3_FLx_E100_B64:**



- **Metrics from loss_values.csv:**

Layers	Filter_Size	Filters/Layer	Epochs	Batch_Size	Train_Time	Loss	Val_Loss
4.0	3.0	32.0	100.0	40.0	1045,95	0,0003	0,00044
4.0	3.0	32.0	300.0	64.0	2166,66	0,00021	0,00048
4.0	3.0	32.0	100.0	64.0	847,36	0,00033	0,00051

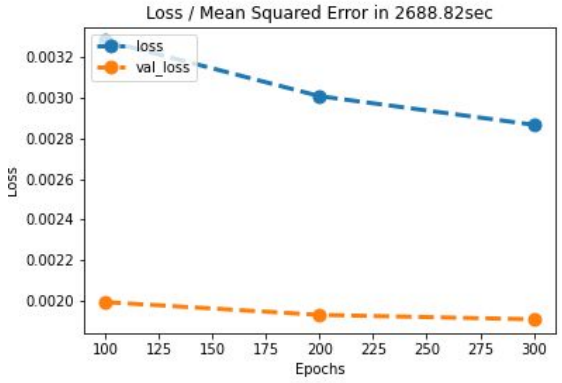
4.0	3.0	32.0	200.0	64.0	1444,88	0,00024	0,00055
4.0	3.0	32.0	100.0	80.0	813,55	0,00036	0,00057
4.0	2.0	32.0	100.0	64.0	777,87	0,00045	0,00057
4.0	3.0	64.0	100.0	64.0	1357,03	0,00019	0,00064
4.0	3.0	16.0	100.0	40.0	698,55	0,00059	0,00065
5.0	3.0	32.0	100.0	40.0	2584,62	0,00024	0,00069
4.0	3.0	16.0	100.0	64.0	484,79	0,00065	0,0007
5.0	3.0	16.0	100.0	64.0	794,88	0,00056	0,00091
6.0	3.0	16.0	100.0	64.0	1863,05	0,00042	0,00119
4.0	4.0	32.0	100.0	64.0	1737,81	0,00049	0,00141
6.0	3.0	32.0	100.0	40.0	4743,8	0,00032	0,00146

2. EXPLANATION OF RESULTS

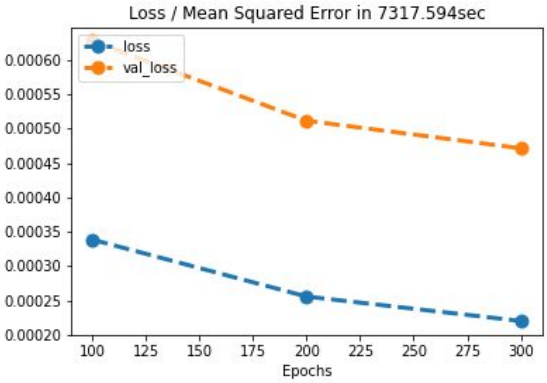
From the experiments we observe that the best values for the hyperparameters are: Layers=4, Filter_Size=3, Filters/Layer=32, Epochs ∈ [100,300], Batch_Size ∈ [40,64]. To choose the values for the hyperparameters we observe that when layers are increased, the loss plot falls. On the other hand, validation_loss for layer=4 has the minimum loss (<4 & >4 val_loss is increasing). Also, for Filter_Size, the best choice is 3 because this value has the minimum loss & val_loss. Besides that, 3 is usually the one chosen because we have symmetry with the previous layer pixel and the output, so we will not have to account for distortions across the layers which happen when using an even sized kernel. The Filters/Layer catch few of some simple features of images (edges, color tone, etc) and the next layers are trying to obtain more complex features based on simple ones. From the error-graphs, the best value for Filters/Layer is 32 because in this value, the val_loss is the minimum. For Epochs and Batch_Size, as we increase the epochs, the losses are decreasing and respectively as we decrease the Batch_Size, the values for

losses are decreasing, as well. From loss_values.csv, we observe that the Layers and Filters/Layer affect negatively training time and the deeper the CNN, the higher are the loss values. Finally, Dropout layer isn't required because the model isn't overfitting eg:

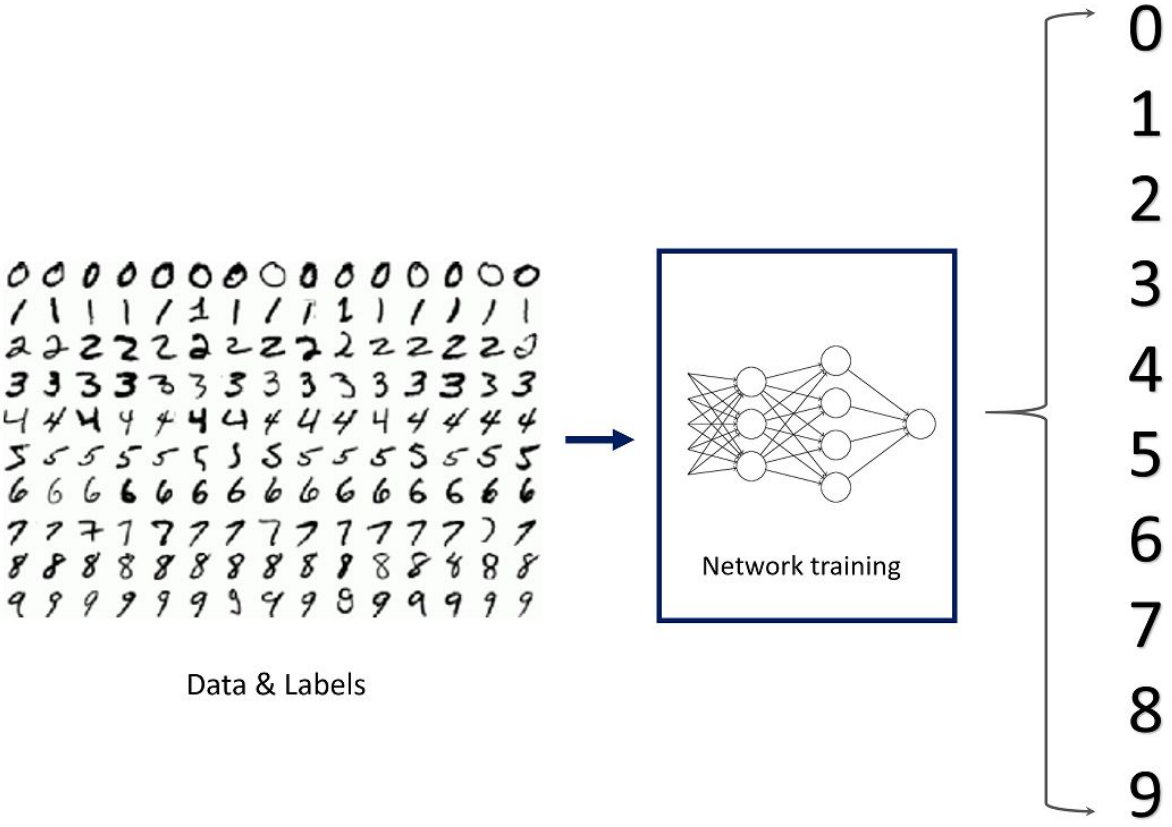
With Dropout:



No Dropout:



Part 2: Classification



Data & Labels

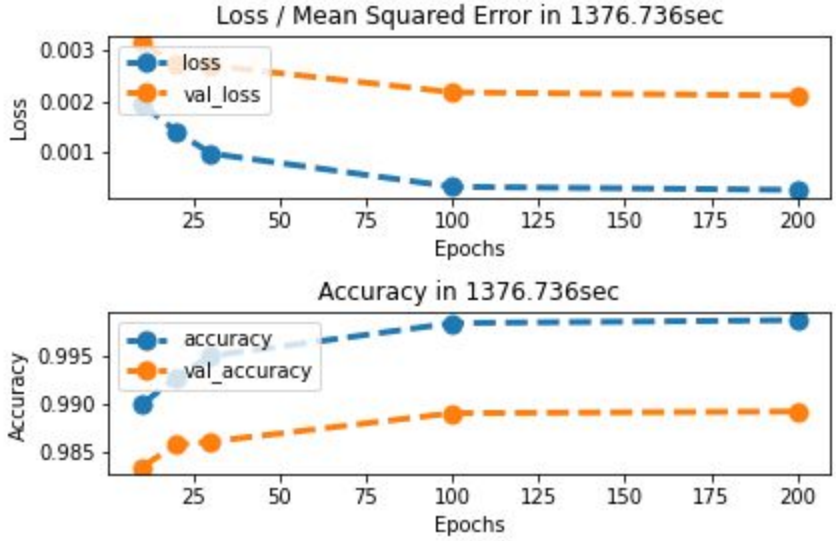
The classification model splits in two parts. In the first part, we use the encoder with the trained weights from the autoencoder which we have already trained. The encoding part encodes the test data in order to make the 1D array that is needed for classification. The second part consists of a fully connected network layer and an output layer so as to extract the correct predictions for our labels. The fully connected network for the experiments uses the softmax activation function. No Dropout filter is used because from the experiments we concluded that the val_loss is much lower than loss, which is not a normal training and most importantly no overfitting was observed.

The classification is executed by: python3 classification.py -d –d <training set> –dl <traininglabels> -t <testset> -tl <test labels> -model <autoencoder h5>. The user choices are: 1) Execution with different hyperparameters, 2) Plot of the error-graphs, 3) Predict the test data. (outputs 2 images of the predicted labels) 4) Exit. If the user chooses (1), then the user choices are: 1) Layers, 2) Fc units, 3) Epochs, 4) Batch_Size. For the graph plots, the program shows a graph with the losses, the accuracies & the changed hyperparameter. The user, executing the program once, can change multiple hyperparameters and the graph plots will include all the changes of the hyperparameters, keeping the initial hyperparameters constant and each time the selected hyperparameter will be changed.

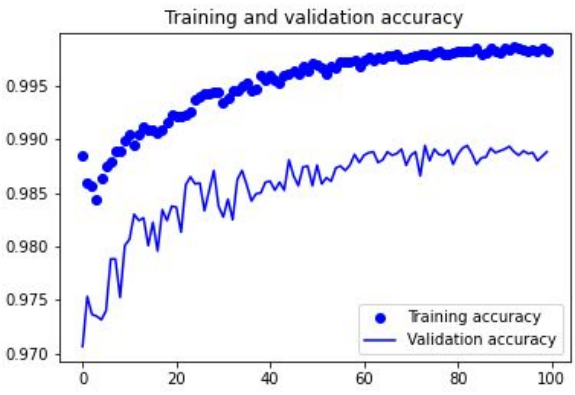
1. EXPERIMENTS

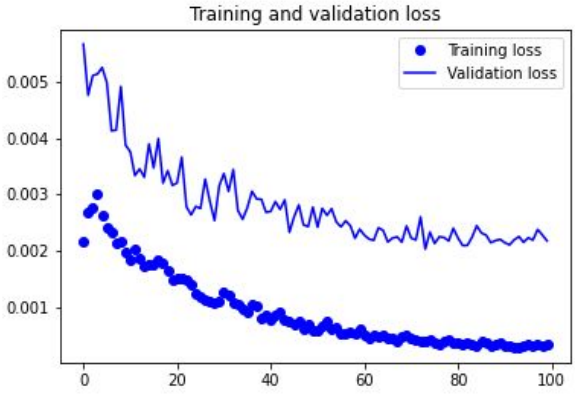
We executed the classification.py with optimized autoencoder model **L4_FS3_FL32_E100_B40** and different values for hyperparameters and these are the results:

- **L4_FC32_Ex_B64:**

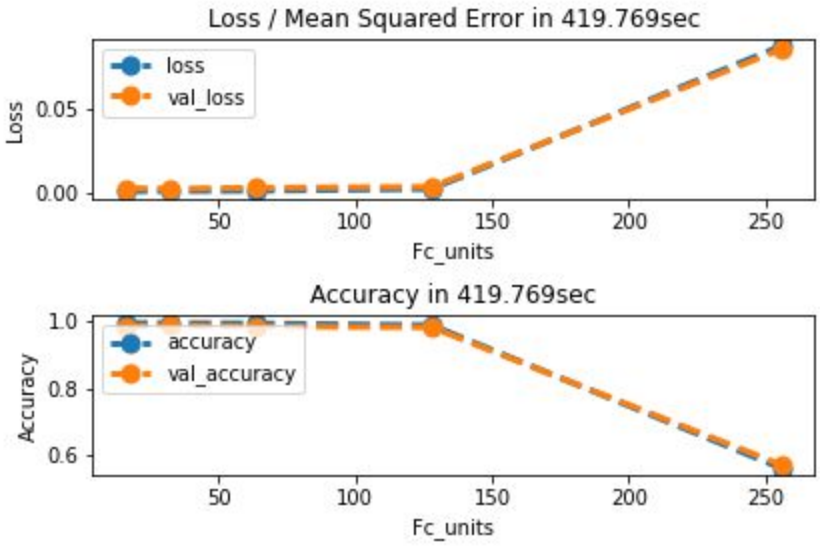


For the training with Epochs 100:

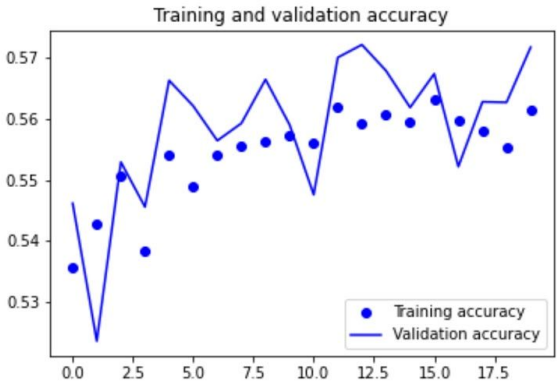


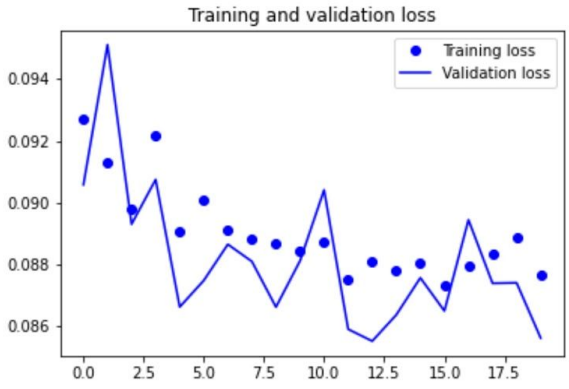


- **L4_FCx_E20_B64:**

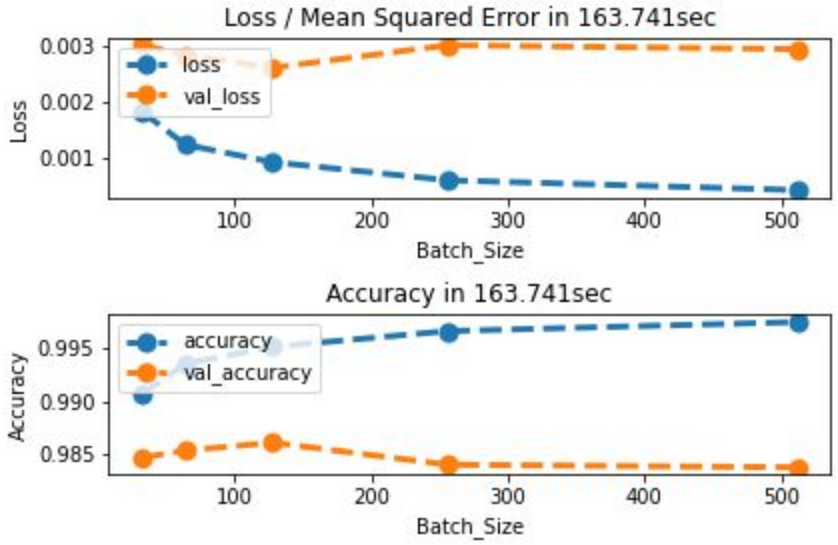


For the training with Fc units 256:

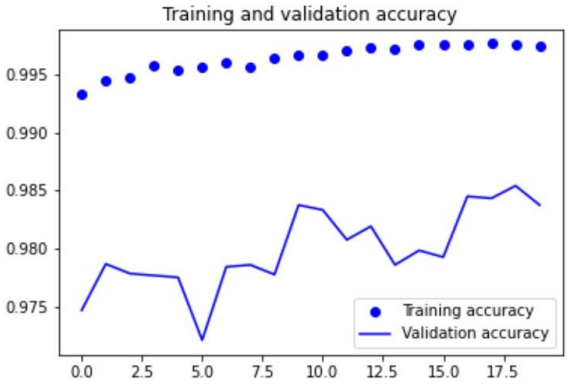


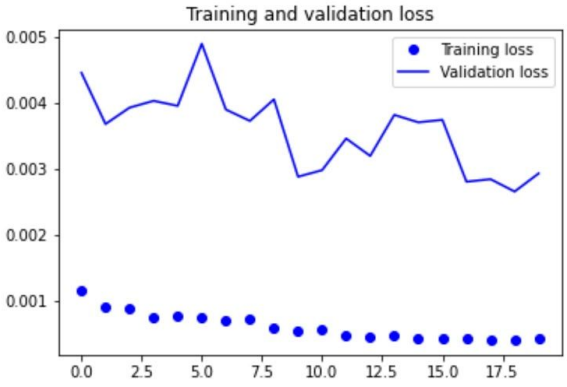


- **L4_FC32_E20_Bx:**

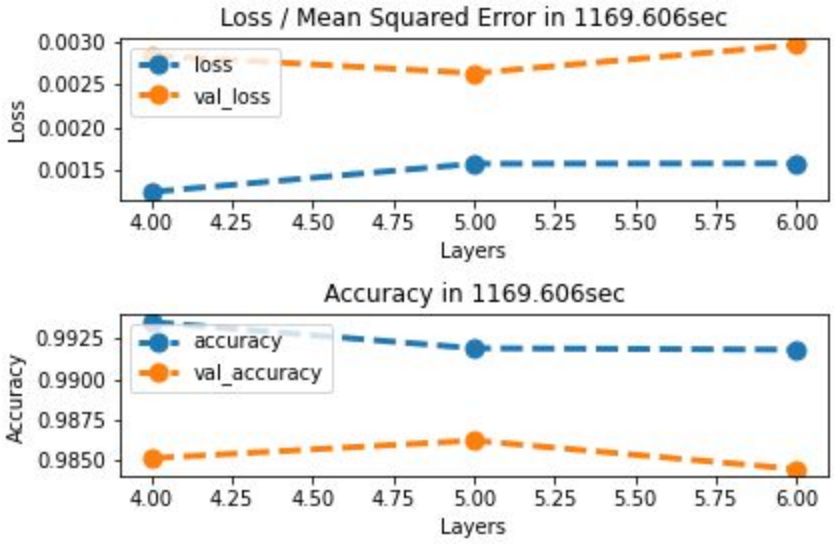


For the training with Batch size 512:

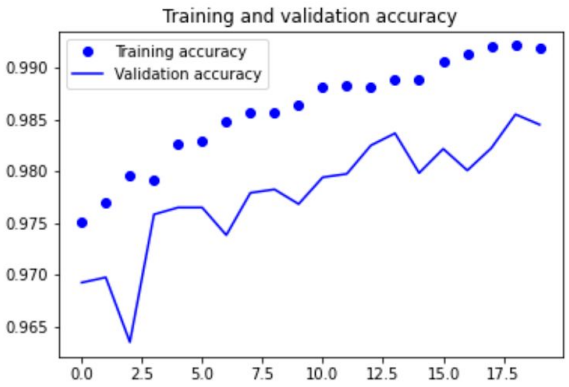


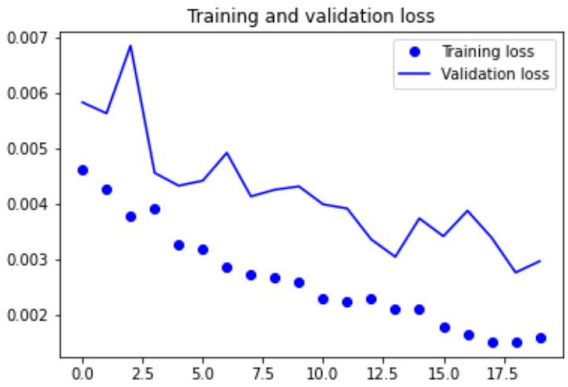


- **Lx_FC32_E20_Bx:**



For the training with 6 layers:





- **Metrics from classification_loss_values.csv:**

Layers	Fc_units	Epochs	Batch_Size	Train_Time	Loss	Val_Loss	Accuracy	Val_Accuracy
4.0	32.0	200.0	64.0	1.518,93	0,00024	0,0021	0,9987	0,9892
4.0	32.0	100.0	64.0	726,20	0,0004	0,00226	0,9979	0,9882
4.0	32.0	20.0	64.0	341,91	0,00111	0,00252	0,9941	0,9869
5.0	32.0	20.0	64.0	460,14	0,00157	0,00263	0,9919	0,9862

4.0	32.0	20.0	128.0	240,39	0,00093	0,0026	0,9951	0,986
4.0	32.0	30.0	64.0	211,14	0,00083	0,00283	0,9956	0,9853
4.0	16.0	20.0	64.0	343,56	0,00105	0,00281	0,9944	0,9851
4.0	32.0	20.0	32.0	633,38	0,00183	0,00301	0,9906	0,9846
6.0	32.0	20.0	64.0	1.169,60	0,00157	0,00296	0,9918	0,9844
4.0	64.0	20.0	64.0	350,09	0,0013	0,00297	0,9932	0,9844
4.0	32.0	20.0	256.0	185,44	0,00061	0,003	0,9966	0,984
4.0	32.0	20.0	512.0	163,74	0,00043	0,00293	0,9974	0,9837
4.0	128.0	20.0	64.0	373,89	0,00234	0,00359	0,988	0,9816
4.0	32.0	10.0	64.0	74,47	0,00181	0,00388	0,9904	0,9797
4.0	256.0	20.0	64.0	419,76	0,08766	0,08561	0,5613	0,5716

From the diagram we understand that a good choice by time and accuracy would be:

PREDICTIONS

1. **Layers 4.0 Fc_units 32.0 Epochs 20.0 Batch_Size 64.0**

CLASSIFICATION REPORT

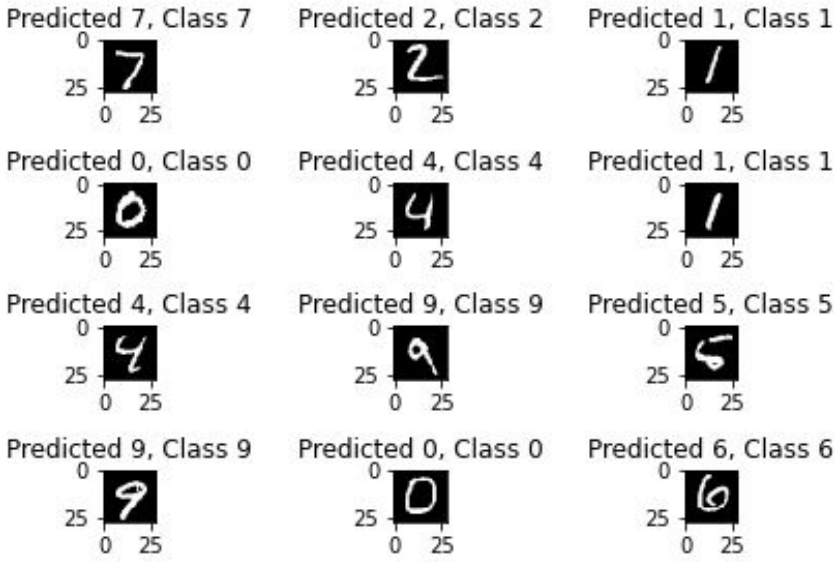
```
Found  9871  correct labels
Found  129  incorrect labels
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       980
           1       0.99      1.00      0.99      1135
           2       0.98      0.99      0.98      1032
           3       0.99      0.99      0.99      1010
           4       0.99      0.99      0.99       982
           5       0.98      0.99      0.98       892
           6       0.99      0.98      0.99       958
           7       0.98      0.99      0.99      1028
           8       0.99      0.98      0.98       974
           9       0.98      0.98      0.98      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```

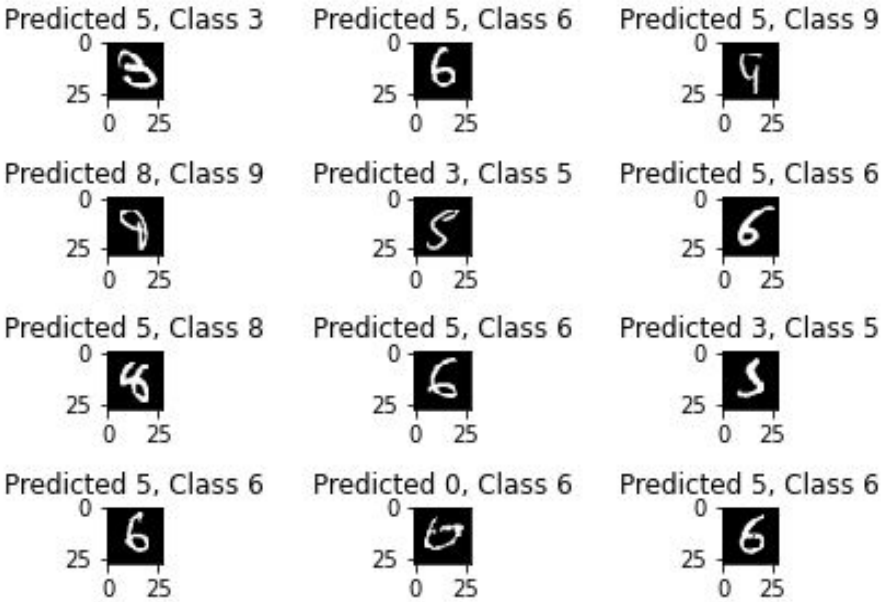
Some labels from this execution:

CORRECT PREDICTIONS



Some labels from this execution:

INCORRECT PREDICTIONS



2. Layers 4.0 Fc_units 32.0 Epochs 100.0 Batch_Size 64.0

REPORT

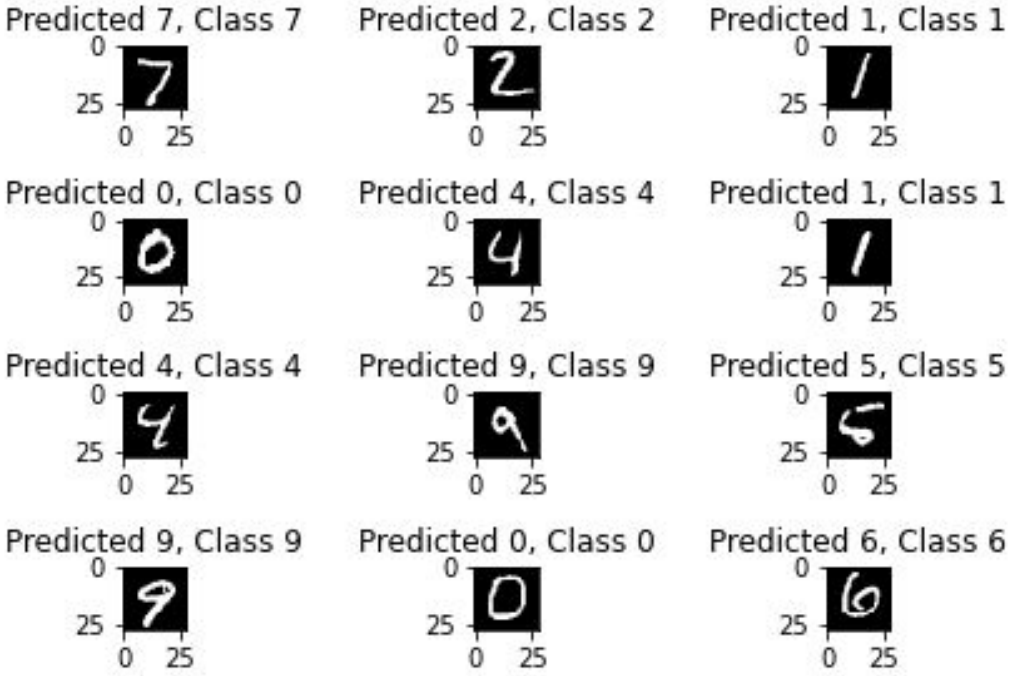
```
Found  9888  correct labels
Found  112  incorrect labels
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       980
           1       0.99      1.00      1.00      1135
           2       0.98      0.99      0.99      1032
           3       0.99      0.99      0.99      1010
           4       0.99      0.98      0.99       982
           5       0.99      0.99      0.99       892
           6       0.99      0.99      0.99       958
           7       0.99      0.98      0.98      1028
           8       0.99      0.99      0.99       974
           9       0.99      0.98      0.99      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```

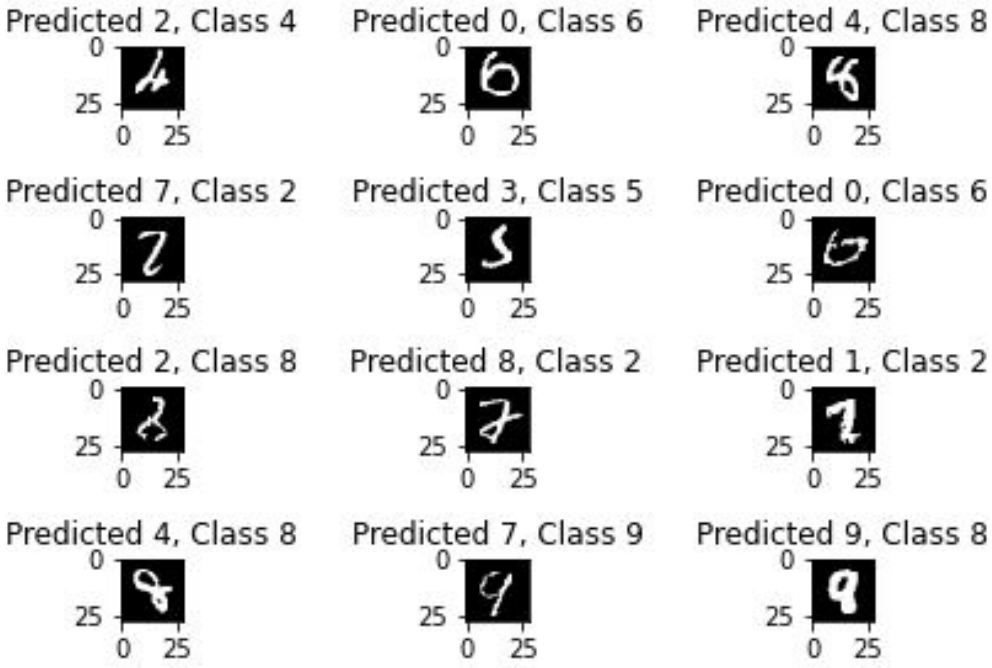
Some labels from this execution:

CORRECT PREDICTIONS



Some labels from this execution:

INCORRECT PREDICTIONS



2. EXPLANATION OF RESULTS

From the experiments, it is observed that the increase of the layers doesn't affect the results in a positive way, but it only increases the training time. Therefore, it would be recommended to keep the value 4. The increase of the number of nodes of fully connected layer more than 128 nodes results in the increase of loss and the decrease of accuracies. Consequently, it is beneficial to keep nodes' value low. So with the combination of the other hyperparameters we choose 32 nodes for our predictions. On the other hand, when the epochs increase, accuracies are improved significantly at the expense of time. That's why we believe that it is useful to stay at 100. In the end, the increase of batch size gains profit on time and an unimportant improvement is observed

concerning the results. We prefer to keep batch size in [64, 128] based on val_accuracy (more efficient value 64).