Practical work 11 - Convolutional Neural Networks

Students

- Liechti Matthieu
- Loup Olivia

1. Learning algorithm to train the neural networks

What is the learning algorithm being used to train the neural networks?

The learning algorithm used is this code is SGD (stochastic gradient descent).

What are the parameters (arguments) being used by that algorithm?

In this code we used the optimizer RMSprop. We also have diffrent parameters like :

- Batch-size = 128
- n_epoch = 20
- Learning_rate = 0.001
- Hidden neuron = 300

What cost function is being used?

The cost function (also referred to as the loss function) being used in your code is categorical crossentropy. The categorical crossentropy loss compares the predicted probability distribution with the true one-hot encoded label and computes the cross-entropy loss, which penalizes the model more if its predicted probability for the correct class is low. `model.compile(loss='categorical_crossentropy', optimizer=RMSprop(), metrics=['accuracy'])`

2 Improvements.

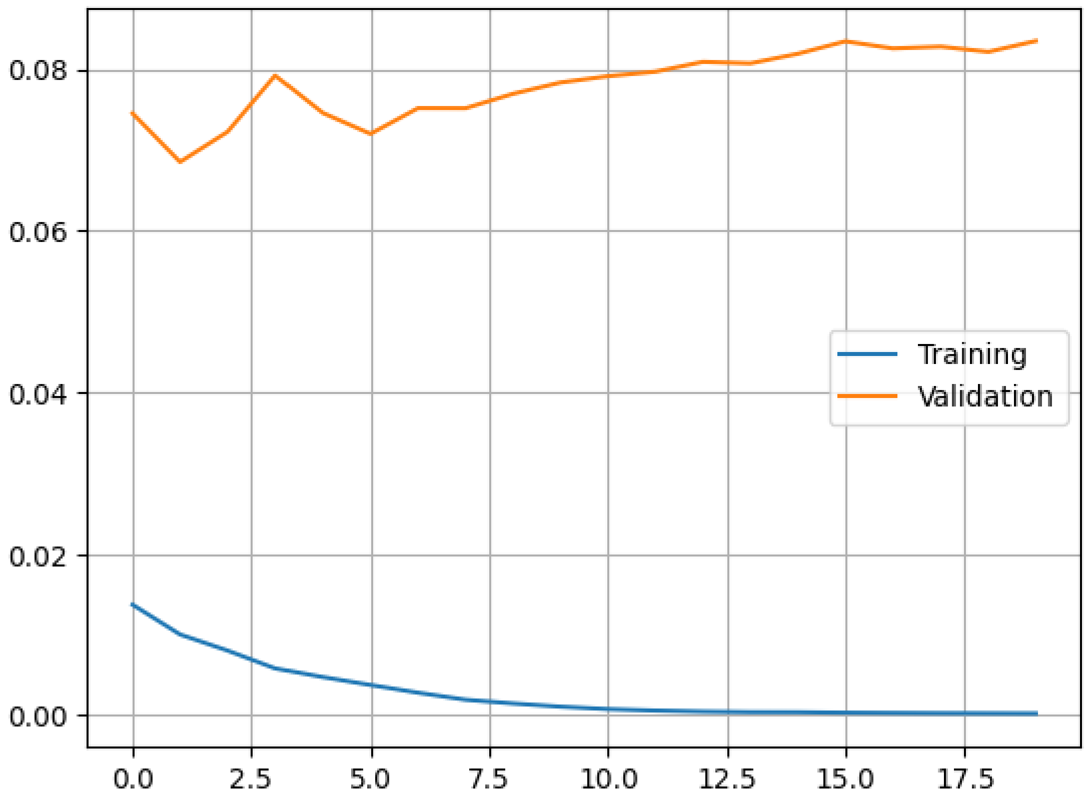
2.1 MLP_Raw

In the first time, we will play with the MLP_Raw notebook. We are going to try diffrent configuration to train this neural network. The first layer input is composed with 784 neurons. It's the number of pixels per image (28x28). Inbetween we have 300 hidden neurons and for the output we have 10 neurons. It correspond to the 10 features (digits) to recognize. First we just run the code without modification. We obtains the following relusts:

- Validation accuracy: 0.9787999987602234
- Test accuracy: 0.9803000092506409

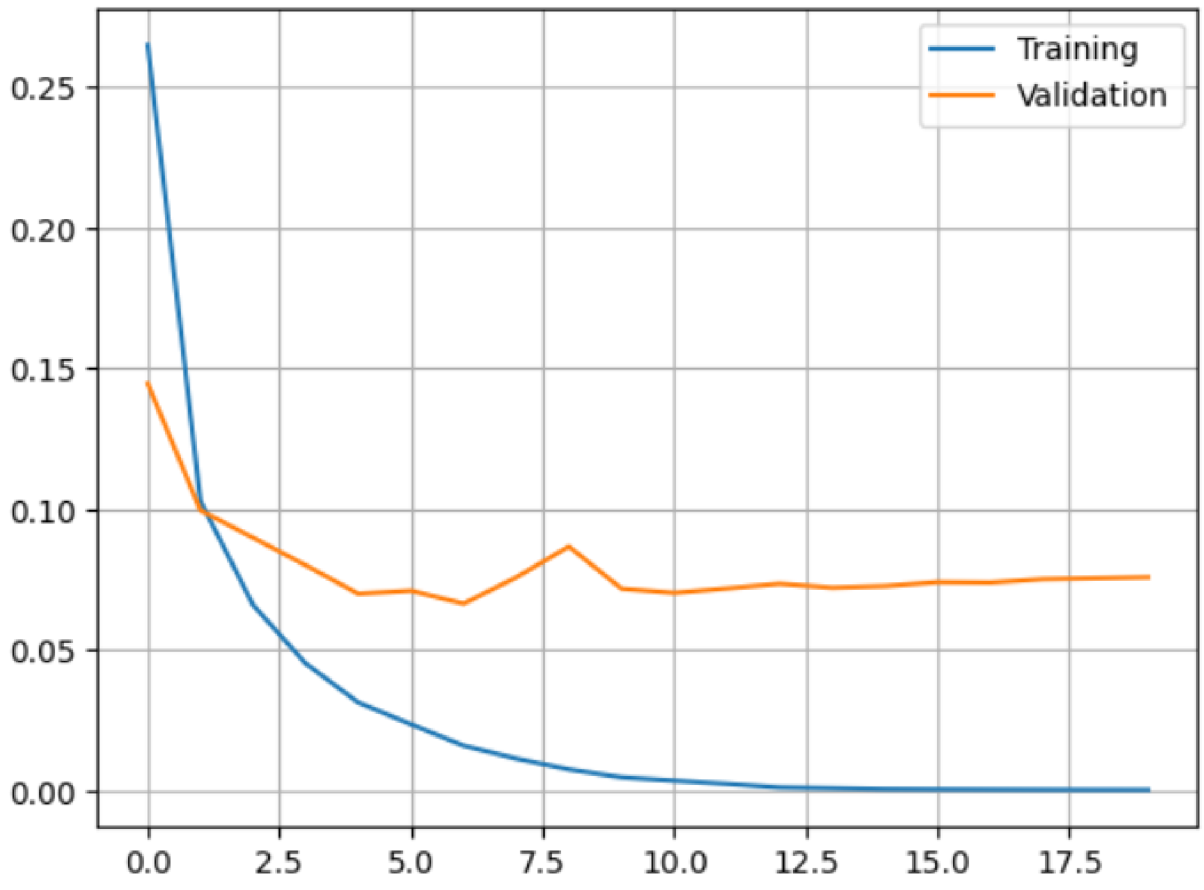
Then we just augment the number of "n_epoch" to 20 and we had a slightly better result :

- Validation accuracy: 0.9825000166893005
- Test accuracy: 0.9822999835014343



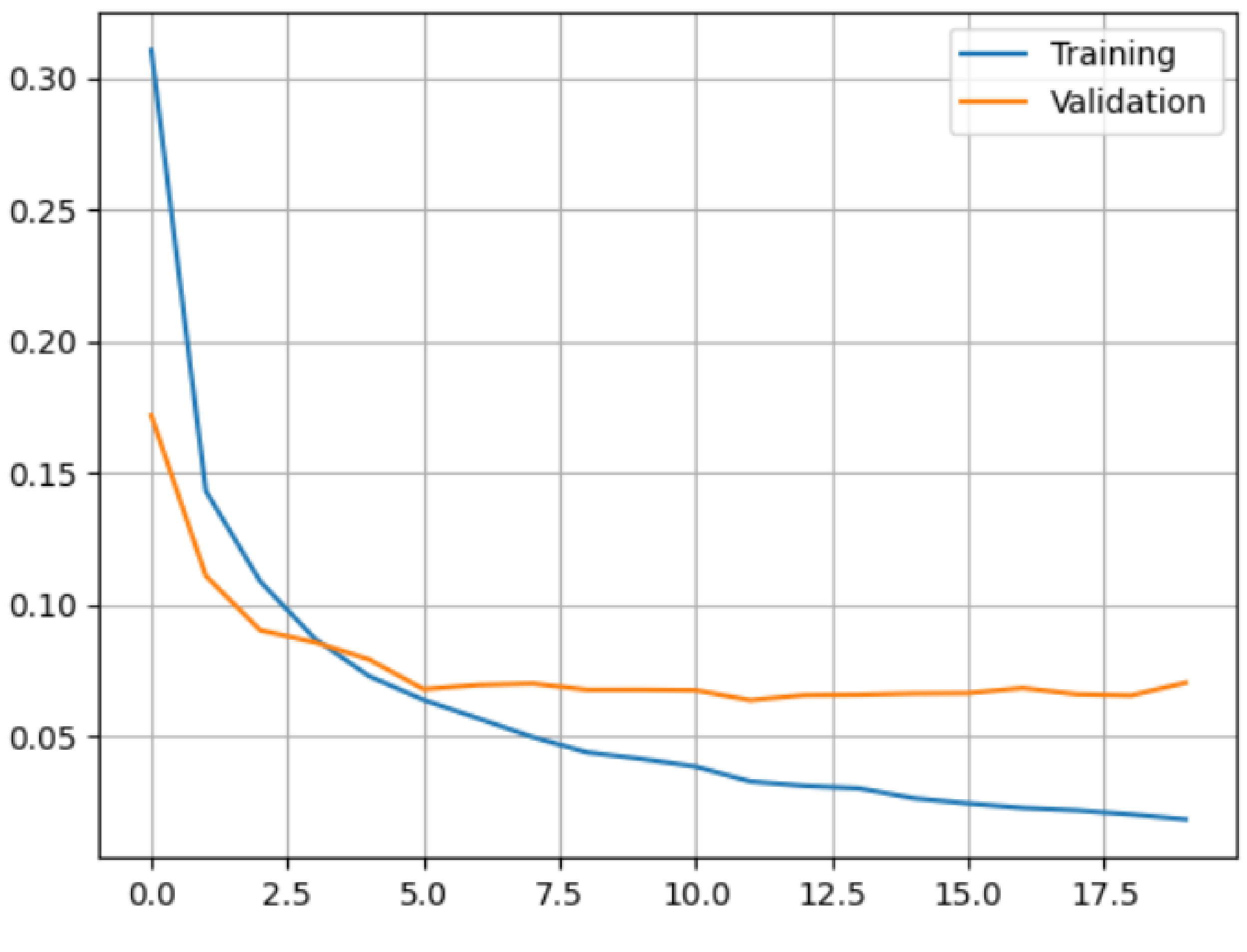
To continue the tests, we modify the number of hidden neurons to 1000.

```
Validation score: 0.07577256113290787
Validation accuracy: 0.9839000105857849
```



We also try to add a dropout layer of 0.5.

```
Validation score: 0.07028599828481674
Validation accuracy: 0.9836000204086304
```

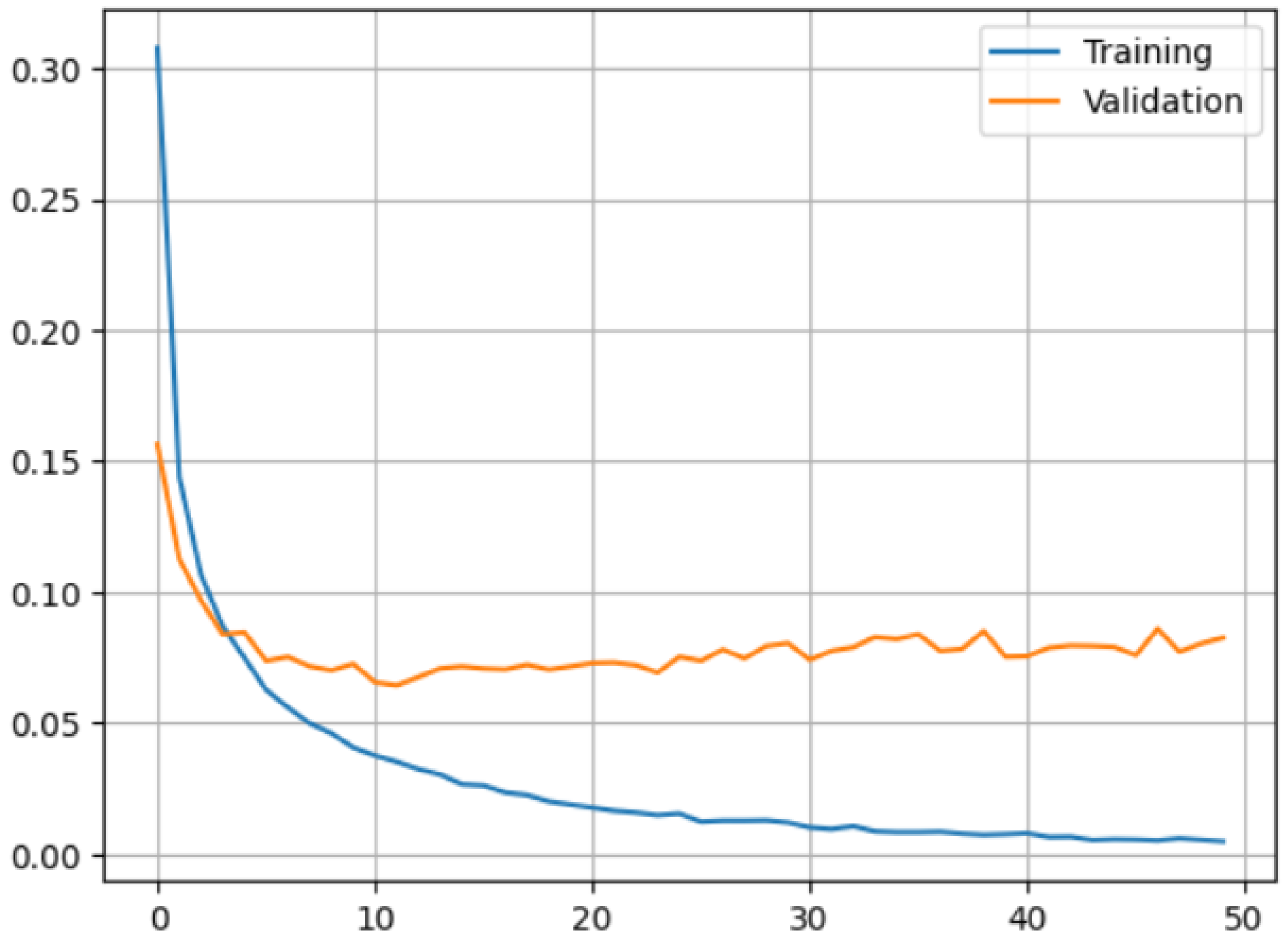


The number of weights in this model can be calculated as follows:

- In the hidden layer, each of the 1000 neurons is connected to each of the 784 neurons neurons. There are therefore 784 × 1000 = 784'000 weights. In addition, there are 1000 bias terms (one for each neuron), i.e. a total of 784'000 + 1000 = 785'000 parameters in the hidden layer.
- In the output layer, each of the neurons (10) is connected to each of the 1000 neurons in the hidden layer. So there are 1000 × 10 weights plus bias terms for each class. each class, giving a total of 1000 × 10 + 10 parameters in the output layer. The total number of parameters in the network is therefore 785,000 + 1000 × 10 + 10 = 795'010. The dropout layer adds no weight to the model.

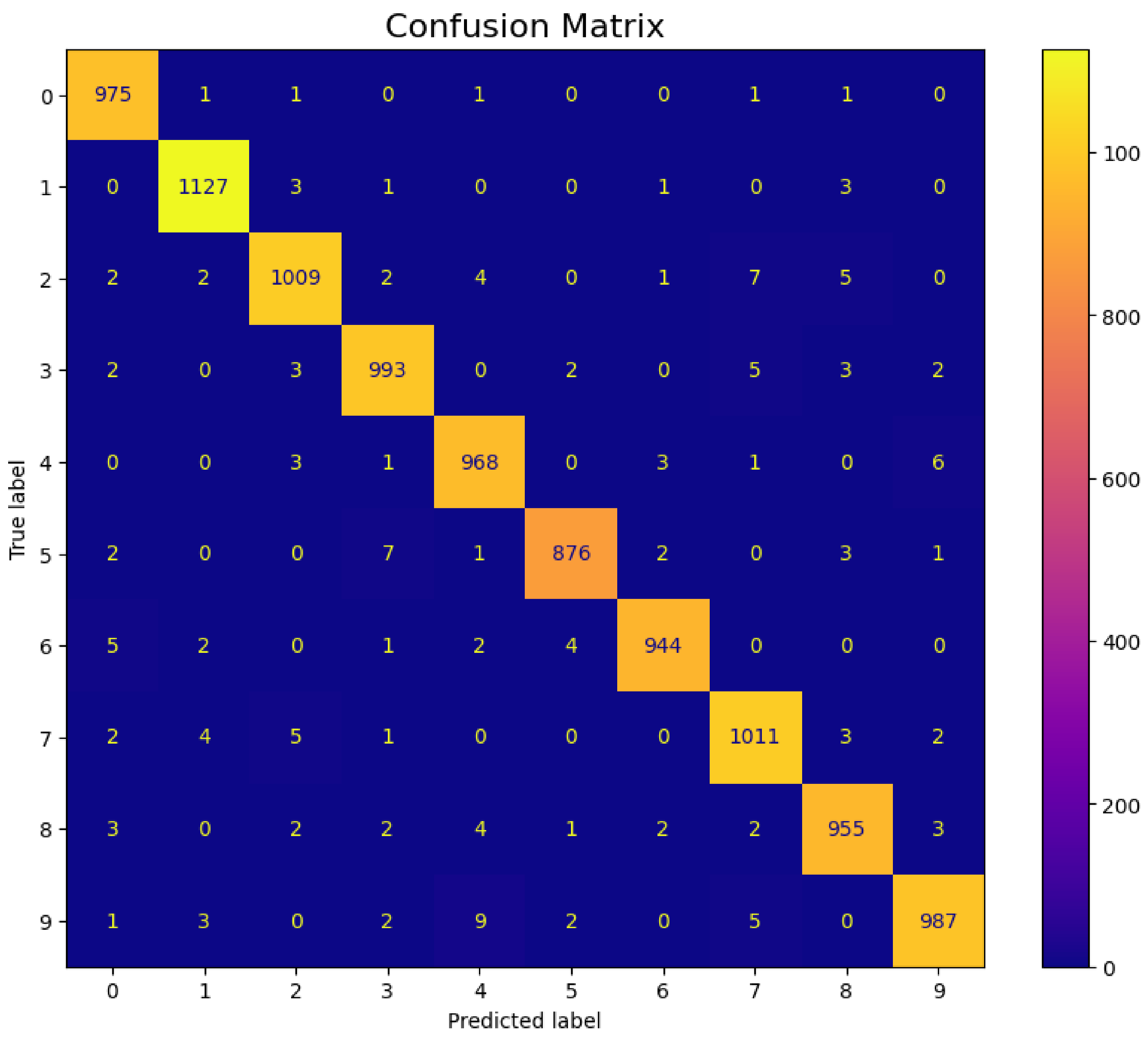
To finish this first notebook we try to augment the number of epoch to 50.

```
Validation score: 0.08252383768558502
Validation accuracy: 0.9846000075340271
```



The final result with the test set is :

- Test score: 0.07585226744413376
- Test accuracy: 0.984499990940094



	0	1	2	3	4	5	6	7	8	9
0	975	1	1	0	1	0	0	1	1	0
1	0	1127	3	1	0	0	1	0	3	0
2	2	2	1009	2	4	0	1	7	5	0
3	2	0	3	993	0	2	0	5	3	2
4	0	0	3	1	968	0	3	1	0	6
5	2	0	0	7	1	876	2	0	3	1
6	5	2	0	1	2	4	944	0	0	0
7	2	4	5	1	0	0	0	1011	3	2
8	3	0	2	2	4	1	2	2	955	3
9	1	3	0	2	9	2	0	5	0	987

The diagonal is visible and reflects the good performance of the model, which had no trouble finding the right digits. There are a few errors, but never more than 10. The biggest confusion seems to be between “9” and “4” + “3” and “5”, without appearing high.

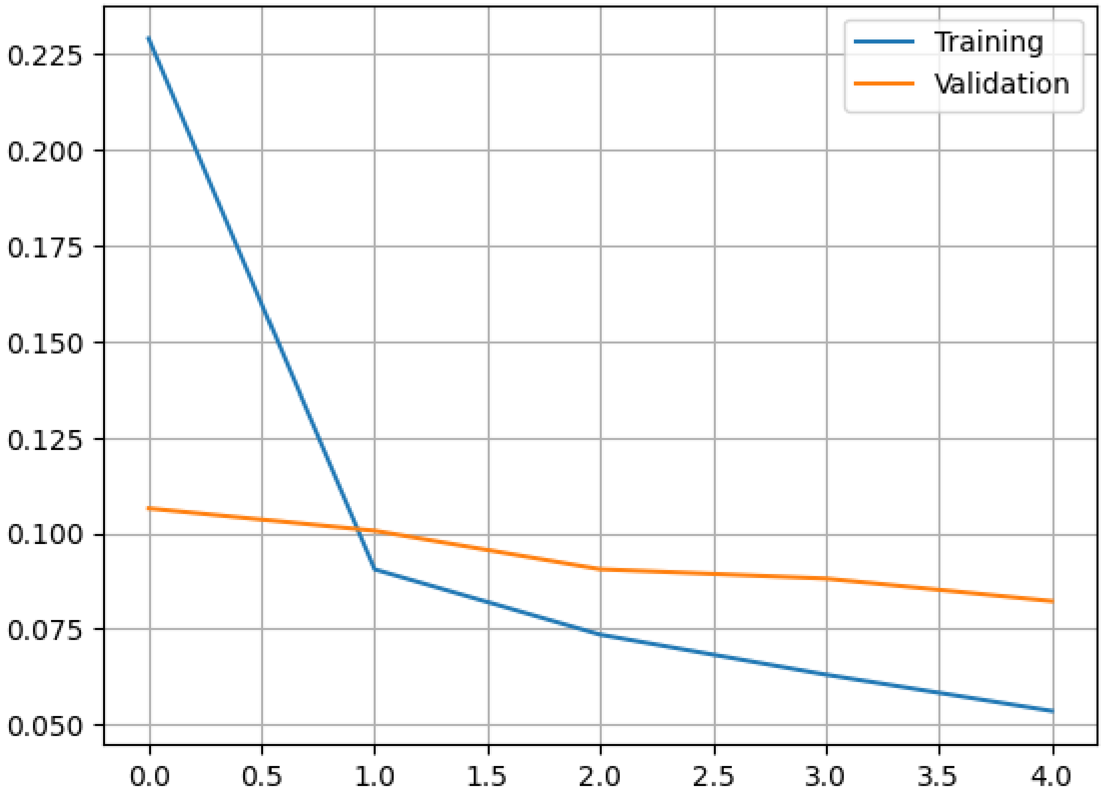
2.2 MLP_HOG

Like the previous notebook, the dataset used is MNIST (digits between 0 and 9). However, this time the HOG feature method is applied to recognize image features rather than raw inputs (28x28) directly. The same sets and their distribution are similar to Exercise 1. The starting parameters are the following :

- Pixel per Cell : 4
- Orientation : 8
- Epoch : 5
- Batch size : 128
- Hidden neurons : 300
- Optimizer : RMSprop

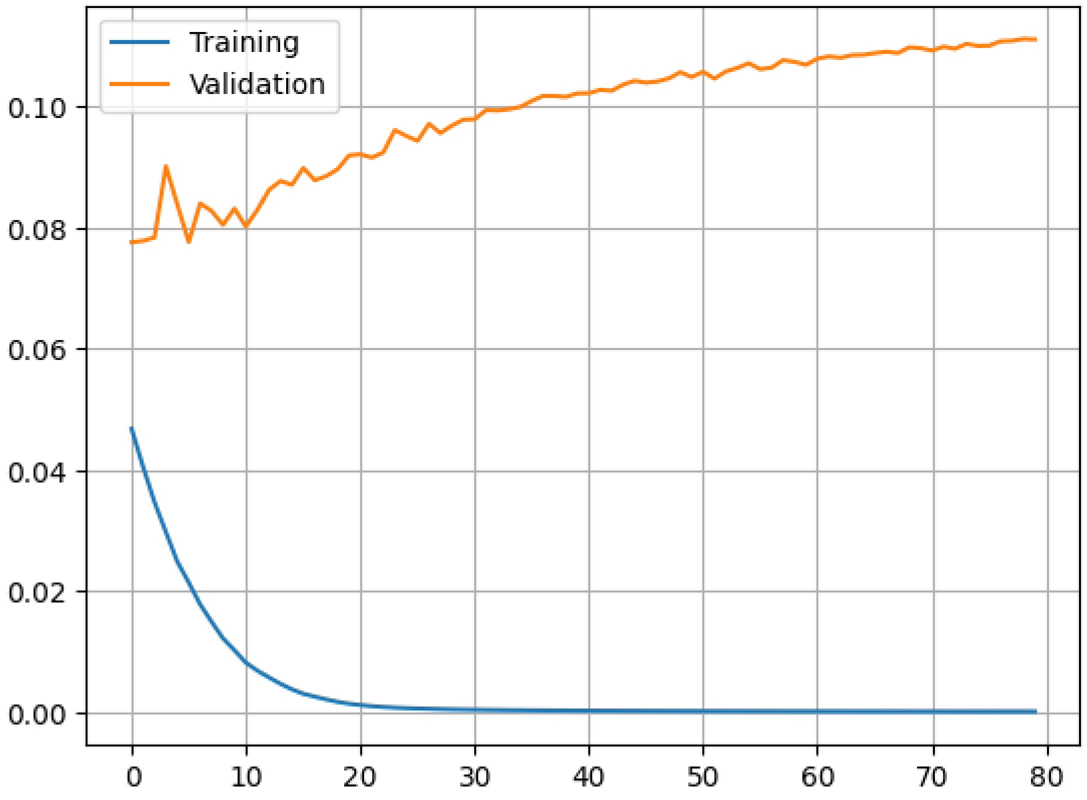
The model is a simple MLP (feed-forward network) with a hidden layer layer and an output layer using the softmax activation function. For the input we have a vector size of 1568 and the number ouf outpu is equal to the number of features.

The result of the first run shows an accuracy on the validation set of 0.9743000268936157.



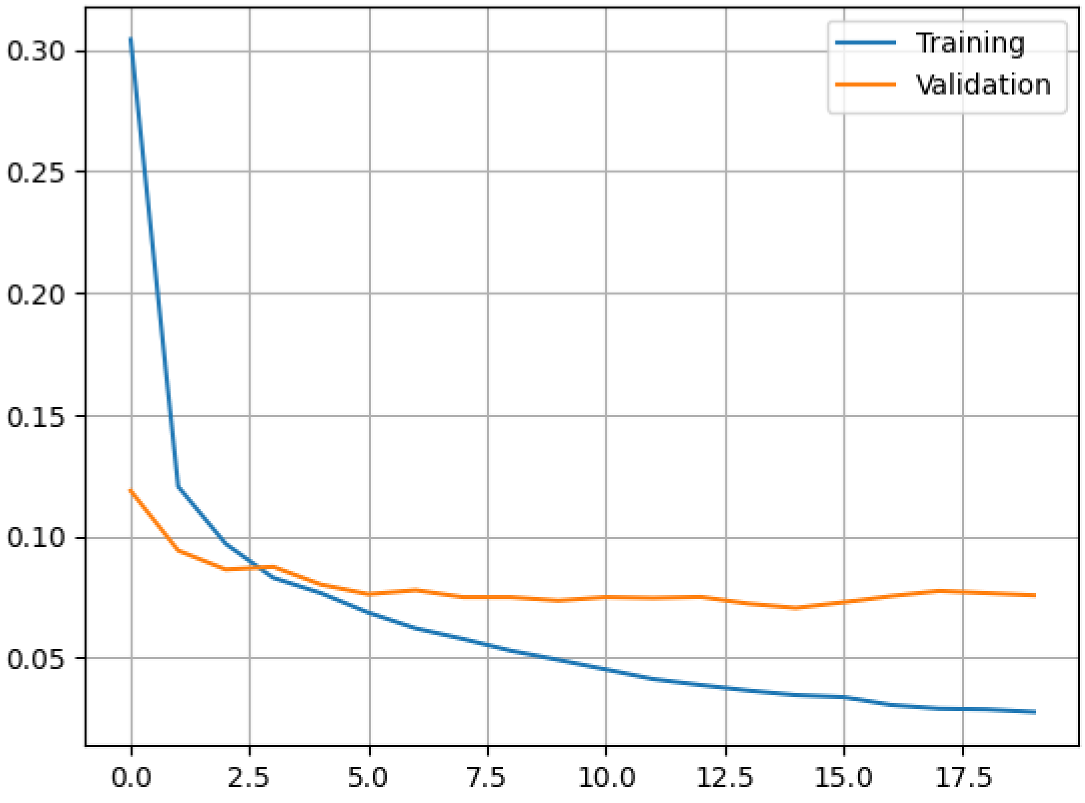
As in the last Notebook, we modify the number of Epoch.

```
Validation score: 0.1111113652586937
Validation accuracy: 0.9800000190734863
```



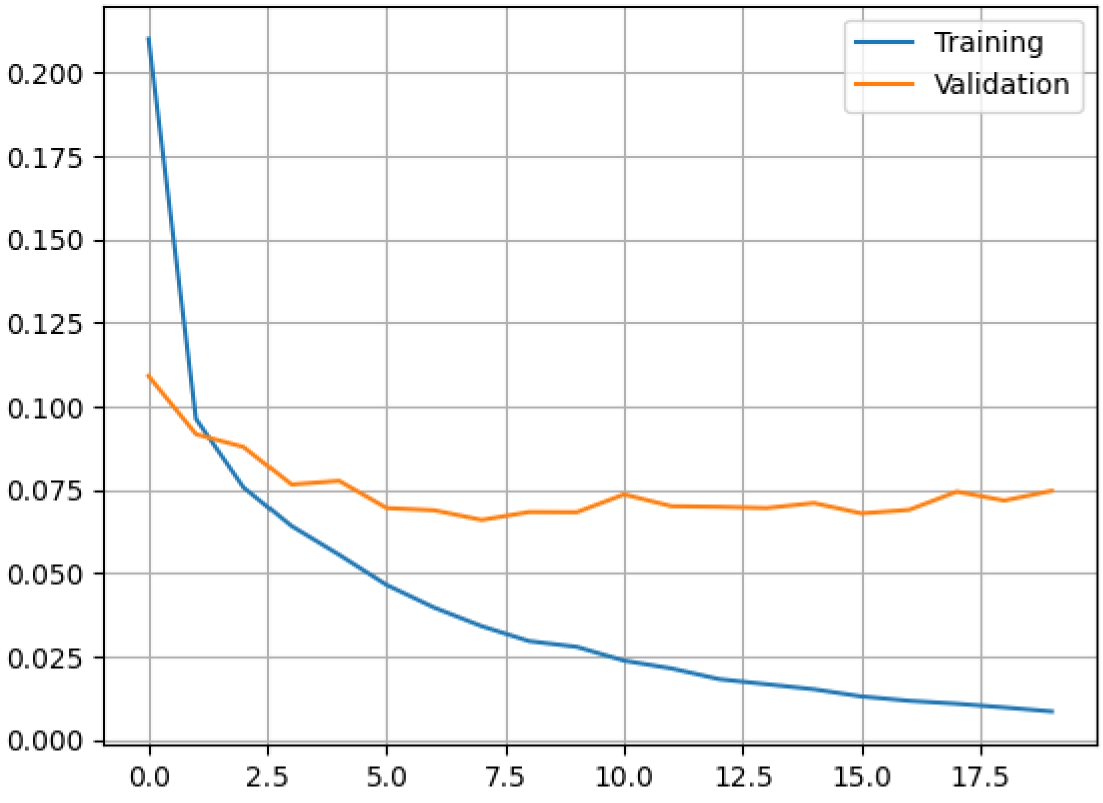
For a number of Epoch of 80, we can clearly see the convergence but we can see tha the validation curve tend to augment. It probably mean that we have little overfitting. For the following test we reduce the number of Epoch and we add a dropout to the model.(Epoch = 20)

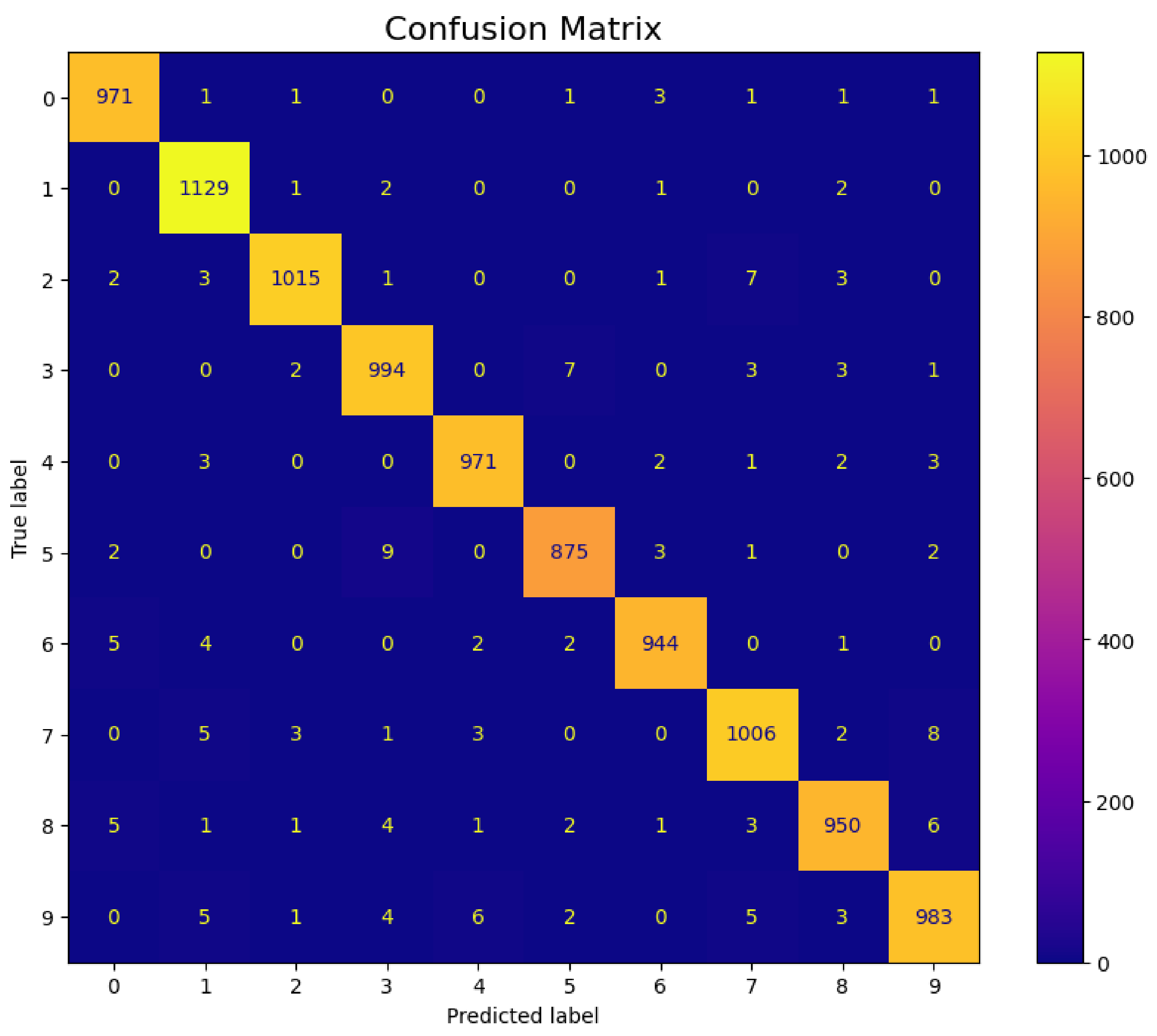
```
Validation score: 0.07568573206663132
Validation accuracy: 0.9807000160217285
```



The accuracy is very close to the last test. To finish we juste augment the number of hidden neurons. (1000)

```
Validation score: 0.07465364038944244
Validation accuracy: 0.9821000099182129
```





We can see that we have almost the same result as the MLP_Raw. The final accuracy on the test Set is :0.9837999939918518. The confusion matrix show the same caracteristics as the last one. The calculation is carried out in exactly the same way as for the raw inputs with 1568 inputs instead of 784. The result is given by the following calculation: 1568 × 1000 + 1000 + 1000 × 10 + 10 = 1'579'010 parameters.

We also tried to variate the parameter Pix_p_cell to 7 to see the diffrences. The result was worse than the last config. we obtain the following reslut:

- Validation accuracy: 0.9689000248908997
- Test accuracy: 0.9725000262260437

The test accuracy is 1% less thant the last with 4 Pix_p_Cell.

2.3 CNN

Here, we will describe the CNN notebook. the notebooks supplied included an initial configuration of hyperparameters hyperparameters for training on the MNIST dataset, which we used as a starting point. The details and performance of this basic configuration are as follows:

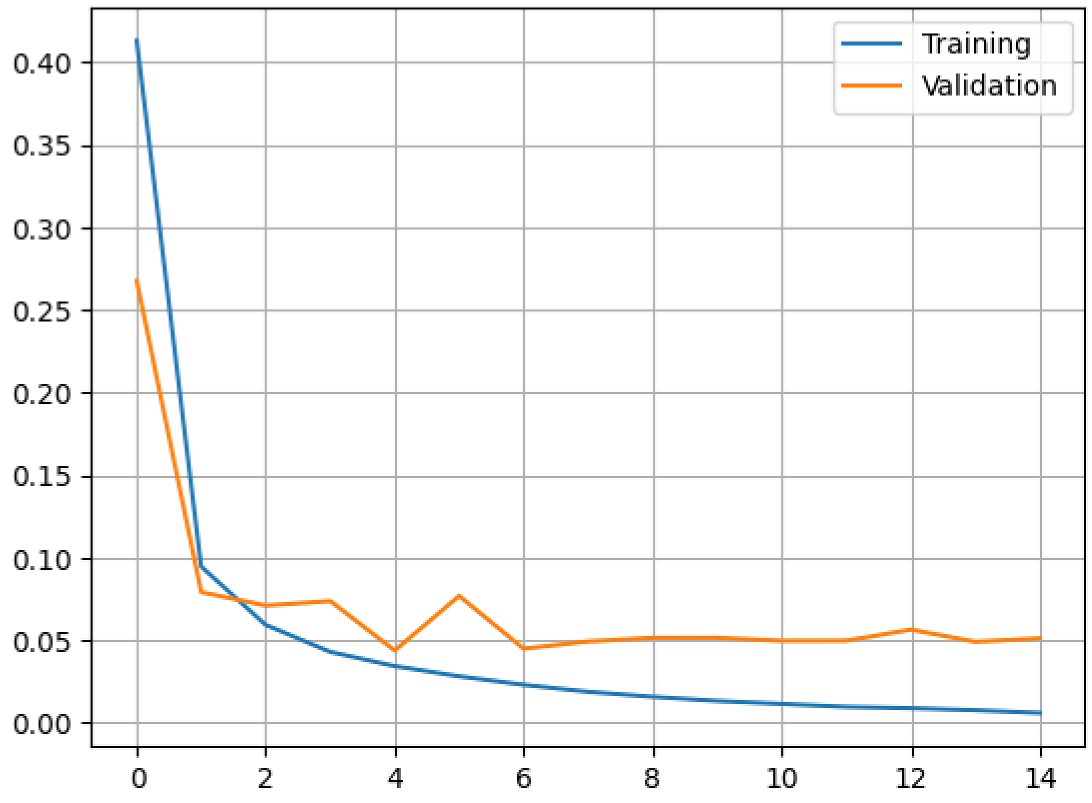
- Epoch : 5

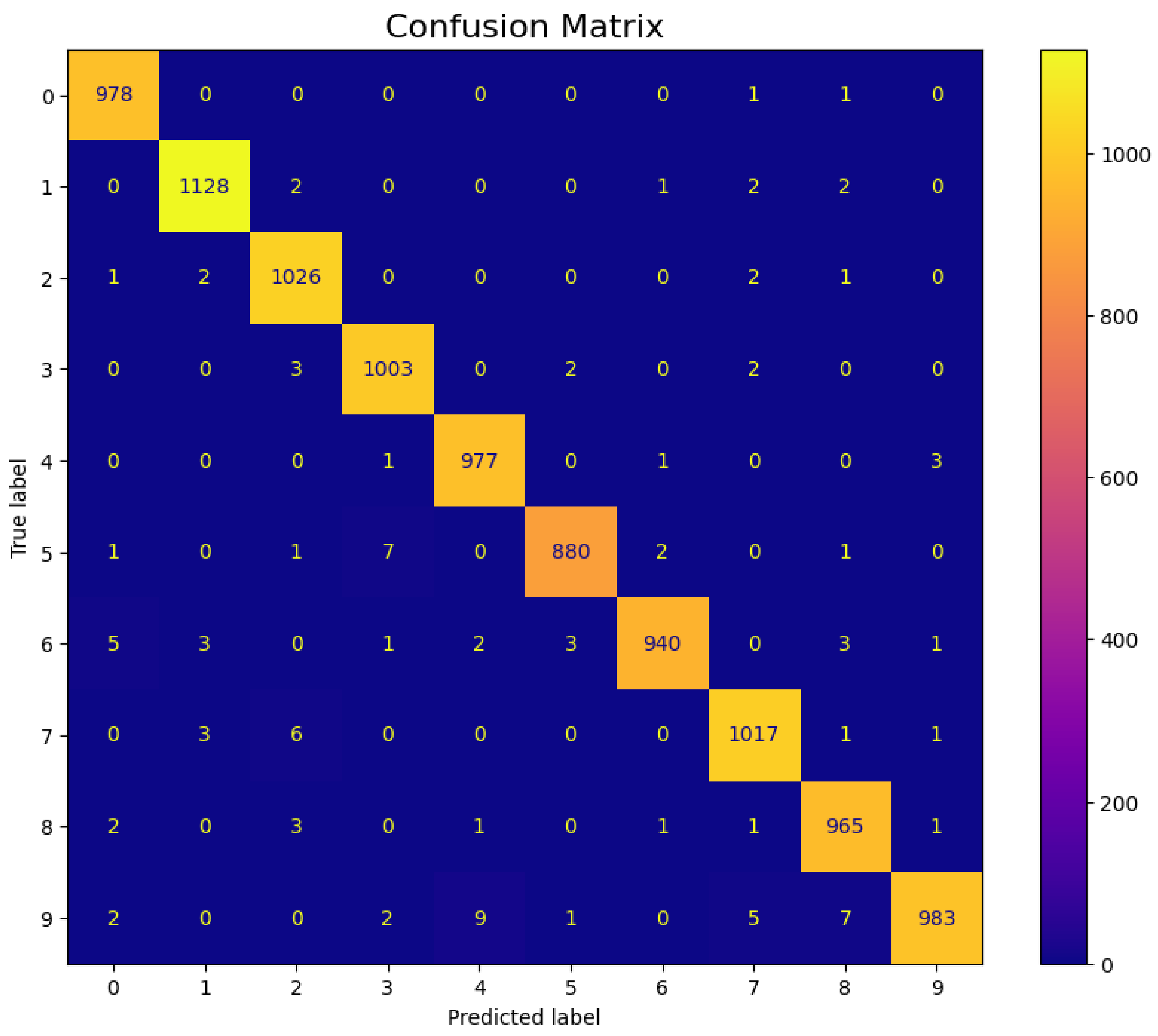
- Batch size : 128
- Learning rate : 0.001
- Optimizer : RMSprop

The model consists of three convolutional layers with max pooling, followed by a flatten layer and two dense layers. The last dense layer uses a softmax activation function for multiclass classification (0 to 9 for MNIST). The input is a shape tensor (w, h, 1), which means that these are grayscale images images with a certain height and width. In our case, it's 28x28. As always, the output is the ten digits features.

The initial test set accuracy is equal to 98.6% We test a lot of variation and the best we found is the next one. We modify the filter of "l1" and "l2" to 32 and we augment the core size to 7x7 for thos two lignes. We also augment the number of epoch to 15. The final result is 99% for the validation accuracy and 98.97% for the test set accuracy.

```
Validation score: 0.05126409977674484
Validation accuracy: 0.9900000095367432
```





	0	1	2	3	4	5	6	7	8	9
0	978	0	0	0	0	0	0	1	1	0
1	0	1128	2	0	0	0	1	2	2	0
2	1	2	1026	0	0	0	0	2	1	0
3	0	0	3	1003	0	2	0	2	0	0
4	0	0	0	1	977	0	1	0	0	3
5	1	0	1	7	0	880	2	0	1	0
6	5	3	0	1	2	3	940	0	3	1
7	0	3	6	0	0	0	0	1017	1	1
8	2	0	3	0	1	0	1	1	965	1
9	2	0	0	2	9	1	0	5	7	983

We have found that this system works relatively well. However, we can observe slight deviations in the following recognitions. In a minority of cases, 4 and 8 are confused with 9. The 3 with 5 and 2 with 7. To calculate the number of weights, we need to take into account the weights in the weights in Conv2D layers (filters and biases) and weights in dense layers (including including bias). This can be calculated by adding the product of the dimensions of the weights of each layer and adding the number of biases for each layer (see theory).

- Layer l1 : $32 \times (7 \times 7 + 1) = 1600$
- Layer l2 : $32 \times (7 \times 7 \times 32 + 1) = 50208$
- Layer l3 : $16 \times (3 \times 3 \times 32 + 1) = 4624$
- Layer l4 : $144 \times 25 + 25 = 3625$
- Layer l5 : $25 \times 10 + 10 = 260$

The sum is equal to 60317.

3. Shallow ones VS deep neural network

Do the deep neural networks have much more “capacity” (i.e., do they have more weights?) than the shallow ones? explain with one example.

The shallow are limited by the number of layer, in case this bring less capacity to learning than the deep neural network. Moreover by this limit of numbers of layer, this system could have less weight than the deep neural network.

In example with an object detection in image, shallow ones will be limited with one layer and detected multiple layer to find an object could be complicate if the image is to "rich". With the case of deep neural network, this could be easier because it could use more layer so have more weight and more risk of overfitting.

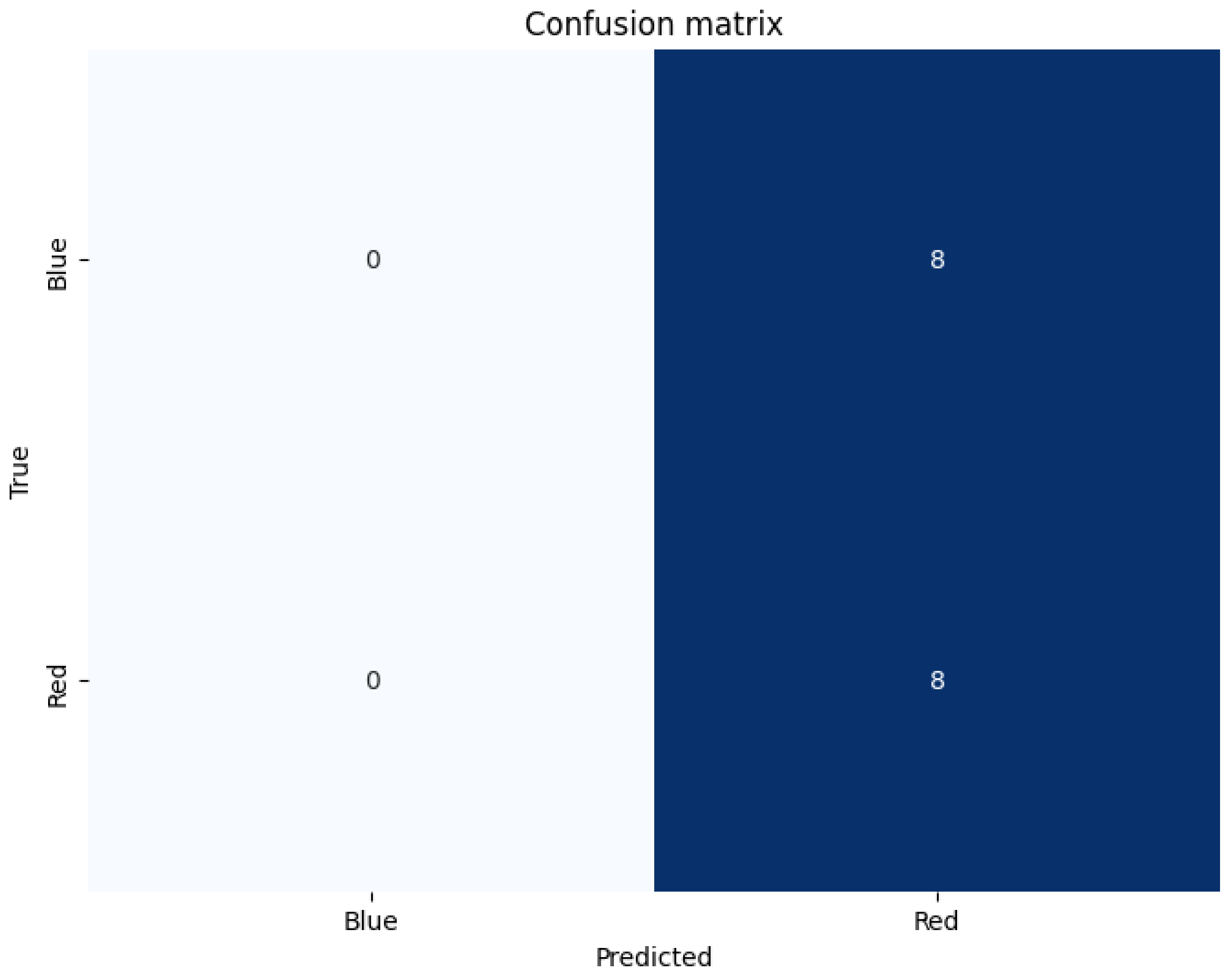
4. CNN with pneumonia

Train a CNN for the chest x-ray pneumonia recognition. In order to do so, complete the code to reproduce the architecture plotted in the notebook. Present the confusion matrix, accuracy and F1-score of the validation and test datasets and discuss your results.

validation set

The F1-score are : 0.667
It's not really the best. When we see the number of data, it's few values. Maybe use more could be more relevant. This could be a result of the number of epoch or layer.

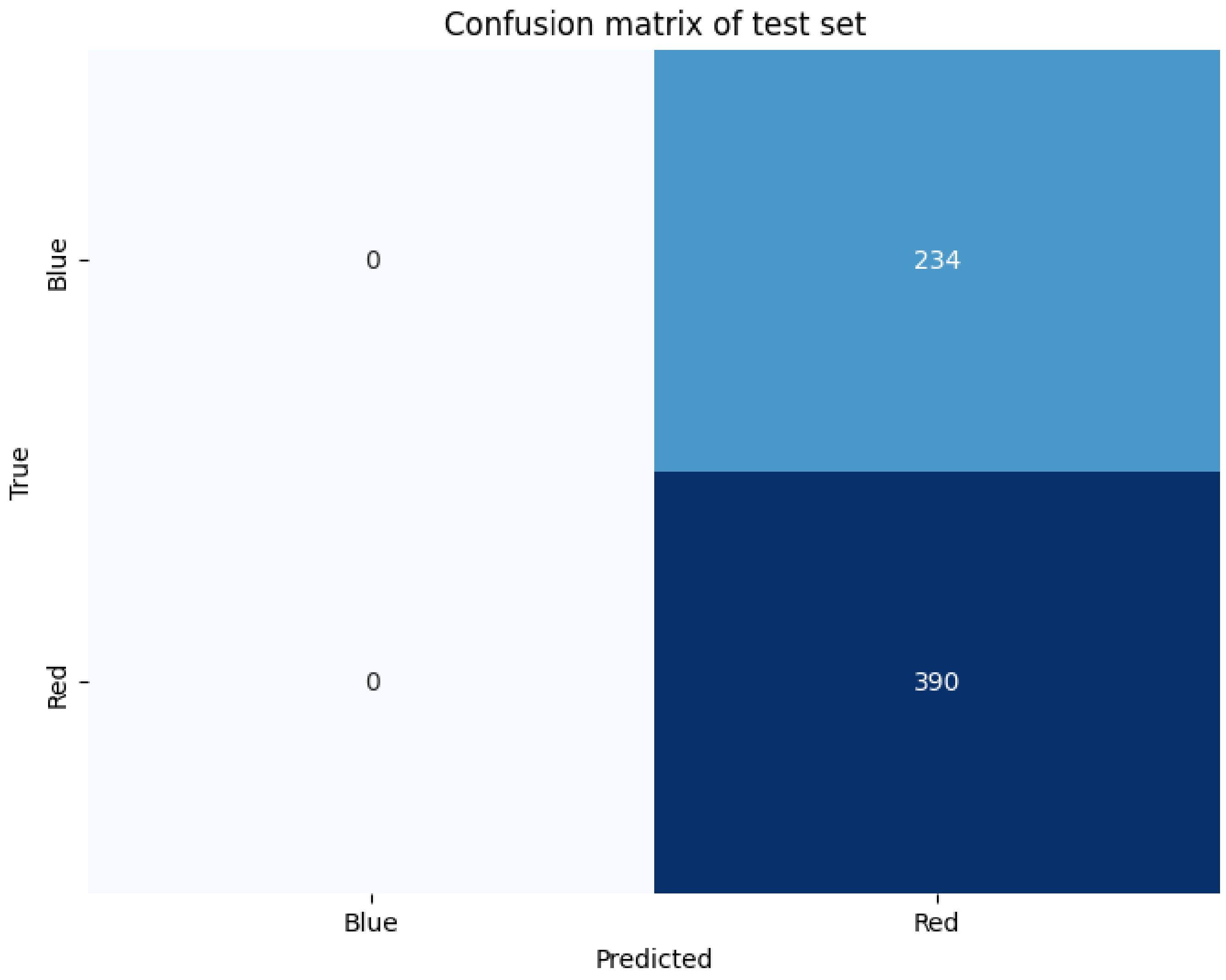
Matrix :



test set

The F1-score are : 0.769 The score are better.

Matrix :



validation set VS test set

The F1-score of test set are bigger than validation set. It could have a relation with the number of parameters. In général, the score are good but the best. It could be interesting to use more epoch or change the layer. The risk is to bring overfitting and reduce sscore. Moreover, the time to wait about model could be really big.