ARN - Report - Labo04

Anthony Coke, Guilain Mbayo, Mehdi Salhi

May 11, 2022

Contents

Learning algorithm

> 1. What is the learning algorithm being used to optimize the weights of the neural networks? What are the parameters (arguments) being used by that algorithm? What cost function is being used ? please, give the equation(s)

> MLP_from_raw_data.ipynb

The algorithm used is RMSprop.

The arguments used by this algorithm are: - Learning rate A Tensor, floating point value, or a schedule that is a tf.keras.optimizers.schedules.LearningRateSchedule, or a callable that takes no arguments and returns the actual value to use. The learning rate. Defaults to 0.001. - rho: Discounting factor for the history/coming gradient. Defaults to 0.9. - momentum: A scalar or a scalar Tensor. Defaults to 0.0. - epsilon: A small constant for numerical stability. This epsilon is “epsilon hat” in the Kingma and Ba paper (in the formula just before Section 2.1), not the epsilon in Algorithm 1 of the paper. Defaults to 1e-7. - centered: Boolean. If True, gradients are normalized by the estimated variance of the gradient; if False, by the uncentered second moment. Setting this to True may help with training, but is slightly more expensive in terms of computation and memory. Defaults to False. - name: Optional name prefix for the operations created when applying gradients. Defaults to “RMSprop”. - **kwargs: keyword arguments. Allowed arguments are clipvalue, clipnorm, global_clipnorm. If clipvalue (float) is set, the gradient of each weight is clipped to be no higher than this value. If clipnorm (float) is set, the gradient of each weight is individually clipped so that its norm is no higher than this value. If global_clipnorm (float) is set the gradient of all weights is clipped so that their global norm is no higher than this value.

The used cost function is the categorical crossentropy function. It’s equation is:

$$\text{Loss} = -\sum_{i=1}^{\substack{\text{output}\\ \text{size}}} y_i \cdot \log \hat{y}_i$$

Figure 1: ARN-Labo04-CrossEntrEquation

Model Complexity

> 2. Model complexity: for each experiment (shallow network learning from raw data, shallow network learning from features, CNN, and Fashion MNIST), select a neural network topology and describe the inputs, indicate how many are they, and how many outputs. Compute

> the number of weights of each model (e.g., how many weights between the input and the hidden layer, how many weights between each pair of layers, biases, etc..) and explain how do you get to the total number of weights.

> MLP_from_raw_data.ipynb Inputs: Outputs: Activation function: tanh Activation function for output layer: softmax Neurons in hidden layer: 250 Batch size: 4096 Dropout: 0.5 Number of epoch: 150

> MLP_from_HOG.ipynb

> CNN.ipynb

> Fashion_MNIST.ipynb

Deep Neural Networks

> 3. 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 deep neural network have more hidden layer than the shallow ones, but it doesn't necessary mean that it has more neurons in it. For exemple, in this lab we use 300 neurons in the hidden layer for the shallows network (raw_data and HOG), against only 25 neurons for the deep one (CNN). The deep neural networks have more capacity, because they usually need less components to achieve the same goal or better than a shallow neural network. If we compare the weights of each model, the shallow one will have more weight than the deep one. For exemple, a model with 2 entries, 6 neurons in one hidden layer and 2 output, we get 2 * 6 + 6 * 2 = 24 links that have each their weight. For the same model but with 3 hidden layers, we got 2 * 2 + 2 * 2 + 2 * 2 + 2 * 2 = 16 links, and so 16 weights.

Tests

> 4. Test every notebook for at least three different meaningful cases (e.g., for the MLP exploiting raw data, test different models varying the number of hidden neurons, for the feature-based model, test pix_p_cell 4 and 7, and number of orientations or number of hidden neurons, for the CNN, try different number of neurons in the feed-forward part) describe the model and present the performance of the system (e.g., plot of the evolution of the error, final evaluation scores and confusion matrices). Comment the differences in results. Are there

> particular digits that are frequently confused?

> MLP_from_raw_data.ipynb

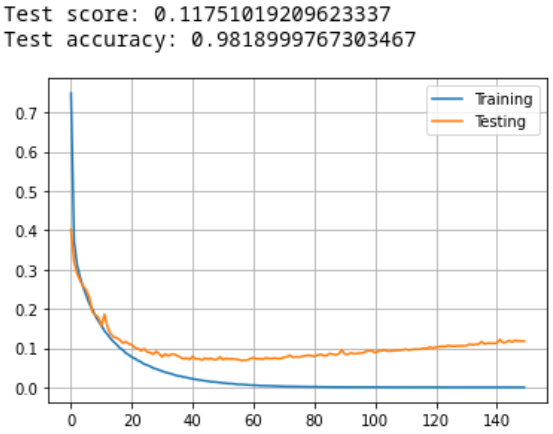


Figure 2: ARN-RAW-Plot-tanh-softmax_Batch2048_NoDropout_Epoch150

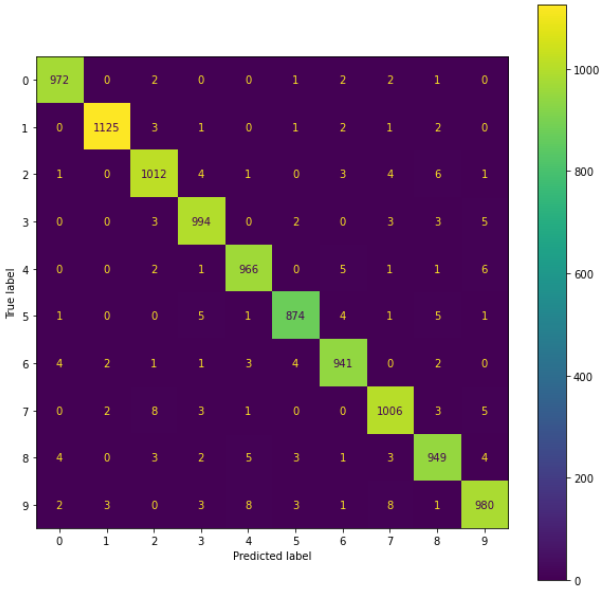


Figure 3: ARN-RAW-ConfMat-tanh-softmax_Batch2048_NoDropout_Epoch150

We can see in this experiment that there's clearly an overfitting.

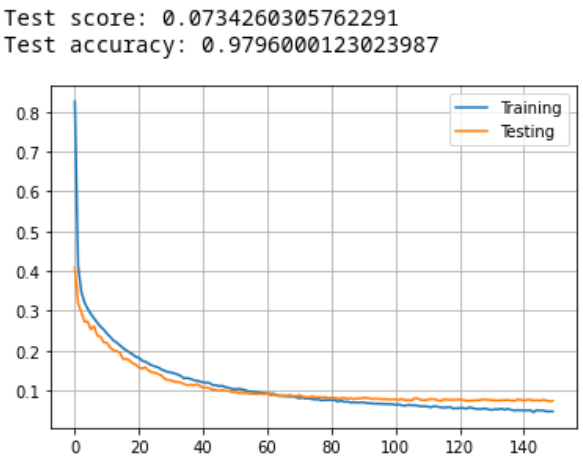


Figure 4: ARN-RAW-Plot-tanh-softmax_Batch2048_Dropout_Epoch150

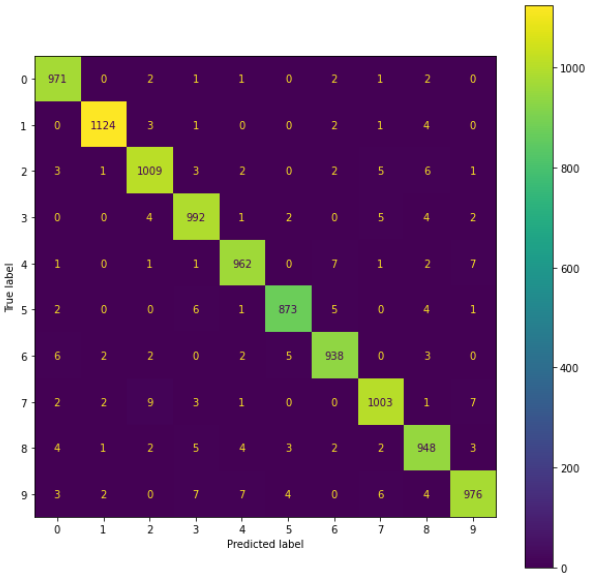


Figure 5: ARN-RAW-ConfMat-tanh-softmax_Batch2048_Dropout_Epoch150

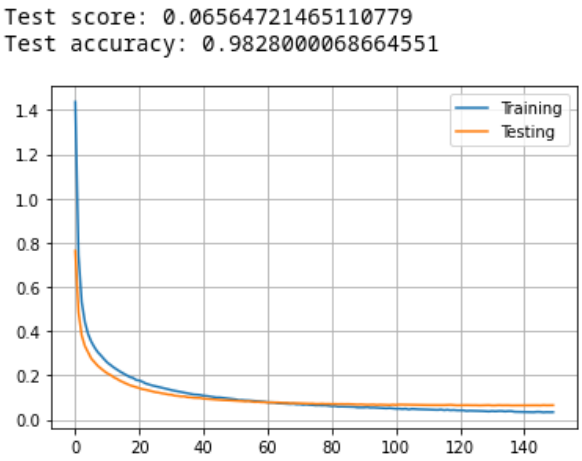


Figure 6: ARN-RAW-Plot-sigmoid-softmax_Batch2048_Dropout_Epoch150

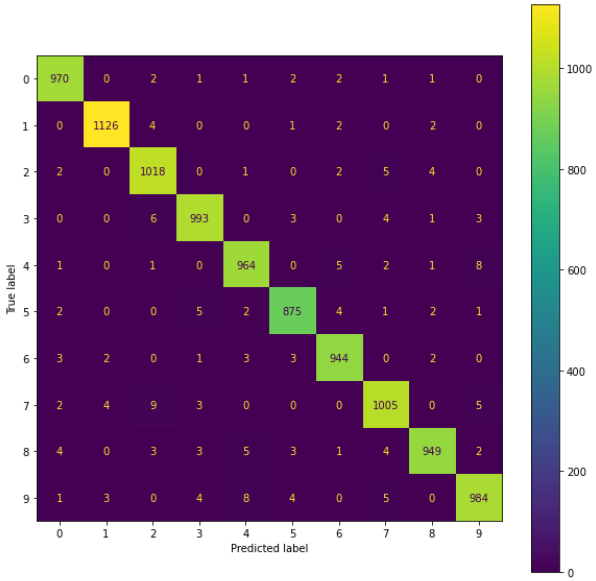


Figure 7: ARN-RAW-ConfMat-sigmoid-softmax_Batch2048_Droptout_Epoch150

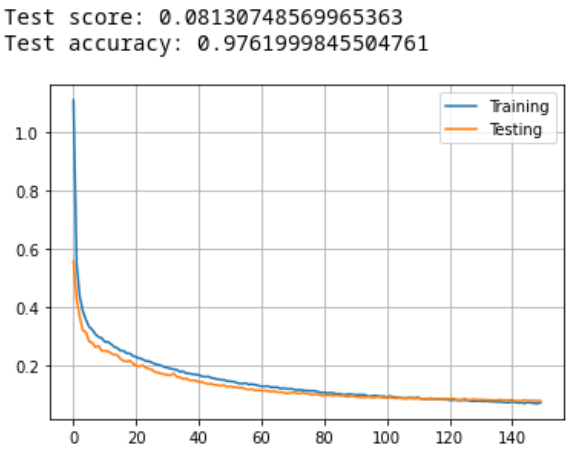


Figure 8: ARN-RAW-Plot-tanh-softmax-Neur250_Batch4096_Dropout_Epoch150

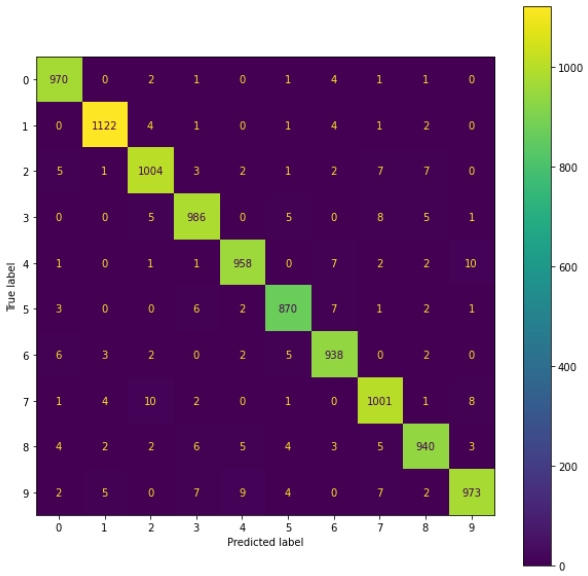


Figure 9: ARN-RAW-ConfMat-tanh-softmax-Neur250_Batch4096_Dropout_Epoch150

```
Test score: 0.06945059448480606
Test accuracy: 0.9797000288963318
```

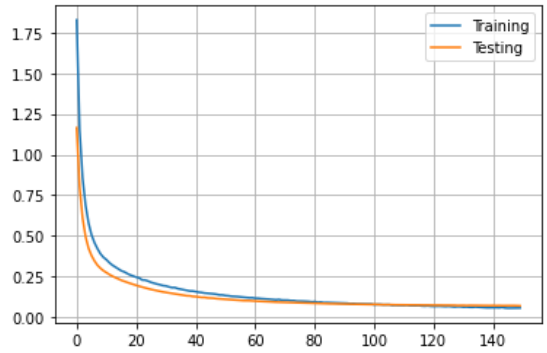


Figure 10: ARN-RAW-Plot-sigmoid-softmax-Neur250_Batch4096_Dropout_Epoch150

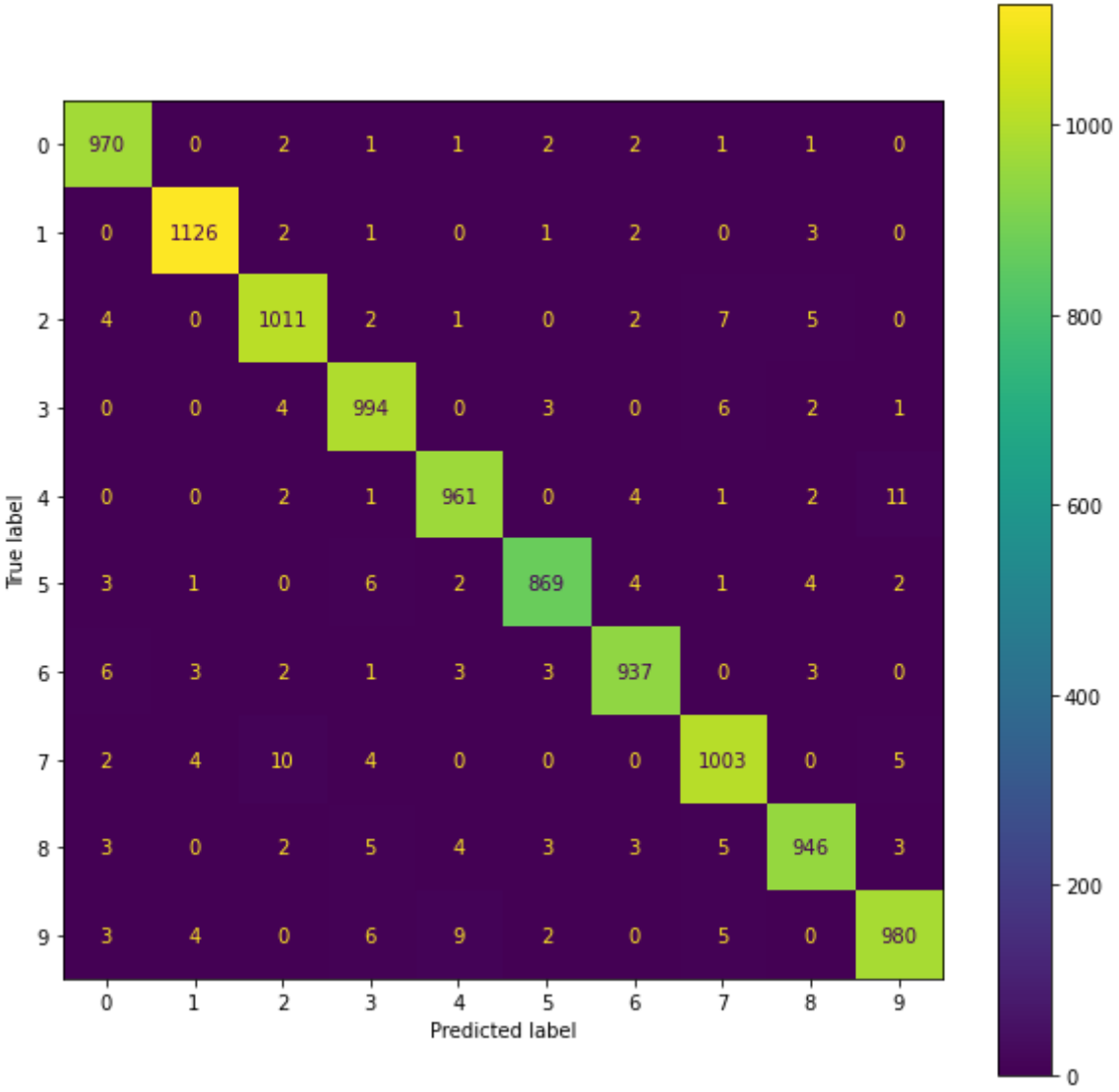


Figure 11: ARN-RAW-ConfMax-sigmoid-softmax-Neur250_Batch4096_Dropout_Epoch150

```
Test score: 0.06911627948284149
Test accuracy: 0.983299970626831
```

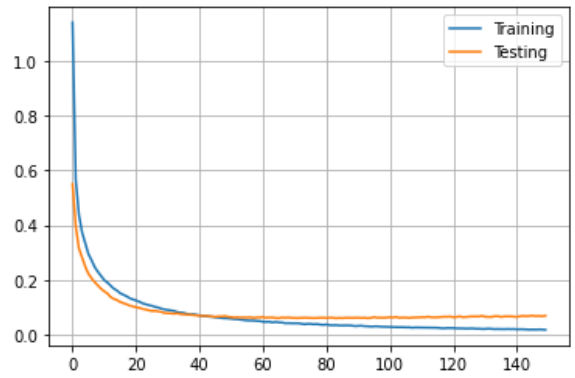


Figure 12: ARN-RAW-Plot-relu_softmax-Neur250_Batch4096_Dropout_Epoch150

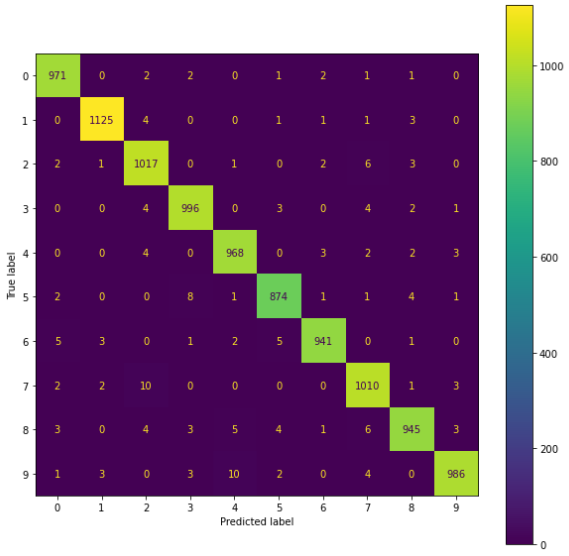


Figure 13: ARN-RAW-ConfMat-relu-softmax-Neur250_Batch4096_Dropout_Epoch150

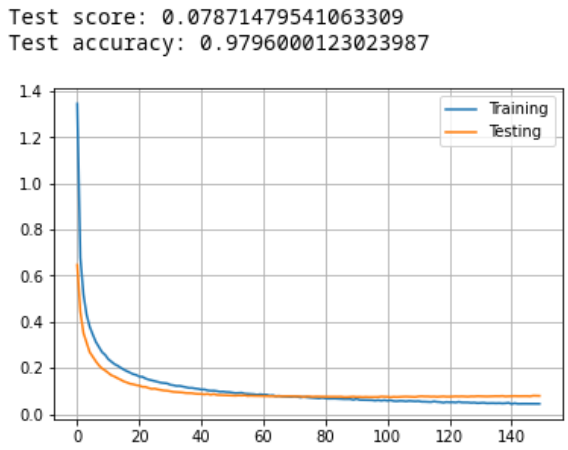


Figure 14: ARN-RAW-Plot-relu-softmax-Neur150_Batch4096_Dropout_Epoch150

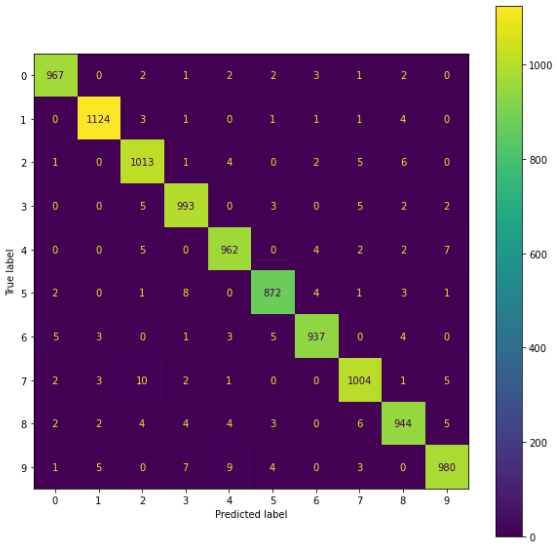


Figure 15: ARN-RAW-ConfMat-relu-softmax-Neur150_Batch4096_Dropout_Epoch150

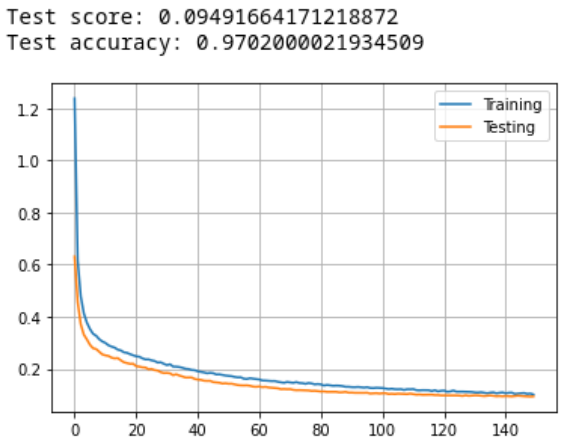


Figure 16: ARN-RAW-Plot-tanh-softmax-Neur150_Batch4096_Dropout_Epoch150

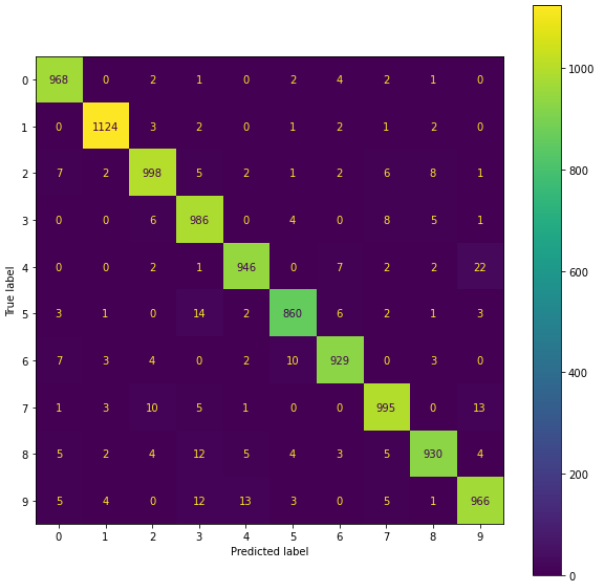


Figure 17: ARN-RAW-ConfMat-tanh-softmax-Neur150_Batch4096_Dropout_Epoch150

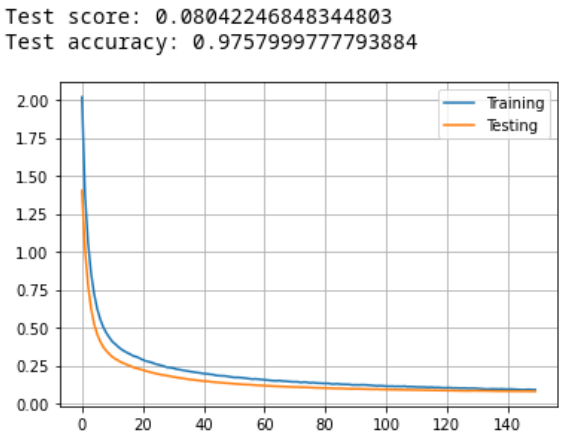


Figure 18: ARN-RAW-Plot-sigmoid-softmax-Neur150_Batch4096_Dropout_Epoch150

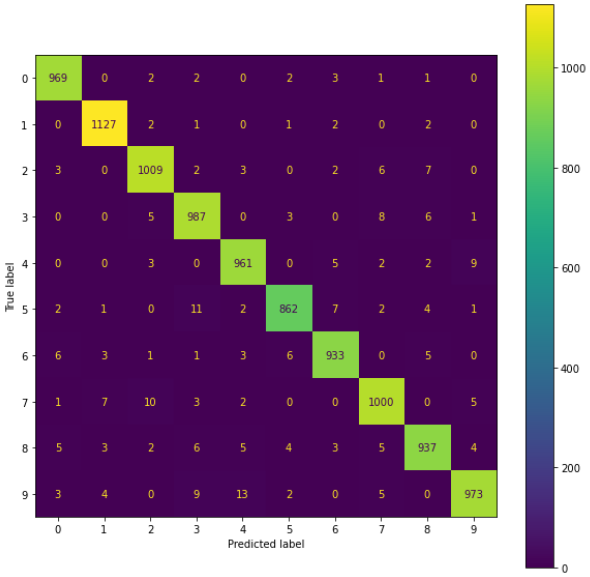


Figure 19: ARN-RAW-ConfMat-sigmoid-softmax-Neur150_Batch4096_Dropout_Epoch150

> MLP_from_HOG.ipynb

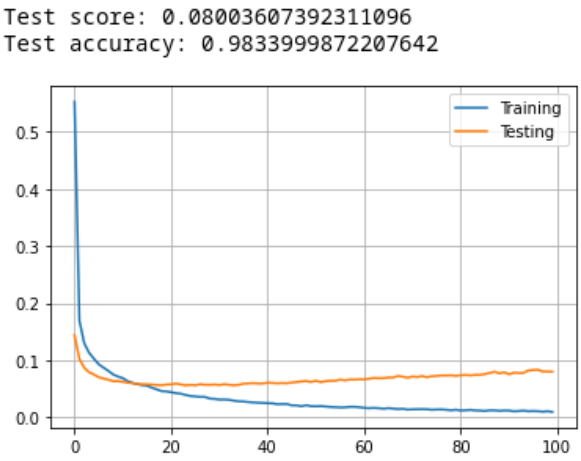


Figure 20: ARN-HOG-Plot-relu-softmax-Neur200_Batch512_Dropout_Epoch100

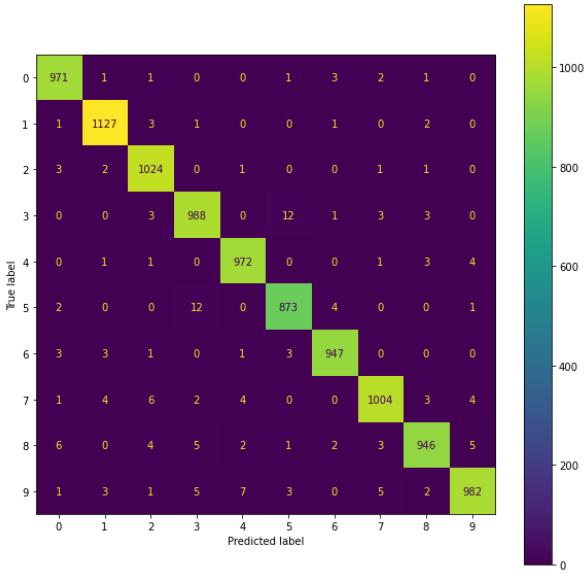


Figure 21: ARN-HOG-ConfMat-relu-softmax-Neur200_Batch512_Dropout_Epoch100

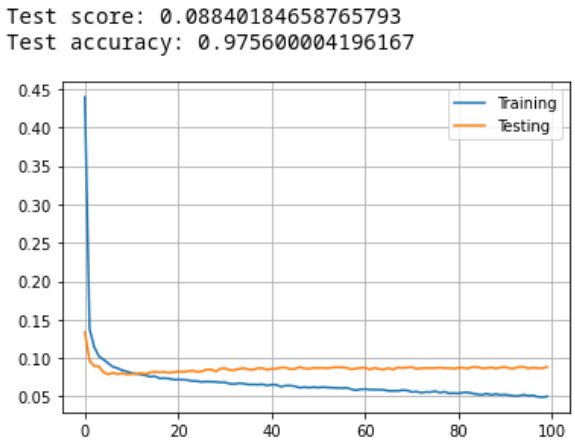


Figure 22: ARN-HOG-Plot-tanh-softmax-Neur200-Batch512_Dropout_Epoch100

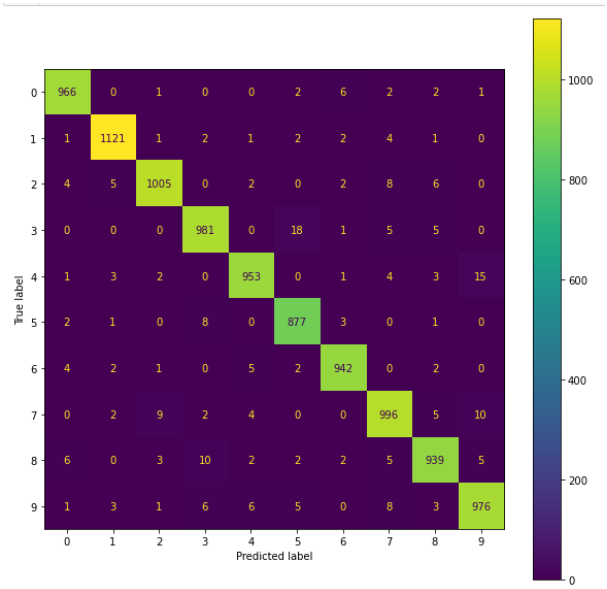


Figure 23: ARN-HOG-ConfMat-tanh-softmax-Neur200_Batch512_Dropout_Epoch100

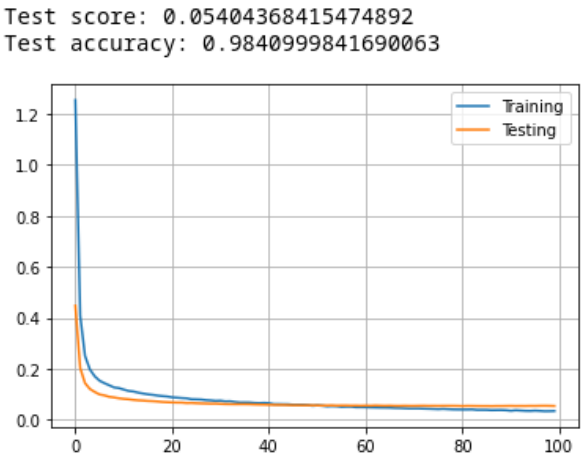


Figure 24: ARN-HOG-Plot-sigmoid-softmax-Neur200_Batch512_Dropout_Epoch100

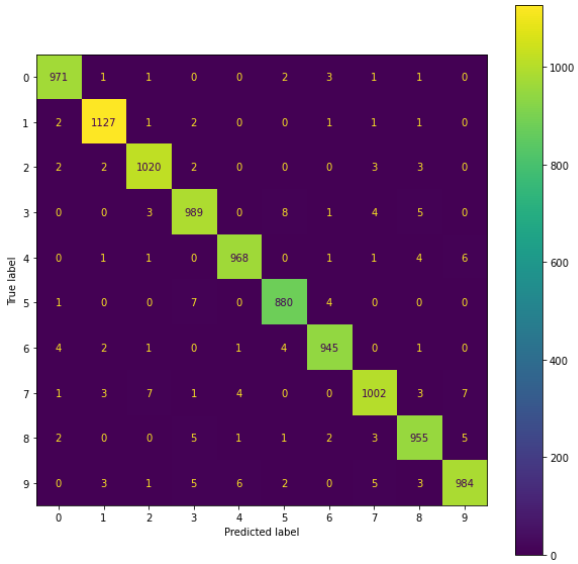


Figure 25: ARN-HOG-ConfMat-sigmoid-softmax-Neur200_Batch512_Dropout_Epoch100

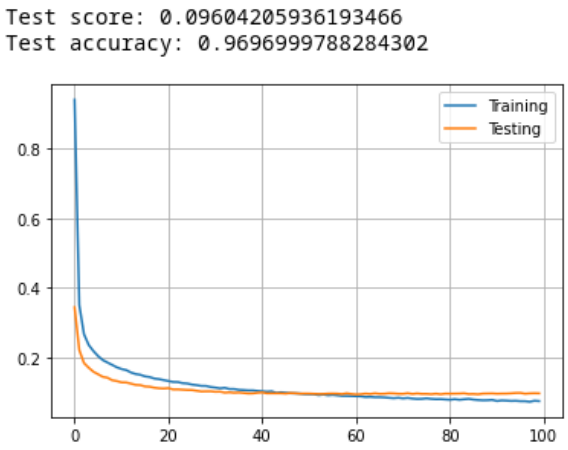


Figure 26: ARN-HOG-Plot-relu-Pixel7_Neur200_Batch512_Dropout_Epoch100

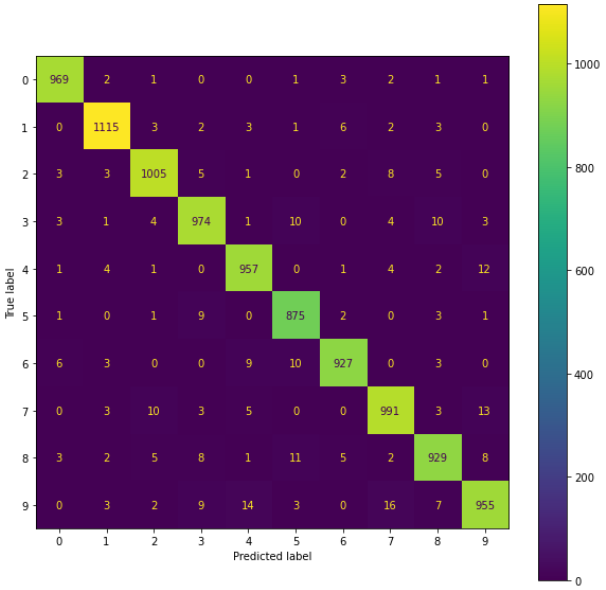


Figure 27: ARN-HOG-ConfMat-relu-Pixel7_Neur200_Batch512_Dropout_Epoch100

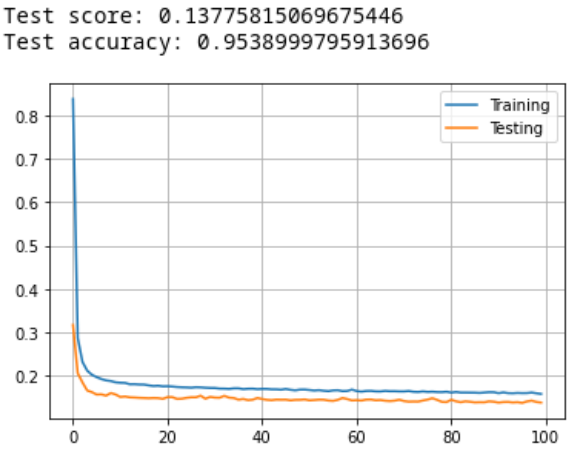


Figure 28: ARN-HOG-Plot-tanh-Pixel7_Neur200_Batch512_Dropout_Epoch100

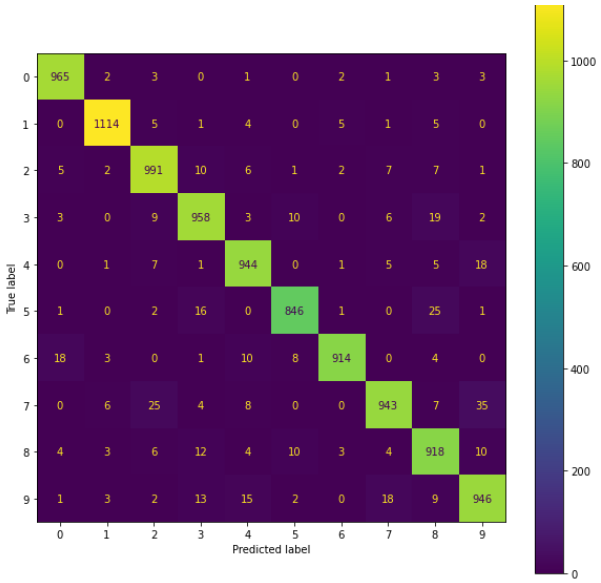


Figure 29: ARN-HOG-ConfMat-tanh-Pixel7_Neur200_Batch512_Dropout_Epoch100

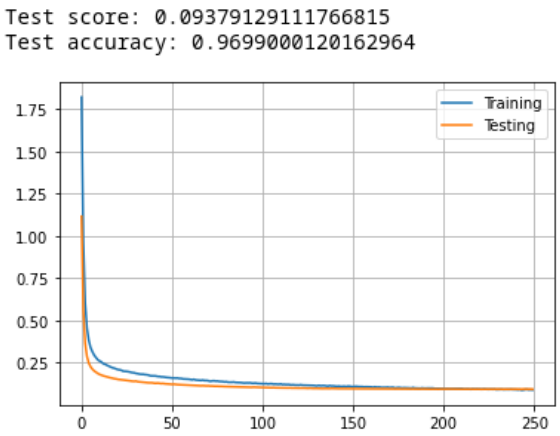


Figure 30: ARN-HOG-Plot-sigmoid-Pixel7_Neur200_Batch512_Dropout_Epoch250

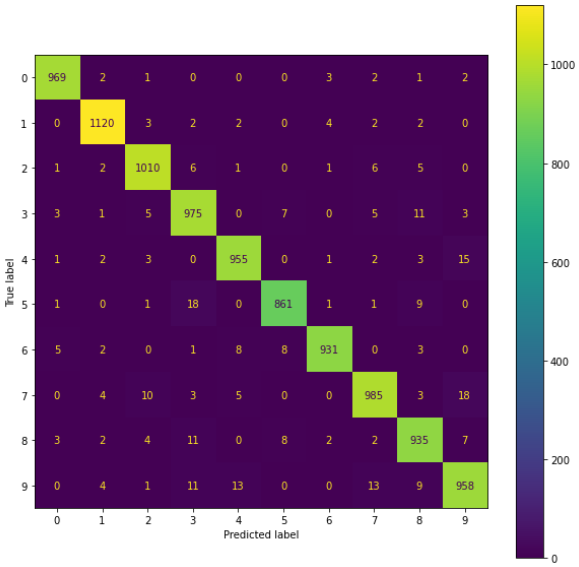


Figure 31: ARN-HOG-ConfMat-sigmoid-Pixel7_Neur200_Batch512_Dropout_Epoch250

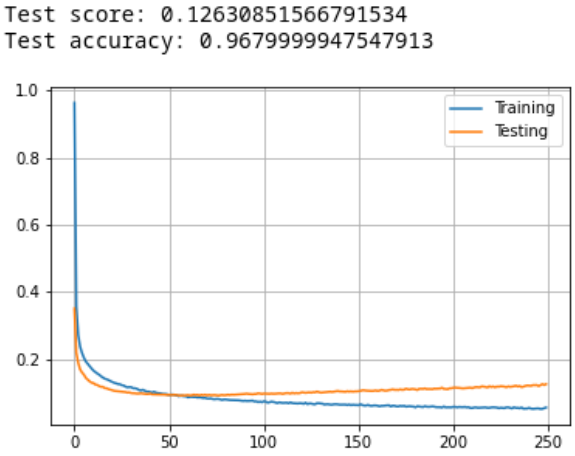


Figure 32: ARN-HOG-Plot-relu-Pixel7_Neur200_Batch512_Dropout_Epoch250

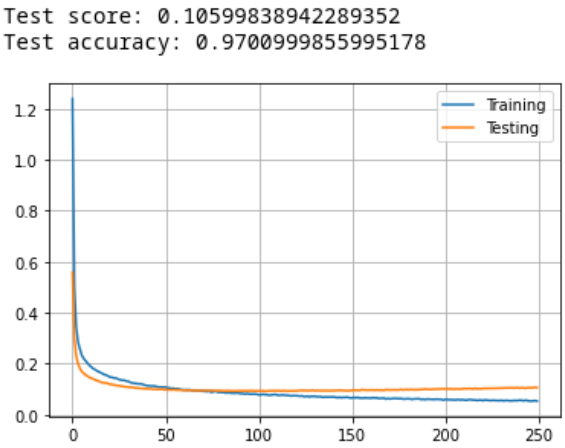


Figure 33: ARN-HOG-Plot-relu-Pixel7_Neur200_Batch1024_Dropout_Epoch250

```
Test score: 0.1260688304901123
Test accuracy: 0.9575999975204468
```

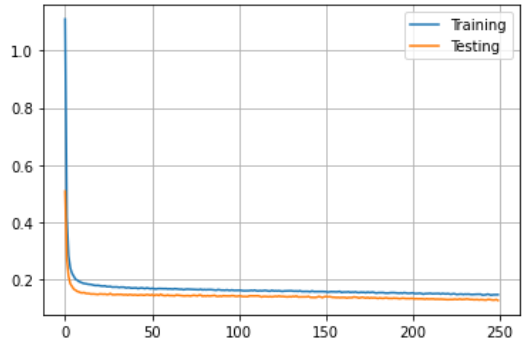


Figure 34: ARN-HOG-Plot-tanh-Pixel7_Neur200_Batch1024_Dropout_Epoch250

```
Test score: 0.0942075327038765
Test accuracy: 0.9684000015258789
```

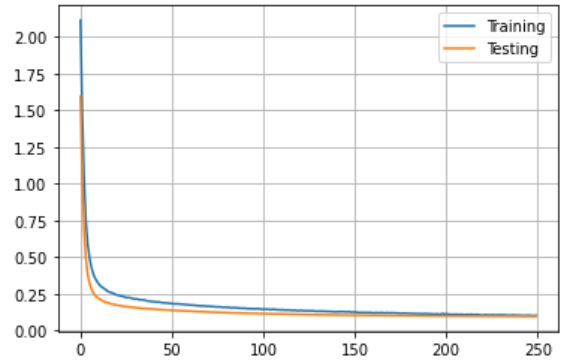


Figure 35: ARN-HOG-Plot-sigmoid-Pixel7_Neur200_Batch1024_Dropout_Epoch250

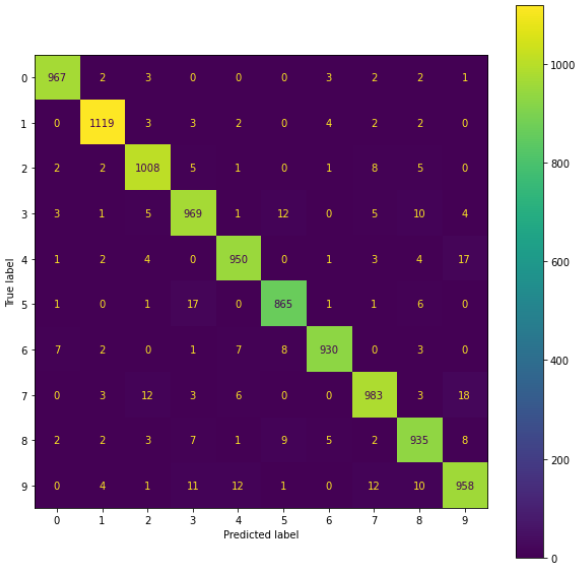


Figure 36: ARN-HOG-ConfMat-sigmoid-Pixel7_Neur200_Batch1024_Dropout_Epoch250

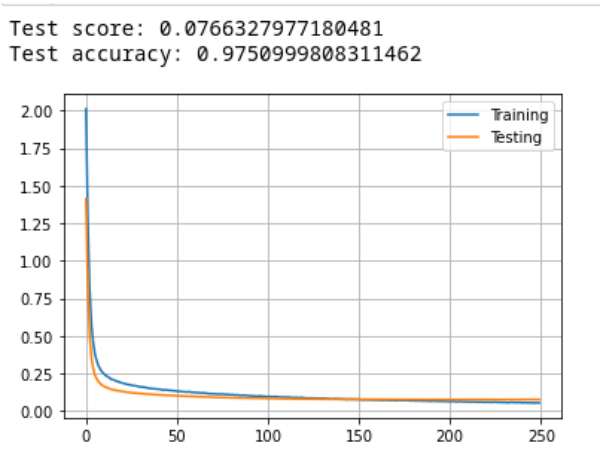


Figure 37: ARN-HOG-Plot-sigmoid-Pixel7_Or16_Neur200_Batch1024_Dropout_Epoch250

```
Test score: 0.11305059492588043
Test accuracy: 0.9733999967575073
```

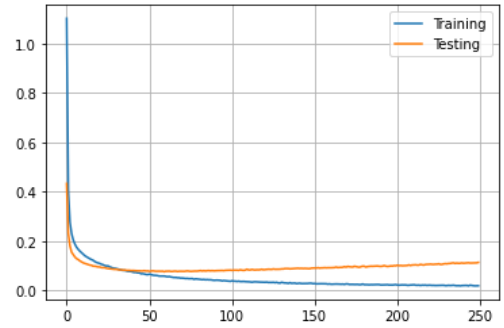


Figure 38: ARN-HOG-Plot-relu-Pixel7_Or16_Neur200_Batch1024_Dropout_Epoch250

```
Test score: 0.10891010612249374
Test accuracy: 0.9648000001907349
```

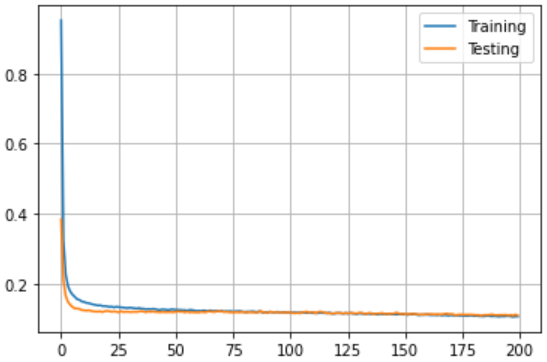
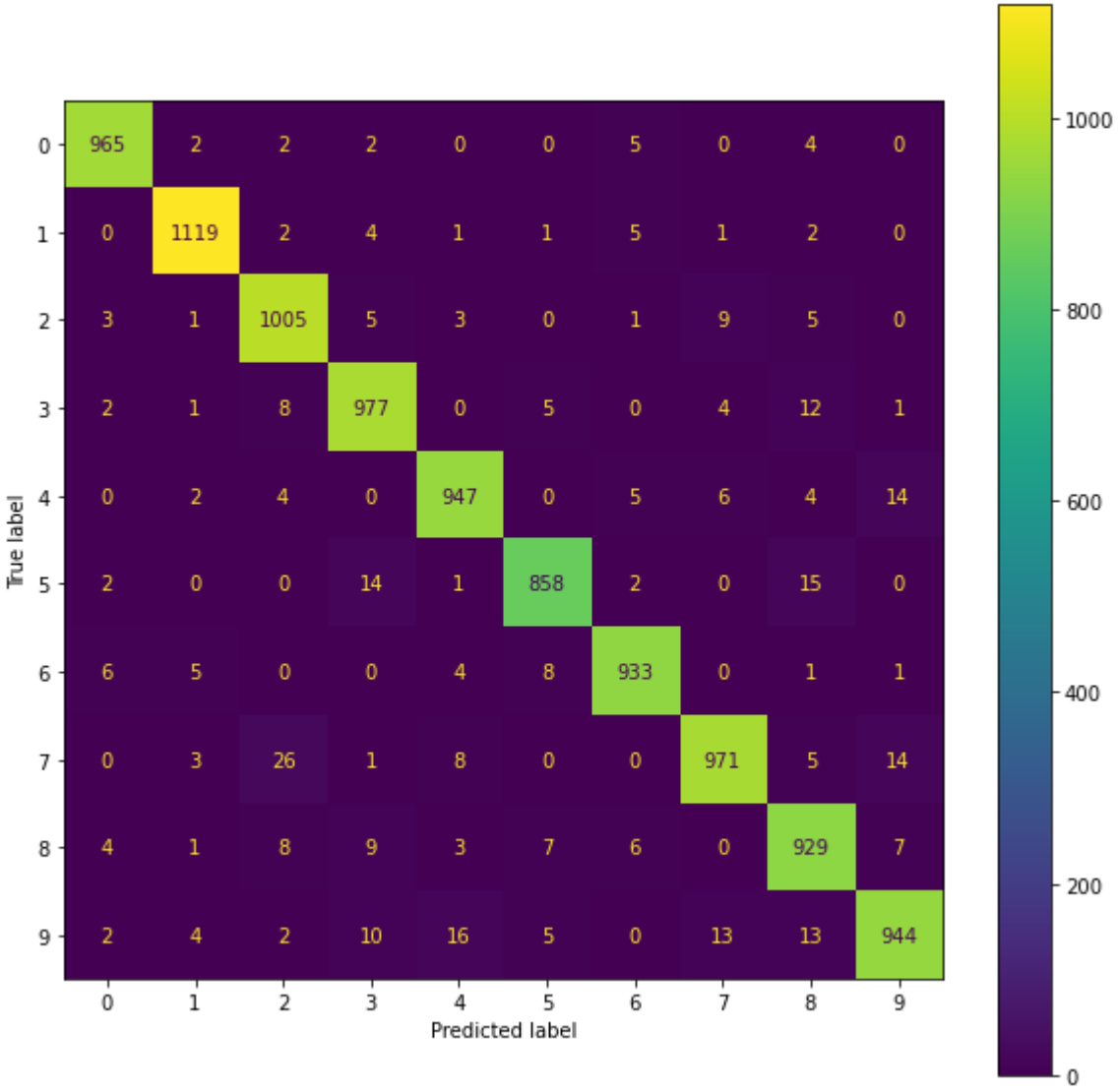


Figure 39: ARN-HOG-Plot-tanh-Pixel7_Or16_Neur200_Batch1024_Dropout_Epoch200


True label
Predicted label
0 965 2 2 2 0 0 5 0 4 0
1 0 1119 2 4 1 1 5 1 2 0
2 3 1 1005 5 3 0 1 9 5 0
3 2 1 8 977 0 5 0 4 12 1
4 0 2 4 0 947 0 5 6 4 14
5 2 0 0 14 1 858 2 0 15 0
6 6 5 0 0 4 8 933 0 1 1
7 0 3 26 1 8 0 0 971 5 14
8 4 1 8 9 3 7 6 0 929 7
9 2 4 2 10 16 5 0 13 13 944
0 1 2 3 4 5 6 7 8 9
1000
800
600
400
200
0

width=50%}

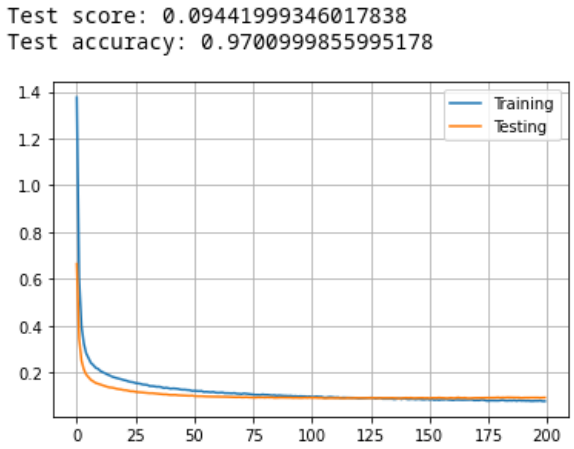


Figure 40: ARN-HOG-Plot-relu-Pixel7_Or8_Neur150_Batch1024_Dropout_Epoch200

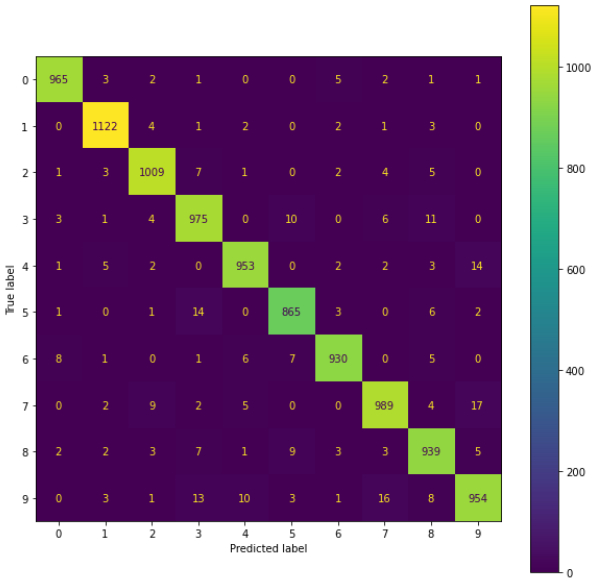


Figure 41: ARN-HOG-ConfMat-relu-Pixel7_Or8_Neur150_Batch1024_Dropout_Epoch200

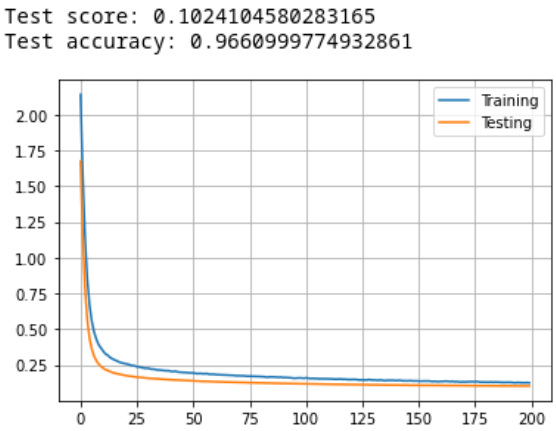


Figure 42: ARN-HOG-Plot-sigmoid-Pixel7_Or8_Neur150_Batch1024_Dropout_Epoch200

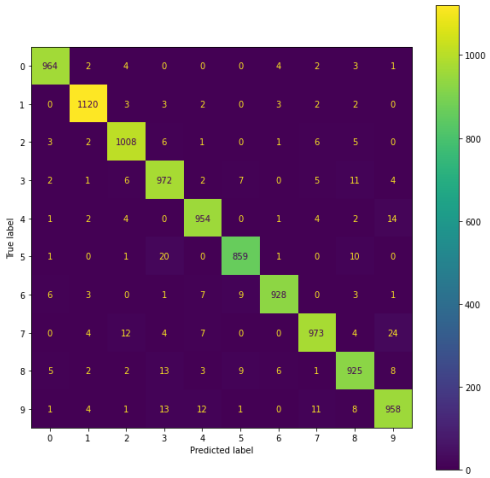


Figure 43: ARN-HOG-ConfMat-sigmoid-Pixel7_Or8_Neur150_Batch1024_Dropout_Epoch200

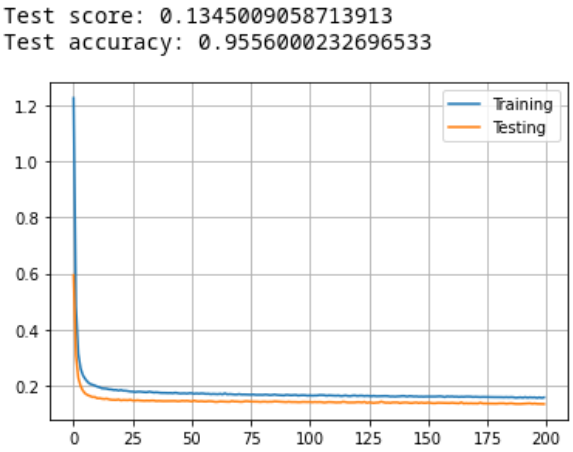


Figure 44: ARN-HOG-Plot-tanh-Pixel7_Or8_Neur150-Batch1024_Dropout_Epoch200

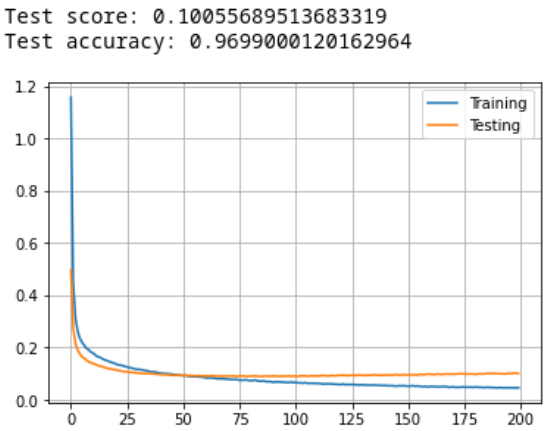


Figure 45: ARN-HOG-Plot-relu-Pixel7_Or8_Neur250_Batch1024_Dropout_Epoch200

```
Test score: 0.09292542189359665
Test accuracy: 0.9690999984741211
```

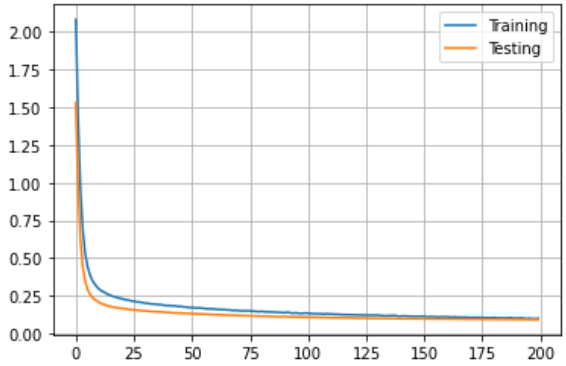


Figure 46: ARN-HOG-Plot-sigmoid-Pixel7_Or8_Neur250_Batch1024_Dropout_Epoch200

```
Test score: 0.13129283487796783
Test accuracy: 0.9555000066757202
```

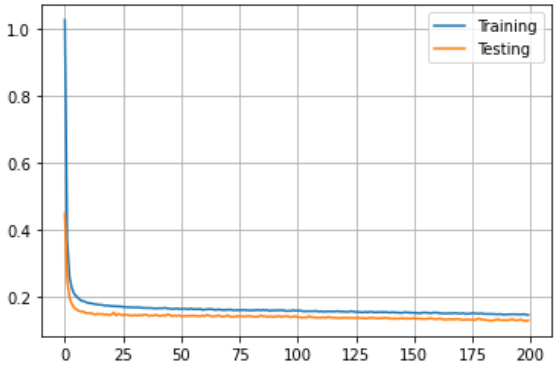


Figure 47: ARN-HOG-Plot-tanh-Pixel7_Or8_Neur250_Batch1024_Dropout_Epoch200

> CNN.ipynb