ARN - Report - Labo04

Anthony Coke, Guilain Mbayo, Mehdi Salhi

May 11, 2022

Contents

Learning algorithm **3**

Model Complexity **3**

Deep Neural Networks **4**

Tests **4**

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

> MLP_from_HOG.ipynb

> CNN.ipynb

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

> 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

> MLP_from_raw_data.ipynb

> MLP_from_HOG.ipynb

> CNN.ipynb

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

```
Test score: 0.11751019209623337
Test accuracy: 0.9818999767303467
```

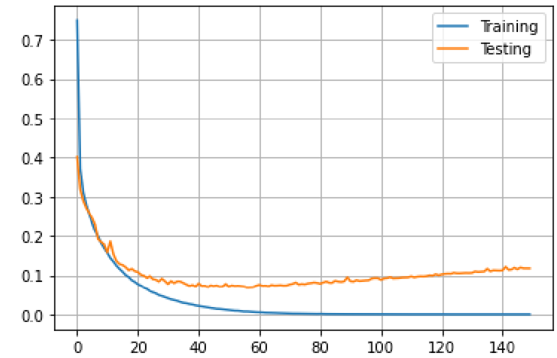

Figure 2: ARN-RAW-Plot-tanh-softmax_Batch2048_NoDropout_Epoch150

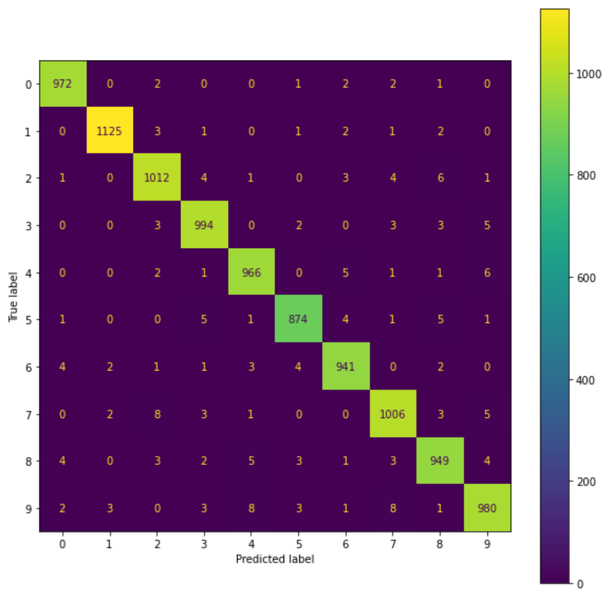

Figure 3: ARN-RAW-ConfMat-tanh-softmax_Batch2048_NoDropout_Epoch150

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

```
Test score: 0.0734260305762291
Test accuracy: 0.9796000123023987
```

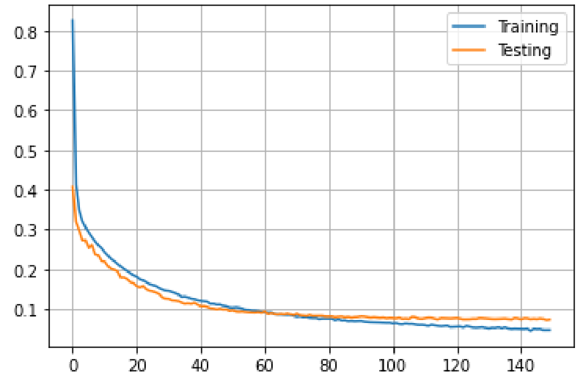

Figure 4: ARN-RAW-Plot-tanh-softmax_Batch2048_Dropout_Epoch150

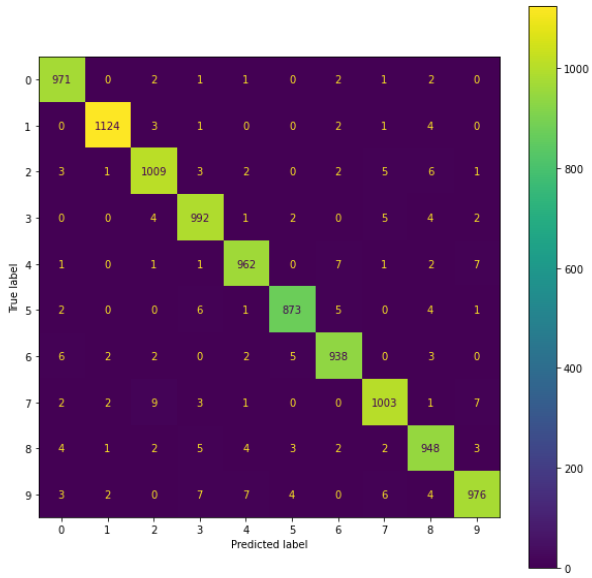

Figure 5: ARN-RAW-ConfMat-tanh-softmax_Batch2048_Dropout_Epoch150

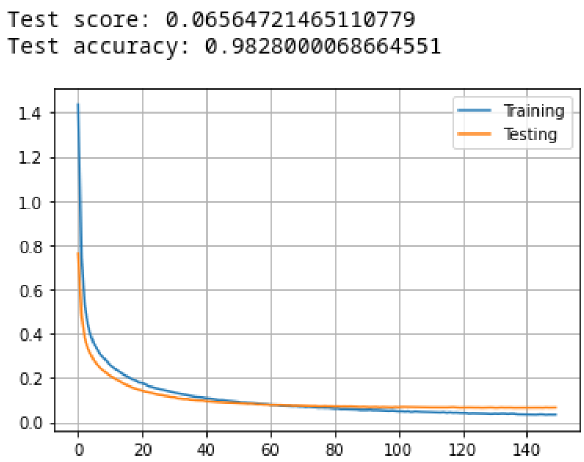

Figure 6: ARN-RAW-Plot-sigmoid-softmax_Batch2048_Dropout_Epoch150

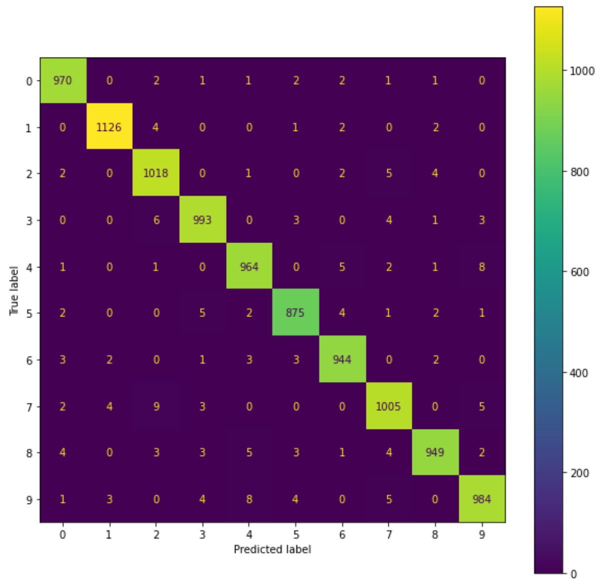

Figure 7: ARN-RAW-ConfMat-sigmoid-softmax_Batch2048_Droptout_Epoch150

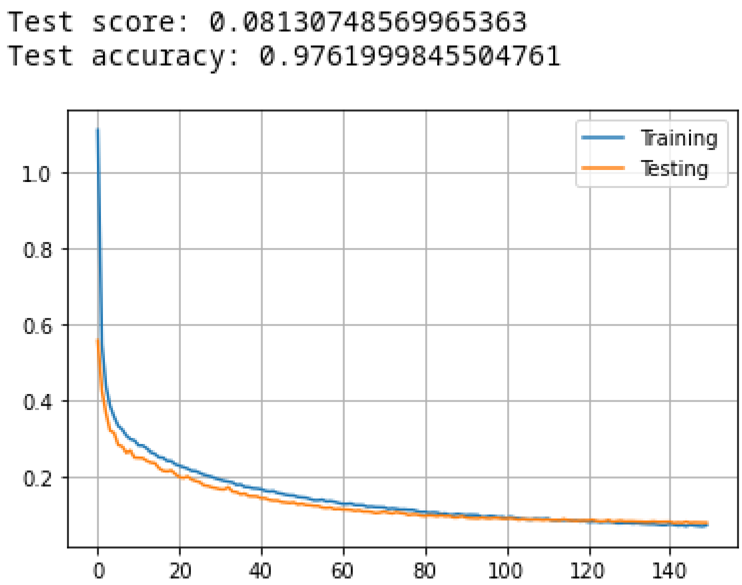

Figure 8: ARN-RAW-Plot-tanh-softmax-Neur250_Batch4096_Dropout_Epoch150

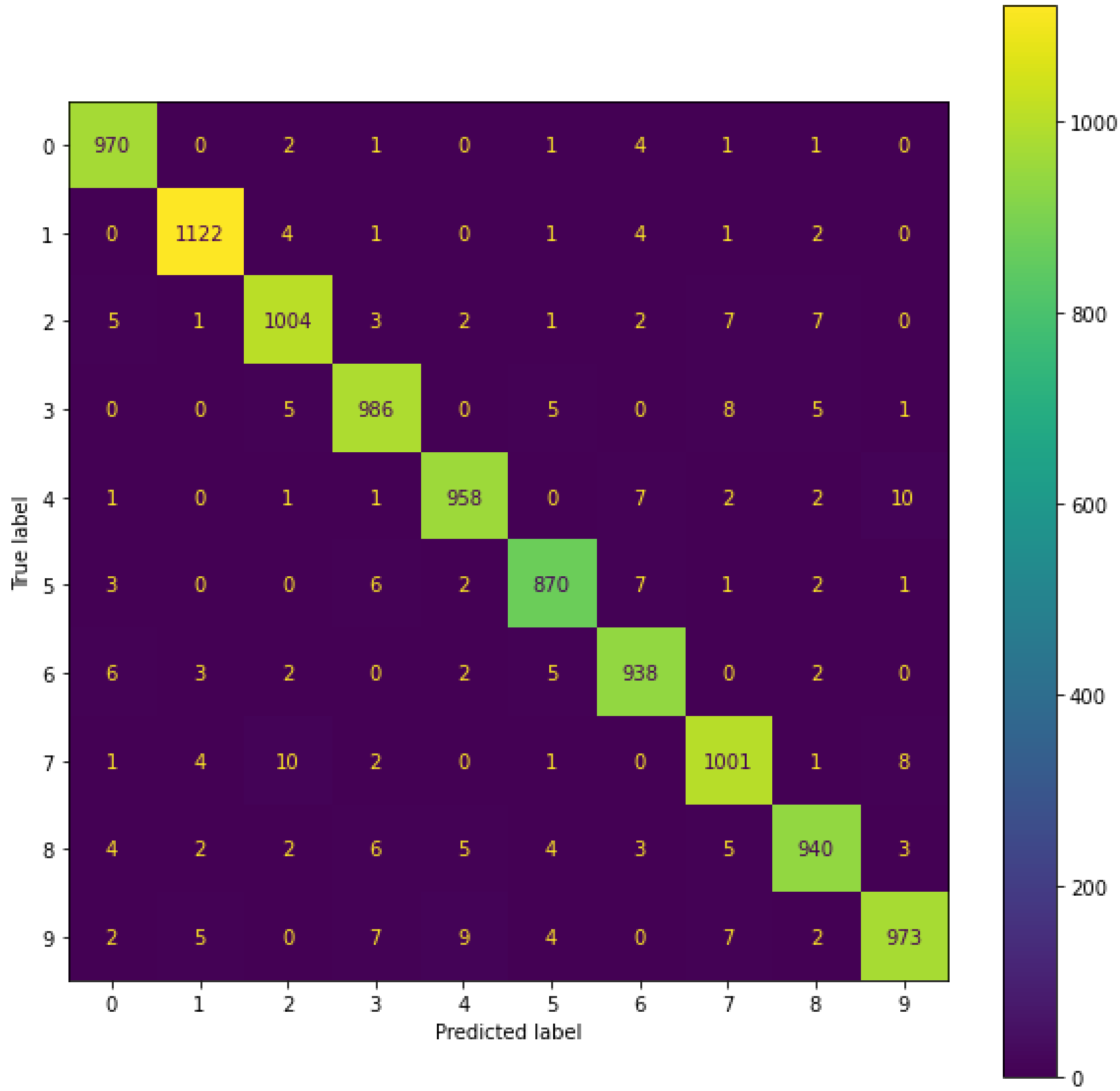

Figure 9: ARN-RAW-ConfMat-tanh-softmax-Neur250_Batch4096_Dropout_Epoch150

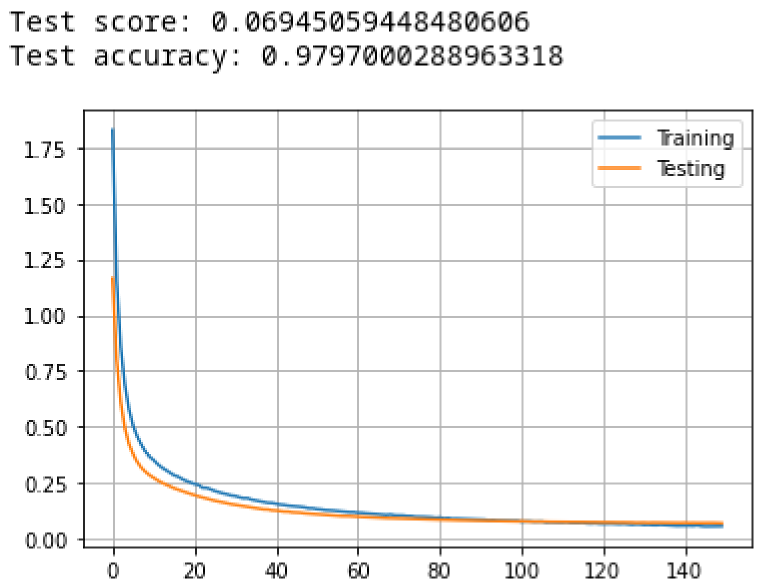

Figure 10: ARN-RAW-Plot-sigmoid-softmax-Neur250_Batch4096_Dropout_Epoch150

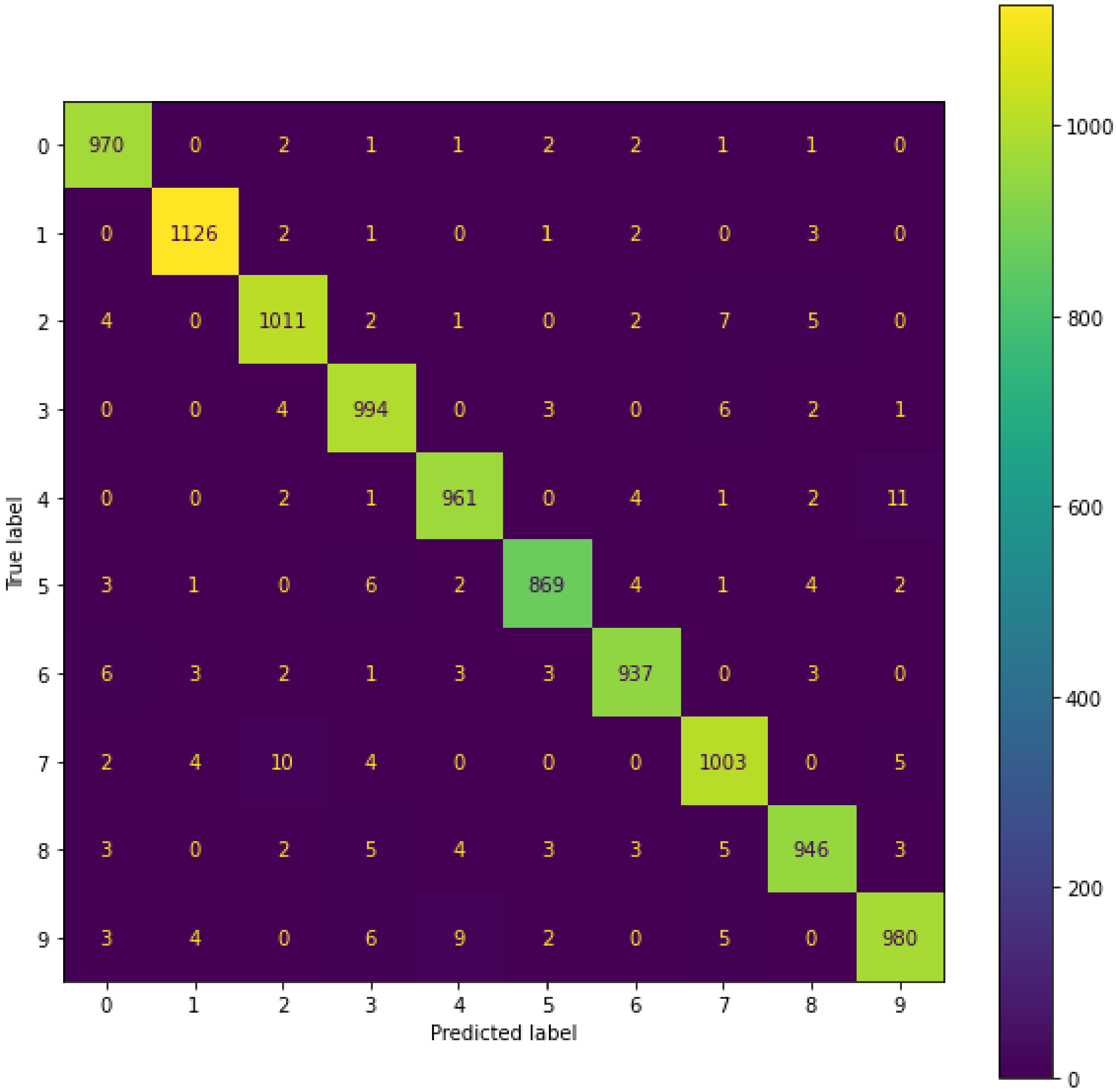

Figure 11: ARN-RAW-ConfMax-sigmoid-softmax-Neur250_Batch4096_Dropout_Epoch150

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

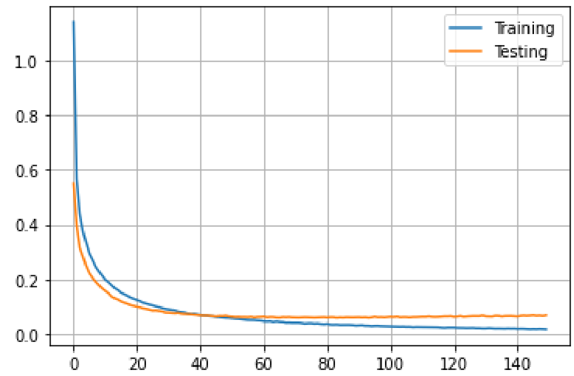

Figure 12: ARN-RAW-Plot-relu_softmax-Neur250_Batch4096_Dropout_Epoch150

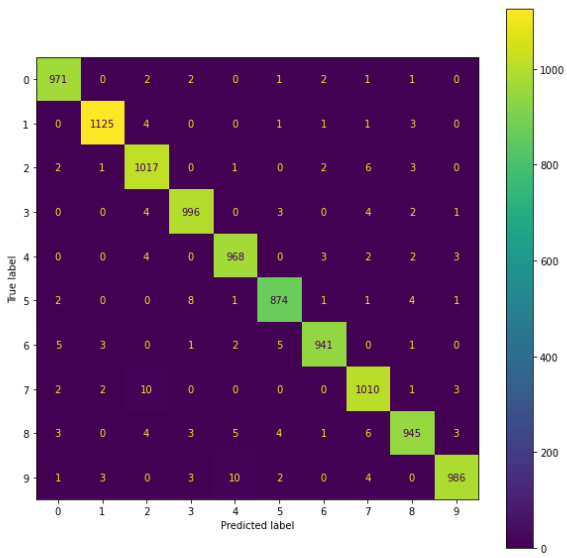

Figure 13: ARN-RAW-ConfMat-relu-softmax-Neur250_Batch4096_Dropout_Epoch150

```
Test score: 0.07871479541063309
Test accuracy: 0.9796000123023987
```

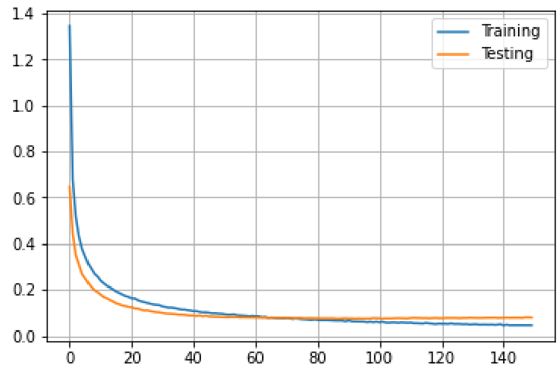

Figure 14: ARN-RAW-Plot-relu-softmax-Neur150_Batch4096_Dropout_Epoch150

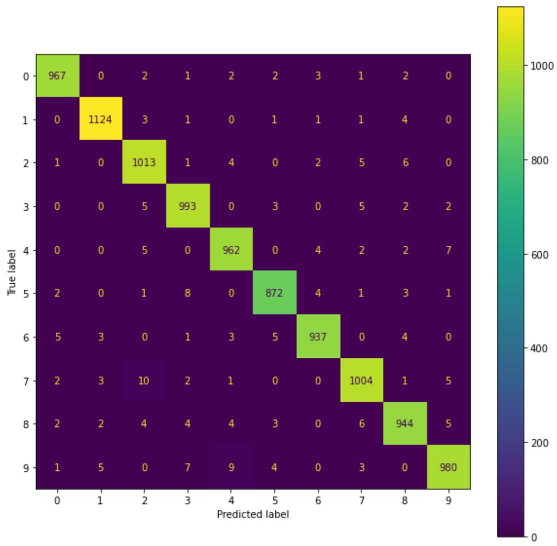

Figure 15: ARN-RAW-ConfMat-relu-softmax-Neur150_Batch4096_Dropout_Epoch150

```
Test score: 0.09491664171218872
Test accuracy: 0.9702000021934509
```

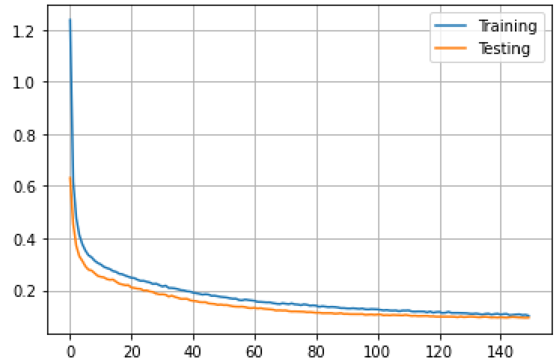

Figure 16: ARN-RAW-Plot-tanh-softmax-Neur150_Batch4096_Dropout_Epoch150

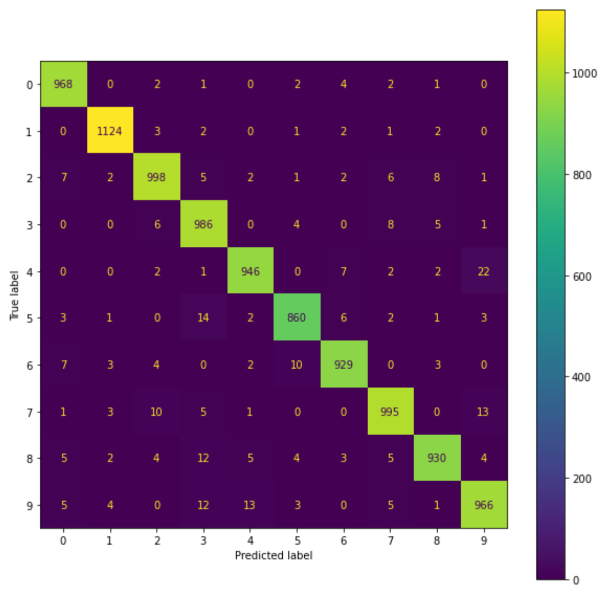

Figure 17: ARN-RAW-ConfMat-tanh-softmax-Neur150_Batch4096_Dropout_Epoch150

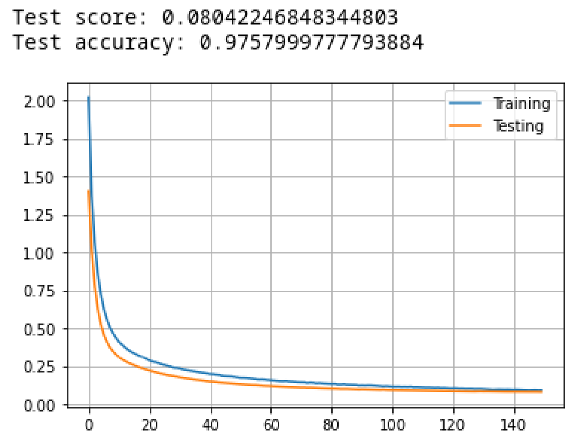

Figure 18: ARN-RAW-Plot-sigmoid-softmax-Neur150_Batch4096_Dropout_Epoch150

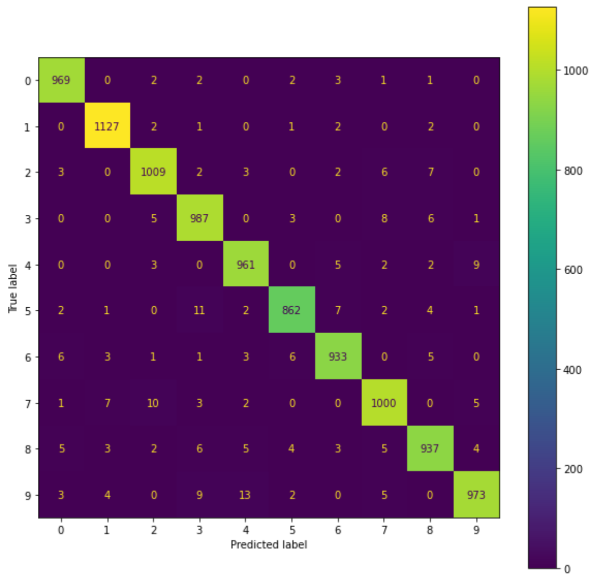

Figure 19: ARN-RAW-ConfMat-sigmoid-softmax-Neur150_Batch4096_Dropout_Epoch150

> MLP_from_HOG.ipynb

> CNN.ipynb