Master's in Computer Engineering

Machine Learning

Assignment 1 Report

OCR

Optical Character Recognition

Miguel António Gabriel de Almeida Faria | 2019216809

Sancho Amaral Simões | 2019217590

• U C •

Index

1. Objective

The purpose of this work is to develop neural networks that allow the recognition of handwritten numeric characters, specifically the ten Arabic numerals: 1, 2, 3, 4, 5, 6, 7, 8, 9, 0. Two distinct neural network architectures were used, with and without filter. Other parameters were varied as the neural networks activation functions, training functions, learning rate, among others, in order to analyse the most promising configurations.

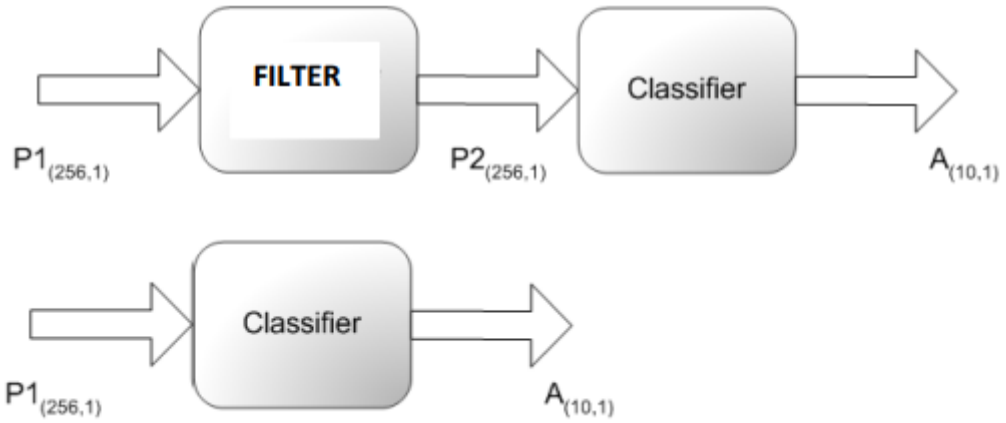

Figure 1 – Different architectures implemented

If a filter is used, it can be of two types: an associative memory or a binary perceptron. When only the classifier is used, its number of layers may be one or two.

2. Dataset

The choice of dataset is very important in order to obtain good results, since it defines the future capacity of generalization of the system as its robustness. As the writing of numbers is done by hand, it is necessary to keep in mind that there are multiple possible variations in the design of the figures, being dependent of the person or even the system that writes it (such as a computer or a seven-segment display). Therefore, the group agreed that the dataset should be made up of digits written by the two elements of the group, in order to increase its heterogeneity and include a greater number of possibilities, allowing the network to have better accuracy in the results, even in cases with writing imperfections.

Also, in order to have a notion of how should the dataset be formalised and built, the group considered three main characteristics:

- Form: handwritten or "machine" (7-segment displays)
- Size: small, medium or large
- Perfection: low, medium or high

$$|\tau| = |E| * |D| * |F| * |T| * |P|$$

$\|\tau\|$	$\|E\|$	$\|D\|$	$\|F\|$	$\|T\|$	$\|P\|$
Nº train elements (900)	Nº of trials (5)	Nº of digits (10)	Nº of forms (2)	Nº of sizes (3)	Nº of perfections (3)

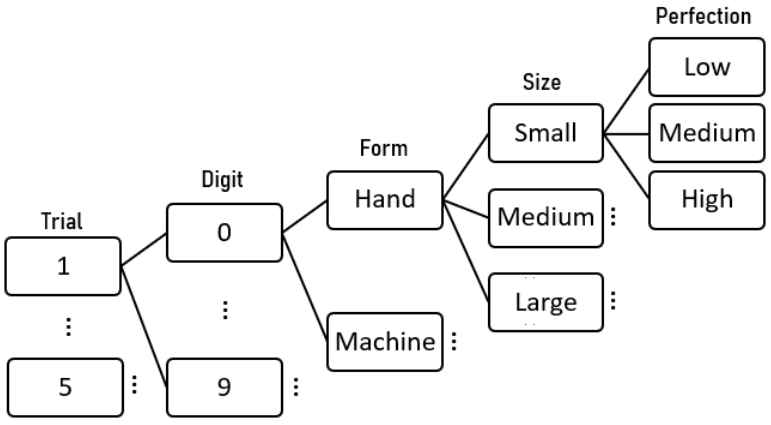

Figure 2 – Expression and visualization of the dataset generation plan

Despite the work in question being sung to the two elements of the group, it is important to say that some knowledge was acquired with lay people consulting the subject. This knowledge consists of different handwriting and different ways of drawing the digits and has been integrated into the part of the training input called *corner cases*. This was another important step towards developing more accurate and noise-robust neural networks.

3. Procedure

The generation of the neural networks was made considering several parameters:

- Activation function – *hardlim*, *purelin*, *logsig*, *elliotsig* and *softmax* .
- Train function – *learn(p, pn, gd, gdm, h, hd, wh)* and *train(gd, gda, gdm, scg*)*.
- Train type – incremental and batch.
- Learn rate – range of the possible values (i.e. (0:0.1:1)).
- Seed - range of the possible values (i.e. (0:1:3)) – used to randomly initialize the neurons' weights and biases. This allowed to correctly compare the performance between two neural networks with, for example, different activation functions, but with the same seed.
- Number of layers (classifier) – one and two.
- Number of hidden neurons (classifier) - range of the possible values (i.e. (10:10:50)).

The aforementioned parameters ranges were specified in a class with constant properties so that many combinations of parameters could be tested iteratively. There are much more efficient approaches that explore such a great search space of hyperparameters than simple iteration. One of these approaches is the *genetic algorithm*, but, due to the time and resources (processing power) shortage, the group could not implement it.

This way, after training the nets, the group could analyse the different *train performance* and *validation performance**[2], in order to select the most promising ones. The neural networks were all dumped into *.mat* files with names that succinctly indicate their configuration.

It is important to emphasize the training structure of the *OCR* system as a whole: it was necessary to train the filter for when used in series with the classifier; the classifier had to be trained in two different ways, one with direct input (designed by the user) and another with the filter output, for when used in series with it.

* Note: The *trainlm* train function (*Levenberg.Marquardt*) was put aside since it requires too much memory to perform its calculations.

*[2] Note: There is no *testing performance* since the test phase would be *done a posteriori* by the group and not by the *Deep Learning Toolbox* itself

4. Results

After training hundreds of differently configurated neural networks, large amounts of data were outputted into *.csv* files. Down below, there are some considerations and graphical data regarding the training results.

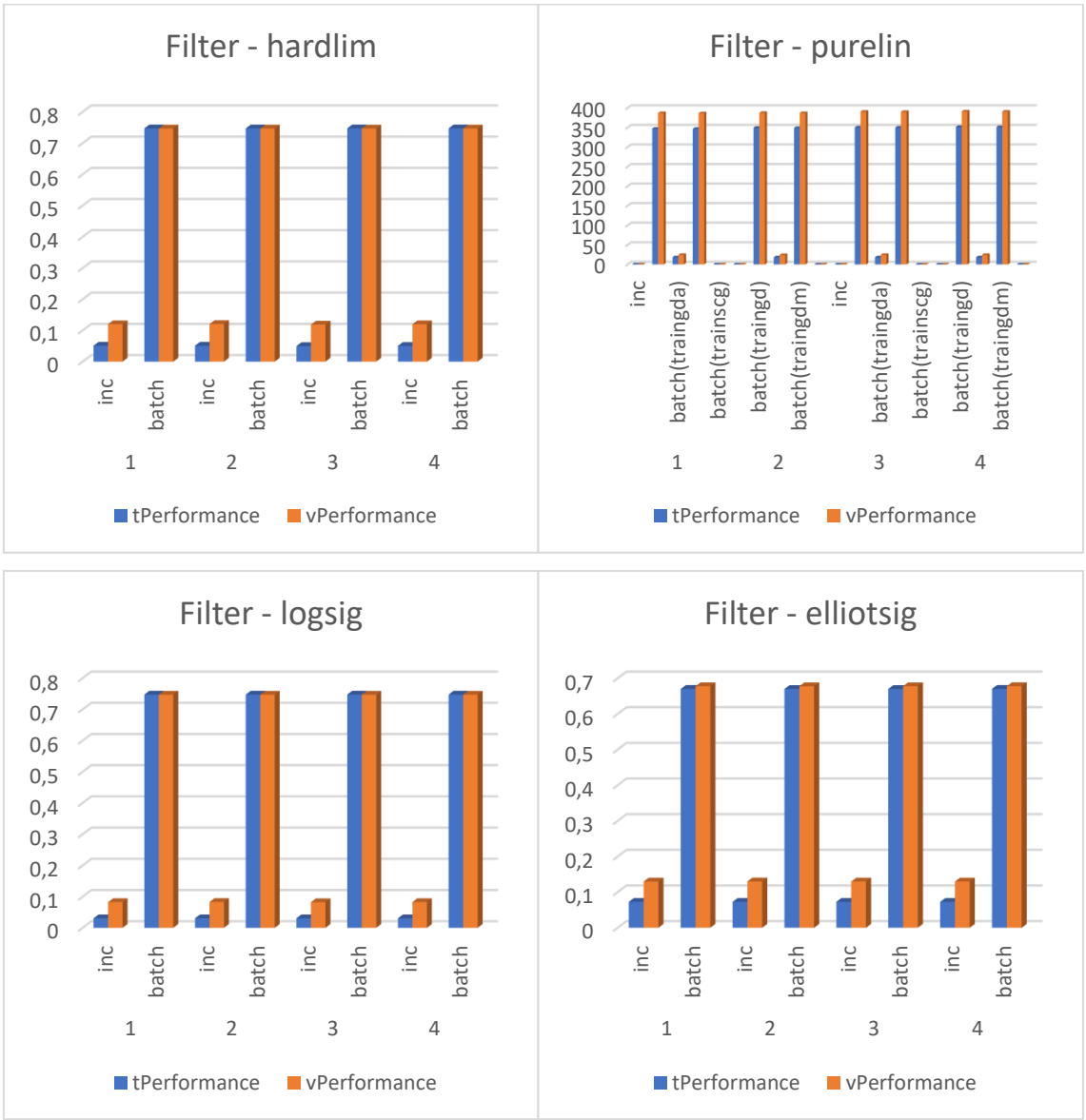

Figure 3 – Comparative graphs between the two types of training for the different activation functions

The numbers above the chart legends represent the seed used to initialize the *RNG* and the neural networks' subsequent weights and biases.

Although the incremental training type is very time-consuming when compared to the batch type, it generally presented better results in terms of minimizing the error of the trained neural networks.

As mentioned before, the neural networks training phase generated large amounts of input and hyperparameter combinations. For this reason, the group had some difficulty in proceeding with the graphical representation of the data. However, the analysis of the tables below (partial truncation of the originals) allows us to draw some conclusions in the context of the performance of the various neural networks.

#	seed	activationFunction	trainFunction	trainType	learnRate	tPerformance	vPerformance
1	2	logsig	learnp	inc	0.100	0.03080	0.08231
2	2	logsig	learngd	inc	0.100	0.03080	0.08231
3	2	logsig	learngd	inc	0.300	0.03080	0.08231
4	2	logsig	learngd	inc	0.500	0.03080	0.08231
5	2	logsig	learngd	inc	0.700	0.03080	0.08231
6	2	logsig	learngdm	inc	0.100	0.03080	0.08231
7	2	logsig	learngdm	inc	0.300	0.03080	0.08231
8	2	logsig	learngdm	inc	0.500	0.03080	0.08231
9	2	logsig	learngdm	inc	0.700	0.03080	0.08231
...							
85	2	purelin	trainscg	batch	0.100	0.04296	0.10324
86	2	purelin	trainscg	batch	0.300	0.04296	0.10324
87	2	purelin	trainscg	batch	0.500	0.04296	0.10324
88	2	purelin	trainscg	batch	0.700	0.04296	0.10324
89	0	purelin	trainscg	batch	0.100	0.04313	0.10325
...							
101	2	hardlim	learnp	inc	0.100	0.04996	0.11866

Figure 4 – Table of the results related with the filter train

The observation of the table above, regarding the training of the filter, allows us to draw the following conclusions:

- The *logsig* activation function is the one that best fits the filter and behaves optimally when used in incremental training.

- The *purelin* activation function presents a good performance at the training level, however the validation function is worse, which may indicate tendencies of the neural network to suffer from *overfitting*.

- The perceptron, when used as a filter, presents similar results to an *ADALINE* (good). This could happen, perhaps, because the function of the mentioned filter is not to separate non-linearly separable classes, but to act as an associative memory.

#	seed	layers	nHiddenNeurons	activationFunction	trainFunction	trainType	learnRate	tPerformance	vPerformance
1	0	1	0	softmax	trainscg	batch	0.500	0.00556	0.05468
2	1	1	0	softmax	trainscg	batch	0.500	0.00556	0.05468
3	0	1	0	softmax	trainscg	batch	0.300	0.00556	0.05468
4	1	1	0	softmax	trainscg	batch	0.300	0.00556	0.05468
5	0	1	0	softmax	trainscg	batch	0.100	0.00556	0.05468
6	1	1	0	softmax	trainscg	batch	0.100	0.00556	0.05468
7	0	1	0	softmax	learngd	inc	0.500	0.00623	0.05455
8	0	1	0	softmax	learngdm	inc	0.500	0.00623	0.05455
9	0	1	0	softmax	learnh	inc	0.500	0.00623	0.05455

...

39	1	2	70	elliotsig	trainscg	batch	0.500	0.00862	0.05793
40	1	2	70	elliotsig	trainscg	batch	0.300	0.00862	0.05793
41	1	2	70	elliotsig	trainscg	batch	0.100	0.00862	0.05793

...

74	0	1	0	logsig	learngd	inc	0.500	0.01390	0.06014
75	0	1	0	logsig	learngdm	inc	0.500	0.01390	0.06014
76	0	1	0	logsig	learnh	inc	0.500	0.01390	0.06014
77	0	1	0	logsig	learnhd	inc	0.500	0.01390	0.06014

...

135	0	1	0	purelin	trainscg	batch	0.500	0.04075	0.07809
136	1	1	0	purelin	trainscg	batch	0.500	0.04075	0.07809
137	0	1	0	purelin	trainscg	batch	0.300	0.04075	0.07809
138	1	1	0	purelin	trainscg	batch	0.300	0.04075	0.07809
...									
1424	1	2	10	hardlim	learnp	inc	0.100	0.53966	0.54302

Figure 5 – Table of the results related with the isolated classifier train

After analysing the table above, regarding the training of the isolated classifier, it is possible to make the following statements:

- The *softmax* activation function dominates when it comes to minimizing the MSE of neural networks (error), in the form of low *tPerf* and *vPerf*. This is due to the fact that the *softmax* activation function presents a very interesting behaviour: it causes the outputs of the neural network to add up to one, which can be interpreted as probabilities or degrees of certainty of the classification.

- The *softmax* activation function provides great results whether it is used in an *incremental* or *batch* train.

- The *elliotsig* activation function (very similar to *tansig*, but much more time efficient) gives good results when used in both layers of the isolated classifier.

- The *logsig* activation function presents a reasonable performance when used in both layers of the isolated classifier.

- The *purelin* activation function (linear) presents reasonable results when used in both layers of the isolated classifier.

- The hardlim activation function, used with the perceptron rule (which makes the classifier a perceptron) proves not to be feasible due to poor performances. This may be due to the fact that Perceptron is unable to effectively classify non-linearly separable classes.

- In this case, the variation of the learning coefficient had no great influence.

For reasons of summary, the table relating to the classifier, when used in series with the filter, is not shown. Its behaviour with regard to activation functions is similar when used is similar, but presents smaller error values. This is due to the fact that the filter considerably reduces the noise and, therefore, its output, which corresponds to the input of the classifier, allows a more efficient training of the neural network corresponding to it.

To obtain the test input that allows us to calculate the precision of the neural networks already trained, a new dataset was generated following the principles mentioned in 2. Thus, choosing some neural networks configuration which presented a good performance in the training phase to test (four, to be exactly), it was possible to extract the correspondent confusion matrices, after the test phase.

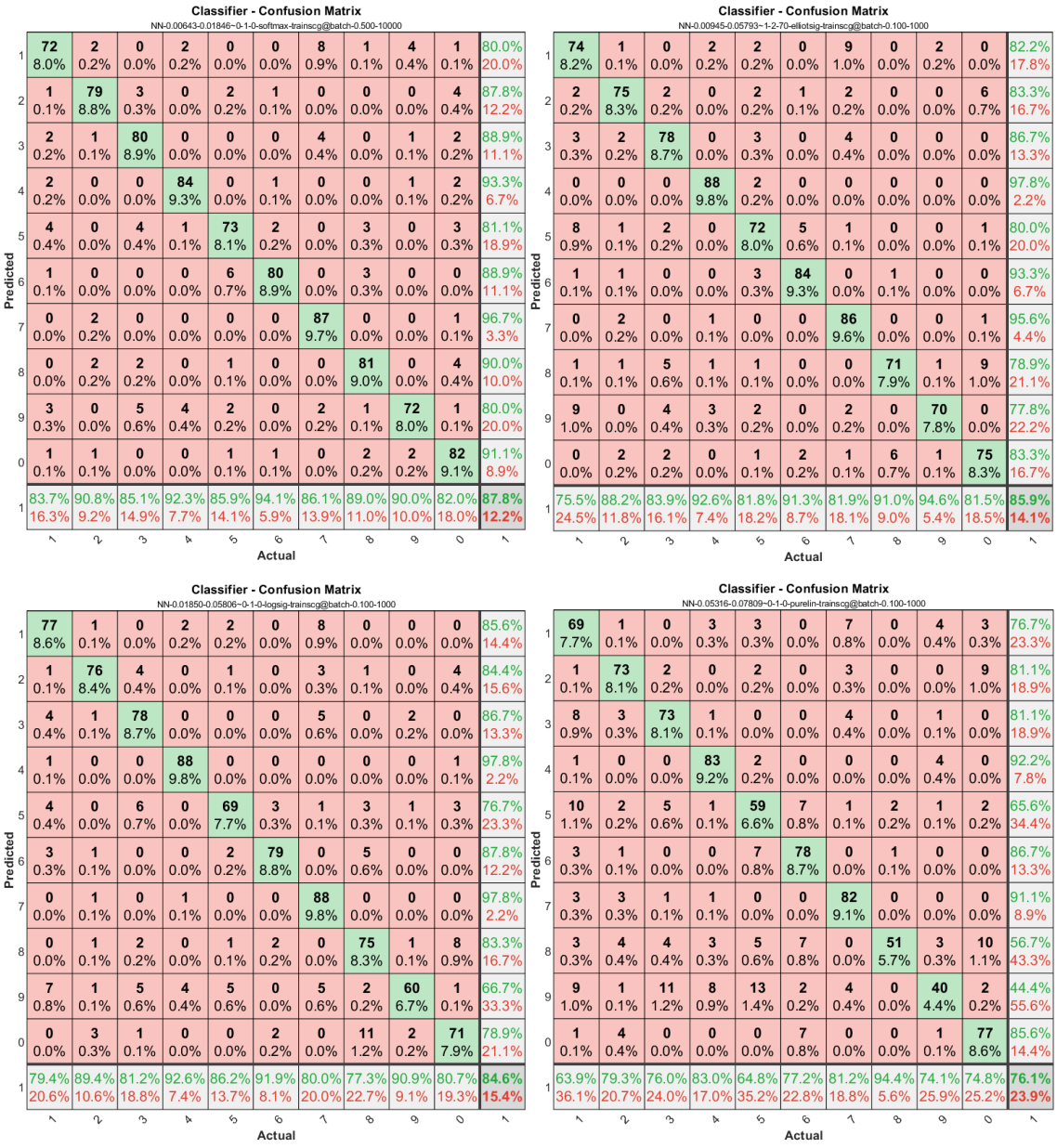

Figure 6 – Confusion matrices for the isolated classifier architecture

From left to right, top to bottom:

- *Softmax* activation function, *batch* training, l. coefficient of 0.5, 10000 epochs
- *Elliotsig* activation function, *batch* training, l. coefficient of 0.1, 1000 epochs
- *Logsig* activation function, *batch* training, l. coefficient of 0.1, 1000 epochs
- *Purelin* activation function, *batch* training, l. coefficient of 0.1, 1000 epochs

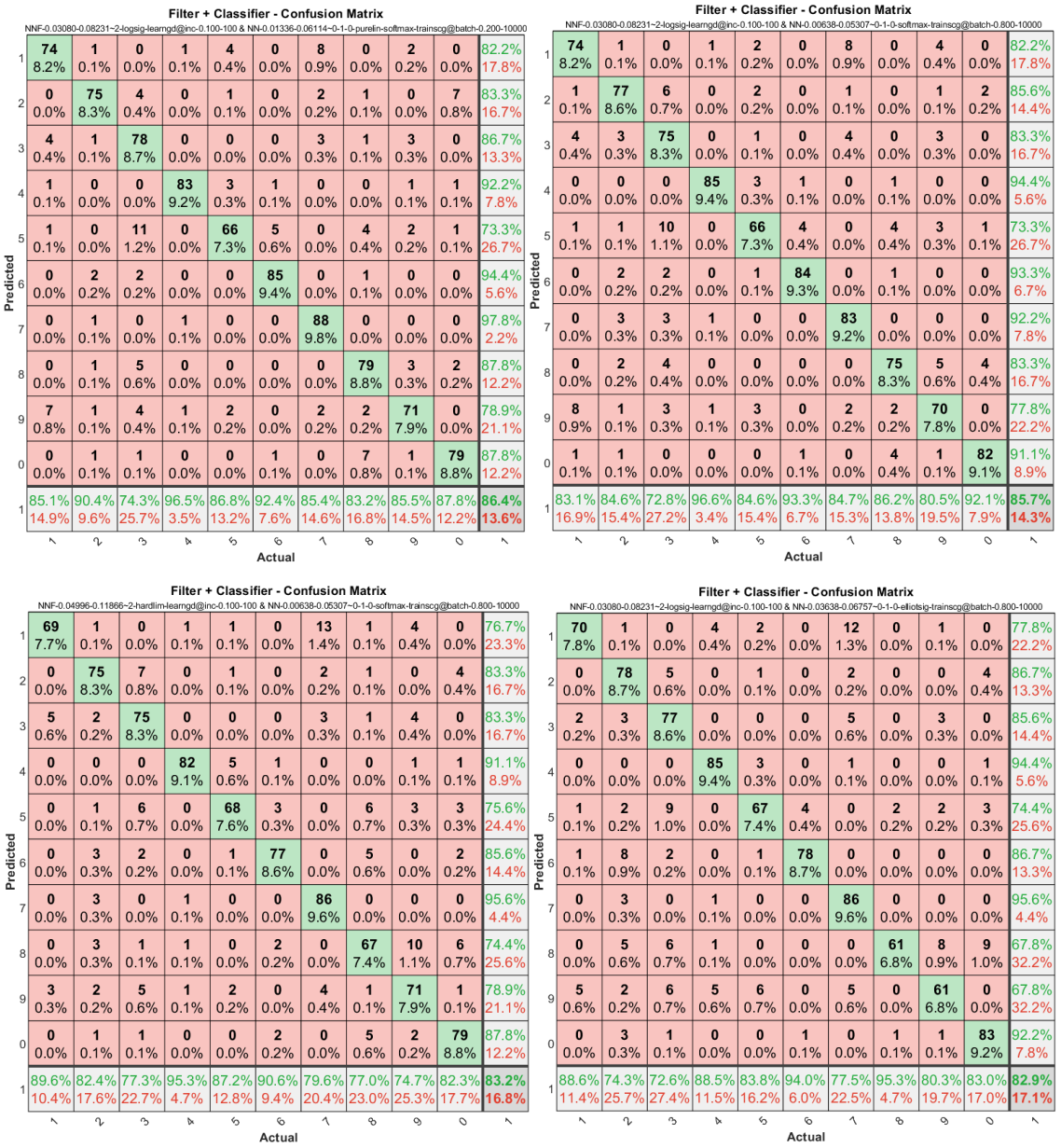

Figure 7 – Confusion matrices for the filter + classifier architecture

From left to right, top to bottom:

- *Logsig* + *purelin-softmax* activation functions, *inc* + *batch* training, l. coefficient of 0.1 + 0.2, 100 + 1000 epochs.
- *Logsig* + *softmax* activation functions, *inc* + *batch* training, l. coefficient of 0.1 + 0.8, 100 + 10000 epochs.
- *Hardlim* + *softmax* activation functions, *inc* + *batch* training, l. coefficient of 0.1 + 0.8, 100 + 10000 epochs.

- *Logsig* + *elliotsig* activation functions, *inc* + *batch* training, l. coefficient of 0.1 + 0.8, 100 + 10000 epochs.

Through the analysis of the confusion matrices represented above, it is possible to deduce that:

- The test results obtained are in line with the observations made in the training phase, insofar as neural networks with better training/validation performance behave more optimally when stimulated by previously unknown inputs.

- Overall, the accuracy of the OCR system is quite decent as a considerably small number of training inputs was generated. In real situations, the size of neural network training datasets reaches the order of millions.

- The generalizability and error resistance of the classification system is high. This result is directly related to the way the training dataset was structured. Taking into account the remaining training results of the neural networks and the confusion matrices (see .csv in data/), it is possible to extrapolate their percentage of well classified inputs to > 80%, in general.

- The filter + classifier architecture presents more consistent results (>82% accuracy, on average). This is probably due to the noise-reduction effect of the filter.

- The classifier with two layers achieves better accuracy than the rest. This result comes from the fact that neural networks with hidden layers are better able to demonstrate non-linear behaviour.

5. Conclusions

The realization of this work is clearly relevant to the understanding of how neural networks serve a fundamental role in the area of machine learning, namely in classification and recognition problems. However, it is necessary to get to the bottom of how they are composed, in terms of their activation functions, weights, biases, training functions, etc. This is because it allows us to make better decisions when designing a potential optimal architecture of neural networks. Unfortunately, this knowledge is not always enough. The performance of neural networks varies unpredictably depending on the value of their hyperparameters. This represented a great difficulty for the working group when defining the neural network configurations: which activation function should I use? Is this learning coefficient too high and could cause unstable training? Should I use x or y training epochs? To get around this indecision, the group then chose to apply the iterative and exhaustive algorithm discussed above. Despite the time spent on this optimization problem (long hours with 2/3 computers connected with the processor at full power), we want to believe that it was worth it and, in fact, we managed to achieve an efficient *OCR* classification system as far as possible. The following statement is somewhat paradoxical: neural networks are trained using function optimization, but they themselves face the serious problem of optimizing their hyperparameters.