Recognizing Digits using Neural Networks
Report

BOUHADANA Samuel - CASSARD Sébastien - LAUQUIN Julie

March 2018

The MNIST dataset is commonly used for benchmarking purpose in machine learning research. It contains images of handwritten digits from 0 to 9.

The goal of this project is to recognize these handwritten digits via neuronal networks. You will find here the results we obtained using Keras (error rate on test images), and the comparison graphs of the effects of the different hyper-parameters on our model.

Contents

1 Presentation **2**
- 1.1 General . . . 2
- 1.2 Data Augmentation . . . 3
- 1.3 Results . . . 3

2 Comparison of Hyper-parameters effects **4**
- 2.1 Activation Function . . . 4
- 2.2 Batch Size . . . 4
- 2.3 Convolution Window . . . 4
- 2.4 Epochs . . . 4
- 2.5 Pooling Window . . . 5

3 Loss functions and optimizers **10**
- 3.1 Loss Function . . . 10
- 3.2 Optimizer . . . 11

4 Comparisons between different artificial network architectures **12**

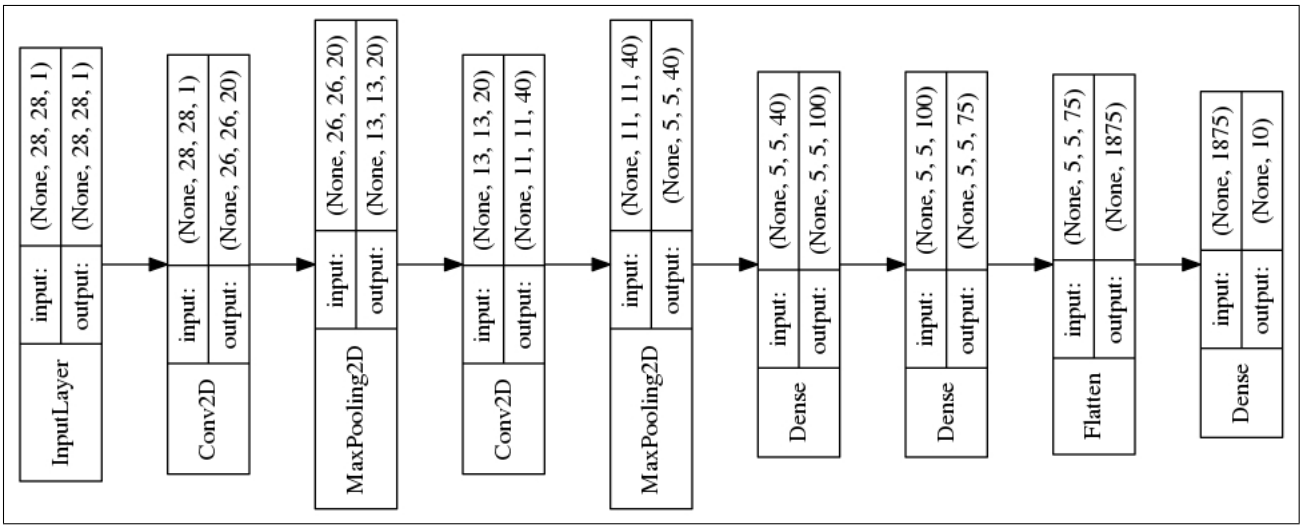
5 Conclusion **13**

1 Presentation

1.1 General

To create our handwritten number recognition system, we chose the following architecture :

- a layer of 10 convolutional neurons
- a layer of 20 convolutional neurons
- a layer of 100 fully connected neurons
- a layer of 50 fully connected neurons
- a last layer of 10 fully connected neurons



More precisely, all neurons have as activation function the function `relu` except for the last layer for which we preferred to use `softmax`. Since the last layer consists of 10 neurons for the 10 digits to be recognized, it made more sense to use the `softmax` function.

For the convolutional neural networks, we chose a 3×3 kernel size and a 2×2 pooling size.

We trained our model on batches of 32 training images (`batch_size=32`) and on 10 epochs (`epochs=10`), with the loss function `categorical_crossentropy`, and the `adam` optimizer. This decomposition of the sample into batches allows a greater speed for the same result as a single training on the entire data.

1.2 Data Augmentation

Data augmentation consists in applying transformations to the training pictures in order to make the training data more rich. For each of the MNIST picture, a random rotation of up to twenty degrees and a random zoom between 90% and 110% is applied before feeding the picture to the neural network.

1.3 Results

With our system, we've achieved 99% accuracy on the 10,000 images in the test database. This is a fairly high result given the number of test samples.

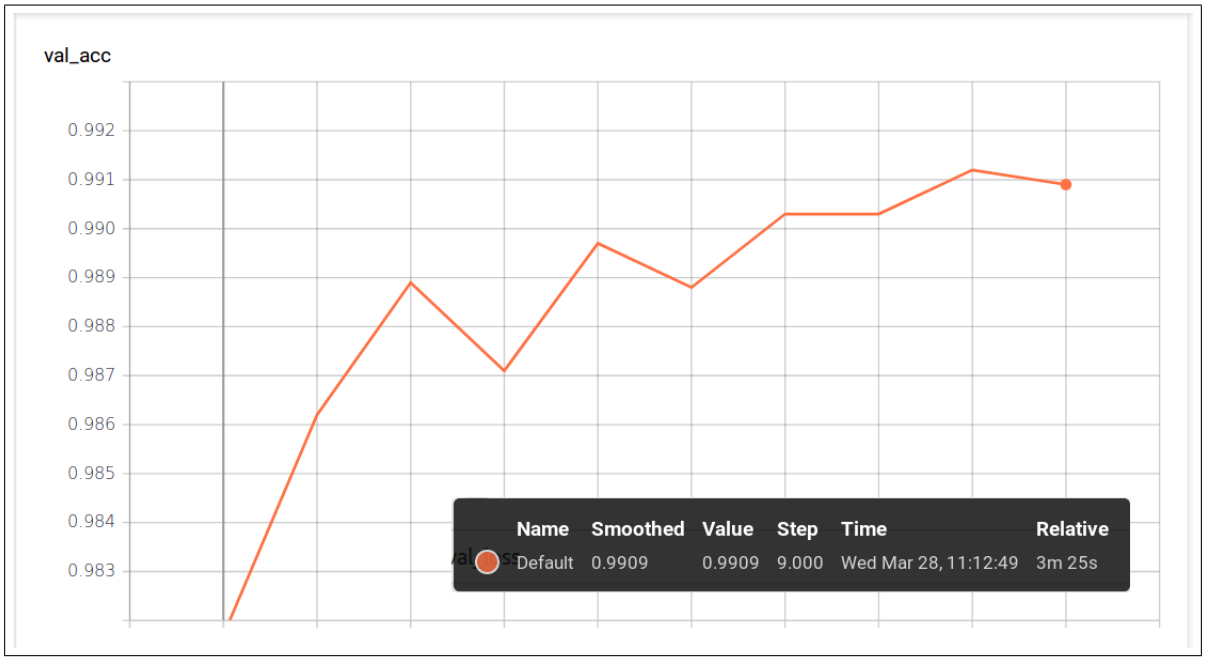


Figure 1: Accuracy of our neuronal network (default)

If we look in more detail at the different cases of failure of the system, we can see that some of the errors come from tricky cases with digits difficult to read, even for a human.

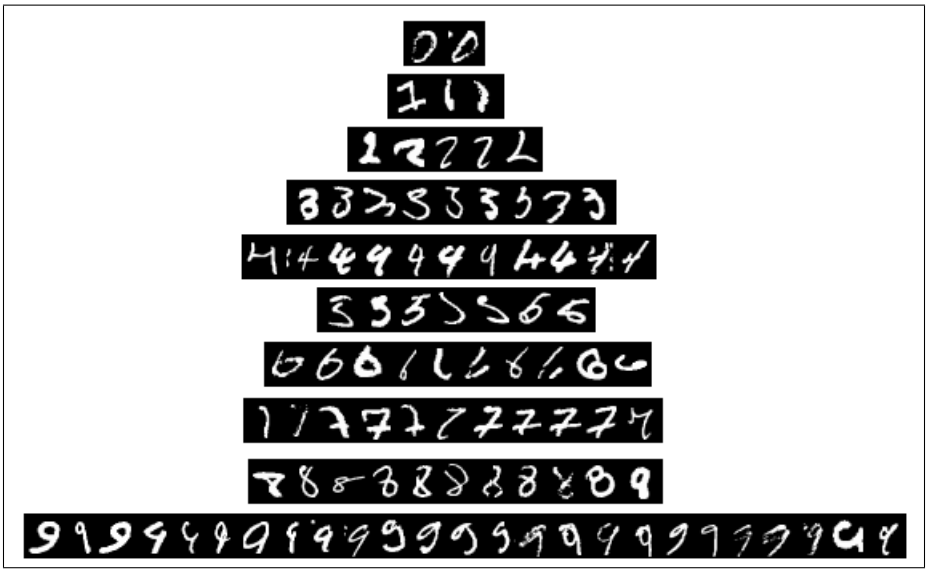


Figure 2: Figures on which the system failed

Also, we notice that the most present number in cases of failure is the 9, and the least present the 0 and the 1. This can be explained by the fact that 9 can be confused with many other digits, such as 3, 5, or even 0.

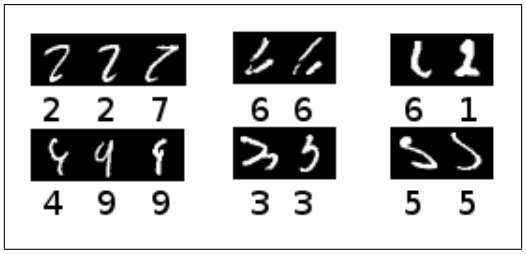


Figure 3: Some tricky cases

These errors might be due to the fact that some numbers like 1 and 7 may have variations in writing.

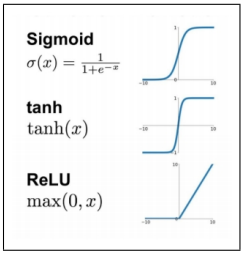
2 Comparison of Hyper-parameters effects

In an attempt to improve our neural network, various hyper-parameters have been tweaked. The resulting graphs can be found at the end of the paragraph.

For each graph, the curve labelled with `val_acc` corresponds to the precision of our system, the other one labelled `val_loss` corresponds to the cost function.

On each graph, the `default` curve was obtained with the characteristics described in 1.1.

2.1 Activation Function



The Activation Function "squashes" the output value of a neural layer to a range.

The `tanh` and `relu` activation functions are faster than `sigmoid` and `softmax`.
`softmax` is less effective than the other three activation functions, so it is better not to use it.

2.2 Batch Size

The batch size does not significantly change the result.
A small batch takes longer to calculate but converges faster.

2.3 Convolution Window

There is a slightly higher convergence for a larger convolution window.

2.4 Epochs

We notice that above a certain threshold ($epochs = 10$), it is useless to increase the number of epochs:
the values of the cost and precision function are always within the same interval and oscillate around a step value.

2.5 Pooling Window

A large pooling window decreases the accuracy of learning, which is normal because the more you look in detail at an image, the more effective the algorithm is.

Figure 4: Activation functions comparison

Figure 5: Batch size comparison

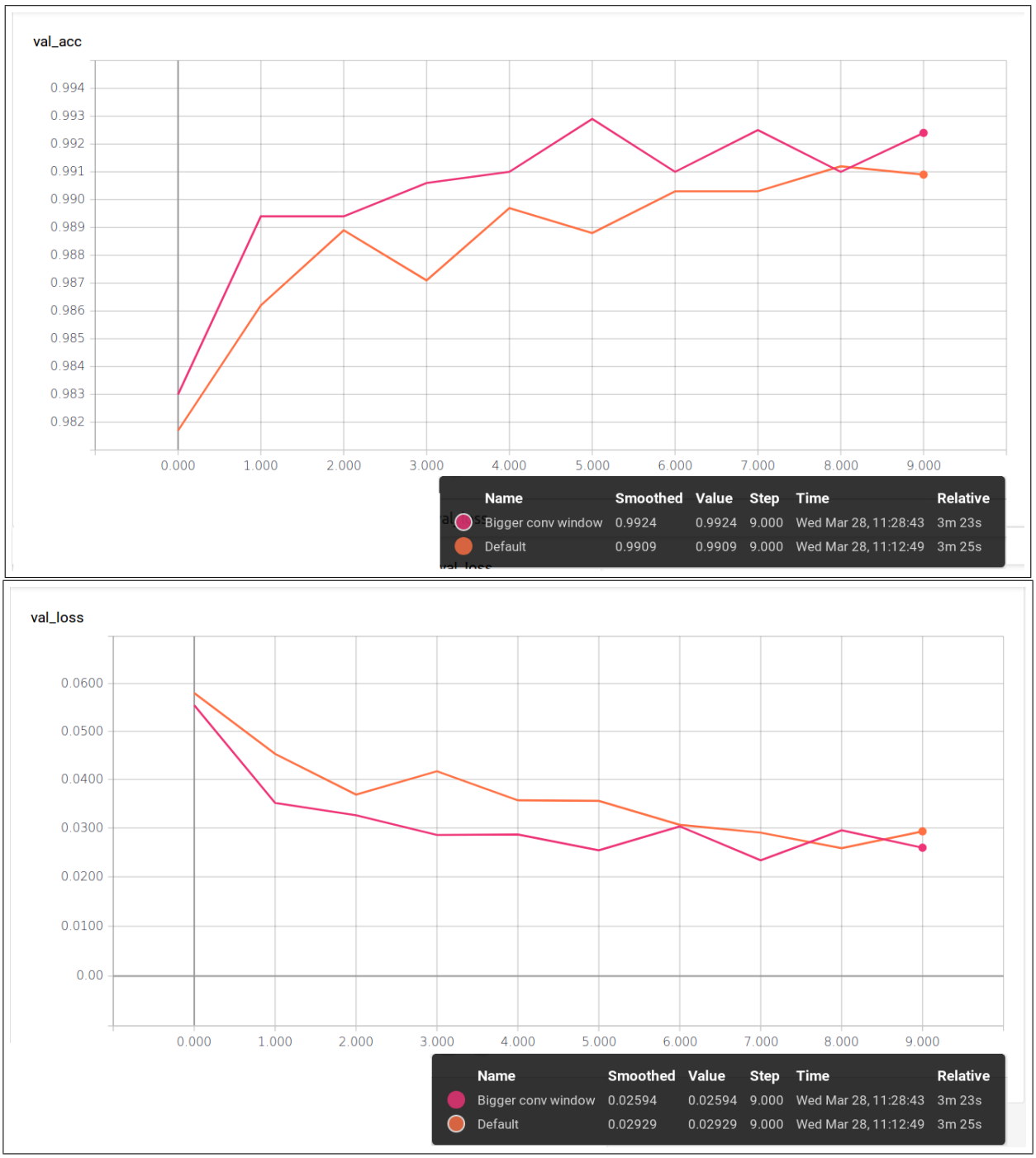


Figure 6: Convolution window comparison

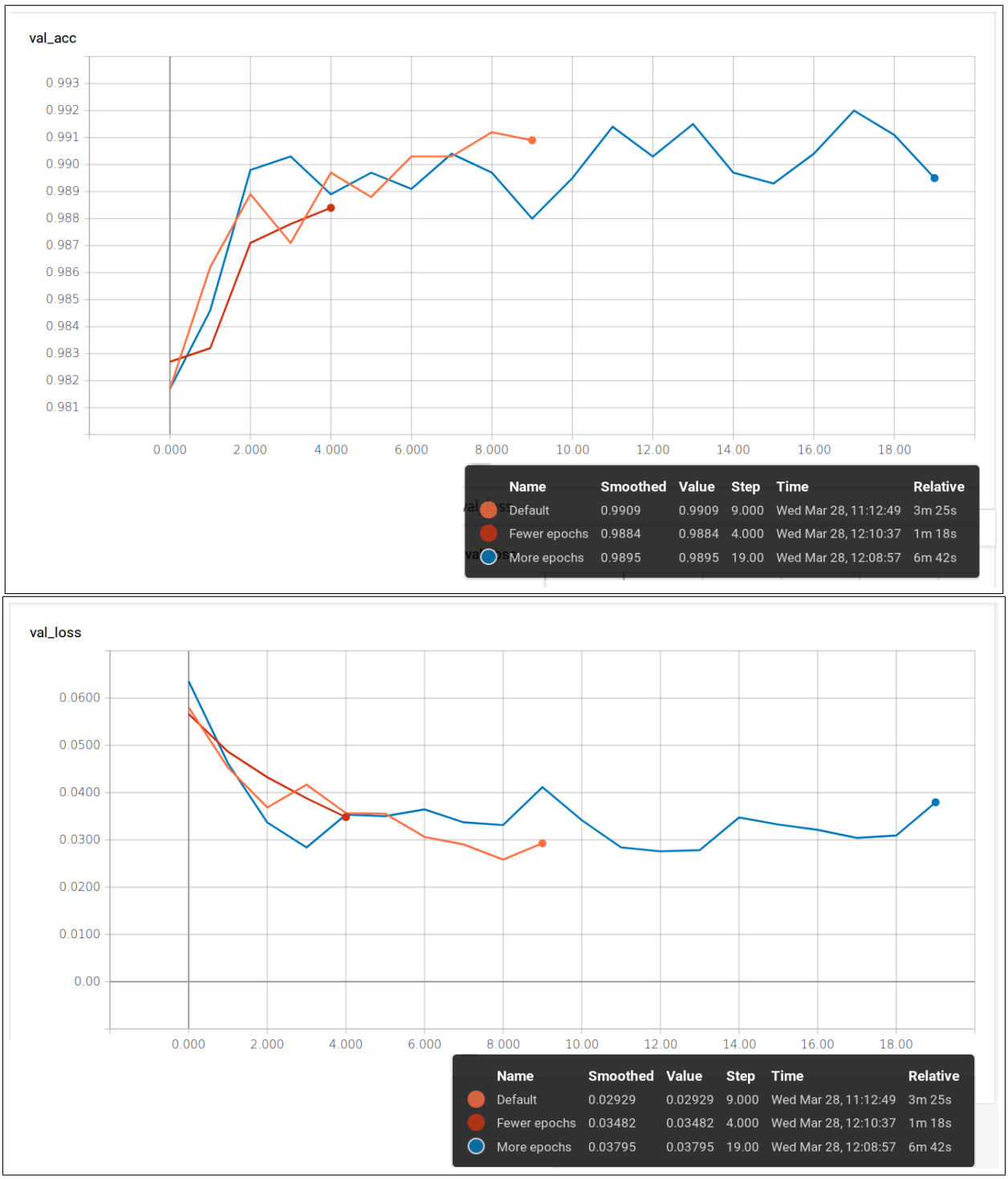


Figure 7: Epochs comparison

Figure 8: Pooling window comparison

3 Loss functions and optimizers

In addition to the hyperparameters, we also tested different loss functions and optimizers. The resulting graphs can be found at the end of the paragraph.

In this paragraph, the curves labelled with `val_acc` corresponds to the precision of our system, the other one labelled `val_loss` corresponds to the cost function.

On each graph, the `default` curve was obtained with the characteristics described in 1.1.

3.1 Loss Function

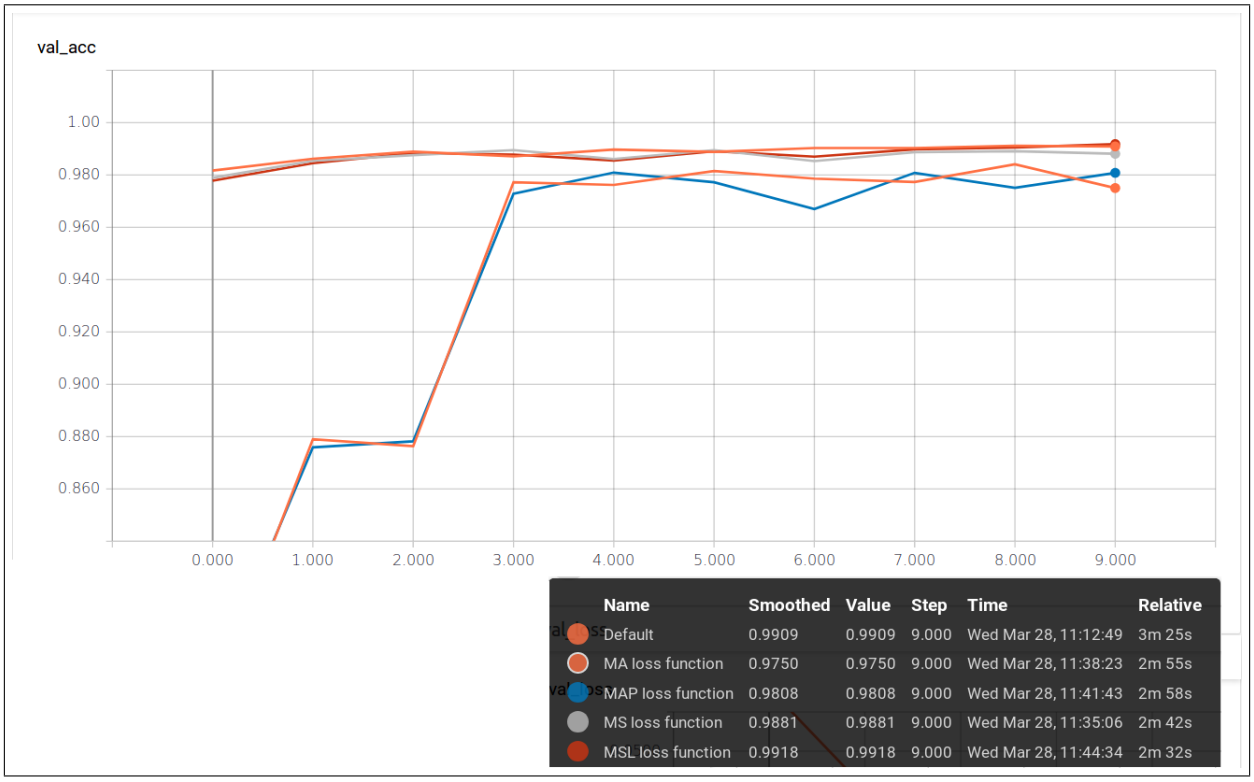


Figure 9: Loss functions comparison

With these results, we find that categorical crossentropy, MSL, and MS are equally effective. But categorical crossentropy takes a little longer to calculate.
For MAP and MA, convergence is much slower and the results slightly lower.
MSL here seems the best loss function, it has the shortest calculation time for the best result.

3.2 Optimizer

Figure 10: Optimizers comparison

Considering the results obtained, it seems that the adam and RMS optimizers give relatively similar results. However, if we look at each epoch, RMS is much less stable than adam.
As for Adagrad, this optimizer is slower to converge and gives weaker results. With these results, adam seems the best optimizer, for its stability and good results.

4 Comparisons between different artificial network architectures

We tested different neural network architectures to compare their performances. We compared a system consisting only of fully connected layer, a system exclusively made of convolutional neurons, and a hybrid system composed of the 2.

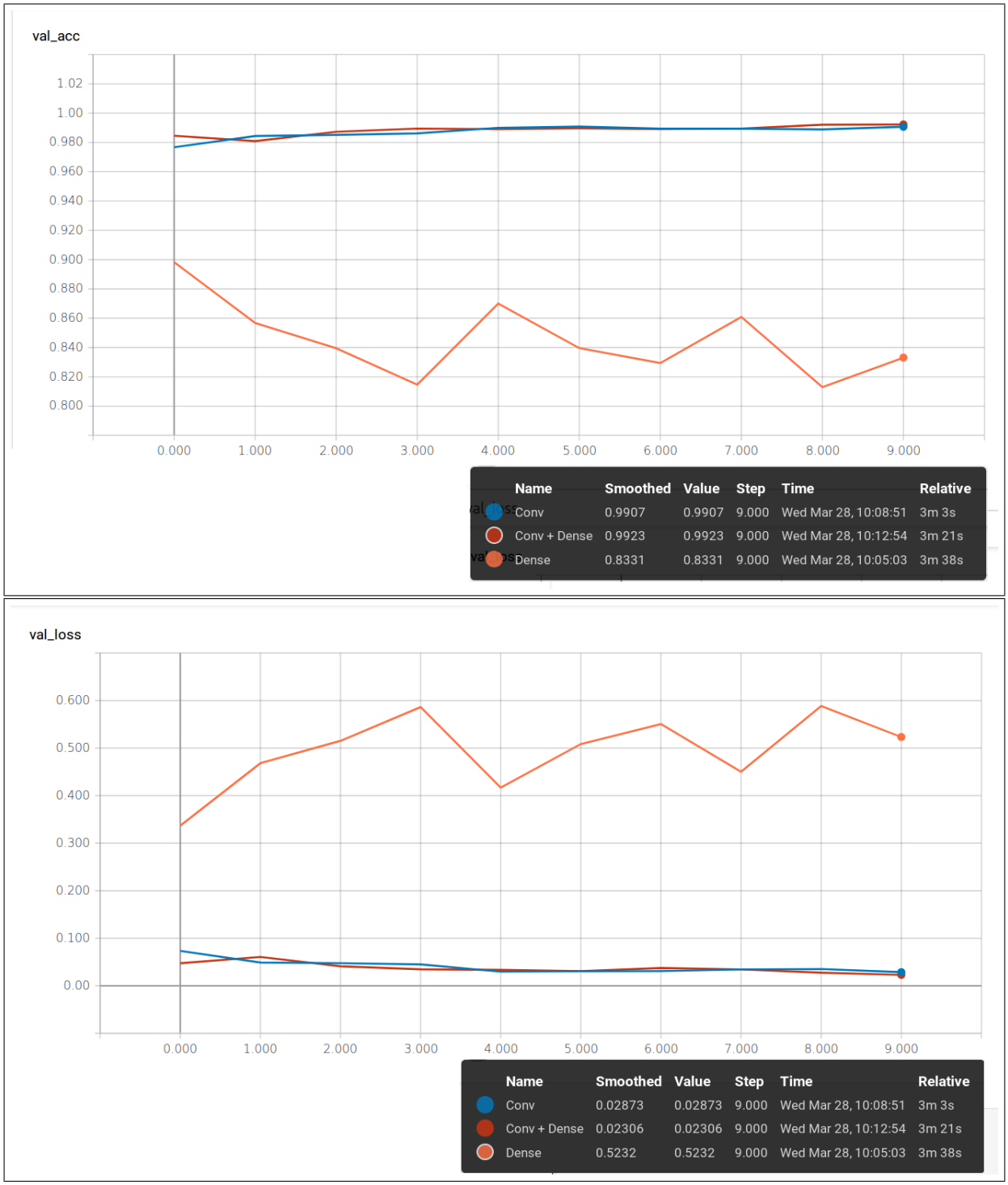


Figure 11: Comparison of different neuronal network architectures

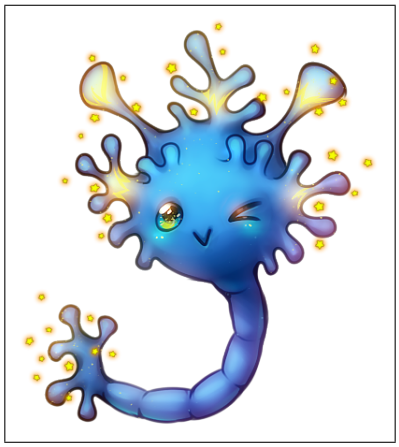
From these two graphs, we can see that architectures made up of convolutional neural networks are more efficient than those made up of fully connected layers.

Both hybrid and convolutional versions give substantially identical results, as much in terms of accuracy as in computing time.

5 Conclusion

With 99% test success, our system is very efficient with good accuracy.

This handwritten number recognition system can be extended to the recognition of the handwritten alphabet (upper and lower case) and symbols. It could then be extended to word or even sentence recognition.



"I'm a cute human neuron !"