Machine Learning - Lab 3: Neural Networks

Tuesday 11h, Group 13: João Cruz, number 84395 - João Rato, number 84397[1]
[1]*Instituto Superior Técnico, Lisboa*
(Dated: 16/11/2019)

I. INTRODUCTION

The human brain served as inspiration to the concept and development of artificial neural networks. With their origins dating back to the 1940s in the simple and limitative McCulloch & Pitts model, they have evolved into a very powerful Machine Learning tool. These first models consisted on one single unit that computed a weighed sum on the inputs, an then applied a non-linear activation function (in these cases, a step function that would depend on some threshold) on the result, after which one would obtain the output. Nowadays, there are several types of neural networks, more advanced and efficient. In this work, we focus on two of them: the Multilayer Perceptron (MLP) and the Convolutional Neural Network (CNN).

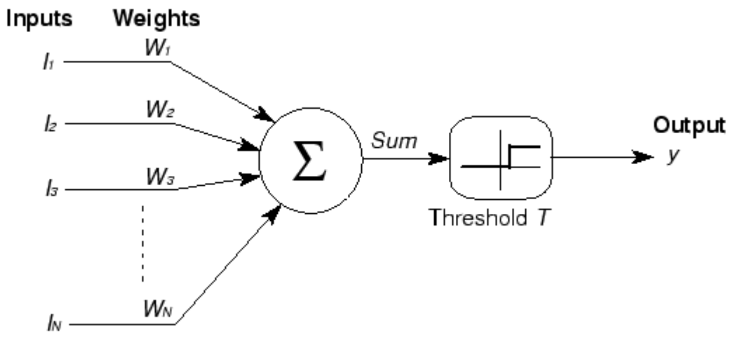

FIG. 1: McCulloch & Pitts model of a neuron

A. Multilayer Perceptron

The fact that the first models used only one unit means that it was impossible to model more complex patterns. As such, the Multilayer Perceptron (MLP) was created in order to combat this issue. It consists on several units that are usually organized in layers. Moreover, the activation function of each unit is continuous and differentiable. In this case, one has an input layer, some number of hidden layers, and an output layer, each with as many units as one deems fit.

Cybenko's theorem is a powerful result that essentially guarantees that an MLP with 1 hidden layer with a finite number of units can approximate a wide variety of continuous functions, something that, for example, McCulloch & Pitts could not. However, the optimal architecture (number of layers, number of units per layer, etc.) depends on the problem. The neurons of each layer communicate with those of the adjacent layers, and each connection between units has a weight associated to it (as do the inputs). Usually, in classification problems, one works with fully connected layers, i.e., every unit of each layer communicates with every unit of the adjacent layers. It is important to keep in mind that every unit can also have a bias term.

Training the network consists in finding the optimal weights that, for a set of input training data, better replicate the desired outputs. This can be done, for example, through gradient descent, using the backpropagation network to know how much a weight influences the final outputs.

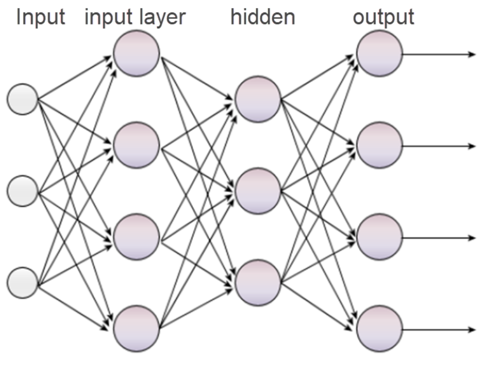

FIG. 2: Multilayer Perceptron

B. Convolutional Neural Network

Convolutional Neural Networks (CNNs) are used mainly in image recognition, since they accomplish it remarkably. Their main achievement is that they learn by themselves which features are best in order to correctly classify images, which solves a big problem since in some cases it might not be evident at all which features would better help a computer distinguish between two pictures.

As seen in Fig. 3, CNNs have essentially two different phases. The first one is what characterizes them, and it's how they learn the features. This happens through a sequence of alternated convolution and pooling layers.

The convolution layers receive an image input (which are sets of information about the pixels, such as the RGB values of each pixel) and use kernels to process them into the outputs. Kernels are essentially filters that are applied to the inputs (for example, going through all groups of 4 adjacent pixels and averaging the colour values in that "square"'s diagonal). By applying several of these kernels on the entire image, the CNN will capture the distinguishing features of the pictures.

The pooling layers are there to make the whole process more efficient. Images can be very large sets of data, which becomes an even bigger problem when working with huge datasets. As such, these layers try to preserve the output from the convolution layers while reducing the overal size of it, reducing the computational power necessary to process the network's tasks. A common example of pooling (which is the one used later in this work) is Max Pooling. In this case, it selects the maximum value of a portion of a convolution layer output, and discards the other values within that group. There are different ways of doing this, but, for example if one has a 4x4 matrix and max pools groups of 3x3, it reduces the matrix's dimensions down to 2x2.

Both of these processes are iterated as many times as required. In the end, the network should the relevant features clearly defined.

Afterwards, the output from the last convolution or pooling layer is flattened and fed to a fully connected layer, which is often used in classification problems. Essentially, this second part is an MLP.

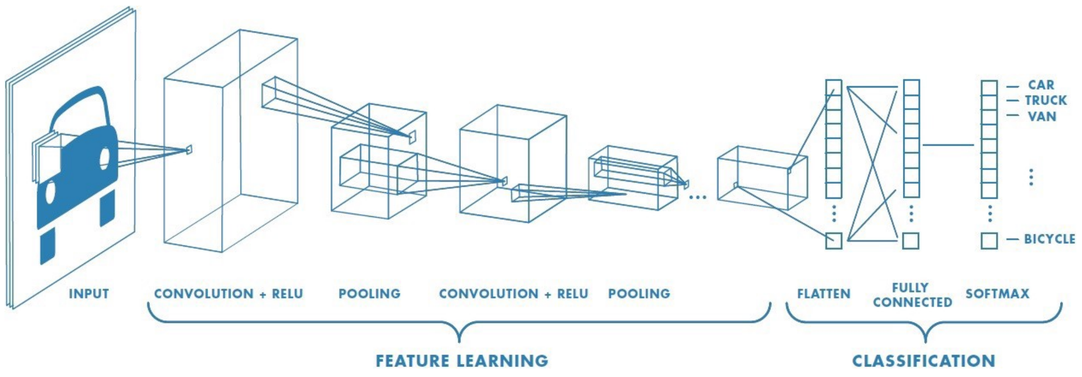

FIG. 3: Convolutional Neural Network

II. FRAMEWORK AND DATA PREPARATION

A. Goal and type of data used

The main goal of this work is to properly identify an handwritten number. The task at hand will be a classification one, relying on supervised learning. The data used for this process are grey scale images (where pixels have a value between 0 and 255: from black to white) with 28x28 pixels each.

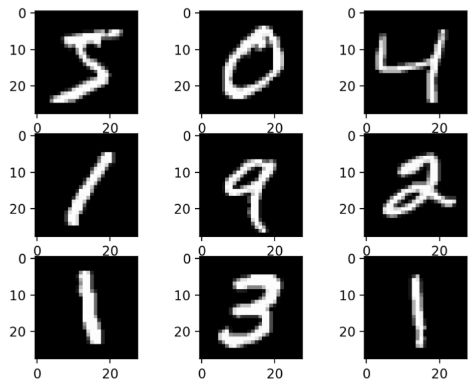

FIG. 4: Illustration of the MNIST dataset

B. Data segmentation

1. *Dataset format*

The data set used for the training of both network types contains 3000 images like the ones shown in Fig 4. On the other hand, the test set is composed of 500 images and, as the name suggests, it is used to test the performance and accuracy of both models. After loading the sets into *numpy* arrays, it is possible to verify that the shapes of those are: (3000,28,28,1) and (3000,) for the train input and labels and (500,28,28,1) and (500,) for the test input and labels, respectively, confirming what was stated beforehand.

The images that follow represent some elements of the training and test data.

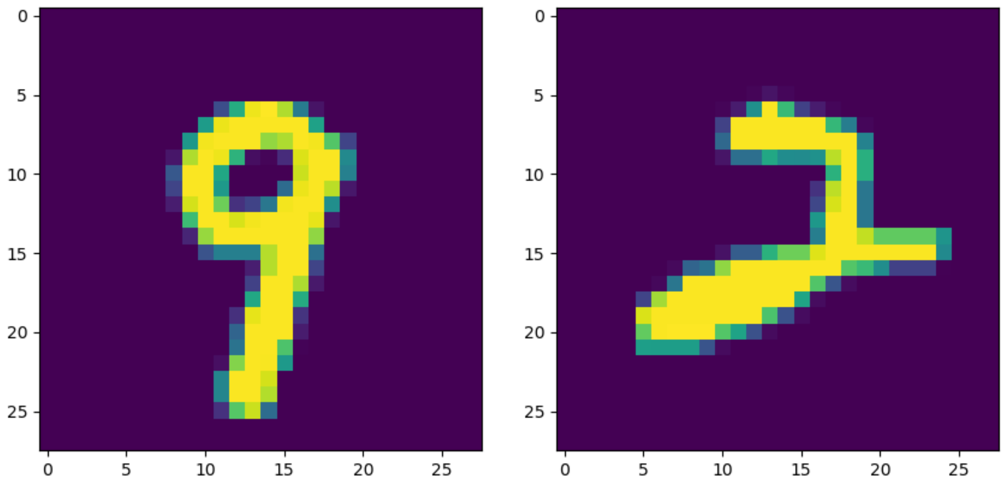

FIG. 5: Display of the first digits in the training and test sets, respectively. The labels for these are ’9’ and ’2’.

2. *Useful alterations to the data*

Some changes to the data were made. For the remainder of the work, the grey scale values were normalised (now being from 0 to 1) for an easier handling of the values. Additionally, a one-hot encoding approach was taken for the label vectors (y). Instead of each element being a number from 0 to 9, representing each class, they were changed to a vector, transforming the 1-dimensional array into a matrix. These vectors are of size 10 (because there are 10 classes) and are full of zeros except at the element that represents the class of the digit. As an example, if a label in the previous format was a ’7’, it is now represented by the one-hot vector: [0 0 0 0 0 0 0 1 0 0].

3. *Validation set*

Finally, in an effort to reduce overfitting by the method of early stopping (a standard regularisation method), a validation set was created from the existing training data. Precisely 30% of that set was chosen to compose this new set: now the training set has 2100 images while the validation set has the remaining 900.

After updating the weight values of the network using the training set, the validation data is used in the forward network so that the validation loss can be computed. Then, while this cycle keeps running, this loss is expected to decrease. If, however, the validation loss increases, that’s a warning sign for overfitting and, depending on the patience parameter (this will be discussed ahead), the algorithm is halted: early stopping.

III. MODEL A: MULTILAYER PERCEPTRON

A. Architecture

As a first attempt to build a model which classifies the digits into their respective labels, an MLP was built. As explained in the *Introduction* section, this type of models feature fully connected layers, meaning that there is a weighted link between all the units of adjacent layers.

This MLP was created using the *keras* framework for *Python*. With the appropriate commands and methods, a model was created with the following architecture.

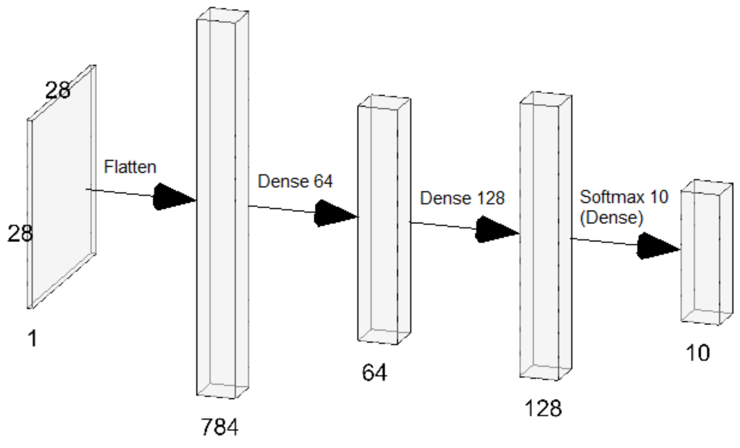

FIG. 6: Architecture used for the MLP model

More specifically, the Flatten layer transforms the 2D image into an 1D array. Moreover, the Dense layers are fully connected layers with ReLU activation functions and the Softmax layer functions as a Dense one without ReLU, where the output is a vector of 10 elements with the probability of that particular image belonging to each class.

B. Other network specifications

1. *Early stopping methodology*

As said before, early stopping is a good regularisation method that avoids overfitting. By checking for the increase of the validation loss one can stop the training of the network before this problem occurs. The addition of the patience parameter allows one to stop the training after a certain amount of successive epochs performed worse than the previous best. This exists so that the training process is given an opportunity to get across flat spots or find some additional improvement, instead of stopping it right away after one epoch outputs a worse result than the best one yet. In this case, the patience was set to 15 epochs.

By setting the parameter *restore_best_weights* to True, it is possible to recover the weights that were used in the best epoch and not in the last one (after the patience). By doing this, there is certainty that the model is the best that was achieved during this specific training process.

2. *Loss function*

Using the architecture and early stopping method described above, the MLP was fitted to the previously prepared data. As a loss function, cross entropy was used (in the categorical format, which uses one-hot encoding vectors). This loss function type is defined with logarithms and is obviously of an higher value if the predicted probability of the image having the label of the real class is low. This is better depicted in the following image.

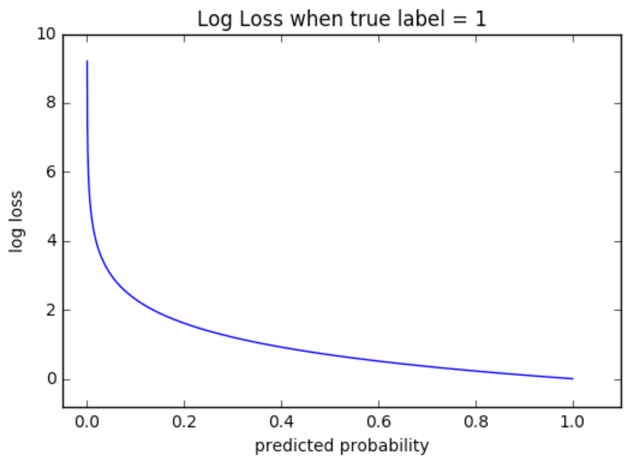

FIG. 7: Loss values using Cross-Entropy[2]

3. Optimiser

The optimiser is the algorithm that backpropagates and updates the gradient. There are a lot of different ones but Adam was the chosen optimiser for this work.

It is a type of stochastic gradient descent, which combines the advantages of two other extensions of stochastic gradient descent: Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp). Both adapt the learning rate as training proceeds. The first one is effective on problems with sparse gradients while the last one is better for online and non-stationary problems. Adam is a mesh of these two and is highly effective for general cases, while being computationally efficient.[3]

4. Hyperparameters

Hyperparameter search is always an important task in machine learning. By finding the optimal architecture, learning rate, etc., one can have much better results. For this work, the hyperparameters were already set.

Therefore, the starting learning rate was set to 0.01 and the *clipnorm* argument was put to 1. This helps avoid gradient explosions since it normalises the gradient vector. Moreover, the batch size was set to 300, which means that the gradient was updated after 300 samples are passed through the network.

Finally, since there is the possibility of every epoch improving the validation loss, a maximum number of epochs for training was defined: 400. Additionally, this value will be used for training procedures with no early stopping.

C. Main results

1. With Early Stopping

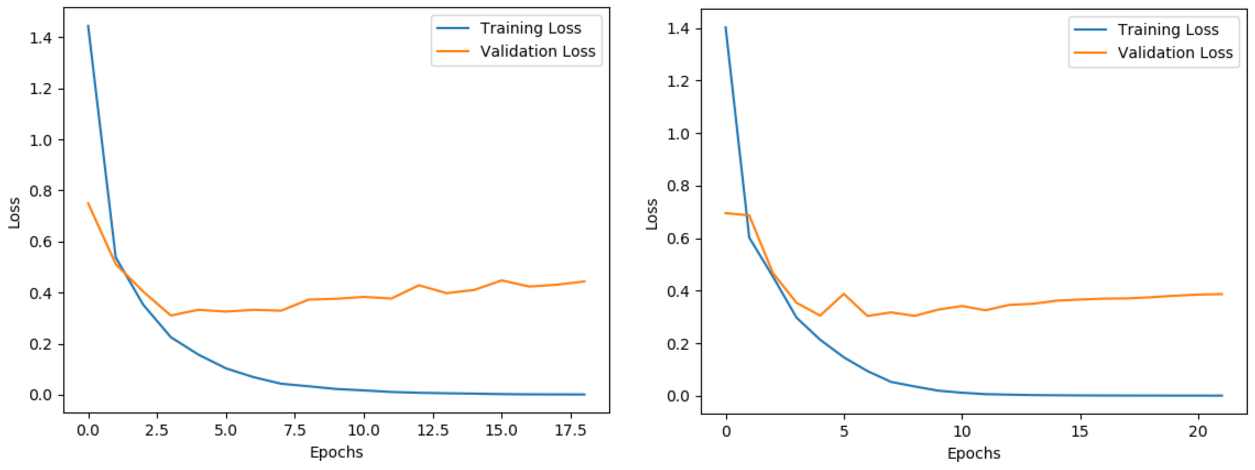

FIG. 8: Evolution of the training and validation losses with early stopping in 2 scenarios

In Figure 8, two different executions are shown. First of all, it is important to justify why the end results are different when the data is the same - remembering that the split between training and validation sets is done randomly but the seed is always the same, so from one execution to another the sets are the same. The reason for the losses being different each time the code is ran lies on the nature of both the MLP initialisation itself and the optimiser. As stated before, the Adam optimiser is a kind of stochastic gradient descent which, as the name suggests, introduces some randomness into the system, which in turn can modify the results a bit. However, most of the visible difference comes from the initialisation of the MLP weights. The usual practice which coincides to what was done in this work is to set the initial weights of the layers with a random number, which applies each time the code is ran. This obviously causes the end result to be different since the gradients will also be substantially different.

Secondly, the two images are shown to emphasise the importance of the patience parameter in the Early Stopping method. Analysing the graphs in detail, one can see that on the first case the 4th epoch happened to have the lowest loss (this is Epoch = 3 in the figure, since the first epoch was taken as 0) and all the following epochs failed to improve this result. This lead to the algorithm being forced to a stop at epoch 19, since that is precisely 15 epochs after 4. It is right to conclude that running these 15 additional epochs proved to be an useless effort.

On the other hand, however, in the second case, the best epoch was the 5th (Epoch = 4 in the graph) initially. Then, the following epoch worsened the loss and the patience counter started. Nevertheless, the 7th epoch produced a better result than the 5th and, as such, reset the patience counter. After this one, all the remaining epochs yielded worse results and the algorithm stopped at 7+15 = 22 epochs. This serves to show the exact purpose of patience. If there was none, the algorithm would have stopped the training at a sub-optimal state. Instead, by waiting out a bit, the model was improved, even if not by much, in this case.

For the remainder of the analysis, the case corresponding to the image on the right of Figure 8 was considered, since it had a smaller validation loss.

To further analyse the performance of the model, the accuracy was measured using the test set. The value for this was 0.912, meaning that around 91% of the images in the test dataset were correctly placed under their label.

Furthermore, it was also possible to verify which classes were more susceptible to be confused with one another. The tool to visualise this is the confusion matrix, in which the columns represent the predicted labels and the rows the true labels. For this case, the confusion matrix was:

$$\begin{bmatrix} 42 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 49 & 0 & 2 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 54 & 3 & 0 & 0 & 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 55 & 0 & 0 & 0 & 1 & 2 & 0 \\ 2 & 0 & 0 & 0 & 43 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 2 & 38 & 1 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 1 & 38 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 1 & 0 & 0 & 50 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 52 & 0 \\ 0 & 0 & 1 & 0 & 5 & 0 & 0 & 3 & 0 & 40 \end{bmatrix}$$

This showcases what was already expected. Since the accuracy is fairly high, the diagonal has larger numbers than the other elements (the intersection of both the predicted and true labels for the same class is high).

Additionally, one can see a few more interesting patterns. The largest off-diagonal value is 5 and is the confusion between 4 (predicted) and 9 (true). This is also expected since both numbers are usually also confused by humans by the nature of being similar in handwriting. Despite this introducing an error into our model, it is one that showcases realism and that somehow confirms that its training resulted in having similar principles as humans, in what regards distinguishing digits.

Furthermore, there is also a fair amount of confusion between the digits 3 (predicted) and 2 (true), 2 (predicted) and 7 (true) and finally, 7 (predicted) and 9 (true), all with 3 occurrences, which are also fairly similar numbers.

2. Without Early Stopping

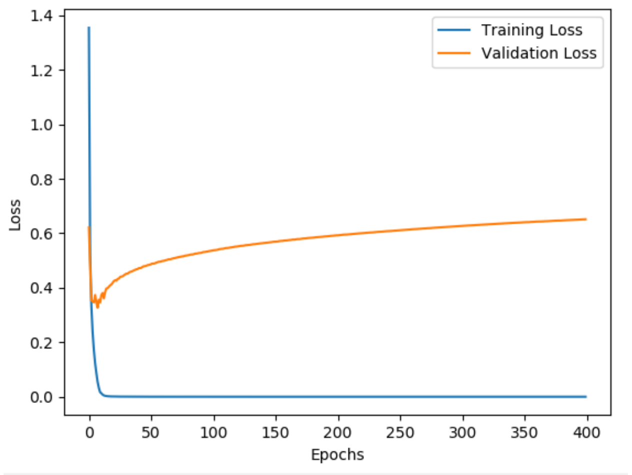

FIG. 9: Evolution of the training and validation losses without early stopping

By turning off the Early Stopping mechanism, the algorithm runs through all 400 epochs with no care for the validation loss. Obviously this means that the computation time increases quite a bit. In Figure 9, it is visible that this loss increases logarithmically. Additionally, as expected, the training loss comes really close to 0, since it overfits the data, correctly predicting all the labels in the training set.

Moreover, it can be stated that the epoch which yielded the best results for the validation loss was the 8th Epoch, but since there was no early stopping, the model was trained 492 more epochs. Bear in mind that the *restore_best_weights* parameter does not come into play here since that was part of the early stopping method. This then means that the weights of the MLP are the ones calculated at Epoch 400 rather than Epoch 8. Because of this, the final model will be highly specialised towards the training data and not the rest (as shown by the increase of the validation loss).

As before, to further analyse the network, the accuracy and confusion matrix were computed. For this specific run-time, the accuracy was 0.932 which means that about 93% of the images in the test set were correctly identified by the model. The fact that this value is higher than that of the case with early stopping is interesting and will be approached in the final section of this work.

The confusion matrix was:

$$\begin{bmatrix} 41 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 51 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 54 & 2 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 54 & 0 & 0 & 0 & 1 & 2 & 1 \\ 1 & 0 & 0 & 0 & 42 & 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 40 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 40 & 0 & 0 & 0 \\ 0 & 0 & 2 & 1 & 0 & 0 & 0 & 51 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 49 & 1 \\ 0 & 0 & 1 & 0 & 2 & 0 & 0 & 1 & 1 & 44 \end{bmatrix}$$

It is visible that now there exists a bit more dispersion among the confused labels as opposed to before, where the amount of confusion was greater for a specific pair of labels (example: labels 4 and 9 had 5 wrongly attributed data points).

This time, the predicted labels 2, 3, 6, 7 and 9 were the most prone to confusion where the confusion between 9 and 4 was still noticeable. Overall, the same pairs of labels that were mentioned before were still the ones that got confused the most.

IV. MODEL B: CONVOLUTIONAL NEURAL NETWORK

A. Architecture

We then move towards the CNN, whose speciality is image classification. The architecture employed is shown in Fig. 10 and was implemented, as in the previous task, using the *keras* framework for *Python*.

Its feature learning part (cf. *Introduction, Convolutional Neural Network*) is comprised of a convolution layer followed by a Max Pooling layer. Both of these layers are then repeated once.

The output from the final pooling layer is then flattened and fed to a Dense layer similar to the ones in the MLP, followed by a Softmax layer which, once more, works the same way as before.

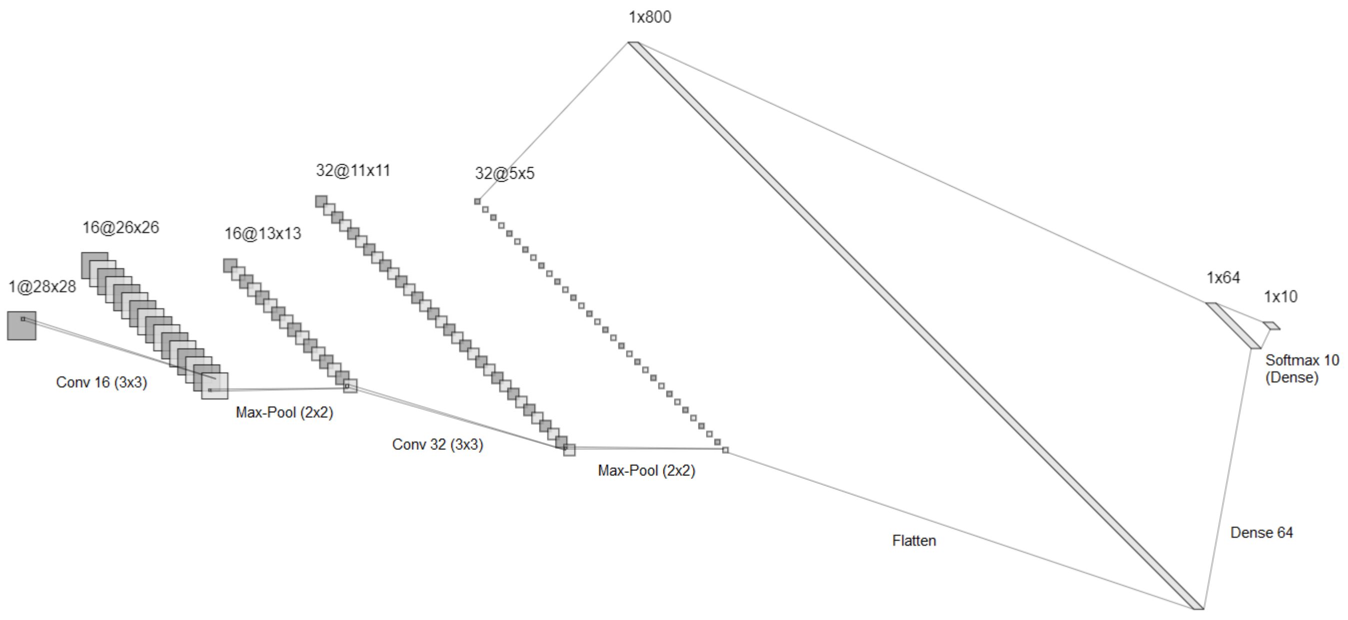

FIG. 10: Architecture used for the CNN model

B. Other network specifications

Everything regarding the Early Stopping, loss function, optimiser method and the hyperparameters is the same as in the MLP, and as such will not be repeated. Note that the CNN was only run using Early Stopping, unlike in the previous task.

C. Main results

In Fig. 11, the evolution of the training and validation losses with the epochs is shown.

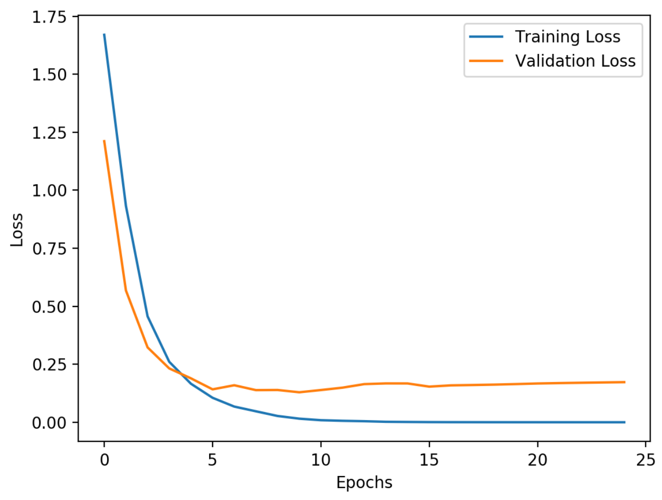

FIG. 11: Evolution of the training and validation losses

Once more due to the random nature of the method (the initialisation of the parameters and the Adam optimiser), the output of the network is always slightly different. However, that has been addressed in the MLP and there's no need to repeat it.

Comparing Fig. 8 with Fig. 11, it can be seen that the CNN takes more epochs to run, however the validation loss drops more and faster. While the training loses, albeit slightly different, have similar behaviours, the validation loss line in the CNN crosses the training loss line in a higher epoch (evidence for faster dropping). Moreover, the validation loss starts stabilising at a value lower than 0.25, whereas in the MLP it stabilises almost at 0.4.

This showcases firsthand that the CNN performs better than the MLP. It is important to keep in mind that it is unrealistic to expect an overwhelmingly better performance from the CNN, though, since the MLP already did quite well. However, it is simply and objectively better.

The accuracy of the model was computed, and was found to be 0.964 for this case. This means that this network got around 96% of the test classifications correct, which is an increase of 5% comparing to the MLP, a considerable improvement.

The confusion matrix was also determined, and is as follows:

$$\begin{bmatrix}
43 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 51 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 59 & 1 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 56 & 0 & 0 & 0 & 1 & 0 & 1\\
1 & 0 & 0 & 0 & 44 & 0 & 1 & 0 & 0 & 0\\
0 & 0 & 0 & 1 & 0 & 40 & 1 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 41 & 0 & 0 & 0\\
0 & 0 & 3 & 0 & 1 & 0 & 0 & 50 & 0 & 0\\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 51 & 1\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 49
\end{bmatrix}$$

It is observable that most of the misclassifications in the MLP's confusion matrix disappeared. The highest that remains (occurring 3 times) is the confusion of true 7s with 2s. However, overall, it is an improved model.

V. FINAL COMMENTS AND CONCLUSIONS

Having successfully implemented both required networks, there are some considerations to be made. First of all, as expected, the CNN achieved a higher performance (with an accuracy around 96%, compared to the MLP's 91%), which is visible by comparing validation losses, accuracies and confusion matrices.

However, the MLP has its merits, and it's worth it to analyse the unexpected fact that, without Early Stopping, the accuracy is higher. As such, a comparison between the cases with and without Early Stopping is required. Without it, the execution time is evidently longer, since all the 400 epochs are computed. This also means that the model will be overfitted, since we know from the Early Stopping that around 8 epochs would be enough. This is backed by the fact that the training loss without Early Stopping is of the order of 10^{-6}, with a validation loss of

≈ 0.7 whereas with Early Stopping we obtain a training loss of the order of 10^{-3}, with a validation loss of ≈ 0.4. And even though we have an overfitted model, we still get a higher accuracy. This can be due to the fact that all the possible scenarios are contained in the training set (since a single number can only be written in a few ways) and, as such, overfitting the data means that, for each case in the test set, the model will know with more precision which label it corresponds to. However, since the improvement in the performance is not that significative, it is still an acceptable choice to use Early Stopping since it drastically reduces the execution time.

Aside from the accuracy comparison, it is also useful to compare overall computational performance between the two networks. The CNN scenario shown required 25 epochs to run. Taking the patience into account, this translates to about 10 effective iterations, which is slightly more than the MLP (which only took 7 effective iterations). Moreover, each epoch takes around 10x more time to run in the CNN, being ≈ 300 micro seconds to complete an epoch in the CNN and ≈ 30 in the MLP, which is a consequence of the CNN being a deeper network. The MLP does have slightly more parameters, 59850 in total, compared to the 56714 in the CNN. This means that with less parameters, even though it is a deeper network, the CNN achieves a better performance (accuracy-wise). Still, the parameter amount of both models is similar. Even though the MLP consists of fully connected layers (which naturally yield more parameters), the fact that the CNN is deeper makes up for the reduction in the number of connections (and consequently, parameters) in each layer.

Overall, this means that the CNN is better at performing the image recognition task (due to the fact that it is purposely designed to identify different features in data, which is the best technique in image recognition: identifying edges and shapes in an image from the usage of different filters (kernels), for example), but it is a trade-off compared to the MLP, which does a slightly worse job but in a shorter time frame.

References

[1] *Neural Networks, Lecture Slides, Jorge S. Marques, IST*

[2] *Loss Functions, Cross-Entropy*: `https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html`

[3] *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning*: `https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/`

[4] *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*: `https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53`