1 2 9 0 FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE D COIMBRA

OCR- Optical Character Recognition

Project developed by:

Elena Capurro, nº, 2023185835

João Pinto, nº 2020220907

Introduction

This report explores the task of recognizing handwritten Arabic numbers, specifically the digits 0 to 9. We will be using neural network models to tackle this problem and automate digit recognition.

To address digit recognition, we'll compare two neural network architectures:

i) **Filter + Classifier**: This approach combines filtering layers and a classifier to capture detailed features in the handwritten digits. Both being a one-layer NN.

ii) **Classifier Only**: This architecture focuses solely on digits classification, aiming for simpler, wider and more efficient recognition process. We will explore NN with one-layer, two-layer and compare results with a special kind patternnet.

User interface

In order to make a classification of one or more digits, the user needs to run the createDesignApp.m function. This will open the pop-up shown in figure. In this simple interface, the classifier chosen to be used to recognize the digits needs to be selected from a dropdown menu that contains the three option that well be further explored in the next paragraphs of this report. After selecting the classifier type, the blue button needs to be pressed in order to open the grid to actually draw the digits. After drawing the digits, they can be classified by clicking the middle button of the mouse (in alternative shift+left). Automatically a grid with the recognized numbers at the place they were drawned will open.

To enable the classification of one or more handwritten digits, users are provided with an intuitive user interface implemented through the `createDesignApp.m` function. This user interface simplifies the digit recognition process, making it accessible and user-friendly.

Upon launching the `createDesignApp.m` function, a user-friendly pop-up window opens, as shown in figure 1. The primary components of this interface include a dropdown menu and a button.

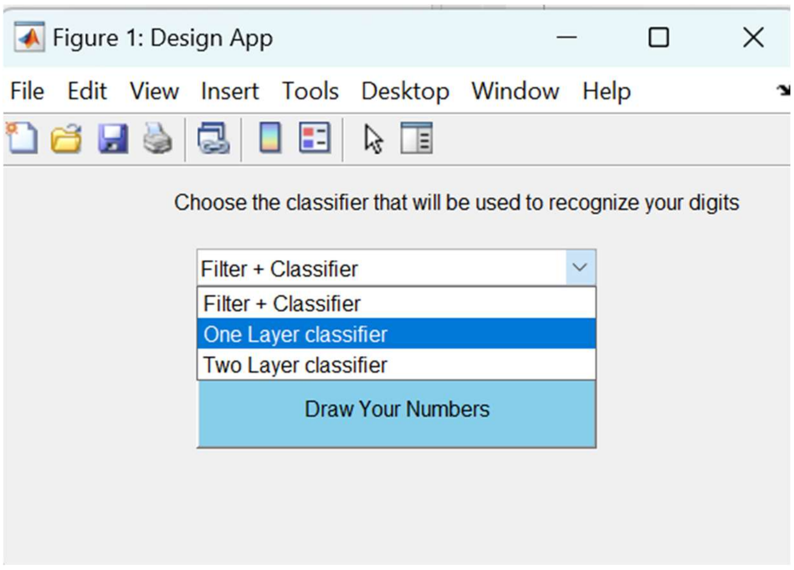
Figure 1: Design App

Choose the classifier that will be used to recognize your digits

The first step in the digit recognition process is selecting the classifier type. This selection is made from a dropdown menu that offers users three distinct options: “filter+classifier”, “one layer classifier” and “two layers classifier”. Details about the three different classifiers can be found in the next paragraphs of this report.

After selecting the desired classifier type from the dropdown menu, users can proceed by clicking the blue "Draw Your Numbers" button. This action opens an interactive grid, enabling users to hand-draw one or more digits directly onto the interface.

Upon drawing the digits, users can trigger the classification process by clicking the middle button of the mouse or, alternatively, using the shift+left mouse button combination. This action initiates the classification procedure.

Once the classification process is complete, the application immediately provides users with visual feedback. It displays a grid containing the recognized numbers placed in their respective positions as they were drawn.

Data set

To start our project, both of us built a data set of 500 digits ranging from 1 to 0 to match the Perfect Arial digits. In the beginning, we would only use each data set separately in each computer, but we noticed that the results did not match, mostly because we all have different handwriting, and the neural networks would be trained with different characteristics.

We decided to combine both data sets into a big one with 1000 different digits, this improved the performance of the classification system by, on average, 25%. Since, in this way, we have a lot more variety of characteristics which in turn, helps the classifiers to operate more successfully with more accuracy.

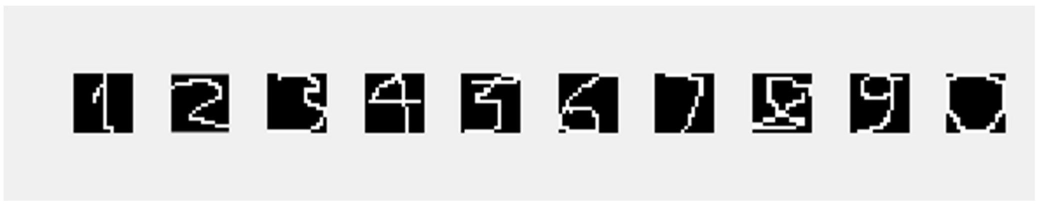

Fig 1. Some of João’s Digits

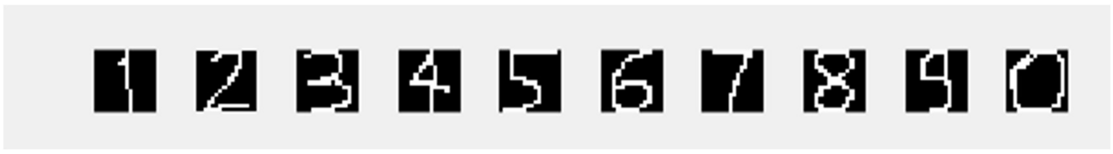

Fig 2. Some of Elena’s Digits

Neural networks architecture

In this section, we will discuss the two neural network architectures that we are going to study as part of this project. These architectures are essential for the digit recognition task.

Filter + Classifier

With our filter, we decided to use associative memory because, after some research and testing, we found out it can “correct” imperfect input data as opposed to binary perceptron. Although binary perceptron is more suitable for tasks where the input data is relatively clean, we thought that in this case where we need to write ourselves the digits, not a computer, the data would not be as clean as we would like, hence choosing associative memory.

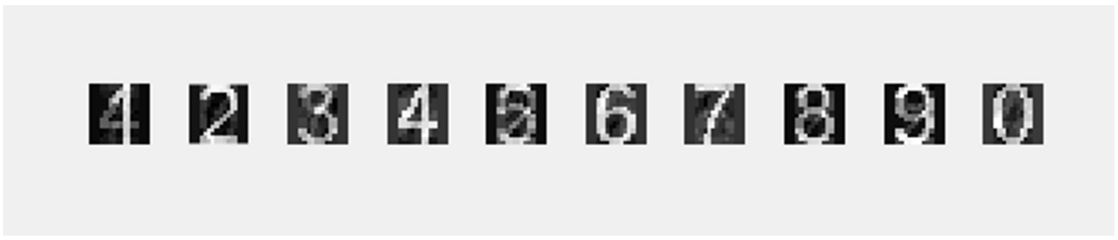

Fig 3. First 10 digits of P_f_both (filter applied to combined Data Set) to be used in the training of the classifier.

The second neural network module is the classifier, which has a single layer. It receives as input the output of the associative memory filter. We will be training this classifier with the matrix *P_f_both*, which is the matrix of the combined Data Set filtered with associative memory. We tried all combinations of given activation functions and training functions and conclude that using *logsig* as the activation function and *trainscg* as training function gives the best performance and accuracy. Also, we configured the output processing to ensure the output falls within a specific desired range, which improved the accuracy by 5%. After testing the NN 20 times with a testing set created from scratch we got on average, 89% accuracy.

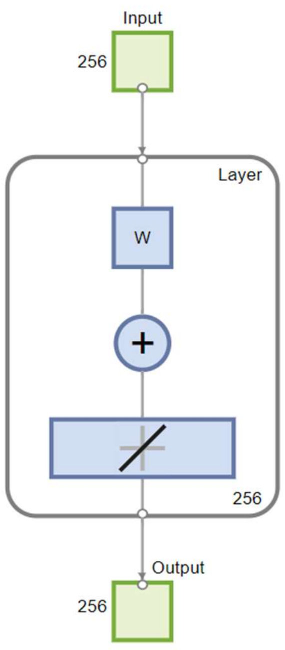

Fig 4. Associative memory filter

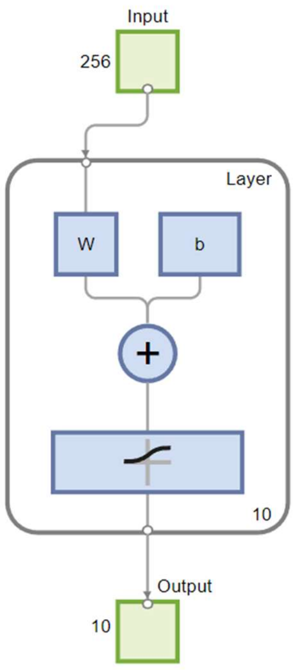

Fig 5. One-Layer Classifier

Classifier

Classifier with one layer

This classifier is the same as the last one, only difference is that it is trained with P_both, which is the matrix with 1000 digits without filtering (Fig 5).

The accuracy measured with the same test set as the last one was on average 90% with standard deviation of 0.04. This increase is likely due to the fact that the filter can lead to overfitting which becomes too specialized to the trained data and performs less accurate on unseen data.

Classifier with two layers

With two layers at our disposal, we were able to use as activation function on the first layer the logsig function and in the second layer we used the softmax function. The softmax function is fundamental to NN in multi-class classification scenarios, because it provides interpretable class probabilities, enabling effective training.

For training, the function used was “trainscg”, which is fast and gives good results in terms of performance and accuracy when testing.

In the hidden layer, we had to evaluate the performance according to the numbers of neurons on that layer. We started with 15 neurons all the way to 150 neurons, and only noticed changes in the performance when we got to 100 neurons, which makes sense since the training set has 1000 digits, if the classifier has 10 neuros at the output layer and 100 neurons at the hidden layer, we would be starting to get overfitting problems.

With that being said, the accuracy of the classifier with two layers varied from 94 to 97% on average between 15 and 150 neurons. When we configure the layer to only have 10 neuros the accuracy also drops to 90% on average.

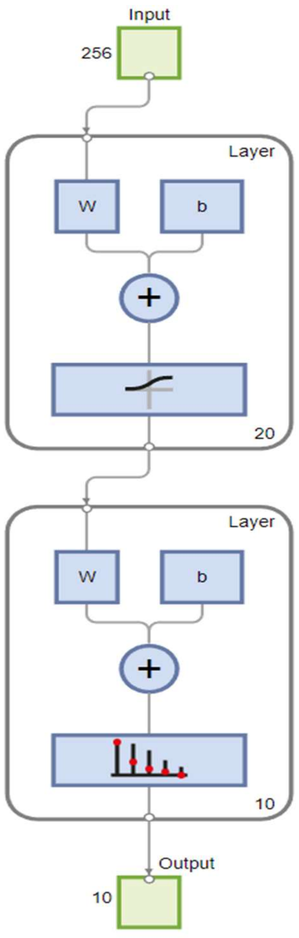

Fig 5. Two-Layer Classifier

Patternnet

Patternnet is a specialized two-layered neural network architecture for pattern recognition. The accuracy of the patternnet with 20 neurons in the hidden layer was 93% on average which almost matches with the accuracy of the previous NN.

Comparison

The architecture with the best results is the two-layer neural network and patternnet. With around 94 to 97% accuracy with our testing set, the two-layer classifier sits on top according to performance.

We couldn't have any good results using hardlim and purelin functions, so using logsig as the activation function is the way to go as well as the training function, the one we had the best results was "trainscg".

Results

The classification system is able to achieve the main objective of recognizing handwritten digits in most of the cases. We were expecting to get a different outcome in terms of performance across the two architectures. At first, we thought the classifier with only one layer would not be able to guess that many digits correctly. The expectation was that the difference in accuracy between that classifier and the two-layered and filtered would be in the dozens, not a mere 5 %.

In terms of robustness, some deviation from the training set used to train the classifiers can lead to misinterpretations of the system. Even so, it is able to correctly classify a high percentage, as we can see in the picture below:

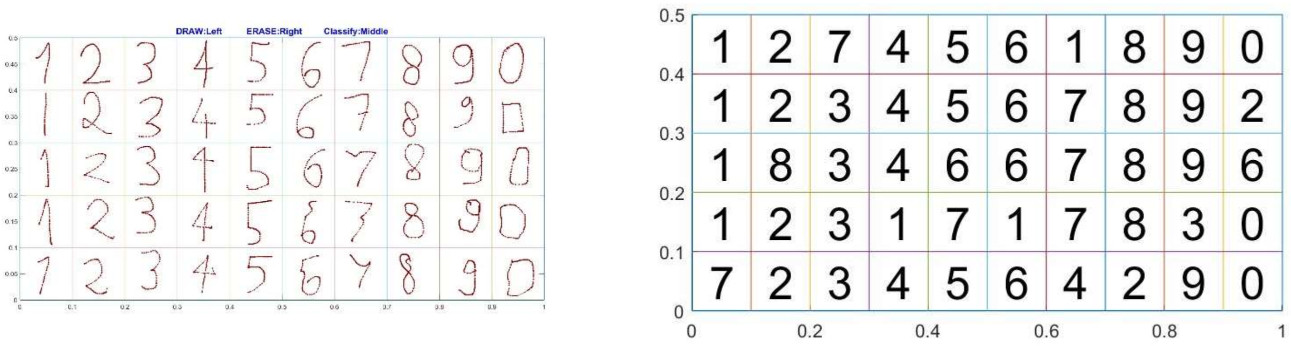

Fig 6. Test using one-layer classifier (no filter)

In the example shown in the figure, the digits were drawn in different ways as the arial font. The one-layer classifier with no filter was still able to recognize a high percentage of digits. It can therefore be stated that the classifier is robust.

Conclusion

In conclusion, the project's objective was satisfactorily achieved, but with some unexpected outcomes. The two-layer neural network architecture consistently achieved an accuracy between 94% and 97% with the testing set, surpassing other architectural choices. The utilization of “logsig” as the activation function and “trainscg” as the training function proved to be effective in optimizing the classifiers' performance.

However, it's worth noting that the filter and classifier approach, which incorporated associative memory as the filter, was expected to deliver better performance. The performance was, in fact, slightly lower than anticipated, but the one-layer classifier demonstrated its robustness by recognizing a significant percentage of digits with a high level of accuracy (fig 6).

To make the system more robust in the future, the training set should be more expandable. We should ask for people to crowdfund writing some digits to help improve the generalization capacity. This project really underscores the value of combining diverse datasets to get amazing results.