Second project to subject Computational Intelligence

Greek text recognition by Neural Network

January 8, 2018

Authors: Ladislav Šulák, xsulak04@stud.fit.vutbr.cz
Krisztián Benko, kristianbnk@gmail.com

Contents

1 Introduction **1**

2 Analysis **2**

2.1 Research work . . . 2

2.2 Dataset . . . 2

2.3 Decomposition of the project . . . 3

3 Design **3**

3.1 Neural networks in image recognition . . . 3

3.2 Neuroph Studio . . . 5

3.3 Java desktop GUI application . . . 5

4 Implementation **7**

4.1 Neural network creation . . . 7

4.2 Training of neural network . . . 8

4.3 Testing of neural network . . . 8

4.4 Implementation of application . . . 9

5 Experiments **10**

6 Results **11**

6.1 Improvements during our experiments . . . 11

6.2 Notes and future improvements . . . 11

7 Conclusion **12**

1 Introduction

This documentation describes a design and a development of Greek text recognition by neural networks. Every aspect of the development is described - from analyzing of the issue, choosing the solution and dataset, design of application, implementation of the final application to results and proposal improvements. The aim of this project was to create an application, which could help foreign people, which don't know Greek language. The aim of this particular project wasn't to done the whole application for cell phones with Android operating system, only the neural network part was supposed to be done, because of the lack of the time.

The report is conceived as follows: in the section 2 we describe the main points of our assignment such as the motivation, dataset and some details about the this project and it's steps. In the next section 3 it is described how we approached to this problem and there are also details about the algorithm we created. In the section 4 we describe each particular part of the application in more details. Section 5 deals with testing scenarios on a dataset we chose. Section 6 we analyze obtained results, conclusions and possible future improvements. The last section 7 summaries the observations we found.

2 Analysis

The main requirement of this project was to implement an application, which involves neural network and also to choose a dataset for the input of this application. So first we had to do a research to find out what possible applications are, where it makes sense to implement neural network. After we found the application, we decided what will be the input and how we will get the dataset. The whole project can be decomposed to more separated tasks we were dealing with.

2.1 Research work

At first, we were looking for suitable applications with the usage of some kind of neural network. In the best case we wouldn't have to implement the application, but just use it, create the dataset and work on testing and results. Our choice was an application for recognition Greek text, because we didn't find any application, which could satisfy our ideas. To be more particular, we chose an Android application, which will be able to capture the text by camera and translate it to user's language in final version. However, we supposed to do just the initial steps, in which we get the dataset of handwritten and typed letters of Greek language, create and train the neural network. Next plans were that after the neural network will be prepared, we will create a desktop application in Java, for preprocessing of pictures and testing. According to next articles, the best approach is to identify and divide the whole text to lines and then words or also to letters, which are going to be recognized separately. We were inspirited by following papers and articles: [1], [2], [3]. We found some similar applications, but we were not satisfied with their function. So our choice was, that we are going to develop the application from scratch. We found a lot of recommendations to use Neuroph Neural Network framework[1] for image recognition by neural network.

2.2 Dataset

The next part of the project's assignment was to find out the suitable data for our application. We have got 2 approaches: either we will create a big dataset of Greek words and recognize the text by words, or we will get some handwritten letters and the typed letters we could generate. It was impossible for us to find dataset with Greek words. So we chose the second approach. We found a huge dataset, but it contained only the half of the Greek alphabet[2]. We did a big research to find the letters, which left, but unsuccessfully. So we started to look for Greek font types, from which we could generate our dataset. We found 41 different Greek font types. From each font type we created 3 pictures, because we used image filters: dilation and thinning. By this way we had 123 pictures for each letter, what we considered as enough for the start, because we did not have so much time for experiments and the training of the neural network might took too much time with bigger dataset. We wrote our own generator of pictures in Python programming language, which was able to generate pictures of different size, with different letters and fonts. The filters were implemented in Java application, so they could be used to previously generated pictures and by this way we created more pictures.

[1]http://neuroph.sourceforge.net/
[2]https://martin-thoma.com/write-math/

2.3 Decomposition of the project

The whole project from our perspective was broken down into following parts:

1. Find suitable application
2. Define the data
3. Presentation of the previous two steps
4. Create a software design of the application
5. Implement the application
6. Experimentation
7. The results and its analysis
8. Presentation of the application and the results
9. Final report

3 Design

Now as we know what application we are going to create, we have to design how we will implement each part of the application. At the start we take a look at how we should done the neural network and then we determine how it could be improved. Then we are going to design the application, which will use neural network plugin.

3.1 Neural networks in image recognition

Image processing

Neural networks are one technique which can be used for image recognition. Every image can be represented as two-dimensional array, where every element of that array contains color information for one pixel. Each color can be represented as a combination of three basic color components: red, green and blue. So, to represent some image in a RGB system we can use three two-dimensional arrays, one for each color component, where every element corresponds to one image pixel. We can merge these three arrays into a single one-dimensional array so it contains all red values, then all green and at the end all blue values. Thats how we create flattenedRgbValues[] array. The dimension of this array is [imageHeight * imageWidth * 3] Now we can use this one-dimensional array as input for neural network, and to train neural network to recognize or classify them. Multi layer perceptrons are type of neural networks suitable for this tasks.

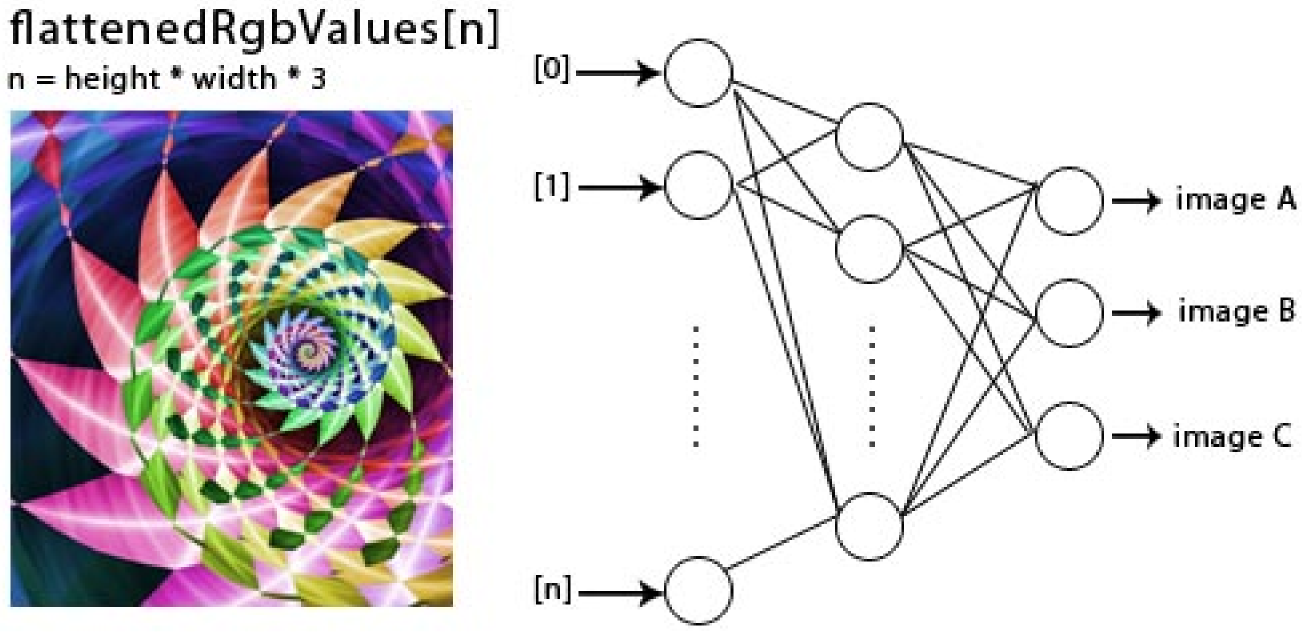


Figure 1: Feeding multi layer perceptron with color information from image. Each input neuron corresponds to one color component (RGB) of one image pixel at a specific location.

Each output neuron corresponds to one image or image class. So if network output is [1, 0, 0] that means that input is recognized as 'image A'. We can create training set for training neural network as set of pairs of input (flatten RGB arrays), and output vectors (where corresponding image neuron is 1). Images would be better to represent in binary black and white mode, pixel as [0, 1]. So it uses less number of input neurons. For some applications (like character recognition for example) binary black and white mode may be optimal solution. All provided images will be scaled to smaller size (width x height). Scaling images will make them smaller, and they will be easier and faster to learn. The image dimensions determine the size of input vector, and number of neurons in input layer. In next section we'll provide some details about the neural network and learning algorithm.

Neural network

In this section we are going to talk about some features of neural networks, which were important for this project. Neural networks are typically organized in layers. Layers are made up of a number of interconnected 'nodes' which contain an 'activation function'. Patterns are presented to the network via the 'input layer', which communicates to one or more 'hidden layers' where the actual processing is done via a system of weighted 'connections'. The hidden layers then link to an 'output layer' where the answer is output.

Hidden layers

Hidden layers are layers between input and output layer. The trick is to have the smallest possible number of layers and neurons which can successfully learn the training set. The smaller number of neurons - the faster learning, better generalization. Suitable number of hidden neurons also depends of the number of input and output neurons, and the best value can be figured out by experimenting.

Activation function
The important characteristic of the activation function is that it provides a smooth transition as input values change, i.e. a small change in input produces a small change in output. If the activation function is linear, then you can stack as many hidden layers in the neural network as you wish, and the final output is still a linear combination of the original input data. This linearity means that it cannot really grasp the complexity of non-linear problems like XOR logic or patterns separated by curves or circles.

Meanwhile, step function also has no useful derivative (its derivative is 0 everywhere or undefined at the 0 point on x-axis). It doesn't work for backpropagation. So we are going to choose Sigmoid, which is activation function that transform linear inputs to nonlinear outputs. Network can be trained by using Backpropagation learning algorithm.

3.2 Neuroph Studio

Neuroph is lightweight Java neural network framework to develop common neural network architectures. It contains well designed, open source Java library with small number of basic classes which correspond to basic NN concepts. Also has nice GUI neural network editor to quickly create Java neural network components. Creating and training neural network for image recognition consists of the following steps:

1. Create Neuroph project
2. Create image recognition neural network
3. Train network
4. Test network
5. Save & deploy network
6. Import .nnet file as Neuroph plugin in Java application

3.3 Java desktop GUI application

This application will consist of 4 parts. In first part the user will be able to use image filters on the picture to prepare the picture for training or testing. In the next part the user will be able the use the filters on a file of pictures. In the third part the user will create the the neural network by using the training and testing set of pictures. And in the last part the user will test a specific picture on character recognition and the application will return as result the recognized character.

Picture filters

Binarization
It enhances the performance of document processing techniques like OCR and layout analysis. Image Binarization is the conversion of document image into bi-level document image. Image pixels are separated into dual collection of pixels, i.e. black and white. The main goal of image

binarization is the segmentation of document into foreground text and background.

Dilation

It is typically applied to binary images, but there are versions that work on grayscale images. The basic effect of the operator on a binary image is to gradually enlarge the boundaries of regions of foreground pixels (i.e. white pixels, typically). Thus areas of foreground pixels grow in size while holes within those regions become smaller.

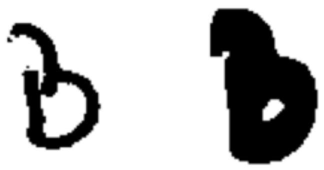


Figure 2: Dilation is shown in the picture.

Thinning

Thinning is a morphological operation that is used to remove selected foreground pixels from binary images, somewhat like erosion or opening. It can be used for several applications, but is particularly useful for skeletonization. In this mode it is commonly used to tidy up the output of edge detectors by reducing all lines to single pixel thickness. Thinning is normally only applied to binary images, and produces another binary image as output.

Figure 3: Thinning is shown in the picture.

Using this filters we will be able to create new pictures to enlarge the dataset and pre-process the picture for recognition.

4 Implementation

In this section we are going to describe the implementation of the application. Here will be described what prerequisites were needed, how we created the neural network, how we implemented each part of the application.

For implementation purposes we chose Java programming language in version 1.8. The whole project was implemented and tested under Windows 10 operating systems.

4.1 Neural network creation

The network was created in Neuroph Studio, according to chapter 3.2. For the transfer function we used Sigmoid. After a couple of experiments we created 4 layers neural network, from which 2 layers are hidden with 86 neurons each. We started with much different combination like 1 hidden layer with 12 neurons, but the performance of training was much worse. The configuration of hidden layers might be improved with a couple of experiments. The final neural network has 24 outputs and 900 inputs, what is because of the pictures of size 30x30px. This neural network was created in Neuroph Studio v2.94. However, the plugin for Java contained a bug, where the plugin's version instead of 2.9 was 2.8, what caused problems during deserialization. That's why we finally created the network in Neuroph Studio just for visualization and some experiments and results.

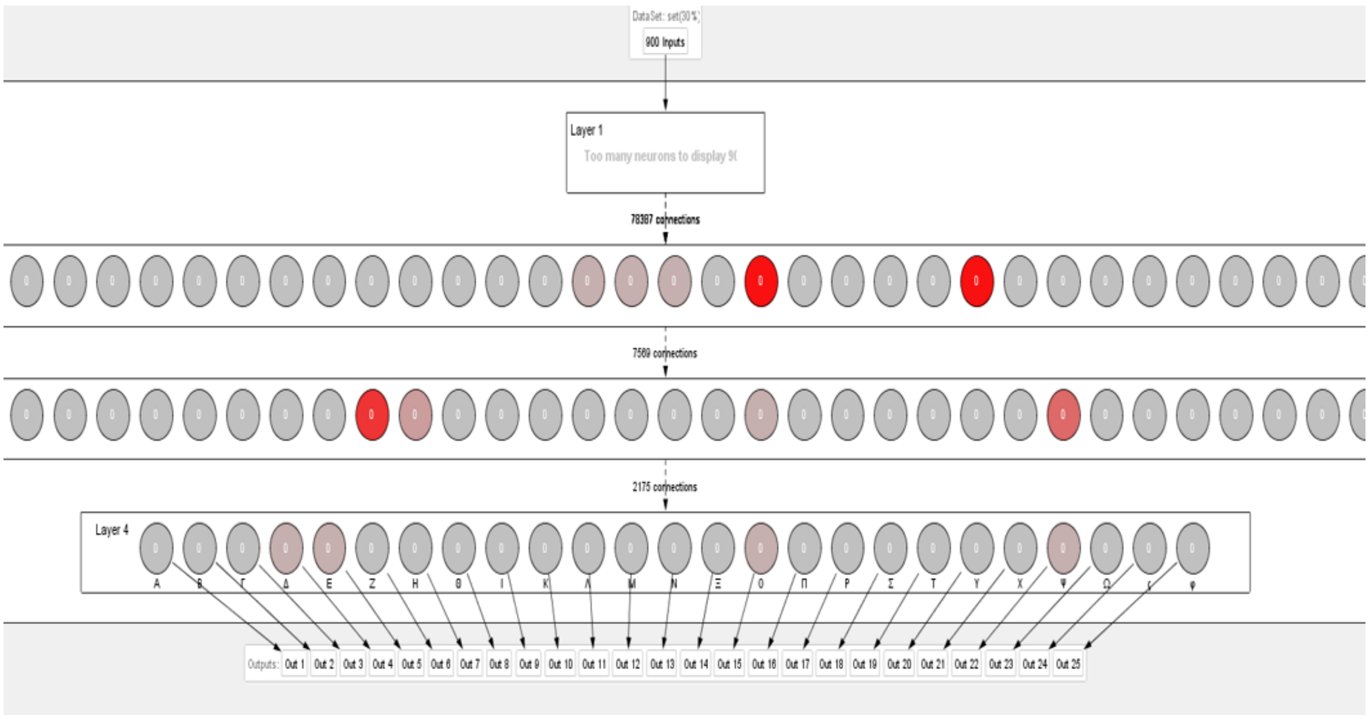

DataSet: set(30%)
900 Inputs
Layer 1
Too many neurons to display 9(
78387 connections
7569 connections
2175 connections
Layer 4
Α Β Γ Δ Ε Ζ Η Θ Ι Κ Λ Μ Ν Ξ Ο Π Ρ Σ Τ Υ Χ Ψ Ω ς φ
Outputs:
Out 1 Out 2 Out 3 Out 4 Out 5 Out 6 Out 7 Out 8 Out 9 Out 10 Out 11 Out 12 Out 13 Out 14 Out 15 Out 16 Out 17 Out 18 Out 19 Out 20 Out 21 Out 22 Out 23 Out 24 Out 25

Figure 4: Visualization of final neural network.

We have also tried to create the neural network in Weka[3]. But there was a problem of the lack of the tutorials, how to use this tool for image recognition or especially for character recognition. After a while we found out that we should probably implement our filter functions for pictures to be able to use this tool and then implement the neural network in Java as well.

[3]https://www.cs.waikato.ac.nz/ml/weka/

Finally we found a usage of Neuroph Studio or especially a usage of neuroph neural network. We found an implementation in Java, where the creation of neural network was solved using Neuroph plugin but version 2.92. After a couple of modifications we were able to use it in our application.

4.2 Training of neural network

For training of the neural network we divided the dataset, what was 2975 pictures, to 2 parts in ration 80:20. So 80% of the pictures were used for training and the rest for testing. 500 epochs of training took approximately 2 hours. The stopping criteria was maximal error 0.01. And we set the Learning rate parameter to 0.2 and Momentum parameter to 0.7.

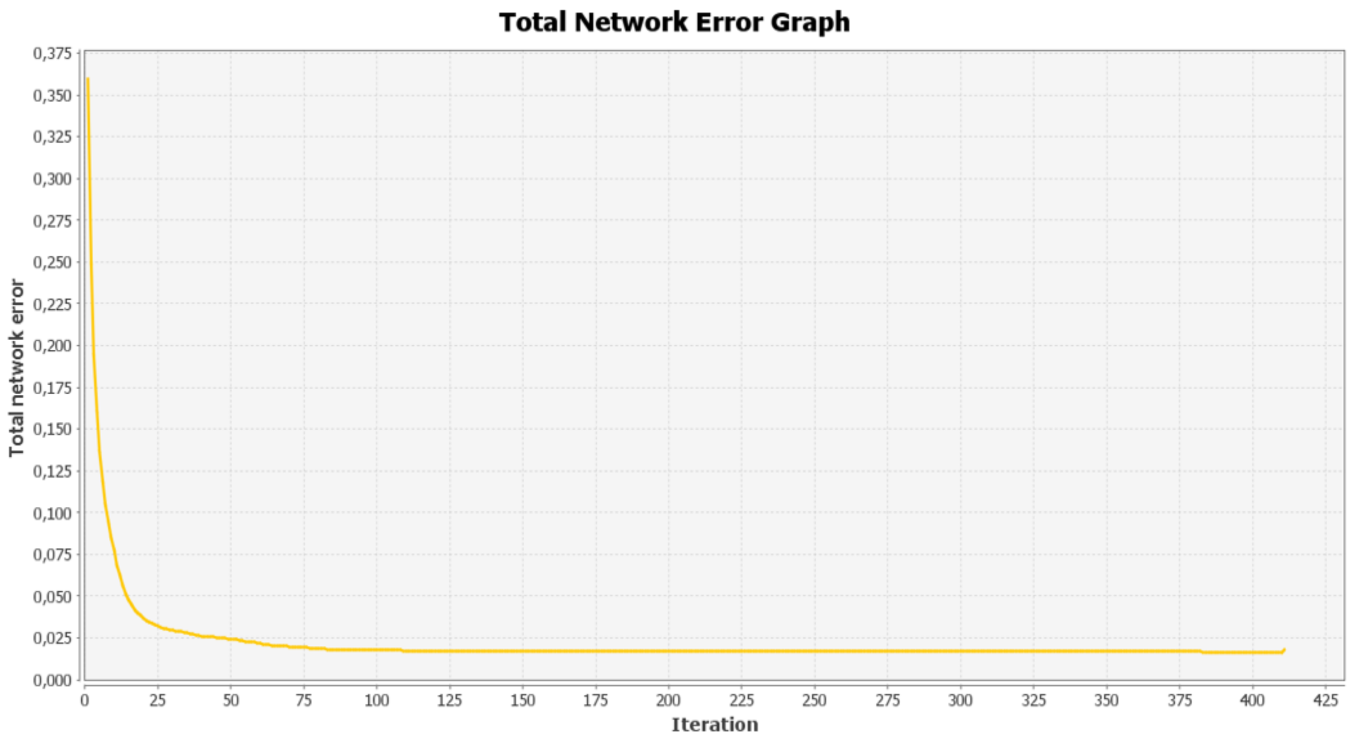


Figure 5: Visualization of Total network error graph.

As you can see in the picture, the training was stopped after more than 400 epochs, because it took too much time and the error rate was not extremely changing.

4.3 Testing of neural network

For testing we used the rest 20% pictures from the dataset and we ran testing. From testing we have got some statistics:

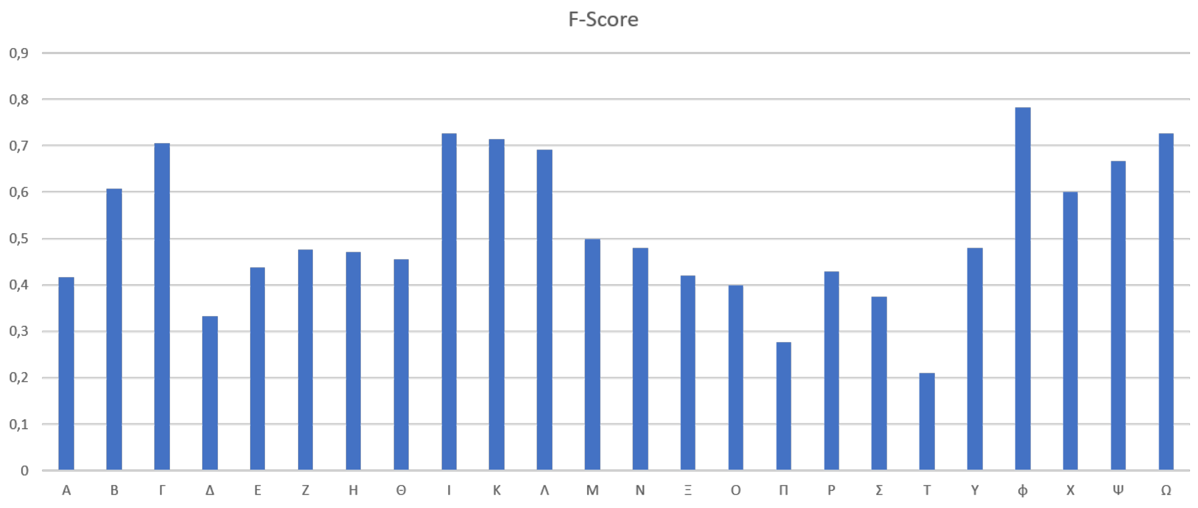


Figure 6: F-Score of each letter from alphabet.

4.4 Implementation of application

As it was mentioned in chapter 4.1, in application Neuroph plugin v2.92 was used. So, the created neural network in Neuroph Studio couldn't be used. However, it was not a big problem, because we created a creating of neural network in the application and as the dataset was done, we just used it and in the final we got almost the same results. The difference was in training of the neural network, because after every epoch we ran testing on the neural network, so at the end we stopped training when the success rate was not increasing anymore. This neural network was saved and used in recognition.

The picture filters were implemented according to chapter 3.1, we are not going to go into deeper implementation details, because this documentation shouldn't be about image filters implementation. These filters are well known filters, so there were already implemented and used a lot of times. Some examples you can find on this website: Image filters[4].

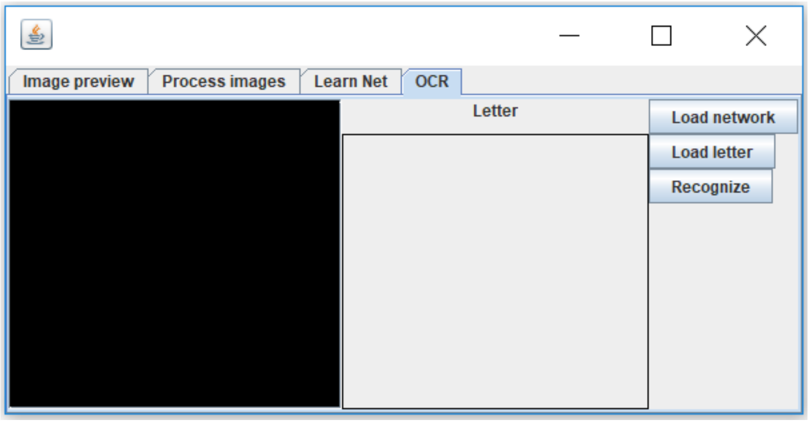
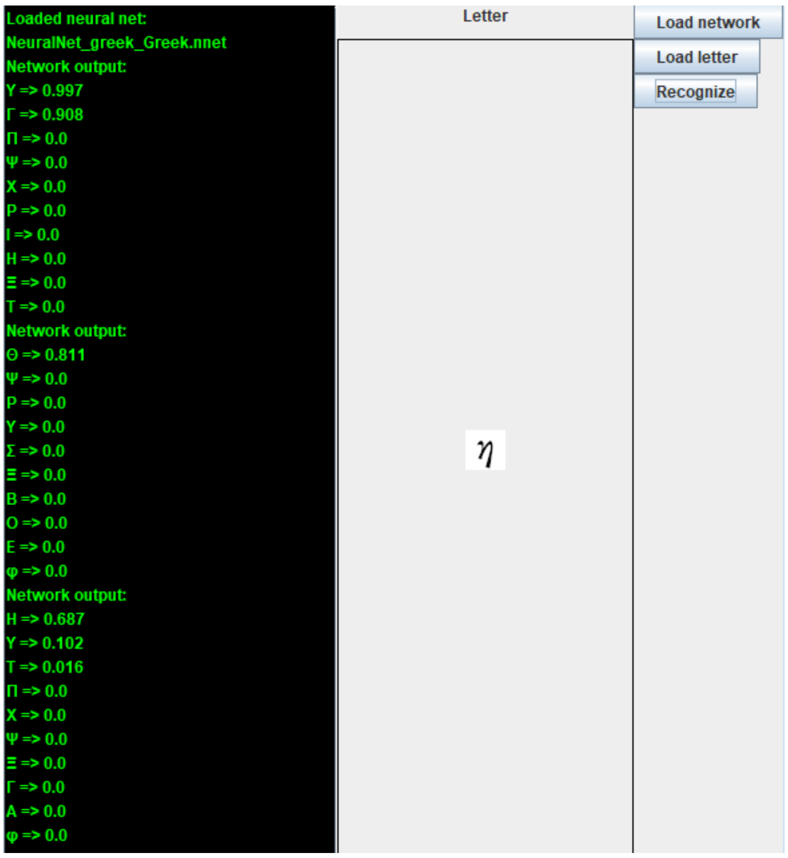


Figure 7: Initial application.

[4]https://homepages.inf.ed.ac.uk/rbf/HIPR2/

5 Experiments

For the experiments we used a couple of letters captured with camera of the mobile phone, which was a Samsung S5. We used our image filters on these pictures to pre-process them and prepare for recognition. In our dataset we created for every font type of each letter 3 pictures with different thicknesses, so using gray scale, binarization and rescaling seemed to be enough for pre-processing.



```
Loaded neural net:
NeuralNet_greek_Greek.nnet
Network output:
Υ => 0.997
Γ => 0.908
Π => 0.0
Ψ => 0.0
Χ => 0.0
Ρ => 0.0
Ι => 0.0
Η => 0.0
Ξ => 0.0
Τ => 0.0
Network output:
Θ => 0.811
Ψ => 0.0
Ρ => 0.0
Υ => 0.0
Σ => 0.0
Ξ => 0.0
Β => 0.0
Ο => 0.0
Ε => 0.0
φ => 0.0
Network output:
Η => 0.687
Υ => 0.102
Τ => 0.016
Π => 0.0
Χ => 0.0
Ψ => 0.0
Ξ => 0.0
Γ => 0.0
Α => 0.0
φ => 0.0
```

Figure 8: Experiments in character recognition.

In the picture above we can see 3 experiments. First experiment was with letter γ. As we can see it was recognized as Υ, but γ had also a high probability. The next letter was Θ, in this can we were successful and letter was recognized correctly. As well as we recognized correctly the letter from the last experiment, what you can also see in the picture. It was letter η.

6 Results

Creation of neural network was longer process. We were experimenting a lot with the layers and dataset. Composition of layers was important for performance of the net's ability to learn. To decide what data use for training and testing, was important for final usage of the neural network during recognition process. Finally we came to point when we have some neural network, which is is able to recognize some letters but it is still not perfect. In this section you will also see, why it was useful to use neural network for recognition and also some suggestions for possible future improvements.

6.1 Improvements during our experiments

At the start after very first experimentations, when the results were worse than results in section 5, we started to improve our dataset. The improvements were mainly focused on the input pictures of neural network and their features. First we found out, that it matters if on which position in the picture is the letter placed. So we tried to place the letter in the center and fill the whole picture with this letter. We were doing this also in preparation process of the letter, which we wanted to recognize. Then we found out, that if in our dataset we have only letters with normal thickness, it will be difficult to recognize a letter with a big thickness or small. So we used dilation a thinning and it helped. What was very important - to use filters: binarization, rescaling, centering on picture, which were being recognized.

6.2 Notes and future improvements

As you can see in experiments, there was messed up the γ with Υ. That was caused probably because they look similar and also that we have a lot of outputs. So next improvement could be to use clustering, what means to create for example 4 groups of letters which look similar and then the neural network will have to determine the right output on smaller domain. We could also divide the pictures of letter to lowercase letters and uppercase letters. As next we could tune the parameters and architecture of neural network.

When it comes to application, we would like to create an Android application. We should be able to deal with a whole text. So some graphical operation should be done, where we divide the text to lines, words and letters. This is actually partially done, dividing the words into letters is not done yet. As we will recognize the word, it could occur that we won't recognize all the letters correctly, so auto-correction should be done according to recognized language.

7 Conclusion

The aim of this project was to find, design and implement a practical application with the usage of neural network. We chose a recognition of Greek text using neural network. Our original plan was to create an Android application as final version, what shouldn't be done in this project, because the time cap was too small. At the start we saw the biggest problem in dataset, because we were not sure how and when will we get all the letters of Greek language, even if we found in preparation process a lot of datasets on internet, but they were usually depreciated or not working. So, we decided to create our dataset using Greek alphabet, different types of fonts and image filters. We created a Python script, which generated the dataset for us. For creating the neural network we used Neuroph Studio, where we created an initial network and during experiments we tuned the parameters of the ANN. Unfortunately, we found a bug in Neuroph plugin, so we were not able to use the neural network from this studio and we had to find other approach. Finally the ANN is created in Java application, what represents the pilot version of the final Android application. Pre-processing of the input picture for recognition, creating and training of neural network and letter recognition is done in this application. Regarding to correctness of this solution, the letters were not recognized all the time correctly with the highest probability, but they always got some probability. By future work could be the recognition improved. First plan was to recognize the whole text, not just letters. The text should be divided to lines, words and letters. Some work is already done but it is not fully done yet. After the words will be divided to letters, we will use auto-correction to create words from letters, so if some letter's recognition went bad, we would be able to fix it. Solving these task we should be able to use this functionality in Android application.

References

[1] Dan Saadati Batuhan Balci and Dan Shiferaw. Handwritten text recognition using deep learning.

[2] A. Z. TALIB D. N. HAKRO and G. N. MOJAI. Multilingual text image database for ocr. 12 2014.

[3] Amit Sahu Harshal Bobade. Character recognition technique using neural network, 03-04 2013.