Gliwice, dn. 10 IX 2017 r.



**Biology Inspired Artificial Intelligence** Project Report

**Topic:**

Digits recognition

Authors:

1. Gurbiel Marcin
2. Kowalski Bartosz

# Project objectives

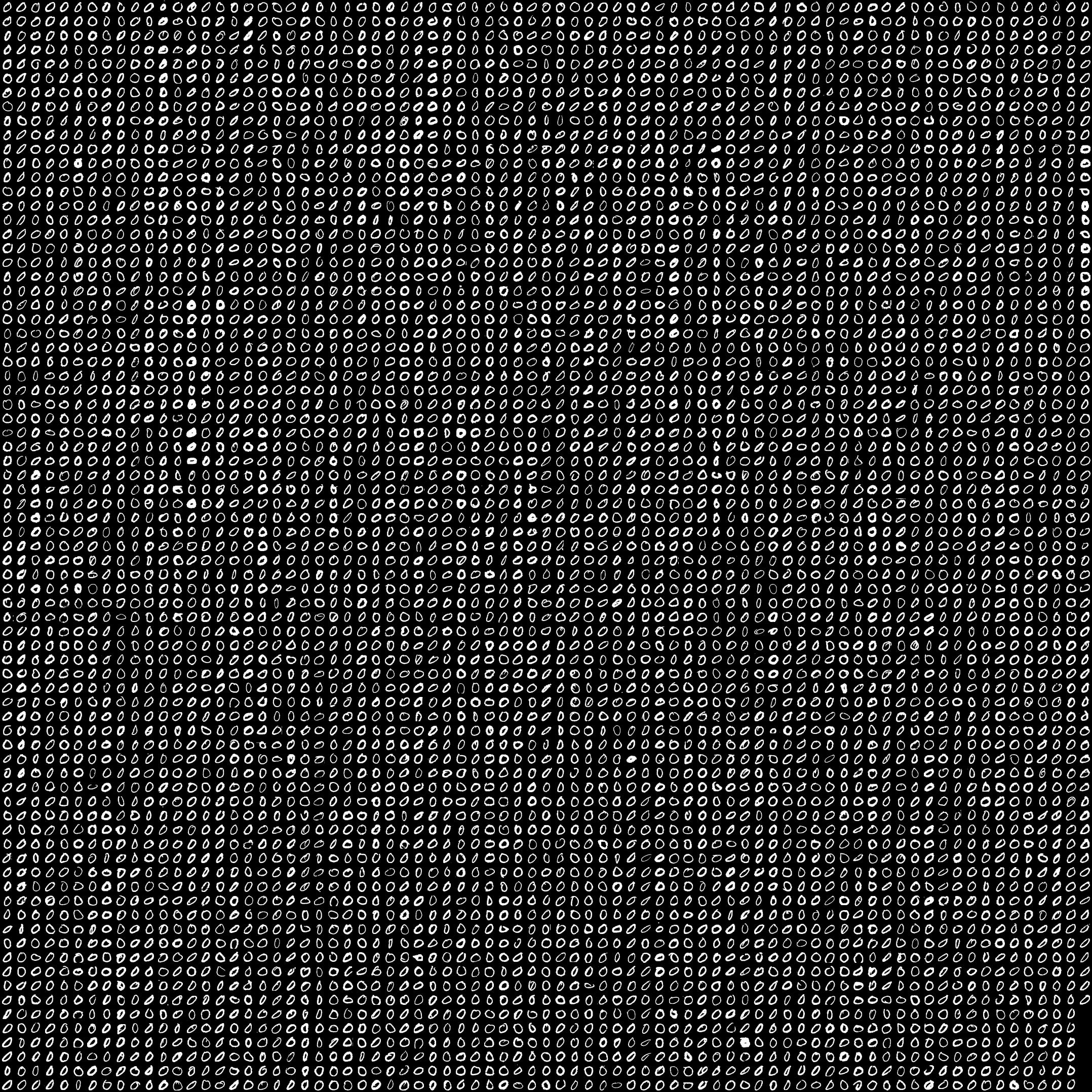
The goal of the project was developing an application capable of recognizing a defined set of manually written digits. Algorithm used in the application should be based on an artificial neural network.

# Input data

The input data is a set of manually written digits.

Input data for train and recognize is represented by collection of bits in TXT files. We create these files using a second application.

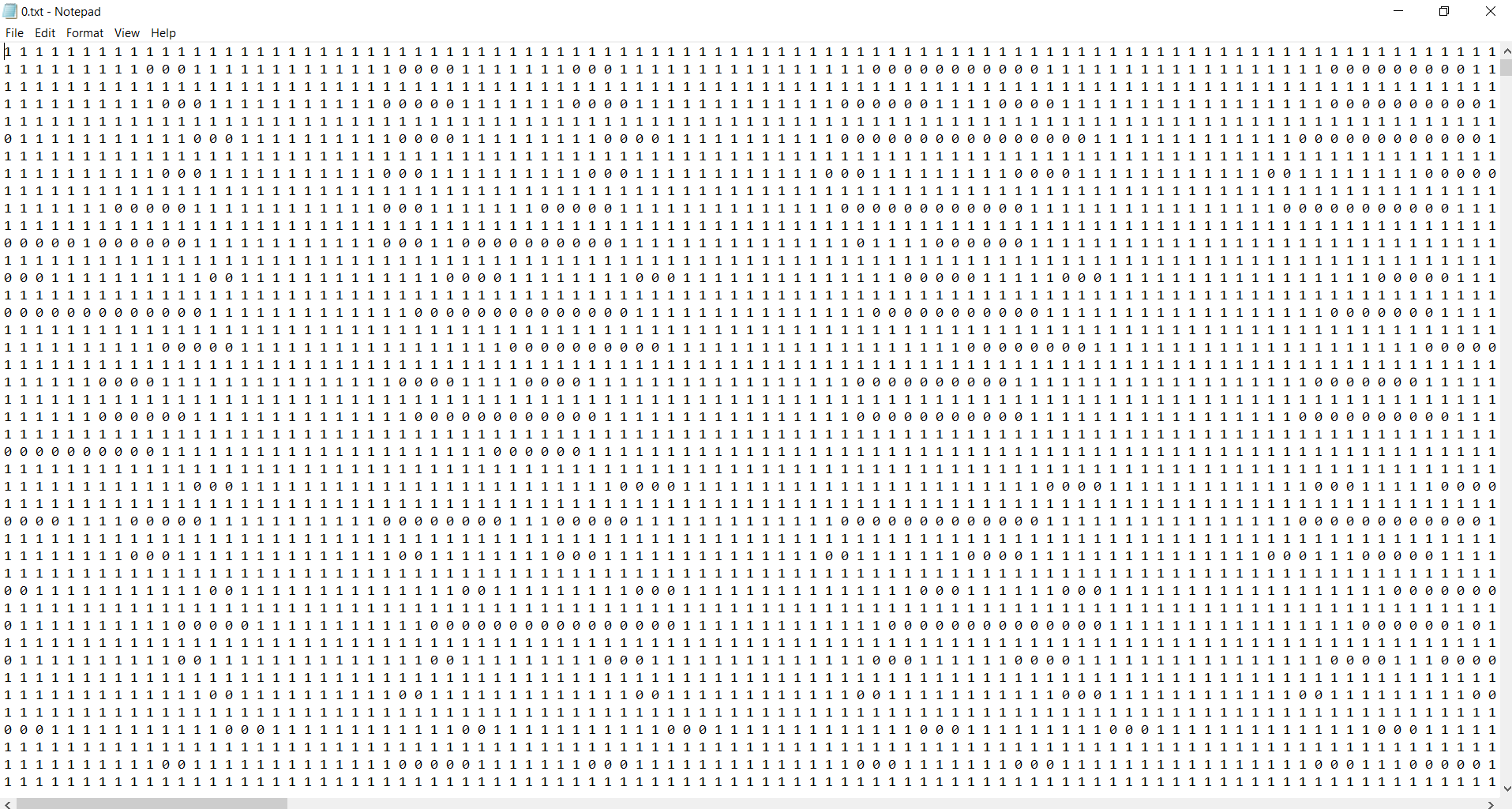
Below is presented an example of image for `0`



Each photo contains collections of handwritten numbers. The image’s extension is supposed to be JPG whereas required height of the one sample of digit has to be equal 28 pixels

With 10 images we get 10 TXT files containing the bit collections of each sample.

Below is presented an example of image for `0`



Each file line is one image sample, where 1 is equal to RGB (0.0,0) and 0 RGB (1,1,1). Pixel’s color is approximate to such values with that code:

1.0 - (pixel.R / 255.0 + pixel.G / 255.0 + pixel.B / 255.0) / 3.0) < 0.5 ? 0.0 : 1.0

The images were downloaded from http://www.cs.nyu.edu/~roweis/data.html

# Algorithm

## Network details

The neural network used in the application is the authors’ own implementation. The network consists of three layers:

* Input layer
* One hidden layer
* Output layer

It was not necessary to use more than a single hidden layer, it is however possible to create more layers using the developed solution.

The input layer consists of 28 \* 28 + 1 = 785 perceptrones (28 is a size of image and 1 is an additional neuron), each one indicating whether the pixel was white or black. Inputs are taken from an in-memory collection.

The count of perceptrones in the hidden layers can me modified by the user. There are available three rule-of-thumb calculating methods:

* Geometric mean:
* Half of inputs:
* Two thirds of inputs summed with outputs:

The count of perceptrones in the output layer is equal to the count of characters in the font’s input image (in any tests performed, all 86 characters were used, thus the output layer’s count is equal to 86).

The learning factor is set by the user from the range of 0.05 to 0.2. Initial weights of the perceptrones’ interconnections are generated randomly in runtime from scope [-0.1;0.1].

## Training and testing sets

Due to using 6 fonts, each character has 6 representations. To form an input data set, for each character one randomly chosen representation is discarded. The remaining representations become the training set. Discarded representations are stored and used for testing purposes.

## Training the network

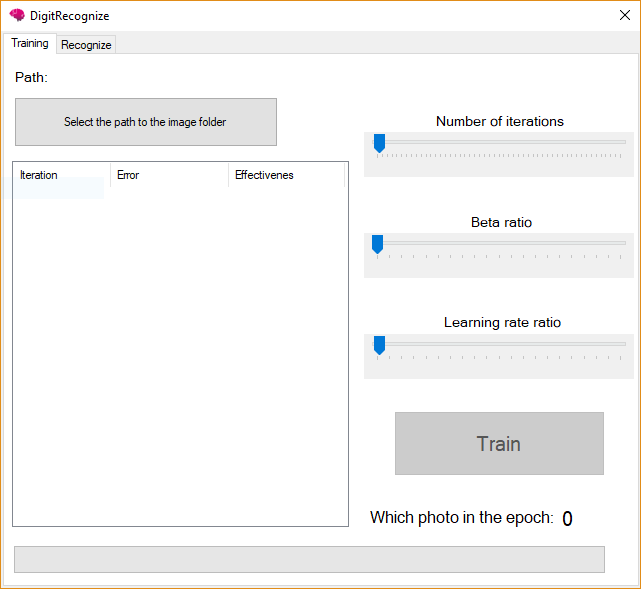
All training characters from all fonts are gathered into one collection. Before starting the training process, the collection is shuffled. The training process is executed for each element, and then the process is repeated. The number of cycles can be adjusted by user.

## Testing the network

After the training process, the neural network is submitted to tests. Each tested element is classified by the network. The result values are compared with expectated values. Decision error and normalized decision error are then calculated from formulas shown below:

# Application usage

After running the application, the following window should appear:



❶

❺

❹

❸

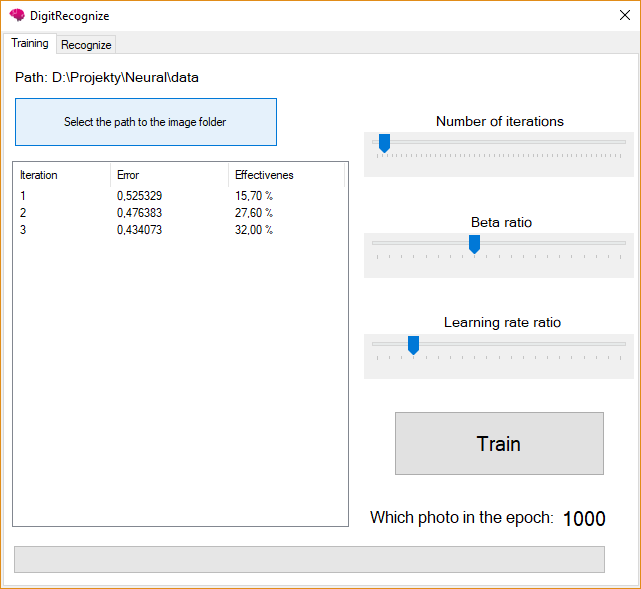
❷

❶

First of all the user should select the path to folder with images. To do this, he should click on the button marked as “1” in the picture. After click the user will see folders explorer, where he can find right folder with learning materials.

Nextly the user should set a teaching parameters like iterations count (marked as “2”), beta ratio(marked as “3”) and learning ratio(marked as “4”).

When all settings are set all you need is to click in to “Train” button, marked in the picture as “5”.

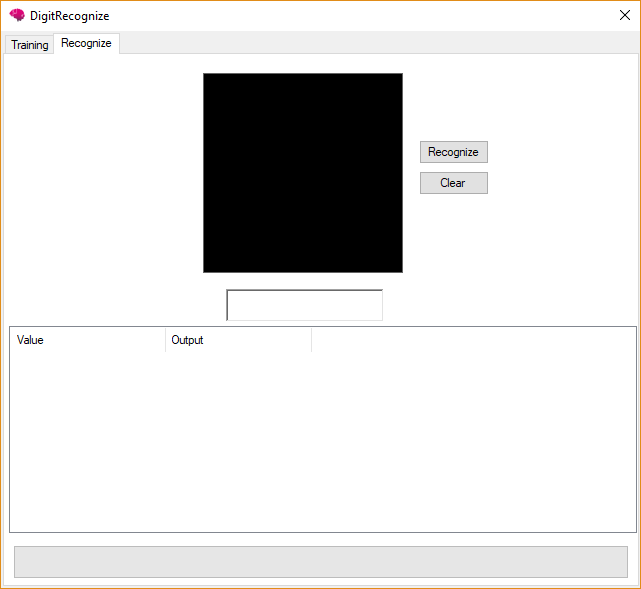


❷

❶

After the training process is completed, the user has possibility to check an error and effectiveness of executed training process in a table marked as “1’.

Now the user can try efficiency of the neural by himself. To do this he need to click in the second tab “Recognize” marked as “2”.



❺

❹

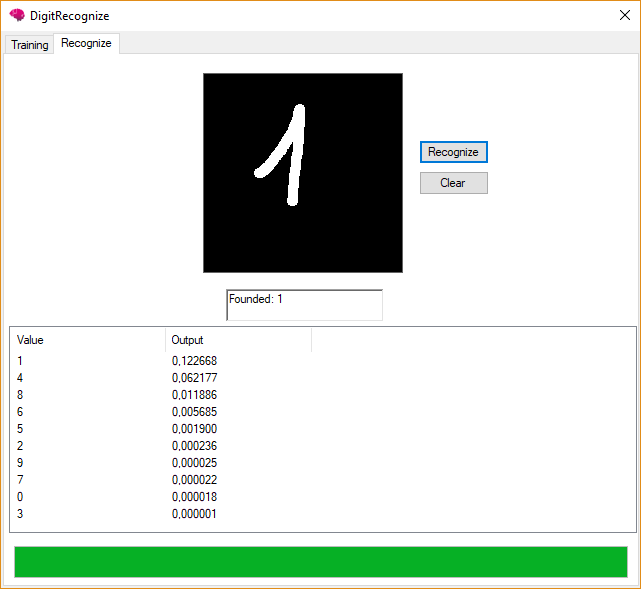
❸

❷

❶

A picture above presents second application tab “Recognize”. Here the user can write a digit in a black field marked as “1”. After that can try recognize a digit by clicking “Recognize” button marked as “2”, or he can clear the black field, by clicking “Clear” button, to try write a digit one more time.

After click “Recognize” button a recognized digit will appear in a text box marked as “4”. Every execute of recognizing is triggering logs printing to a table marked as “5”. This table shows values that the neural recognized with their corresponding weights.



# Testing

The purpose of the testing process was to estimate influence of input parameters, such as learning rate, training iterations, count of input photos and beta rate on the decision error and accuracy/efficiency of recognizing. In every test was only one variable.

Learning rate

For this case learning rate value is a variable, and others values are fixed.

Photos count = 5400

Beta rate = 1

Training iterations count = 20

Beta

For this case beta value is a variable, and others values are fixed.

Photos count = 5400

Learning rate = 0,15

Training iterations count = 20

Iterations count

For this case iterations count is a variable, and others values are fixed.

Photos count = 5400

Learning rate = 0,15

Beta = 1

Photos count

For this case photos count is a variable, and others values are fixed.

Iterations count = 20

Learning rate = 0,15

Beta = 1

As we have predicted – in all three test cases the best learning factor value oscillates around 0.1. Lower values cause requirement of training network with more cycles, whereas greater values speed up the process but also make the network forget the trained knowledge sooner.

# Conclusions

All tests proved that our self-developed network is capable of learning. After careful adjustments of parameters such as learning factor, training cycles count and number of hidden neurons, we achieved network with decision error equal to 3,5% (only 3 mistakes in 86 test character set). The learning factor value should oscillate around 0.1. If our goal is to absolutely minimise the error value, we can select hidden neurons count calculated with formula:

Nevertheless, the time required to train the network properly increases gradually. Results achieved with the *half inputs* rule were not significantly worse, and the time required to train the network was acceptable.

It is to be noted that the fonts we selected to train the network are all plain serif fonts. One of them was developed by one of the authors and it appeared that it suffered issues regarding distinguishing the figure 1 and the lowercase letter l. Different results could have been obtained if the network was tested with a mix of serif and sans serif fonts, handwritten fonts or more complex decorative fonts.