Gliwice, dn. 10 IX 2017 r.



**Biology Inspired Artificial Intelligence** Project Report

**Topic:**

Digits recognition

Authors:

1. Gurbiel Marcin
2. Kowalski Bartosz

# Project objectives

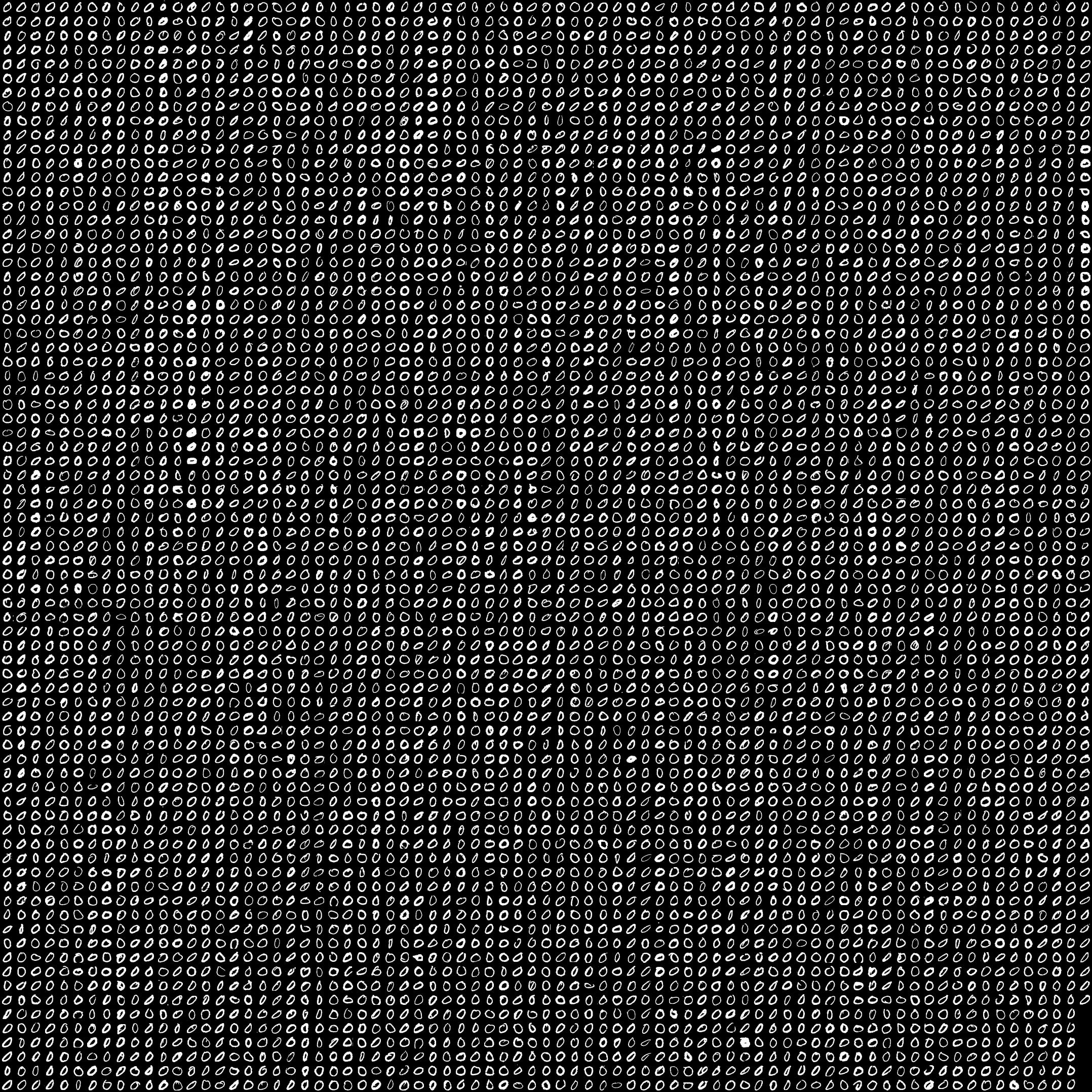
The goal of the project was developing an application capable of recognizing a defined set of manually written digits. Algorithm used in the application should be based on an artificial neural network.

# 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.

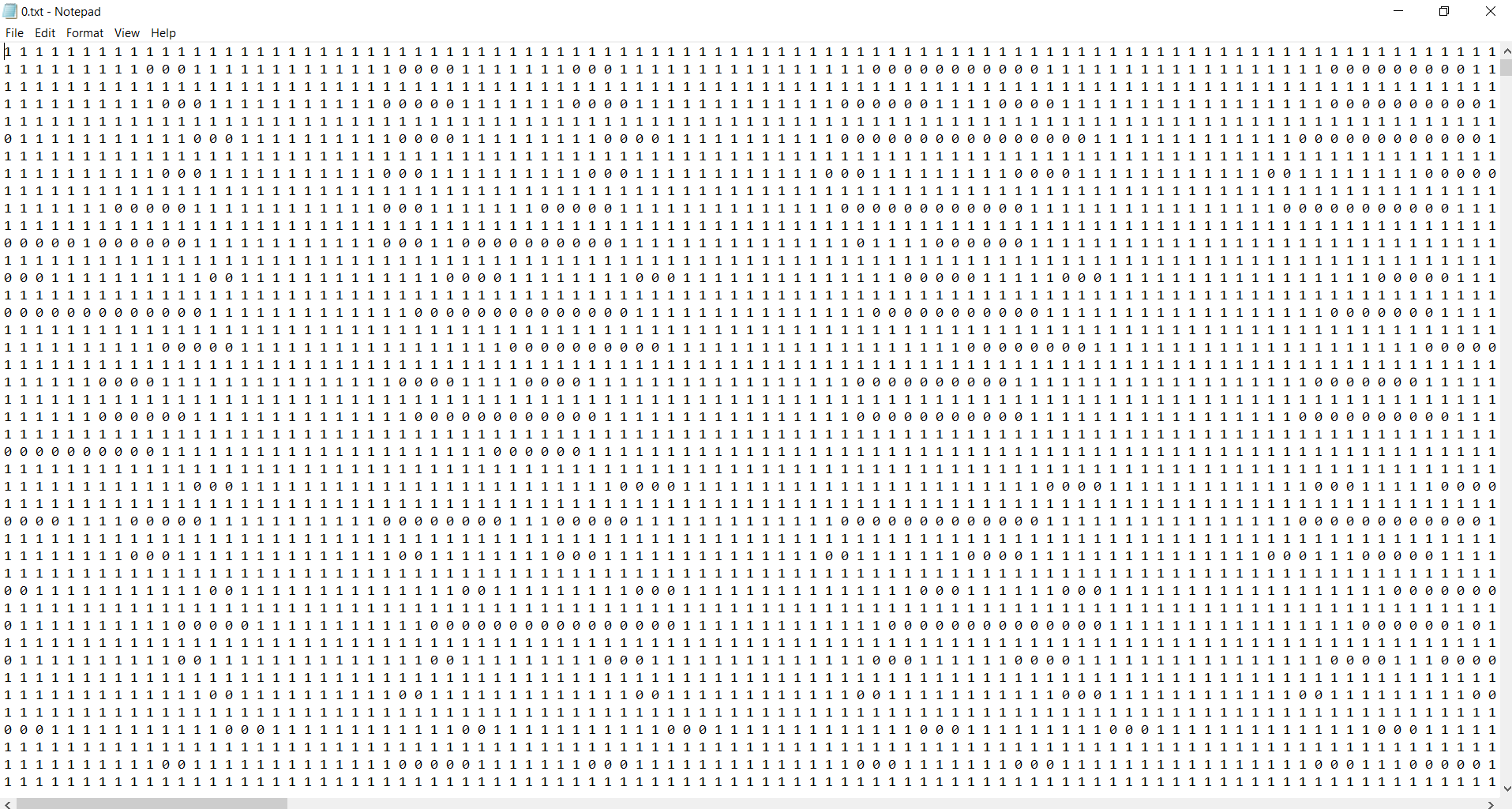
Below is presented an example of image for `0`



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’t be modified by the user. For this, we used one calculating method:

* Half of inputs:

The count of perceptrones in the output layer is equal to the count of digits(from 0 to 9, the output layer’s count is equal to 10).

User can set:

1. Learning ratio from the range of 0.01 to 1
2. Beta ratio from the range of 0.1 to 2
3. Maximum number of iterations from the range of 1 to 100
4. Maximum number of photos of each digit from the range of 1 to 10000 (if maximum number of photos is bigger than number of examples in files, then this variable is set to number of examples).

Initial weights of the perceptrones interconnections are generated randomly in runtime from scope [-0.1;0.1].

## Training and testing sets

Due to using above images, each digit has about 5500 representation. To form an input data set, a suitable number of examples from the file is read. The remaining representations become the training and testing set.

## Training the network

All training digits from all examples 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.

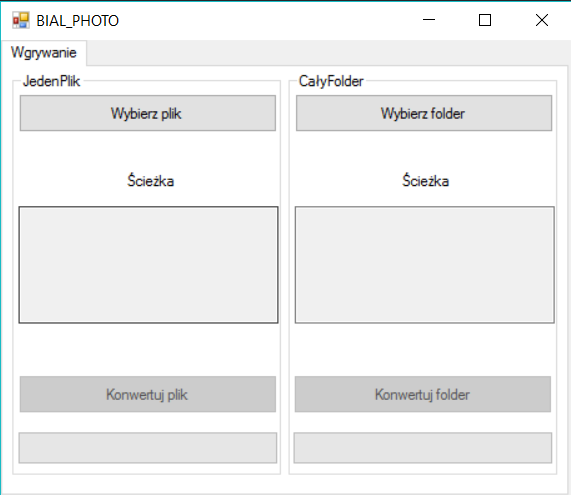
## Testing the network

After each iteration (epoch), the neural network is submitted to tests. Each tested element is classified by the network. The result values are compared with expectated values. Effectiveness of network is calculated from formula shown below:

# Application usage

## ImageConverter

After running the application, the following window should appear:



❹

❹

❸

❸

❷

❷

❶

❶

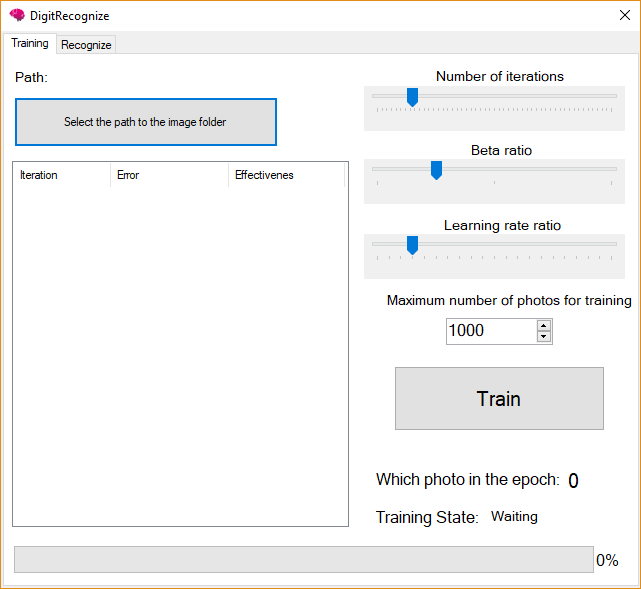
## In the first place, the user should choose whether he wants to convert one file or the entire file folder (marked as “1”). To do this, he should click one of the buttons marked as “2” in the picture. After click user will see folders/files explorer, where he can find right folder/file with images to convert.

When user select a path to folder or file, it will appear in the text box labelled as 3.

In the end user, should click on the convert button, to start converting files. Files will be saved in the subfolder of the main application directory named "data".

## 4.2 DigitRegonize

On the start of application, program looking for saved network state. This is a json file, saved in subfolder of the main application directory named “SavedNetwork” and file name is “Network”. After that, the following window should appear:



❶

❺

❹

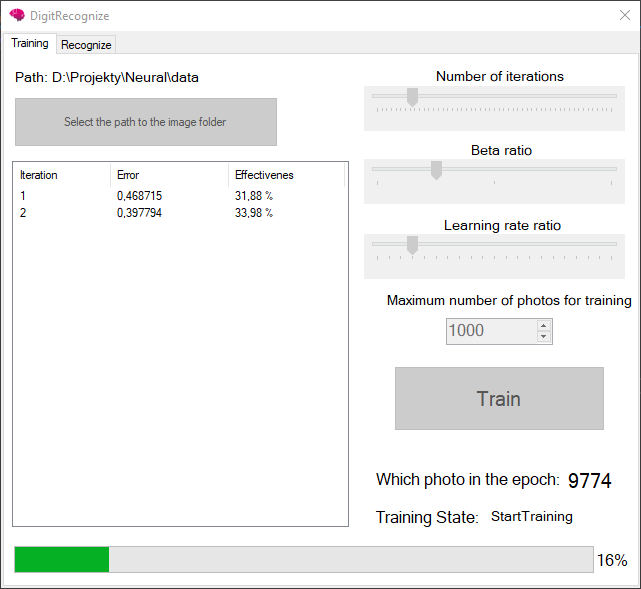
❸

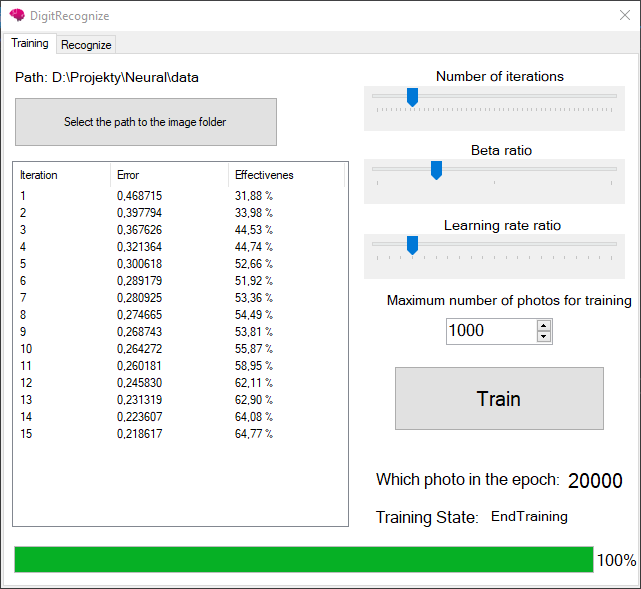
❷

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”), learning ratio(marked as “4”) and maximum number of photos for training( marked as “5”).

When all settings are set all you need is to click in to “Train” button

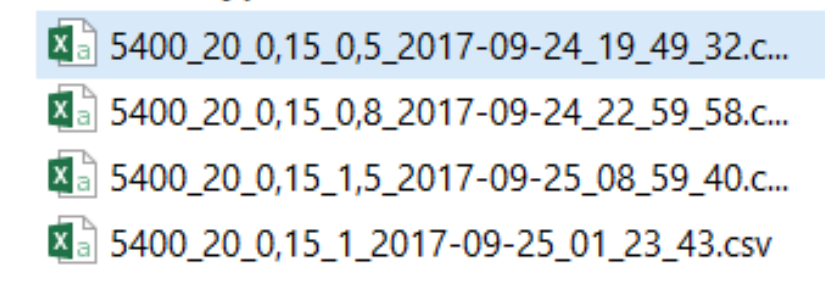




❷

❶

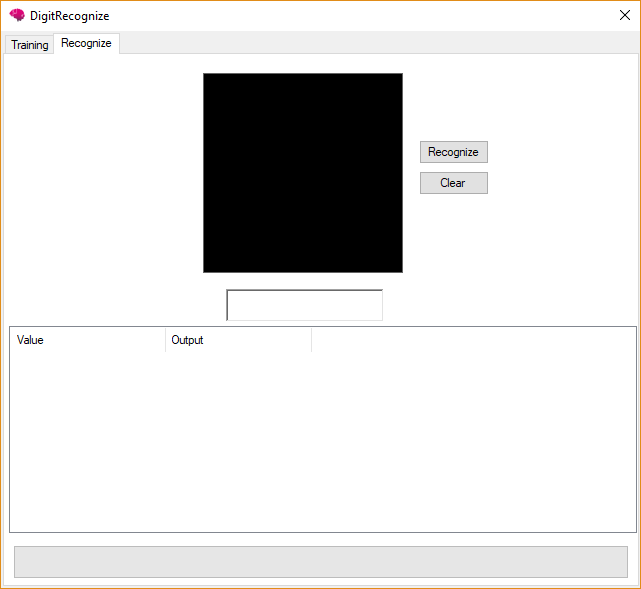
After the training process is completed, the user has possibility to check an error and effectiveness of executed training process in a table marked as “1’. Also results of the training, user can check in CSV files. All of the files are save in subfolder “Progresses” of the main application directory.



For the file name, we are using template:

numberOfExamplesOfEachDigit\_nuberOfIterations\_learningRate\_beta\_date\_time

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

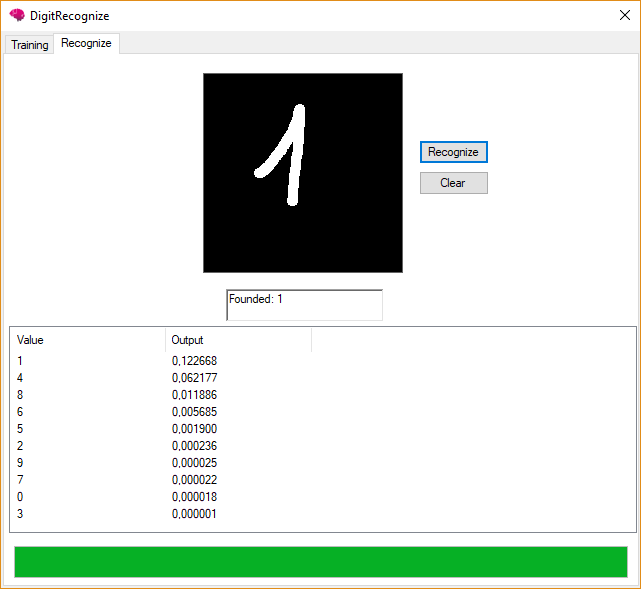
❸

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.



On the end of the application, program save the state of network in the Network.json file.

# 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

# 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, beta ratio and number of examples we achieved network with error equal to 0.3 (94% of effectiveness).

As we have predicted – in all test cases the best learning factory value oscillates around 0.1 and beta around 0.5. Also, it can be seen that the amount of iteration and the number of examples affects somehow the learning of the network.

Thanks to this project we had the opportunity to design and create our own handwriting recognition program using neural networks. We had the opportunity to broaden our knowledge of this topic and learn new types and algorithms for neural network learning. Our application can be a basis for creating our own OCR (Optical Character Recognition) implementation.