Gliwice, dn. 10 IX 2017 r.



**Biology Inspired Artificial Intelligence** Project Report

**Topic:**

Digits recognition

Authors:

1. Gurbiel Marcin
2. Kowalski Bartosz

# Project objectives

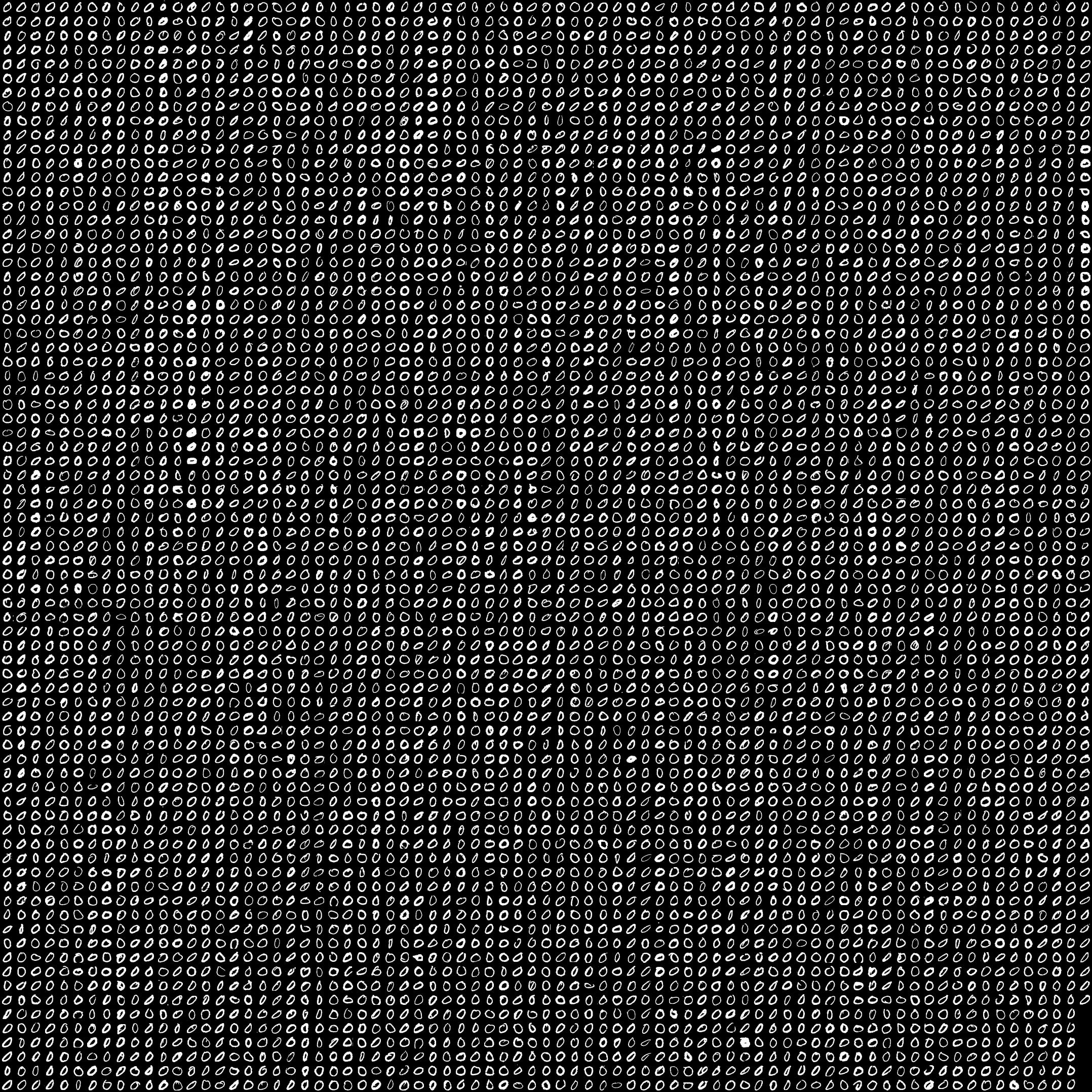
The goal of the project was developing an application capable of recognizing a defined set of manually written digits. Algorithm used in the application should be based on an artificial neural network.

# 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.

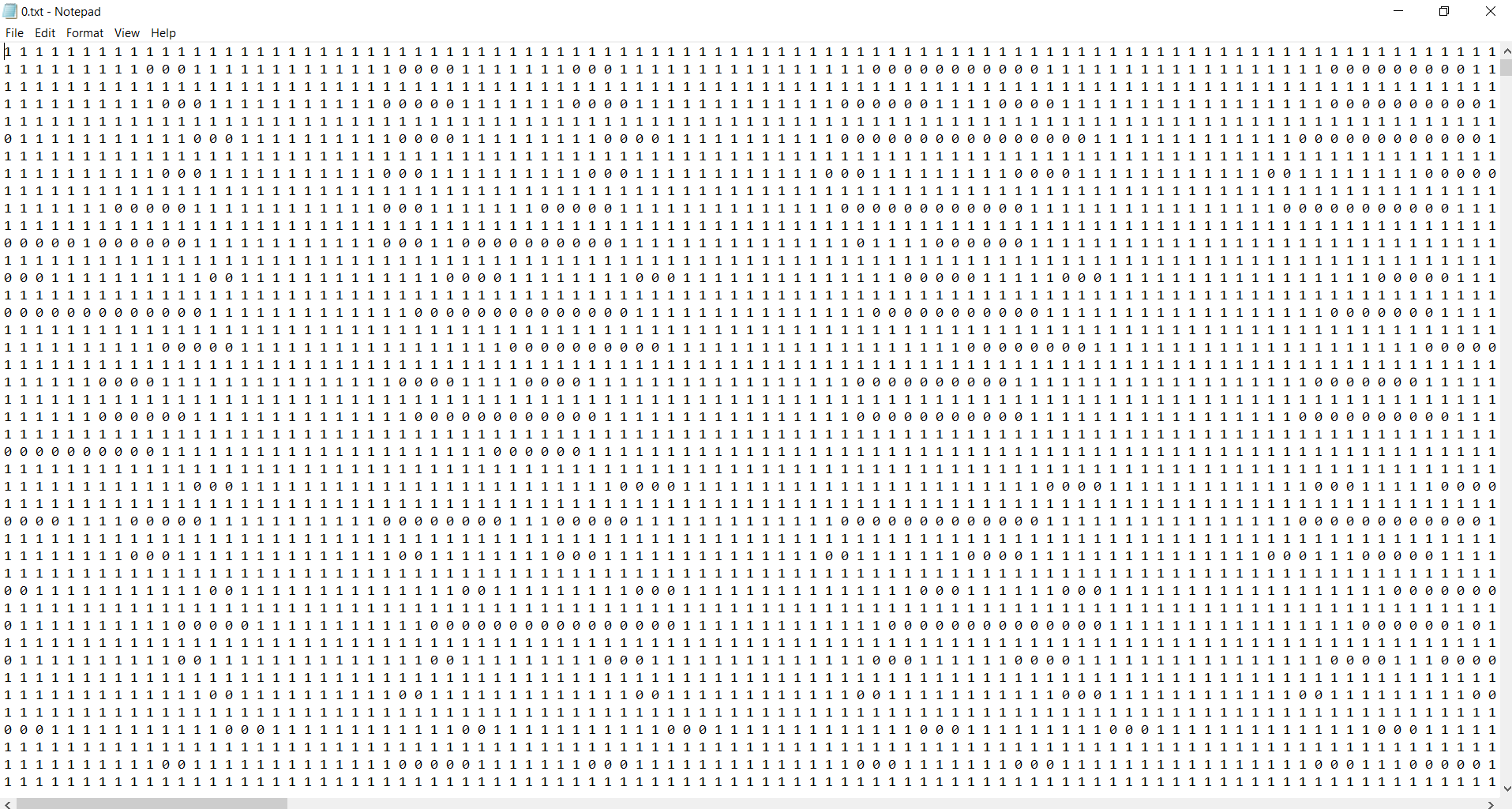
Below is presented an example of image for `0`



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

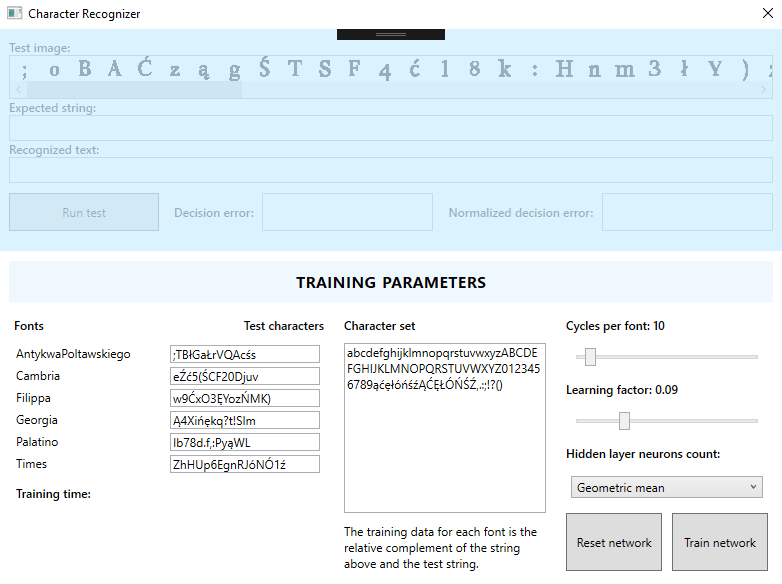
All training characters from all fonts are gathered into one collection. Before starting the training process, the collection is shuffled. The training process is executed for each element, and then the process is repeated. The number of cycles can be adjusted by user.

## 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:



❼

❻

❺

❹

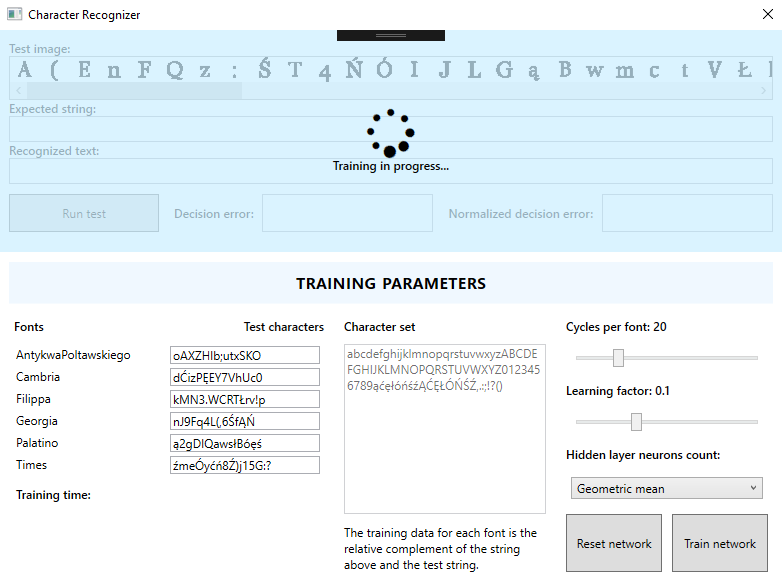
❸

❷

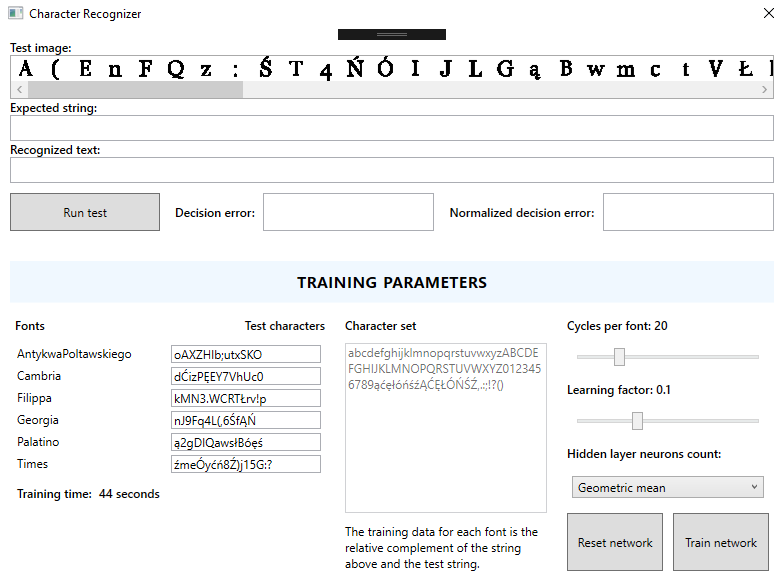
❶

1. Fonts section – displays available fonts (detected embedded resources) and randomly chosen, modifiable character sets that will be used as testing representations of given font.
2. Character set section – displays all characters recognizable by the network. This section is prefilled with the string contained in all prepared images. If different images are used, it should be adjusted to reflect the contents of the images.
3. Cycles per font slider – allows to adjust the number of training cycles
4. Learning factor slider – allows to adjust the learning factor of perceptrones
5. Hidden layer neurons count algorithm – allows to choose the algorithm resoinsible for calculation of the number of hidden layer neurons
6. Reset/Train network section – buttons allowing the user to invoke actions in the neural network. Consecutive training sessions will be put on top of each another, whereas resetting the network discards previous training.
7. Results section – unavailable unless the network is trained

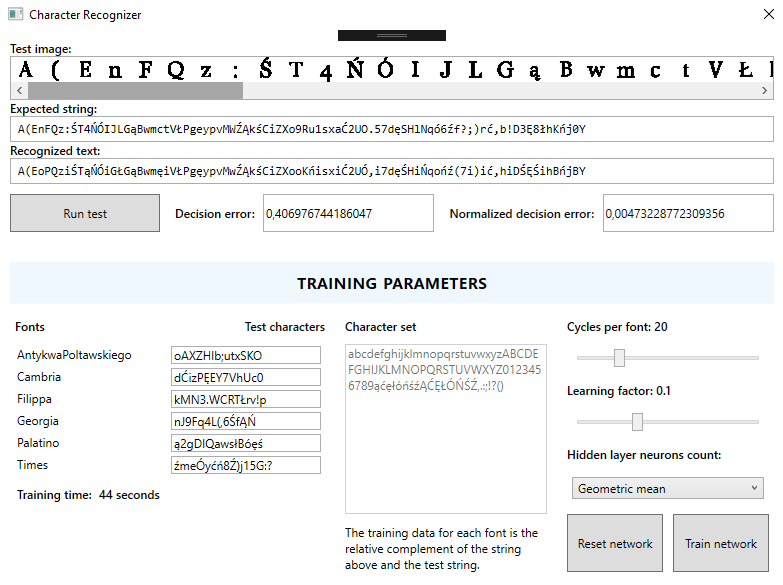
The user is capable of adjusting the parameters. To train the network, the user is supposed to click the *Train network* button. Once the training process is invoked, it can last several minutes.



After the training process is completed, the *Run Test* button is available. A test image is generated from random combination of the test element set.



After running the test, results are displayed:



❹

❸

❷

❶

1. Expected string – the text expected to be recognized
2. Recognized text – the result of the network’s recognition
3. Calculated decision error
4. Calculated normalized decision error

# Testing

The purpose of the testing process was to estimate influence of input parameters, such as learning factor, number of neurons in hidden layer and training cycles (per font), on the decision error and normalized decision error values. Only the examined parameter was changed during its tests, all the other parameters were assumed as constant values. We used all 86 available characters for testing.

## 5.1 Learning factor’s influence on decision error

In every test case, we checked the influence of the learning factor by training network with parameter values of: 0.05, 0.08, 0.11, 0.14, 0.17 in three test cases:

* Cycles = 30, Number of hidden neurons = Geometric mean
* Cycles = 35, Number of hidden neurons = Half inputs
* Cycles = 25, Number of hidden neurons = Two third inputs + outputs

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Learning factor | Test case 1 | | | Test case 2 | | | Test case 3 | | |
| Errors | DE [%] | NDE [‰] | Errors | DE [%] | NDE [‰] | Errors | DE [%] | NDE [‰] |
| 0,05 | 59 | 68.6 | 7.98 | 13 | 15.1 | 1.76 | 12 | 13.9 | 1.62 |
| 0,08 | 31 | 36.0 | 4.19 | 5 | 5.8 | 0.68 | 10 | 11.6 | 1.35 |
| 0,11 | 14 | 16.3 | 1.89 | 3 | 3.5 | 0.40 | 7 | 8.1 | 0.95 |
| 0,14 | 27 | 31.4 | 3.65 | 12 | 14.0 | 1.62 | 9 | 10.5 | 1.21 |
| 0,17 | 32 | 37.2 | 4.32 | 38 | 44.2 | 5.14 | 15 | 17.4 | 2.02 |

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.

It is worth noting that our network predicted correctly 83 of all 86 testing characters, so the decision error equals 3.5% and the normalized decision error is equal to 0.4‰.

## 5.2 Training cycles count influence on decision error

In every test case, we have checked the influence of number of cycles by training the network with parameter values of: 23, 30, 37, 44, 51, in three test cases:

* Learning factor = 0.1, Number of hidden neurons = Geometric mean
* Learning factor = 0.11, Number of hidden neurons = Half inputs
* Learning factor = 0.09, Number of hidden neurons = Two third inputs + outputs

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Training cycles count | Test case 1 | | | Test case 2 | | | Test case 3 | | |
| Errors | DE [%] | NDE [‰] | Errors | DE [%] | NDE [‰] | Errors | DE [%] | NDE [‰] |
| 23 | 25 | 29 | 3.38 | 10 | 11.6 | 1.35 | 9 | 10.4 | 1.21 |
| 30 | 13 | 15.1 | 1.76 | 10 | 11.6 | 1.35 | 9 | 10.4 | 1.21 |
| 37 | 11 | 12.7 | 1.49 | 6 | 7 | 0.81 | 8 | 9.3 | 1.08 |
| 44 | 13 | 15.1 | 1.76 | 6 | 7 | 0.81 | 5 | 5.8 | 0.67 |
| 51 | 8 | 9.3 | 1.08 | 5 | 5.8 | 0.67 | 11 | 12.7 | 1.49 |

Analyzing the test results we noticed that generally, increasing the number of training cycles minimizes the decision error values. In the first and the third test case, we observed a kind of deviation – for instance in the third test case, increasing training cycles by 7 doubled the decision and normalized decision error values. It can be explained by the randomized training process – to reduce its influence it is probably necessary to prepare more tests.

## 5.3 Learning cycles influence on decision error

In every test case, we have checked influence on number of neurons in hidden layer by training with 3 rules of thumb:

* Geometric mean (253 neurons)
* Half inputs (375 neurons)
* Two thirds of inputs plus outputs (586 neurons)

Test cases:

* Learning factor = 0.08, Cycles = 30
* Learning factor = 0.11, Cycles = 25
* Learning factor = 0.09, Cycles = 35

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Hidden neurons count | Test case 1 | | | Test case 2 | | | Test case 3 | | |
| Errors | DE [%] | NDE [‰] | Errors | DE [%] | NDE [‰] | Errors | DE [%] | NDE [‰] |
| 253 | 25 | 29 | 3.3 | 17 | 19.7 | 2.2 | 13 | 15.1 | 1.75 |
| 375 | 10 | 11.6 | 1.35 | 10 | 11.6 | 1.35 | 9 | 10.4 | 1.21 |
| 586 | 8 | 9.3 | 1.08 | 9 | 10.4 | 1.21 | 7 | 8.1 | 0.94 |

Increasing the number of hidden layer neurons decreased the values of decision and normalized decision error values. Although we obtained the best results using the rule of *two thirds of inputs + outputs*, such high number of hidden neurons had significant influence on the training time. The benefits from reducing normalized decision error are not satisfying enough to accept the increased duration of the training process.

# 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.