CHARACTER RECOGNITION

Handwritten character Recognition: Training a Simple NN for classification using MATLAB

Character recognition

Handwritten character recognition: Training a simple NN for classification with MATLAB

INTRODUCTION

Character recognition, usually abbreviated to optical character recognition or shortened OCR, is the mechanical or electronic translation of images of handwritten, typewritten or printed text (usually captured by a scanner) into machine-editable text. It is a field of research in pattern recognition, **artificial intelligence** and machine vision. Though academic research in the field continues, the focus on character recognition has shifted to implementation of proven techniques.

For many document-input tasks, character recognition is the most cost-effective and speedy method available. And each year, the technology frees acres of storage space once given over to file cabinets and boxes full of paper documents.

PROBLEM DESCRIPTION

Before OCR can be used, the source material must be scanned using an optical scanner (and sometimes a specialized circuit board in the PC) to read in the page as a bitmap (a pattern of dots). Software to recognize the images is also required.

The character recognition software then processes these scans to differentiate between images and text and determine what letters are represented in the light and dark areas.

Older OCR systems match these images against stored bitmaps based on specific fonts. The hit-or-miss results of such pattern-recognition systems helped establish OCR's reputation for inaccuracy.

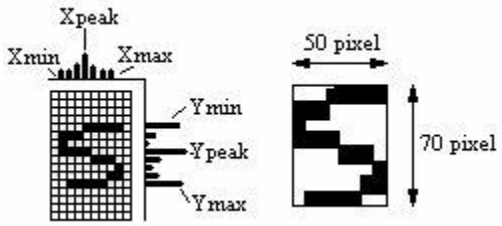
Today's OCR engines add the multiple algorithms of **neural network technology** to analyze the stroke edge, the line of discontinuity between the text characters, and the background. Allowing for irregularities of printed ink on paper, each algorithm averages the light and dark along the side of a stroke, matches it to known characters and makes a best guess as to which character it is. The OCR software then averages or polls the results from all the algorithms to obtain a single reading.

OCR software can recognize a wide variety of fonts, but handwriting and script fonts that mimic handwriting are still problematic, therefore additional help of neural network power is required. Developers are taking different approaches to improve script and handwriting recognition. As mentioned above, one possible approach of handwriting recognition is with the use of neural networks.

Neural networks can be used, if we have a suitable dataset for training and learning purposes. Datasets are one of the most important things when constructing new neural network. Without proper dataset, training will be useless. There is also a saying about pre-processing and training of data and neural network: "Rubbish-in, rubbish-out". So how do we produce (get) a proper dataset? First we have to scan the image. After the image is scanned, we define processing algorithm, which will extract important attributes from the image and map them into a database or better to say

dataset. Extracted attributes will have numerical values and will be usually stored in arrays. With these values, neural network can be trained and we can get a good end results. The problem of well defined datasets lies also in carefully chosen algorithm attributes. Attributes are important and can have a crucial impact on end results. The most important attributes for handwriting algorithms are:

1. Negative image of the figure, where the input is defined as 0 or 1. 0 is black, 1 is white, values in between shows the intensity of the relevant pixel.
2. The horizontal position, counting pixels from the left edge of the image, of the center of the smallest rectangular box that can be drawn with all "on" pixels inside the box.
3. The vertical position, counting pixels from the bottom, of the above box.
4. The width, in pixels, of the box.
5. The height, in pixels, of the box.
6. The total number of "on" pixels in the character image.
7. The mean horizontal position of all "on" pixels relative to the center of the box and divided by the width of the box. This feature has a negative value if the image is "leftheavy" as would be the case for the letter L.
8. The mean vertical position of all "on" pixels relative to the center of the box and divided by the height of the box.
9. The mean squared value of the horizontal pixel distances as measured in 6 above. This attribute will have a higher value for images whose pixels are more widely separated in the horizontal direction as would be the case for the letters W or M.
10. The mean squared value of the vertical pixel distances as measured in 7 above.
11. The mean product of the horizontal and vertical distances for each "on" pixel as measured in 6 and 7 above. This attribute has a positive value for diagonal lines that run from bottom left to top right and negative value for diagonal lines from top left to bottom right.
12. The mean value of the squared horizontal distance tunes the vertical distance for each "on" pixel. This measures the correlation of the horizontal variance with the vertical position.
13. The mean value of the squared vertical distance times the horizontal distance for each "on" pixel. This measures the correlation of the vertical variance with the horizontal position.
14. The mean number of edges (an "on" pixel immediately to the right of either an "off pixel or the image boundary) encountered when making systematic scans from left to right at all vertical positions within the box. This measure distinguishes between letters like "W" or "M" and letters like "I" or "L."
15. The sum of the vertical positions of edges encountered as measured in 13 above. This feature will give a higher value if there are more edges at the top of the box, as in the letter "Y."
16. The mean number of edges (an "on" pixel immediately above either an "off pixel or the image boundary) encountered when making systematic scans of the image from bottom to top over all horizontal positions within the box.
17. The sum of horizontal positions of edges encountered as measured in 15 above.



Picture 1: Example of image defined attributes

GOAL OF THE SEMINAR

The objective of this seminar is to identify handwritten characters with the use of neural networks. We have to construct suitable neural network and train it properly. The program should be able to extract the characters one by one and map the target output for training purpose. After automatic processing of the image, the training dataset has to be used to train "classification engine" for recognition purpose. The program code has to be written in MATLAB and supported with the usage of Graphical User Interface (GUI).

SOLUTION APPROACH

To solve the defined handwritten character recognition problem of classification we used MATLAB computation software with Neural Network Toolbox and Image Processing Toolbox add-on. The computation code is divided into the next categories:

Automatic Image Preprocessing

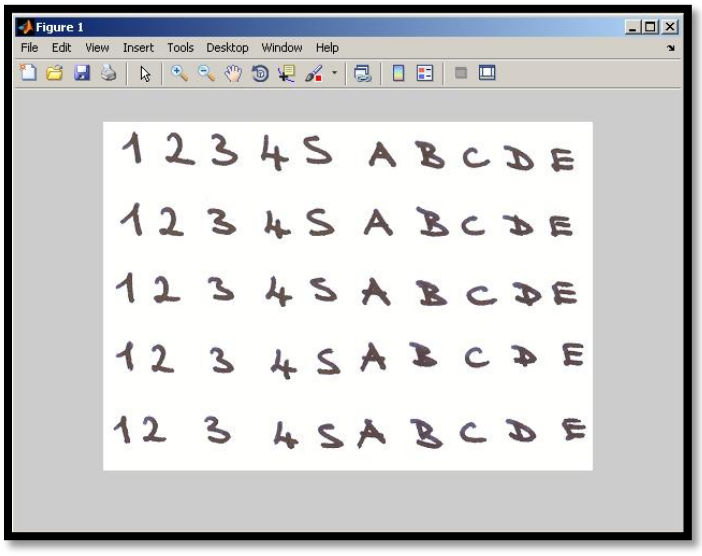
The image is first being converted to grayscale image follow by the threshing technique, which make the image become binary image. The binary image is then sent through connectivity test in order to check for the maximum connected component, which is, the box of the form. After locating the box, the individual characters are then cropped into different sub images that are the raw data for the following feature extraction routine.

The size of the sub-images are not fixed since they are expose to noises which will affect the cropping process to be vary from one to another. This will causing the input of the network become not standard and hence, prohibit the data from feeding through the network. To solve this problem, the sub-images have been resize to 50 by 70 and then by finding the average value in each 10 by 10 blocks, the image can be down to 5 by 7 matrices, with fuzzy value, and become 35 inputs for the network. However, before resize the sub-images, another process must be gone through to eliminate the white space in the boxes.

Read Image

This cell of codes read the image to MATLAB workspace.

```
I = imread('training.bmp');
imshow(I)
```

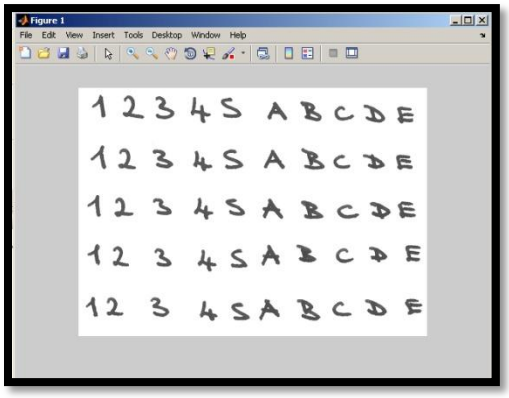


Picture 2: Image of training.bmp image

Convert to grayscale image

This cell of codes convert the RGB to gray.

```
Igray = rgb2gray(I);
imshow(Igray)
```

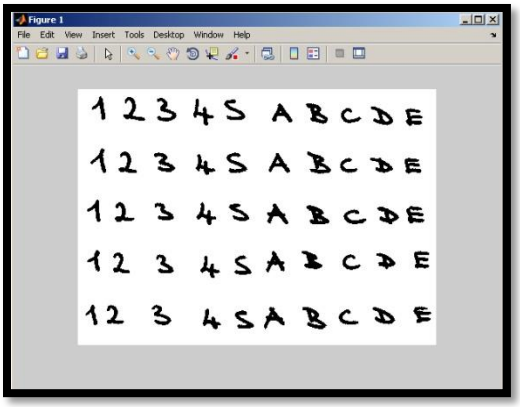


Picture 3: Image converted into grayscale

Convert to binary image

This cell of codes convert the gray to binary image.

```
Ibw = im2bw(Igray,graythresh(Igray));
imshow(Ibw)
```

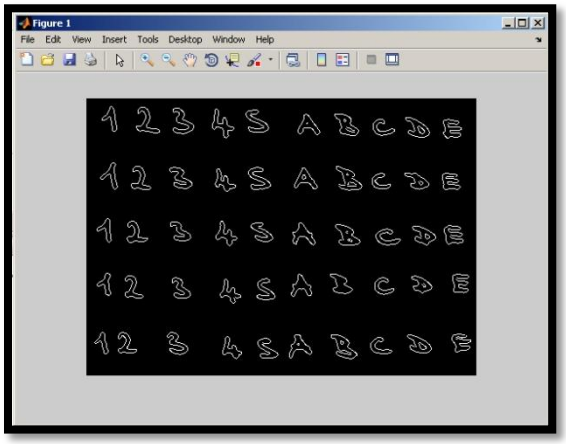


Picture 4: Image converted into binary image

Edge detection

This cell of codes detect the edge of the image.

```
Iedge = edge(uint8(Ibw));
imshow(Iedge)
```



Picture 5: Detected edges of the image

Morphology

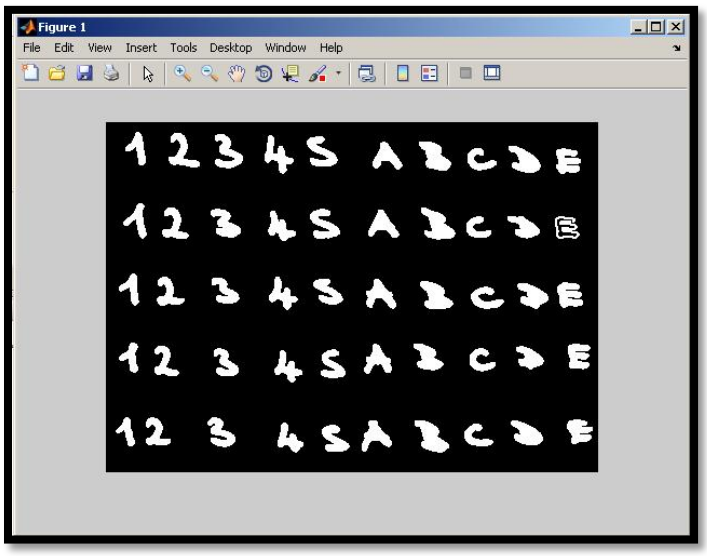
This cell of codes perform the image dilation and image filling on the image.

Image Dilation

```
se = strel('square',2);
Iedge2 = imdilate(Iedge, se);
imshow(Iedge2);
```

Image Filling

```
Ifill= imfill(Iedge2,'holes');
imshow(Ifill)
```



Picture 6: Image convertet into grayscale

Blobs analysis

This cell of codes find all the objects on the image, and find the properties of each object.

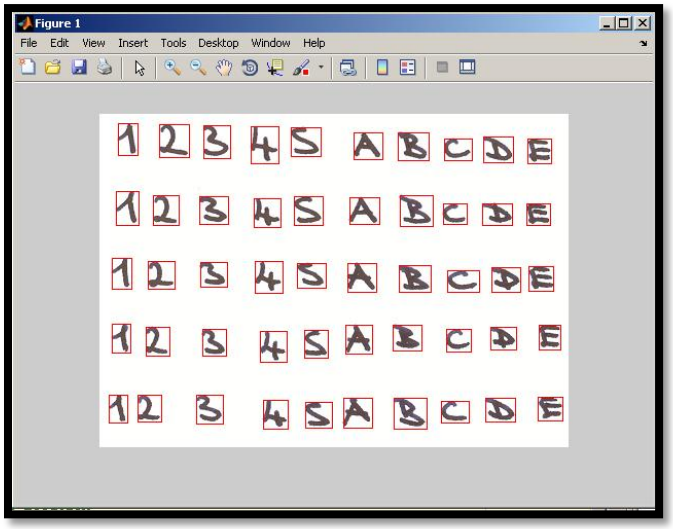
```
[Ilabel num] = bwlabel(Ifill);
disp(num);
Iprops = regionprops(Ilabel);
Ibox = [Iprops.BoundingBox];
Ibox = reshape(Ibox,[4 50]);
imshow(I)
```

50

Plot the Object Location

This cell of codes plot the object locations.

```
hold on;
for cnt = 1:50
rectangle('position',Ibox(:,cnt),'edgecolor','r');
end
```

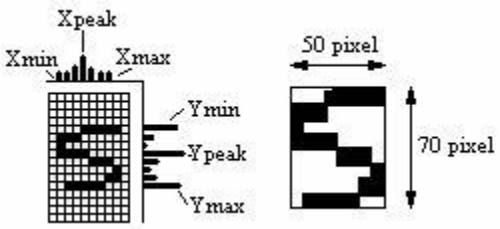


Picture 7: Image with object location

By this, we are able to extract the character and pass to another stage for future extraction and proccesing

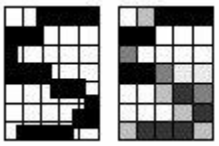
Feature Extraction

The sub-images have to be cropped sharp to the border of the character in order to standardize the sub-images. The image standardization is done by finding the maximum row and column with 1s and with the peak point, increase and decrease the counter until meeting the white space, or the line with all 0s. This technique is shown in figure below where a character "S" is being cropped and resized.



Picture 8: Cropped and resized picture

The image pre-processing is then followed by the image resize again to meet the network input requirement, 5 by 7 matrices, where the value of 1 will be assign to all pixel where all 10 by 10 box are filled with 1s, as shown below:



Picture 9: Image resize again to meet the network input requirement

Finally, the 5 by 7 matrices is concatenated into a stream so that it can be feed into network 35 input neurons. The input of the network is actually the negative image of the figure, where the input range is 0 to 1, with 0 equal to black and 1 indicate white, while the value in between show the intensity of the relevant pixel.

By this, we are able to extract the character and pass to another stage for future "classification" or "training" purpose of the neural network

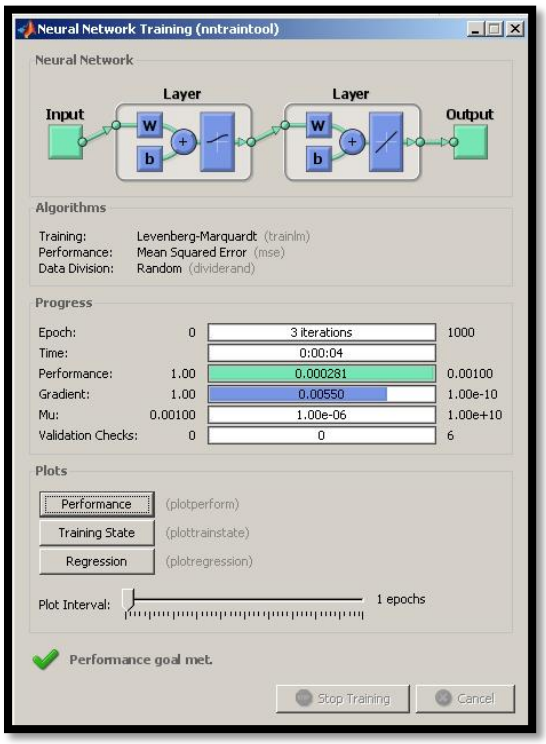
Creating Vectors data for the Neural Network (objects)

These few line of codes creates training vector and testing vector for the neural network. This is to match the input accepted by the neural network function. The front 4 rows will be used to train the network, while the last row will be used to evaluate the performance of the network.

```
P = out(:,1:40);
T = [eye(10) eye(10) eye(10) eye(10)];
Ptest = out(:,41:50);
```

Creating and training of the Neural Network

Create and Train NN. The "edu_createnn" is a function file to create and train the network by accepting the training-target datasets. Because of the nature of the problem, we used feed forward back propagation neural network for classification or with other words we were experimenting with multi layer perceptron (MLP). Below is a picture of MLP structure with two hidden layers [35 10] neurons and sigmoid, linear activation function.



Picture 10: Training the Neural Network

```
%Training with the help of training function
net = edu_createnn(P,T);
```

We can also experiment and test the network separatly

```
net = newff(P,T,[35], {'logsig'})
%net.performFcn = 'sse';
net.divideParam.trainRatio = 1; % training set [%]
net.divideParam.valRatio   = 0; % validation set [%]
net.divideParam.testRatio  = 0; % test set [%]
net.trainParam.goal = 0.001;
[net,tr,Y,E] = train(net,P,T);
```

Testing the Neural Network

Simulate the testing dataset with the sim command.

```
%% Testing the Neural Network
[a,b]=max(sim(net,Ptest));
disp(b);

% Result
1 2 3 4 5 6 7 8 9 10
```

Detailed explanations of the entire code can be also found inside the main MATLAB program code within green comments. Please refer to the MATLAB program for the whole code.

Encoding/Decoding

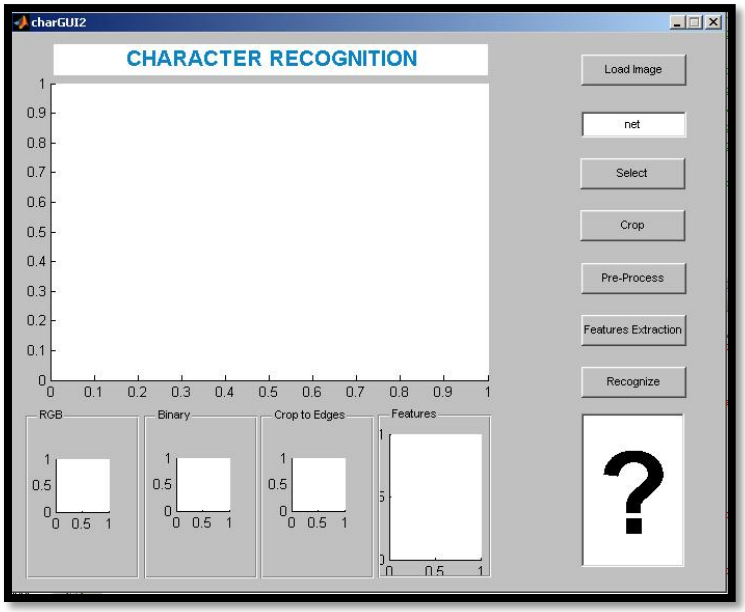
In our recognition we use/train the net with ten different signs (characters). Five were numbers and five were characters, so input is a matrix of size 1x10 and [1,2,3,4,5,'A','B','C','D','E'] values. Because we cannot define network target as a character we have to use numbers – integers. So on output of the neural network we get matrix with values [1,2,3,4,5,6,7,8,9,0] instead of above, first input matrix. From that reason we have to encode/decode the output of our application. The encoding is made with simple code table, in which every index gets a new value. So index 0 gets character E, index 6 gets A, index 3 gets 3 and so on. I enclosed encoding code below.

```
if num>5
   num = num -5;
   % In this case, output is character, defined as a number from num
   charset = ['A','B','C','D','E'];
else
   % Else, output is number, defined from num
   charset = [1,2,3,4,5];
end

oknum = charset(1,num);
set(handles.editResult, 'string',oknum);
```

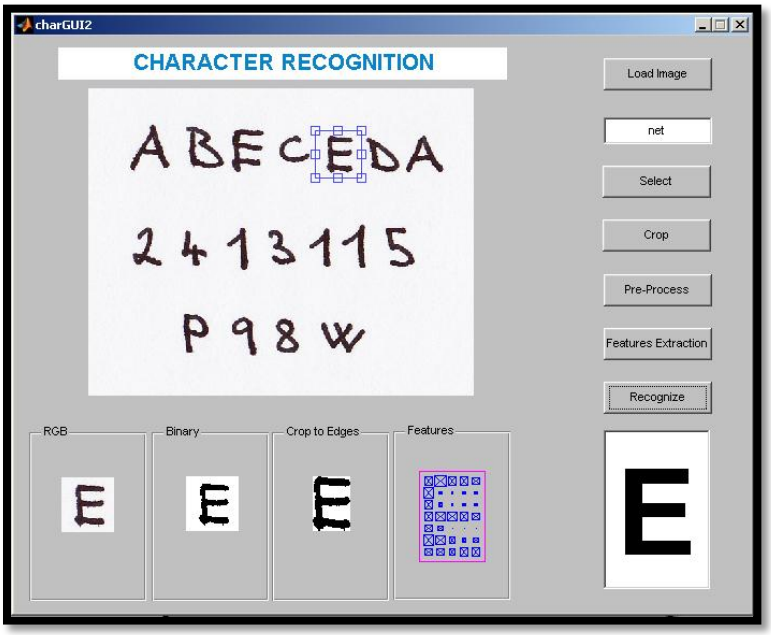
Application and Graphical User Interface

The character recognition application can be used in two different ways. First way is to type every command inside the MATLAB console and workspace on hand. The second way is to use already pre-prepared Graphical User Interface. The GUI consists of two files. First file include all necessary programming code, and the second file include visible interface shapes and forms. The interface works like the workflow of recognition process. First we load the image, than we select the character and after that we click crop, pre-process, feature extraction and finally recognize. On every stage, GUI shows us a new image, which is unique for the each step. The images can be viewed in the Main window, RGB, Binary, Crop to Edges and Features window.



Picture 11: Graphical User Interface of Applicationfor character recognition

For test purposes I made a test also with the new set of characters. This time they are not ordered. You can clearly see the new word ABECEDA in which letter E is recognized correctly. If we try to recognize letter P, W or number 9, 8 on this picture, recognition will not be correct. False recognition is in fact logical, because our neural network is not properly trained for those characters. It is trained only for A,B,C,D,E and 1,2,3,4,5.



11: Test example and true recognition of letter E

EXPECTED RESULTS AND CONCLUSION

When I look back on the whole process of handwritten character recognition, I can combine my conclusions into more points:

- Handwritten character recognition is a complex problem, which is not easily solvable. The main arguing which I had at the beginning was around dataset and database. I put a lot of energy into finding already pre-prepared datasets, which could be useful for our neural network, but at the end of my research I end up with no training data and no neural

network. The main problem lies in pre-processing of data. Datasets from the "internet" were quite large, they contained a lot of data (a lot of data is excellent for training, validation and testing), but too little information on attributes and image extractions (which were used by them-authors of datasets). From that reason, I was able to train the neural network, but not able to test it on my own example. Final result was only X percents of true classification, which was not what I expected. Next, I decided to go on my own and make my own recognition example, which will include also self testing algorithm. The bad side of own application is that, I will not have a lot of training data, therefore output from the neural network will be questionable.

- Described application of character recognition can be divided into three main parts. Image preprocessing to get the training data, training the neural network and at the end testing with final recognition results.

- Image preprocessing to get the training data for the neural network is based on input training.bmp image which has exactly 50 characters (25 characters and 25 numbers). Every type of character from all 50 characters is repeated only five times. Compared with other professional neural network application 5 repetitions is very small dataset. Other datasets usually consist of around 800 repetitions, so 800 to my 5 is quite a difference. Another downsize is, that I prepared my training data only from 10 characters [1,2,3,4,5,A,B,C,D,E] and not from the whole alphabet. I tried to train the network also with 32 English alphabet characters, but the outcome from the net was not usable. Too many characters were classified false. I discovered that optimal number of characters (for such application) is about ten. More than that is already destroying recognition result. In any case, I am working with only a few data and a saying that Rubbish-in, Rubbish-out is true. From the other point of view I learned quite a lot how to construct, deal and implement character recognition application.

- Training and testing the neural network was only a matter of two MATLAB commands. I decided to use Multilayer Perceptron with two hidden layers of 35 and 10 neurons as a neural network. I also enclosed the whole MATLAB program, where it can easily be seen all of performance, training state and regression graphs... At this point I would like to mention, that I have 50 data (characters) on the picture. 40 of them were used for training and last 10 were used for simulating and testing.

- Final testing of the Graphical User Interface show, that such an approach is quite user friendly. User does not have to type in MATLAB command all command on hand, but can click on the buttons and test the character recognition. GUI uses already pre-trained neural network from precedent computation steps. The result of NN is numerical matrix in form like [1,2,3,4,5,6,7,8,9,0]. Final result which displays in the GUI is calculated with the help of encoding/decoding and index mapping. Encoding is necessary, because we can not define targets for neural network as a characters, but only as numbers. So, at the end, numbers has to be "translated into characters". The last test was done on new image test.bmp, which includes word ABECEDA and few other characters. Most of the characters are recognized as positive match. Those characters that are not recognized as true are on the image also for testing purposes, because I deliberately put there for example character P, which was not included into the neural net training. Therefore also recognition is not possible.

APPENDIX

- My own writings on which images was produced

- MATLAB application