Character Recognition

Using MATLAB Artificial Neural Network Tool and MNIST Dataset

Samim Zahoor IT-03-13 Asif Iqbal IT-17-13

June 2016

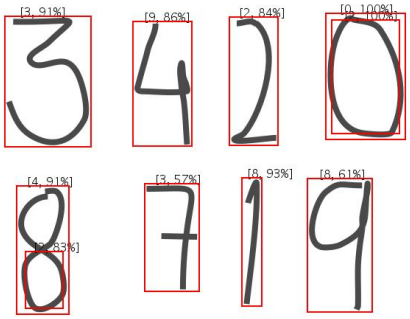

Figure 1: Recognizing digits in an image

1 The project

1.1 Introduction

The aim of this project is Recognizing Handwritten Digits using a Two-Layer Perceptron and the MNIST Dataset. We use MATLAB Neural Network Toolbox to implement the Two-Layer Perceptron. Neural Network Toolbox provides algorithms, functions, and apps to create, train, visualize, and simulate neural networks. You can perform classification, regression, clustering, dimensionality reduction, time-series forecasting, and dynamic system modeling and control. The toolbox includes convolutional neural network and autoencoder deep learning algorithms for image classification and feature learning tasks.

1.2 MNIST Database

The MNIST database of handwritten digits, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting.The original black and white (bilevel) images from NIST were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. the images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field.

2 Implementation

2.1 Load the datasets.

The methods loadMNISTImages and loadMNISTLabels are used to load the MNIST dataset as it is stored in a special file format.

2.2 Training the Neural Network.

We use the neural network toolbox which provides a simple and robust GUI. Start the neural net wizard in Matlab using the command **nnstart;**.

The figure shows number of epochs and other useful information while training the network.

2.3 Using the trained network.

We use the helper function **run_simulation** for using the network. An important part of this part is to make the input images compatible with the network input. This involves some image processing. First we divide the image into parts using **regionprops()** function to create bounding boxes for each of the digit in the image. Resizing the digits to **28 X 28** sub images with padding on the outside. Finally the image is converted into a column vector **784** of rows. The code for this function is given.

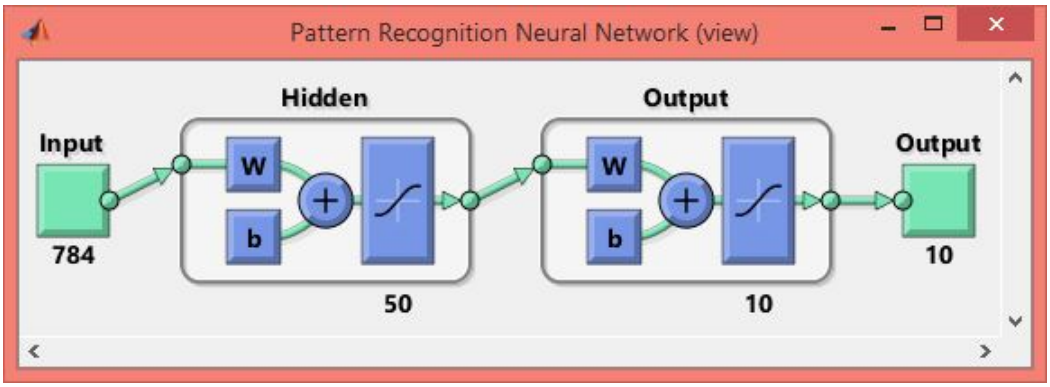

Figure 2: Matlab Neural Network Wizard

```
function [  ] = run_simulation( image_path ,samnet )
    %UNTITLED2 Summary of this function goes here
    %   Detailed explanation goes here

    I = imread(image_path);
    I_gray = rgb2gray(I);

    %Detect edges.
    I_edge_detected = edge(I_gray, 'sobel');

    %Do image dilation
    se = strel('square', 5);
    I_dilated = imdilate(I_edge_detected, se);

    %create bounded boxes
    bounded_boxes = regionprops(I_dilated, 'BoundingBox');
    op_struct = struct();

    imshow(padarray(I_dilated,[60,60]));
    %vector_of_images = zeros(100);

    for k = 1 : length(bounded_boxes)

      %get each bounded box one by one.
      thisBB = bounded_boxes(k).BoundingBox;

      %crop image according to bounded boxes detected by regionprops()
      %box_num is the box number detected above. For test example it should be
      %same as the number predicted because they are in order

      s = I_dilated(round(thisBB(2):thisBB(2)+thisBB(4)),thisBB(1)...
          :thisBB(1)+thisBB(3));

      %Resize sub image and add zero padding
      s_resized = imresize(s, [20,20]);
      s_padded = padarray(s_resized, [4,4]);

      %reshape the image into a vector to make it compatible with input
      ip = reshape(s_padded, 28*28,1);
      op = sim(samnet,ip);
```

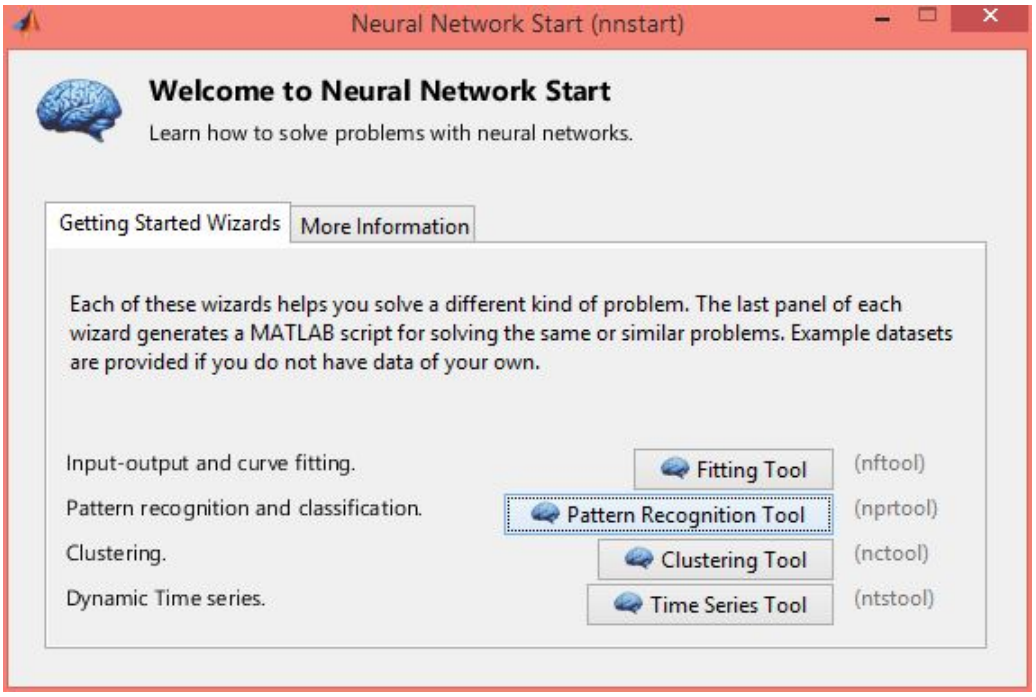

Figure 3: Matlab Neural Network Wizard

```
    [confidence, value] = max(op);
    op_struct(k).value = value-1;
    op_struct(k).confidence = confidence;

end

%Plot the original Image with each bounded box that is detected. The value
%of the number predicted and also the confidence measure is printed above
%the box.

subplot(1,1,1)
n = length(bounded_boxes);

RGB = I;
for k = 1 : n
    %get the value and confidence for each bounded box number and print
    %them over the image. The image is overwritten by text.
    thisBB = bounded_boxes(k).BoundingBox;

    x_pos = thisBB(1)+thisBB(3)/7;
    y_pos = thisBB(2)-20;

    conf_str = num2str(round(op_struct(k).confidence*100));
    value_str = num2str(op_struct(k).value);

   RGB = insertText(RGB,[x_pos y_pos],['[',value_str, ', ', conf_str,
'%]'],'FontSize',16,'TextColor','black', 'BoxOpacity',0.0);
end;
```

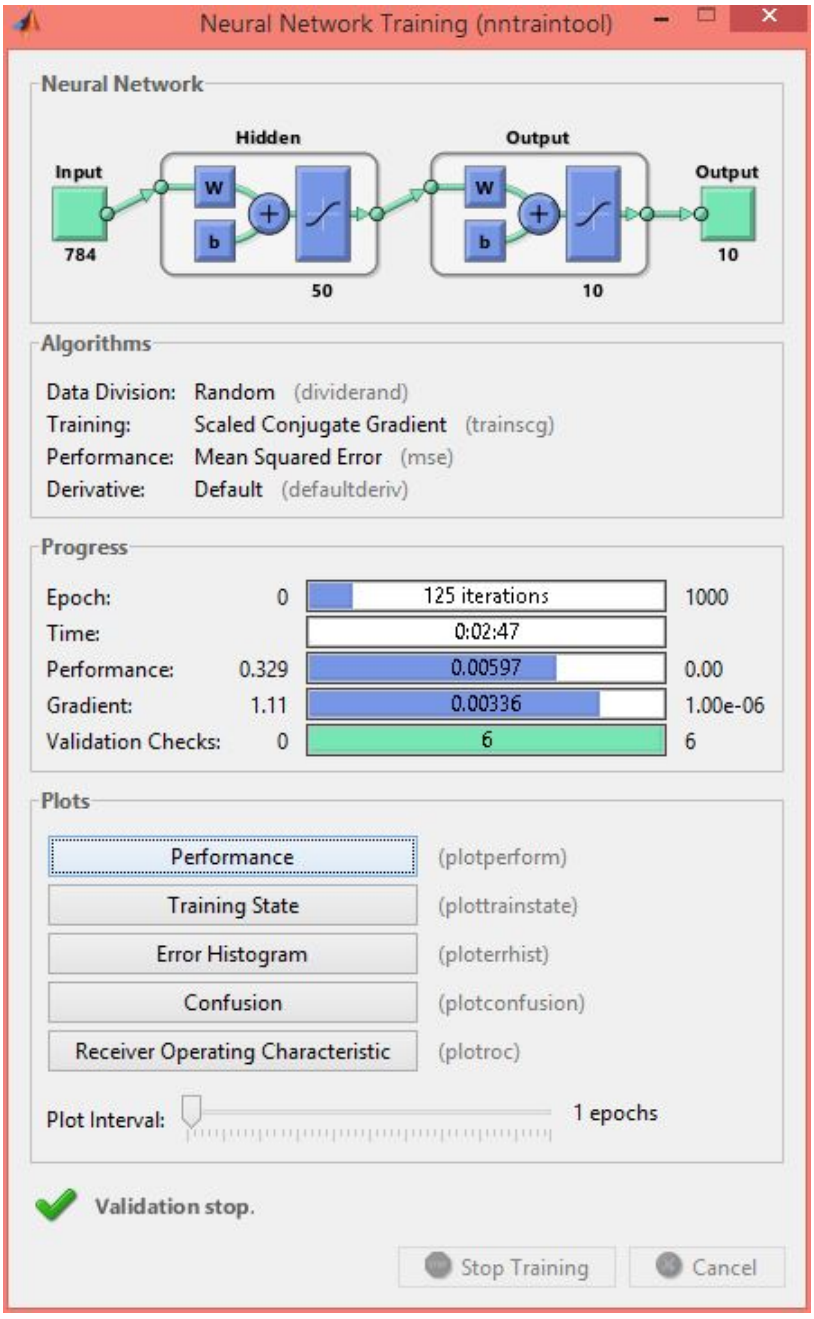

Figure 4: Training process.

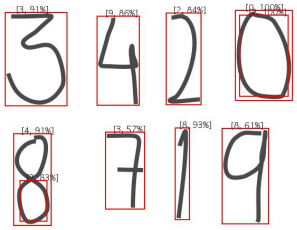

Figure 5: Recognizing digits in an image

```
    imshow(RGB);
    for k = 1 : n
      %Draw bounded box rectangles.
      thisBB = bounded_boxes(k).BoundingBox;
      rectangle('Position', [thisBB(1),thisBB(2),thisBB(3),thisBB(4)],...
      'EdgeColor','r','LineWidth',2 )

    end;

end
```

2.4 Final Output

After some running the above script we get the final output. Each object in the image is recognized and marked by a bounding box. The value that it is classified as is printed at the top of the box along with the confidence measure.