Project I: Digit Recognition

Haitham Babbili

Chalmers University

8 May 2020

Abstract

The aim of this project is to increase the understanding of machine learning and Fully Convolution Network (FCN) and how to design FCN in an efficient way to get more than 97% correct answer after train it with MNIST dataset of the image contain digits and label for this digits then test it by input to this FCN to detect a digit in an image. Therefore, a fully convolutional neural network used to estimate the digit. Non-max suppression and thresholding is implementation to get the greatest approach to the correct digit.

1. Method

The MNIST handwritten digit database was chosen to train our FCN. MNIST has 60,000 labelled training examples and 10,000 examples for testing. The training example was divided into 75% for training and 25% for validation while the testing kept untouched to make an evaluation on the performance of our FCN. Reshape the image data into 4D matrix 28x28x1x60000 (1 is because of greyscale) and convert labels data into categorical to be used on FCN after normalized to keep it in range between 0 to 1. In this way, all images have the same size. This image will be the input of FCN. The network consists of layers; each layer has a number of nodes connected to nodes in the next layer with weight w_k while the next node has initial value call bias w_0. The input value multiplied with a weight w_k and add to bias then feed it to active function (sigmoid)

$$\boxed{p = \frac{e^y}{1 - e^y}}$$

This probability used to calculate the log-likelihood loss in every node at every layer. But the network contains a huge amount of weights and to train it, that will take a lot of time and calculates. The Full Conventional Network will use, which is simply CNN, but not all layers are connected to each other that will allow us to use it in a full image in different sizes. The first layers in full-CNN is an image input layer 28x28x1 non-Normalized.
The next layer is the convolution Layer which apply 10 5x5 sliding convolution filters and calculates the multiplication between the input and weight the adding a bias term of the weights and the input, and then adding a bias term. Then batchNormalizationLayer use to normalize the output of the convolution layer. After that to increase the training speed get better sensitivity to use the ReLU layer and also act like an activation function or sigmoid function. And the next layer is maxPooling2dLayer (2,'Stride', 2) which means downsample the regions by 2x2 and keep the maximum value, the output will still have the same number of filters but less dimensions. The 6th layer was convolution Layer which apply 10 3x3 sliding convolution filters. The 7th is batchNormalizationLaye and 8th layer is the ReLU layer then maxPooling2dLayer (2,'Stride', 2).
The next layers are repeating the secon three-layer with the same number of filters and dimensions. The resone is to make the network reach to [1x1 10] and for that reason the averagePooling2dLayer has been added. SoftmaxLayer use to get probability output. At the end pixelClassificationLayer used for semantic segmentation and isolate the background image from the digit. After defining the layers, the training option and the condition should be defined like which activation function should be chosen, iteration rate, and learning rate, epoch, drop point for iteration and which validation data can be used. The next step will be to train the network. Therefore, the trainNetwork Matlab function used with layers and training options that defined before. Then testing the network by using the test image from the MNIST dataset and calculate the accuracy by dividing the summation of the number of testing accurately data over the number of the test dataset.

After training and testing the accuracy of the network by using training and testing dataset from MNIST. The image provided in the canvas was implemented on the network by using semantic segmentation function which give each pixel a label and percentage on how sure our network is about labeling. Before that the image was converted to gray and normalize then rescaled to fit the MINST image size. After semantic segmentation the image result is rescaled to the original size and the threshold chosen depending on the value in that image. Scale space was created to handle the differences between the image and the input layer size. The non-maxima suppression is to find the points which is higher than their neighboring points by use the threshold.

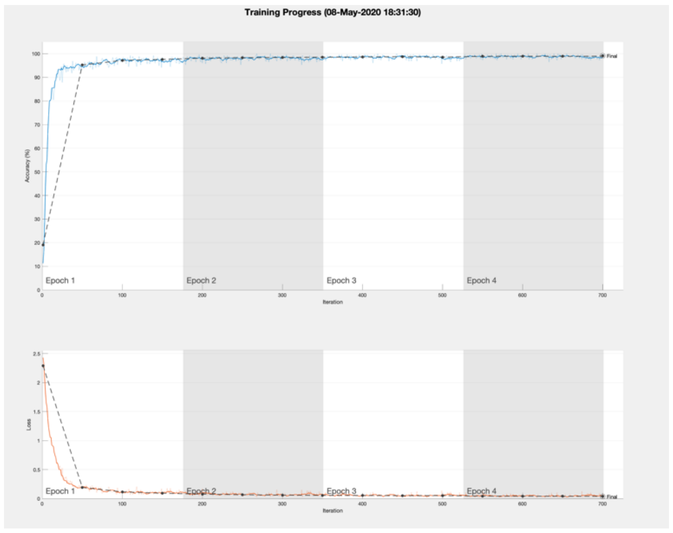
Experimental Evaluation

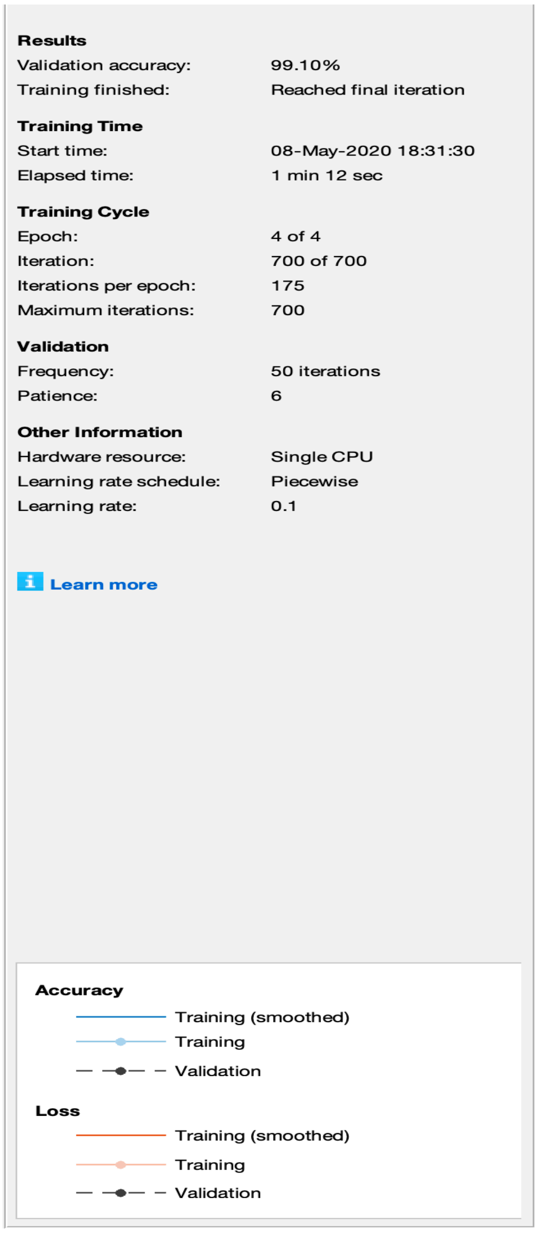
The parameter used for training the FCN layers

- the input image layer [28x28x1]
- first convolution2dLayer (5,10)
  size of slide window 5x5x10 and learn parameter (5x5)x10+10= 260
  Where 10 is number of filter
- first max pool(2,2) that mean down-samble by halve and filter number stay same
- second convolution2dLayer(3, 10) which mean smaller size of filter and same number of filter 3x3x10 so learning parameter will be (3x3x10)x10+10 = 910
- second max pool(2,2)
- convolution2dLayer(3, 10) which mean smaller size of filter and same number of filter 3x3x10 so learning parameter will be (3x3x10)x10+10 = 910
- convolution2dLayer(3, 10) which mean smaller size of filter and same number of filter 3x3x10 so learning parameter will be (3x3x10)x10+10 = 910
- total learning parameter = 260+910+910+910 = 2990

The parameter used for the FCN options

- `'sgdm',`
- `'ExecutionEnvironment','auto',`
- `'InitialLearnRate', 0.1,`
- `'MaxEpochs',4,`
- `'LearnRateDropFactor',0.5,`
- `'LearnRateDropPeriod',50,`
- `'LearnRateSchedule','piecewise',`
- `'ValidationData',{cnn_valid_images,cnn_valid_labels},`
- `'ValidationFrequency',50,`
- `'ValidationPatience',6,`
- `'MiniBatchSize',256,`
- `'Plots','training-progress'`





Results

Validation accuracy:	99.10%
Training finished:	Reached final iteration

Training Time

Start time:	08-May-2020 18:31:30
Elapsed time:	1 min 12 sec

Training Cycle

Epoch:	4 of 4
Iteration:	700 of 700
Iterations per epoch:	175
Maximum iterations:	700

Validation

Frequency:	50 iterations
Patience:	6

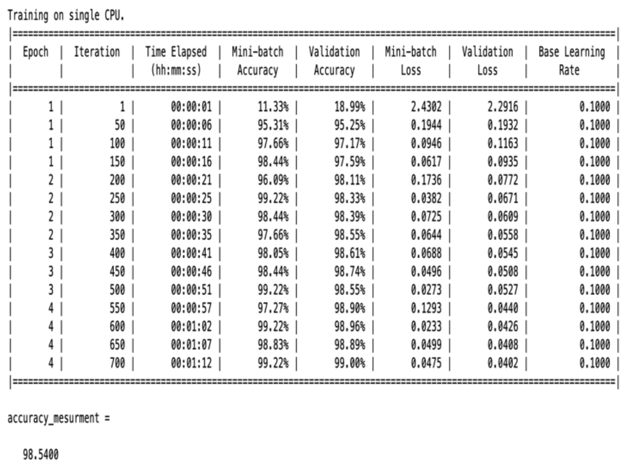
Other Information

Hardware resource:	Single CPU
Learning rate schedule:	Piecewise
Learning rate:	0.1

Learn more

Accuracy
- Training (smoothed)
- Training
- Validation

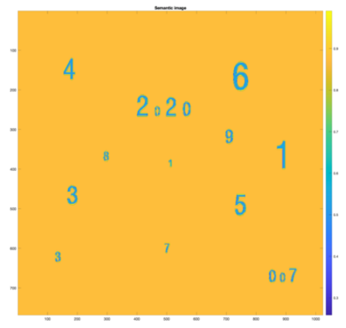
Loss
- Training (smoothed)
- Training
- Validation



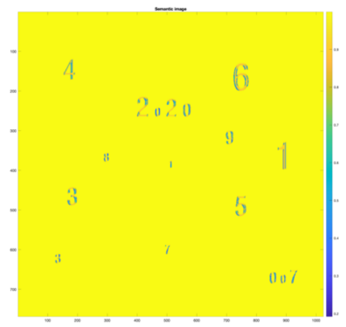
```
Training on single CPU.
|==========================================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Validation  |  Mini-batch  |  Validation  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |   Accuracy   |     Loss     |     Loss     |      Rate       |
|==========================================================================================================|
|       1 |           1 |       00:00:01 |       11.33% |       18.99% |       2.4302 |       2.2916 |          0.1000 |
|       1 |          50 |       00:00:06 |       95.31% |       95.25% |       0.1944 |       0.1932 |          0.1000 |
|       1 |         100 |       00:00:11 |       97.66% |       97.17% |       0.0946 |       0.1163 |          0.1000 |
|       1 |         150 |       00:00:16 |       98.44% |       97.59% |       0.0617 |       0.0935 |          0.1000 |
|       2 |         200 |       00:00:21 |       96.09% |       98.11% |       0.1736 |       0.0772 |          0.1000 |
|       2 |         250 |       00:00:25 |       99.22% |       98.33% |       0.0382 |       0.0671 |          0.1000 |
|       2 |         300 |       00:00:30 |       98.44% |       98.39% |       0.0725 |       0.0609 |          0.1000 |
|       2 |         350 |       00:00:35 |       97.66% |       98.55% |       0.0644 |       0.0558 |          0.1000 |
|       3 |         400 |       00:00:41 |       98.05% |       98.61% |       0.0688 |       0.0545 |          0.1000 |
|       3 |         450 |       00:00:46 |       98.44% |       98.74% |       0.0496 |       0.0508 |          0.1000 |
|       3 |         500 |       00:00:51 |       99.22% |       98.55% |       0.0273 |       0.0527 |          0.1000 |
|       4 |         550 |       00:00:57 |       97.27% |       98.90% |       0.1293 |       0.0440 |          0.1000 |
|       4 |         600 |       00:01:02 |       99.22% |       98.96% |       0.0233 |       0.0426 |          0.1000 |
|       4 |         650 |       00:01:07 |       98.83% |       98.89% |       0.0499 |       0.0408 |          0.1000 |
|       4 |         700 |       00:01:12 |       99.22% |       99.00% |       0.0475 |       0.0402 |          0.1000 |
|==========================================================================================================|

accuracy_mesurment =

   98.5400
```

The Accuracy of measurement after 4 epochs equal to 98.54%.

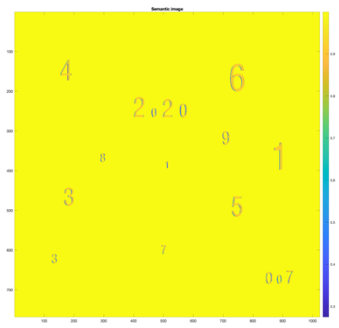
The evaluation stopped after training the FCN get good result and the result does not came much better. That was after 4 epochs.



The semantic segmentation image



The semantic segmentation image size 4



The semantic segmentation image size 8

The image after non-max size 8

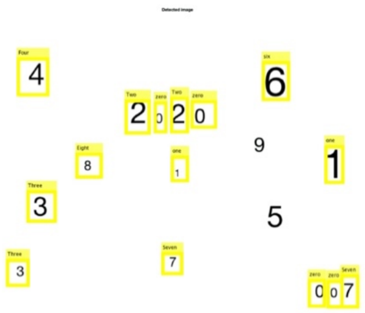
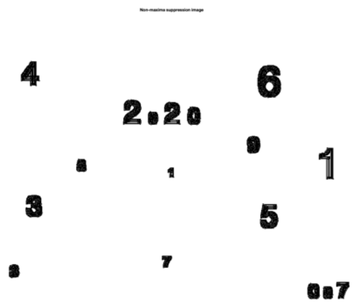
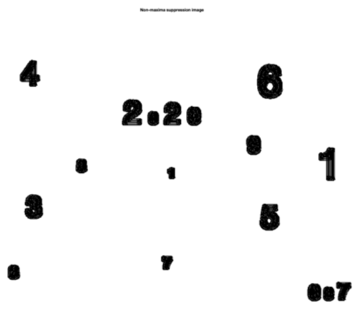


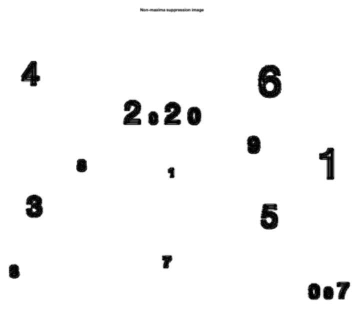
Image after detect the right digit.



The image after non-max



The image after non-max size 4



Theoretical part

A - full conventional network is exactly the same as CNN, but not all layers are connected to each other that will allow us to use it in a full image in different sizes. Let assume that the parameter in this case will apply on CNN so that mean at convolution layer there is 5x5 convolutions with 20 channels, therefor the filtering operation will follow the role:

$$\boxed{\omega_0 + (I \times \omega)(x, y) = 5 \times 5 \times 20 + 1 = 501}$$

Then will apply to every pixel in the image at the first stage. That will tack a much time and math computation process after that at fully connected layer will calculating the multiplication between the output from max pool layers and weight matrix and then add a bias separately for every pixel to get segmentation (goes through its own backpropagation process to determine the most accurate weights and better label to the image) that mean a huge resource (more than one multicore possessor with good machine) and calculation process and a long time until the result is presented for us. Hence, in case of bigger image size and leek of time and resource we will apply fully convolutional network which will apply the calculations once on full image. FCN it faster and gives ability to pass to the network images of different shapes, so FCN more efficient.

B - To convert a fully connected layer to a fully convolution layer, the size of the filter in a fully connected layer should be the same size in the input image. In these cases, a fully connected layer will have the same convolution filter in the fully convolution layer and the output image will be almost the same size as the input image.

C - Scale Invariant Feature Transform (SIFT) creates scale space representation and then detects key points are the local extrema of this representation. This is builds up robustness against scale changes. SIFT builds a scale space representation based on Difference-of-Gaussian (DoG) responses. In order to construct the space, it iteratively applies a Gaussian filtering. This results in octaves, as a set of increasing blurred images. Then, by subtracting adjacent images in this series, derives a set of DoG responses, which is SIFT's scale space representation. A low-order recursive filter, with a symmetric denominator, is used in constructing an efficient scale space.

Using SIFT to build a representation for scale space then from this representation can be took key points to create a model more staple against changing in scale based on Difference of Gaussian responses (DoG). In order to construct that scale, repeat DoG operation many times, the result will be a group of images increasingly blurred. After subtracting the adjacent image from that group of images will give use DoG representation and that is the scale space representation of SIFT