Project I: Digit Recognition

Haitham Babbili

Chalmers University

27 March 2020

Abstract

The aim of this project is to increase the understanding of machine learning and convolution neural network and how to design CNN 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 CNN to detect a digit in an image. Therefore, a fully convolutional neural network used to estimate the digit and non-max suppression to get the greatest approach to the correct digit.

1. Method

The MNIST handwritten digit database was chosen to train our CNN. 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 CNN. Reshape the image data into 4D matrix 28x28x1x60000 (1 is because of greyscale) and convert labels data into categorical to be used on CNN. In this way, all images have the same size, so we avoided multi-scale processing of the image. This image will be the input of CNN. CNN consists of layers; each layer has a number of nodes connected to nodes in the next layer with weight wk while the next node has initial value call bias w0. So, the input value multiplied with a weight wk and add to bias then feeded to active function (sigmoid)

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

This probability used to calculate the log-likelihood loss in every node at every layer. But on CNN it contains a huge amount of weights and to train the 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.

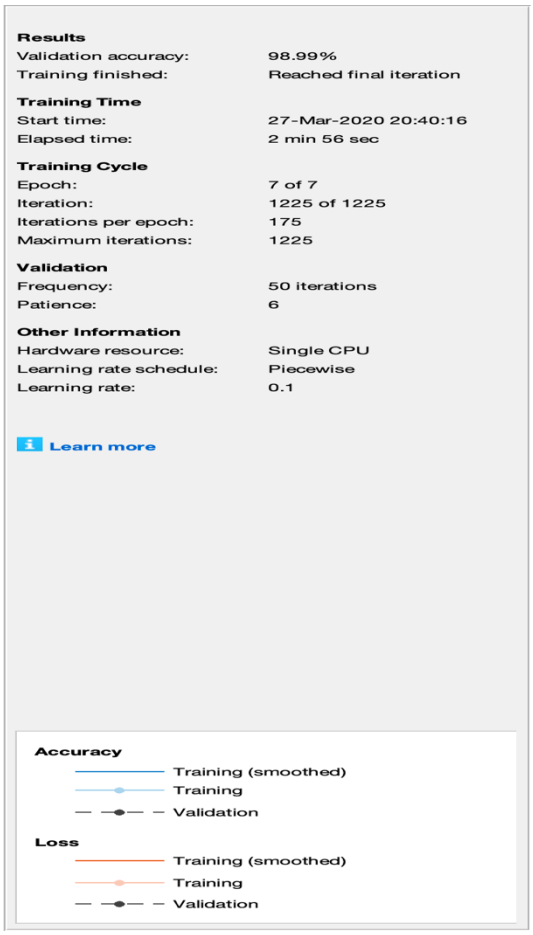
The next layer is the convolution Layer which apply 20 5x5 sliding convolution filters and calculates the multiplication between the input and weight the adding a bias term of the weights and the input, and then adding a bias term. Then batchNormalizationLayer use to normalize the output of the convolution 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 next three layers are repeating the first three-layer, but with a smaller number of filters and fewer dimensions. Last but not least fullyConnectedLayer (10) which act as a linear combination of all output by calculating the multiplication between the output from prevision layers and weight matrix and then add a bias. 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.

2. Experimental Evaluation

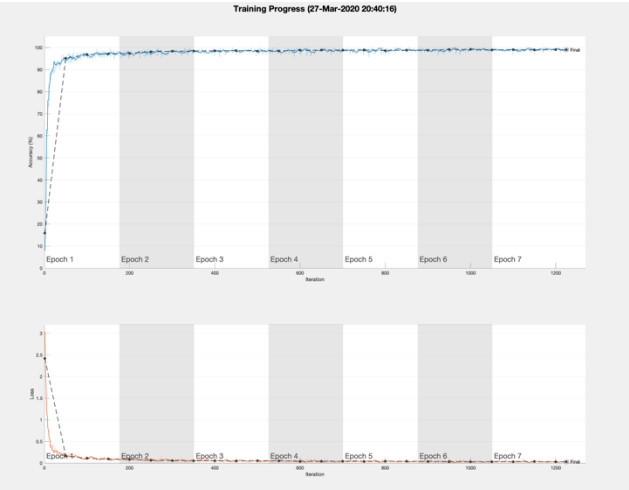
The parameter used for training the CNN

- the input image layer [28x28x1]
- first convolution2dLayer (5,20)
  size of slide window 5x5x20 and learn parameter (5x5)x20+20= 520
  Where 20 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 less number of filter 3x3x10 so learning parameter will be (3x3x8)x10+10 = 730
- second max pool(2,2)
- fullyConnectedLayer(10) and learn parameter 4x4x10x10= 1610
- total learning parameter = 520+730+1610 = 2860

After training and testing the accuracy of the network by using training and testing dataset from MNIST. mat. The image provided in the canvas was implemented on the trained network by using classify () function to get the right digit in the image. The classify function use two-parameter, one is a net which is the training CNN result and test_image function. The test_image.m function is processed to transfer the image to greyscale then scale it to fit the MNIST dataset format which we train our network on it after that separate the digit from the background and draw a box around that digit and then resize it again to fit MNIST dataset. In the end, finding the field in the tested image by compared to the labelled test set, then finding where the prediction results equal to zero.

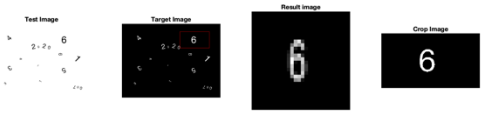


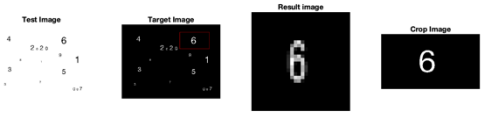
The figure below shows training and validation loss.

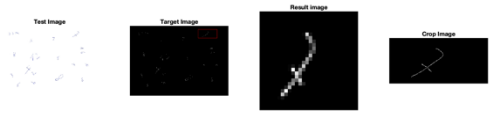


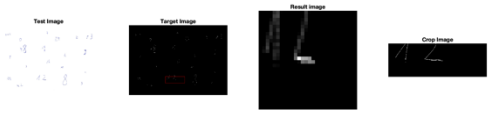

```
Training on single CPU.
Initializing input data normalization.
|========================================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Validation  |  Mini-batch  |  Validation  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |   Accuracy   |     Loss     |     Loss     |      Rate       |
|========================================================================================================|
|       1 |           1 |       00:00:02 |       7.81% |      15.88% |       3.0420 |       2.4153 |          0.1000 |
|       1 |          50 |       00:00:10 |      92.97% |      95.17% |       0.1757 |       0.1658 |          0.1000 |
|       1 |         100 |       00:00:17 |      95.31% |      96.78% |       0.1712 |       0.1099 |          0.1000 |
|       1 |         150 |       00:00:25 |      97.27% |      97.13% |       0.1054 |       0.0952 |          0.1000 |
|       2 |         200 |       00:00:32 |      95.70% |      97.33% |       0.1004 |       0.0834 |          0.1000 |
|       2 |         250 |       00:00:40 |      97.66% |      98.10% |       0.0864 |       0.0629 |          0.1000 |
|       2 |         300 |       00:00:47 |      98.44% |      98.30% |       0.0610 |       0.0579 |          0.1000 |
|       2 |         350 |       00:00:55 |      99.22% |      98.45% |       0.0462 |       0.0538 |          0.1000 |
|       3 |         400 |       00:01:02 |      98.83% |      98.54% |       0.0470 |       0.0507 |          0.1000 |
|       3 |         450 |       00:01:10 |      96.48% |      98.59% |       0.1453 |       0.0478 |          0.1000 |
|       3 |         500 |       00:01:17 |      98.05% |      98.43% |       0.0496 |       0.0545 |          0.1000 |
|       4 |         550 |       00:01:24 |      97.66% |      98.43% |       0.0498 |       0.0517 |          0.1000 |
|       4 |         600 |       00:01:31 |      98.44% |      98.79% |       0.0592 |       0.0388 |          0.1000 |
|       4 |         650 |       00:01:38 |      98.44% |      98.82% |       0.0617 |       0.0399 |          0.1000 |
|       4 |         700 |       00:01:45 |      98.83% |      98.81% |       0.0360 |       0.0401 |          0.1000 |
|       5 |         750 |       00:01:53 |      99.22% |      98.91% |       0.0269 |       0.0365 |          0.1000 |
|       5 |         800 |       00:02:00 |      96.48% |      98.67% |       0.0956 |       0.0399 |          0.1000 |
|       5 |         850 |       00:02:07 |      98.83% |      98.81% |       0.0364 |       0.0410 |          0.1000 |
|       6 |         900 |       00:02:14 |      98.44% |      98.76% |       0.0329 |       0.0385 |          0.1000 |
|       6 |         950 |       00:02:21 |      98.05% |      99.13% |       0.0657 |       0.0301 |          0.1000 |
|       6 |        1000 |       00:02:28 |      98.83% |      99.13% |       0.0514 |       0.0303 |          0.1000 |
|       6 |        1050 |       00:02:36 |      99.22% |      98.98% |       0.0277 |       0.0338 |          0.1000 |
|       7 |        1100 |       00:02:43 |      98.44% |      99.03% |       0.0280 |       0.0296 |          0.1000 |
|       7 |        1150 |       00:02:50 |      96.88% |      98.90% |       0.0877 |       0.0340 |          0.1000 |
|       7 |        1200 |       00:02:57 |      99.22% |      99.09% |       0.0214 |       0.0308 |          0.1000 |
|       7 |        1225 |       00:03:01 |      98.83% |      98.99% |       0.0249 |       0.0316 |          0.1000 |
|========================================================================================================|

accuracy_mesurment =

   98.6900
```

the Accuracy of measurement after 7 epochs equal to 98.69%.









3. 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:

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

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 - Using SIFT to build a representation for scale-space, then from this representation can take key points to create a model more stable against changing in scale based on the 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 of DoG representation and that is the scale-space representation of SIFT.

A scale space can be constructed efficiently by using a low-order recursive filter with a symmetric denominator.