Deep Learning Lab Report

Haocheng An

December 2, 2017

1 Introduction

Deep Learning is a very important tool in machine learning. By using several layers of convolutional neural networks(CNN), we reduce the dimension of a picture to a single short vector. Then we create multiplie fully connected layer so that not all informations about the pictuers are lost. By referring this vector, we can tell what class does the image belongs to. In this lab, I investigate the effect of deep learning network on classifying the picture given, where each reflects a digit from 0 to 9. I use tensorflow to investigate such effects. During the implementation part, I use to the starter code given in CS 342 offered in UT Austin website as a reference.[3]

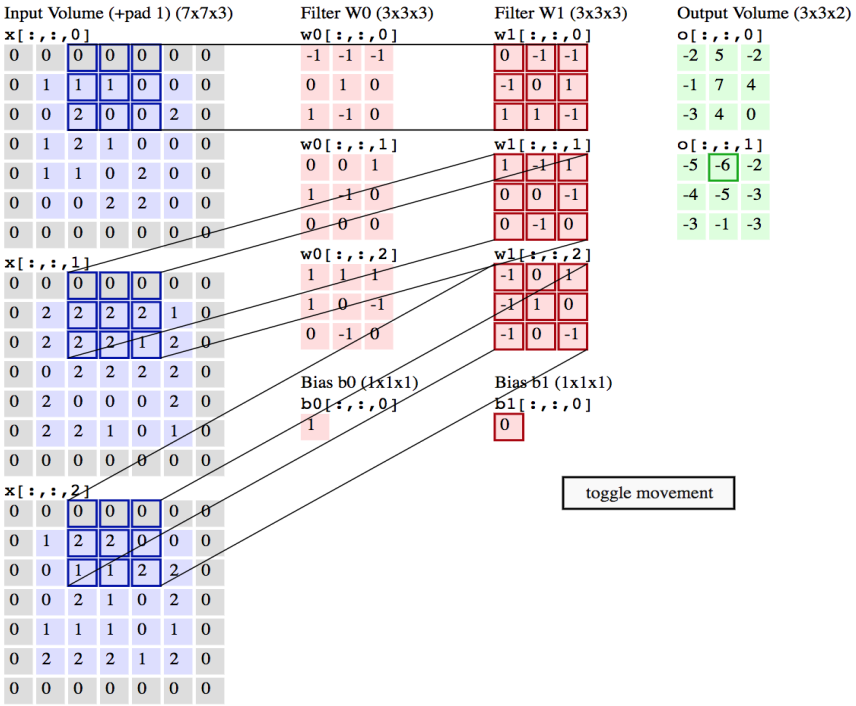
2 Methodology

2.1 Read In Images

First of all, we need to read in images. The images were given in *.mat* format and it basically stores a dictionary. Except for metadata of this file, there are 4 keys: trainimages, trainlabels, testimages and testlabels. I use the built-in library "scipy.io" to read it in. The first two are read in at the start of the program as the training set. The last two are read in at the end of the program as the testing set.

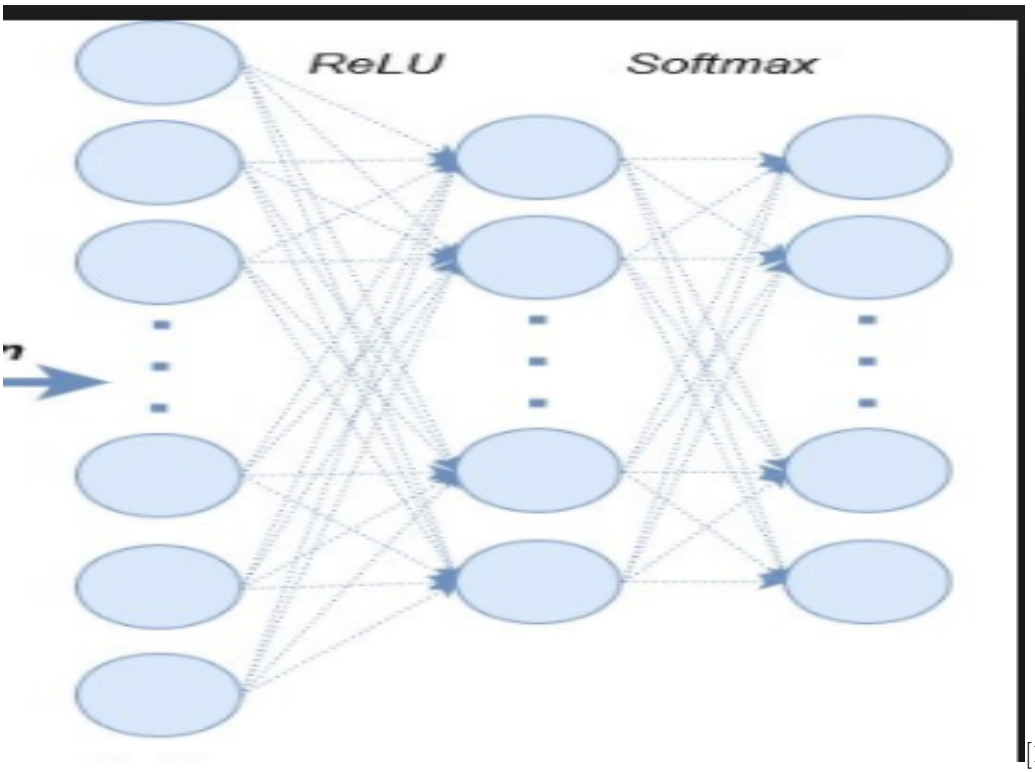
2.2 Implement Neural Networks

The idea of the CNN part of the networks can be described as graph shown below



[2]

After getting the images we need, I center the image by substract mean and divide it by its standard deviation. Then, I implement the neural networks. The implementation uses Google neural networks' libarary tensorflow. First, we have 5 convolutional layers. For each layer, we take the previous layer as the input. Then, we specify some kernel size,(for this lab, I specify as (4,4)) some filter numbers,(it can be varied and gives us different models) and some stride(I set it to be 2 for every layer) with paddings if necessary by calling $tensorflow$ library function. $tf.contrib.layers.conv2d$[4] For the specific parameters, I will discuss in the result part. Then I have a fully connected layer with output of $num_output = 10$ as we have 10 different classes of possible input. An visualization of fully connected layer can be seen as below



[1]

2.3 Optimization

After getting the output tensor, we compute the softmax cross entropy difference between the actual value (*label*) and the output value we get (*convolution network*). In order to do this, we call $tf.nn.sparsesoftmaxcrossentropywithlogits$[5]. Later, we set learning rate and the vector parameters to reduce random errors by using momentum optimizer by setting the learning rate to be 0.0001 and momentum to be 0.9.

2.4 Training Data

I train the model for 20 iterations. For each time, it randomly shuffles the pictures we have and take out 32 of them for training. The latter iteration is built on the previous one.

2.5 Testing Model

After train the model, I put in all the 10000 images with label testing data for testing to see how the accuracy is.

3 Result

Let us examine effects of differences on *numoutputs* can have on our accuracy. The *numoutputs* reflect how many filters I have. In order to do so, we need to keep other parameters unchanged. To be specific, I set $kernelsize = [4, 4]$,

which means I only operates on $4x4$ kernel once at a time. stride = 2, it means every time we jump 2 pixels to the next. *padding* = *same*, which is same to what described in the big picture described above. In this case, we set it to be 0 to expand pictures so that I can operates on the border case. I consider three different models, monotone increasing outputs, monotone decreasing outputs and decrease first then increase. They give us different result as shown below.

```
conv1 = tf.contrib.layers.conv2d(
  inputs=white_inputs, num_outputs=21,kernel_size=[4, 4],stride=2,padding="same",scope='conv1')

conv2 = tf.contrib.layers.conv2d(
  inputs=conv1,num_outputs=18,kernel_size=[4, 4],stride=2,padding="same",scope='conv2')

conv3 = tf.contrib.layers.conv2d(
  inputs=conv2,num_outputs=13,kernel_size=[4, 4],stride=2,padding="same",scope='conv3')

conv4 = tf.contrib.layers.conv2d(
  inputs=conv3, num_outputs=16,kernel_size=[4, 4],stride=2,padding="same",scope='conv4')

conv5 = tf.contrib.layers.conv2d(
  inputs=conv4,num_outputs=20,kernel_size=[4, 4],stride=2,padding="same",scope='conv5')
```

```
Convnet
[   0] Accuracy:  0.736       Loss:  0.786
[   1] Accuracy:  0.870       Loss:  0.351
[   2] Accuracy:  0.880       Loss:  0.311
[   3] Accuracy:  0.883       Loss:  0.295
[   4] Accuracy:  0.886       Loss:  0.284
[   5] Accuracy:  0.888       Loss:  0.276
[   6] Accuracy:  0.890       Loss:  0.269
[   7] Accuracy:  0.892       Loss:  0.265
[   8] Accuracy:  0.892       Loss:  0.262
[   9] Accuracy:  0.894       Loss:  0.257
[  10] Accuracy:  0.894       Loss:  0.254
[  11] Accuracy:  0.895       Loss:  0.252
[  12] Accuracy:  0.896       Loss:  0.248
[  13] Accuracy:  0.896       Loss:  0.249
[  14] Accuracy:  0.896       Loss:  0.246
[  15] Accuracy:  0.897       Loss:  0.244
[  16] Accuracy:  0.897       Loss:  0.244
[  17] Accuracy:  0.897       Loss:  0.242
[  18] Accuracy:  0.898       Loss:  0.240
[  19] Accuracy:  0.898       Loss:  0.239
<IPython.core.display.HTML object>
(28, 28, 1)
Input shape: (10000, 28, 28, 1)
Labels shape: (10000,)
ConvNet Easy Validation Accuracy:  0.8906
```

For monotone part, the network setup is similar to the screen shot given above. The monotone decreasing part, I set numoutputs for each layer to be 21,18,15,12,9 respectively.

```
Convnet
[   0] Accuracy: 0.533      Loss: 1.320
[   1] Accuracy: 0.675      Loss: 0.840
[   2] Accuracy: 0.683      Loss: 0.788
[   3] Accuracy: 0.687      Loss: 0.766
[   4] Accuracy: 0.689      Loss: 0.755
[   5] Accuracy: 0.691      Loss: 0.744
[   6] Accuracy: 0.692      Loss: 0.739
[   7] Accuracy: 0.693      Loss: 0.733
[   8] Accuracy: 0.693      Loss: 0.730
[   9] Accuracy: 0.694      Loss: 0.724
[  10] Accuracy: 0.695      Loss: 0.721
[  11] Accuracy: 0.695      Loss: 0.720
[  12] Accuracy: 0.695      Loss: 0.717
[  13] Accuracy: 0.695      Loss: 0.717
[  14] Accuracy: 0.696      Loss: 0.714
[  15] Accuracy: 0.696      Loss: 0.713
[  16] Accuracy: 0.696      Loss: 0.712
[  17] Accuracy: 0.696      Loss: 0.711
[  18] Accuracy: 0.697      Loss: 0.708
[  19] Accuracy: 0.696      Loss: 0.709
<IPython.core.display.HTML object>
(28, 28, 1)
Input shape: (10000, 28, 28, 1)
Labels shape: (10000,)
ConvNet Easy Validation Accuracy:  0.6939
```

the monotone increasing part, I reverse them and make it 9,12,15,18,21. The performance is shown below:

```
Convnet
[   0] Accuracy: 0.644      Loss: 0.972
[   1] Accuracy: 0.757      Loss: 0.613
[   2] Accuracy: 0.766      Loss: 0.574
[   3] Accuracy: 0.771      Loss: 0.556
[   4] Accuracy: 0.773      Loss: 0.547
[   5] Accuracy: 0.775      Loss: 0.539
[   6] Accuracy: 0.776      Loss: 0.533
[   7] Accuracy: 0.778      Loss: 0.529
[   8] Accuracy: 0.779      Loss: 0.525
[   9] Accuracy: 0.780      Loss: 0.520
[  10] Accuracy: 0.780      Loss: 0.518
[  11] Accuracy: 0.780      Loss: 0.517
[  12] Accuracy: 0.782      Loss: 0.513
[  13] Accuracy: 0.782      Loss: 0.512
[  14] Accuracy: 0.782      Loss: 0.510
[  15] Accuracy: 0.782      Loss: 0.509
[  16] Accuracy: 0.783      Loss: 0.508
[  17] Accuracy: 0.783      Loss: 0.507
[  18] Accuracy: 0.784      Loss: 0.505
[  19] Accuracy: 0.784      Loss: 0.504
<IPython.core.display.HTML object>
(28, 28, 1)
Input shape: (10000, 28, 28, 1)
Labels shape: (10000,)
ConvNet Easy Validation Accuracy:  0.7718
```

4 Conclusion

As our goal is to classify the digit, the higher the accuracy, the better the model. With the result shown above, we can get a preliminary conclusion that the number of output with decreasing first and then increasing works best while the monotone increasing works second and monotone decreasing one works the worst. This only is a preliminary result. Many different factors still need to take into consideration. As every iteration we pick different images from the traning set randomly, it is possible that the model stuck in a small interval given its small learning rate. Also, these handwritting pictures may not the most representable one and the testing picture may contain some new feature that the training image does not has. Even the test image gives us high accuracy, we still can't guarantee it works best as the typicalness of those pictures are highly susceptible.

References

[1] Andy *Convolutional Neural Networks Tutorial in TensorFlow* *http://adventuresinmachinelearning.com/convolutional-neural-networks-tutorial-tensorflow/*

[2] Justin Johnson *Convolutional Neural Networks (CNNs / ConvNets)* *http://cs231n.github.io/convolutional-networks/*

[3] Philipp Krähenbühl *CS 342 - Homework 4* *http://www.philkr.net/cs342/homework/04/*

[4] Tensorflow. *tf.contrib.layers.conv2d.* [*https://www.tensorflow.org/apidocs/python/tf/contrib/layers/conv2*
, 2017.

[5] Tensorflow. $tf.nn.sparse_softmax_cross_entropy_with_logits.$ [$https : //www.tensorflow.org/api_docs/python/tf/nn/sparse_softmax_cross_entropy_with_logits$].
, 2017.