Machine Learning Engineer Nanodegree Capstone

Digit-Level House Number Recognition

Using Convolutional Neural Network

Peter Lai
October 31st, 2016

I. Definition

Overview

Vision-based recognition system is fundamental in building modern AI agent. Information in pixel is far more abundant than any other kinds of carriers. For example, a capture of windshield view in a self driving system could contain informations like whether the car is in the right line, the location of the obstacle, traffic sign and a lot more simultaneously. We are now capable of doing object detection, but a modern AI agent should also be able to tell what exactly an object is and what it means.

To build a intelligent visual cognitive system, one of the most fundamental is natural text recognition. Recognizing text from natural pictures, frames captured from videos or handwritten material is a hard task. However, we can begin with recognizing a part of the text. In 1998, LeCun et al.(Courant Institue, NYU) tried different models to recognize the MNIST handwritten digits from 0 to 9 and achieved 1.6% error rate at best. In 2014, Ian J. Goodfellow et al.(Street View and reCAPTCHA Team, Google) applied a deep convolutional neural network to recognize multi-digit house numbers. Instead of separating the door plate in advance, the team put 5 softmax classifiers over the convolutional feature map. The final model achieved accuracy of 96.03% on sequence transcription. They also get a character-level accuracy of 97.84% percent.

In this project I will use the Street View House Number or SVHN datasets to build a natural digit recognition model using convolutional neural network. House number recognition is essential in automatic map making and intelligent travel technologies like self driving car. It can also be seen as an introduction to word-level text recognition and interpretations.

The data used in this project are images of cropped house number digits captured in daily street view. Detailed information on the datasets will be discussed in part II.

Problem Statement

1. Problem Definition

Fig. 1

This project will focus on the single digit recognition. Given a cropped part of a door plate, the model should be able to recognize what number it is (from 0-9). For example, for a cropped image like **Fig. 1**, the model will return 3 because 3 is center aligned in the image.

This is a multi-class classification problem. In this kind of problem, we want a vector of probability on the output for a class label. For example, for the digit '1', the output should looks like
`[p0, p1, p2, p3, p4, p5, p6, p7, p8, p9]`
where p1 is the largest among all the other elements in the vector.

The model can be represented as

$$d = \text{argmax } F(X)$$

Given a image X, model F should return a vector of probability, the prediction digit d is the index where give the highest probability.

2. Solution Definition

In this recognition task, the goal is to get as many correct predictions as possible on the test set, in other words, maximize the accuracy on the test set.

3. Strategy

The strategy I use in this problem is to train a convolotional neural network on a large dataset of images. A further discussion on convolutional neural network is in Algorithms and Techniques in part II of this report.

An architecture worth trying is LeNet-5, the architecture is shown in **Fig. 2**. However, the SVHN data is more complicated, therefore increasing the convolutional layers' depth might be necessary. In addition, the model's output layer should be modified to be a softmax then argmax layer.

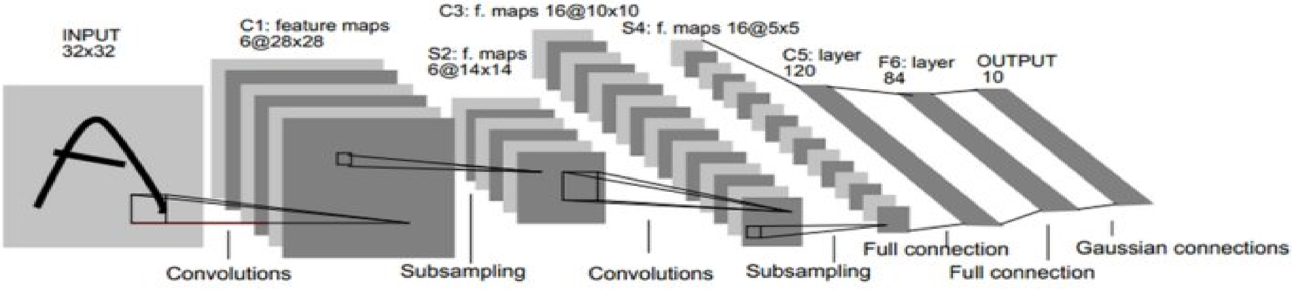


Fig. 2

Metrics

1. **Accuracy** (to maximize), is the ratio of how many labels the model correctly recognizes in the dataset.

2. **Precision** and **Recall** (to maximize), will be presented in a confusion matrix. For predictions pointing to a certain class, precision is the ratio of how many images that truly belong to this class. For images belong to a certain class, recall is the ratio of how many images that are successfully recognized. The confusion matrix for the problem looks like **Table 1.** In convenience, I just show the precision and recall for 0 and 3.

Prediction / Class	0	1	2	3	4	5	6	7	8	9	Recall
0	200	3	8	3	2	12	1	3	41	3	0.72
1	3	*	*	2	*	*	*	*	*	*	*
2	5	*	*	1	*	*	*	*	*	*	*
3	6	5	2	130	4	4	2	2	1	2	0.82
4	8	*	*	4	*	*	*	*	*	*	*
5	3	*	*	6	*	*	*	*	*	*	*
6	0	*	*	7	*	*	*	*	*	*	*
7	2	*	*	0	*	*	*	*	*	*	*
8	1	*	*	1	*	*	*	*	*	*	*
9	3	*	*	3	*	*	*	*	*	*	*
Precision	0.87	*	*	0.83	*	*	*	*	*	*	

Table 1

3. **Cross Entropy**, also known as Softmax loss (to minimize) the loss function used in training, measuring how close the model is from the ideal model. The cross entropy can be represented as :

$$\text{Cross\_Entropy} = (\Sigma_i \ln(e^{F(x)(y)}/\Sigma e^{F(x)}))/n$$

where i is the index for images, n is the number of images and F(x)(y) is the probability for the image X to belong to class y.
The cross entropy is not the a metric in evaluate the model. It is the method of calculating the loss in training.

II. Analysis

Data Exploration and Visualization

1. **Data Format :**
Data was downloaded from http://ufldl.stanford.edu/housenumbers/. The website has two categories of data. The dataset used in this project was stored in .mat files, named as "train.mat", "test.mat" and "extra.mat". Two ndarrays will be returned by reading each file using Scipy.io.loadmat() function.

The first ndarray one (name it X) stands for the images, the second (name it y) contains the class labels correspond to X with 0 marked as 10 (MATLAB style).
X is of shape :
`(pixel width, pixel height, number of channels, number of items)`
y is of shape :
`(number of items, )`

Table 2 shows the number of images in each file.

File Name	train_32x32.mat	test_32x32.mat	extra_32x32.mat
Number of Record	73257	26032	531131

Table 2

The images in this dataset is in same size 32x32x3, where 3 represent RGB channels.

2. **Distribution :**
In the Jupiter notebook `model/preprocess_mat.ipynb`, the first few cells will explore the data and plot the distributions of the class labels in all three datasets using function `peek_distribution` in `model/display_center.py`

See **Fig. 3.1, 3.2, 3.3** :

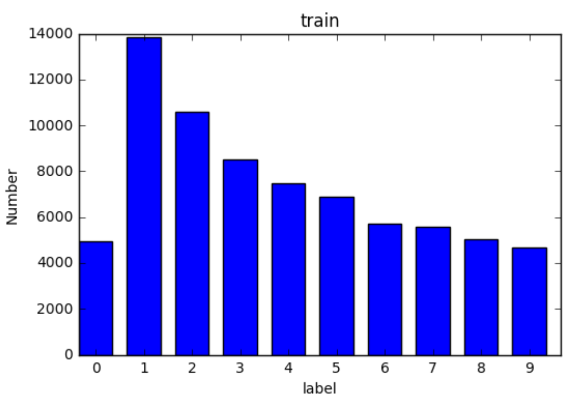


Fig. 3.1

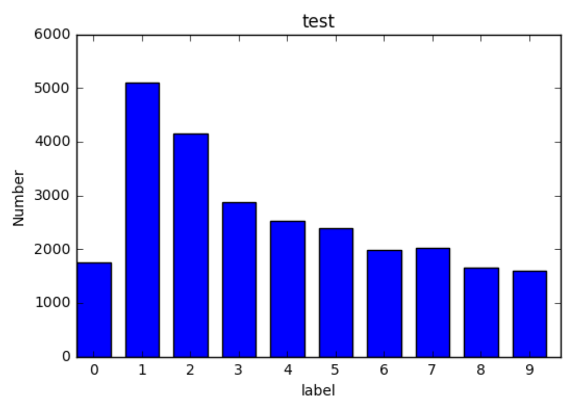


Fig. 3.2

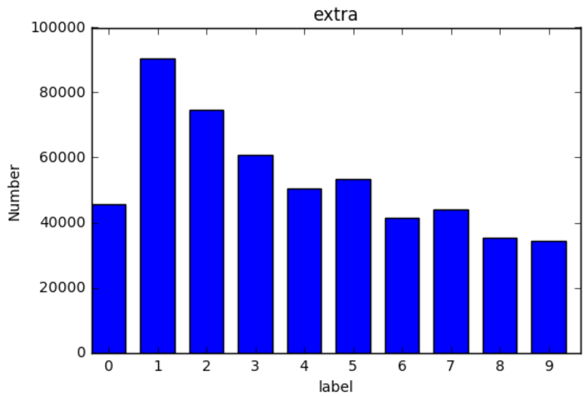


Fig. 3.3

Algorithms and Techniques

Frameworks and Tools

- The core scripts are presented in Jupyter Notebook using python.
- Scientific Computation & Visualization : Numpy, Matplotlib.
- scikit-learn: A machine learning library that implements classic machine learning algorithms.
- Tensorflow: This is a framework that help users to build computational graph with various kinds of nodes that have functionality of both forward and backward propagating computation, which make it easier to construct deep learning architecture.

Mathematics

- Convolution Neural Network

a. Feature Extractor

Convolution Neural Networks(Fukushima, 1980; LeCun et al., 1988) are neural networks that contain convolution layer. Intuitively, convolution layers are layers that divides a image into small pieces by a windows which moves both horizontally and vertically by a certain stride of pixels. Each piece is seen as a feature, for example, the sharp head of the letter 'A'. Those features, as group of pixel values, will then be dot product with a group of weight and become a value for the next layer.

Mathematically, convolution here means covering a m x m weight matrix, or kernel on a m x m area in a n x n matrix of pixel values and doing dot product between the two. The dot product values will then feed as new 'pixels' of new 'images' and become the input of the next layer. Usually, there are multiple kernels for one layers so that we create structures with multi-layers that looks like RGB channels, as shown in **Fig.4**. As a result of the process, neurons in convolutional layers are arranged in 3 dimensions : width, height and depth, as shown in **Fig. 5**. Instead of flattening a image into a single array of pixel, convolution maintains the information of relative position of pixels.

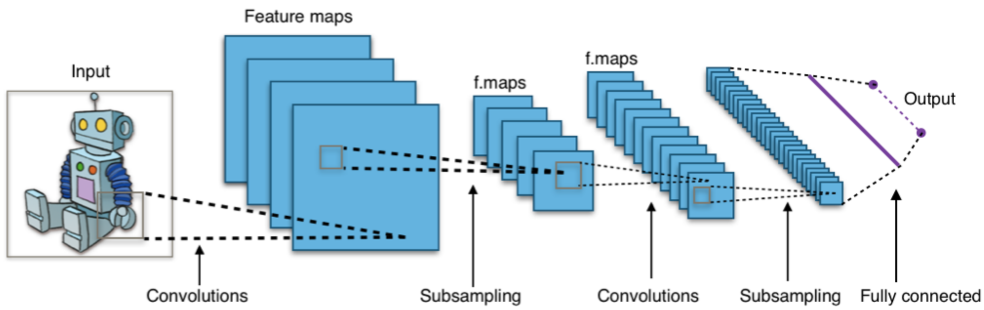


Fig. 4
source: https://en.wikipedia.org/wiki/Convolutional_neural_network#/media/File:Typical_cnn.png
Feature extracting : A small piece of the image become a pixel value for the next layer by dot producting with the kernel.

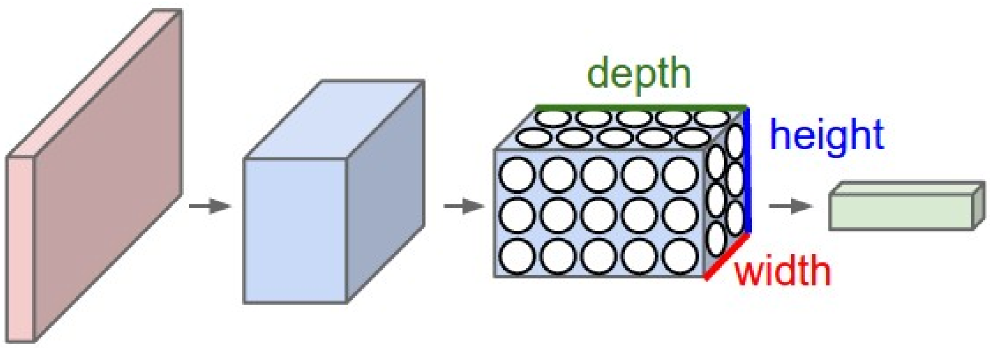


Fig. 5
source: http://cs231n.github.io/convolutional-networks/

b. Activation Function

Activation is an important step for neural network. It should stop the propagation when a neuron is not activated, which is simply whether the numbers is positive. There are lots of activation functions, like step, sigmoid and tanh, but most of them have various problems. In practice, people use rectified linear unit, ReLU right behind convolution. This unit or its improved version are popular because it's easy to both forward-propagate and back-propagate. See **Fig. 6,** the blue line represents the rectified linear unit.

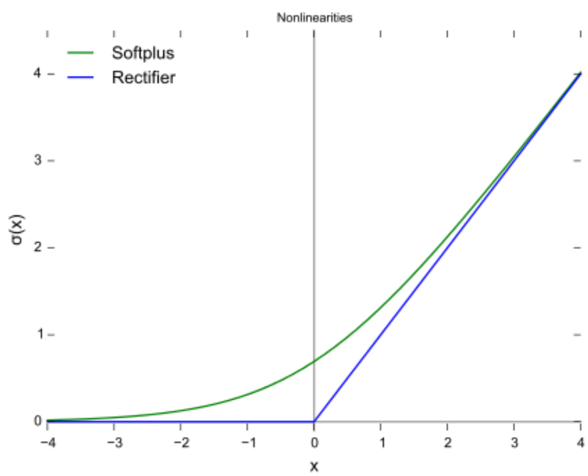


Fig. 6
https://en.wikipedia.org/wiki/Rectifier_(neural_networks)

c. Pooling

One disadvantage of convolutioning is computationally expensive because it takes time for the kernels to extract features and more space to store the parameters. Pooling is a way to reduce the amount of parameters and computation along the process. The idea looks pretty like convolution, but instead pooling take the max or average of the kernel-covered area of the image, as shown in **Fig. 7.**

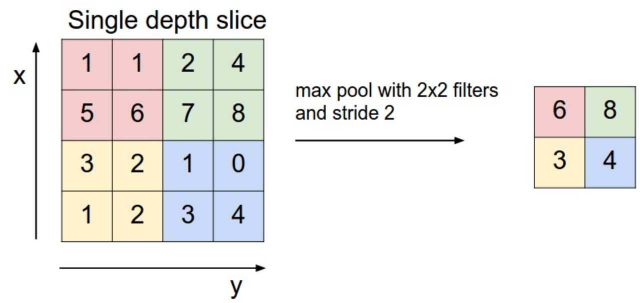


Fig 7

source: http://cs231n.github.io/convolutional-networks/#pool

d. Fully connected Layers

It's common to connect fully connected layers before we compute the final vector of probability. The most intuitive reason is that feature maps is not probability values and the shape of the convolutional output is not convenient to deal with. As shown in **Fig. 8**, fully connected layers take a 1 dimensional vector and do matrix multiplication with a matrix of weights. The process produces neurons of numerical values instead of feature maps. It also do a dimension transformation so it is applied to compute the output vector of probability.

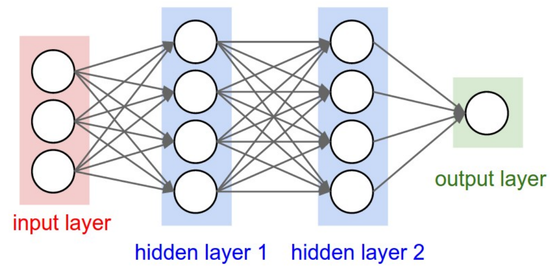


Fig. 8

source: http://cs231n.github.io/convolutional-networks/#pool

e. Connecting to Fully Connected Layers

Convolution layers gives output in image like shape. Before taking it as the input to create one hot encoding vector, they should be flattened into stripe like vectors.

- Training

Training a deep neural network with convolution layers take more techniques than traditional fully connected neural networks. For me there are two basic ideas should be kept in mind: first is to make the information flow efficiently, second is when you are trying to go to the global minimum, adjust your pace according to the terrain.

a. Initialization of Weights

The rule "make it random small" no more applies to deep neural networks. This is very intuitive: if you make all the weights random small, when the information, or numbers are back-propagating, the minus power is accumulating, therefore the numbers could become zero when it arrive some deep neurons and stop the back-propagation, which means the upstream layers will never be trained. In this project, I simply increase the standard deviation of the distribution from upstream to downstream to avoid the issue.

b. Dropout

Dropout is to block the numbers flowing forward and backword in some neuron randomly. It can be seen as an ensemble technique, because in every step, only a part of the network is trained. However, dropout is not implemented in validation and test. It's the equivalence to combine those partial model. And most of the cases, this technique will give more accuracy.

c. Stochastic Gradient Descent

When it comes to large dataset, it would be too heavy for computer to use all the data in one iteration. One solution is to use Stochastic Gradient Descent **** cite **** Basically, we take a mini batch of the dataset and feed it to the network. It may take more iteration for the model to converge, but in works in practice.

d. Update and Optimizer

Basically, the update space is a bowl shape with steep and smooth directions. In this project, I applied Adam update(Kingma and Ba, 2014), which make the update more on the steep direction.

Adam optimization can be represented as follow : (in python)

```
m = beta1*m + (1-beta1)*dx
v = beta2*v + (1-beta2)*(dx**2)
x += - learning_rate * m / (np.sqrt(v) + eps)
```

source : http://cs231n.github.io/neural-networks-3/#ada

Detailed discussion on the mathematics behind the formula would be long, so it's skipped.

- Logging and Visualization

In experiment, I simply use print function to dynamically see the numerical value of the loss and the accuracy.

When the model is satisfactory, I used Tensorflow built-in TensorBoard to visualize the architecture and the tracing the loss and accuracy.

III. Methodology

Data Preprocessing

I took roughly four steps to preprocess the data in `model/preprocess_mat.ipynb`:

1. Reshape the Data

The data is not friendly for Numpy, so I just made the index of items the first dimension.

The shape after this step:

```
(number of items, pixel width, pixel height, number of channels)
(number of items, 32, 32, 3)
```

2. Transfer the Images in Grayscale

Since the task is to recognize the digits' profile, grayscale is enough. I applied mean grayscale in this project.

$$\text{grayscale\_val} = (R + B + G) / 3$$

3. GCN the Images

Global contrast normalized the images. The approach I apply is the same as what this describes : http://dp.readthedocs.io/en/latest/preprocess/
For each image, I subtract the mean of the pixels' gray scale. Then I divide every pixel by the standard deviation across the grayscale values.
After the GCN method, the data become zero-means and the values are scaled down.
Here is an example of a image before and after the GCN transformation:

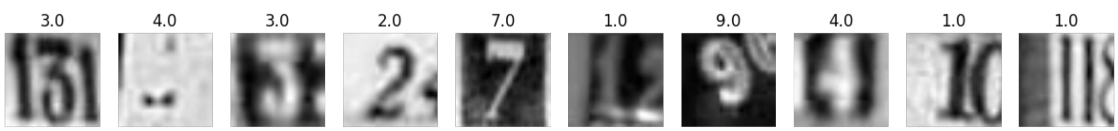
before :

```
[[[  33.66666667   19.           17.         ...,   64.33333333   72.           75.66666667]
  [  34.           20.33333333   17.         ...,   43.           59.           73.66666667]
  [  39.66666667   21.66666667   18.66666667 ...,   32.33333333   40.33333333   64.66666667]
 ...,
  [  80.66666667   80.           75.33333333 ...,   98.33333333   96.66666667   95.33333333]
  [  78.           79.           76.66666667 ...,   99.           98.66666667   98.66666667]
  [  80.33333333   81.           77.         ...,  100.66666667  100.           96.        ]]
```

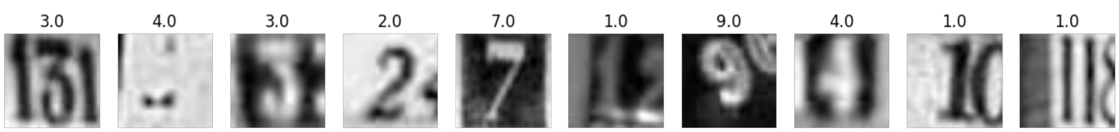
after :

```
 [[[-1.28437269 -1.91528671 -2.00132044 ...,  0.03481117  0.36460714  0.52233565]
   [-1.27003373 -1.85793089 -2.00132044 ..., -0.88288195 -0.19461211  0.43630191]
   [-1.0262715  -1.80057507 -1.92962567 ..., -1.34172851 -0.99759359  0.04915013]
 ...,
   [ 0.73741997  0.70874206  0.50799669 ...,  1.49738459  1.42568981   1.36833399]
   [ 0.62270833  0.6657252   0.56535251 ...,  1.5260625   1.51172354   1.51172354]
   [ 0.72308102  0.75175893  0.57969147 ...,  1.59775727  1.56907936   1.3970119 ]]
```

before :



after :



4. Collect a Normal Distributed Training Set and Validation Set

As the data exploration shown, real world data has more 1s and 2s than other numbers.
In training, the model should have equal chance to meet all ten labels. Therefore, I collect data from both `train_32x32.mat` and `extra_32x32.mat` to build my own training dataset of normal distribution. I also collect a normal distributed validation set, using the method shown in this paper

5. Shuffle the Data

I simply randomize the index and recollect the data.

Benchmark

A benchmark model I consider is the **[Conv. Maxout + dropout]** one used by [Goodfellow et al.].

The team preprocessed the images with local contrast normalization as what used by [Zeiler & Fergus (2013)]

The team picked 400 from the original training set and 200 from the extra one for each class to build the validation set. They used the rest of the training data and extra data for training and didn't touch the testing data.

Their model consists of three convolution & maxout layers and a fully connected maxout layer, followed by a fully connected softmax layer. In the paper, they did not specify the number of nodes for each layers. The best accuracy on the test set the team achieved then was 97.53%.

Implementation

(Note: conv means conv followed by relu, fc means fully connected)

Due to hardware limitation, my strategy to gradually scale up my model from a simple **[conv – fc – fc]** network.

In this project I use Stochastic Gradient Descent to do the update. I equally slice the shuffled dataset into mini batches and use them sequentially. The update strategy is Adam update (Kingma and Ba, 2014).

The implementation workflow is shown in **`model/cnn_mat.ipynb`**. Workflow is briefly three steps :

1. Unpickle the .pickle file created by **`model/preprocess_mat.ipynb`** and store the training, validation and testing datasets.

2. Build a TensorFlow computational graph and Run in a TensorFlow session.

3. Check the result according to the metric discussed before.

Refinement

Overview of this part

The **`model/experiment.ipynb`** contains all the milestone model I've tried. Due to hardware limitation, I didn't fine tune some hyper parameters like batch size, bias initialization and training steps. Instead I focus on architecture, weights initialization and update manners with these parameters unchanged.

The title of a experiments illustrate the key factor I focus on in that certain experiment. For example, if an experiment is named 'add another convolutional layer', I mean the experiment has all the other factor tuned to make the deeper network works, but they are less important in my non-demonstrated experiments.

Experiments

Here are some milestone model I've tried :
(Note: conv means conv followed by relu, fc means fully connected, validation set only used for intermediate testing)

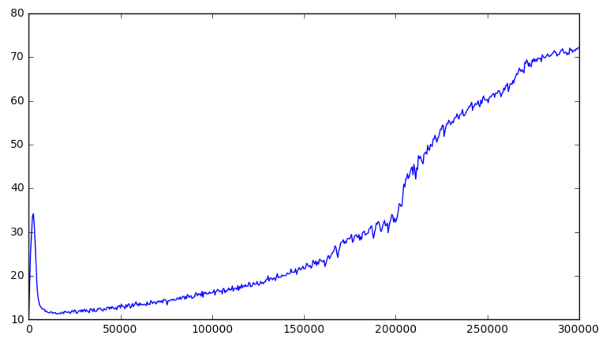
1. The Original Model :

architecture :

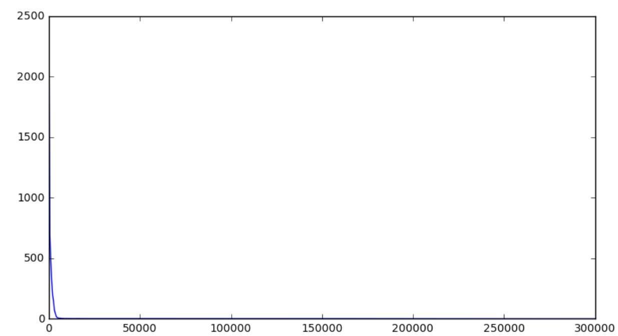
conv (zero padding)	5 x 5 x 32, stride 2
fc	8192 x 128
fc	128 x 10

weights initialization : TensorFlow default
bias initialization : 0.0 for conv, 1.0 for fc
learning rate : 1e-3
batch size : 256
steps : 30k
result :

a. test accuracy: 71.09%
b. learning curve:



validation accuracy



cross entropy

comment : The learning rate is not good for the model. Training curve looks funny.
The cross entropy drops very sharply in the starting phase. The plot won't tell, but actually the cross entropy did not become lower than 1.0 within 30k steps.

c. confusion matrix

```
        Confusion Matrix of testing data:

[[1245   28   60   15   63   10  127    2   51  143]
 [ 111 3786  174   55  620    9   18  280   27   19]
 [  36  217 3099  179  103    7   13  340   26  129]
 [  20   58  192 1781  161  151   39  132  183  165]
 [  73  160   37   71 1935   18  136   37   30   26]
 [  16   15   17  361  121 1580  127   13  120   14]
 [  85   20    4   42  171  148 1372    7  114   14]
 [   8   95  116  102   56    5    2 1622   10    3]
 [  71   12   25   78   78   36  149    7 1019  185]
 [ 114   13  174   52   41    3   17    8  105 1068]]
```

```
    Classification report of testing data:
             precision    recall  f1-score   support
          0       0.70      0.71      0.71      1744
          1       0.86      0.74      0.80      5099
          2       0.80      0.75      0.77      4149
          3       0.65      0.62      0.63      2882
          4       0.58      0.77      0.66      2523
          5       0.80      0.66      0.73      2384
          6       0.69      0.69      0.69      1977
          7       0.66      0.80      0.73      2019
          8       0.60      0.61      0.61      1660
          9       0.60      0.67      0.64      1595
avg/total       0.72      0.71      0.71     26032
```

comment: See the conclusion

conclusion :

As shown in the confusion matrix, there are 620 '1's that are classified as '4's. Intuitively, the reason might be that the model cannot recognize the little triangle of '4' and takes it as '1'. It seems the model doesn't extract enough feature now. Therefore more convolutional layers might help.

2. One More Convolutional Layer

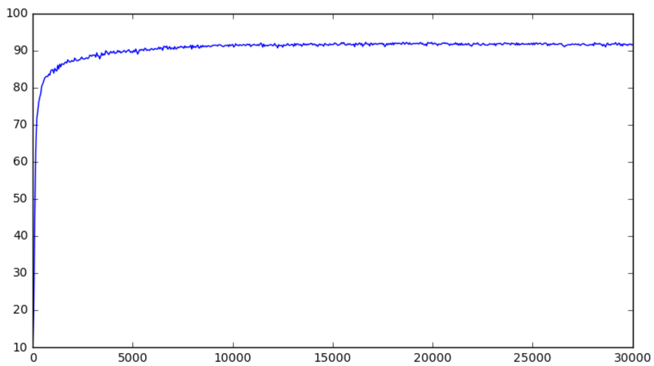
architecture :

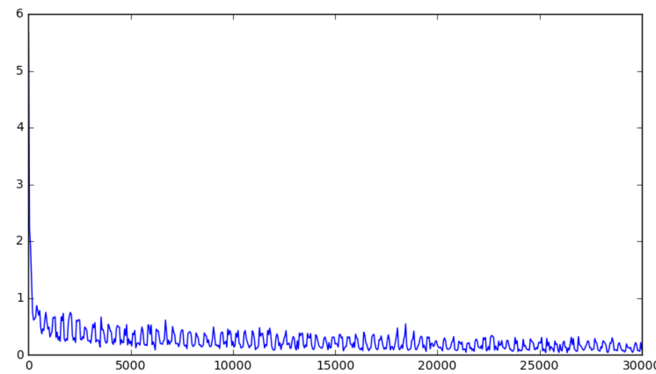
conv (zero padding)	5 x 5 x 32, stride 2, stddev 0.01
dropout	dropout rate 0.5
conv (zero padding)	5 x 5 x 32, stride 2, stddev 0.01
max pool	kernel 2 x 2, stride 2
fc	512 x 128, stddev 0.3
fc	128 x 10, stddev 0.3

weights initialization : see the table, as stated in the methodology, the weights should not block the number flowing, so the weights is increasing downstream.
bias initialization : 0.0 for conv, 1.0 for fc
learning rate : 1e-3
batch size : 256
steps : 30k
result :

a. test accuracy : 90.09%
comment : looks fine, the features map works

b. learning curve :





validation accuracy　　　　cross entropy

comment : the learning curve is more reasonable, and the cross entropy is by large dropping below 1.0

c. confusion matrix :

```
Confusion Matrix of testing data:
[[1635   14   12   11    8    4   28    4    9   19]
 [  97 4718   37   61   66   11   15   64   13   17]
 [  16   49 3854   94   28   20   11   39   13   25]
 [  19   55   49 2540    8   68   23   12   37   71]
 [  23   68   16   24 2324   10   15   13    8   22]
 [  13   19   14  101   16 2138   42    5   14   22]
 [  49   16    3   21   22   55 1760    1   37   13]
 [  13   66   41   24   17    6    6 1837    2    7]
 [  27   13   12   45   14   13   75    4 1409   48]
 [  34    7   33   15    7   18   11    4   23 1443]]
```

```
Classification report of testing data:
             precision    recall  f1-score   support
          0       0.85      0.94      0.89      1744
          1       0.94      0.93      0.93      5099
          2       0.95      0.93      0.94      4149
          3       0.87      0.88      0.87      2882
          4       0.93      0.92      0.92      2523
          5       0.91      0.90      0.90      2384
          6       0.89      0.89      0.89      1977
          7       0.93      0.91      0.92      2019
          8       0.90      0.85      0.87      1660
          9       0.86      0.90      0.88      1595
avg / total       0.91      0.91      0.91     26032
```

comment : position where the value is higher than 90, precision or recall that are lower than 0.9 in marked red

conclusion :

Compared with the previous model, this architecture fixed the '1' vs. '4' issue (number in green). It seems feature extraction works. But the model still struggling with '0' vs. '1', '2' vs. '3' and '5' vs. '3' (in red). Therefore, adding another convolutional layer is still promising.

3. Another Convolutional Layer

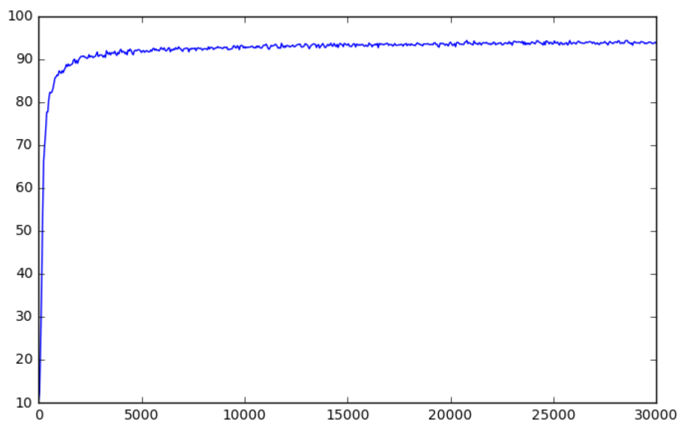
architecture :

conv (zero padding)	5 x 5 x 32, stride 2, stddev 0.01
conv (zero padding)	5 x 5 x 64, stride 2, stddev 0.02
dropout	dropout rate 0.5
conv	3 x 3 x 256, stride 2, stddev 0.04
max pool	kernel 2 x 2, stride 2
dropout	dropout rate 0.5
fc	1024 x 128, stddev 0.3
fc	128 x 10, stddev 0.5

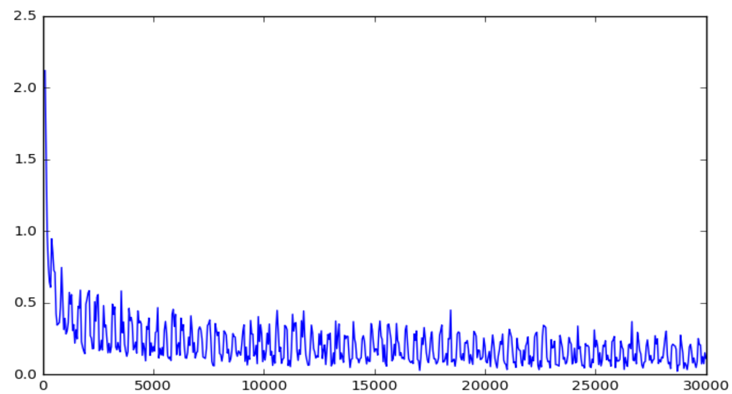
weights initialization : stated in the table
bias initialization : 0.0 for conv, 1.0 for fc
learning rate : 1e-3
batch size : 256
steps : 30k
result:

a. test accuracy : 94.03%

b. learning curves



validation accuracy



cross entropy

c. confusion matrix

```
        Confusion Matrix of testing data:

[[1685   12    4    2    5    6   12    3    4   11]
 [  66 4799   24   15   51   10   17   97   14    6]
 [   9   34 3950   23   16    5   11   78    4   19]
 [   7   40   22 2601   11   53   21   21   41   65]
 [  11   46    7    6 2412    4    6   14    2   15]
 [   8   11    9   45    6 2233   41    6    9   16]
 [  39   11    3    9    4   30 1858    2   15    6]
 [   4   31   11    1    3    3    1 1961    1    3]
 [  17    9   10   17    5   11   74    5 1473   39]
 [  29    6    9    6   12    7    4    6    9 1507]]
```

```
Classification report of testing data:
             precision    recall  f1-score   support
          0       0.90      0.97      0.93      1744
          1       0.96      0.94      0.95      5099
          2       0.98      0.95      0.96      4149
          3       0.95      0.90      0.93      2882
          4       0.96      0.96      0.96      2523
          5       0.95      0.94      0.94      2384
          6       0.91      0.94      0.92      1977
          7       0.89      0.97      0.93      2019
          8       0.94      0.89      0.91      1660
          9       0.89      0.94      0.92      1595
avg / total       0.94      0.94      0.94     26032
```

comment : red is where the value is higher than 60, or precision or recall is lower than 0.9

conclusion :

Again we can look at the confusion matrix to see how feature extraction works in the recognition. First, all the position marked red in the previous experiment is all fixed, but a new issue on ‘1’ vs. ‘7’ is introduced, which is very natural in real life.

I used CPU to train, so I decided to stop adding more convolutional layer. Next I will tune the weight initialization, and learning rate.

4. Weights Initialization

overview : Weights initialization is important and an active fields in research. This experiment didn't use too many mathematical theory to guide the process. The idea is just tunning the standard deviation. Higher stddev means the weights have more chance to distributed in a larger interval, so that weights are less likely to block the information flow because they are too small. Basically, the stddev drops while going upstream of the network.

weight initialization :

conv 1	0.01
conv 2	0.05
conv 3	0.07
fc 1	0.1
fc 2	0.2

other : same as experiment 3

result :

a. test accuracy : 94.94%

b. learning curve : skipped no more information than experiment 3

c. confusion matrix

```
Confusion Matrix of testing data:

[[1704   12    2    4    1    4    7    3    3    4]
 [  67 4888   23   17   25   13   10   34   14    8]
 [   9   27 4027   29    9    4    4   17   11   12]
 [  18   32   18 2633    2   48   18    4   45   64]
 [  14   47   11    9 2405    1    9   10    4   13]
 [   7   10    9   39    4 2251   39    2   13   10]
 [  44   11    2    3    4   28 1859    1   20    5]
 [   6   63   28    8    7    6    0 1895    1    5]
 [  19   13   11   13    5    5   40    1 1533   20]
 [  31   10   11    6    1    7    2    1    6 1520]]
```

```
Classification report of testing data:
             precision    recall  f1-score   support
          0       0.89      0.98      0.93      1744
          1       0.96      0.96      0.96      5099
          2       0.97      0.97      0.97      4149
          3       0.95      0.91      0.93      2882
          4       0.98      0.95      0.96      2523
          5       0.95      0.94      0.95      2384
          6       0.94      0.94      0.94      1977
          7       0.96      0.94      0.95      2019
          8       0.93      0.92      0.93      1660
          9       0.92      0.95      0.93      1595
avg / total       0.95      0.95      0.95     26032
```

conclusion :

The architecture is promising to be my best model. Weights initialization works well here and raised the test accuracy by 0.91%. A relatively good improvement.

5. My Best Model

Based on experiment 4, I introduced exponential learning rate decay and get 95.25% on test set.
`learning_rate = 1e-3 * 0.98^(global_step/500)`.
The confusion matrix is below :

```
Confusion Matrix of testing data:

[[1693    7    1    4    3    2   11    6    4   13]
 [  62 4856   23   18   36   11    7   65   17    4]
 [   6   21 4003   35   19    1    7   32    9   16]
 [  14   27   19 2665    8   34   18   10   46   41]
 [  17   30   12    2 2432    2    4   10    6    8]
 [   7   13    9   44    6 2250   35    2    8   10]
 [  37   13    3    5    5   14 1877    3   15    5]
 [   4   37   14    7    3    2    1 1950    0    1]
 [  25    8    2   14    6    3   35    1 1546   20]
 [  25    7   13    6    2    4    2    3   10 1523]]
```

```
Classification report of testing data:
             precision    recall  f1-score   support
          0       0.90      0.97      0.93      1744
          1       0.97      0.95      0.96      5099
          2       0.98      0.96      0.97      4149
          3       0.95      0.92      0.94      2882
          4       0.97      0.96      0.96      2523
          5       0.97      0.94      0.96      2384
          6       0.94      0.95      0.94      1977
          7       0.94      0.97      0.95      2019
          8       0.93      0.93      0.93      1660
          9       0.93      0.95      0.94      1595
avg / total       0.95      0.95      0.95     26032
```

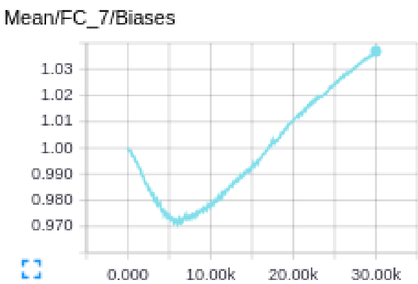
conclusion :

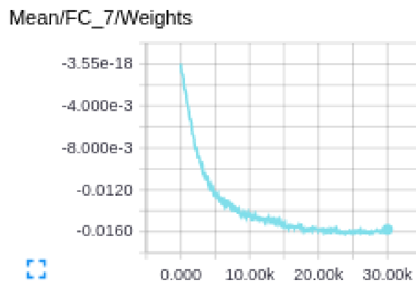
Stochastic Gradient Descent is a slow method. As we can see in experiment 3, loss is unstably decreasing, and the final loss is not the minimum of the curve. Learning rate decay decreases the footage when the the model approaching the global minimum so that it avoid stepping over the optimal.

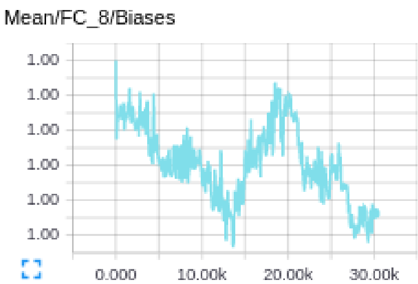
IV. Result

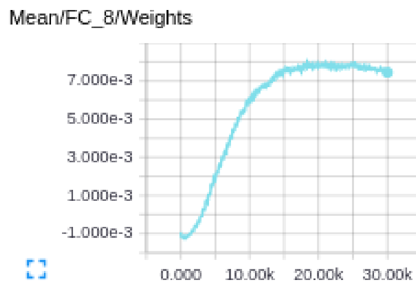
Model Evaluation and Validation

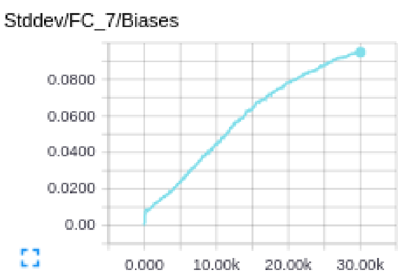
a. The confusion matrix is shown in experiment 5.
b. Weights and Biases

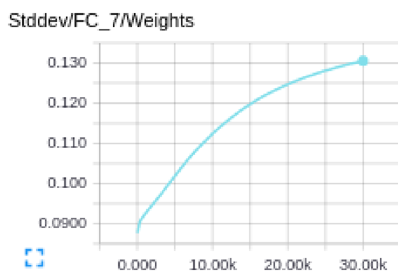


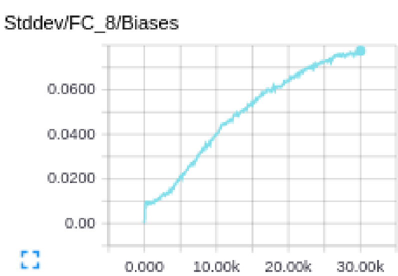


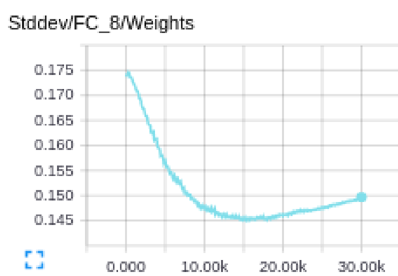






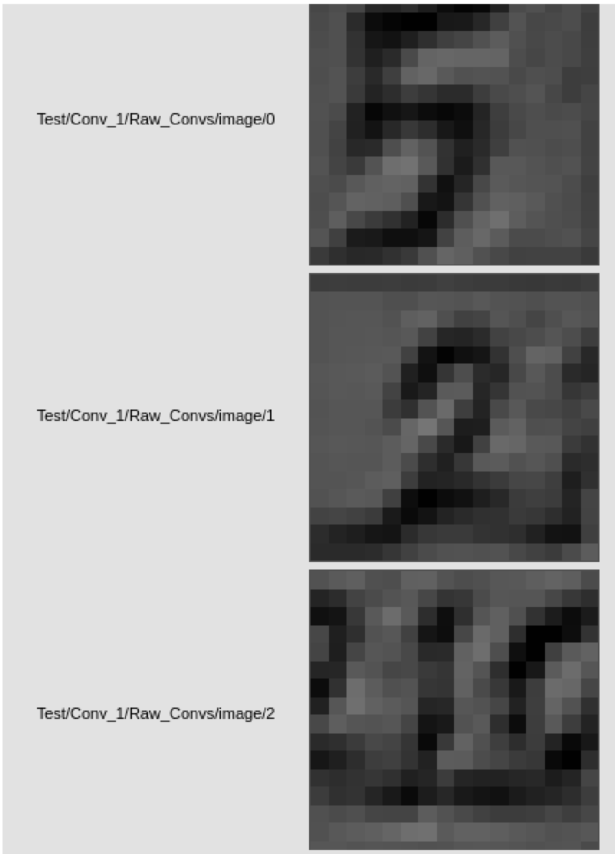




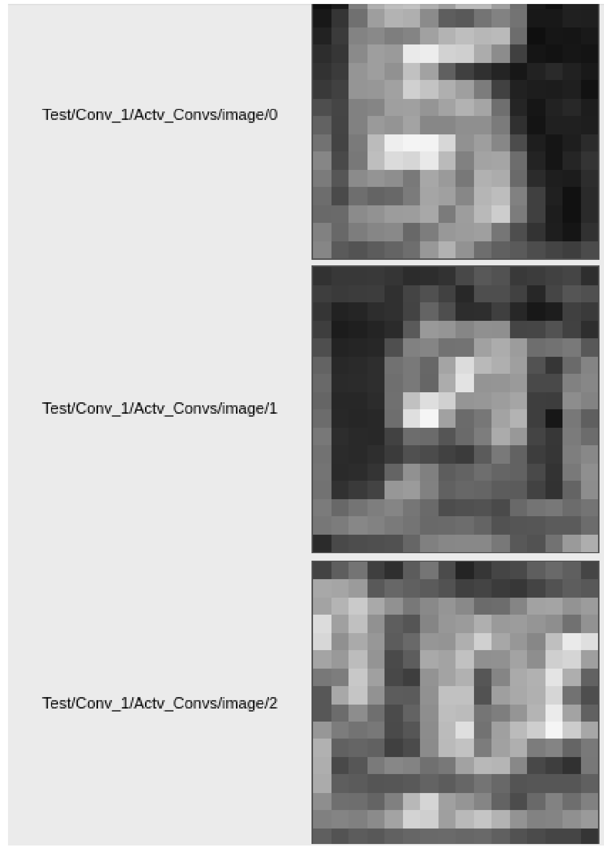


(Note: fc7 is fc1, fc8 is fc2, because I assigned indices along the net)

c. convolutional layers : (these may make no sense)



conv 1 before relu



conv 1 after relu

Justification

I use the same method to build the validation set and test on the same test set as the benchmark model. However, I didn't used all the rest of the data for training, and my training set is normal distributed. In addition, I used global contrast normalization instead of local contrast normalization for data preprocessing.

In terms of architecture, I just maxout on the last conv and use stride 2 kernels. In the last conv, I use 3 x 3 instead of 5 x 5 kernel. Though it is not specified in the paper, the benchmark model should have used stride 1 kernel so that they can maxout each convolutional layers. I skipped the discussion on other parameters since they are not specified in the paper that gives the benchmark model.

Compared with the benchmark model, mine loses about 2% on test accuracy. This might be caused by the design of architecture. My laptop would stuck forever when I tried 3 convolutional layers with stride 1. To save the training time, I did not use all the data or re-pickle the data to let the chance of training for each image be truly equal, so that might skip some difficult images and cause the accuracy lost.

V. Conclusion

Reflection

Stories :
It takes me a long time to finish this project. Picking deep learning as my capstone is quite ambitious, especially when you don't have any GPU support. At the beginning, I want to make the app or at least build the multi-digit model. But when it took my laptop 2-3 hours to run a epoch. I think I should just pick the single digit one before I update my hardware.
I started my journey on deep learning by taking the deep learning course on Udacity. However, I quickly found the course is not enough for me to fully understand how convolutional works. My first few models didn't work at all. So I turn to cs231n by stanford and apply the techniques and principle to build my model. And it worked!

Learned :
1. Math Is the Most Important.
In TensorFlow, the math is well packaged. In some cases, works for developers are just stacking nodes and layers, which is exactly what I decided to do at the beginning. However, I continuously came across various unexpected issues like adding layers didn't help improve the model. It was math that helps me out when I understood the idea of information flow along the network, which was my golden guide while doing refinements.

2. Documentation is Essential in Development
When we are tool users, documentations help us quickly master the tools. Documentation with examples can minimize the chances we get stuck in development. When we a tool developers, we should always keep in mind that we are not alone in creativity. Tools is valuable only when many people can easily use it.

3. Read More Papers
There are more and more papers in technology. Reading papers help keep track of the latest achievement people makes. It's necessary to fresh our mind often today.

Improvements

1. Update the hardware or use AWS to implement a model with stride 1 convolutional layers.
2. Use more data to train the model or make the data more difficult. For example, distort, rotate, blur or shear the images to create more data.
3. Try more to tune the hyper-parameters and improve the logging style.
4. Dig more on TensorBoard improve the structure of the summary.
5. Learn some advanced knowledge on deep learning and improve the technologies.
6. Work on the multi-digit recognition task.
7. Add more support to the project to make it more publishable, like build an app with the model built in.