Capstone Project

Machine Learning Engineer Nanodegree

YOONSU PARK
Oct 22 2016

Definition

Project overview

With the big advance in computing power with the enhanced deep learning technique, I can harness computer to understand and recognize the meaningful pattern in image, sound and even the surrounding environments.

Machine learning engineers are on the door step of the full blossom of Artificial Intelligence to perceive the world, just like a young child's conscious awareness operating as the child recognizes and identifies his parents, his toys, and other objects in his surroundings. His conscious awareness grows daily through the application of the four mental functions that Jung , renowned psychologist called “thinking”, ”feeling”, ”sensing”, ”intuiting” just like the deep neural network being trained to recognize the image of dataset.

Neural network has roots in the mechanism of neural systems of human brain connected with synapses to process input signals from its dendrites and producing output signals along its axon.

In this project, I would decode and recognize the sequences of digits from the natural images of Street View House Numbers (SVHN) through training Convolutional neural networks (CNN), which is the special case of the neural network with convolutional layers and subsampling layers.

This model could enable us to find the housing number of a specific location in a street as a format of continuous multiple digit characters with 94.7% prediction accuracy from test data, which is shy of human recognition 97% but could be a good starting point for us to improve to excel the capability of human vision recognition.

Problem statements

After scanning through images, the model will extract only the parts which contain digits, then classify digits through training convolution neural network. Our primary dataset for this project is the public street view dataset, provided in both the full number and the cropped digits.

In this project, I will design and implement the CNN model that learns to recognize single digit using real-world data , SVHN with cropped digit of .mat file format([1]format 2) as well as full number ([2]format 1).

I'd read real-word data(SVHN) in both cropped digits and full numbers using the convolution network model with Local Contrast Normalization and Gaussian filter to train and recognize single digit and multiple digit up to a sequence of 5 digit numbers.

And then I'll validate that the model performs well using the validation dataset.

I will explore the model to predict both the location (bounding box) and label (sequence of digits) simultaneously. I can break down this project into following details.

1.Preparation part – single digit process and training

- Read Cropped Digits: train_32x32.mat, test_32x32.mat , extra_32x32.mat

[1] Cropped Digits: train_32x32.mat, test_32x32.mat , extra_32x32.mat

[2] Full Numbers: train.tar.gz, test.tar.gz , extra.tar.gz

- Prepare 4 dimensional data as X and real number class labels as Y from each dataset as follows.(number % : (size of each data/ size of whole data) x 100)
  train_data : 33.3% , train_labels : 11.6%
  test_data : 33.3%, test_labels : 4.1%
  extra_data : 33.3%, extra_labels: 84.3%
  - inputs(X) is image data, the class labels (Y) is digit number from 1 to 10
- Create a validation dataset for hyper-parameter tuning and convert class labels to one hot encodings.
- Create validation set with 2/3from training samples(400 per class) and 1/3from extra samples (200 per class)
- Convert RGB color image to gray scale
- Global contrast normalizes by (optionally) subtracting the mean across features and then normalizes by either the vector norm or the standard deviation (across features, for each example)
- Construct and train a CNN and train the network using Training Data.
- Save data and their class labels in pickle file for later main step.

2. Conversion part – dataset conversion to csv file from .mat file.
  -Convert to csv file
  - As for train, valid, test dataset , draw the analysis graph of class label-wise frequency and the shape and height of bounding box .

3. Main part - multi digit processing and training
- I would download and extract real-word SVHN dataset of tar.gz format.
- I‘d recreate a list of such dictionaries as boxes and filename from extracted dataset.
- And then they were split into train, validation and test dataset.
- I generate dataset suitable for convolution network training such as image size adjustment and change to grayscale.
- In order to be best fit for convolution training , dataset were normalized through local Contrast Normalization and Gaussian filter.
- I'd create a 7 layers CNN Architecture with 3 convolution layers , 3 hidden layers with max pooling and fully connected layer

Problem in this project is to classify multiple digit up to a sequence of 5 digit numbers from images. So inputs(X) could be image data and the class labels(Y) could be sequence of 5 digit numbers.
To solve the problem ,I'd perform optimization step such as dropout ratio tuning and bias variable adjustment to test the convolution network model's performance and finally pick the best performer. And then randomly choose digit images and run them through the chosen model to find the example results.

Metrics

I would calculate the percentage of correctly predicted a single digit in images to measure the performance of chosen model as following module as follows.

```
def accuracy(predictions, labels):
        return (100.0 * np.sum(np.argmax(predictions, 1) == labels)
                / predictions.shape[0])
```

And as for the prediction accuracy of the continuous multiple digit for test data as follows.

```
def accuracy(predictions, labels):
        return (100.0 * np.sum(np.argmax(predictions, 2).T == labels) / predictions.shape[1] /
                predictions.shape[0])
```

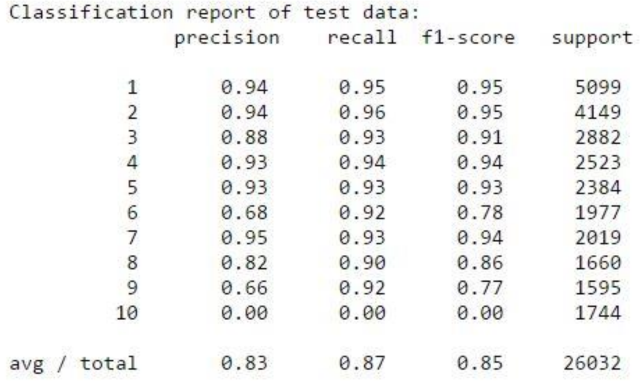
In above functions, the size of predictions[0] is 5 and that of predictions [1] is 64

In accuracy function, I′ d count the number of predictions which is equal to the labels, then divide it by the total number of predictions. In this model, the prediction outputs are 1 or 0 by softmax function with one hot encoded labels.

And also as for test data accuracy of the single digit prediction, I′d use Classification report as follows. In this report, I′d analyze digit class-wise Precision, which tells us what proportion of digit image, the model predict as a specific digit class, actually had the same digit label with the specific digit class *(Precision = True positive/(true positive +false positive)).*

And Recall, tells us what proportion of digit images that actually are labeled as specific digit were predicted by the model as the same specific digit class. *(Recall = true positive/(true positive + false negative)).*

And f1 score shows harmonic mean of precision and recall *: (precision + recall) / 2 X (precision X recall)*



```
Classification report of test data:
             precision    recall  f1-score   support

          1       0.94      0.95      0.95      5099
          2       0.94      0.96      0.95      4149
          3       0.88      0.93      0.91      2882
          4       0.93      0.94      0.94      2523
          5       0.93      0.93      0.93      2384
          6       0.68      0.92      0.78      1977
          7       0.95      0.93      0.94      2019
          8       0.82      0.90      0.86      1660
          9       0.66      0.92      0.77      1595
         10       0.00      0.00      0.00      1744

avg / total       0.83      0.87      0.85     26032
```

And I′d use Confusion matrix for the performance of classification(digit class 1 to 10(represents 0)), which is shown in later Analysis/ Exploratory Visualization part.

As for this module, I′d find test data accuracy by executing *accuracy(test_prediction.eval(), test_labels[:,1:6])* , which means returning two arguments, one is numpy array from tensor by tes_prediction.eval() and the other is 5 label sequence of test data predicted sequences.

Analysis

Data Exploration

I would use SVHN for input data for this project. SVHN is a real-world image dataset for developing machine learning and object recognition algorithms with minimal requirement on data preprocessing and formatting. It can be seen as similar in flavor to MNIST, but include an order of magnitude more labeled data (over 600,000 digit images) and comes from a significantly harder, unsolved, real world problem (recognizing digits and numbers in natural scene images). SVHN is obtained from house numbers in Google Street View images.

In Figure 1 I present the cropped digits for preprocessing and in Figure 2 I present full numbers for main project.

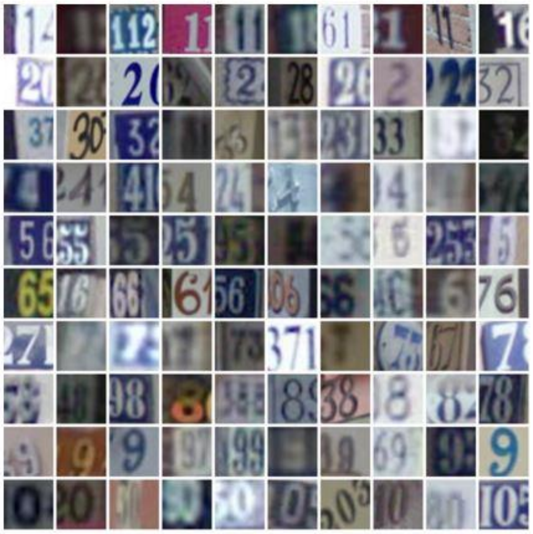


Figure 1 - Cropped digits for preprocess project part (*.mat)

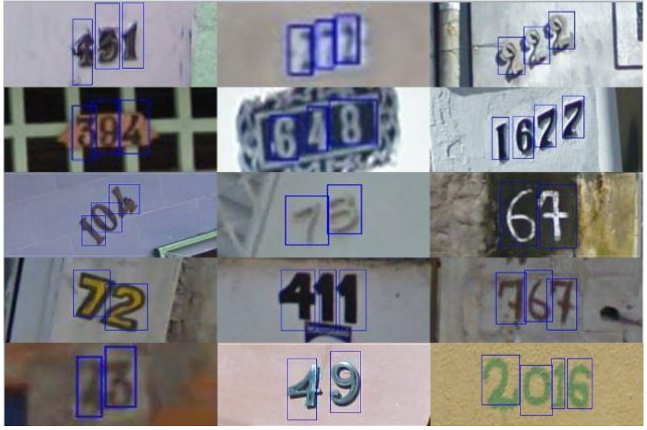


Figure 2 - full numbers for main project part(*.tar.gz)

In preparation project part of with single digit image, I use Cropped Digits as input data separated into three parts of 32x32 pixel images such as 32 train_32x32.mat, test_32x32.mat , extra_32x32.mat.

- As for each dataset I'd create 4 dimensional data as X and 2 dimensional class labels as Y .
- For example, As for test_data with 4-dimension(32,32,3, 26032), it has 26032 data elements with 32 by 32 pixel consists of 3 RGB values.

It is performed by following module and loaded test data has 32 by 32 pixel size.

test_data = scipy.io.loadmat('test_32x32.mat')['X']

The original character bounding boxes are extended in the appropriate dimension to become square windows as shown in Figure 1. Aspect ratio distortion could make recognize digit in image is that resizing them to 32-by-32 pixels does not introduce aspect ratio distortions[2] since as for this cropped digits, just like MNIST-like format, all digits have been resized to a fixed resolution of 32-by-32 pixels.

In other words, SVHN dataset provide char-level annotations by bounding boxes. Classifier need a fixed size input while the bounding boxes are always not square, which can't be wrapped into a fixed size without changing the aspect ratio, A usual method used in is to expand the bounding boxes and generate square digit images. But it adds much irrelevant background information. So I need more efficient way to handle input data, In order to reduce interference information, I need to   reduce the background information and maintain the information of source images. I first crop the digit images and resize to 32*32 according to the expanded bounding box annotation, and generate a binary map at the same size as the digit (32 x 32). I initial the binary map using the resized bounding boxes[11].

-I would test the accuracy of the classifier in classifying the data points using SGDClassifer which implements a plain stochastic gradient descent learning routine with decay. It shows, precision, recall, f1 score, for which I explained in previous Metrics parts. Precision, Recall, f1 score for training data using this classifier is 0.31, 0.17, 0.12, which shows much more poor performance comparing with the performance of classification report with single digit CNN classifier in previous " Metrics" part.

As for main project part with multiple continuous digit image, I would use tar.gz files. And then these files are extracted to raw images and digitStruct.mat file, in which bounding box information are stored.

These .mat file has dictionary structure which consists of bounding box of each digit and image file name such as following structure of train_data[4440] and loaded by the hd5f python library.

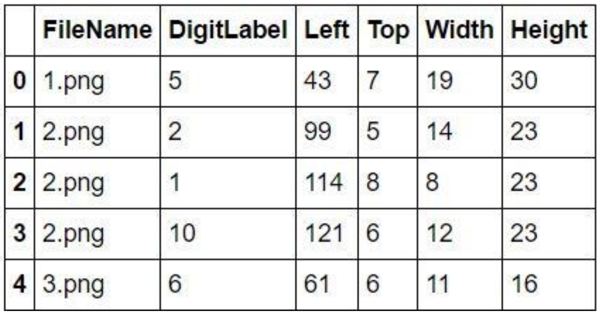
```
{'boxes': [{'width': 14.0, 'top': 20.0, 'label': 1.0, 'left': 65.0, 'height': 35.0},
{'width': 20.0, 'top': 19.0, 'label': 2.0, 'left': 80.0, 'height': 35.0}, {'width':
19.0, 'top': 20.0, 'label': 4.0, 'left': 100.0, 'height': 35.0}], 'filename':
'4441.png'}
```

Exploratory Visualization

1. Skewness toward digit character "1"

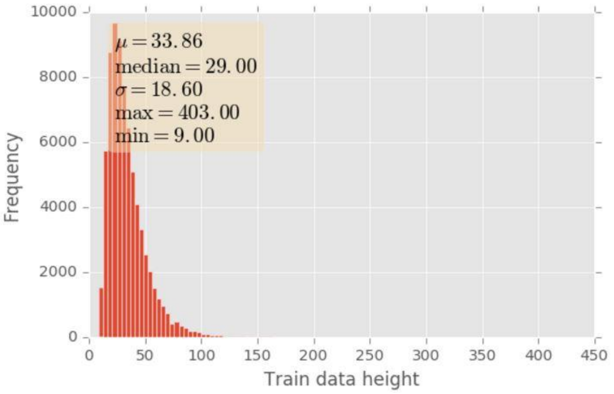
I would transform these DigitStruct.mat file to csv file for further data feature analysis. And read them as pandas DataFrame as follows.

In following graph , I can see the height of train, test data is much spread out with the lager height of digit character. Following histogram of SVHN characters height in the original image shows that resolution variation is large. (Median: 29 pixels. Max: 403 pixels. Min: 9 pixels).

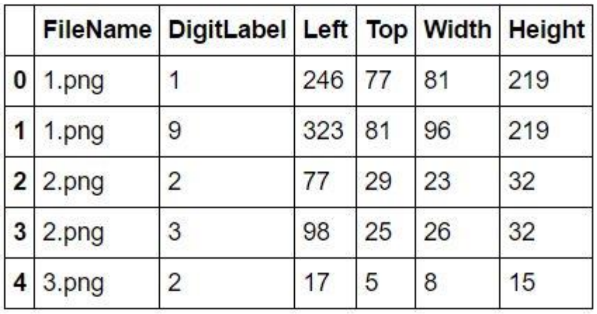


	FileName	DigitLabel	Left	Top	Width	Height
0	1.png	5	43	7	19	30
1	2.png	2	99	5	14	23
2	2.png	1	114	8	8	23
3	2.png	10	121	6	12	23
4	3.png	6	61	6	11	16

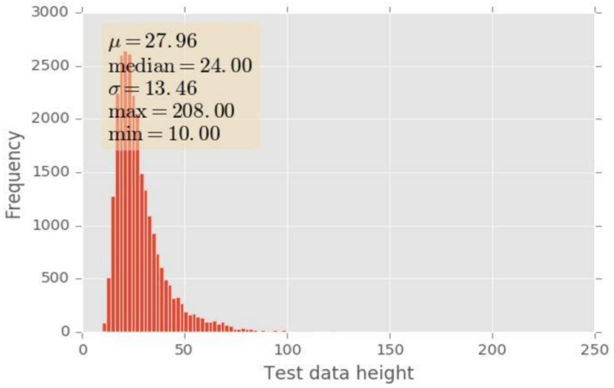
train data DataFrame converted from DigitStruct.mat(sampling 5 data points)



Train data height-wise Frequency

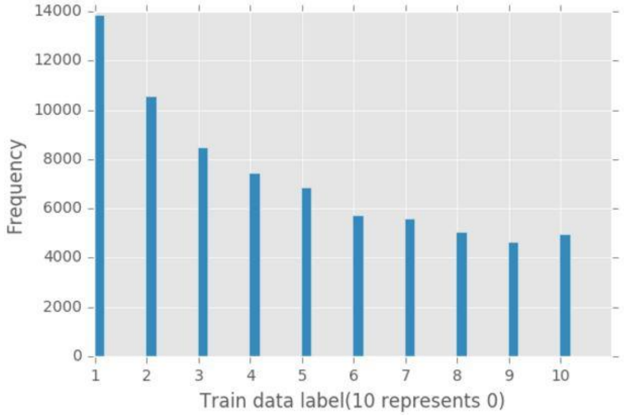


	FileName	DigitLabel	Left	Top	Width	Height
0	1.png	1	246	77	81	219
1	1.png	9	323	81	96	219
2	2.png	2	77	29	23	32
3	2.png	3	98	25	26	32
4	3.png	2	17	5	8	15

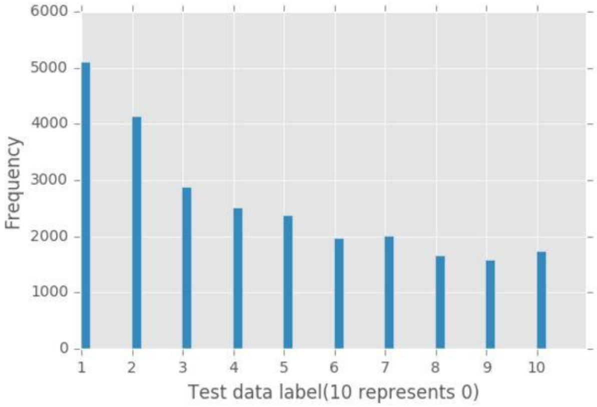


test data DataFrame converted from DigitStruct.mat(sampling 5 data points)

Test data height-wise Frequency



Test data number of digit character

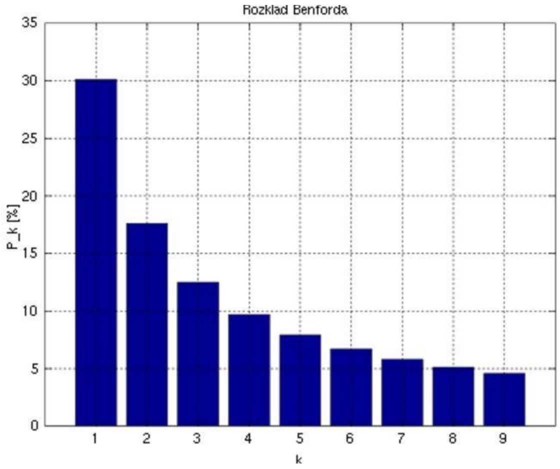


Train data number of character

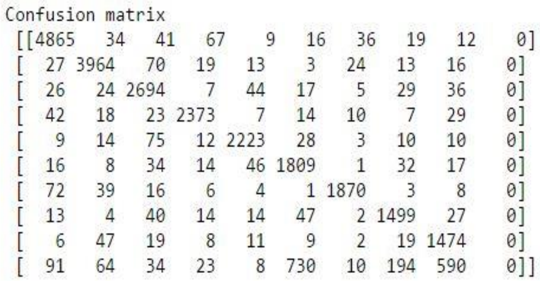
Above graph shows that digit 1 happens to be most frequent in train and test data altogether and digit 2 is the second most frequent . It shows data's skewness toward 1. Let me explain this data feature as follows.

If given an input digit image, the probability of simply guessing correctly is 10% for each digit image,
But , SVHN train , test dataset is skewed toward 1 , this situations is comply with Benfords law. According to the law, "many naturally occurring collections that the leading digits are not uniformly distributed, but heavily skewed towards smaller digits such as 1" (WIKIPEDIA).

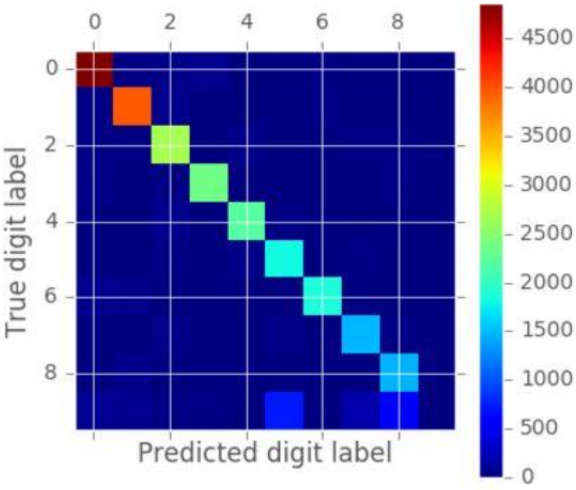
Even though, train, test dataset have skewness toward 1, in other words, the frequency of 1 ,2 is much higher than other digits, it is hard to say it imply the imbalance of classification, since following confusing matrix shows high frequency digit such as 1,2 has the proportional high accuracy . As shown in following confusion matrix graph, As for the test data, the diagonal is where the classification is more dense. The diagonal toward 1 is much more dense, which shows the good performance of classification. I could say the high frequent digit image such as 1 has high accurate prediction in proportion to the frequency since the most dense cell(top, left) is 1.(correct prediction -predict label is 1, while actual label is 1)



“ Benford’s law” (WIKIPEDIA)



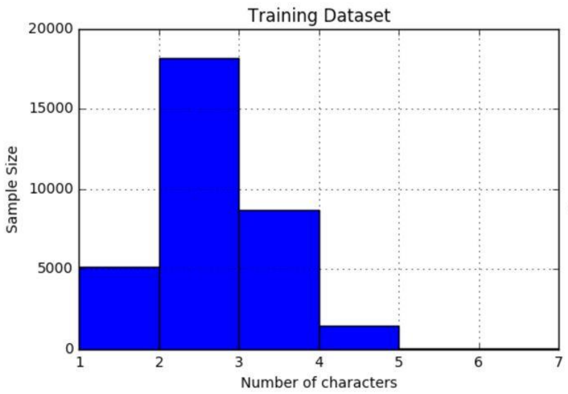
```
Confusion matrix
 [[4865   34   41   67    9   16   36   19   12    0]
 [  27 3964   70   19   13    3   24   13   16    0]
 [  26   24 2694    7   44   17    5   29   36    0]
 [  42   18   23 2373    7   14   10    7   29    0]
 [   9   14   75   12 2223   28    3   10   10    0]
 [  16    8   34   14   46 1809    1   32   17    0]
 [  72   39   16    6    4    1 1870    3    8    0]
 [  13    4   40   14   14   47    2 1499   27    0]
 [   6   47   19    8   11    9    2   19 1474    0]
 [  91   64   34   23    8  730   10  194  590    0]]
```

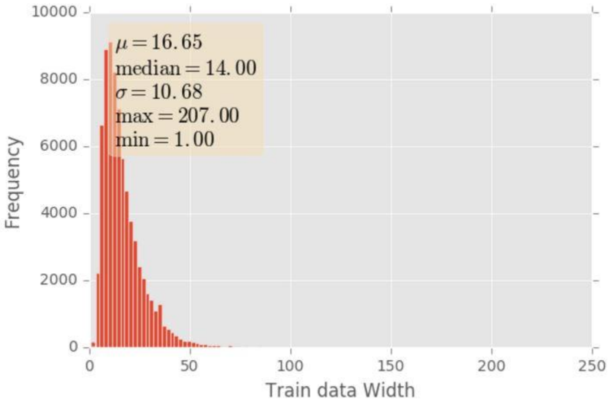


Confusion matrix graph

Above confusion matrix graph clearly shows the network performing better on classes with a higher percentage of the data distribution, such as classes 1, 2. This is partly because of the skew discussed above.(right bar shows the frequency of digit in each category

2. Number of digits in image to train

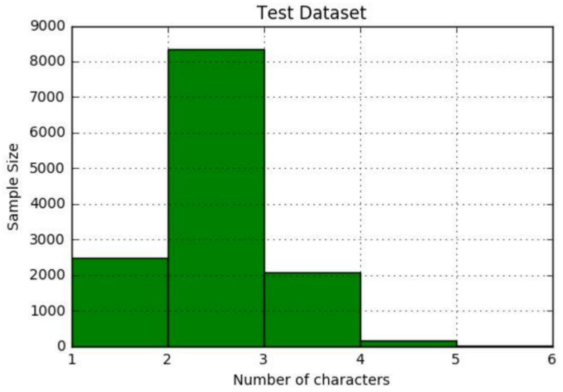


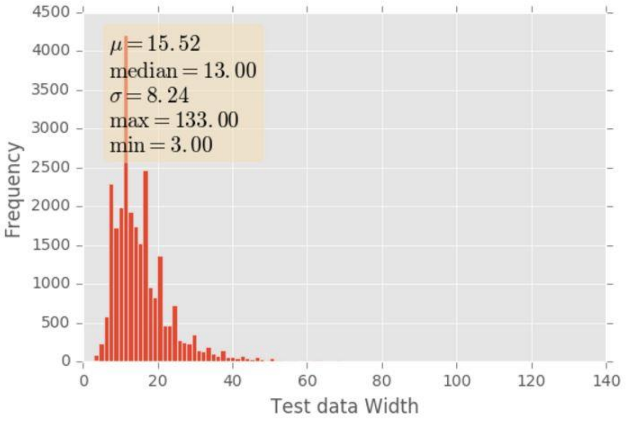


```
Number of character:corresponding frequency pair -  train data
{2: 18130, 3: 8691, 1: 5137, 4: 1434, 5: 9, 6: 1}
```

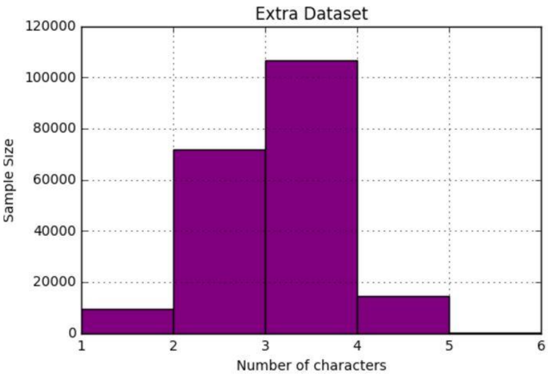
And the shape of width-frequency graph for train data in the right corresponds with that of number of character graph in the left exactly.

It shows 2 characters data are most frequent and then follows 3, 1, 4, 5 characters. I would choose to train data with characters number 1,2,3,4,5 which is frequent enough to train. So I can ignore data whose number of digit are bigger than 5 as for the training purpose.





Number of character:corresponding frequency - {2: 8356, 1: 2483, 3: 2081, 4: 146, 5: 2}
It shows 2 characters data are most frequent in test data then follows 1, 3, 4, 5 characters.
It is optimal to test data with number of digits from 1 to 5.



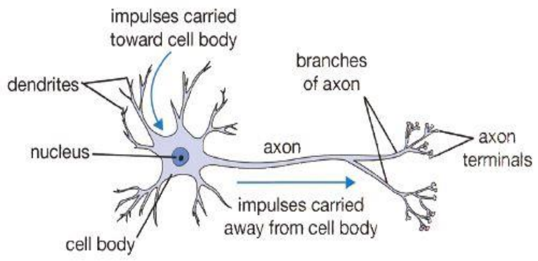
Right graph shows 3 characters are the most frequent in extra data then follows 2,4,1,5 characters.

As for all train, test, extra data, It is optimal to choose image with number of characters from 1 to 5

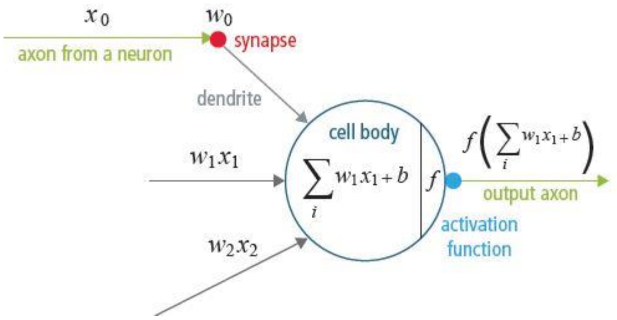
Algorithms and Techniques

In this project, I would choose the convolution neural network(CNN) to recognize the sequences of digits from the natural images through supervised learning.

Since CNN is the special case of the neural network, First of all, I would explain neural network.



Human brain neuron mechanism



Neural network mechanism

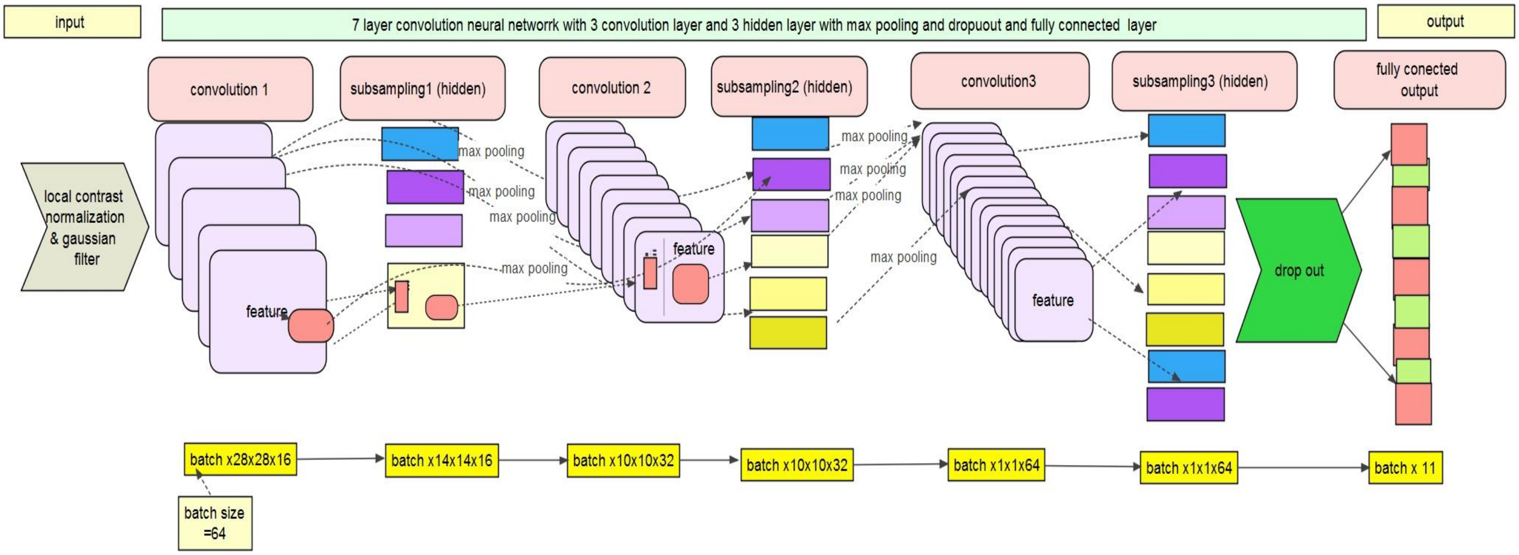
> “In the neural network computational model, the signals that travel along the axons (e.g., x0) interact multiplicatively (e.g., w0x0) with the dendrites of the other neuron based on the synaptic strength at that synapse (e.g., w0). Synaptic weights are learnable and control the influence of one neuron or another. The dendrites carry the signal to the cell body, where they all are summed. If the final sum is above a specified threshold, the neuron fires, sending a spike along its axon.” (Samer Hijazi 2) And the firing rate of the neuron is modeled with an activation function f such as Rectified Linear Unit (ReLU)function ”

A CNN is a special case of the neural network described above. It is neurobiological motivated by the findings of locally sensitive and orientation-selective nerve cells in the visual cortex. But a CNN consists of one or more convolutional layers, often with subsampling layers or hidden layers, which are followed by one or more fully connected layers as in a standard neural network.

So, Why I choose CNN for this project ?

- CNN is special kind of multi-layer neural networks to extract relevant features implicitly.
- CNN is a feed-forward network that can extract topological properties from an image.
- Convolutional Neural Networks are designed to recognize visual patterns directly from pixel images with minimal preprocessing.
- They can recognize patterns with extreme variability (such as digit characters).

As for this project, I would deploy 7 layers CNN Architecture with 3 convolution layers , 3 hidden layers with max pooling and fully connected layer as follows.



CNN model deployed in the project

After direct input from image pixel, the dataset go through the local Contrast Normalization and Gaussian filter.
(1) Local Contrast Normalization(LCN) : It subtracts from every value in a feature a Gaussian-weighted average of its neighbors (high-pass filter) and then divide every value in a layer by the standard deviation of its neighbors over space and over all feature maps. So Subtractive plus Divisive LCN performs a kind of approximate whitening. And Gaussian filter applied to an image will blur the image and reduce fine details but reduce the noise in an image.

(2) Convolution layers : It extracts feature and share their parameters across space. After convolution, width and height of the resulting matrix is lesser or equal to the input image. The depth is increased according to the filters applied.

(3) Subsampling layer (hidden layer) : I use *tf.nn.relu()* which computes rectified linear: max(features, 0) but this layer's output dimension is same as input without changes.

(4) Pooling : it's function enables the model to reduce the data rate or the size of the data but
keep the maximum information in the reduced dimension of data as many as possible, which is carried by max pooling in this project, a max value is selected from values of equally divided input matrix. So, it reduces the computational cost by reducing the number of parameters to learn
-Subsampling layer with pooling reduces the resolution of the features. It makes the features robust against noise and distortion.

(5) Dropout : It is used to reduce overfitting since it has a function, "early stopping" by using dropout at the cost of taking quite a lot longer to train. In this project use various dropouts. Among them, dropout probability, 0.9375 has the best test accuracy 94.7% , which is somewhat better than that of 0.5 with 94.3% accuracy at same bias variable 0.0. From this results. So, I could say I could get a little bit better performance without suffering from overfitting with a bigger dropout probability 0.9375 comparing with the dropout probability 0.5.

(6) Fully connected layer performs mathematically sum a weighting of the previous layer of features, indicating the precise mix of "ingredients" to determine a specific target output result. All the elements of all the features of the previous layer get used in the calculation of each element of each output feature.

After direct input from image pixel, my CNN model performs as following sequence.
-Convolutional layer 1, batch_size x 28 x 28 x 16
- Sub-sampling(hidden) layer 1, batch_size x 14 x 14 x 16 and max pooling
-Convolutional layer 2, batch_size x 10 x 10 x 32
-Sub-sampling (hidden) layer 2, batch_size x 10x 10 x 32 and max pooling
-Convolutional layer 3, batch_size x 1 x 1 x 64
-Sub-sampling(hidden) layer 3 , batch_size x 1 x 1 x 64
-Dropout batch_size x 1 x 1 x 64
-Fully-connected layer, Output layer :Weight size: batch_size x 11(blank, 0 ~9)

Benchmark

In preparation project part , I'd train our proto-type CNN model with cropped image data for predicting a single digit and attained 84.7% accuracy. According to the research paper of " Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks" (Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, Vinay Shet Google Inc., Mountain View, CA), they achieve over 96% accuracy in recognizing complete street numbers(SVHN) and on a per-digit recognition task, they'd achieve 97.84% accuracy.

To be a meaningful project to predict SVHN data with good accuracy, I set a goal to attain at least 90% accuracy or accuracy range from 90% to 95% on test data, since I can consider this is close to human accuracy with an acceptable margin since this is not like Google map building project in which it is extremely important to have at least human level accuracy such as 97 ~98%.

Methodology

Data Preprocessing

In preparation project part to predict single digit image, I use Cropped Digits as input data separated into three parts of 32x32 pixel images such as 32 train_32x32.mat, test_32x32.mat , extra_32x32.mat. I'd use scipy.io.loadmat() function to read the data since they are all MATLAB's .MAT format.
As for each dataset, I'd create 4 dimensional data as X and 2 dimensional class labels as Y .

As for main project part with multiple continuous digit image, I would use a gzipped files. These files are extracted to raw images and digitStruct.mat file in which bounding box information is stored.
These .mat file has the dictionary which consists of bounding box of each digit and image file.

I'd generate dataset suitable convolution network training and then normalize and convert them to grayscale
When converting from RGB to grayscale, the specific weights to channels R, G, and B need to be applied.
These weights are: 0.2989, 0.5870, 0.1140.

im = np.dot(np.array(im, dtype='float32'), [[0.2989],[0.5870],[0.1140]])

By mean subtraction , I can achieve centering the cloud of data around the origin along every dimension and attain the normalization of data through the operatioin of dividing the centered data by standard deviation(std)

im = (im - mean) / std
dataset[i,:,:,:] = im[:,:,:]

I'd split them into train, validation and test dataset.

Implementation

I would implement this project with three parts such as preparation project part, conversion part, main project part with 5 jupyter notebook files.

Part 1 : I'd perform data processing for cropped digit image and CNN training to classify single digit.

1. **capstone_preparation_project.ipynb**
   - download and extract the SVHN dataset.
   - checks download file size with original file size
   - load.mat file(train,test,extra) using scipy.io.loadmat()
   - merge and prune training data
   - crate validation set
   - convert to grayscale
   - normalization
   - save processed data to pickle
   - using SGDClassifier, show Training score, Validation score, precision, recall, f1 score.
   - open pickle file and reformat to 4-dimension from 3-d data
   - local contrast normalization and Gaussian filter
   - create convolution neural network model and train single image input data

- show accuracy
- save trained model as .ckpt to use in capstone_main_CNN.ipynb.ipybn for CNN training.

Part 2 : I'd convert to csv file from digitstruct.mat file to analyze data structure and visualization.

1**. Capstone_digitStruct.ipynb**
- read digitStruct file and prepare for csv conversion

2. **capston_digitStructMatToCsv.ipynb**
- convert to csv file
- convert csv format to pandas DataFrame for visualization and data analysis.
  - Mean , median, standard deviation analysis
  - Test label-wise frequency analysis
  - Data height, width analysis

Part 3 : I'd download the continuous multiple image and analyze, manipulate and train them with the convolution network model .

1**.capstone_main_preprocess_project.ipynb**
- download and extract the svhn dataset as a format full image, gzipped file format.
- check download file size with original file size
- extract to raw images and digitStruct.mat file in which bounding box information are stored.
- generate train, test, extra dataset from digitStruct.mat and create validation set and save as pickle file

2. **capstone_main_CNN_project**
- I load the saved pickle file from capstone_main_preprocess_project.ipynb
- local contrast normalization and Gaussian filter
- define accuracy function
- create convolution neural network model and train continuous multiple image data
- initialize Weights, biases and 5 classifiers to be used to recognize each digit
- I'd read the trained model from capstone_main_preprocess_project.ipynb. I train our training dataset using it
- Test with testdata and show accuracy

Convolution Neural Network model of this project

I'd create CNN model with 7 layers after local contrast normalization
convolutional layer, batch_size x 28 x 28 x 16
sub-sampling(hidden) layer, batch_size x 14 x 14 x 16 and max pooling
convolutional layer, batch_size x 10 x 10 x 32
sub-sampling (hidden) layer, batch_size x 10x 10 x 32 and max pooling
convolutional layer, batch_size x 1 x 1 x 64
hidden layer , batch_size x 1 x 1 x 64
Dropout batch_size x 1 x 1 x 64
fully-connected layer, Output layer weight size: batch_size x 11(blank, 0 ~9)

After training the model, I'd calculate the prediction accuracy for minibatch, valid data and test data.

Refinement

The model trained with cropped .mat data in capstone_preparation_project.ipynb attained an accuracy of 87.3% of test accuracy with just 100000 steps. I'd read this model and then load data with bounding box and train to predict continuous multiple digit image with new updated CNN model including adjusted bias variable as Ill as updated dropout probability. I achieved up to 94.7% test accuracy by these refinement as follows.

In performance enhancement step, I'd test following various combination of parameters and bias and choose Model4 for the optimal solution.

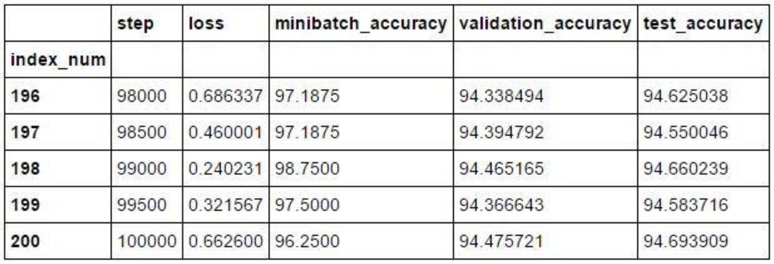
This table shows the test accuracy of this model4 with dropout probability 0.9375 and bias 0.0 at last 100000 step . The test accuracy is about 94.7%.

As for the update CNN model, I added Local contrast normalization step and Gaussian filter to the model. As I explained in Algorithm and Technique part, Local contrast normalization performs local subtractive and divisive normalizations, enforcing a sort of local competition between adjacent features in a feature map, and between features at the same spatial location in different feature maps[12]. Gaussian filter would reduce the noise in an image.

.

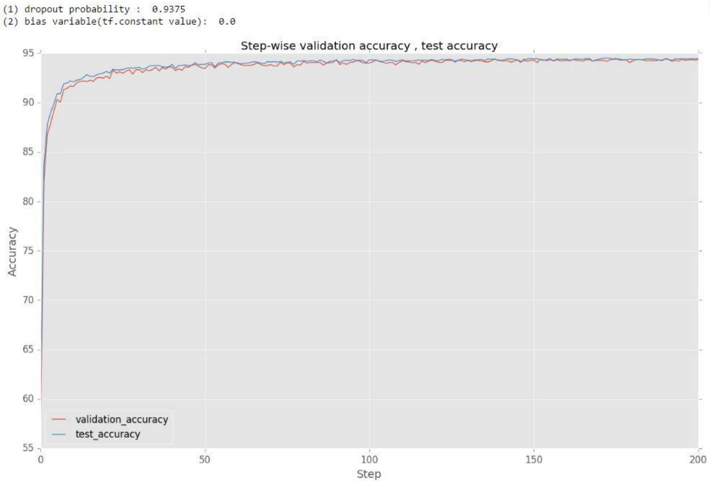
	Dropout probability	Bias variable (tf.const)	Accuracy: minibatch at step 100000	Accuracy: Validation at step 100000	Accuracy: Test at step 100000
Model 1	0.9375	1.0	55.0	57.5	64.2
Model 2	0.5	1.0	55.0	57.5	64.2
Model 3	0.5	0.0	96.3	94.3	94.4
Model31	0.6375	0.0	97.8	94.5	94.5
Model32	0.7375	0.0	97.8	94.4	94.5
Model33	0.8375	0.0	96.5	94.6	94.6
Model 4	0.9375	0.0	96.3	94.5	94.7
Model41	1.0	0.0	98.1	94.4	94.3

CNN model-wise accuracy with a dropout , a bias



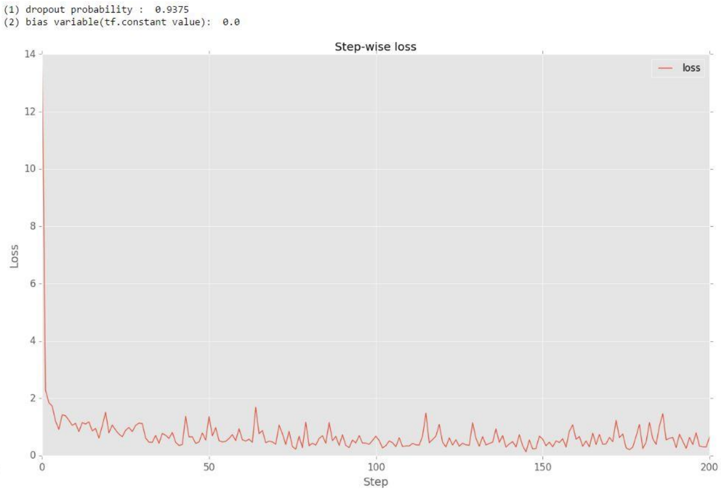
	step	loss	minibatch_accuracy	validation_accuracy	test_accuracy
index_num					
196	98000	0.686337	97.1875	94.338494	94.625038
197	98500	0.460001	97.1875	94.394792	94.550046
198	99000	0.240231	98.7500	94.465165	94.660239
199	99500	0.321567	97.5000	94.366643	94.583716
200	100000	0.662600	96.2500	94.475721	94.693909

Iteration step-wise loss and accuracy



(1) dropout probability : 0.9375
(2) bias variable(tf.constant value): 0.0

x-axis : index_number(number of iteration): (x500)



(1) dropout probability : 0.9375
(2) bias variable(tf.constant value): 0.0

number of iteration : (x500)

For the purpose of reducing overfitting and increase test accuracy, I'd implement a dropout layer to the network which achieve the test accuracy up to 94.7% in case of dropout probability, 0.9375 , which is the best accuracy among models as shown in above table, "CNN model-wise accuracy with a dropout, a bias".

In order to find optimum dropout probability , I'd choose 6 dropout probabilities from 0.5 to 1.0 with intervals such as 0.5, 0.6375, 0.7375, 0.8375, 0.9375, 1.0 and find 0.9375 performs best. As I explained in Algorithm and Technique part, a bigger dropout probability such as 0.9375 could achieve a little bit better performance without suffering from overfitting. But dropout probability 1.0 which implies no dropout get test accuracy 94.3% , which is lower than any other test accuracy of 5 dropout probabilities with bias 0.0.

I'd notice that bias is very important to produce higher accuracy. When I choose bias 0.0 , it produce much better test accuracy. In this model, bias is used in two parts as follows.

hidden= tf.nn.relu(conv + bias)
logits1=tf.matmul(flat, Weight) + bias

Together with shared weights , shared bias comprise a kernel or filter, which perform mapping from input layer to hidden layer.

For example, at 1st convolution layer, with 16 depth slice, each size [28 x 28], there are 12,544 neurons (28 x 28 x 16) at each batch(batch_size, 64), while patch size is 5, weight is [5,5,1,6] and bias is [16], Together this, this add up to 326,614 parameters(28 x 28 x 16 x 26) on convolution network alone, which is very high number (26 is derived from 5(patch) x 5(patch) + 1(bias)). But with shared weight and bias, My model can constrain the neuron in each 16 depth slice to use same Weight and bias as 416 parameters(16 x 5 x 5 (weight) + 16(bias)) , in which there are 16 unique set of weights(one for each slice) and 16 bias.

In ReLU function in which weights and bias exist as arguments , bias acts as the threshold of ReLU function activation as follows.

$$My\ convolution\ model\ has\ neuron, y = f(\sum w\ x + b), \text{x is input, w is Weight, b is bias.}$$

$$\text{My activation function is a rectifying linear unit (relu), i.e. } f(x) = \max(0, x),$$

$$\text{so, y} = \max(0, \sum_i w\ x + b\ )$$

relu will only allow signal to propagate forward if it is greater than 0. So neuron only "activates" (has a non-zero output value) when $\sum_i w\ x + b \ > 0,\ in\ other\ words,\ \sum_i w\ x > -b,$ So the bias term for a neuron will act as an activation threshold in my model setup (ReLU nonlinearities) . Since my model adaptively learn these bias terms via backpropagation, I could interpret this as my model is allowing neurons to learn when to activate and when not to activate. So when initial bias is set to 0 for 16 bias in 1st kernel/filter as described above, neuron of my model is propagated forward if $\sum_i w\ x > 0$, i.e. sum of product of shared weight and input is greater than 0. The model with this bias, 0 has better accuracy than the model with bias 1, which propagated when $\sum_i w\ x > -1$. But it is irrelevant to present the generalized guide to set the bias to get the best performance since the performance with a specific bias could be highly affected by the depth of convolution layer and hidden layer and combination of parameters and selection of activation functions . etc. so, this subject is beyond of this report.

Results

Model Evaluation and Validation

At the last 100000 step of iteration , I could attain the Minibatch loss of 0.66, Minibatch accuracy of 96.3%, validation accuracy of 94.5% and test accuracy of 94.7%. I'd deploy the model with 3 convolution layer 3 hidden layer with 2 max pooling, one dropout, in which I'd select the bias 0.0 for RELU activation function threshold to activate to propagate convolution network, and dropout probability 0.9375 which enable to reduce overfitting by "early stopping" since the bigger dropout probability such as 0.9375 would give a better performance without suffering from overfitting comparing with dropout probability 0.5.

Also , I'd use AdagradOptimizer with learning rate of decay at 0.05 , which adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameters. Since it is well-suited for dealing with sparse data such as sensor detected data and image data.

I'd create 5 classifiers and logits to classify a continuous 5 digit character.

Justification

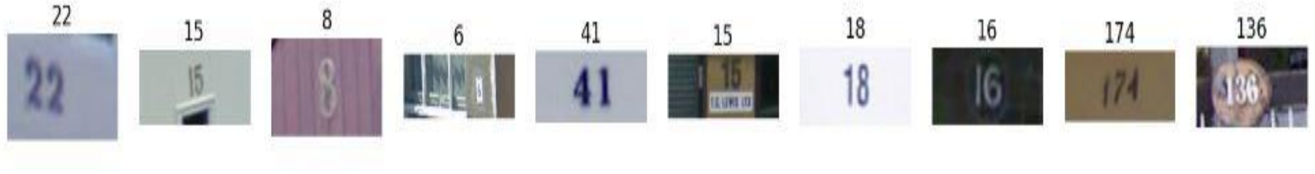
In Benchmark section, I set the accuracy goal to 90%, for which I'd attain the test accuracy of 94.7% with 100000 number of iteration, which is shy of the human recognition accuracy of 97% but much better than SGDclassifier. While I have achieved the goal in terms of prediction accuracy, This CNN system has a room to improve to attain human-level performance through more detailed fine-tuning such as hyper –parameters.

Conclusion

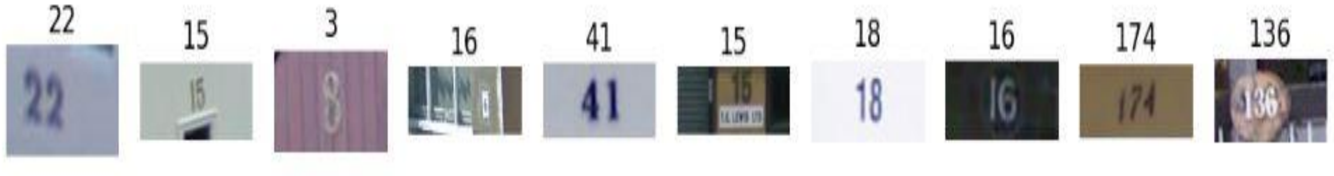
Free Form Visualization

- For the reference of training model-wise accuracy analysis graph, I’d attach “free_form_visualization.pdf” file.

In this part, I would predict both the location (bounding box) and label (sequence of digits) simultaneously to localize where the numbers are.
First of all, I need to check how well our model works. So, I choose images randomly and then make prediction using it.
Following images show the random images and their labels.



After passing these images through our CNN model, I can see the prediction result from our model as follows.



According to this results, I could say our CNN model fairly good at predicting test data as Ill as training data.

Reflection

CNN give the best performance in pattern/image recognition problems and even outperform humans in certain cases and housing number recognition can be one of these fields. I have achieved promising results using the optimization and hyper parameters tuning with CNN. I have developed 7 layered CNN with max pooling and dropout for recognizing multiple digit images, achieving 94.7% prediction accuracy on the test dataset.
During the project, the most challenging aspect is the hardware setup to accommodate this CNN model. As for CNN training, I’d use notebook computer with 8GB RAM, intel core7, Ubuntu OS , but result in system freeze after 8000 iteration step. Then , I’d use HP DL380 16GB RAM with dual intel Zeon CPU to train the model, but it takes 12 hours to train dataset.
So finally, I’d use a computer with 32GB RAM , intel core 7 with GTX1080 GPU, which make it possible to finish training full dataset just in 10 minuets. So I could say GPU is essential to train deep learning models.

I’d use two kinds of SVHN dataset. One is cropped .mat files for predicting one character and the other is gzipped data with bounding box to predict continuous multiple digit.

The latter shows far better test accuracy. So I would consider data with bounding box is preferable for this project.

I would try various dropout probability , which make meaningful difference for the test accuracy and furthermore ,I found the bias variable for the convolution layer also have a huge impact on test accuracy.

As for data preparation, I convert the original color house number image to grayscale by multiply specific Weights to channels R, G, and B of image data .

Improvement

I can improve accuracy by implementing recurrent neural networks (RNNs) using long short-term memory (LSTM). RNNs can deliver the current state of the art in time-series recognition tasks as well as this image recognition project, which means it can predict 5 digits at once without 5 classifiers which this project deployed.

Also, I can improve the model with Deep belief networks, another promising type of network utilizing restricted Boltzman machines (RMBs)/autoencoders. Deep belief networks are capable of being trained greedily, one layer at a time, and hence are more easily trainable for very deep networks

References

-[1] The Udacity Forum
-[2] http://ufldl.stanford.edu/housenumbers
-[3] http://cs231n.github.io/convolutional-networks/
-[4] Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural
Networks/ Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, Vinay Shet
Google Inc., Mountain View, CA
-[5] http://yann.lecun.com/exdb/publis/pdf/jarrett-iccv-09.pdf
-[6]https://discussions.udacity.com/t/how-to-deal-with-mat-files/160657
-[7] http://ufldl.stanford.edu/tutorial/supervised/ExerciseConvolutionAndPooling/
-[8] https://www.tensorflow.org/versions/r0.10/tutorials/deep_cnn/index.html
-[9] Using Convolutional Neural Networks for Image Recognition /
Samer Hijazi, Rishi Kumar, and Chris RoIn, IP Group, Cadence
- [10]Reading Digits in Natural Images with Unsupervised Feature Learning /
Yuval Netzer1 , Tao Wang2 , Adam Coates2 , Alessandro Bissacco1 , Bo Wu1 , Andrew Y. Ng
-[11] Reading Digits in Natural Images with Unsupervised Feature Learning Yuval Netzer1 , Tao Wang2 , Adam Coates2 , Alessandro Bissacco1 , Bo Wu1 , Andrew Y
-[12] What is the Best Multi-Stage Architecture for Object Recognition?/ Kevin Jarrett, Koray Kavukcuoglu, Marc’Aurelio Ranzato and Yann LeCun