Capstone Project

Machine Learning Engineer Nanodegree

Sayaka Nogawa
December 19th, 2016

Table of Contents

Difinition

Project Overview

This project aims to recognize house numbers in images using deep learning model. Image recognition is one of the most important technologies in the present world. It is used in famous applications and smartphones.

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 incorporates 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.

Probrem Statements

In this capstone project, the goal is to train a model to solve the problem of multi-chalactor digit recognition using the Google Street View image dataset.
The classes being the digits 0 to 9. The model have to finally be able to predict all digits.

My approach are following:

1. **Download and load training and testing datasets file**

2. **Separate two variables**
   The data is a 4D matrix image. The labels are 1D matrix.

3. **Data preprocessing**
   Resize the images to all heve the same size.

4. **Train the deep convolutional network with data**

5. **Adjusting the model**
   Find the best accuracy on test data and Optimizer.

Metrics

To measure the performance of the model, we using the ‘accuracy’. Accuracy is using the percentage of correctly predicted sequence.

The accuracy function is following:

```
def accuracy(predictions, labels):
  return (100.0 * np.sum(np.argmax(predictions, 2).T == labels)) / labels.size
```

Figure 1. accuracy fanction

This is the similar function as defined in the assignment of the Udacity Deep Learning Course.

Analysis

Data Exporation

The SVHN dataset has a dataset of street numbers, along with bounding boxes. This data set has two formats, one is the original image and the other is the image clipped to 32 × 32 like MNIST (many of the images do contain some distractors at the sides).

Dataset can be found at http://ufldl.stanford.edu/housenumbers/.

Example Images:

Please see the 2 examples below.

Figure 2. Format 1: Full Numbers

These are the original, variable-resolution, color house-number images with character level bounding boxes, as shown in the examples images above. (The blue bounding boxes here are just for illustration purposes. The bounding box information are stored in **digitStruct.mat** instead of drawn directly on the images in the dataset.)

The original full numbers dataset is divided into train data, validation data and test data (73257 digits for training, 26032 digits for testing, and 531131 extra training data). These are stored in a digitStruct.mat file. Loading this file requires like the hd5py python library. Since these are already divided into three data sets, we do not have to separate them, we only randomly select and shuffle.

Each element in digitStruct has the following fields:
name : which is a string containing the filename of the corresponding image.
bbox : which is a struct array that contains the position, size and label of each digit bounding box in the image.

Figure 3. Format 2: Cropped Digits

All digits have been resized to a fixed resolution of 32-by-32 pixels. Loading the .mat files creates 2 variables:
X : which is a 4-D matrix containing the images
y : which is a vector of class labels.
To access the images, X(:,:,:,i) gives the i-th 32-by-32 RGB image, with class label y(i).

Exploratory Visualization

The character height is distance between the top and the bottom of the bounding box
Resolution variation is large. (Median: 28 pixels. Max: 403 pixels. Min: 9 pixels.

Dataset	Median	Max	Min
Train	28px	403px	9px

Table 1. SVHN haracters height

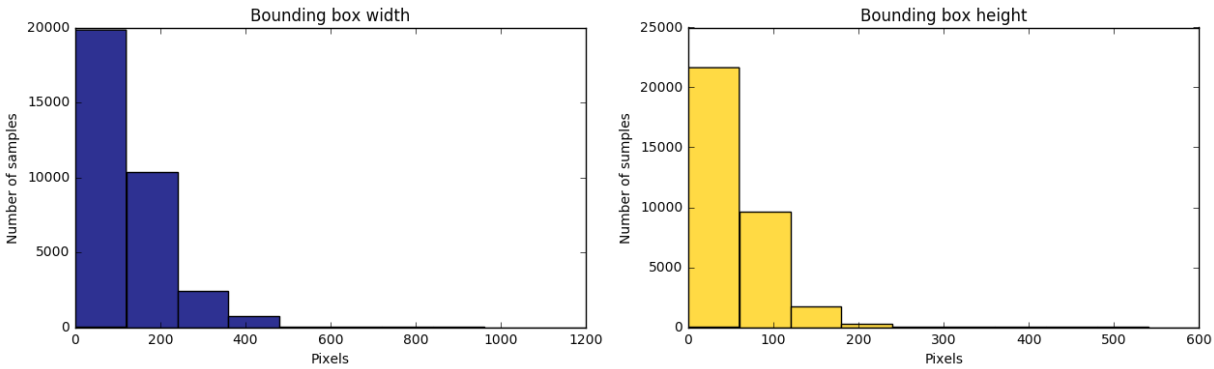


Figure 4. Train dataset's bounding box width and height

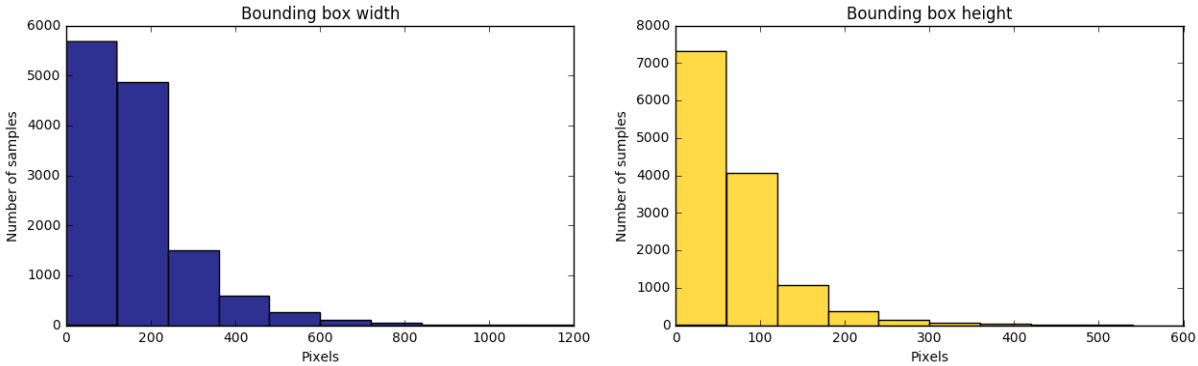


Figure 5. Test dataset's bounding box width and height

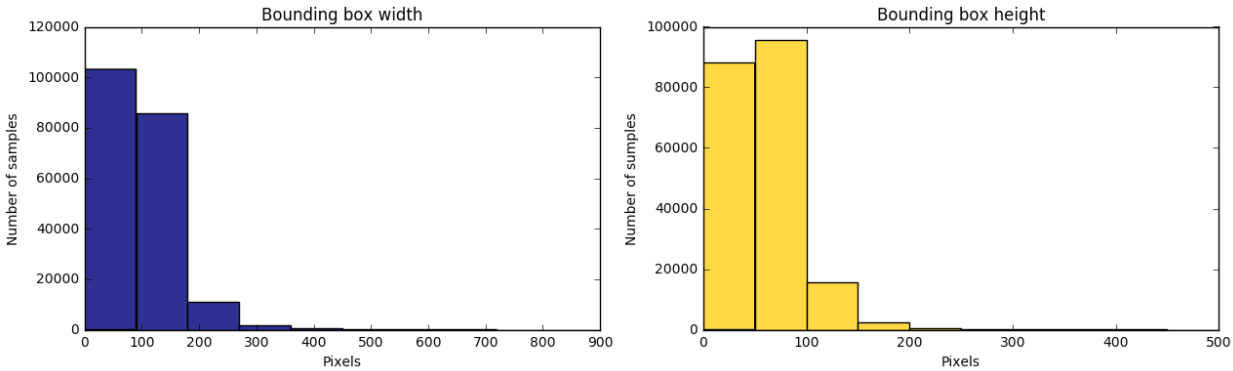


Figure 6. Extra dataset's bounding box width and height

Algorithms and Techniques

I am useng a CNN (Convolutional Neural Network) to recoginize images. CNN is a Neural Network which introduces Convolution Layer which creates by convolving information of a region in a filter.

Below is an explanation of each part of CNN architecture:

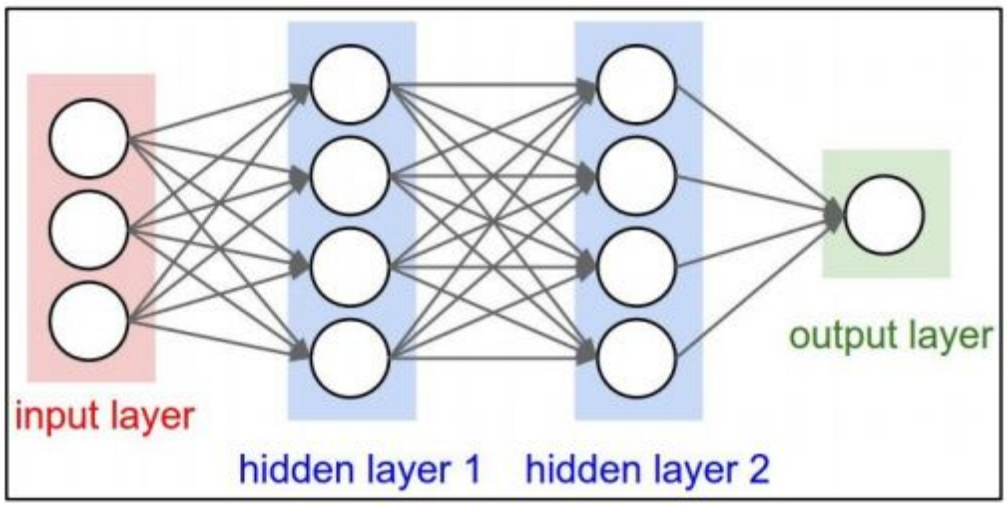


Figure 7. *Convolutional Neural Networks*

1. **Input Layer**
   Hold the raw pixel values of the image

2. **Convolution Layer**
   Layer to perform convolution of feature amount
   Convolution Layer is created by applying while moving the filter, and the number of filters is created. By repeating this and connecting it with an activation function (ReLU etc.), a network is constructed. By convolution, it is possible to extract features on a region basis instead of points, and it becomes robust to image movement and deformation. In addition, it is also possible to extract features which are not known unless they are region based, such as edges.

3. **Pool Layer**
   Layers to shrink and manage layers

There is merit that compression of the image size makes it easy to handle in the later layer.

4. Fully Connected Layer

From the feature amount, the layer to be final judged

It is a layer connected to all the elements of the previous layer, mainly used in layers that make final judgment.

5. Output Layer

Contains the class scores

Dropput

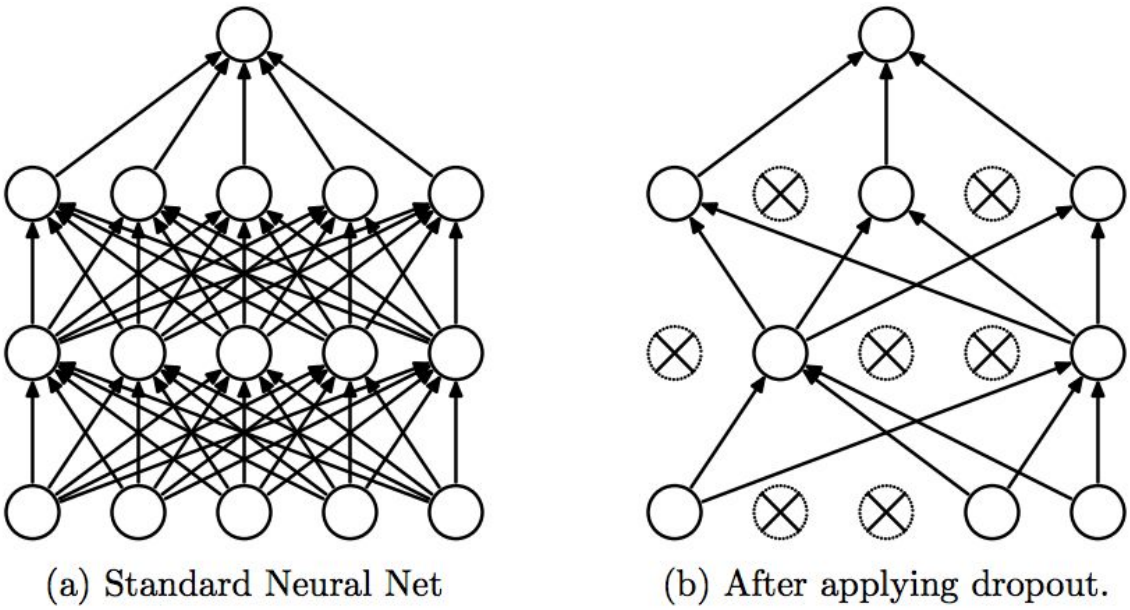


Figure 8. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*

In Dropout, when learning a neural network, learning is performed by invalidating some of the nodes in the layer (treat as though it does not exist at all in one update) with an update, and in the next update, another node repeat learning by disabling it. This makes it possible to forcibly reduce the degree of freedom of the network at learning time to raise generalization performance and to avoid over learning. In the hidden layer, it is said that it is generally better to invalidate about 50%. In my code, keep_prob represents the dropout rate.

Benchmarks

According to a survey of this report (Reading Digits in Natural Images with Unsupervised Fature Learing - Deep Learning
), the human recognition rate of this data set was 98%. Practically, it should be 98% or more, but considering the machine power of Macbook, I think that it is better to go about 90%.

Methodology

Data Preprocessing

- **Reshape**
  I crop the original image to 32 x 32 to remove redundant information.

- **Transform Image to grayscale**
  Since the SVHN dataset is the one of the real world, the color is attached (3 color channels). To recognize digits, colors are less important and sometimes get in the way. Therefore, I decided to convert the images to grayscale.

- **Nomalize images**
  The final preprocessing step is to delete all the images that exceed 5 digits.

- **Create validation dataset**
  Validation data was created using the train_test_spilit function.

train dataset	**230540**
test dataset	**13068**
valid dataset	**5214**

Table 2. SVHN haracters height

- **Shuffle the data**
  Suffule data for the train_dataset and the extra_datase.

The following is the image after the preprocessing is completed.

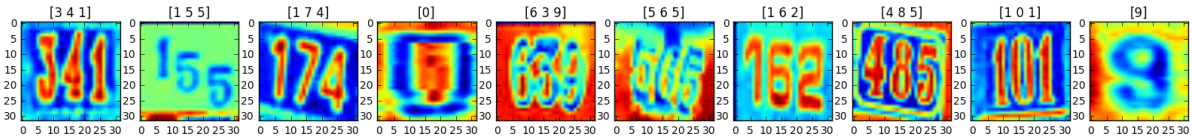


Figure 9. *Data preprocessing*

Implementation

Python version :
Python 2.7

Typical use library :
Tensorflow (Google)
numpy
h5py
matplotlib
scikit-learn

Architecture of the CNN:

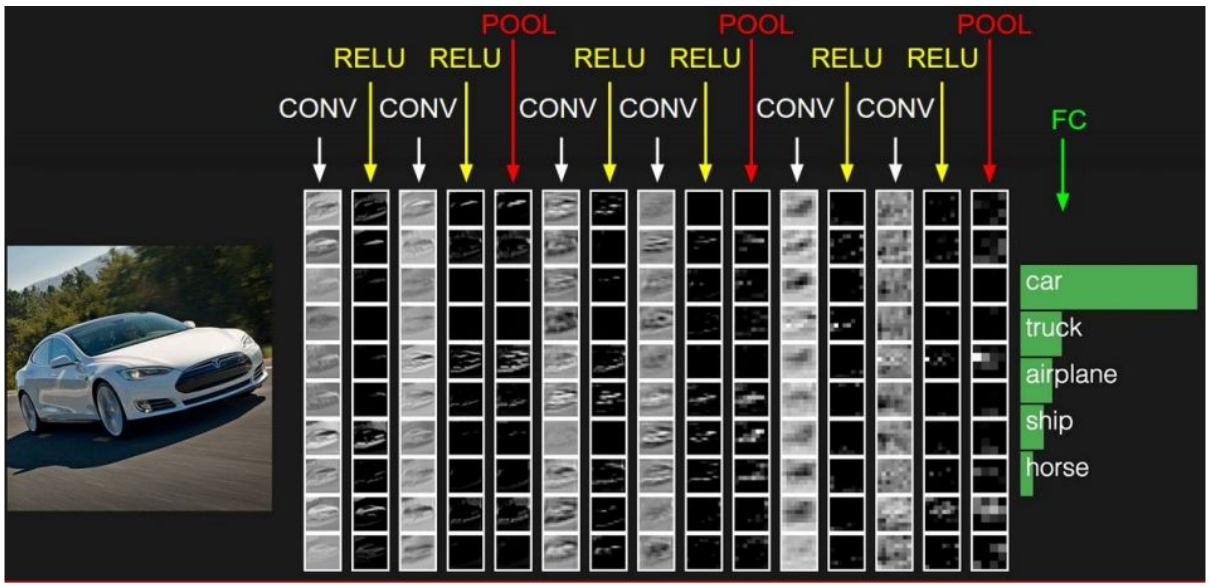


Figure 10. *Convolutional Neural Networks*

1. **Convolutional layer 1**
   batch_size x 28 x 28 x 16 / convolution size : 5 x 5 x 1 x 16

2. **Pooling layer 1**
   batch_size x 14 x 14 x 16

3. **Activation Funtion**

4. **Convolutional layer 2**
   batch_size x 10 x 10 x 32 / convolution size : 5 x 5 x 32 x 32

5. **Pooling layer 2**
   batch_size x 5 x 5 x 32

6. **Activation Funtion**

7. **Hidden layer**
   layer size : 32 x 64

8. **Dropout layer**
9. **Output layer**
   layer size : 64 x 10

Refinement

In this model, adjustment of parameters is important.
I mainly adjusted the following parameters.

- **bach_size**
- **hidden units**
- **learning rate**
- **number of layers**
- **dropout probability**
- **epochs**

Results

Model Evaluation and Validation

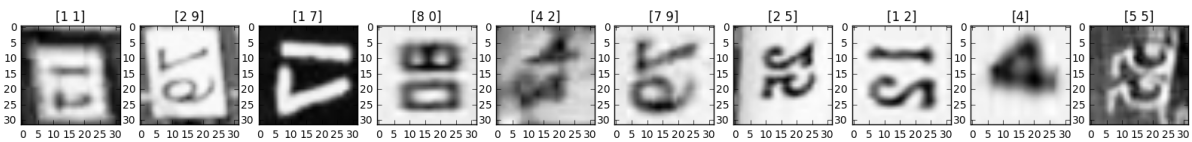
Machine : Macbook pro Early 2015
Processor : 2.7 GHz Intel Core i 5
Memory : 8GB

Test Accuracy : **92%**

Parameters :
batch_size = 64
patch_size = 5
image_size = 32
label_size = 6
num_channels = 1 # gray scale
num_labels = 11
depth = 16
num_hidden = 64
keep_prob = 0.5

num_step = 100001
learning_rate = tf.train.exponential_decay(0.15, global_step, 1000, 0.96)

The following is a random sample of test data. It is nearly correctly read.



Justification

The best model I build reaches 92% test accuracy. As per my benchmark I was expecting accuracy of more than 90%. Although it is good precision, there is still

room for improvement because I think that it is a bit insufficient to put it into practical use if it can not recognize the same level as a person (human's performance on this dataset reaches 98% accuracy).

Conclusion

Reflection

In this project, we have used SVHN dataset of the real world to identify the house number by algorithm using neural network. By using deep learning, what we can not do until now can be done automatically.

Deep learning is a field that has been drawing much attention now. For example, recently, computer vision has been introduced at Amazon stores, allowing people to do shopping without having to go through a cash register.

In addition, this live camera application algorithm can be used not only for hobby use but also for deterring crime. More accuracy will improve and if you can clearly identify the number of cars such as alphabets and numbers, it will help a safe society.

Improvement

I think the possibility of the following improvement.

- **Hardware**
  Building a better environment using the GPU will result in better accuracy in a shorter time. After that, it may be possible to create a model that can be improved beyond the recognition rate of people.

- **Algorithm**
  I may also be able to adjust various parameters and find a more optimal model. There are different approaches to deep learning, so there may be more appropriate approaches to this data set.

References

1. The Street View House Numbers (SVHN) Dataset
2. Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks - Google Research
3. Reading Digits in Natural Images with Unsupervised Fature Learing - Deep Learning
4. Convolutional Neural Networks
5. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*
6. Udacity Deep Learning Course
7. Udacity Forum
8. An overview of gradient descent optimization algorithms
9. Number plate recognition with Tensorflow