Capstone Project

Machine Learning Engineer Nanodegree

Sayaka Nogawa
December 19th, 2016

Table of Contents

Definition

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.

Problem Statements

In this capstone project, the goal is to train a model to solve the problem of multi-character 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

The given function calculates the argmax for the predictions and the labels, takes the number of predictions where it is equal to the labels and then calculates the percentage accuracy using the number of labels. The metric I have chosen is inherently simple, but I am able to accurately report the accuracy of the forecast.

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.

Dataset	Median	Max	Min
Train	28px	403px	9px

Table 1. SVHN characters height

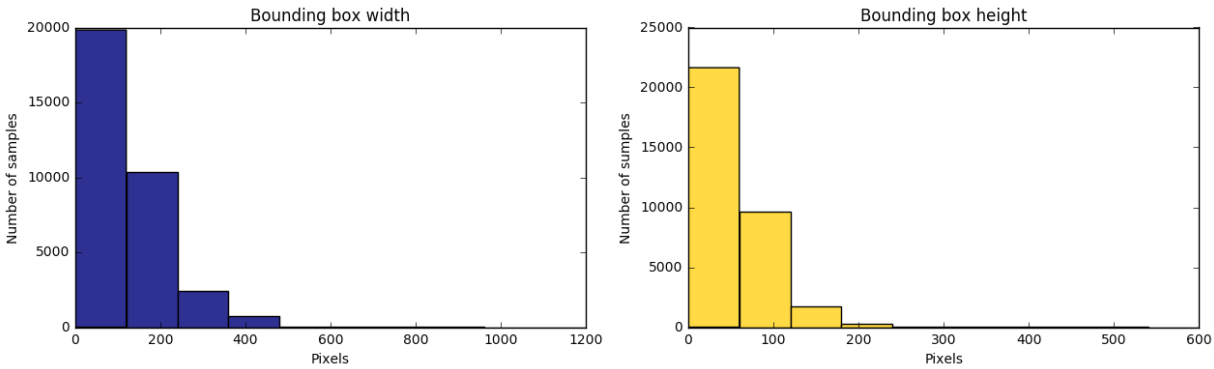

Figure 4. Train dataset's bounding box width and height

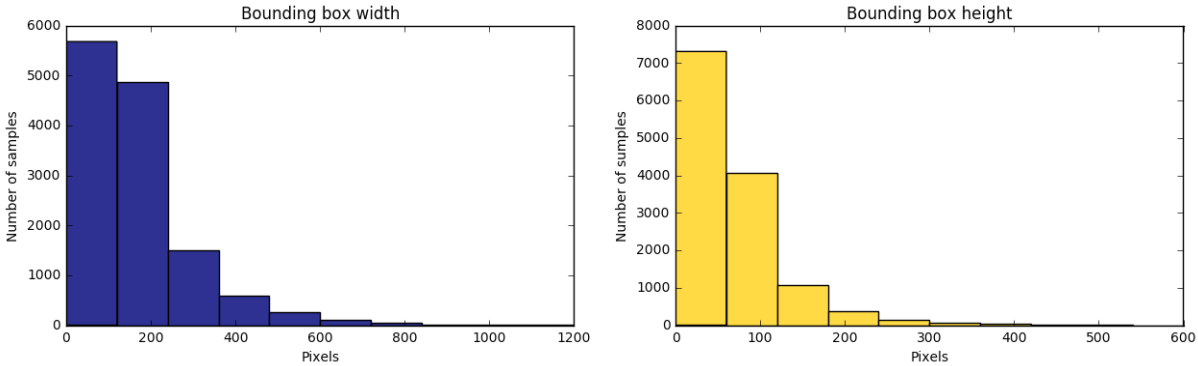

Figure 5. Test dataset's bounding box width and height

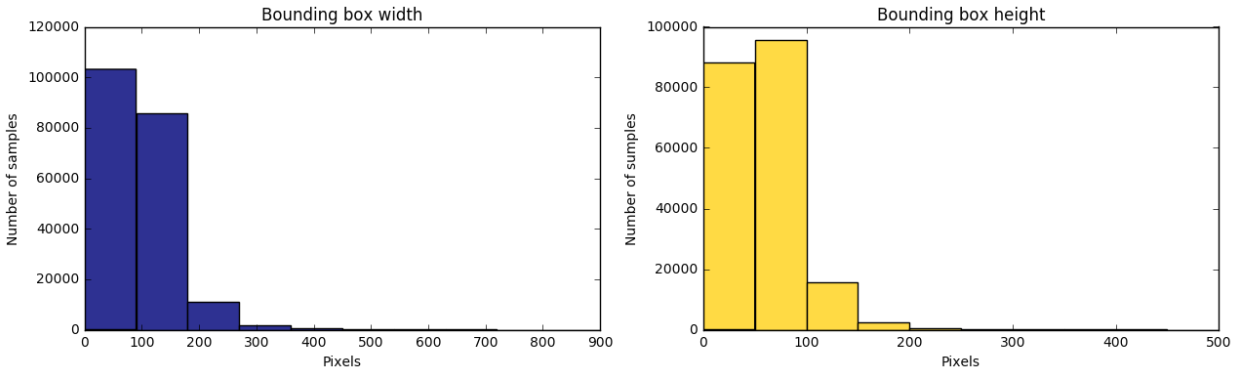

Figure 6. Extra dataset's bounding box width and height

Algorithms and Techniques

I am using 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:

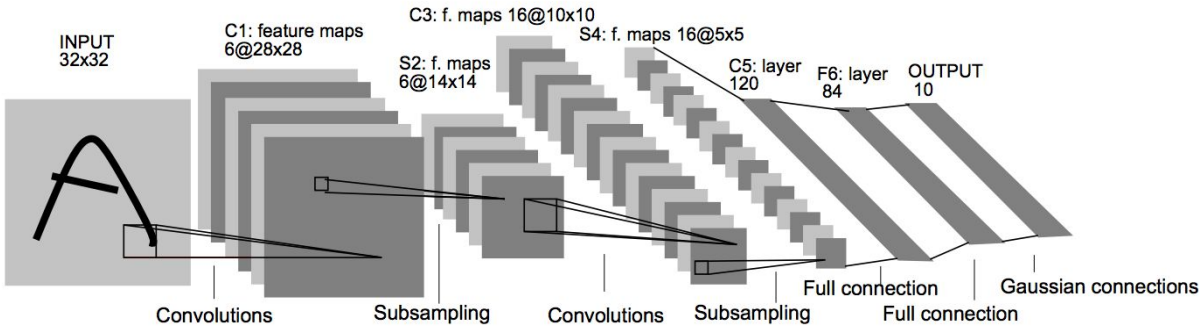

Figure 7. *Gradient-Based Learning Applied to Document Recognition*

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
   Pooling is a method of shrinking a large image while keeping important information, dividing the image into small windows, and taking the maximum value from each of the delimited windows. Since the maximum value is taken in each window, within the window, the degree of conformity of each feature is

maintained. Here, the exact position of the feature in the window is not important, but it is important that there is a feature somewhere in the window. As a result, CNN can find the location of the feature in the image without worrying about the exact position of the feature. When pooling is performed, the number of output images is the same, but each pixel number decreases. This is very convenient for managing the computational load, and if the 8 megapixel image is 2 megapixels, processing will be much easier downstream.

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

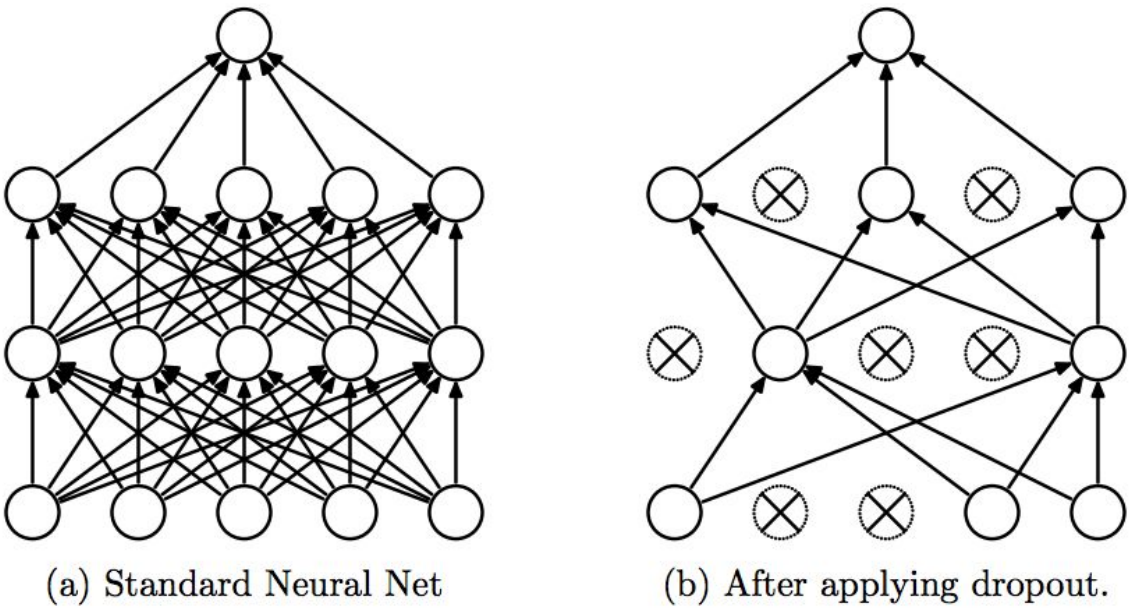

Figure 8. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*

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%.

In practical use, it will not be put into practical use unless it is the same performance as a human being or higher. In order to correctly recognize figures from real world images, I think that accuracy of 98% or more is necessary.

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. Data of 5 digits or more often has too many features or ambiguous images. In order to raise the accuracy of learning, it is narrowed down to 5 digits or less.

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

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

Table 2. Create validation dataset

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

The following is the image after the preprocessing is completed.

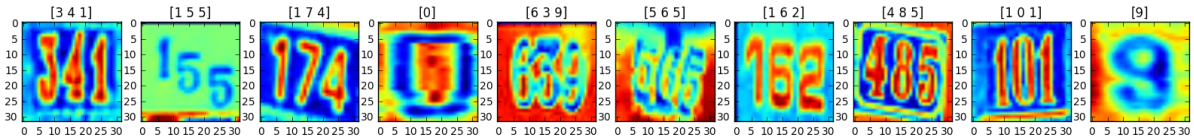

Figure 9. *Data preprocessing*

Implementation

This project uses Convolutional Neural Networks. Increasing the number of layers of CNN is more accurate than when the layer is thin.
The algorithm used here will be explained in the following order.

1. **Convolution layer 1**
   The algorithm begins with the first convolution layer with shape 5 x 5 x 1 x 6. It receives images has input.
   Imput size: 64(batch_size) x 32 x 32 x 1 → Output size: 28 x 28 x 1 x 16(depth1)

2. **Activation function**
   Next comes an activation function (Relu) which combines the first convolution output and the first layer biases. Output size: 28 x 28 x 1 x 16

3. **Pooling layer 1**
   The following layer is a pooling layer. The layer is effective in reducing dimensions. Output size is: 64 x 14 x 14 x 16

4. **Convolution layer 2**
   The second convolution layer recives a input with shape 14 x 14 x 16.
   The convolution has the shape 5 x 5 x 16 x 32 (depth2) → Output size is: 64 x 10 x 10 x 32

5. **Activation function**
   The second an activation function add the second convolution with the second biases. Output size is: 64 x 10 x 10 x 32

6. **Pooling layer 2**
   The second pooling layer reduce future the dimensions. Output size is: 64 x 5 x 5 x 32

7. **Convolution layer 3**
   The second convolution layer recives a input with shape 5 x 5 x 32.
   The convolution has the shape 5 x 5 x 32 x 64(num_hidden) → Output size is: 64 x 1 x 1 x 64

8. **Activation function**
   The third an activation function combine the last convolution layer with the third biases. Output size is: 64 x 1 x 1 x 64

9. **Dropout layer**
   Use the dropout function to prevent over learning. Here, the dropout rate is set to 50%.

10. **Reshape**
    The input is reshaped to the dimentions 64 x 64.

11. **Output layer**
    The output layer, an all conected layer recieve as input with shape 64 x 64.

12. **Output 5 logits (64 x 11)**

After the model is build, the dataset is converted to that Model in the Tensorflow Graph.

$$logits1,\ logits2,\ logits3,\ logits4,\ logits5\ =\ model(tf\_train\_dataset,\ True)$$

Next, the softmax cross entropies are calculated and reduced for the model against the labels using the tf.reduce_mean function.

```
loss = tf.reduce_mean(
    tf.nn.sparse_softmax_cross_entropy_with_logits(logits1, tf_train_labels[:,1])
  + tf.nn.sparse_softmax_cross_entropy_with_logits(logits2, tf_train_labels[:,2])
  + tf.nn.sparse_softmax_cross_entropy_with_logits(logits3, tf_train_labels[:,3])
  + tf.nn.sparse_softmax_cross_entropy_with_logits(logits4, tf_train_labels[:,4])
  + tf.nn.sparse_softmax_cross_entropy_with_logits(logits5, tf_train_labels[:,5]))
```

After that, the Gradient Descent Optimizer is then applied to minimize the loss.

```
learning_rate = tf.train.exponential_decay(0.5, global_step, 1000, 0.99)
```

Finally, the softmax function are calculated for the training dataset, testing dataset and validation dataset.

```
train_prediction = tf.pack([tf.nn.softmax(logits1), tf.nn.softmax(logits2),
tf.nn.softmax(logits3), tf.nn.softmax(logits4),tf.nn.softmax(logits5)])

valid_prediction = tf.pack([tf.nn.softmax(model(tf_valid_dataset)[0]),
tf.nn.softmax(model(tf_valid_dataset)[1]), tf.nn.softmax(model(tf_valid_dataset)[2]),
tf.nn.softmax(model(tf_valid_dataset)[3]), tf.nn.softmax(model(tf_valid_dataset)[4])])

test_prediction = tf.pack([tf.nn.softmax(model(tf_test_dataset)[0]),
tf.nn.softmax(model(tf_test_dataset)[1]), tf.nn.softmax(model(tf_test_dataset)[2]),
tf.nn.softmax(model(tf_test_dataset)[3]),  tf.nn.softmax(model(tf_test_dataset)[4])])
```

Refinement

In this model, adjustment of parameters is one of the important factors to improve accuracy.Below is a table with some of the parameters adjusted.

- **Bach size**

Another parameters:
patch_size = 5
image_size = 32
label_size = 6
num_channels = 1
num_labels = 11
depth1 = 16

depth2 = 32
num_hidden = 64
keep_prob = 0.5
learning_rate = tf.train.exponential_decay(0.15, global_step, 1000, 0.96)

batch_size	Minibatch loss at step 10000	Minibatch accuracy	Validation accuracy	Test accuracy
12	**7.353090**	**51.7%**	**60.4%**	**64.2%**
64	**2.245547**	**87.8%**	**84.3%**	**86.7%**

- ## Learning rate

Another parameters:
batch_size = 64
patch_size = 5
image_size = 32
label_size = 6
num_channels = 1
num_labels = 11
depth1 = 16
depth2 = 32
num_hidden = 64
keep_prob = 0.5
learning_rate = tf.train.exponential_decay(, global_step, 1000, 0.96)

learning_late	Minibatch loss at step 10000	Minibatch accuracy	Validation accuracy	Test accuracy
0.15	**2.400050**	**88.1%**	**84.7%**	**86.7%**
0.3	**6.439435**	**57.5%**	**60.4%**	**64.2%**
0.5	**6.437223**	**57.5%**	**60.4%**	**64.2%**

- ## Decay rate

Another parameters:

batch_size = 64
patch_size = 5
image_size = 32
label_size = 6
num_channels = 1
num_labels = 11
depth1 = 16
depth2 = 32
num_hidden = 64
keep_prob = 0.5
learning_rate = tf.train.exponential_decay(0.15, global_step, 1000,)

decay_rate	**Minibatch loss at step 10000**	**Minibatch accuracy**	**Validation accuracy**	**Test accuracy**
0.93	**2.306002**	**88.4%**	**85.4%**	**87.4%**
0.96	**2.245547**	**87.8%**	**84.3%**	**86.7%**
0.99	**2.400050**	**88.1%**	**84.7%**	**86.7%**

- **Decay steps**

Another parameters:
batch_size = 64
patch_size = 5
image_size = 32
label_size = 6
num_channels = 1
num_labels = 11
depth1 = 16
depth2 = 32
num_hidden = 64
keep_prob = 0.5
learning_rate = tf.train.exponential_decay(0.15, global_step, , 0.93)

decay_steps	**Minibatch loss at step 10000**	**Minibatch accuracy**	**Validation accuracy**	**Test accuracy**
1000	**2.306002**	**88.4%**	**85.4%**	**87.4%**

10000	**2.096426**	**88.4%**	**84.9%**	**86.9%**

- **Dropout**

Another parameters:
batch_size = 64
patch_size = 5
image_size = 32
label_size = 6
num_channels = 1
num_labels = 11
depth1 = 16
depth2 = 32
num_hidden = 64
learning_rate = tf.train.exponential_decay(0.15, global_step, 1000, 0.93)

keep_prob	Minibatch loss at step 10000	Minibatch accuracy	Validation accuracy	Test accuracy
0.5	**2.306002**	**88.4%**	**85.4%**	**87.4%**
0.8	**1.536305**	**92.8%**	**89.8%**	**91.1%**
0.9	**0.901736**	**96.2%**	**91.0%**	**92.3%**
0.95	**0.860656**	**96.9%**	**91.6%**	**92.9%**
0.98	**0.834543**	**95.0%**	**91.8%**	**93.1%**

Here describes the process of tuning individual parameters so far. In conclusion, adjustment of individual parameters gradually increased accuracy.

Individually, batch size 64 gained better accuracy than 32. If the batch size is 16, the accuracy was very low. In the learning rate, the accuracy was the highest when the numerical value was 0.15%, because the precision decreased when the learning rate was 0.3% or 0.5%. Decay rate was good at 0.93. Even if the numbers were increased to 0.96 and 0.99, there was only a slight change. Also, when decay step is 1000, the accuracy has improved more than 10000. Finally, increasing the dropout rate has made it possible to further improve accuracy in this model over other parameter adjustments.

Results

Model Evaluation and Validation

The best model I got is below.

Final Parameters :
batch_size = 64
patch_size = 5
image_size = 32
label_size = 6
num_channels = 1
num_labels = 11
depth = 16
num_hidden = 64
keep_prob = 0.98

num_step = 30001
learning_rate = tf.train.exponential_decay(0.15, global_step, 1000, 0.93)

Minibatch loss at step 10000	Minibatch accuracy	Validation accuracy	Test accuracy
0.853304	**92.8%**	**92.4%**	**93.6%**

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

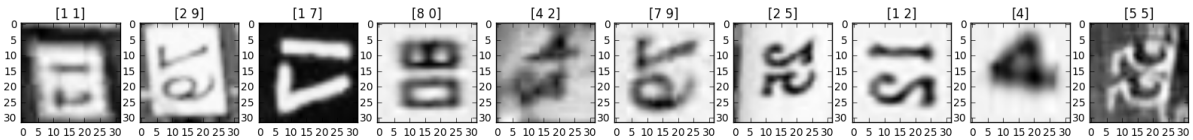

Figure 10. *Sample of test data*

By the time we get this result, we have looked at tuning various parameters. Two of the batch size and the learning rate can be found that if accuracy is not set precision will be extremely reduced. Regarding the dropout rate, when adjusting other parameters by adjusting, the precision which was around 85% increased significantly. With these, I was able to obtain the above result. Additionally, I think that when we have time, we can obtain better accuracy by increasing the number of steps. This is because accuracy will increase slightly for the same model.

Justification

The best model I build reaches 93.6% test accuracy.
As per my benchmark I was expecting accuracy of more than 98% because the human recognition rate of this data set was 98%. However, with this model its accuracy could not be reached. If I could use a GPU or have a lot of computers I will be able to further improve accuracy.

Conclusion

Free-Form Visualization

I tried visualizing this algorithm with Tensorboard. Tensorboard is a very handy tool that makes it easy to visualize how the model works. It is also useful during code adjustment. When making a mistake, even when the model does not fit well, it helps us to notice where the problem is. Let's visualize at the end of this project if the model is moving as expected.

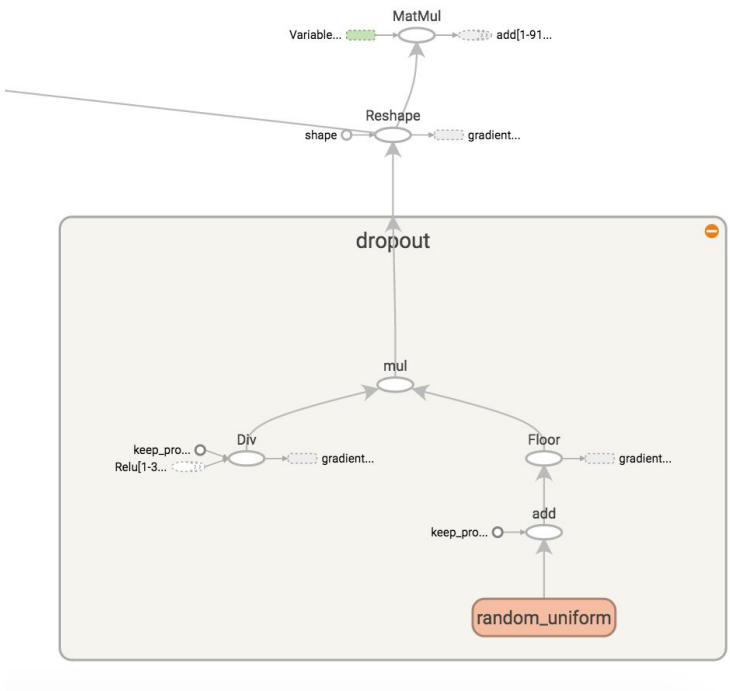

Figure 11. *Tensorboad visualization 1*

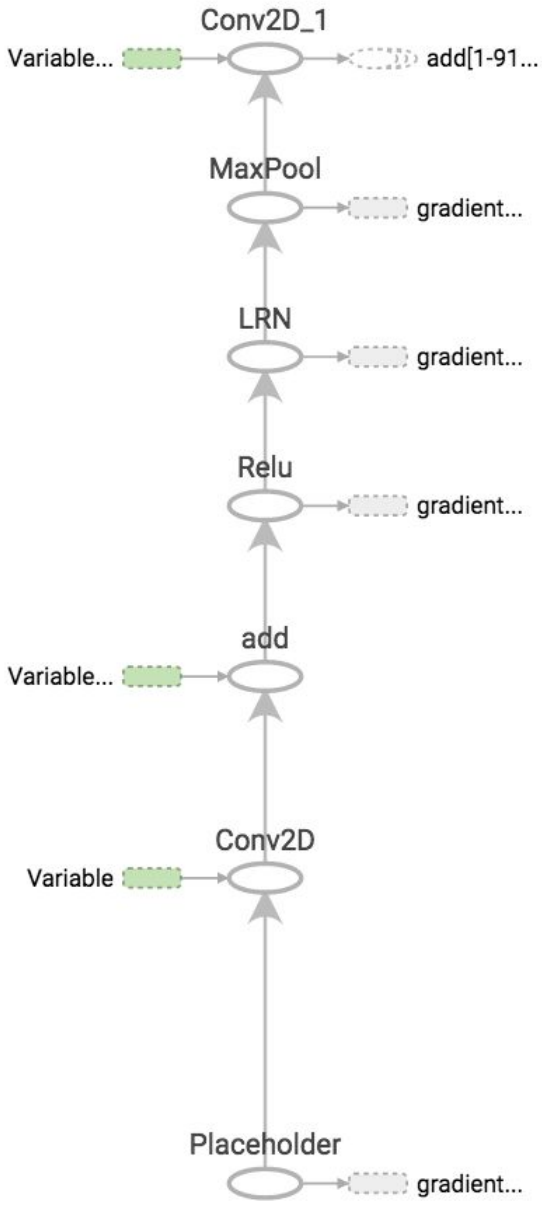

Figure 12. *Tensorboad visualization 2*

In the figure above, we can see that what is mentioned in the Implementation section is working correctly.

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.

I was suffering from preprocessing the data in this project. This data is a more complicated image than MNIST which truncates what exists in the real world, so I repeated it over and over. Further, it took time to adjust the parameters during the execution stage of the Tensorflow. Since the precision at the first execution was about 85%, we increased training number and lower learning rate than MNIST model. I tried various values such as batch size. What I learned is that adjustment of parameters is important in deep learning.
However, when the process succeeded and the accuracy of the model got better, I had a very pleasant experience!

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 therefore there may be more appropriate approaches to this dataset.
  For example, in order to improve the accuracy of image discrimination, it is necessary to adjust many parameters such as patch size, slide size, learning coefficient, iteration number, momentum. There may be combinations that are more accurate than I did.

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. Convolutional Neural Networks Applied to House Numbers Digit Classification
4. Reading Digits in Natural Images with Unsupervised Fature Learing - Deep Learning
5. Gradient-Based Learning Applied to Document Recognition
6. Convolutional Neural Networks
7. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*
8. Udacity Deep Learning Course
9. Udacity Forum
10. An overview of gradient descent optimization algorithms
11. Number plate recognition with Tensorflow