

# Machine Learning Engineer Nanodegree

## Capstone Project – Multi-digit Recognition

Raymond Ngiam  
March 31st, 2017

### I. Definition

#### Project Overview

Automatic identification of multi-character text from unconstrained natural images is a useful functionality in the domain of cartography.

With the advent of Google Street View, i.e. panorama images of many streets in the world acquired by cars, tricycles, boats, snowmobiles, and etc, there is an abundance of information that can be extracted to make our existing maps more informative.

In particular, most of the houses in the map, for example Google Map, are not numbered. We can identify and locate a street, a hospital, or a well known building like Times Square from Google Map, but we cannot pinpoint a specific house with its full address. One of the reasons for this is that it is too tedious to man-label all house numbers into a map.

With the availability of automatic identification of multi-character text from unconstrained natural images, labeling of individual houses can be automated using Street View images.

#### Problem Statement

The goal is to create an application that can interpret number strings in real-world images. In essence, this is a classification problem where number strings observed in a real world image are classified into classes range from 0 to 9.

Recognizing string of digits is substantially more difficult than identifying one digit per image. A model has to be trained such that it can decode sequences of digits from realistic natural images, where the digits are not neatly lined-up and have various skews, fonts and colors.

For this classification task, we will limit the maximum length of the number strings to five digits.

In addition to this, we will also tackle the problem of bounding box prediction for the individual digits detected. This is a regression task where the inputs are images of digit string and the outputs are top left corner coordinate (row, column), and size (width, height) of the bounding boxes, see Figure 1.

There are 4 parameters per bounding box. Since we have maximum of 5 digits in the digit string images, we will predict 5 sets of bounding box parameters, i.e. 20 regression outputs per image.

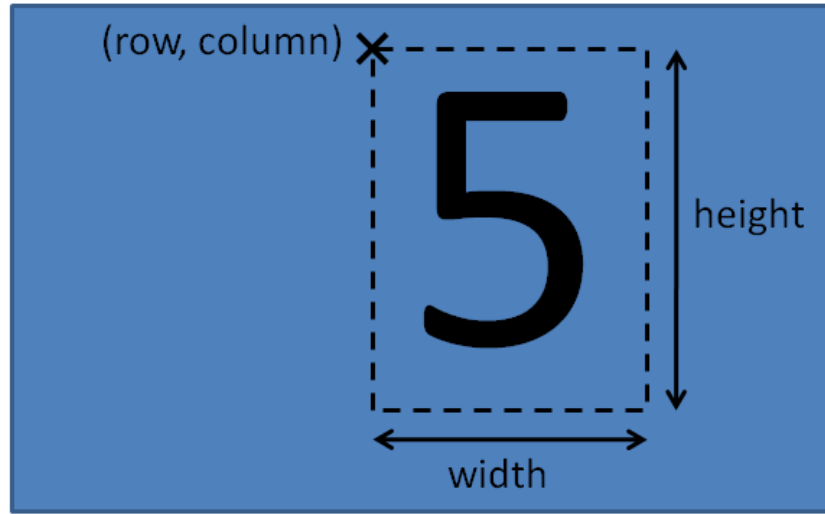


Figure 1 - Bounding box parameters.

## Metrics

One potential evaluation metric for the multi-digit classification task is accuracy. There are two types of accuracy in the context of this problem, i.e. character level accuracy and full sequence accuracy. For our case where maximum sequence length is limited to 5, these two metrics are given as follows:

$$Accuracy_{char} = \frac{\sum_{i=1}^N \sum_{j=1}^5 (digit_{i,j} == label_{i,j})}{5N}$$

$$Accuracy_{full} = \frac{\sum_{i=1}^N \prod_{j=1}^5 (digit_{i,j} == label_{i,j})}{N}$$

where  $N$  = number of digit string images

$digit_{i,j}$  =  $j$ -th predicted digit in the  $i$ -th digit string image

$label_{i,j}$  = ground truth for  $j$ -th digit in the  $i$ -th digit string image

$$(digit_{i,j} == label_{i,j}) = \begin{cases} 1, & \text{if } digit_{i,j} = label_{i,j} \\ 0, & \text{if } digit_{i,j} \neq label_{i,j} \end{cases}$$

Particularly, full-sequence accuracy will be used as the final evaluation metric both the benchmark model and our final model. This is a meaningful evaluation metric as it quantifies the capability of a model in terms of predicting the correct full sequence digits from an image.

We will not consider character level accuracy or “partial” accuracy, whereby some digits were correctly predicted on the image, but not all.

For the bounding box regression tasks, we will be using Intersection over Union (IoU) between the predicted and ground truth bounding boxes as the metric of performance. IoU for a single bounding box is given by the following definition, see Figure 2.



$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

Figure 2 - Intersection over Union definition. Image credit to Adrian Rosebrock's blog post "Intersection over Union (IoU) for object detection" [1]

For the case where the instances have multiple bounding boxes, we will take the average IoU of all the bounding boxes:

$$IoU_{multiple} = \frac{\sum_{i=1}^N \sum_{j=1}^{L_i} IoU_{i,j}}{\sum_{i=1}^N L_i}$$

where  $N$  = number of digit string images

$L_i$  = actual digit length of the  $i$ -th digit string image

$IoU_{i,j}$  = IoU of  $j$ -th digit in the  $i$ -th digit string image

## II. Analysis

### Data Exploration

The dataset to be used is the Street View House Numbers Dataset (SVHN) [2]. It is a large-scale dataset of house numbers in Google Street View images. It contains over 600,000 digit images.

In this project, we will be using the training and extra datasets which contain 33,402 and 202,353 digit sequence images respectively as the training data. For testing, we will use the test dataset which contains 13,000 images.

The following figures, Figure 3, 4 and 5, show the first 10 images from each dataset. Note that it is difficult, sometimes even for humans, to recognize the digits on them.



Figure 3 - First 10 images in training dataset, with label above each image.



Figure 4 - First 10 images in testing dataset, with label above each image.



Figure 5 - First 10 images in extra dataset, with label above each image.

The images in these datasets are not of uniform dimension. Table 1 below shows image dimension for the first 10 images in all three datasets:

#	Training Dataset Image Dimension (Width x Height)	Extra Dataset Image Dimension(Width x Height)	Test Dataset Image Dimension(Width x Height)
1	741 x 350	166 x 141	99 x 47
2	199 x 83	295 x 261	182 x 48
3	52 x 23	137 x 96	101 x 31
4	161 x 79	79 x 50	75 x 31
5	140 x 68	81 x 50	215 x 81
6	74 x 35	90 x 44	94 x 43
7	99 x 54	63 x 44	99 x 38
8	54 x 22	88 x 54	133 x 62
9	79 x 34	47 x 43	94 x 31
10	74 x 37	52 x 41	87 x 55

Table 1 - Image dimension for the first 10 images for all three datasets.

The digit string length frequency for the datasets is summarized in Table 2 below:

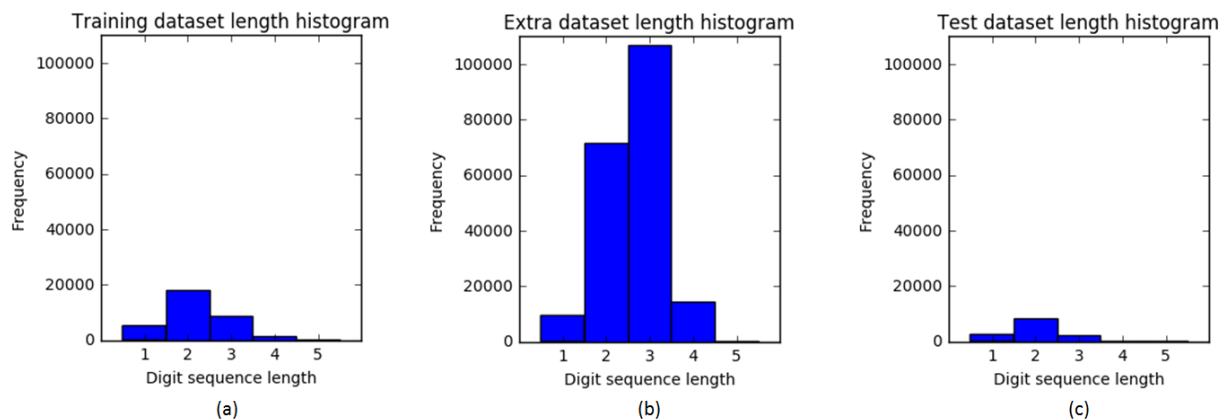
Digit Sequence Length	Training Dataset Frequency	Extra Dataset Frequency	Testing Dataset Frequency
1	5137	9385	2483
2	18130	71726	8356
3	8691	106789	2081
4	1434	14338	146
5	9	115	2
6	1	0	0

*Table 2 - Digit length frequency for all three datasets.*

Out of the three datasets, there is one instance with digit sequence length of 6 in the training dataset. Hence, we will exclude this data point from the training dataset.

## Exploratory Visualization

Let's visualize length distribution of the three datasets via histogram, see Figure 6.



*Figure 6 - Histogram of digit sequence length in (a) training dataset, (b) extra dataset, and (c) test dataset.*

Apparently, the distributions of digit length for all three datasets are not uniform. They appear to be skewed to the right with high density in length 2 and 3.

## Algorithms and Techniques

The approach to be used in tackling this problem is the Convolutional Neural Network (CNN), which is the state-of-the-art method for image classification.

CNN is effective for the application of image classification due to one reasonable assumption made, i.e. translation invariance of useful features. In other words, if one feature is useful to compute at some spatial position  $(x_1, y_1)$ , then it should also be useful to compute at a different position  $(x_2, y_2)$  [3].

This is achieved by sharing parameters across space. A convolutional layer in a CNN typically has many depth slices, whereby each depth slice represents one feature computed by convolution with input from previous layer.

For example, a color image of size  $[256 \times 256 \times 3]$  is fed into a convolutional layer with patch size of  $[5 \times 5]$ , depth of 16 and at stride 2, the output of this layer is  $[128 \times 128 \times 16]$ , see Figure 7. The convolutional layer's output has 16 depth slices. Each of these depth slices has 76 parameters ( $5 \times 5 \times 3 = 75$  weights and 1 bias). The entire convolutional layer has only 1216 parameters ( $16 \times 5 \times 5 \times 3 = 1200$  weights and 16 biases).

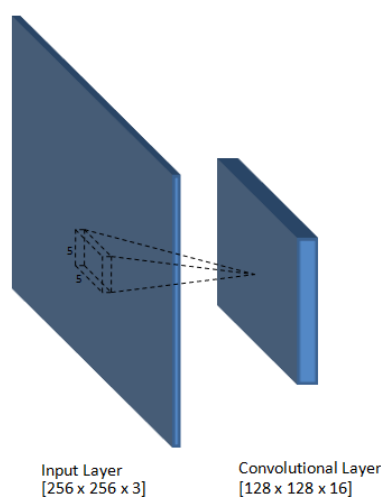


Figure 7 - A color image of size  $[256 \times 256 \times 3]$  is fed into a convolutional layer with patch size of  $[5 \times 5]$ , depth of 16 and at stride 2.

This in turns also significantly reduces the number of parameter compared to a regular fully connected layer which has over 51 billion parameters in the same configuration. Thus, this makes training a large CNN feasible in real applications.

## Benchmark

Benchmark model to be used is the model by Goodfellow, et al. (2013) [4], which reported a full sequence accuracy of 96.03% and a character-level accuracy of 97.84%.

## III. Methodology

### Data Preprocessing

The images available in the SVHN datasets are color images. Since color is not a factor that distinguishes different digits, we will transform the color images into gray images with the following transformation:

$$G(x, y) = 0.299 R(x, y) + 0.587 G(x, y) + 0.114 B(x, y)$$

The above transformation weights the color components similar to human visual perception, which gives higher weight for the green component. The output image is a gray image as shown in the figure below:



*Figure 8 - (a) An example of color image in the dataset,  
(b) Transformed gray image.*



For the training of neural networks, normalized training input with zero mean and equal variance would make the optimization problem well conditioned, thus improve the optimization performance.

Hence, we will subtract each image with its own mean. Figure below shows the gray histogram of an example image before and after this transformation.

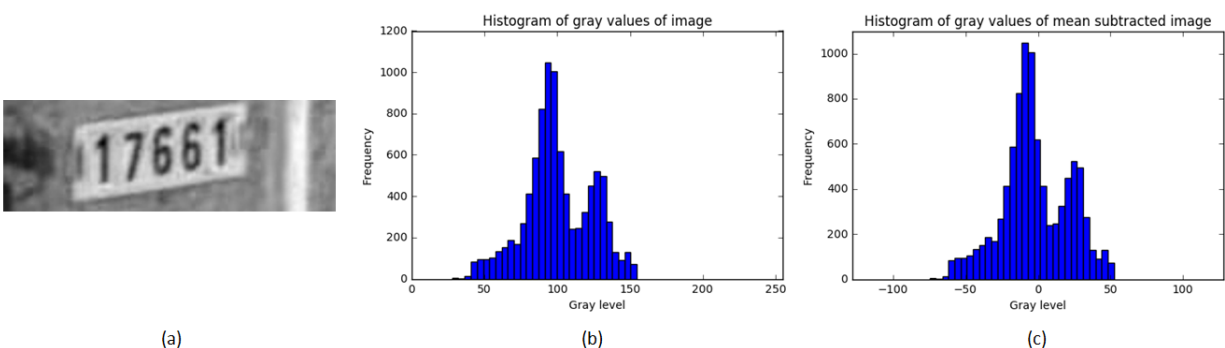


Figure 9 - (a) Gray image, (b) Gray histogram for the image, and (c) Gray histogram for the mean subtracted image.

As we saw in Data Exploration section, images in the SVHN datasets are of varying dimensions. However, input of a convolutional network must be of a consistent image dimension. Hence, we must resize all the images to a fixed image dimension.

The image resizing procedure we used was inspired by Goodfellow et al. [4]. The procedure goes as follows:

1. Combine the multiple bounding boxes into one big bounding box and the big bounding box is expanded by 30% in width and height.

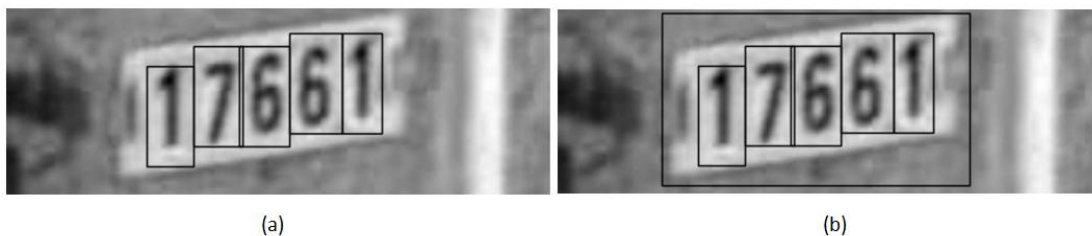


Figure 10 - (a) Gray image with character level bounding boxes, (b) Gray image with character level bounding boxes and an expanded big bounding box.

2. Crop the image based on the expanded big bounding box and resize the image to dimension (64, 64). Note that the bounding boxes are updated as the images are being cropped and resized.

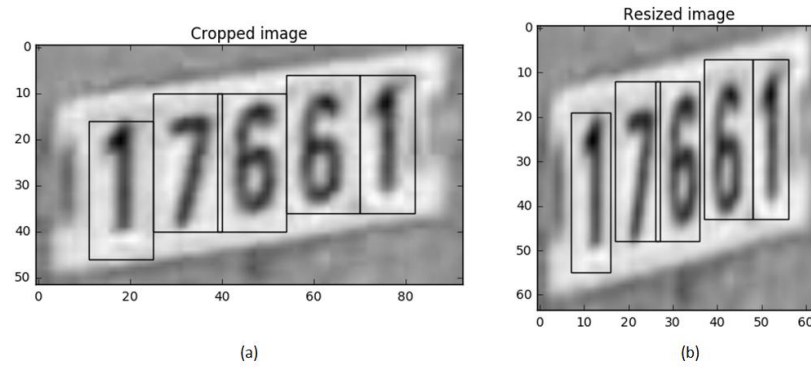


Figure 11 - (a) Cropped image with character level bounding boxes, (b) Resized image with character level bounding boxes.

3. Randomly crop images of dimension (54, 54) from the resized image.

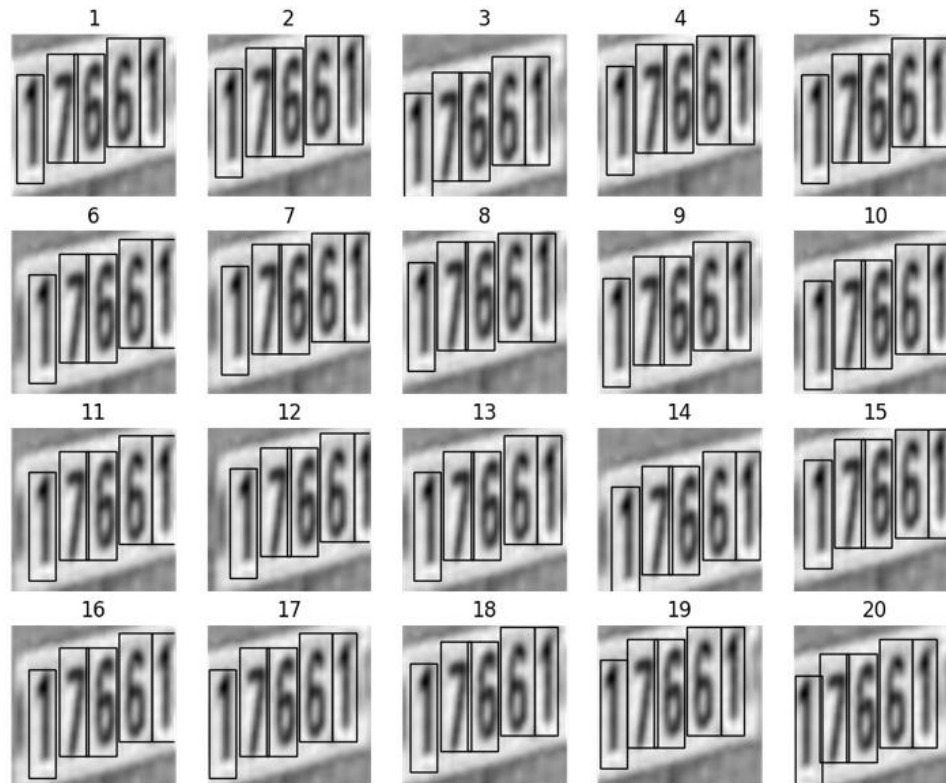
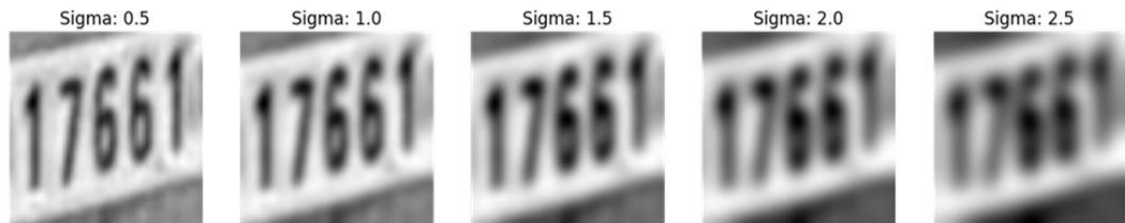


Figure 12 - 20 randomly cropped images of dimension (54, 54) from a resized image which is of dimension (64, 64).

The rationale behind these steps is for oversampling the same image with slight translational variations. This is particularly useful if certain label classes have smaller sample size relative to the rest.

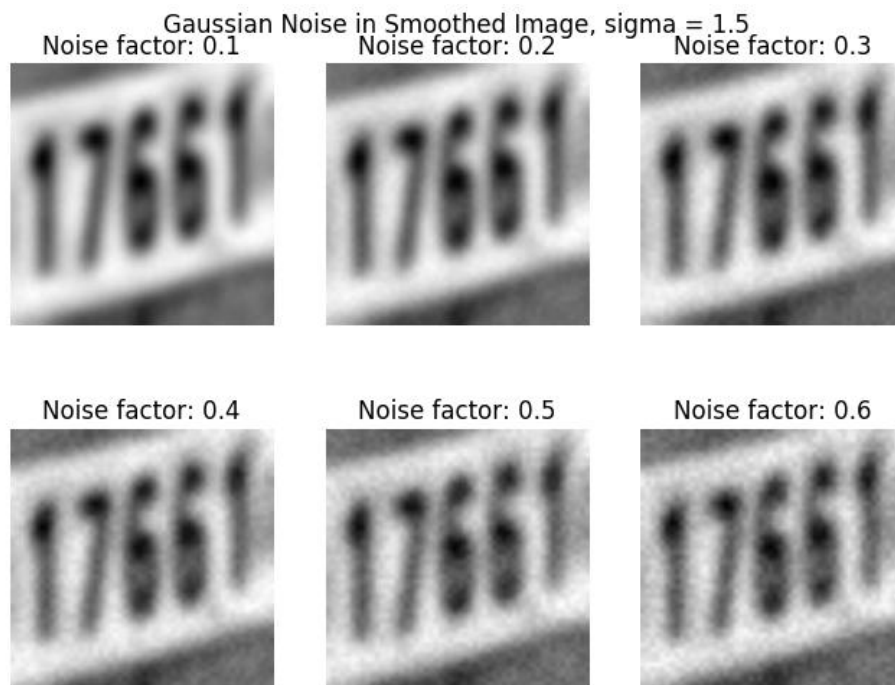
In addition to the procedure listed above, we added the following steps:

1. Add Gaussian smoothing to the randomly cropped image to simulate different extents of blurriness due to improper focusing of the camera.



*Figure 13 - Gaussian smoothing of a gray image at 5 different sigma values, i.e. 0.5, 1.0, 1.5, 2.0, and 2.5.*

2. Add Gaussian noise to the smoothed image to simulate sampling noises of the camera sensor.



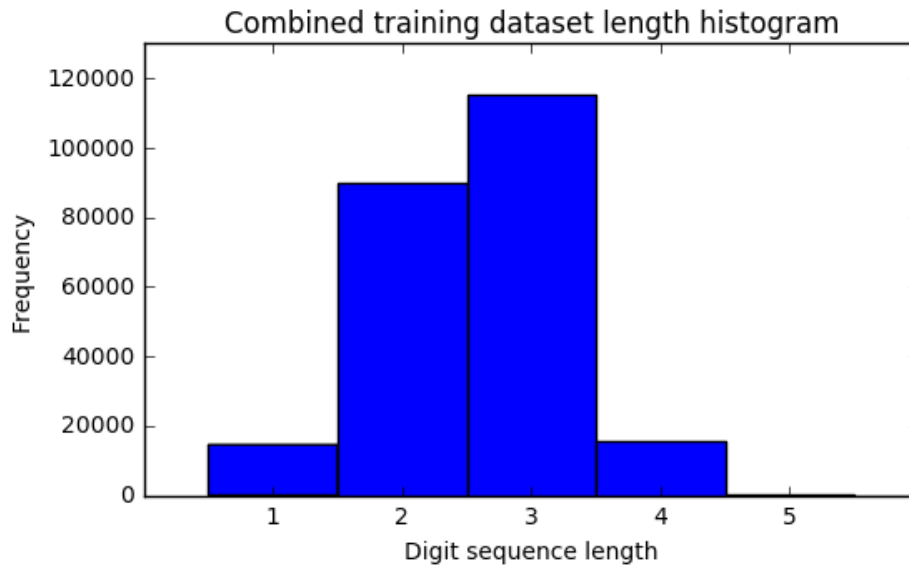
*Figure 14 - Gaussian noise is added to a gray image smoothed at sigma = 1.5 at 6 different noise factors, i.e. 0.1, 0.2, 0.3, 0.4, 0.5 and 0.6.*

### 3. Further resize the image to dimension (32, 32)

We will use the training and extra dataset in SVHN dataset as the training data for our model. The two datasets combined to a total of 235,754 samples. However, as mentioned in the Data Exploration section, the digit length distribution of these datasets is not uniformly distributed. The table and figure below show the distribution of digit length for the combined training dataset.

Digit Sequence Length	Combined Training Dataset Frequency
1	14522
2	89856
3	115480
4	15772
5	124

*Table 3 - Digit length frequency for the combined training dataset.*



*Figure 15 – Histogram of digit length for the combined training dataset.*

This will cause difficulties for our model in learning the digit length of the data due to label class imbalance.

We will handle this by performing data sampling according to the image processing procedure mentioned above. Particularly, we perform under and over sampling from the combined training data as depicted in Table 4. The length distribution of the final training data is as shown in Figure 16.

Digit Sequence Length	Combined Training Dataset Frequency	Translation variation	Blurriness variation [sigma]	Noise variation [noise factor]	Sampling	Final Training Dataset Frequency
1	14522	2	-	-	-	29044
2	89856	1	-	-	30000	30000
3	115480	1	-	-	30000	30000
4	15772	2	-	-	-	31544
5	124	40	2 [1.5, 2.0]	3 [0.2, 0.3, 0.4]	-	29760

Table 4 – Sampling plan for the final training dataset.

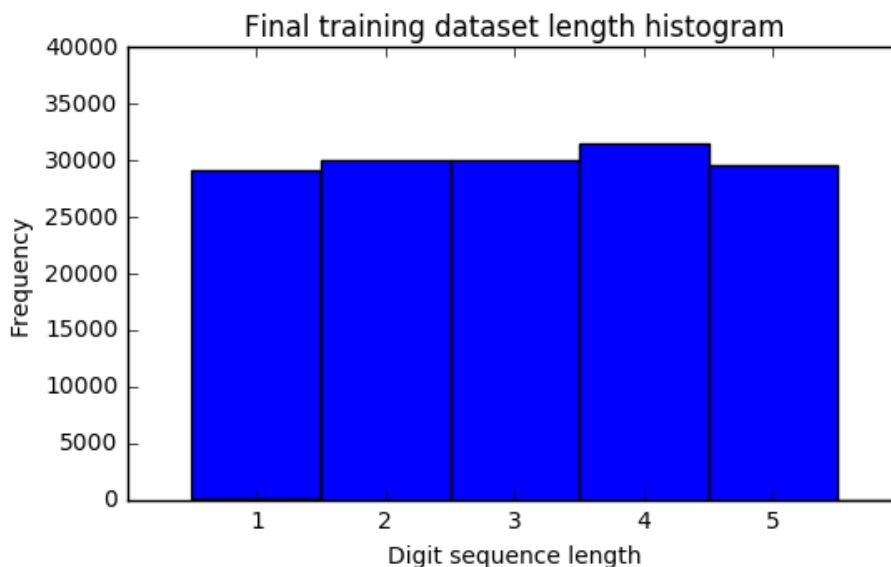
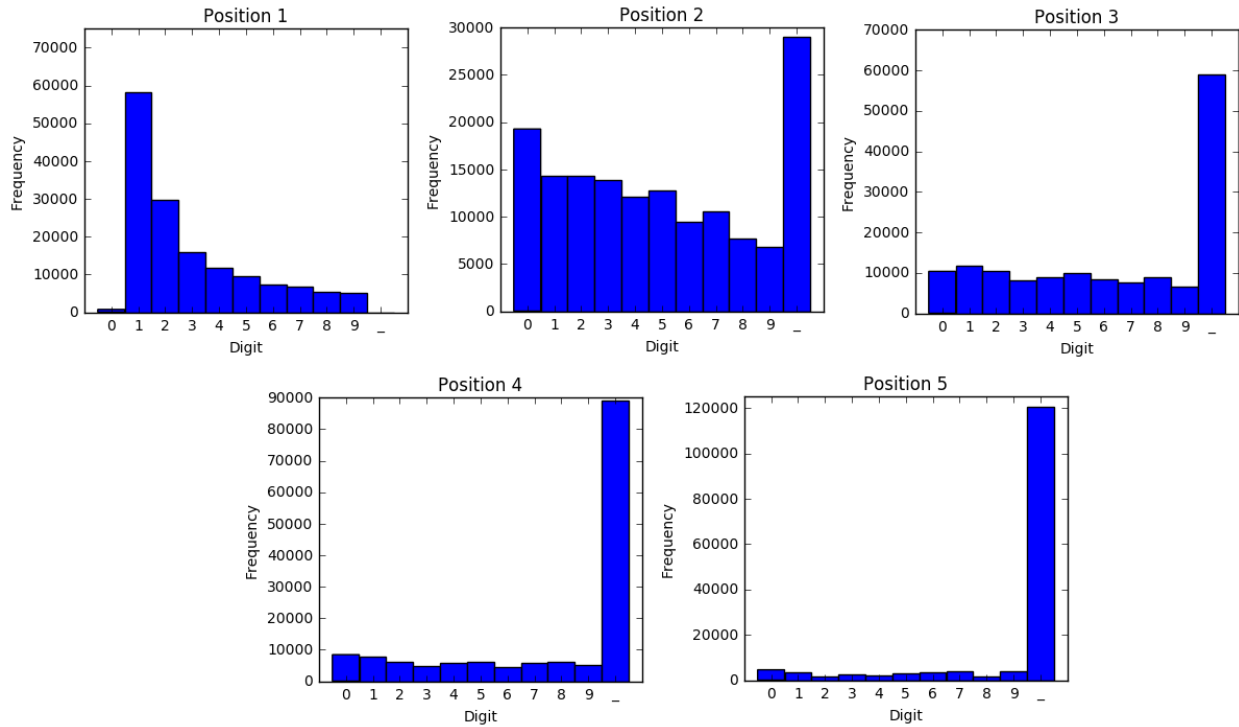


Figure 16 – Histogram of digit length for the final training dataset.

The SVHN datasets contains digit labels of varying lengths. We convert these labels into consistent sized arrays of 5 digit positions. Each digit position can take 11 possible values ('0' to '9', plus a null digit, '\_'). Figure 17 below shows histograms of digit label for each digit position.



*Figure 17 – Histogram of digit label for each of the 5 digit positions.*

Note that except for digit position 1, the four other digit positions have high density in the null digit label and this density increases by multiples of 30000 from digit position 2 to digit position 5. This is inevitable due to the structure of our data labels.

For example, consider we have only data with digit length 5 initially, all 5 digit positions should not have any null digit. If we then append data with digit length 4 to this initial data, the first 4 digit positions of the entire data up to this point should not have any null digit. However, for digit position 5, there will be an addition of null digits equal to number of digit length 4 samples added.

This issue of heavily imbalanced class labels in one class, i.e. the null digit, can cause difficulties in the training the deep convolutional network for classification of the digits. We will address this issue in the Implementation section.

If we ignore the null digit from the analysis, the digit label distribution for position 3, 4, and 5 are close to uniformly distributed among the 10 class labels, i.e. '0' to '9'.

For digit position 1 and 2, the digit label distributions are not uniform, and skewed to the right. Particularly, digit label distribution for digit position 1 is more heavily skewed and the number of sample with class label '0' is only 738 compared to the class label '1' which we have 58,105 samples.

Hence, we expect the classification rate for digit position 1 and 2 of the final model to be poorer compare to the other 3 digit positions.

## Implementation

Our model architecture consists of two major components, namely the deep convolutional layers, and the fully connected layers.

The deep convolutional layer consists of 7 layers without considering the flattening layer as depicted below:

```
INPUT: [32x32x1]
LAYER 1: INPUT -> CONV-40-5 -> ReLU -> BatchNorm -> MAXPOOL-2-2 -> DROPOUT-0.3 -> [16x16x40]
LAYER 2: LAYER 1 -> CONV-80-5 -> ReLU -> BatchNorm -> DROPOUT-0.3 -> [16x16x80]
LAYER 3: LAYER 2 -> CONV-80-5 -> ReLU -> BatchNorm -> MAXPOOL-2-2 -> DROPOUT-0.3 -> [8x8x80]
LAYER 4: LAYER 3 -> CONV-160-5 -> ReLU -> BatchNorm -> DROPOUT-0.3 -> [8x8x160]
LAYER 5: LAYER 4 -> CONV-160-5 -> ReLU -> BatchNorm -> MAXPOOL-2-2 -> DROPOUT-0.3 -> [4x4x160]
LAYER 6: LAYER 5 -> CONV-240-3 -> ReLU -> BatchNorm -> DROPOUT-0.3 -> [4x4x240]
LAYER 7: LAYER 6 -> CONV-360-3 -> ReLU -> BatchNorm -> MAXPOOL-2-2 -> DROPOUT-0.3 -> [2x2x360]
FLATTEN: [1440]

with:
    CONV-[depth]-[patch]
    MAXPOOL-[patch]-[stride]
    DROPOUT-[dropout rate]
```

In general, each layer starts with a convolution operation followed by a Rectified Linear Unit (ReLU) activation. Batch normalization [5] is then applied. The purpose of batch normalization is to ensure the outputs to all subsequent layers are well centered around zero with equal variance. It is essential as it makes the optimization problem well conditioned and thus it allows the network to learn at much higher learning rates.

Batch normalization usually has two trainable parameters which are scale and offset. Since we are using only ReLU activation function, the batch normalization will only has one trainable parameter, i.e. the offset, since scaling operation does not affect the output of a ReLU activation function.

Max pooling with patch 2x2 at stride = 2 is carried out at every odd number layer, i.e. layer 1, 3, 5 and 7. Lastly, dropout is applied at the end of every layer with dropout rate of 0.3. See Figure 18 for an illustration of the deep convolutional layers.

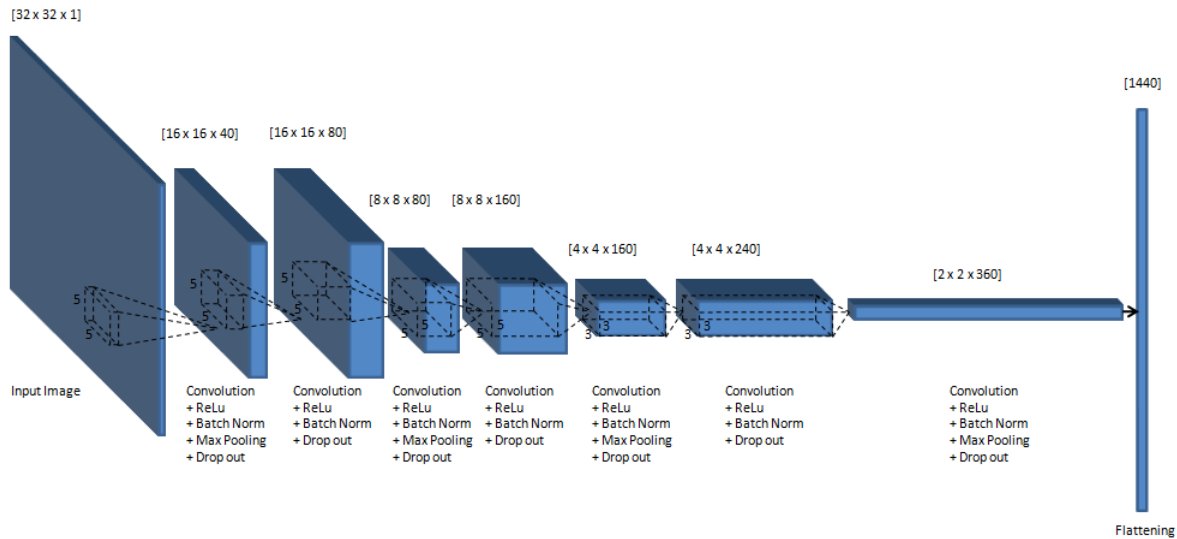


Figure 18 – Deep convolutional layers.

In the fully connected layers, the flatten output from the deep convolutional layers are connected to 2 separate hidden layers.

```

FLATTEN: [1440]
LAYER 8-1: FLATTEN -> FC-1000 -> ReLU -> BatchNorm -> DROPOUT-0.5 -> [1000]
LAYER 8-2: FLATTEN -> FC-1000 -> ReLU -> BatchNorm -> DROPOUT-0.5 -> [1000]
LAYER 9-1: LAYER 8-1 -> FC-5 -> Softmax -> [5]
LAYER 9-2: LAYER 8-1 -> FC-11 -> Softmax -> [11]
LAYER 9-3: LAYER 8-1 -> FC-11 -> Softmax -> [11]
LAYER 9-4: LAYER 8-1 -> FC-11 -> Softmax -> [11]
LAYER 9-5: LAYER 8-1 -> FC-11 -> Softmax -> [11]
LAYER 9-6: LAYER 8-1 -> FC-11 -> Softmax -> [11]
LAYER 9-7: LAYER 8-2 -> FC-20 -> [20]

```

with:

```

FC-[output node]
DROPOUT-[dropout rate]

```

All these hidden layers have the same configuration. Initially the input nodes will be fully connected to the output nodes with some weights plus biases. Then, ReLU activation is applied, followed by batch normalization. Lastly, dropout is applied with dropout rate of 0.5.

The first hidden layer is fully connected to two output layers. The first output layer is used for the digit length prediction. It consists of 5 output nodes with softmax activation function. The second output layer is used for the digit classification of the 5 digit positions. It consists of 5 different clusters whereby each cluster has 11 output nodes with softmax activation.



The second hidden layer is fully connected to 20 output nodes without any activation function. These 20 output nodes are for the prediction of bounding box locations and sizes. Each of the 5 digit positions has a bounding box where each bounding box has 4 parameters, i.e. top left corner coordinate (row, column) and size (width, height). Hence this amounts to a total of 20 output values. See Figure 19 for an illustration of the fully connected layers and output layers.

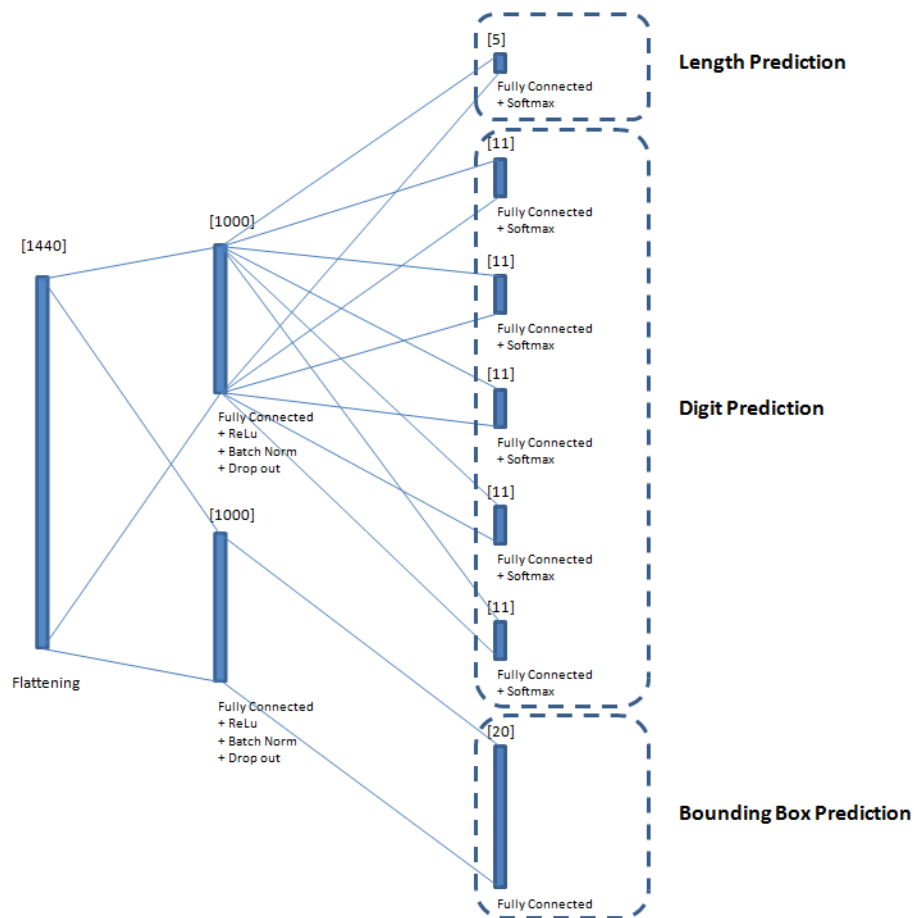


Figure 19 – Fully connected layers.

The weight initialization [6] is used to initialize all weights in the deep convolutional and fully connected layers before the training. This procedure initializes weights by random sampling from a normal distribution of mean 0 and standard deviation  $\sqrt{2/\text{fan\_in}}$ .

Here, fan\_in is number of inputs. For a convolutional layer, it is equal to  $\text{patch\_size} \times \text{patch\_size} \times \text{input\_depth}$ . For a fully connected layer, it's equal to number of input nodes.

Next, we will discuss about the loss functions used for the training of our model. The loss function can be divided into three parts, i.e. loss for length prediction, loss for digit prediction and loss for bounding box prediction.

Firstly, the batch loss for length prediction is computed as the average cross entropy of predicted length given as follows:

$$loss_{length} = \frac{1}{N} \sum_{i=1}^N -\log P(l_i = L_i)$$

where  $N$  = number of digit string images in a batch

$l_i$  = predicted length for the  $i$ -th digit string image

$L_i$  = actual length for the  $i$ -th digit string image

$P(l_i = L_i)$  = probability of  $l_i$  equal  $L_i$

The batch loss for digit prediction is slightly more complicated. As mentioned in the Data Preprocessing section, we have highly imbalanced class label for the null digit, '\_' at digit position 2, 3, 4 and 5. With this in mind, we will only compute the digit prediction loss for digits within the ground truth digit length. Thus, we will ignore all the null digit labels when computing the digit prediction loss.

For each digit string image input, we compute the sum cross entropy over the available digit according to the ground truth digit length. We then compute the average over the total number of digit string images.

$$loss_{digits} = \frac{1}{N} \sum_{i=1}^N \sum_{j=1}^{L_i} -\log P(d_{i,j} = D_{i,j})$$

where  $N$  = number of digit string images in a batch

$L_i$  = actual length for the  $i$ -th digit string image

$d_{i,j}$  = predicted  $j$ -th digit for the  $i$ -th digit string image

$D_{i,j}$  = actual  $j$ -th digit for the  $i$ -th digit string image

$P(d_{i,j} = D_{i,j})$  = probability of  $d_{i,j}$  equal  $D_{i,j}$

Lastly, the batch loss for bounding box prediction is also computed in similar manner as for the batch loss for digit prediction. However the error measure we used here is the root mean square error instead of cross entropy.

$$loss_{bbox} = \sqrt{\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^{L_i} \text{reduce\_sum}((\mathbf{b}_{i,j} - \mathbf{B}_{i,j})^2)}$$

where  $N$  = number of digit string images in a batch

$L_i$  = actual length for the  $i$ -th digit string image

$\mathbf{b}_{i,j}$  = predicted bounding box for  $j$ -th digit of the  $i$ -th digit string image, vector containing 4 elements [left, top, width, height]

$\mathbf{B}_{i,j}$  = actual bounding box for  $j$ -th digit of the  $i$ -th digit string image, vector containing 4 elements [left, top, width, height]

$\text{reduce\_sum}((\mathbf{b}_{i,j} - \mathbf{B}_{i,j})^2)$  = sum over all elements in vector  $(\mathbf{b}_{i,j} - \mathbf{B}_{i,j})^2$

The final loss function is then defined as the summation of these three losses:

$$loss_{final} = loss_{length} + loss_{digit} + loss_{bbox}$$

The model is trained with an Adam Optimizer [7] using the final loss function. We use the default parameters for this optimizer as in Tensorflow:

Learning rate: 0.001

Beta 1: 0.9

Beta 2: 0.999

Before the training of the model, we perform a random sampling of 500 samples from the final training samples of size 150,348 to form a validation dataset. This left us with 149,848 training data.

The random sampling for validation dataset is done in stratified manner where the distribution of digit length in the validation dataset is similar to the original dataset.

We train the model at batch size of 64 for 30 epochs of the training data.

During training, we reshuffle the training data after each epoch. Note that without reshuffling, the batches of training sample that are used for training

will be nearly the same over every epoch. Hence, there is no much variation between the batches of training sample. By adding a random shuffle between each training epoch, we introduce variation among the training sample batches over the epochs.

At every 500 steps, we check the full sequence accuracy and IoU of the held out validation dataset. If one of the metrics is improved, we will save the graph as a checkpoint. Note that we save the checkpoints for the best full sequence accuracy and IoU in two different files. We will use the checkpoint with the best full sequence accuracy for digit prediction (length and digits), and checkpoint with the best IoU for bounding box prediction.

## Refinement

In the deep convolutional layers, we increase the convolution depth in the last two layers to further improve the feature extraction capability of the network. For layer 6, the depth is increased from 240 to 480. For layer 7, the depth is increased from 360 to 720.

```
INPUT: [32x32x1]
LAYER 1: INPUT -> CONV-40-5 -> ReLU -> BatchNorm -> MAXPOOL-2-2 -> DROPOUT-0.3 -> [16x16x40]
LAYER 2: LAYER 1 -> CONV-80-5 -> ReLU -> BatchNorm -> DROPOUT-0.3 -> [16x16x80]
LAYER 3: LAYER 2 -> CONV-80-5 -> ReLU -> BatchNorm -> MAXPOOL-2-2 -> DROPOUT-0.3 -> [8x8x80]
LAYER 4: LAYER 3 -> CONV-160-5 -> ReLU -> BatchNorm -> DROPOUT-0.3 -> [8x8x160]
LAYER 5: LAYER 4 -> CONV-160-5 -> ReLU -> BatchNorm -> MAXPOOL-2-2 -> DROPOUT-0.3 -> [4x4x160]
LAYER 6: LAYER 5 -> CONV-480-3 -> ReLU -> BatchNorm -> DROPOUT-0.3 -> [4x4x480]
LAYER 7: LAYER 6 -> CONV-720-3 -> ReLU -> BatchNorm -> MAXPOOL-2-2 -> DROPOUT-0.3 -> [2x2x720]
FLATTEN: [2880]
```

with:

```
CONV-[depth]-[patch]
MAXPOOL-[patch]-[stride]
DROPOUT-[dropout rate]
```

For the two fully connected layers, namely layer 8-1 and layer 8-2, we increase the number of output node from 1000 to 3000. At the same time, we also increase the dropout rate from 0.5 to 0.8. This is to further regularize the model so that it can learn better.

```
FLATTEN: [2880]
LAYER 8-1: FLATTEN -> FC-3000 -> ReLU -> BatchNorm -> DROPOUT-0.8 -> [3000]
LAYER 8-2: FLATTEN -> FC-3000 -> ReLU -> BatchNorm -> DROPOUT-0.8 -> [3000]
LAYER 9-1: LAYER 8-1 -> FC-5 -> Softmax -> [5]
LAYER 9-2: LAYER 8-1 -> FC-11 -> Softmax -> [11]
LAYER 9-3: LAYER 8-1 -> FC-11 -> Softmax -> [11]
LAYER 9-4: LAYER 8-1 -> FC-11 -> Softmax -> [11]
LAYER 9-5: LAYER 8-1 -> FC-11 -> Softmax -> [11]
LAYER 9-6: LAYER 8-1 -> FC-11 -> Softmax -> [11]
LAYER 9-7: LAYER 8-2 -> FC-20 -> [20]
```

with:

```
FC-[output node]
DROPOUT-[dropout rate]
```

In addition to the architectural changes, we also reduce the learning rate from 0.001 to 0.0001. The rationale behind this change is that we intend to train the refined model with more epochs, and we do not wish the model's learning ability to be saturated due to high learning rate.

Lastly, we train the model in multiple sessions of 30 epochs before the model starts to overfit.

## IV. Results

### Model Evaluation and Validation

For the sake of simplicity, from here onwards, we will name the initial model as Model 1, and the refined model as Model 2.

Figure 20 shows the graph visualization of our models using TensorBoard. Model 1 and 2 have the same overall graph representation as they only differ internally in convolution layer 6, 7 and the two fully connected layers.

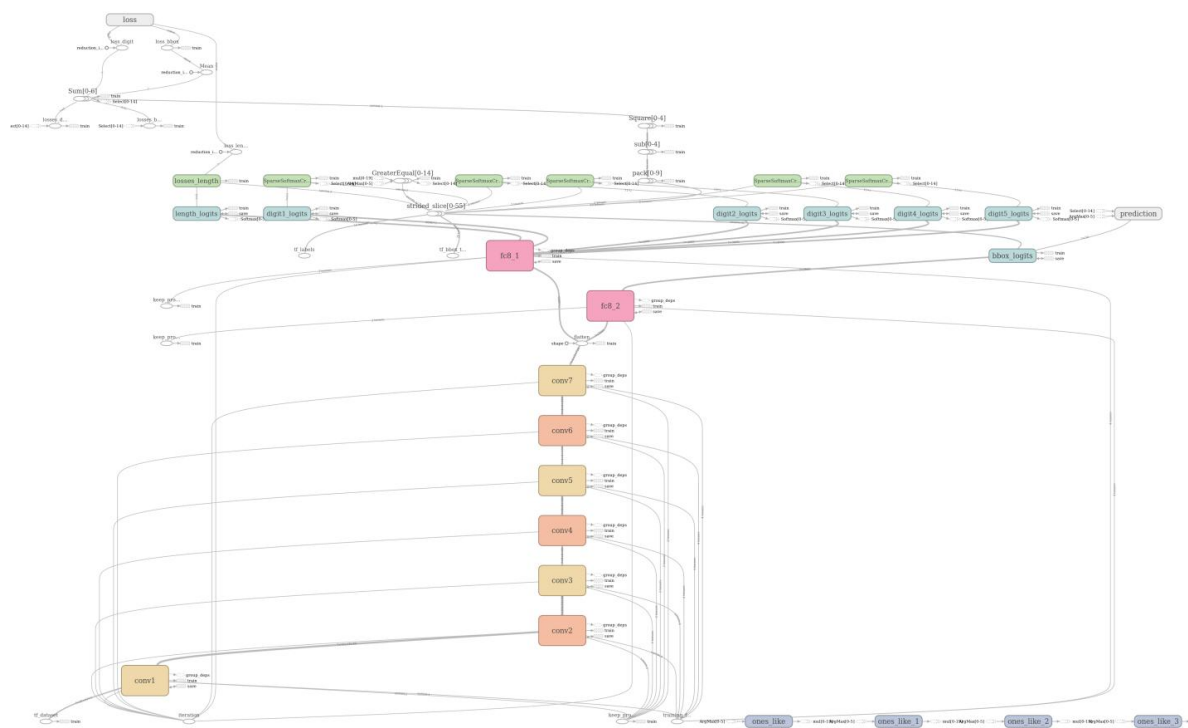


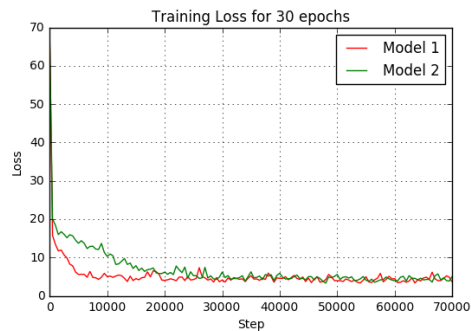
Figure 20 – Model's graph visualization using TensorBoard.

Table 5 shows the performance of the two models with test data, after being trained for 30 epochs.

	Model 1 (30 epochs)	Model 2 (30 epochs)
<b>Test Full Sequence Accuracy</b>	91.02 %	89.52 %
<b>Test IoU</b>	79.10 %	78.30 %

*Table 5 – Performance of Model 1 and 2 with the test data after 30 epochs of training.*

The training loss of both models for 30 epochs is shown in Figure 21.



*Figure 21 – Training loss of Model 1 and 2 for 30 epochs (1 epoch  $\approx$  2,341 steps).*

Note that for Model 2 with a lower learning rate (0.0001), the minibatch training loss reduces at a much slower rate compared to Model 1. The training loss for Model 1 starts to saturate after 10,000 steps (approximately 4.27 epochs), whereas the training loss for Model 2 starts to saturate after 30,000 steps (approximately 12.81 epochs).

Figure 22 and 23 show the full sequence accuracy and IoU performance of the two models for 30 epochs.

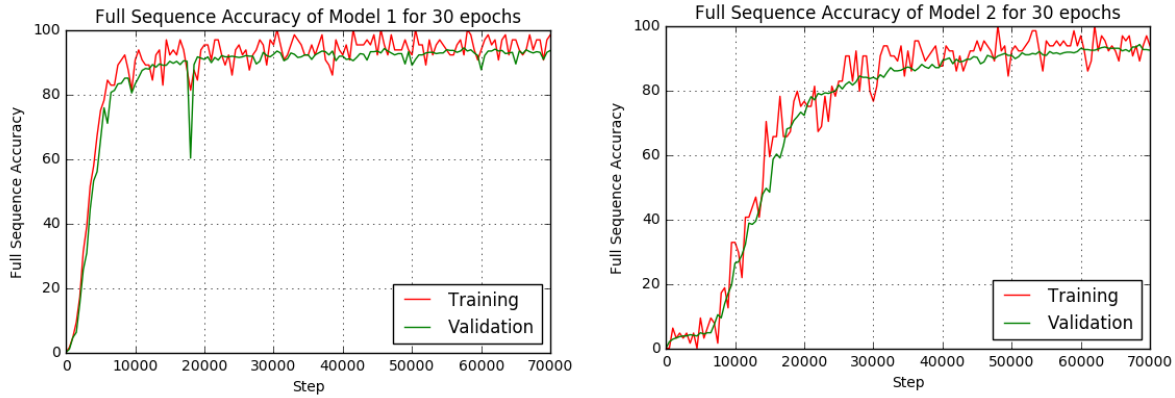


Figure 22 – Full accuracy of Model 1 and 2 for 30 epochs (1 epoch  $\approx$  2,341 steps).

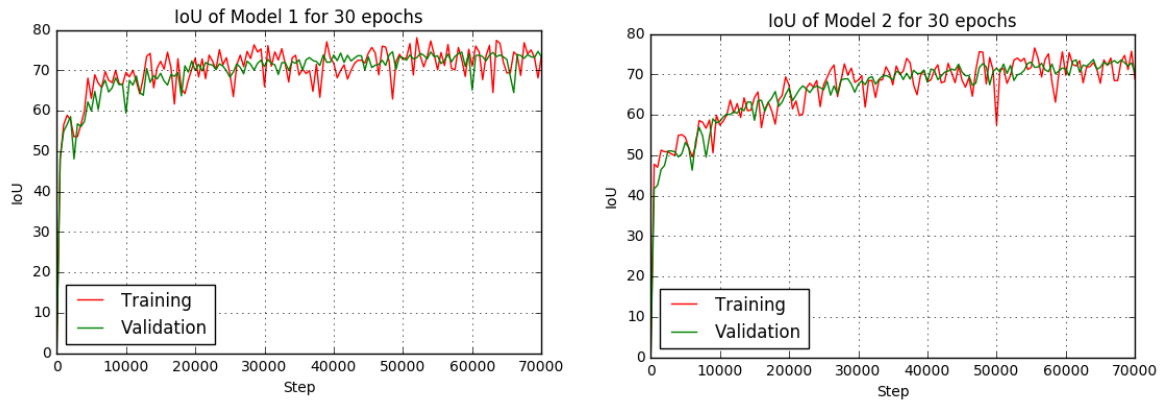


Figure 23 – IoU of Model 1 and 2 for 30 epochs (1 epoch  $\approx$  2,341 steps).

We observed that the performance of Model 1, both full sequence accuracy and IoU, seem to be saturated at 30 epochs. On the other hand for Model 2, although the improvement rate of full sequence accuracy and IoU start to slow down when approaching 30 epochs, there seems to be some rooms for improvement.

We ended up training Model 2 for over 115 epochs before the model overfits.

Figure 24 shows the training loss, full sequence accuracy and IoU performance of Model 2 for 115 epochs.

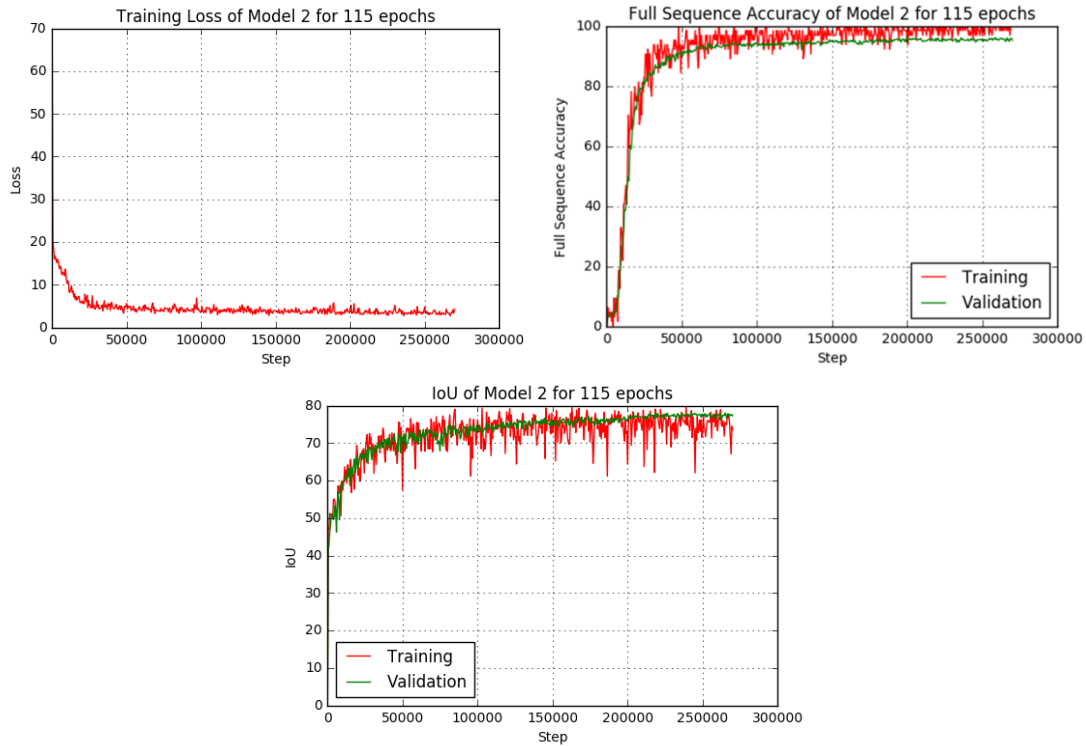


Figure 24 – Training loss, full sequence accuracy and IoU of Model 2 for 115 epochs (1 epoch  $\approx$  2,341 steps).

The performance of the final model based on the test is as listed in Table 6.

	Model 2 (115 epochs)
<b>Test Full Sequence Accuracy</b>	92.86%
<b>Test IoU</b>	79.30%

Table 6 – Performance of Model 2 with the test data after 115 epochs of training.

Since our model generalizes well with unseen test data, the results from the model can be trusted.

To evaluate the robustness of our final model, we replicated another set of test data where small noise perturbation (Gaussian noise with noise factor 0.2) is added to every image. The prediction performance for both sets of test data is as listed in Table 7 below.



	Original Test Data	Test Data with Noise
<b>Test Full Sequence Accuracy</b>	92.86%	91.98%
<b>Test IoU</b>	79.30%	78.94%

*Table 7 – Performance of the final model with the original test data and the test data with noise perturbation.*

With the presence of small noise, our final model’s full sequence accuracy deteriorates by 0.88%, and IoU deteriorates by 0.36%. We can see that small perturbations in input space do not affect the prediction results significantly. Hence our model should be robust enough for the problem.

We also investigated the final model’s accuracy of digit prediction for each of the 5 digit positions based on the test dataset. The result is as listed in Table 8.

	Classification Accuracy
<b>Digit 1</b>	96.45%
<b>Digit 2</b>	95.94%
<b>Digit 3</b>	98.15%
<b>Digit 4</b>	99.82%
<b>Digit 5</b>	99.98%

*Table 8 – Final model’s digit prediction accuracy for the 5 digit positions based on test dataset.*

We observed that the classification accuracy for digit position 1 and 2 are slightly lower compared to the other 3 digit positions. This is consistent with our initial expectation mentioned the Data Preprocessing section, i.e. due to the high imbalance in class labels, classification rate for digit position 1 and 2 will be poorer compared to the rest.

## Justification

Compared to the benchmark model by Goodfellow, et al., which achieved full sequence accuracy of 96.03%, our best model achieved full sequence accuracy of 92.86%. Our model's full sequence accuracy is 3.17% shy from the benchmark model.

The primary reason for this discrepancy is due to limited computing resources. In contrast to the benchmark model which is trained for approximately six days using 10 replicas in DistBelief [4, 8], our best model is trained using a single GPU (Nvidia GeForce GTX 760, 2.0GB RAM) for duration lesser than one day.

Due to limited amount of memory and parallel computing power, we use input image size of 32x32 which is smaller than what is used in the benchmark model, i.e. 54x54.

Apart from that, our fully connected layer for the digit prediction is also smaller compared to the benchmark model. We are using a single fully connected layer of 3000 nodes for the digit prediction task, whereas the benchmark model used two layers of fully connected layers each containing 3072 nodes.

One point worth noting is that our models perform an additional task of bounding box prediction which is not available in the benchmark model. We achieved an average IoU of 79.30% for bounding box prediction in the test dataset.

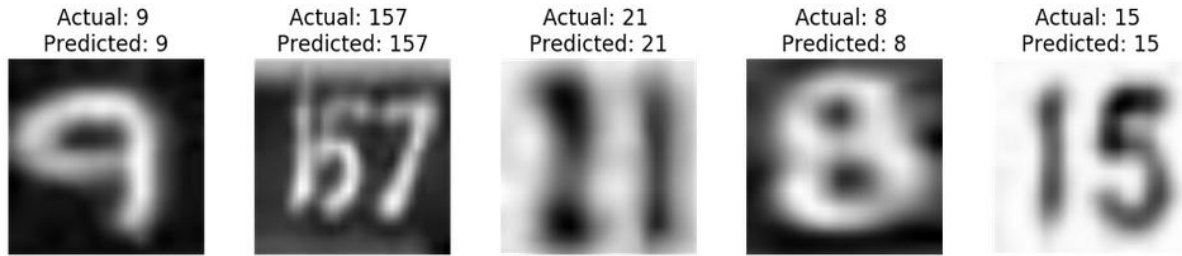
## V. Conclusion

### Free-Form Visualization

In this section, we will show the prediction output of our final model.

Figure 25 shows the length predicting capability of the model. Out of the 13,068 test samples, our model is capable of predicting length of 12,864 instances correctly, equivalent to 98.44% accuracy. The figure only shows 5 sampled instances for both correct and wrong predictions in the test dataset.

Correct predicted length: 12864 out of 13068



Wrong predicted length: 204 out of 13068

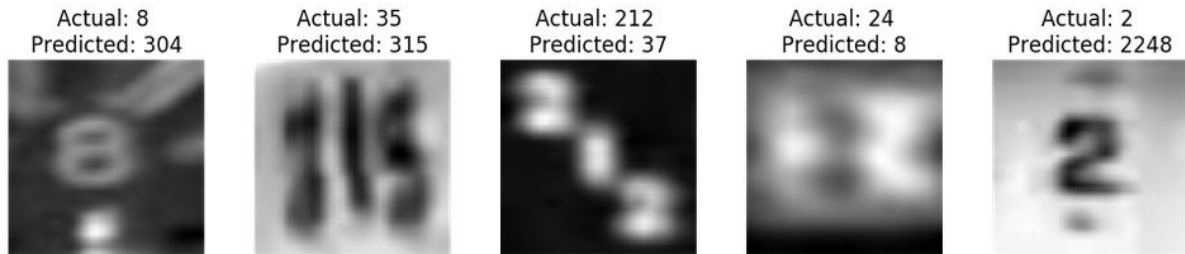


Figure 25 – Length predictions in test dataset. (a) Correct predictions. (b) Wrong predictions

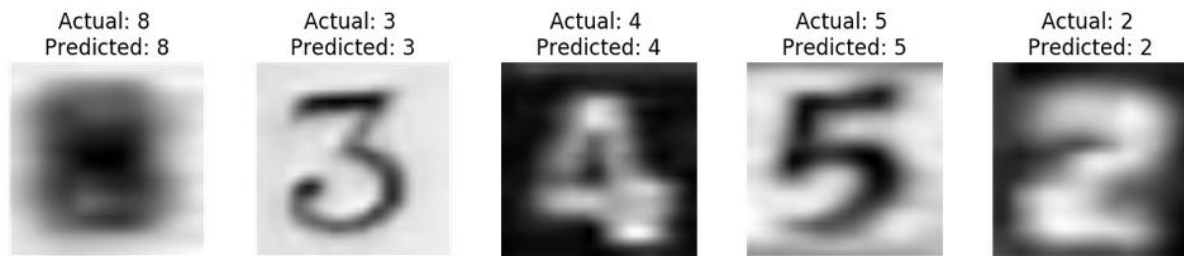
Next we will show the full sequence prediction performance of our final model according to different digit string lengths. Table 9 shows the full sequence accuracy results based on the test dataset.

	Correct Prediction	Wrong Prediction	Total	Correct Percentage
Length 1	2358	125	2486	94.85%
Length 2	7777	579	8356	93.07%
Length 3	1871	210	2081	89.91%
Length 4	129	17	146	88.36%
Length 5	0	2	2	0.00%

Table 9 – Final model's full sequence accuracy for the digit strings of different lengths based on test dataset.

Figure 26 to 30 show the visualization of 5 samples each for the correct and wrong full sequence predictions according to the ground truth label length.

Correct result - length 1: 2358 out of 2483



Wrong result - length 1: 125 out of 2483

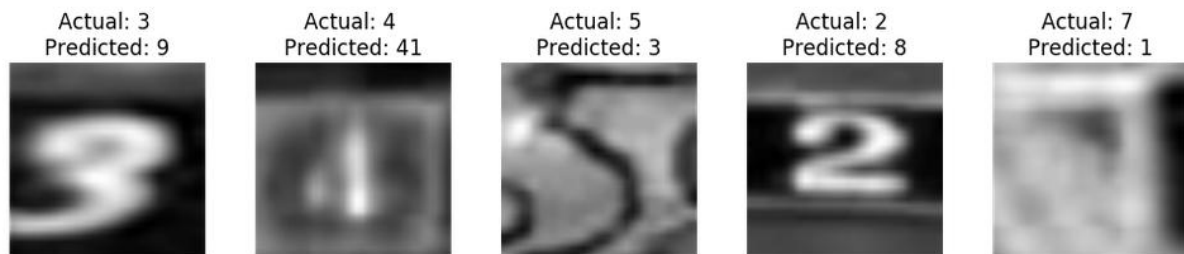
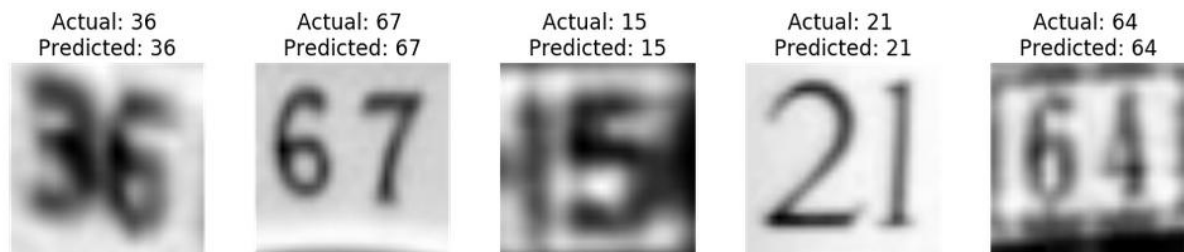


Figure 26 – Full sequence prediction of length 1 in test dataset. (a) Correct predictions. (b) Wrong predictions

Correct result - length 2: 7777 out of 8356



Wrong result - length 2: 579 out of 8356



Figure 27 – Full sequence prediction of length 2 in test dataset. (a) Correct predictions. (b) Wrong predictions

Correct result - length 3: 1871 out of 2081



Wrong result - length 3: 210 out of 2081

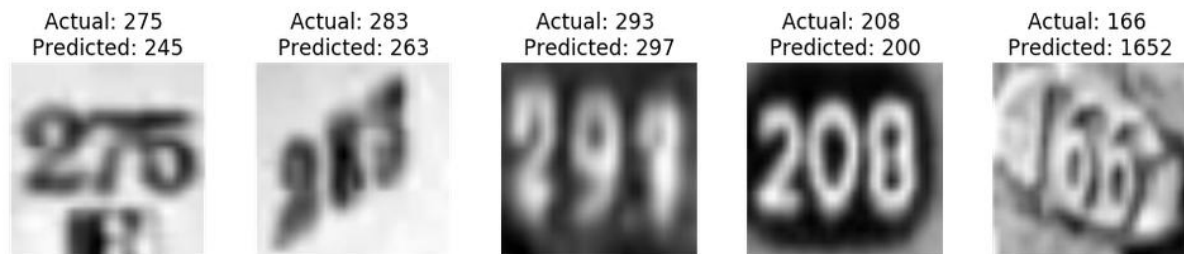


Figure 28 – Full sequence prediction of length 3 in test dataset. (a) Correct predictions. (b) Wrong predictions

Correct result - length 4: 129 out of 146

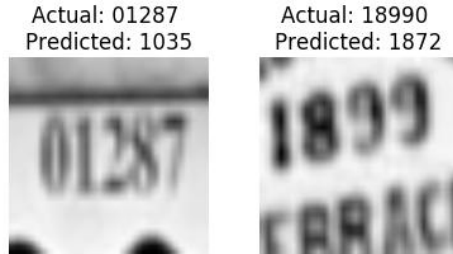


Wrong result - length 4: 17 out of 146



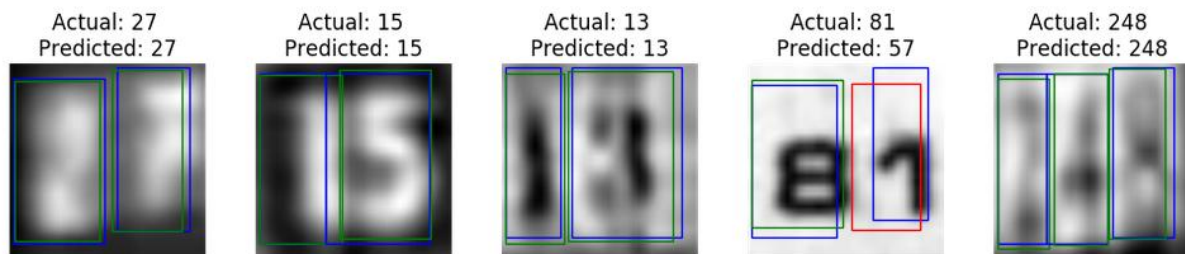
Figure 29 – Full sequence prediction of length 4 in test dataset. (a) Correct predictions. (b) Wrong predictions

Correct result - length 5: 0 out of 2  
Wrong result - length 5: 2 out of 2



*Figure 30 – Full sequence prediction of length 5 in test dataset. Only wrong predictions are available as our model predicted wrongly for 2 out of 2 samples.*

Lastly, we will see the bounding box prediction outcome of our final model. Figure 31 shows 5 sampled results of bounding box prediction based on the test dataset. The ground truth bounding boxes are shown in blue, whereas the predicted bounding boxes are shown in either green or red. Green box indicates the predicted bounding box has an IoU percentage of over 70%, while red box indicates the predicted bounding box has an IoU percentage of less than 70%.



*Figure 31 – Bounding box predictions in test dataset. Blue boxes refer to ground truth bounding boxes, green boxes refer to predicted bounding boxes with IoU score more or equal to 70%, and red boxes refer to predicted bounding boxes with IoU score less than 70%.*

## **Reflection**

In this project, we have implemented a deep convolutional network for multi-digit recognition and bounding box prediction based on the SVHN dataset. Our final model achieves full sequence accuracy of 92.86% for the digit recognition task and IoU of 79.30% for the bounding box prediction task.

One challenging aspect during the implementation of this project is the label imbalance for the digit string length and the digit class label of the five digit positions. The label imbalance issue can cause difficulties for the model in learning the classification of digit string length and digit class. In this project, we addressed only the digit string length imbalance issue. We handled this by performing sampling to ensure the distribution of digit string length in the final training data is uniform.

During the training of our models, we noticed one particular implementation detail has a significant effect in enabling our model to learn better. This is the random shuffling between training epochs. With this in place, we noticed our model can learn over more epochs of training without overfitting.

## **Improvement**

We observed that imbalanced digit class labels especially for position 1 and 2 have caused our model to perform less well in terms of full sequence accuracy. However, this cannot be further alleviated from the angle of data sampling, as this is an outcome of our initial data sampling to make the digit length distribution uniform. We cannot devise a sampling plan that makes digit distribution for all positions uniform without affecting the length distribution.

One potential solution is to tackle this problem during the training of the model instead. By reweighting loss during training according to class label weights in each training batch, we could possibly alleviate the effect of imbalanced class labels for the digit classification.

## References

- [1] Adrian Rosebrock's blog post "Intersection over Union (IoU) for object detection", <http://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/> (Retrieved Mar-16-2016).
- [2] Netzer, et al. "Reading digits in natural images with unsupervised feature learning." NIPS workshop on deep learning and unsupervised feature learning. Vol. 2011. No. 2. 2011.
- [3] CS231n Convolutional Neural Networks for Visual Recognition – Convolutional Networks, <http://cs231n.github.io/convolutional-networks> (Retrieved Feb-16-2016).
- [4] Goodfellow, et al. "Multi-digit number recognition from street view imagery using deep convolutional neural networks." arXiv preprint arXiv:1312.6082 (2013).
- [5] Ioffe, Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." arXiv preprint arXiv:1502.03167 (2015).
- [6] He, et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." Proceedings of the IEEE international conference on computer vision. 2015.
- [7] Kingma, Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).
- [8] Dean, et al. "Large scale distributed deep networks." Advances in neural information processing systems. 2012.