Capstone Project: Deep Learning

Street View House Numbers Recognition

Ryan Gao

Nov. 24 2016

I. Definition

1. Overview

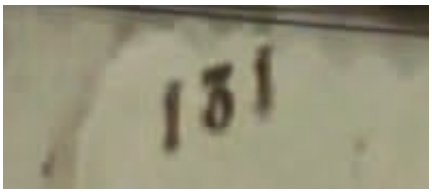
Street View House Numbers is a real-world image dataset for developing machine learning and object recognition algorithms with minimal requirement on data preprocessing and formatting. The dataset contains over 600,000 labeled digits cropped from street level photos. Traditional approaches to recognizing arbitrary multi-digits numbers typically seperate out the localization, segmentation, and recognition steps.

In this project, I built a 5-layer convolutional neural network, and focus on recognizing multiple digits simultaneously without segmentation. The model achieves 84.8% accuracy on character-level recognition and 74.5% accuracy on sequence level recognition.

2. Problem Statement

Street number transcription is a special kind of sequence recognition. It is obvious that digit recognition is much simpler than fully general object classification. There are only 10 classes we need consider. So the problem will be how do we recognize the digits as a sequence. Given an image, the task is to identify the number in the image. The number to be identified is a sequence of digits, $s = s_1, s_2, \ldots, s_n$.

Figure 1. image samples



3. Metrics

When determining the accuracy of the digit transcriber, there two forms of metrics I use in this project.

Firstly, due to the characteristics of the model I built, which recognize the digits separately then combine them to a sequence, it's important to know how well of each digit is recognized. So I need calculate the single character recognition accuracy, which is defined as `def accuracy_single()` .

$$character\ accuracy = \frac{\sum single\ digit\ recognize\ correctly}{\#\ total\ digits\ in\ the\ dataset} \times 100\%$$

Secondly, as a sequence, one mistaken recognition may make a huge difference in understanding them in some application. Thus, I compute the proportion of the input images for which the length n of the sequence and every element s_i of the sequence is predicted correctly. This accuracy function is defined as `def accuracy_multi()`

$Let\ I_i = indicator\ of\ the\ ith\ image\ recognized\ correctly$

$$sequence\ accuracy = \frac{\sum_1^n I_i}{n} \times 100\%$$

II. Analysis

1. Data Exploration

The SVHN dataset is a dataset of about 200K street numbers, along with bounding boxes for individual digits. The dataset is in two format, one is of original images, the other is MNIST-like 32-by-32 images centered around a single character. Since our goal is to recognize the multiple digits simultaneously. So the following dataset all refers to the original ones.

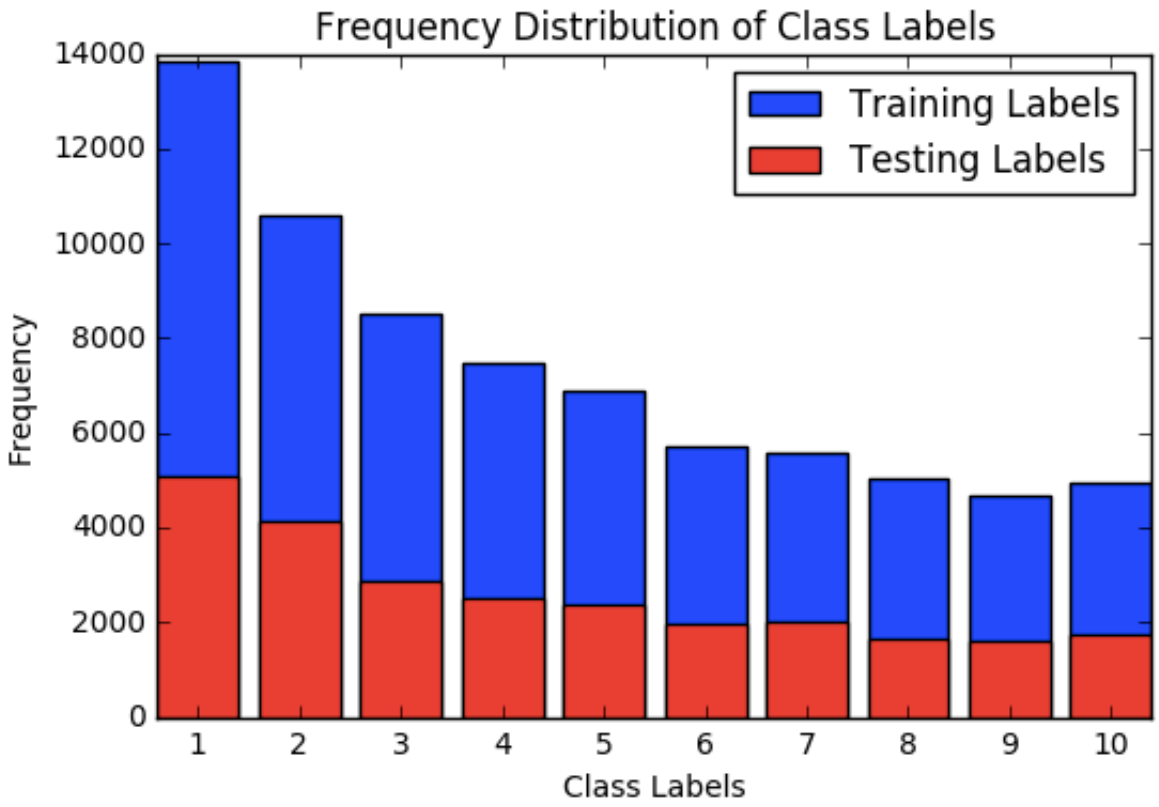
Figure 2. images with bounding box

The original-formatted dataset is divided into three subsets: train set, extra set and test set. Each set contains the original images in png format, together with a digitStruct.mat file. The digitStruct.mat file contains a struct called digitStruct with the same length as the number of original images. 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.
Eg: digitStruct(300).bbox(2).height gives height of the 2nd digit bounding box in the 300th image.

Table 1. train and test dataset

	Number of images	Number of digits
train.tar	33402	73257
test.tar	13068	26032

Figure 3. Distribution of digits in images



One special property of the street number transcription problem is that the sequences are of bounded length. Actually there is only one image containing more than 5 digits in the train and test dataset.

3. Algorithms and Techniques

The general literature on image understanding is vast. Object classification and detection has been driven by many large scale visual recognition challenge such as ImageNet. Various model and methods have been used in both research and industrial work. Here I focus on reviewing the techniques that either have been the major component of the model or help reduce training time and improve accuracy.

Local Response Normalization

In neurobiology, there is a concept called “lateral inhibition”. This refers to the capacity of an excited neuron to subdue its neighbors. We basically want a significant peak so that we have a form of local maxima. Local Response Normalization (LRN) layer implements the lateral inhibition in the CNN layers.

Denoting by $a^i_{x,y}$ the activity of a neuron computed by applying kernel i at position (x, y) and then applying the ReLU nonlinearity, the response-normalized activity $b^i_{x,y}$ is given by the expression

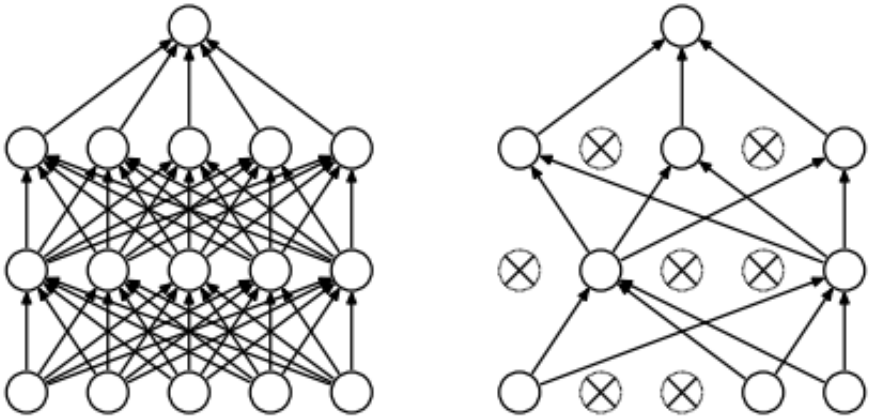
$$b^i_{x,y} = a^i_{x,y} / (k + \alpha \sum_{j=max(0,i-n/2)}^{min(N-1,i+n/2)} (a^j_{x,y})^2)^\beta$$

where the sum runs over n "adjacent" kernel maps at the same spatial position, and N is the total number of kernels in the layer.

Response normalization reduces our single digit and sequence recognition error rates by 0.9% and 0.7%, respectively.

Dropout

Figure 4. dropout



(a) Standard Neural Net (b) After applying dropout.

"Dropout" consists of setting to zero the output of each hidden neuron with probability 0.5. The neurons which are “dropped out” in this way do not contribute to the forward pass and do not participate in back-propagation. So every time an input is presented, the neural network samples a different architecture, but all these architectures share weights. This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of particular other neurons. It is, therefore, forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.

Without dropout, my network exhibits substantial overfitting. Applying dropout gives me roughly 5% accuracy improvement on character-level recognition and 7% improvement on sequence-level recognition.

Adam Optimization

Adam is a method for efficient stochastic optimization that only requires first-order gradients with little memory requirement. The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. Adam combines the advantages of two popular methods: AdaGrad , which works well with sparse gradients, and RMSProp, which works well in on-line and non-stationary settings. Some of

Adam's advantages are that the magnitudes of parameter updates are invariant to rescaling of the gradient, its stepsizes are approximately bounded by the stepsize hyper-parameter, it does not require a stationary objective, it works with sparse gradients, and it naturally performs a form of step size annealing.

4. Benchmarks

Human performance on sequence-level recognition reaches 98%. And the state-of-art of sequence transcription accuracy is 96.03%. The model is built by Goodefellow $et\ al$. Their model also achieves a character-level accuracy of 97.84%. However, the parameters of their model is so large that it took them 6 days to train the model.

Apparently, the MacBook I use couldn't bear such computational task and the number of parameters must be reduced due to the 8G RAM. The threshold I set for this project is to reach 85% accuracy on sequence transcription.

III. Methodology

1. Preprocess

1. Dataset composition

Since there are given no information about how the sampling of these images was done. I compose the validation set from the randomized training samples, yielding 6000 samples.

Table 3. train, test and validation dataset

	sample count
train_dataset	27401
test_dataset	13068
valid_dataset	6000

Figure 5. Distribution of number length in each dataset

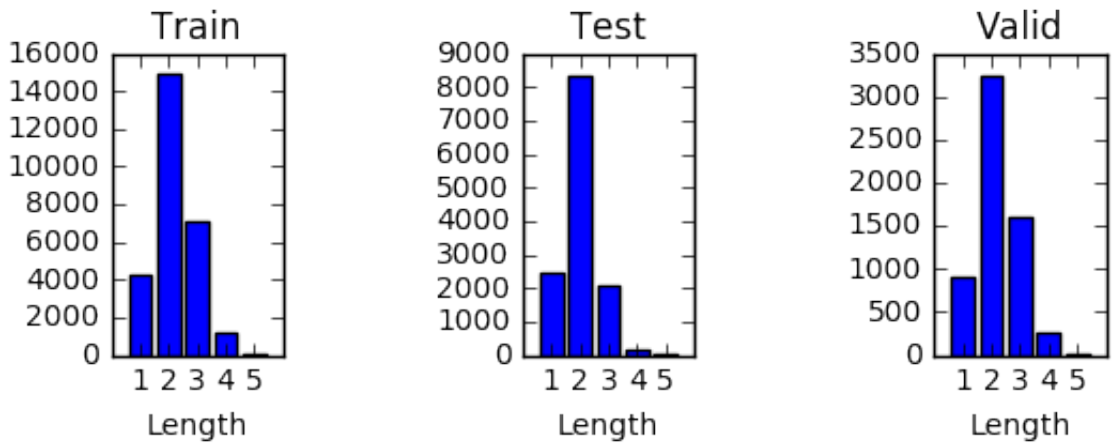
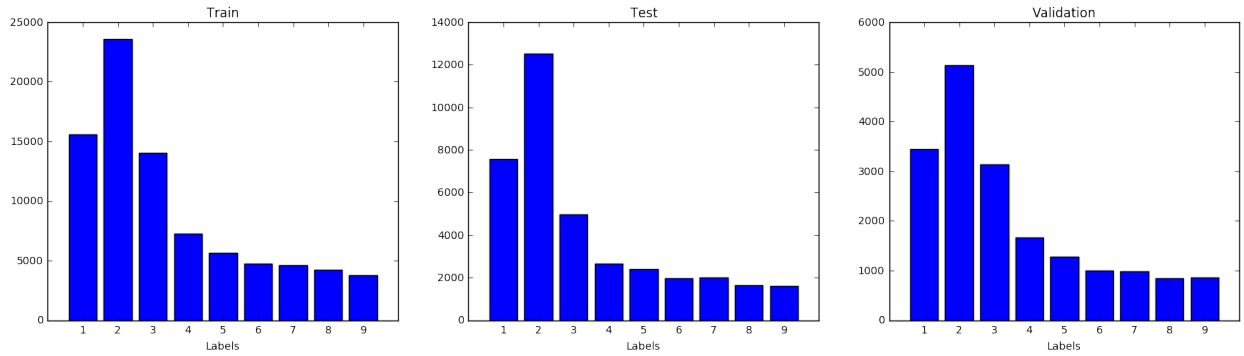


Figure 6. Distribution of number count in each dataset



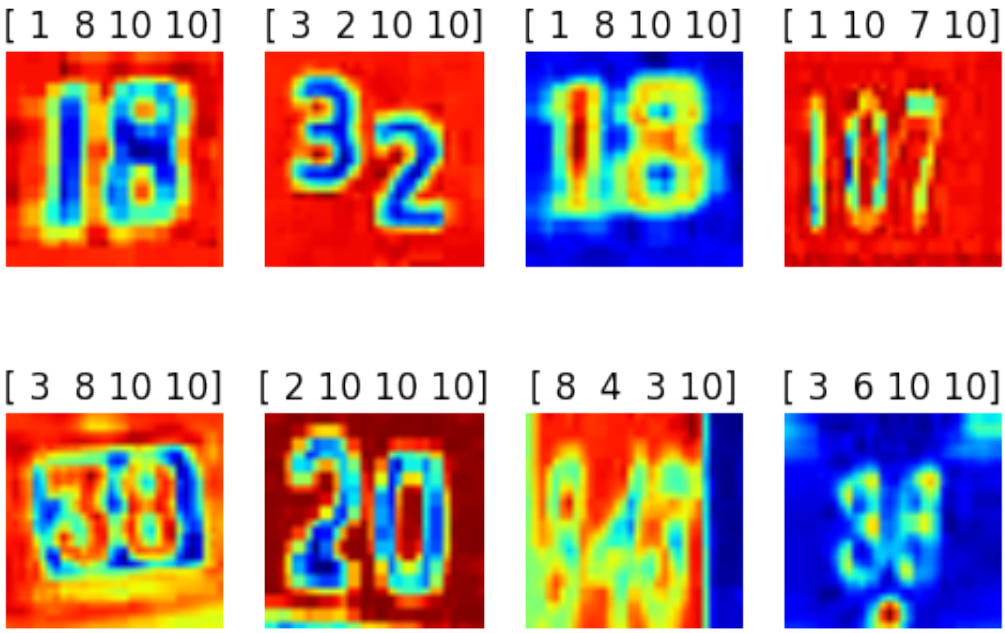
2. Reshape & Normalization

I preprocess the dataset in two aspects.

First, I find the small rectangular bounding box boundaries by reading the digitStruct.mat. I then crop the original images to remove redundant information. Therefore the size of images is reduced. After cropping, I resize the images to $\mathbf{32} \times \mathbf{32}$ for computational convenience.

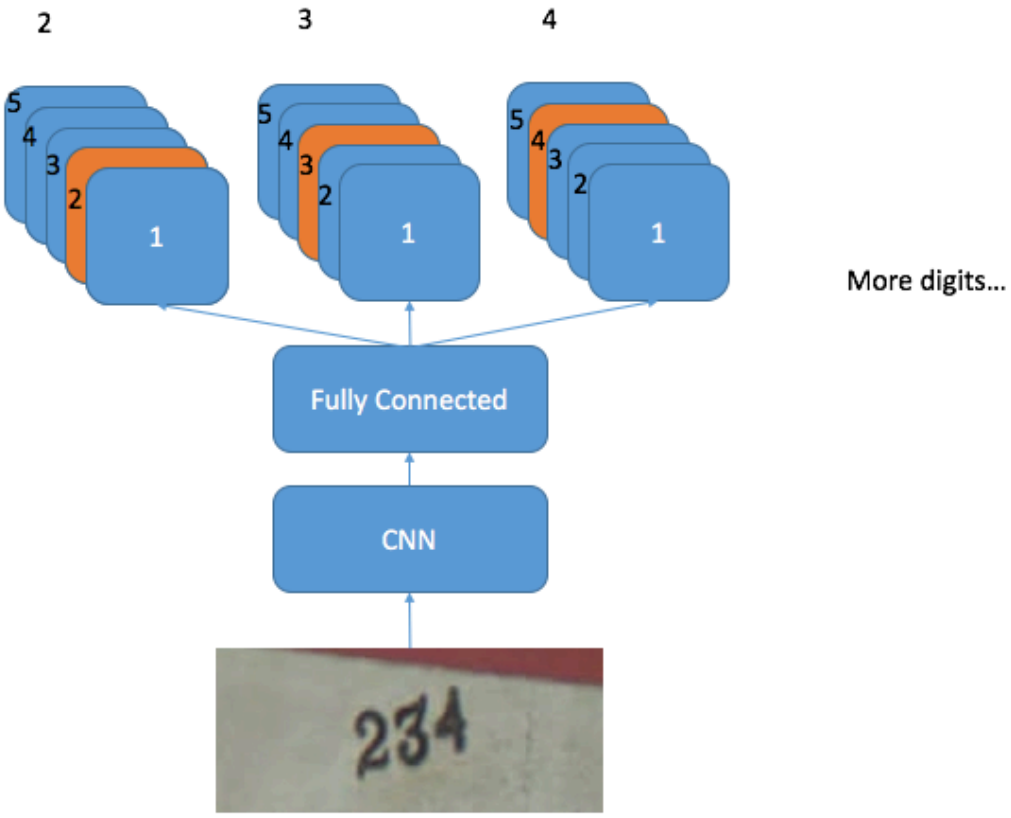
Second, unlike the MNIST dataset, the SVHN dataset comes from the real world image, which has 3 color channels. Our goal is to recognize the digits, so the color doesn't matter. I first convert the images to greyscale, then subtract the mean of each image and divide by its standard deviation. The image information is stored into numpy ndarray.

Figure 7. images after normalization (10 stands for none)



2. Implementation

Figure 8. Architecture



Loss function

As we can see in Figure 8, each digit is transcribed separately. In the model, I assign five connected layers for each digits. The loss of each logit contains its L2 norm regularization. In Tensorflow there 2 types of softmax-cross-entropy function. Here we use `tf.nn.sparse_softmax_cross_entropy_with_logits()` which measures the probability error in discrete classification tasks in which each entry is in exactly one class. The final loss is the sum of each digit's loss.

Code snippet 1. Loss function

```
  def model(dataset)
      ...
       logits_1 = tf.matmul(hidden4_drop, s1_w) + s1_b
       logits_2 = tf.matmul(hidden4_drop, s2_w) + s2_b
       logits_3 = tf.matmul(hidden4_drop, s3_w) + s3_b
       logits_4 = tf.matmul(hidden4_drop, s4_w) + s4_b
       logits_5 = tf.matmul(hidden4_drop, s5_w) + s5_b
       return [logits_1, logits_2, logits_3, logits_4, logits_5]

   logits = model(dataset)
   loss_per_digit = [tf.reduce_mean(
               tf.nn.sparse_softmax_cross_entropy_with_logits(
                   logits[i],
                   tf_train_labels[:,i+1]
               )) + beta_regul * tf.nn.l2_loss(sw[i])
              for i in range(5)]

   loss = tf.add_n(loss_per_digit)
```

Metrics

Before we define accuracy function. Let's find out what the predictions look like.

The `prediction_softmax()` function returns each logits applied with softmax function. Prediction has the shape (5, batch_size, 11), denoting five digits , the size of data batch and number of labels. We know that the number of digits in the image various a lot. When computing accuracy, we need remove those 10s (default value for None).

Code snippet 2. Prediction function

```
   def prediction_softmax(dataset):
       prediction = tf.pack([
           tf.nn.softmax(model(dataset)[0]),
           tf.nn.softmax(model(dataset)[1]),
           tf.nn.softmax(model(dataset)[2]),
           tf.nn.softmax(model(dataset)[3]),
           tf.nn.softmax(model(dataset)[4])])
       return prediction

   test_prediction = prediction_softmax(tf_test_dataset)
```

Code snippet 3. Examples of the predictions and labels

```
>>>np.argmax(predictions, 2).T
>>>array([[ 1, 10,  7, 10, 10], # 107
       [ 5,  8,  3, 10, 10], # 583
       [ 9,  4, 10, 10, 10], # 94
       [ 7,  1, 10, 10, 10], # 71
       [ 7,  1, 10, 10, 10]]) # 71

>>>np.argmax(predictions, 2).T == labels[:,1:6]
>>>array([[ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [False, False,  True,  True,  True],
       [ True, False,  True,  True,  True],
       [ True, False, False,  True,  True]], dtype = bool)
```

In `accuracy_single()` , I first slice the label to its original length, then sum up all true recognition of single digit. In `accuracy_multi` , as long as there is a `False` in the sequence, I consider it an error and sum it up as error rate then subtract it from total image count in the batch .

Code snippet 4. Metric functions

```
def accuracy_single(predictions, labels):
    """calculate character-level accuracy"""
    a = np.argmax(predictions, 2).T == labels[:,1:6]
    length = labels[:,0] # each sequence's length
    summ = 0.0
    for i in range(len(length)):
        summ += np.sum(a[i,:length[i]])
    return(100 * summ / np.sum(length))

def accuracy_multi(predictions, labels):
    """calculate sequence-level accuracy"""
    count = predictions.shape[1] # batch size
    return 100.0 * (count - np.sum([1 for i in np.argmax(predictions,
2).T ==       labels[:,1:6] if False in i])) / count
```

3. Refinement

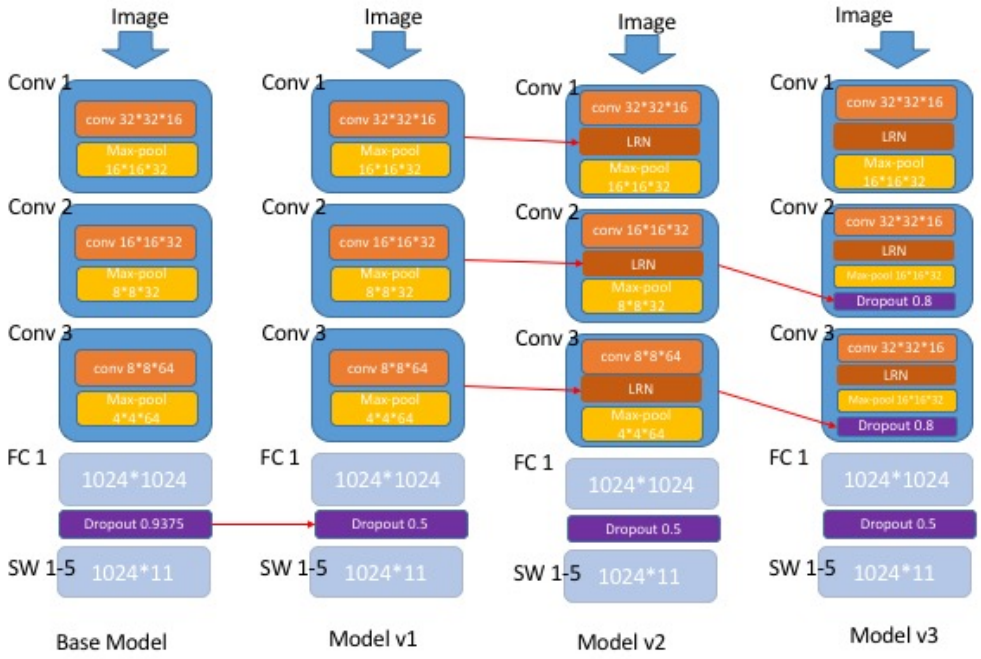
There are many techniques for refinement and many hyper-parameters need to be tuned. To do a grid search on deep neural network will cost huge amount of effort and time. The refinement approach I adopted is observing the loss and accuracy on validation set and then decide which approach may improve the accuracy on test dataset.

The base model contains 3 ConvNet each followed by a max-pooling layer and 1 fully connected layer followed by 5 locally connected layers. Dropout is only applied on fully connected layer with value of 0.9375. Learn rate is initialized as 0.001. I use Adam optimizer for stochastic optimization.

I mainly adjust the model in 3 ways:

1. Add/remove one layer
2. Increase/decrease the number of hidden units
3. Change the probobality in dropout

Figure 9.



IV. Results

1. Model Evaluation and Validation

Table

Model	Accuracy_single	Accuracy_multi
Base model	77.8%	63.4%
Model v1	82.3%	71.5%
Model v2	82.2%	72.2%
Model v3	84.8%	75.9%

The ConvNet architecture is composed of three repeatedly stacked feature stages and one fully connected hidden layer, and five locally connected layers for recognizing each digits. Each convolution stage contains a convolution module, followed by a pooling module.

All convolution kernels are of size 5×5 . The first convolution layer produces 16 features convolution filters, while the second and third convolution layers ouput 32 and 64 features. All convolution layer use zero padding and zero stride on the input so that the representation size remains the same. All the max pooling window size is 2×2 with a stride 2×2 . The fully connected layer contains 1024 nodes each.

I trained with dropout applied to all hidden layers but not the input. Regularization constant, learning rate and its decay were tuned on the validation set. I use Adam optimizer for stochastic optimization.

Table 4. Final model parameters

Layer	Filter	Stride	Depth	Output
conv_1	5×5	1×1	16	$32 \times 32 \times 16$
max_pool	2×2	2×2	16	$16 \times 16 \times 16$
conv_2	5×5	1×1	32	$16 \times 16 \times 32$
max_pool	2×2	2×2	32	$8 \times 8 \times 32$
conv_3	5×5	1×1	64	$8 \times 8 \times 32$
max_pool	2×2	2×2	64	$4 \times 4 \times 64$

2. Justification

The best model I built so far reaches 75.9%, while the human's performance on this dataset reaches 98% accuracy. This is not accurate enough for transcribing sequence digits in real world. However, when using the MNIST-like dataset, I reached 89% accuracy. The model definitly need further improvement.

V. Conclusion

1. Reflection

Our experiment demonstrate that neural networks can do the segmentation and localization of ordered sequences of object simultaneously. It's worth noting that in this project our approach is trained fully supervised only, and we use the digits' boundaries in the preprocess stage to remove redundant information. Also, even many digits' boundaries(blue boxes) provided by the dataset are transcribed by the AMT workers. There are various ill forms of street numbers in the world. Without these boundaries and preprocess, the street view numbers recognition accuracy may decrease a lot.

Figure 10. Street view house numbers

2 vs 239

1879 vs. 1879-1883

Yet, the model is promising when transcribing the sequences with bounding boxes. One specific application is recognizing car license plates. The car license plates are in rectangular shape, which make it easy for computer vision algorithms to detect the boundaries, and in a specific sequence which is easy to recognize by this sequence-transcribe model.

Figure 11. Number plate recognition with Tensorflow

2. Improvement

Due to the Mac's computational limitation, the training process is too long and I couldn't add more hidden layers or units. For further accuracy improvement, we can add more hidden layers by using faster processors of GPUs.

On the other hand, in the preprocess step I reshape the image to 32×32 format. However, there are many images of higher resolution in the dataset . And in real world, the digital camera contains more and more pixels. Thus, a training model that could process larger image size need implementing when we want to build an end-to-end transcriber in application.

References

1. Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, Andrew Y. Ng *Reading Digits in Natural Images with Unsupervised Feature Learning*
2. Pierre Sermanet, Soumith Chintala, Yann LeCun *Convolutional Neural Networks Applied to House Numbers Digit Classification*
3. Diederik P. Kingma,Jimmy Lei Ba *ADAM- A METHOD FOR STOCHASTIC OPTIMIZATION*
4. Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, Vinay Shet *Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks*
5. G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R.R. Salakhutdinov. *Improving neural net-works by preventing co-adaptation of feature detectors.*
6. Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky,Ilya Sutskever, Ruslan Salakhutdinov *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*
7. Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton *ImageNet Classification with Deep Convolutional Neural Networks*

8. Matthew Earl *Number plate recognition with Tensorflow*