Decoding a Sequences of Digits in Real World Photos Using a Convolutional Neural Network.

1. Definition

1.1 Project Overview:

Recent developments in Deep Neural Networks are quickly advancing the state of the art in computer vision systems. These layering techniques are showing promising performance and versatility. While previous work in visual recognition often required a separate hand coded components to propose candidate segmentations and preform classification; in this project I develop a single simple Convolutional Neural Network model for localizing and classifying a sequence of digits contained within real world images. Digit recognition provides a well defined problem domain for a classification task. While related problems like classifying a handwritten character in scanned document or book has been well studied and are virtually solved [Sermanet], detecting and classifying text from a real word image is a much more difficult problem. [Netzer] Recognition in the real world becomes complicated by environmental factors such as lighting, shadows, specularities, and occlusions as well as by image acquisition factors such as resolution, motion, and focus blurs.[Netzer]

1.2 Problem Statement:

Using a Convolutional Neural Network I will train a model developed in Tensorflow which can correctly recognize and decode sequences of digits from natural images. After training the model I will then use a python script with the PIL imaging library to read out these digit sequences in realtime. This project builds on the work of the the google team [Goodfellow] as a guide o develop a unified approach that integrates localization, segmentation, and recognition three steps via the use of a deep convolutional neural network that operates directly on the image pixels. [Goodfellow]

The tasks involved in this project include:

1. Download a dataset of single well cropped house numbers and preprocess the images.
2. Train a classifier to recognize a single digit using these images.
3. Save the weights from the single digit classifier to be reused for a multi digit classifier.
4. Download a dataset of irregular sized multi-digit sequences.
5. Preprocess the multi-digit dataset so the images into regular sized and digit centered images.
6. Train 5 independent classifier reusing the initial weights from the single digit model, to recognize and decode a sequence of 5 digits.
7. Transcribe a series of digits in realtime using a python script to load the model weights.

1.3 Metrics:

The metric used to measure how well the model is performing is classification accuracy. Classification accuracy is defined as the number of correct predictions made as a ratio of all predictions made. This measurement takes into account both true positives and true negatives equally weighted. A correct prediction is obtained when every element in the sequence correctly transcribed in the correct order. There is no evaluation credited for partial accuracy. As an example, having the digit sequence "3766", and the prediction of "3763" is classified as a false positive. The formula used to calculate accuracy is as follow:

$$\text{Accuracy} = \left(\frac{\text{Number of Correct Prediction}}{\text{Number of All Predictions}} \right) * 100$$

2. Analysis

2.1 Data Exploration

The SVHN dataset (http://ufldl.stanford.edu/housenumbers/) is used to train the CNN model. This dataset is comprised of two separate components. A single digit cropped dataset and unprocessed full-size multi-digit real world images of house numbers.

The cropped dataset contains images that are 32 pixel by 32 pixel, 3 channel (RGB) color images. These images files are similar to the venerable MNIST [LeCun] dataset in that are neatly centered and cropped to a specific dimension. This dataset contains 73,257 for training and 26,032 digits for testing purposes. I further split the training dataset into a 90% training of size 65,931 and a 10% validation dataset set of 7326 for monitoring the performance of the model while training.

The cropping procedure used by the authors of the dataset creates some idiosyncrasies. Many of the images contain some remnant artifacts of adjacent digits around the edges of the crop. These artifacts are necessary in order to preserve the aspect ratio of the images. I found a second peculiarity in the labeling, the digit "0" is represented as "10". This creates some an issue when transforming the labels into a one-hot encoding, later discussed.

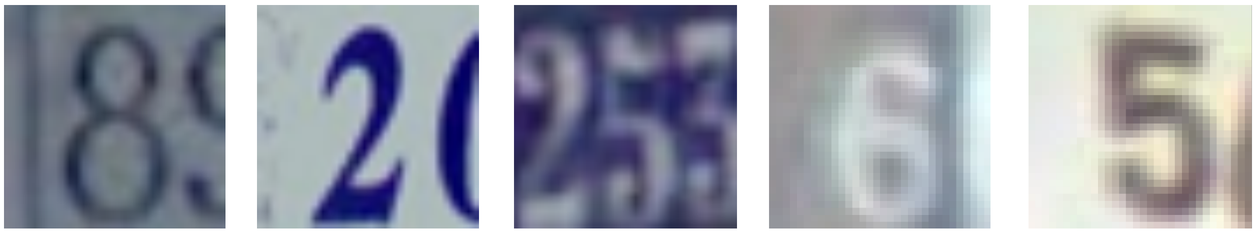


Figure 1. Example images from the cropped dataset.

The SVHN full dataset is comprised of variable resolution, variable size, variable length, color house number images. In total there are around 200k annotated images images used for training and validation. [Netzer] The SVHN dataset is comprised of 3 data sets, Train, Test and Extra. Each of these sub datasets have an accompanying reference file (digitStruct.mat) file with is a data file which provides the following information for the dataset:

1. *Filename*: a file name used as a pointer to the image file.
2. *Bounding Box:* which is comprised of a pixel coordinate for Top, Left, and a value for the Height, Width of each digit. Each of the bounding box elements are contained within an ordered array which is used to place the digit in the sequence
3. *Label:* for each digit in an image as an ordered array.

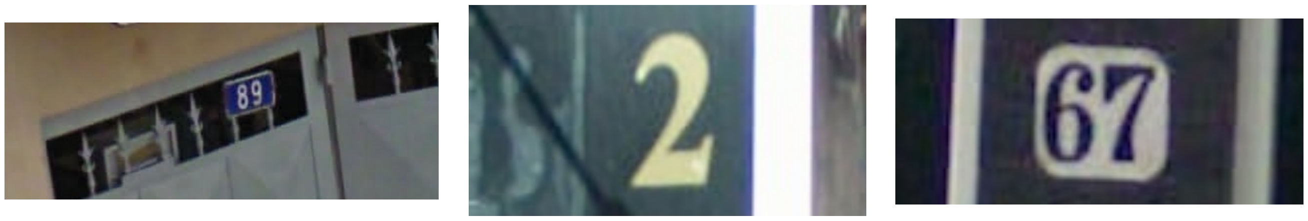
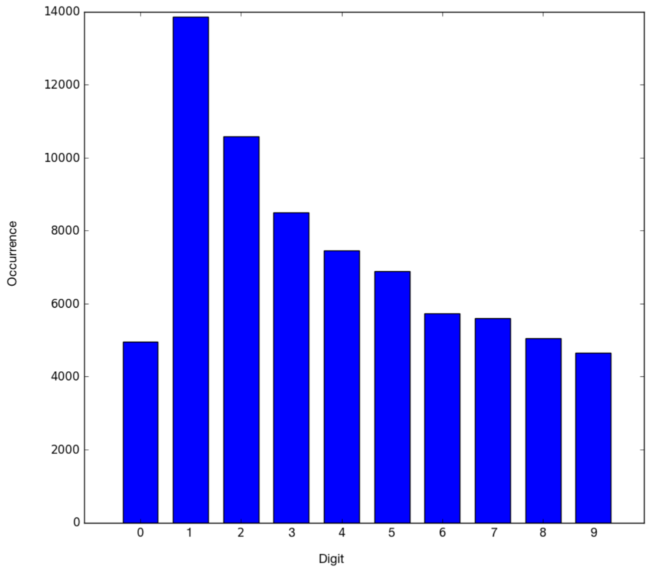


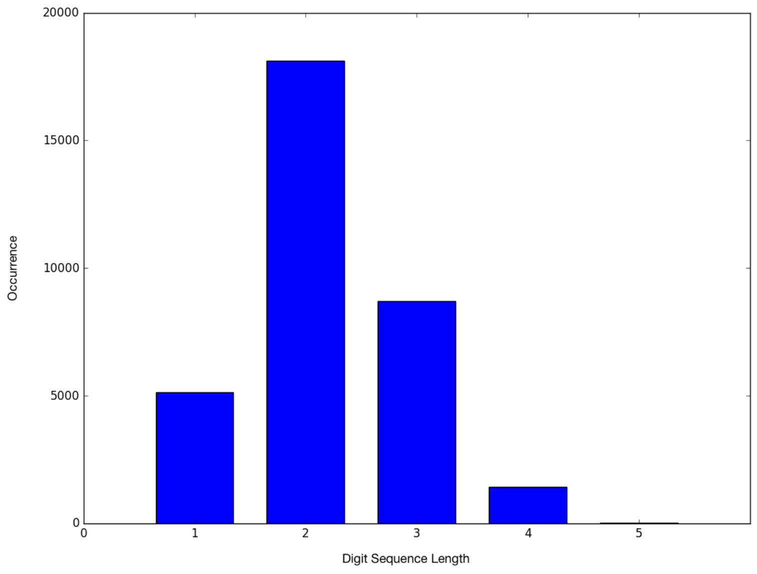
Figure 3. Example images from the full dataset.

The large variation in image size, ranging in area from 300 to 438876 pixels requires some standardization to become useful for training purposes. This process is further discussed in section 3.1.

2.2 Exploratory Visualization

The cropped dataset is neatly procured for testing, however I had noticed some peculiar features while exploring the full dataset. First to note is the distribution of digits is not equal across the range of digits. It can be seen from figure 4A that the distribution is skewed towards the lower value digits, this exposes the lower digits to many more training examples. The plot 4B depicts the distribution of the digit sequence lengths, this is used to provide roughly how many digits are detected per image, with the most





frequent digit sequence length at 2.

Figure 4A. Distribution of digits in the full dataset.

Figure 4B. Distribution of digits sequence length.

2.3 Algorithms and Techniques

The model is developed using a multi layer Convolutional Neural Network, offend referred to as ConvNets or CNNs. Like other neural networks a CNN is composed of multiple layers. Each layer has a simple function, it transforms a 3D volume of inputs with some differentiable function. This layering is what gives the network "Depth" as in Deep Learning. Each layer of the CNN is composed of many copies of the same neuron, each neuron having having tied parameters of learnable weights and biases. Every neuron in the graph has a job, take som input and preforms a dot product and optionally follows it with a non linear activation function. This allows the network to have lots of neurons and express computationally large models while keeping the number of actual parameters, values to be learned, and describing how to behave, fairly small. [Cohah Bolog] Built up from these small components the network automatically learn a unique set of features optimized for a given task. [Sermanet] In the case of images the network computes a single score which takes in image pixel data on one end and outputs class scores on the other end.

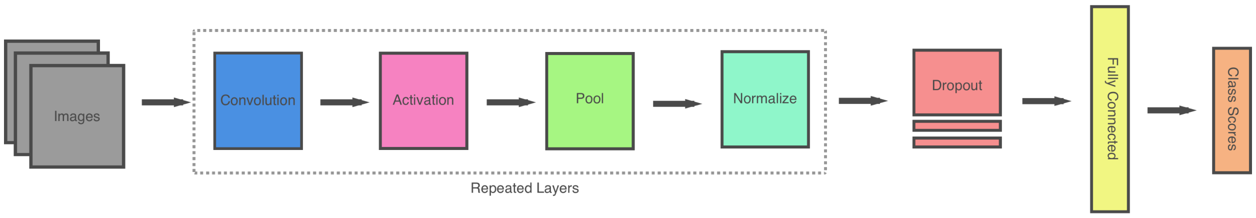


Figure 5. Generic Convolution Neural Network Architecture

The Convolutional Neural Network model developed in this project is composed of many layers, each of the component layers is described below.

1. *Convolution Layer*: Performs a 2D filtering on the image parameters consist of a set of learnable filters, compute dot products between the entries of the filter and the input at any position. These filters are used to generate features at different layers of the data. Training is done through back propagating the errors.

2. *Activation function*: non-linear used to approximate any function and transform them into a range, IE commonly -1 to 1 of 0 to 1, and detect nonlinear features in the data. Activation function answers the question that if some of the input activations are turned on, how should the output become activated.

3. *Pooling Layer*: are used in-between successive Convolution layers in a CNN architecture. This layer's function is to summarize and progressively reduce the spatial size of the activations in order to reduce the amount of parameters and computation in the network; correctly setting the hyper parameters this also offers some control of overfitting.

4. *Dropout*: is a simple method to regularize neural networks and provides protection against overfitting. This is layer is only applied to the training network and keeps only random neurons active. The key idea is to randomly drop units (along with their connections) from the neural. This prevents units from co-adapting too much. [Srivastava et al] creating a thinned training network.

5. *Fully-Connected Layer*: These layers are used at the end of the network and may have a non-linear activation function or a softmax activation in order to output probabilities of class scores. The FC layer is used flattens out the feature maps from previous layers into a vector. Their activations can then be computed with a matrix multiplication followed by a bias offset.

6. *Output Layer* - This is the final output layer and is used to assign a probability estimates to each of the classes. This layer will reduce the size of the input data to the number of output classes. These are often processed with a softmax classifier when the output is one value per class.

The work began by first training the CNN on a single digit transcriber and was latter trained using to detect multiple digits in a sequence where each CNN was responsible for both spatial location and classification. The latter task requires learning spatial locations of the multiple digits. This is an interesting task which often requires a separate model to propose candidate segments, or provide a hight level model of the image. However, the google team had shown there is a simple approach, not requiring a separate model that explicitly required to extract these spatial location. Goodfellow and team were able to perform the entire task completely within the learned features of a convolutional network by using one classifier per digit.

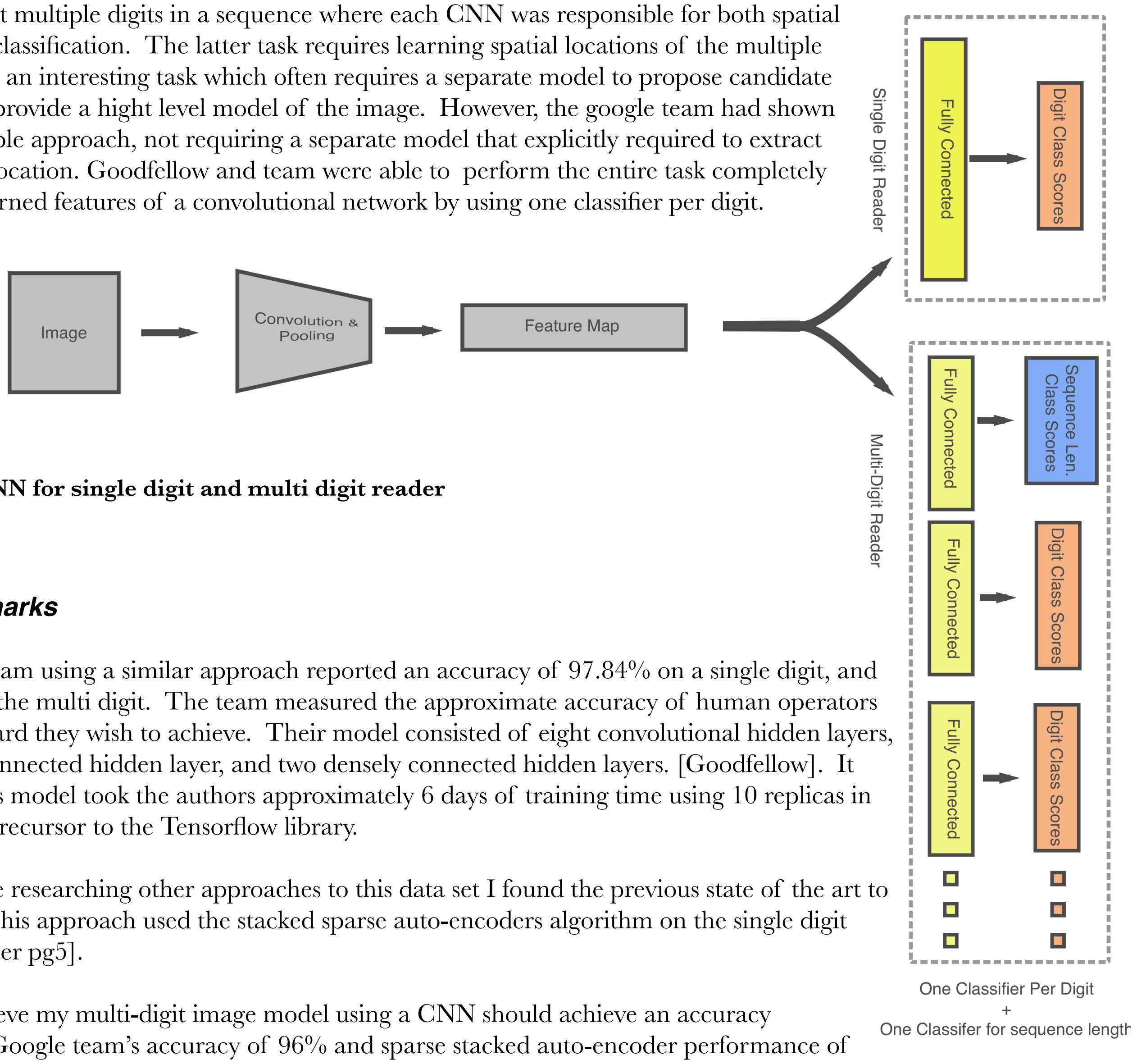


Figure 6. CNN for single digit and multi digit reader

2.4 Benchmarks

The google team using a similar approach reported an accuracy of 97.84% on a single digit, and over 96% on the multi digit. The team measured the approximate accuracy of human operators 98%, a standard they wish to achieve. Their model consisted of eight convolutional hidden layers, one locally connected hidden layer, and two densely connected hidden layers. [Goodfellow]. It was noted this model took the authors approximately 6 days of training time using 10 replicas in Disbelief, a precursor to the Tensorflow library.

While researching other approaches to this data set I found the previous state of the art to be 89.7%. This approach used the stacked sparse auto-encoders algorithm on the single digit dataset. [Netzer pg5].

I believe my multi-digit image model using a CNN should achieve an accuracy between the Google team's accuracy of 96% and sparse stacked auto-encoder performance of 90%. I also expect the cropped single images to slightly out preform the multi digit reader. This is due to the scoring metric as previously explained having a larger likelihood of a single number in the sequence incorrectly causing a false positive result.

3. Methodology

3.1 Data Preprocessing

The cropped single digit dataset images were already heavily preprocessed, requiring little further manipulation. These images are a single digit contained a 32 pixel by 32 pixel matrix. The only changes made to this set was to change the values from their 8-bit color channel in the integer range of 0-255 to a floating point number in the range of 0-1. Next the images were enhanced through a process of Mean Subtraction. This is a process of subtracting the mean pixel value across every channel in the data. As a result this creates a centering the cloud of data around a fixed point origin for every channel of the image channels.

The the labels in the both the single and multi digit datasets required some formatting. The digit "0" is denoted as 10 in the label dataset, which caused some issues when converted to one hot encoding. These were values were reassigned to a value of 0 then the labels were converted to a one hot encoded sequence. The one hot encoding technique places a significant value an indexed value denoting the label, i.e. the number 3 in list of possible digits would be [0,0,0,1,0,0,0,0,0,0] in a zero indexed array.

The variable sized multi-digit full sized images required resizing to a fixed dimension in order to create usable samples. Some the digit sequences with this dataset contained more than 5 digits long, so these images were removed from the training and test sets. Next, the dimensions of the images were standardized to 64 pixels by 64 pixels. This resizing was preformed by finding a cropped box that will fit all the individual characters using the bounding box of each digit. Once the smallest box was found which could contain the digit sequence, the box was then increased by 20% padding around the sequence. Finally image was then cropped around the padding.

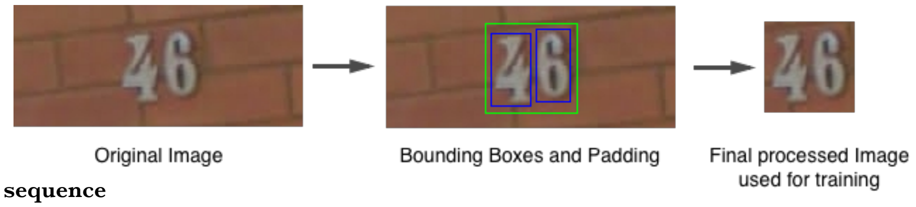

Original Image
Bounding Boxes and Padding
Final processed Image used for training

Figure 7. Digit cropping sequence

The labels were then adjusted to fit one hot encode sequence with an extra value available to denote an empty character; the range of values is 0-10, digits 0-9, and "10" to denote the empty digit. The correct sequence is trained by creating an ordered label array of a fixed length. The correct digit sequence prepended by a number the length of the digit sequence; IE if the target number is "607" The array would be [3, 6, 0, 7, 10, 10] the first digit is the number of digits in the sequence followed by 3 elements indicating the correct digits, followed by 2 10s representing no digit

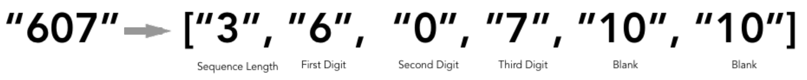


Figure 8. Multi-digit encoding sequence.

3.2 Implementation

The CNN model was applied to two separate tasks as previously described. First it was designed and implemented to transcribe a single digit in 32px by 32 px images, once this task was complete with satisfactory results the insights gained were then applied to a more complex model used to transcribe larger 64px by 64px multi-digit images.

The architecture of the cropped image classifier consist of three convolution layers, followed by a locally connected layer. Each of the convolution layer is set to use a 5 x 5 kernel and the padding is set to "Valid", which preserves the original dimensions of the convolution input. Once the input is reduced to 1x1 dimensions the layer is flattened into a fully connected layer. The larger 64 pixel by 64 pixel images require one more convolution before flattening. The number of units at each spatial location in order is [48, 64, 128, 160] and 160 for the locally connected layers.

Each of the convolution layers is followed my a max-pooling layer with a window size of 2px by 2px. This is used to reduce the spatial dimensions of the image until a 1x1 feature map is achieved. The final layer of the single digit classier uses a pooling layer of 1x1 feature map which is then flattened into 1x160 vector which is the passed to a softmax classifier to obtain the class scores.

The architecture of the larger 4 pixel by 64 pixels uses the same base architecture as the small images, with only the difference occur at the fully connected layer for localization. The feature map is 1x1x160. At this point a classifier is attached for every digit in the sequence. This classifier is responsible for both producing the correct class a long with learning the spatial dimensions of the digits.

3.3 Refinement

The CNN provides several parameters that can be adjusted to adjust the performance of the model. My early results were around 80% for both the single digit and multi digit datasets, but after adjusting the model's hyper parameters and refinements to the architecture the model was able to achieve much better results. Using the TensorBoard application I was able to gain insight into how my model was performing and make the necessary adjustments.

There are two primary classes of refinements that can be made. First, the architecture of the Neural Network, this is comprised of the general architecture has been designed, such as the number of convolution and pooling layers, ideal weights and biases can than be discovered. I came to the previously discussed architecture after some trial and error. I started with lower depth values for the convolutions of [16, 20, 20, 10] but found using better hardware and setting greater depths [48, 64, 128, 160] provided over 5% better results.

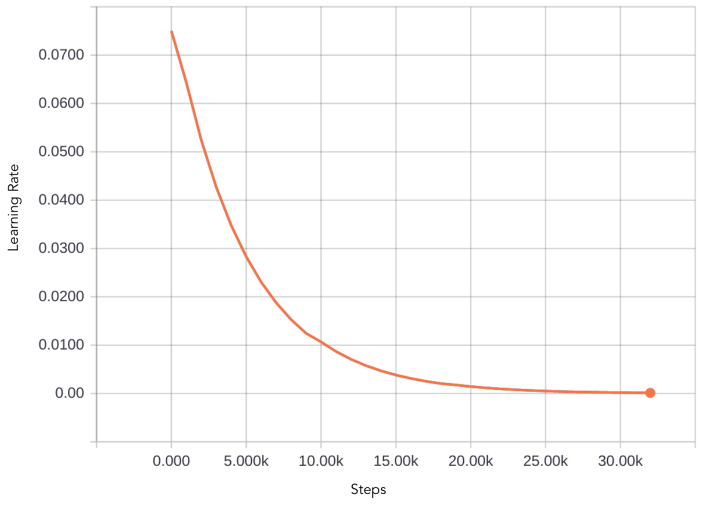
Second, after the architecture is affirmed the next major adjustments is to the learning hyper-parameters. This is the rate as to which our model can gain new insights and measures how much the current situation affects the next step. A stepped decay is aded to reduce the learning rate every epoch, until validation set stops noticeably improving.

Hyper paramaters:

I. Neural Network Architecture
   1. Number of Filters.
   2. Shape of Filter , depth of the output volume
   3. Types of pooling.

II. Training Hyper-Parameters
  1. Length of Training (Number of Epochs)
  2. Initial Learning Rate
  3. Decay Rate
  4. Dropout Keep Rate

Figure 9. Learning Rate



I found that adjusting the learning rate gave me the most effective performance gangs. I began measuring my model's performance after 256 epochs (70k steps), but later discover that with the right learning rate the model was converging closer to 128 epochs(35k steps). The initial learning rate is set to 0.5 with a staircased decay of 0.95. I found this to be the best rate through experimentation. Setting the value too high causes the classier to converge earlier but does not learn all of the intricate features, and setting this value too low also causes the loss function to never converge. Through fine adjustment of the learning rate I was able to increate the test accuracy about 3-4%

A dropout layer with a 85% keep rate for both the single digit and multi-digit models, but after some experimentation this layer was removed from the multi-digit model because I found in adversely affected performance and would lose nearly 2% accuracy in testing. I believe this is because there is a large variance present in the dataset an overfitting is not occurring, thus drop out was merely reducing the training size.

4. Results

4.1 Model Evaluation and Validation

My initial attempts achieved about about 92% after 128 epochs on the single digit transcription, and 94% accuracy on multi-digit transcription after 128 epochs. I used the work of the google team as decried in the Goodfellow paper as an initial guide to build the model. As you can see from the diagrams below the validation set for the single digit appears much smoother, I believe this is due to the difference in batch sizes. The single digit model uses a batch size 256 and the multi-digit model uses a batch size of just 32, to allow the dataset to fit into the GPU's memory.

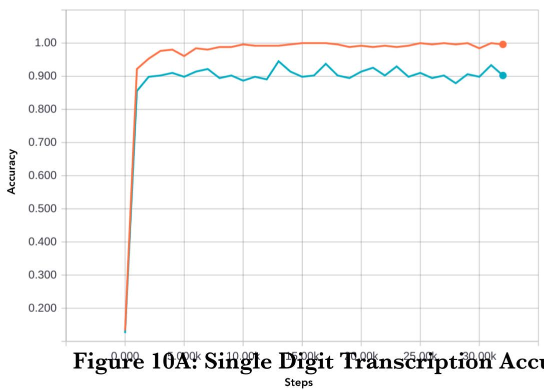


Figure 10A: Single Digit Transcription Accuracy

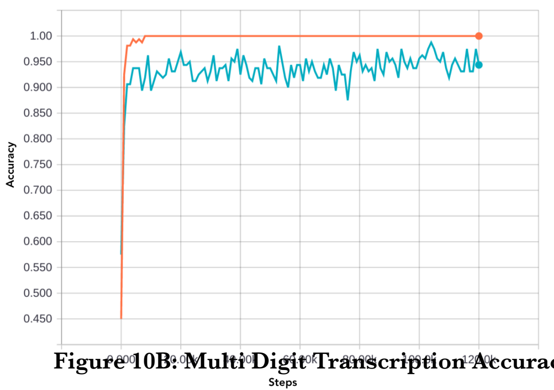


Figure 10B: Multi-Digit Transcription Accuracy

Overall I believe this model performs quite well given its simplicity.

4.2 Justification

After training the model and running the benchmarks against the training dataset I found some interesting results. My results are well within my expected range and after some fine adjustment of the weights and learning rates they even exceeded my initial expectations. One particular surprise, the accuracy was slightly higher on the multi-digit decoder compared to the single digit decoder. The accuracy of the single digit transcription is 92% while the accuracy of the multi digit reader is 94%. While I am very pleased with success of this simple approach I believe while there is room for improvement. The results show that prove that multi digit transcription is possible using a simple a CNN.

5. Conclusion

5.1 Free-Form Visualization

After the training was completed I decided to further experiment with my model using my own real world images. The images below were captured while I walked around my neighborhood. I tested on a variety of digit-sequences captured in the real world not just house numbers. The numbers were then cropped using approximately the same technique used to feed the training model.

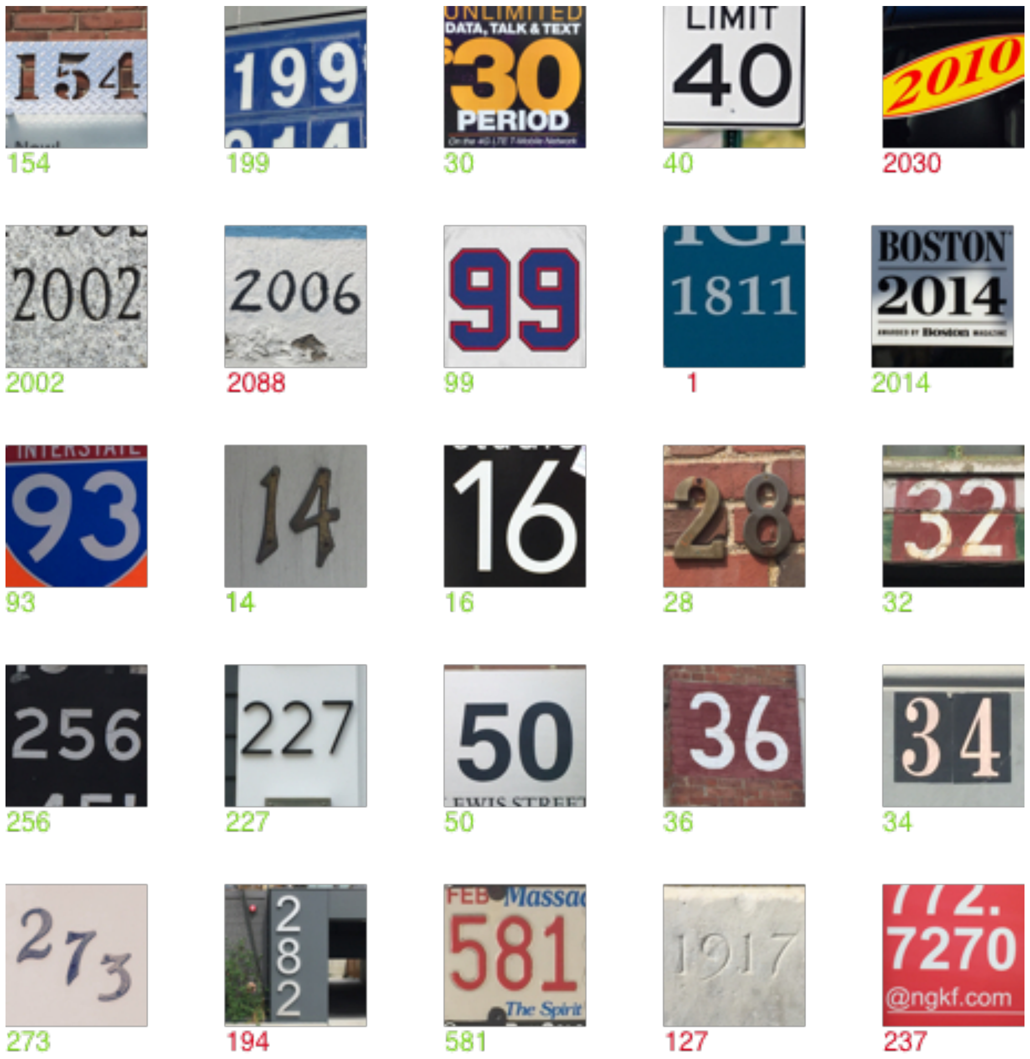


Figure 11: Free form Visualization of digit sequences.

Figure 11 indicates we are able to achieve good results using the model trained solely on house numbers to transcribe other kinds of digits captured in the real world. The green numbers indicate an accurate transcription while red number indicate an incorrect transcription. One issue of particular interest to note is placement of the digits within the image are significant. While horizontal numbers achieve a good accuracy number aligned vertically score a low accuracy, I believe this issue with orientation could be overcome by adding vertically oriented numbers to the training set.

5.2 Reflection

Convolution Neural Networks have a wide variety of uses, built up from small concepts they are powerful when applied to a visual recognition task. Training on a synthetic model then sharing of the insights / weights is a simple but powerful concept. Re-using same filter is used for each pixel in the layer; this makes the network memory efficient and performant.

This project can be summarized by the following steps:

1. Download and preprocess a collection of images for training a single classifier.
2. Experiment on training a classifier on synthetic for a single digit well cropped 32 pixel by 32 pixel images.
3. Save the weights of the convolution and pooling layers for reuse. The single digit classifier was trained using various hyper-parameters until a best fit was found.
4. Building on the previous work, the convolution and pooling variables are reused to train a multi digit sequence reader.
5. Images are captured walking around my neighborhood then cropped in a manner similar to the training images and fed into the model for evaluation

I found the most intriguing and difficult aspect of the project is figuring out how to reuse the Convolution and Pooling layers form the initial single digit reader and apply it to detect spatial locations of digits the multi digit model. Once this issues was resolved, I realized the simplicity and power of CNNs, training a single model could preform complex tasks ranging from classification to localizing an ordered sequence of digits.

5.3 Improvement

While we have shown with a relatively short amount of code that it is possible good results in decoding sequence of digits real world images, this system does have a number of drawbacks.

- Currently it only works with digits unto certain pre-determined length, 5 digits in this example case. Each separate digit classifier requires its own separate weight matrix. This could potentially create a issue when running on mobile devices or low powered embedded devices. The memory cost of this algorithm may prove to be too expensive. As the size of the maximum digit sequence increases the memory cost of the algorithm increases.

- A deeper network with more training example and more randomization would show promising results; this work was primarily an academic exercise to learn about Convolution Neural Networks and Deep Learning. I believe the results for this project show promising results and leave some room for expanding the accuracy of the model.

- The classifier requires some specificity around the input image. First the image must be of a specific dimension (64 pixels 64 pixels). The centroid of the digit sequence must be near the center of the image, and some padding must be present. The classier has show to be highly sensitive to these constraints. I think a further addition to the project could include a detector to find the region of image then run the classifier on the region of interest. I think another CNN like TensorBox[https://github.com/Russell91/TensorBox] could be used to propose the candidate image segments, giving the image recognition task more flexibility in input images.

References:

[1] Goodfellow Ian J, Bulatov Yaroslav, Ibarz Julian, Arnold Sacha, Shet Vinay. Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks Google Inc., Mountain View, CA.

[2] Li, Fei-Fei, Karpathy Andrej, Johnson Justin . Lecture 8: Spatial Location and Detection Lecture 8. CS231N http://cs231n.stanford.edu/slides/winter1516_lecture8.pdf

[3] LeCun Yan, Cortes Corrina, Burges Cristopher J.C. THE MNIST DATABASE of handwritten digits. http://yann.lecun.com/exdb/mnist/

[4] Netzer Yuval, Wang Tao, Coates Adam, Bissacco Alessandro, Bo Wu, Ng Andrew Y.. Reading Digits in Natural Images with Unsupervised Feature Learning. NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011. (PDF)

[5] Olah ,Christopher. Conv Nets: A Modular Perspective. http://colah.github.io/posts/2014-07-Conv-Nets-Modular/ posted on Posted on July 8, 2014

[6] Ren Shaoqing, He Kaiming, Girshick Ross, Sun Jian. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.

[4] Servant Pierre, Chintala Soumith, LeCun Yann. The Courant Institute of Mathematical Sciences - New York University. Convolutional Neural Networks Applied to House Numbers Digit Classification http://arxiv.org/abs/1204.3968v1

[7] Srivastava Nitish, Hinton Geoffrey, Krizhevsky Alex, Sutskever Ilya, Salakhutdinov Ruslan. Dropout: A Simple Way to Prevent Neural Networks from Overfitting.

[8] Wei Yunchao, Wei Xia, Huang Junshi, Ni Bingbing, Dong Jian, Zha Yao, CNN: Single-label to Multi-label.