

# Machine Learning Engineer Nanodegree

## Capstone Project

---

### Multi-Digit Number Recognition with TensorFlow

Robert Cottrell  
January 29, 2017

## I. Definition

### Project Overview

Convolutional deep learning networks have had a dramatic impact on the image recognition space. Even simple models are able to make highly accurate predictions on datasets like the MNIST database of handwritten digits.<sup>1</sup>

The MNIST images, however, are fairly uniform and relatively simple. Recognition on images taken under real world conditions is a considerably more difficult and as yet unsolved problem. Differences in camera quality, lighting conditions, backgrounds, and angles are contribute to make the problem more difficult. And even more challenging is recognizing multiple objects, like sequences of digits in a number, and interpreting them as a single entity.

There are many uses for being able to identify multi-digit sequences in the real world. The use case that inspired this project was Google's use of Street View images to identify house numbers to annotate addresses on a map.<sup>2</sup> The USDOT has used image recognition to monitor commercial trucks for regulation compliance on freeways.<sup>3</sup> Image recognition has even been used to assist sports commentators by tracking soccer players by their jersey numbers.<sup>4</sup>

In this project we will attempt to recreate the techniques and methodologies used by Goodfellow, *et al.*, to recognize mutli-digit numbers. While we will try to remain faithful to the

---

<sup>1</sup> LeCun, Yann et al, "The MNIST Database of Handwritten Digits." Retrieved January 08, 2017, from <http://yann.lecun.com/exdb/mnist/>.

<sup>2</sup> Goodfellow, Ian et al. "Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks" arXiv:1312.6082 [cs.CV] 2014.

<sup>3</sup> Bulan, Orhan et al. "USDOT Number Localization and Recognition From Vehicle Side-View NIR Images." Retrieved January 20, 2017 from [http://www.cv-foundation.org/openaccess/content\\_cvpr\\_workshops\\_2015/W05/papers/Bulan\\_USDOT\\_Number\\_Localization\\_2015\\_CVPR\\_paper.pdf](http://www.cv-foundation.org/openaccess/content_cvpr_workshops_2015/W05/papers/Bulan_USDOT_Number_Localization_2015_CVPR_paper.pdf).

<sup>4</sup> Gerke, Sebastian et al. "Soccer Jersey Number Recognition Using Convolutional Neural Networks" Retrieved January 20, 2017 from [http://www.cv-foundation.org/openaccess/content\\_iccv\\_2015\\_workshops/w21/papers/Gerke\\_Soccer\\_Jersey\\_Number\\_ICCV\\_2015\\_paper.pdf](http://www.cv-foundation.org/openaccess/content_iccv_2015_workshops/w21/papers/Gerke_Soccer_Jersey_Number_ICCV_2015_paper.pdf).

original project, we will also explore how to take maximal advantage of the modern TensorFlow<sup>5</sup> machine intelligence library.

## Problem Statement

The objective of this project is to develop and train a deep learning convolutional network that will be able to identify a sequence of digits comprising a number from an image in a single step. This is in contrast to traditional methods of that divide recognition into a series of stages that might involve scanning an image for possible candidates, segmenting and classifying the individual digits, and then assembling those digits back into a whole number.<sup>6</sup>

The scope of this project will be limited to identify numbers between one and five digits. In addition the model should be able to recognize when there are no digits present in the image as well as when more than five digits are present.

This is, from a high level, a simple image classification problem. However the naive approach of building a softmax classifier with 100,000 classes for each possible number up to five digits long, as well as two additional classes for no digit or more than five digits, will not be successful. Not only would such a classifier be computationally intractable to train, but it would also be infeasible to collect enough samples of each class to fully train the classifier.

Instead, we will break the problem down into several simpler ones. The model will be trained to output five different softmax classifiers, one for each digit to be represented. Even this, however, will not be sufficient to solve this problem. The classifiers would need to be trained to also recognize the absence of their respective digit. This could, however, lead to inconsistent results if, for example, the second digit classifier indicated no digit but the third digit classifier believed one to be present.

Our ultimate solution, therefore, will introduce a sixth softmax classifier output that will be responsible for counting the number of digits, or whether there were no digits or more than five digits identified. Because this counter classifier will be responsible for determining the length of the number, the individual digit classifier will not be required to learn whether a digit for their respective position is available or not.

## Metrics

The evaluation metric for this project will be the transcription accuracy on the test images in the SVHN dataset. Accuracy will be defined as correctly predicting the full house number in the image. No partial credit will be given for identifying part of the number. The model must correctly predict the number of digits present as well as the identity of each of those digits. The model will not, however, be queried for digits not present in the result. For example, if the model has predicted a three digit number, the output of the classifiers for the fourth and fifth digits will not be constrained or otherwise required indicate a “no digit” state.

---

<sup>5</sup> TensorFlow. Retrieved January 20, 2017 from <https://www.tensorflow.org>.

<sup>6</sup> Yang, Xuan et al. “MDig: Multi-digit Recognition using Convolution Neural Networks on Mobile.” Retrieved January 08, 2017 from <http://web.stanford.edu/class/cs231m/projects/final-report-yang-pu.pdf>.

To express our confidence in the model's prediction, we will combine the probabilities reported by the individual classifiers:

$$Pr(S = s | X) = Pr(L = n | X) \prod_{i=1}^n Pr(S_i = s_i | X)$$

Here we let  $\mathbf{X}$  represent the input image and  $\mathbf{S}$  the predicted number sequence. We define  $\mathbf{S}$  as a collection of random variables  $\{S_1, S_2, \dots, S_n\}$  where  $S_i$  is the  $i$ th digit of the number. Each member  $S_i$  in  $\mathbf{S}$  is considered to be independent of the other members. We also define  $\mathbf{L}$  as another random variable that stores the length of number, along with additional values for no digits or more than five digits.

It should be noted that this is an inherently class imbalanced classifier. Not only are the different lengths of numbers unbalanced—there are many more five digit numbers than there are one digit numbers—but the distribution of house numbers in the real world will also tend to skew more toward mid length numbers rather than shorter or longer ones.

Accuracy is often not a good metric for class imbalanced problems and other metrics like the  $F1$  score that trades off precision and recall are often preferable. We feel justified, however, to continue using accuracy for this problem. The large number of possible classes and the interdependence of the digit counter classifier with the positional classifiers make it difficult to exploit strategies like predicting majority classes.

## II. Analysis

### Data Exploration

The Street View House Numbers (SVHN) dataset<sup>7</sup> contains images of house numbers from Google Street View images.<sup>8</sup> The data set comes in two flavors. We will use the use the original images which feature the whole house number. The images are all of various sizes and quality and taken from different angles. The dataset is divided into three different groups: 33,402 training images, 13,068 test images, and 202,353 extra but less difficult images that will be used as additional training data.

In addition to the images, the dataset also includes metadata identifying the ground truth label for the house number as well as the bounding boxes for each individual digit in the number. The metadata was stored in a MATLAB data file, but we were able to develop a Python script to extract the information from the underlying HDF5 data.

---

<sup>7</sup> "The Street View House Numbers (SVHN) Dataset." Retrieved January 08, 2017, from <http://ufldl.stanford.edu/housenumbers/>.

<sup>8</sup> Netzer, Yuval et al, "Reading Digits in Natural Images with Unsupervised Feature Learning", *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*. Retrieved January 23, 2017 from [http://ufldl.stanford.edu/housenumbers/nips2011\\_housenumbers.pdf](http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf).

This preprocessing script looped through each image in the data set, looked up the label for the number, and extracted the bounding boxes for the digits. A new bounding box was calculated that would encompass all of the individual bounding boxes to fully encompass the house number. This bounding box was then augmented by 30% in both the horizontal and vertical directions. The image was then cropped to this bounding box and the resulting image was resized to 64x64 pixels. The resized image was then saved to disk along with a new metadata file that stored the path to the resized image as well as the label of the house number to a CSV file.

Original and Preprocessed Images

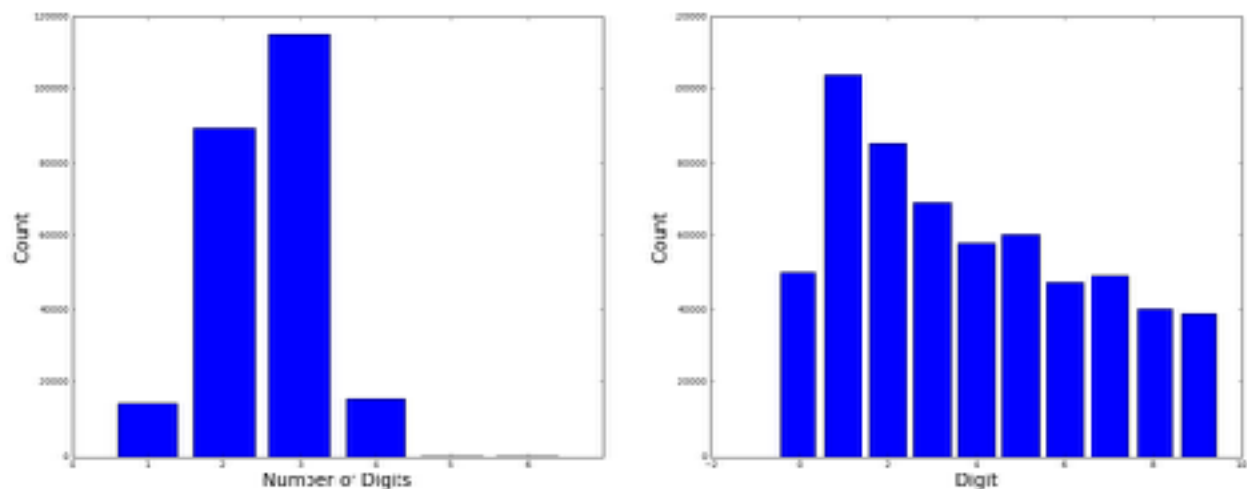


The preprocessing script is available from the GitHub repository accompanying this report as `preprocess.py`.

## Exploratory Visualization

Now that the data has been downloaded and preprocessed, we can begin to explore the distribution of the dataset.

Distribution of Digits



As expected, the distribution of data is highly imbalanced. While there are significant examples of two and three digit house numbers, there are substantially fewer for one and four digits. There are only a handful of five digits numbers and exactly one number with more than five digits. There are no examples of images without any digits. As a result, we can expect that counter classifier may have difficulty generalizing to predict the absence of number, or numbers with five or more digits.

The distribution of individual digit values within the dataset is more balanced. While there are more examples featuring the digits 1, 2, and 3, we have a many examples of each class of digits. The classifier should have plenty of examples of each kind to extract features from. In total, the model will be exposed to 604,388 individual digits.

When faced with a class imbalance, there are two options that are frequently taken. Examples from the overrepresented classes may be removed to even out the distribution, or data augmentation may be used to artificially boost underrepresented classes. Because we would like to keep as much training data as possible to allow the model to learn from as many real world examples as possible and will therefor choose to combat class imbalance through data augmentation.

We will use the same data augmentation technique used by Goodfellow *et al.* Each of the 64x64 pixel images will be randomly cropped to a final 54x54 pixel size for training. This will allow each sample to generate up to 100 variations of itself. In addition to fighting imbalance, the random cropping will also help to improve the translational invariance of the model. To maintain predictability and repeatability of model validation and predictions, however, we will always use a center crop for inference.

## Algorithm and Techniques

The classifier to be trained will be a deep learning convolutional neural network. This architecture is the current state of the art for image classification and detection. Unlike simpler image classifiers, the network will output to six separate softmax classifiers. One classifier will be responsible to counting the number of digits detected, while the other five classifier will classify the the digit at their respective position within the number.

Convolutional networks have a number of different architectural decisions and parameters that must be selected and tuned. For this project, we will be guided by the choices made by Goodfellow *et al.*, but will make adjustments where necessary to simplify the problem or adapt it to the capabilities of the TensorFlow library.

Some of the decisions to consider are:

- \* Number and types of layers. Here we followed the same architecture as Goodfellow, with one notable exception. The Goodfellow model used a locally connected layer between the initial convolution layers and the final fully connected layers, but TensorFlow does not currently support locally connected layers. The loss is unfortunate, because in theory a locally connected layer is able to exploit the positional relationships of pixels in the input. This would have been able to make it easier to localize the starting positions of the digits. We decided to simply leave it out and train with a model that had one fewer layer.

- \* Activation functions. The choice of activation function introduces the nonlinearity into the model that is required to let the network model arbitrary functions. We will be using exponential linear units (ELU), a recent development that has shown itself to perform better than many of the more traditional functions.<sup>9</sup> In contrast, the Goodfellow model used ReLU units for all hidden layers except the first layer, which used a maxout activation.<sup>10</sup>
- \* Dropouts. Using dropout layers is a powerful way to prevent overfitting a model by randomly setting a fixed percentages of outputs to zero during training. This builds robustness into the model by forcing it to build redundant representations because it cannot count on the presence of any given neuron to form its predictions. We will use a dropout layer after each hidden layer in the network and set their keep ratios to 50% to maximize the effect of the dropouts.
- \* Optimizer:. We will use the Adam optimizer, a modern stochastic gradient descent optimizer that has shown itself to work well under a variety of different conditions. An Adam optimizer is able to adapt its learning rate based on moment to more quickly converge on a solution.
- \* Loss function. This is the objective that the optimizer will seek to minimize. Each of the output classifiers will seek to minimize the softmax cross-entropy between the predicted and expected values.
- \* Batch size. The choice of batch size must balance the efficiencies gained by training on multiple inputs at the same time with the requirements of holding all data within the GPU's memory. We will use a batch size of 64, which should give a reasonable compromise.

A secondary objective of this project is to explore some of the more advanced features of the TensorFlow library. TensorFlow exposes an advanced data flow modeling environment that can be used to prototype and develop new or experimental algorithms. In addition it has a rich library of data processing and transformation functions that can be used to developed more sophisticated applications.

One challenging aspect of the problem as we have defined it is that we would like to train the classifiers in such a way that absent digits do not contribute to the back propagation learning. That is, if we are training on a batch of three digit numbers, we do not wish to alter the model based on the outputs of the fourth and fifth digit classifiers. This is nonstandard behavior and will require custom development to implement.

## Benchmark

Goodfellow, *et al.*, reported a transcription accuracy rate of 96.03% on the SVHN test set for their best model. This will be the aspirational target, achieved by a team of deep learning experts with significant expertise and computational resources at their disposal. An independent

---

<sup>9</sup> Clever, Djork-Arné et al. "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)" arXiv:1511.07289 [cs.LG] 2016.

<sup>10</sup> Our first model also used ReLU activation units, but numerical instability caused by a too-high learning rate shifted the outputs of the convolution layers to negative values until the network finally died. Replacing ReLU with ELU solved allowed the model to continue to learn even with this instability. Once the learning rate was corrected, we could have returned to ReLU units but chose to stay with the ELU units which had proved themselves to be very robust. The maxout activation used by the first hidden layer was not available in TensorFlow, so we chose to have our layer use the same activations as the others.

implementation was able to train a model to 95.4% accuracy.<sup>11</sup> We will consider any result over 95% to represent a successful recreation of the experiment.

Goodfellow also proposes an additional metric: coverage at 98%. This represents the number of predictions for which the classifier has a predicted confidence of greater than 98%. These high-confidence predictions could be considered as being of human-level accuracy and automatically included in maps without further human intervention. Goodfellow reports coverage at 95.64%. We have no a priori intuition about this metric.

## III. Methodology

### Data Processing

The steps necessary to download and preprocess the SVHN images were previously documented in this paper. In this section we will document the additional preprocessing steps taken on the data before feeding it to the model for training.

To effectively tune and validate the progress of a model, it's important to have a set of validation data which can be used to measure the effectiveness of changes. This validation set should be separate from the test set that will be used to evaluate the final model. Even though the validation data is never directly used to train the model, the model will indirectly begin to “overfit” to the validation data by making decisions that improve the validation score. By keeping the test set held out until the end, we get a better sense of how well the model is likely to generalize to new unseen data.

To select the validation set, we performed a shuffled, stratified split on the “train” images in the SVHN data set. Shuffling the images is important to ensure that we remove any biases based on time or ordering of the data set. Stratification is important to ensure that the two sets contain roughly the same proportion of each class of data. We chose to use only the harder “train” images to ensure that the validation set would be of similarity difficulty to the test set. Once the validation images were selected, the “extra” images were then appended to the remaining “train” images to form the training data set.

The training images were then divided into groups of one, two, three, four, and five or more digit numbers and written as CSV files to disk. The validation and test images were also written as CSV files. Separating the training data into groups of different lengths will allow us to select batches of images that are all of the same length. This will be necessary to implement our custom loss function. The validation and test images will only be used for inference, so there is no need to separate them by length.

Each CSV data file contains a path to the image and a value for each classifier (number of digits, first digit, second digit, third digit, fourth digit, fifth digit). If the number was longer than five digits, only the first five digits were recorded. If the number of digits was smaller than five, then the labels for the empty digits were set to zero. These zeros, however, are simply

---

<sup>11</sup> “Multi-Digit Number Extraction from Images Using Deep Learning for Torch.” Retrieved January 08, 2017, from <http://itaicaspi.github.io/SVHN-Multi-Digit-torch/>.

placeholders and never queried. By reusing zero as a placeholder, we are able to use TensorFlow's `sparse_softmax_cross_entropy_with_logits` function without introducing an additional "no digit" label.

The script used to divide the data into train, validation, and sets as well as dividing the training data into batch sets is available from the GitHub repository accompanying this report as `split.py`.

Encoding the training inputs and labels in a CSV file enables us to use TensorFlow's data processing pipeline to incrementally load, preprocess, shuffle, and batch data in parallel on background threads. This will allow us to use a larger data set than will fit in memory without slowing down the actual training of the model.

The first step of the processing pipeline is to open the CSV file and parse each line, one at a time. We then use a TensorFlow operation to load the image file into memory and then use another operation to decode the PNG data into a 64x64x3 array of pixel values. Then another operation takes a random 54x54x3 crop of the image. Finally we use another TensorFlow operation to normalize the image by subtracting the mean pixel value and dividing by the standard deviation.<sup>12</sup>

The decoded and preprocessed images are then added to another queue, which will be responsible for shuffling the data and returning batches. We will actually use five separate input pipelines and then switch at runtime to return batches containing images with numbers of the same length.

The code used to generate the data input pipeline is available from the GitHub repository accompanying this report as `input.py`.

## Implementation

The visualization of the model graph in TensorBoard, shown below, gives some indication of the complexity of the model we will be training.

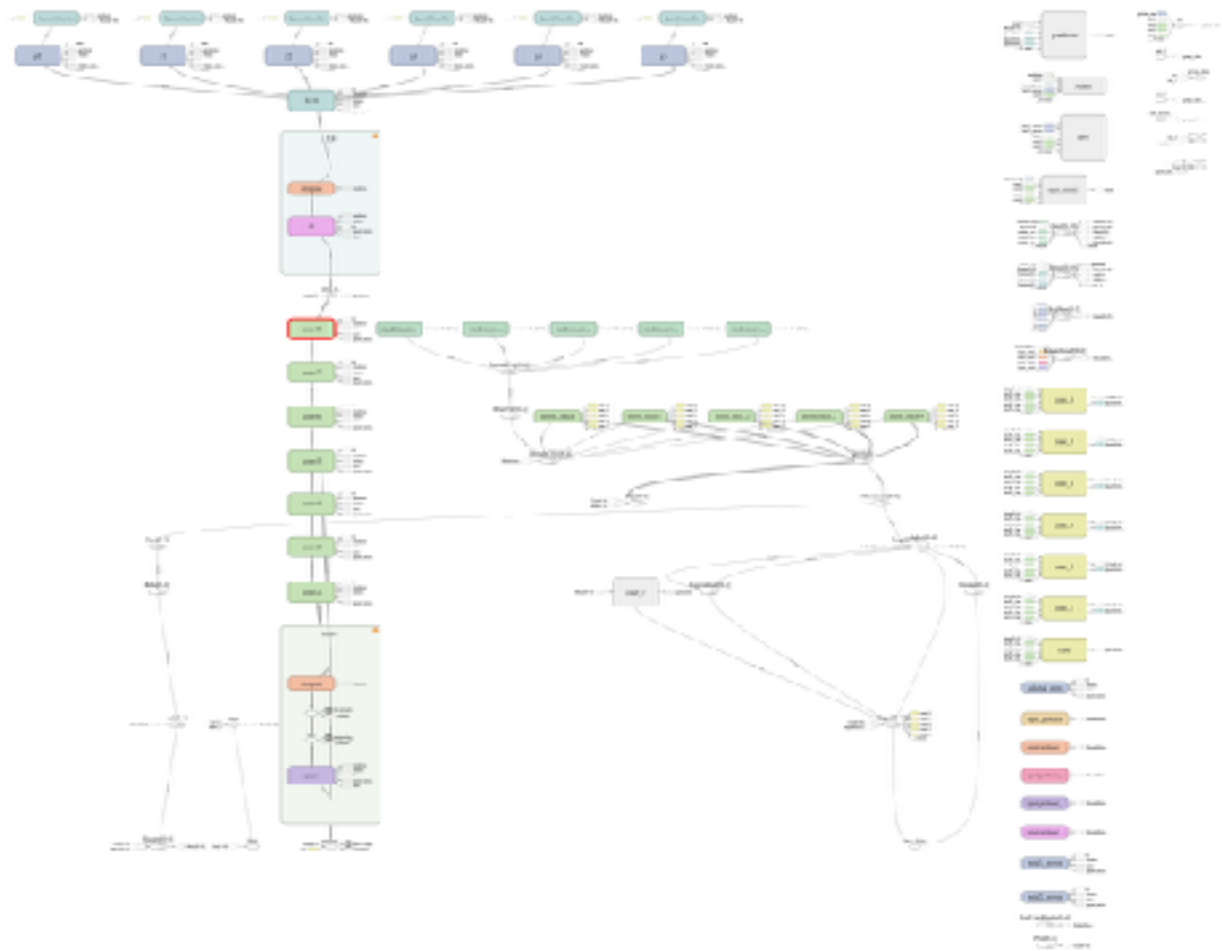
Our model consists of eight convolution layers. Each convolution layer consists of a 5x5 convolution filter with an ELU activation function. The depths of the filters are 48, 64, 128, 160, 192, 192, 192, and 192. Each convolution is followed by a max pooling layer with SAME padding and a 2x2 kernel. The pooling layers have alternating strides of 2x2 and 1x1. The pooling layer is then followed by a local response layer with default TensorFlow parameters. The local response layer is then followed by a dropout layer with a 50% dropout rate.

The convolution layers are followed by two fully connected layers, each with 3,072 nodes and an ELU activation function. Each fully connected layer is followed by a dropout layer with a 50% dropout rate.

---

<sup>12</sup> Goodfellow, et al., merely subtracted the mean pixel value but otherwise did not appear to rescale the value. We chose here to use TensorFlow's built in image standardization operation which also divides by the standard deviation.





The model terminates in six softmax classifiers. The first softmax classifier counts the number of digits and has values zero through five plus a final value for more than five digits. The remaining five classifiers return the digit at their respective position and have the values zero through nine. If the number has fewer than five digits, the remaining classifiers are allowed to float unconstrained.

The code used to generate the model and scoring functions is available from the GitHub repository accompanying this report as `model.py`.

Batches are selected so that each image in a batch has a number with the same number of digits. This will allow us to select at runtime a loss function that considers only those digits. The counter classifier and the digit classifier for each digit position in the batch will contribute to the loss function and participate in backpropagation. The unused digits will not contribute to the loss function and will not participate in backpropagation.

Given the imbalance in the length of numbers, we will use a novel approach to select batches. Our method draws inspiration from reinforcement learning to provide a type of exploit/explore tradeoff for batch selection. After each run, we will track the training accuracy of the batch and maintain a decaying average of those accuracies. We will then “exploit” the lower performing

sizes to train the model where it is weaker. But we will also continue to “explore” the higher performing sizes to ensure that they are still performing well.

While the algorithm to make these batch selections has not been tuned, it's initial use shows significant promise. At the start of training, before the model has been able to learn anything, the different sizes are chosen with equal probability. As the model begins to learn, however, underperforming sizes are sampled with greater frequency to help them catch up. As the model finishes training, selection returns again to a more random rate to better refine the final model. This approach, which lets the model learn how to best select the data it trains on, seems to be more fair and more flexible than arbitrarily choosing certain groups to oversample.

The code used to generate the model and scoring functions is available from the GitHub repository accompanying this report as `train.py`.

## Refinement

Tuning the learning rate turned out to have the most significant impact on the results of the project. We chose to use the Adam optimizer, a modern variant of stochastic gradient descent that is able to adapt the learning rate based on the momentum of the training. The Adam optimizer has a several tuning parameters that default to values that have proven themselves to be optimal over a wide variety of cases. This model was not one of those.

Our initial run used the Adam optimizer with its default learning rate of  $1e-3$ . After approximately 10 hours of training, the outputs of the convolution layers, which had been steadily sliding into negative territory, finally crossed over to be fully negative. As a result, the ReLU activation functions died and the model stopped producing useful results. Validation tests returned effectively random predictions.

To compensate for the negative slide in outputs, the ReLU activations were replaced by ELU activations<sup>13</sup>, which are able to work with both negative and positive outputs. After approximately three days of training, the model reached an accuracy around 92.5% and began to plateau. Training was stopped and then restarted with a lower learning rate to further refine the score. We noted that as soon as training resumed, the spread in the output values of the classifiers immediately stabilized. This suggested that the learning rate must have been too high from the start.

For our third run, we used a learning rate of  $1e-5$ . This time the model trained at an even faster rate. Accuracy reached over 94.5% before noticeably slowing and continue to increase through 95% and beyond. Finally, we started a fourth run with the learning rate set to  $1e-6$ . With this much lower learning rate, the model was still able learn, but progression was significantly slower and it took over six days to reach 94% accuracy.

We decided to use a learning rate of  $1e-5$ , which provided a good balance between training speed and final model accuracy.

---

<sup>13</sup> Once we determined that the numerical instability in the model resulted from a learning rate that was set too high, we could have returned to using ReLU activations. However, we decided to keep the ELU activations, as they had proven themselves to be very robust.

We note that it was only possible to troubleshoot and refine the issues related to the high learning rate because of the extensive summary statistics that we added to the model that we were then able to visualize with TensorBoard while the model was training.

## IV. Results

### Model Evaluation and Validation

Training the final model took approximately three days with a current generation NVIDIA Titan X (Pascal) GPU.

Training was halted after there was no significant improvement in validation accuracy for several hours. Choosing the optimal moment to stop training, however, was somewhat difficult as validation accuracy could fluctuate by as much as 0.1% from one 10-minute checkpoint to the next.

The final evaluation on the SVHN test set is shown below.

	This Project	Goodfellow, <i>et al.</i>
Accuracy	95.22%	96.03%
Coverage at 98%	85.47%	95.64%

Unfortunately, the coverage metric was not added to the evaluation script until after the final model had been trained. As a result, we do not have a good intuition for how the coverage rate changed over time.

The code used to evaluate the model is available from the GitHub repository accompanying this report as `eval.py`.

### Justification

Although the accuracy results of this experiment did not match the benchmark set by Goodfellow, *et al.*, they came close enough to validate the approach that we took building the model. It's likely that additional experimentation fine tuning the model and adjusting the hyperparameters could lead to even better results.

The large difference in coverage rates is harder to explain, especially given how similar the accuracy rates were. However, the coverage rate isn't a completely objective number as it relies on the model's own assessment of its prediction. The model's reported confidence score, for example, could depend on how much regularization was applied to its parameters.

It's possible that our model actually is much less likely to make a prediction with human-level accuracy. On the other hand, it might just be that our model is more conservative in the estimates it gives about its confidence.

Our study of the coverage rate showed that all predictions on the test set with 98% confidence or more were in fact accurate classifications. This is encouraging because it suggests that we may be justified in building automated workflows that can accept that the model's classification is correct without requiring human verification first.

It should be noted that one significant difference between the Goodfellow model and our own was the inclusion of a locally connected layer between the convolution and fully connected layers. This could help to explain why our model was not able to attain the same level of accuracy.





## V. Conclusion

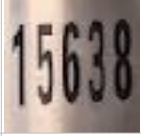


### Free-Form Visualization

While the performance of our model was benchmarked by its accuracy predicting the values of the numbers in the SVHN test set, it would be interesting to see how it would generalize to additional images found in a real world setting.

We collected six images from locations in downtown San Francisco. The images were selected in part on features that we would consider difficult for the model to classify correctly. In addition, we also included a finger-drawn sketch from a mobile app to test the model with an entirely new type of image.

The results of these world world samples are shown below.

IMAGE	PREDICTION	DESCRIPTION
	LABEL: <b>300</b> (100%) DIGITS: <b>3</b> (100%) <b>3</b> (100%) <b>0</b> (100%) <b>0</b> (100%) <b>0</b> (74.2%) <b>0</b> (100%)	Street sign indicating address ranges.
	LABEL: <b>575</b> (99.8%) DIGITS: <b>3</b> (99.9%) <b>5</b> (100%) <b>7</b> (100%) <b>5</b> (100%) <b>5</b> (40.9%) <b>5</b> (100%)	Office building.
	LABEL: <b>7447</b> (86.8%) DIGITS: <b>4</b> (94.7%) <b>7</b> (95.6%) <b>4</b> (100%) <b>4</b> (96.1%) <b>7</b> (100%) <b>4</b> (80.6%)	Partial license plate, at an angle.
	LABEL: <b>280</b> (90.1%) DIGITS: <b>3</b> (90.4%) <b>2</b> (99.9%) <b>8</b> (99.8%) <b>0</b> (100%) <b>1</b> (97.9%) <b>0</b> (86.1%)	Restaurant with number on glass window and bright menu in background.

	LABEL: <b>1538</b> (69.5%) DIGITS: <b>4</b> (100%) <b>1</b> (100%) <b>5</b> (100%) <b>3</b> (69.9%) <b>8</b> (99.4%) <b>8</b> (100%)	MUNI bus stop station ID.
	LABEL: <b>495</b> (99.0%) DIGITS: <b>3</b> (100%) <b>4</b> (100%) <b>9</b> (99.0%) <b>5</b> (100%) <b>5</b> (45.6%) <b>5</b> (87.8%)	Finger drawn sketch in iPhone app.
	LABEL: <b>160</b> (100%) DIGITS: <b>3</b> (100%) <b>1</b> (100%) <b>6</b> (100%) <b>0</b> (100%) <b>0</b> (73.6%) <b>0</b> (100%)	Office building.

These results showed that our model generalized surprisingly well to images that it had not seen before. It was able to correctly classify six of the seven images presented to it. In addition, even the failure case proved interesting.

Most of the images were predicted correctly and with an extremely high confidence rate. The first and last image, for example, were classified with a probability within rounding distance of 100%. Even the hand drawn number, with its badly segmented digits, was classified with a high probability confidence.

The restaurant sign was classified correctly even with the reflective glass surface and the distracting backlight behind the final digit. In this case, the model predicted the three digit length with less confidence, but still was very confident in the value of the individual digits.

The partial license plate was also an interesting given that the model was successfully able to classify the image even given its extreme rotation. While the model was trained with augmented data to improve the translational invariance of its predictions, no special steps were taken to address rotational invariance. Nevertheless, the model was able to correctly classify the number.

It should probably come as no surprise that the model had difficulty with the five digit MUNI bus stop number. With only 125 examples of five digit numbers, the model had very little with which to learn. However, it was still able to suggest the first two and final two digits of the number, expressing a significant amount of uncertainty about the third digit.

The code used to make prediction for new images is available from the GitHub repository accompanying this report as `eval.py`.

## Reflection

The successes of deep learning convolutional networks at image recognition are well known. However, at the start of this project we were skeptical that a single network would not only be able to recognize multiple digits in an image but also transcribe them in the correct sequence. Our results confirm that not only is this possible, but it is possible to achieve with a very high degree of confidence.

While it's unlikely that the approach taken in this project can scale to arbitrarily large sequences, it can still be a useful technique for a wide variety of problems. For example, by adding the ability to recognize letters in addition to digits, we could create a simple license plate or serial number classifier. Or, we could use the model generated here as an additional check for a more traditional OCR recognizer.

One of the most challenging aspects of this project was building a loss function that would satisfy our requirement that only the classifiers for the digits present in a number would contribute to the backpropagation steps. This had several follow-on consequences. First, we would need to select a different loss function at runtime for each number length. This, in turn, implied that our data batches would need to consist of numbers of the same length. And enable that, we would have to guarantee that we could select from different input pipelines.

Fortunately, TensorFlow has a flexible data flow model that allows even complicated operations to be represented as sequences of simple steps. Using TensorFlow's advanced flow control operations, we could create callable Python lambdas that would be invoked by TensorFlow at runtime to perform arbitrarily complex operations. We imagine that leveraging these abilities will become more important as we move beyond simple image classifiers.

Finally, we were surprised to discover the tremendous visualization capabilities provided by TensorBoard. Although using TensorBoard effectively required us to make changes to the way our modeled was built, we were rewarded with extensive visualizations that enabled us to track the progress of training as it progressed over several days. In particular, TensorBoard was instrumental in helping us track down problems caused by a high learning rate and then discover and fine tune a more reasonable value.

## Improvement

There are two major areas of improvement that we could explore moving forward. The first is improvements in the use of TensorFlow. The second is improvements to the architecture of the model.

As shown in our discussion of refining the learning rate, the visualizations created by TensorFlow can be invaluable to track the training of models and troubleshoot issues sooner rather than later. These visualizations, however, do not come for free. They require additional effort to annotate specific tensors to collect summary statistics. Naming tensors also becomes important to ensure that the summary statistics are organized in a coherent and understandable way. With the helper functions we used to construct our model layers, statistics for the weights and biases of layer parameters were created automatically. However, statistics for the input pipeline, classification results, and model progress were not as clear. Spending some additional time to ensure that statistics for these steps are properly generated would be extremely useful.

While we collected statistics for the main training application, periodic validation checks on the model were more ad hoc and no summary statistics were collected. Because of this, it was difficult to track when improvement of the accuracy of the validation set had tapered off. A more automated approach to validation testing, including the creation and display of summary statistics, would make it easier to decide when to stop training before risking overfitting.

It would also be incredibly valuable to exploit the distributed and multi-GPU capabilities of TensorFlow. With training of complicated models (including this one) now taking days or weeks to see results, being able to conduct training faster is extremely important. While we did have an additional GPU that could have been used for training, TensorFlow was not able to immediately take advantage of it. Instead, we would need to spend additional time to take more control over the training process. In particular, our training script would need to decide how to split training across the available devices and then integrate those results back into model at each training step.

There are a few ways we could improve the architecture of the model we used for this project. Because we tried to remain faithful to the architecture of the original Goodfellow model, we used a number of techniques that have since fallen out of favor and should probably be revisited. First the use of local response layers are not used as frequently now as they used to be. Instead newer techniques like batch normalization are being used to improve regularization of neuron outputs. Use of pooling layers to reduce dimensionality are now also being contested, with some models relying solely on padding and striding within convolution layer.<sup>14</sup> Even the necessity for dropout layers to combat overfitting is becoming less of a concern with the extremely large datasets now available to train models.

Given the way that our model reached a relatively high accuracy rate and then plateaued on both the training and validation sets suggests that we may have hit a limit on the complexity of features our model was able to encode. We could attempt to address this by adding additional convolution layers, expanding the number of filters in existing layers, or even trying some novel new architectures. Microsoft's winning ImageNet model, for example, uses residual layers to make features detected at lower layers directly available to higher levels.<sup>15</sup>

Another option that could be explored is the use of alternative color spaces. While the RGB color scheme that we used to train this model is useful to represent colors in computers, it doesn't necessarily follow that it is the best color space to use for image recognition. Other colorspaces such as grayscale, YUV, or HSV, among others, has shown themselves to be useful in various computer vision techniques. We don't even need to arbitrarily select a colorspace. Prepending a 1x1x3 convolutional filter to the start of the model would give the model the flexibility to choose the most useful color transformation itself.

Finally, we could also explore additional techniques to augment the data. In this experiment, we limited ourself to randomly cropping the images. Additional techniques like applying small amounts of random rotation or skew should allow the model to learn more flexible representations of digits. With the additional processing power afforded by modern CPUs and GPUs, additional data augmentation could be applied without unduly extending the training time.

Multi-digit image recognition is not yet a solved problem, but there are still a number of approaches we could use to improve our results.

---

<sup>14</sup> Bojarski, Mariusz et al., "End to End Learning for Self-Driving Cars" arXiv:1604.07316 [cs.CV] 2016.

<sup>15</sup> He, Kaiming, et al., "Deep Residual Learning for Image Recognition" arXiv:1512.03385 [cs.CV] 2015.