Machine Learning Engineer Nanodegree

Digit Recognition in Natural Images

Capstone Project

Kenny Deneweth | January 2017

Project Overview

Projections for low vision for people in the United States are predicted to increase 72% by 2030, according to the National Eye Institute (1). That means approximately 5 million people will have a harder time doing everyday activities such as reading, shopping and cooking.

However, the future is bright. With advancements in deep learning we are closer than ever to solving problems of these sorts. Computers of all shapes and sizes will be able to help reduce the challenges faced today by the visually impaired. For example, a trained algorithm could be placed inside an app on a phone or even inside wearable tech, like Google Glass.

Problem Statement

Optical character recognition (OCR) in documents is widely studied and has virtually been declared as a solved problem (2). Moreover, while reading digits in natural images is a relatively trivial task for humans, it is a complex problem for machines. To solve this problem a convolutional neural network (CNN) will be used to identify a sequence of digits in an image.

I. Preprocessing

1. The CNN algorithm will be trained using the Street View House Numbers (SVHN) Dataset provided by Stanford University.
2. The images have to be cropped, resized, converted to grayscale and normalized before being fed to the CNN.
3. Create proper test, train, and validation sets.

II. Algorithm development

1. The algorithm will first be trained using the TensorFlow library in Python.
2. Training will be done using the training and validation sets. The test set will be held to use at the end of training get a final accuracy.
3. The model will be optimized by tuning hyperparameters and the model architecture, such as adding more layers.
4. Once trained and tuned, the final result will be a model than when fed an image it will output the sequence numbers, in order, that are contained within it.

Metrics

Since this is a classification problem (i.e. the model has to properly classify digits in a natural image), accuracy will be used to measure performance. Accuracy is simply the number of correctly classified images divided by the total number of images that were classified. This will be express as a percentage. For example, if the model classified 100 images and got 97 of them correct, it would have achieved 97% accuracy. Further, the entire digit sequence must be correct in order for the image to be considered correctly classified. Therefore, if even one digit of the entire sequence is misclassified the entire image will be considered to be incorrect.

Data Exploration

The SVHN (street view house numbers) dataset was developed for object recognition algorithms by Stanford University (3). It contains over 248,823 labeled, real-world images. This dataset was originally obtained from house numbers in Google Street View Images. The complete dataset obtained from Standford includes cropped images, and a .mat file that contains bounding boxes and label information.

The dataset is split into 3 groups, training, test and extra. Each group contains 33,402, 13,068, and 202,355 images respectively. These images have challenges that are character of natural images, such as blur, distortion and illumination effects.

The .mat file contains information about each image including a label for each of the digits contained in the image. There are 10 classes where digit 1 is labeled as '1', 9 as '9' and 0 as '10'. The mat file also contains information for bounding boxes, which are coordinates for each image that locate each digit within that image.

Exploratory Visualization

Image 1 contains diverse samples of the full color images from the dataset. Some of the different characteristics we see of the images are: digits reflected the sun (as seen in the '49'), some are badly blurred (23?), others blend with their background (78), as well as offset digits that are not horizontal, but diagonal or vertical sequences.

While the images contained in Figure 1 have been cropped to show a diverse representation of the dataset, the sizes of the original images vary. This is important because we will have to resize the images to be the same for our datasets.

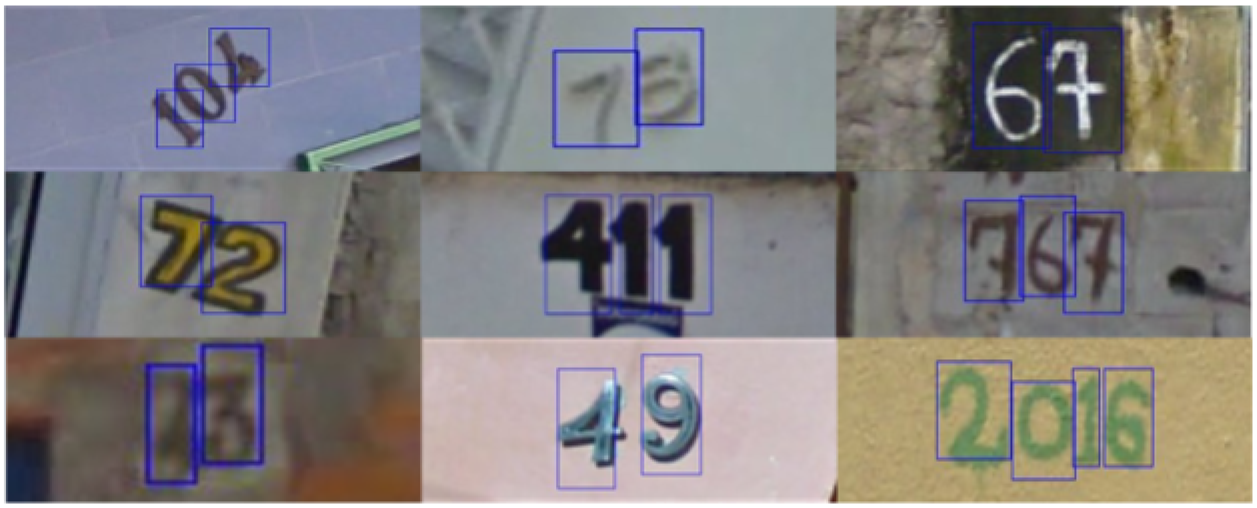


Figure 1: Example of SVHN images with bounding boxes (3).

Algorithms and Techniques

The images were classified using a Convolutional Neural Network (CNN). CNNs are deep learning models that require lots of data to train them. Moreover, CNNs are computationally efficient when it comes to image classification since they share common parameters such as weights.

The model will take an image and it's sequence of labels as input and output a score for each class, these scores are known as logits (classes are 0 through 9 and 11 for blanks). The logits will then be used as input into a softmax function which converts the logits to probabilities. Next, the algorithm will use cross entropy to measure the loss from the classifier output. Last, using an optimizer, in this case gradient descent, the model will minimize the loss to choose the best weights and biases and achieve the highest accuracy when classifying each image.

The model can also be tuned to achieve a higher accuracy. This can be done by adding a learning rate, regularization, dropout, or by adjusting the depth of the convolutional layers and number of nodes in the hidden layers. Regularization is a technique that prevents overfitting by penalizing large weights and ultimately restricts the number of free parameters. Dropout works by randomly setting half of the activations to 0. This forces redundant representations in other activations and helps prevent overfitting.

Benchmark

About 96% accuracy is the benchmark to aim for. An accuracy of over 96% was achieved by Google researchers in a recent paper (4). However, since human accuracy is approximately 98%, I would be interested to see how close I come to this threshold.

Data Preprocessing

First, the images have to be cropped to 30% larger than the sum of all their bounding boxes. This ensures no part of the digits with the image are missing and removes unnecessary blank space. This saves processing power and time by not training on a misguided feature, such as blank space.

Next, is to convert the images to an array of numbers 0-255 (where 0 is black and 255 is white). For the mean time we are going to retain the three color values (Red, Green, and Blue - RGB).

It is important to resize all images so size is not a mistaken as a feature. Otherwise the algorithm may mistakenly correlate size to the classification of the digits. Here all the images are resized to 64 x 64 pixels. Further, converting the images to grayscale also removes what could be mistaken as a feature, color. Color is not important in classifying our images, by removing it we increase performance by reducing dimensionality.

Last, is to normalize the images to have mean of zero and standard deviation of 0.5. This makes training easier because the optimizer has to do less searching to solve the problem (8).

The datasets will then be pickled into subsets of the images to make them manageable for systems with various memory capacity. There will be five datasets in total. The training set will have 385,150 images (split into 3 sub datasets), the test set will have 6,000 images, and the validation set will contain 6,000 images.

Implementation

A convolutional deep neural network (CNN) was used to classify the images in the SVHN dataset. The CNN was constructed with three convolutional layers and one fully connected layer for each of the 5 possible digits in the image, for a total of 5 fully connected layers. The architecture chosen for this model was a Multi-Task model (5) which is ideal for datasets with multiple labels for each input (see figure 2). A multi-task model has both shared and independent layers (referred to as Tasks in Figure 2). The CNN model used had three shared convolutional layers to learn the features of the digits and five independent layers (task layers). These task layers were used to output five independent labels for each image, each representing a digit in the address sequence.

All convolutional layers used a patch size of 5x5 and max pooling of size 2. They followed the pattern of convolutional, relu, max pool. Other original parameters for the layers were:

- Number of iterations (training steps): 20,0001
- Batch size: 64
- Conv layer 1 depth: 32
- Conv layer 2 depth: 64
- Conv layer 3 depth: 128
- Nodes in all hidden layers (fully connected): 512

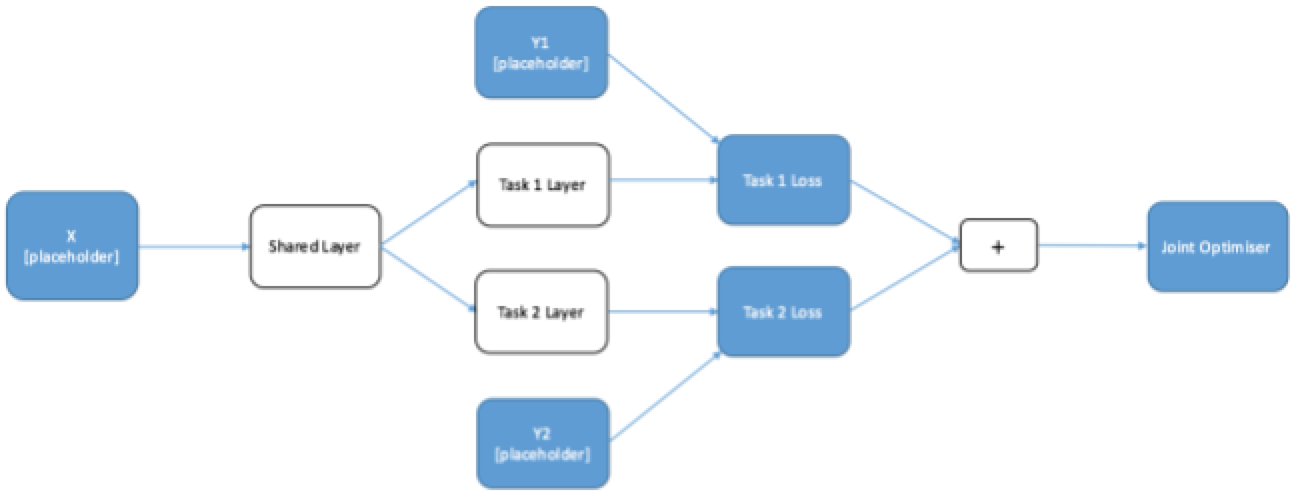


Figure 2: Multi-Task model (5)

As shown in Figure 2, each task’s loss was independently calculated before all losses were summed and used as input to the optimizer. Each loss function took as input the output (logits) from the corresponding task layer and the label to be predicted. For example, the first task loss would take as input: the logits from task layer 1 and the first digit in the address sequence. The second task loss would take as input: the logits from task layer 2 and the second digit in the address sequence, as so on. The model was originally optimized using a gradient descent optimizer to minimize the loss function.

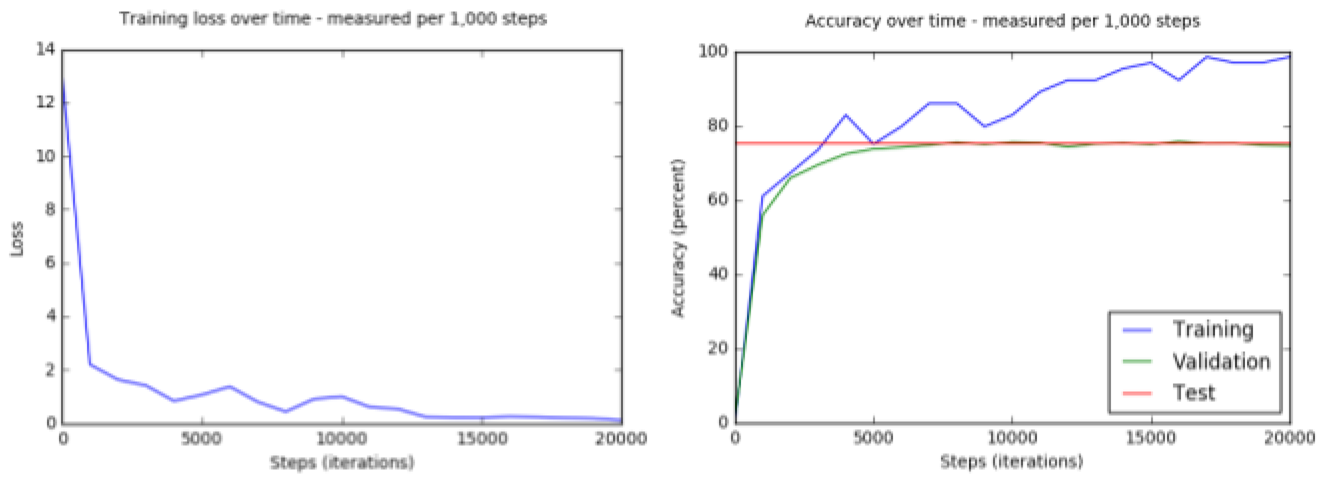
Refinement

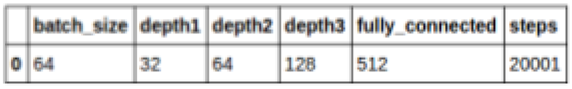
The original model and parameters above produced a final test accuracy of about 75% and validation accuracy of about 75%. Compared to the targeted benchmark of 96% this was relatively poor. However, there are myriad ways to improve models with hyperparameters and layer additions.

The following methods were used to improve the performance of the model:

1. Another convolutional layer was added for better feature extraction
2. The fully connected layer nodes was increased 400%
3. A learning rate was added and adjusted
4. Dropout was added and adjusted - dropout helps prevent overfitting by setting activations to zero and stopping the network from relying on any given activation.
5. Number of iterations/steps increased

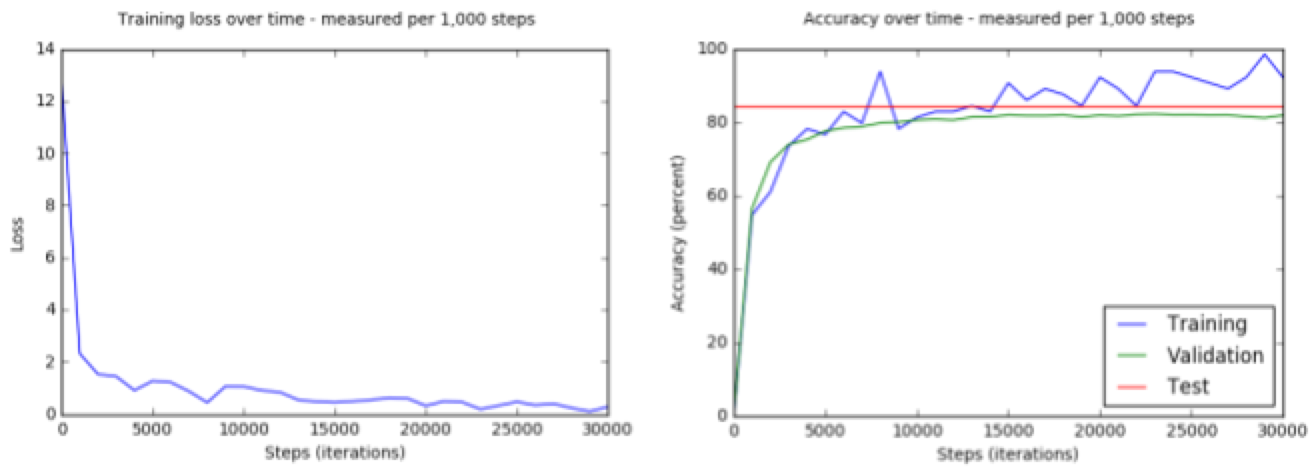
After these adjustments an accuracy of 85% was achieved (see figure 4). While this is not the 96% that was hoped for it is still a significant increase over the initial model. See the improvements section for more details on how a higher accuracy, such as the 96% could be achieved. The model did not suffer from overfitting as much as seen by the decrease in distance from the training and validation accuracy.

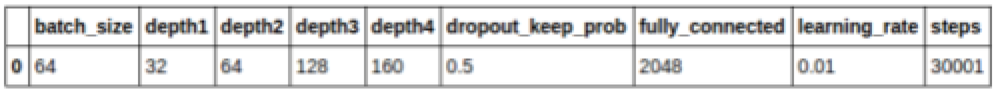




	batch_size	depth1	depth2	depth3	fully_connected	steps
0	64	32	64	128	512	20001

Figure 3: The original model and parameters





	batch_size	depth1	depth2	depth3	depth4	dropout_keep_prob	fully_connected	learning_rate	steps
0	64	32	64	128	160	0.5	2048	0.01	30001

Figure 4: After tuning and architecture adjustments

Model Evaluation and Validation

The final model had an architecture that looked liked the following:

1. The convolutional layers with 32, 64, 128 and 160 feature maps
2. Convolutions were 5x5
3. Max pooling with a window of 2 and a stride of 2.
4. All activation functions were relus
5. The convolutional layers were all shared until the fully connected layers, which were independent of each other. There were 5 fully connected layers with 2,048 nodes.
6. The weights and biases were initialized with xavier_initializer. This initializer is designed to keep the scale of the gradients roughly the same in all layers.
7. The training took place in 30,001 iterations.
8. Dropout kept only 50% of the activations
9. The learning rate was 0.01
10. Batch size was 64

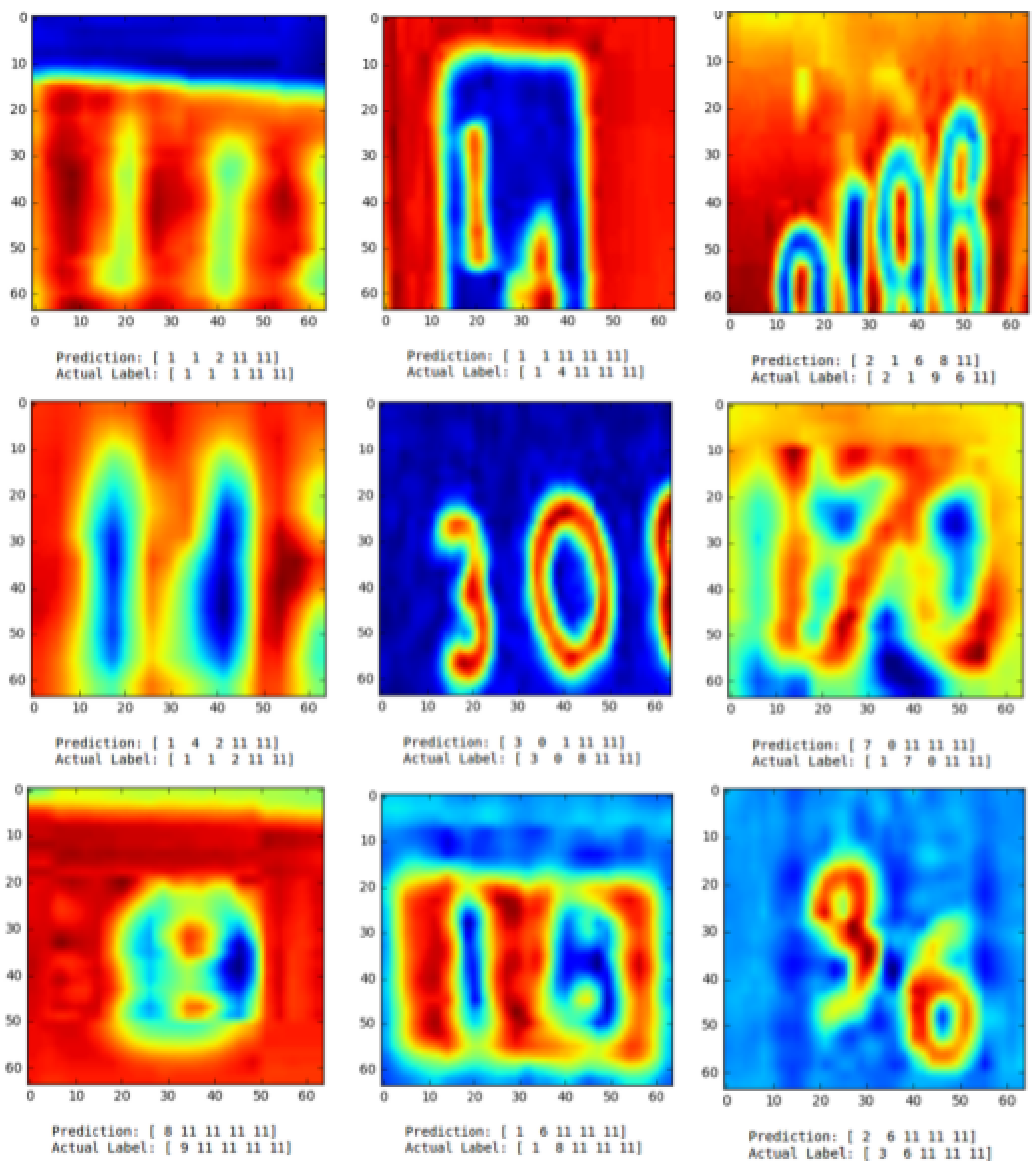
The original dataset came in three sets: test, train, and extra. To ensure proper evaluation and validation of the model the original three sets were split into three new sets: test, train, and validation. The test dataset remain untouched, as these were explained to be the most difficult images and therefore the best to test the final model against. A validation set was created to test the model during creation and evaluation.

The robustness of the final model was verified with unused images from the original datasets. The validation and training sets were created by combining all of the train and extra images. Next, they were randomly shuffled and then divided into the final validation and training sets. Since the test, train, and validation datasets were too large to use entirely for training, every time the preprocess script is run, new datasets are formed. Creating these datasets multiple times and running them through the model and achieving accuracy results within ∓ 2% shows the robustness of the model to generalize and shows it can be trusted.

Justification

With a highest achieved accuracy of 85% the benchmark of 96% was not beat. For a model like this to be useful in providing intelligence to humans, especially for those with a vision impairment, results from this type of model need to be at least on the level of humans. That level is currently 98% for a dataset like the one the model was trained on. More techniques could be done to achieve the desired benchmark, see these in the Improvements section. Therefore, these results are not significant enough to declare the stated problem, of digit recognition in natural images, as solved.

Free-Form Visualization



```
Prediction: [ 1  1  2 11 11]
Actual Label: [ 1  1  1 11 11]

Prediction: [ 1  1 11 11 11]
Actual Label: [ 1  4 11 11 11]

Prediction: [ 2  1  6  8 11]
Actual Label: [ 2  1  9  6 11]

Prediction: [ 1  4  2 11 11]
Actual Label: [ 1  1  2 11 11]

Prediction: [ 3  0  1 11 11]
Actual Label: [ 3  0  8 11 11]

Prediction: [ 7  0 11 11 11]
Actual Label: [ 1  7  0 11 11]

Prediction: [ 8 11 11 11 11]
Actual Label: [ 9 11 11 11 11]

Prediction: [ 1  6 11 11 11]
Actual Label: [ 1  8 11 11 11]

Prediction: [ 2  6 11 11 11]
Actual Label: [ 3  6 11 11 11]
```

Figure 5: The misfits - random samples of misclassified images

The images in figure 5 were randomly drawn from a list of misclassified images. As stated above, all digits in the sequence must be correct in order for the image to be considered correctly classified. What is very interesting about figure 5 is that we start to notice two clear patterns in the misclassified images. One, is that some images are cropped in a way that cuts off part of the digit sequence. Two, some images have prediction labels that closely resemble other digits that are visually similar to the actual label. For example, the image to the furthest left on the bottom row is actually a 9 but even looks like 8 to me. However, while some of these digits may be visually similar the model should be able to tell the difference on some of the more obvious ones, such as the one in the bottom row to the far right.

Reflection

Small bugs in my program caused some big problems for me. For example, in my loss functions I had commented out a section of each loss function to exclude specific things while testing. I thought the code was ok because Python never threw any errors saying it was missing syntax, however it was not. This led to a NaN loss throughout all training iterations. Many hours were spent researching NaN loss and what I found was lots of information about weight initialization, learning rates, regularization, layer size, number of layers and optimization. Once it seemed I had everything correct I started to look very hard for bugs and found it.

Another bug that led to big distractions was my accuracy measurement. It would read the training accuracy correctly but the validation accuracy was very low from start to finish while training my model. This led to a lot of searches to optimize my model for better validation accuracy. This generally pointed in the direction of the enhancements listed above, which led to deeper dives into some of the material.

While I made lots of changes I do not know the results of them all because I changed them before my model outputted a reliable performance metric. Due to time constraints I could not go back and reproduce all changes to see the differences they made.

Building the model and finding the small but critical bugs in my code took a very long time (about 8 weeks - preprocessing the .mat file was a slight challenge too). I did not find how my model was performing until very late in the process. Seeing that my model only performs with about 85% accuracy spending more time doing the things in the Improvements section below would have been very beneficial, especially in dealing with the digits that were cropped out of the images.

A lot of my model's initial architecture was based on the paper by Goodfellow et al. They were able to achieve 96% accuracy with their model. Using confidence thresholding they were able to achieve 95.64% coverage at 98% accuracy. The 98% accuracy is very important because it is on par with human accuracy of the same problem. I really would have liked to achieve an accuracy closer to this number.

In summary, with an accuracy of 85% the final model should not be used in any challenging environments. However, while not tested, I am suspicious that it would be able to predict easier sequences of digits such as those with images that do not have blurred numbers, non-horizontal sequences, or other hard-to-read data.

Improvement

There are several improvements that could be recommended for future improvement. For example, the preprocessing step could be revised to keep more of the original images, more data would help the model perform better. It is undetermined why tens of thousands of images were skipped in the preprocessing stage, this could be further explored to obtain the extra images.

In the paper by Goodfellow et al, they cropped a 54 x 54 pixel image from a random location within the original 64 x 64 image. This generated many randomly shifted versions of each training sample and increased the size of the dataset.

Lastly, during random checks of the images throughout preprocessing and building the model, it was discovered that a myriad of images contained digit sequences that were cut off (possibly an improper crop). A deeper dive into why this occurred (perhaps from a bug that did not read the bounding boxes correctly or bad images in the dataset) this could create higher quality datasets.

References

1. https://nei.nih.gov/eyedata/lowvision/tables#5
2. http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf
3. http://ufldl.stanford.edu/housenumbers/
4. https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/42241.pdf
5. https://jg8610.github.io/Multi-Task/
6. Deep learning general architecture and tuning
    a. https://www.udacity.com/course/deep-learning--ud730
    b. http://cs231n.github.io/convolutional-networks/