Deep Learning Capstone Project

Abhinav Goyal

Table of Contents

Overview and Background

The problem chosen for the Deep Learning Capstone project is that of developing a deep learning model that can recognize digits from a live camera feed. Such a model can be used for novel apps in mobile phones, scanners, fax machines, et al (but this part of the project wasn't addressed in this implementation).

The problem of converting number strings to ASCII (or other formats that computers use) can be thought as a classification problem in a (supervised) machine learning scenario wherein shapes (of certain types) need to be categorized as Digits (0-9). Thus, algorithms such as Logistic Regression, Support Vector Machines, et al can be used for this purpose. However, the technique/method with the best performance and track record in this domain is definitely that of Neural Networks.

Neural networks have been successfully used to recognise digits for a while now. Already, back in the 1990s, Convolutional Neural Networks were used to classify digits in the MNIST dataset and the technology was commercialised for use in banks, postal services, et al. However, natural images pose a bigger challenge vis-a-vis the recognition of numbers that are not present in the MNIST case. These challenges can be classified into two broad categories

- differences in type/quality of data
- problems related to recognizing sequences rather than individual digits

A model trained on MNIST is not likely to do well in a real world scenario and thus we need data more similar to what our live camera app would encounter. SVHN is one such dataset. It contains images taken by Google's Street View program and, as these were taken by cameras, they contain all the ambiguity that our live app is likely to encounter. Much in the way that MNIST became a standard for recognizing one digit, SVHN can be thought of as a standard dataset for recognizing sequences in natural images. Benchmarks of machine learning algorithms on this dataset have now started reaching human level efficiency and the best performance known has been achieved using neural networks and Deep Learning. By measuring our model against this dataset, we can track how well our app is likely to do against the competition.

The SVHN dataset contains two types of sub-datasets/challenges

- dataset containing labelled cropped images with one digit in each of them
- dataset containing complete images with accompanying labels (digit information) and

bounding box information detailing location of digits on these images

From the perspective of our live camera app, these options would correspond to the following use cases

- the user zooms in one digit at a time and the app decodes it
- the user marks the digits using a box and then the app decodes them

The first approach is rather user-unfriendly- the user shouldn't have to point to each digit one by one- and I decided to go with the second approach. However, the first dataset was still quite useful in understanding the dataset and in developing techniques that are required to solve the more general case. Thus, it is a meaningful challenge to solve.

With this in mind, this project used both these datasets in the following steps.

- Preprocessed single images dataset
- Built a model for the single images
- Worked on this until the performance reached the desired goal
- Preprocessed and analysed the complete images dataset
- Adapted the single images model to work with the complete images dataset
- Tuned this model until the performance reached the desired goal

The goal was set to reach **90% (F1 score of 0.9)** for the dataset (and also for the blanks in the complete case). In addition, the goal was to reach **85% accuracy in decoding the entire string correctly**. These measures are defined in the Problem Analysis section below. In our case though, the impetus is on learning from previous attempts at solving the problem. The computational resources used in these ground-breaking attempts were out of the budgetary scope and hence breaking (or even replicating) the performance of the best attempts was not a goal of this project.

A description of the machine on which the project was carried out is presented in the results section.

Problem Analysis

As previously mentioned, the incremental challenges in decoding natural images (vis-a-vis a rather clean dataset such as MNIST) are twofold

- differences in type/quality of data
- problems related to recognizing sequences rather than individual digits

Some of these problems related to the differences in the type/quality of data are

- localizing/determining size of the digits in the image: the size and placement of the digits in the images vary with the sizes of the pictured digits and also the distance from which the images were clicked. Thus, detection techniques need to be able to handle this while localizing digits in the image
- image rotation: numbers needn't always be completely straight in natural images as photographer's hand might shake at the point of taking a picture or holding a camera
- image quality: natural images are often blurred and/or grainy and this needs to be addressed

Some of the problems related to the recognition of a sequence are

- handling distances between numbers: the algorithm has to account for different spacing between numbers in different images
- determining the the length of sequence: the length of the sequence is not known a priori in natural images and thus the software needs to cater to this
- the direction of the sequence: while numbers are usually written left to right, sometimes they are written from top to bottom or in other stylised ways and this needs to be accounted for as well in determining what constitutes a valid sequence

This leads to certain design decisions for an app and, thus, for the model that will be used within this app.

- Should the app place limits on the quality/size of the live images that can be used to decipher information?
- Should the app place limits on the length of the digit strings?
- Should the app look for user guidance in terms of placement of digits within the image frame?
- Should the app rely on the user to mark the digits that need to be decoded?

In the view of user experience and usability, it is best that

- the app do its best even in case where image resolution is poor
- the app try to decode strings of variable length
- the app should try to minimize user input

However, some of these require a lot of computational resources and/or memory which may not always be present on a mobile device. Keeping this in mind, the model developed

- works with images of any quality
- is limited to strings that are at most 5 characters long
- requires the user to zoom in on to the area containing the numbers

Metrics

The app performance was judged using the following metrics

- F1 scores: 2 x precision x recall / (precision + recall). In the multiclass case, we can look at the F1 score for each class and also the averages of the individual scores. The average
  - weighted average
  - detailed

Accuracy

Accuracy is defined as the ratio of correct predictions to the number of total predictions. We will look at 2 kinds of accuracy measures

- accuracy in decoding individual digits: 100 x number of digits predicted correctly / total number of digits
- accuracy in getting a string entirely right: 100 x number of images correctly decoded / (number of test images)

We use accuracy as a measure so as to be able to compare ourselves with results published on this dataset.

F1 score

F1 score is defined as 2 x precision x recall / (precision + recall)

where

- precision = true positives / true positives + false positives
- recall = true positives / true positives + false negatives

True positives for a class are the elements belonging to that class that are predicted correctly. False positives for a class are elements belonging to other classes that are predicted to belong to this class.

Thus, precision can be taken as the probability that a predicted member of a class actually belongs to that class.

False negatives are elements of a class that are predicted to belong to other classes. Thus, recall can be taken as the probability that a member of a class will be predicted correctly.

F1 is a much more robust indicator than accuracy. To understand why, take for example a scenario where an algorithm has to predict between classes A, B and C. The ratio of A and B in the mix is 0.7, 0.2 and 0.1 respectively. Thus probability(A) = 0.7, probability(B) = 0.2, probability(C) =0.1.

Suppose the algorithm simply guesses A all the time, its accuracy would still be deemed 70% as it would be correct 70% of the time. The F1 score would, however be calculated as

Class	True Positives	True Negatives	False Positives	False Negative	Precision	Recall	F1
A	0.7	0.0	0.3	0.0	0.7	1.0	.824
B	0.0	0.8	0.0	0.2	0.0	0.0	0.0
C	0.0	0.9	0.0	0.1	0.0	0.0	0.0

The weighted F1 score for this case is F1_A x p(A) + F1_B * p(B) + F1_C*p(C) = .576. This is a much better indicator of the algorithm's performance!

As previously mentioned, the goal of the project is to reach **90% (F1 score of 0.9)** for the dataset and **85% accuracy in decoding the entire string correctly**.

Building the model

Data Exploration

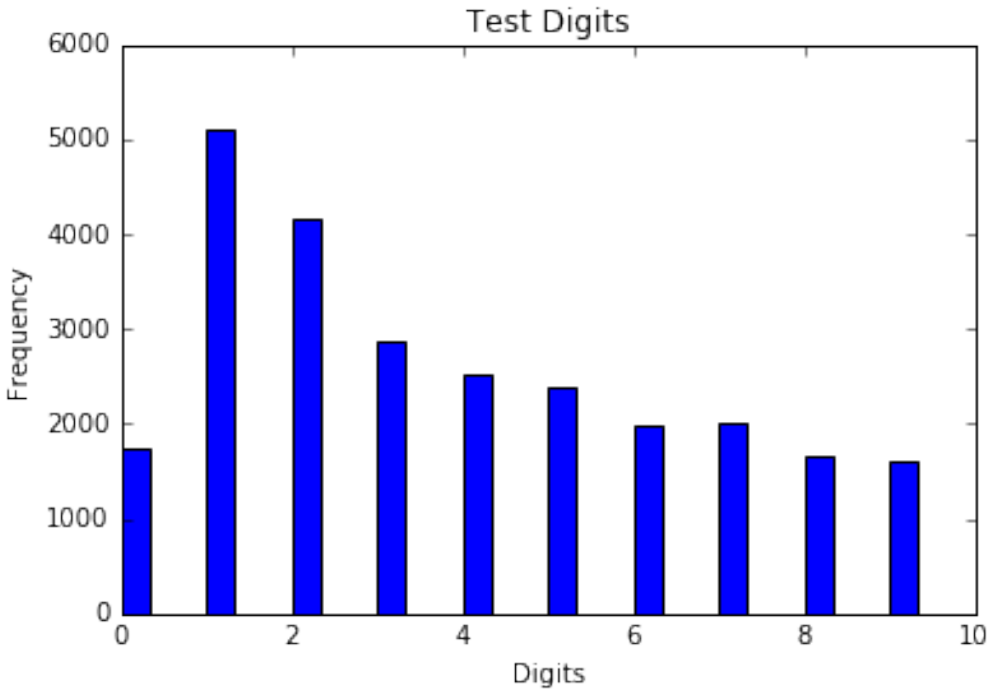
Single/Cropped Images Dataset

The single images dataset contains three .mat (matlab compatible files).

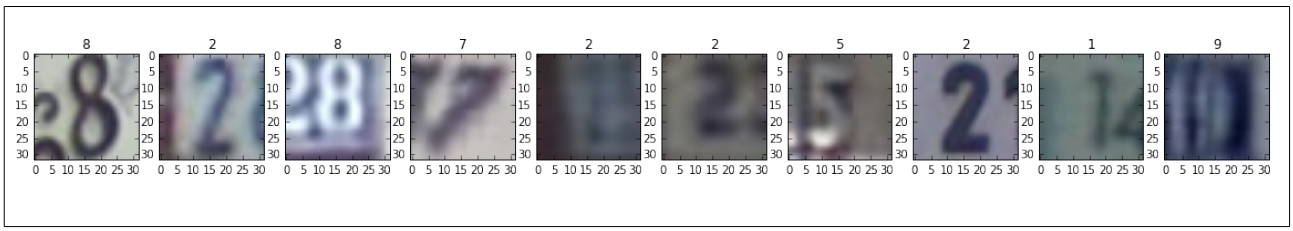
- train_32x32.mat: 73257 images to train on
- test_32x32.mat: 26032 images to test on
- extra_32x32.mat: 531131 extra images to train/validate the model on

All these are square images of size 32x32.

The frequency distribution of the various digits in the test set is shown below. Given Benford's law, it wasn't surprising that 1 was the common digit in the datasets.



Some random images from the train set are displayed below



Complete images with bounding boxes

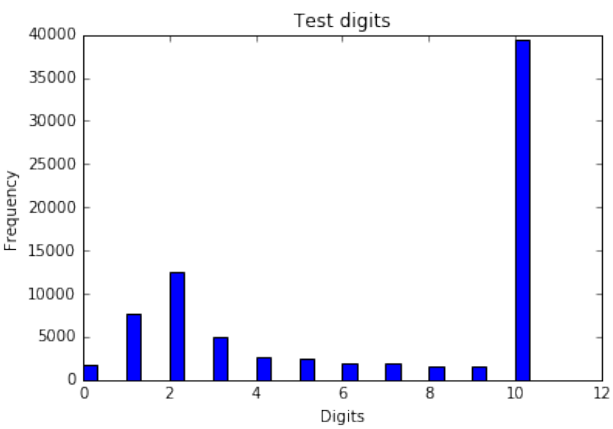
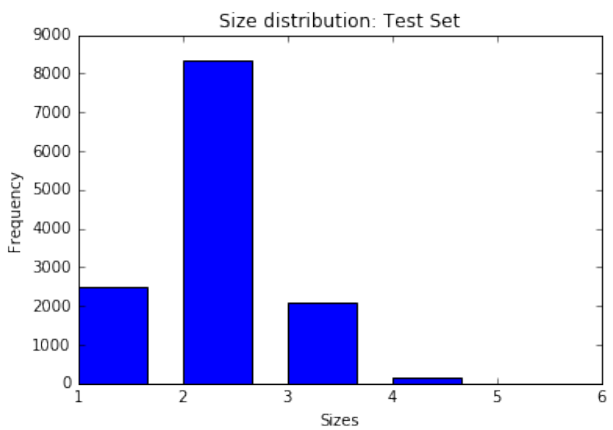
The complete image dataset comes in three files in a tar-gzipped format

- train.tar.gz
- test.tar.gz
- extra.tar.gz

These archives, when extracted, contain the folders containing files and metadata for the respective sets. The sets are as follows

- train set: 33401 images
- test set: 13068 images
- extra set: 202353 images

The distribution of digits in the test set and the various lengths are shown below

Digit Distribution	Size distribution
	

There are some interesting things to note about the test set

- The distribution of digits doesn't follow Benford's law
- There are only 2 images with 5 digits in them!

Model training

As mentioned before, Neural Networks were used for this purpose. A general (rather high level) discussion of Neural Networks is provided here.

Neural Networks

The basic rationale behind the use of Neural Networks is that of function approximation. The output y can be considered a function F(X), where X is the input. In this scenario, the function F can be approximated as a series of smaller functions, f, g, h, et al such that

$$F(x) = h(g(f(X)))$$

In the Neural Network scenario, these functions are approximated using different layers of the network. Each layer of the network comprises various cells or neurons. These cells are fed by the outputs of the previous layer and, in turn, their output is fed to the next layer. A neuron contains adaptive weights, a bias value and an activation function that determine the output of the neuron based on the inputs. The input layer feeds data into the network and calculations are run until the output layer contains the predicted value. This phase is called the *feedforward phase*.

Based on this error, the weights in the network are adjusted on the backward pass or *backpropagation stage*. This is done on the following basis

- at the output layer, we know the prediction as well as the actual output
- thus, we can figure out the loss as the square of the difference between output and prediction
- Based on this, we can update the weights of the output neurons (neurons in the output later)
- We can now calculate the errors for the preceding layer based on the expected/ideal output of the preceding layer and the current output
- Knowing these errors, we can calculate the weight update for the preceding layer and apply the update
- We can proceed like this until we reach the input layer and update the weights for it

- At this point, we can't proceed further since there is no correction to be made to the inputs
- Thus, we move back to the feedforward phase and feed the next (batch of) training inputs

There are a couple of important things to note

- all of this is based on calculus and minimization of error by setting derivatives to 0, Thus, this works only if the network is differentiable end-to-end. Since the network is essentially an alternating series of matrix multiplications and activation functions, the matrix multiplication bits are differentiable. The activation functions also need to be differentiable
- if the activation functions are linear, then the whole network can be replaced by one matrix from an input space to an output space. Thus, in order to approximate non-linear functions, non-linearities are introduced via the activation functions

Given a wide enough, or deep enough, network, any function can be approximated – at least theoretically. In practice, some problems remained such as the vanishing gradient or the exploding gradient which have been addressed in the last few years with novel initialization mechanisms, activation functions, et al. Now, networks as deep as 28 layers have been used in published research and Deep Learning based methods have been used successfully for a wide variety of tasks.

In addition, there are some more important aspects of neural networks that are of considerable importance to this project

- Convolutional Layers:. Convolution operations enable running a convolutional kernel over the input data to generate features such as edges and more complex shapes that are highly important in image classification tasks. Thus, convolution was used in order to generate feature maps that aid the network learn better representations of the digits.
- Pooling Layers: Pooling, as the name suggests, allows “pooling” of several signals together to generate a downsampled output. Pooling is of various kinds. Max pooling, as the name implies, takes the maximum of these signals while average pooling takes the average of the inputs. We use max pooling to boost the shapes
- Rectified Linear Unit (ReLU): The rectified linear unit is defined as reLU(x) = max(0, x). In the past few years, they have gained a lot of traction in the field of neural networks. They are especially important when it comes to continuous domain but have also gained traction in the field of image classification. Although exact answers as to why they work better than the sigmoid or hyperbolic tangent functions are not known, they have empirically proved to work better.
- Dropout: Dropout is a mechanism for regularization of neural networks. Keeping the weights of the neural networks low is quite important as it prevents overfitting. Dropout simply means setting some of the variables to 0 (during the training phase). Regularized

networks are less prone to overfitting.

Data Preparation

From the very outset, it was clear from these images that

- colour isn't important in determining the digits and we can safely disregard this information
- that the digit colour may be lighter or darker compared to the background and, thus, our model has to be invariant to this

For this, it was decided to

- convert the image to greyscale
- let the Conv Net handle the background/foreground invariance

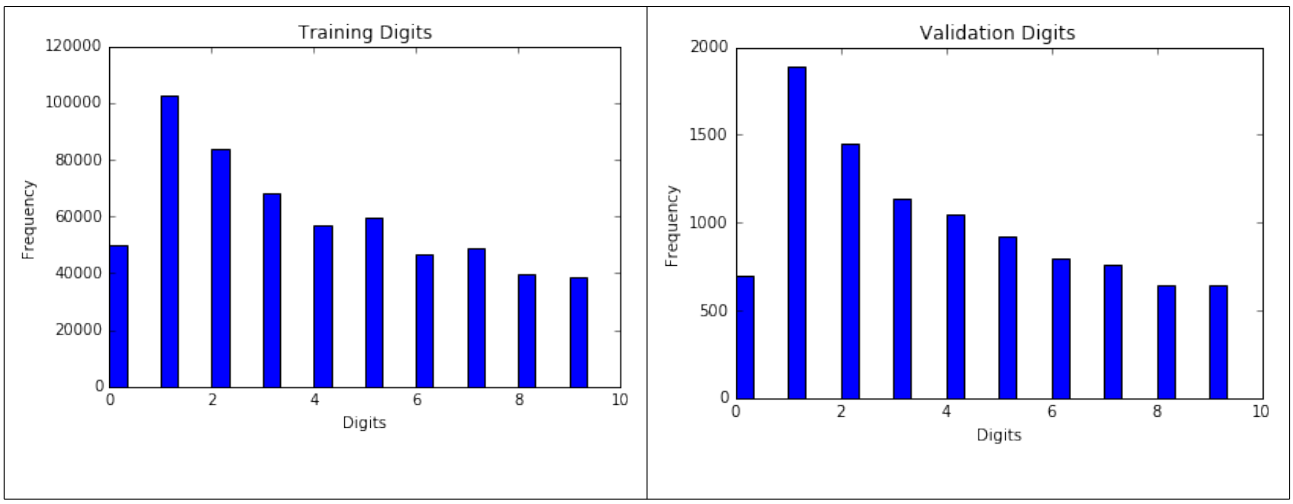
Cropped Images

In order to use data maximally, I

- split the train set into a 90-10 split to generate two sets of sizes 65931 and 7326 respectively
- split the extra set into a 99.5-0.5 split to generate two sets of sizes 528475 and 2656 respectively
- merged the first of the two sets from the splits to generate a training set of size 594406
- merged the second of the two sets from the splits to generate a validation set of size 9982
- randomized the training and validation sets

The distribution of digits in the test and the validation sets is the table below

Distribution of Digits in Training Set	Distribution of Digits in Validation Set



As can be seen, the overall ratios are quite similar (though not exactly same) and that these ratios match the distribution in the test set histogram presented earlier.

The test set was left as is.

Complete images with bounding boxes

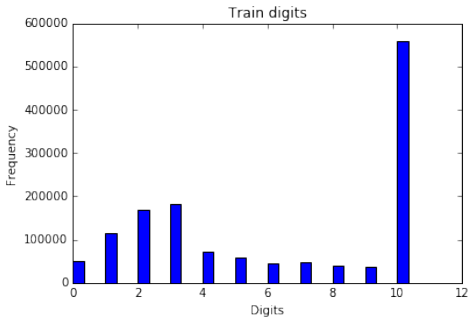
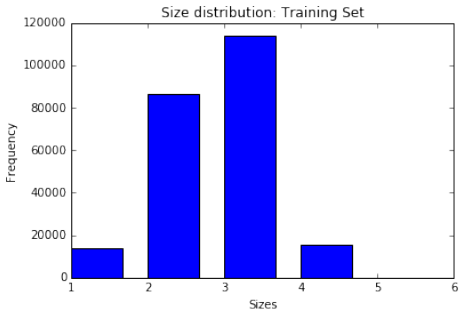
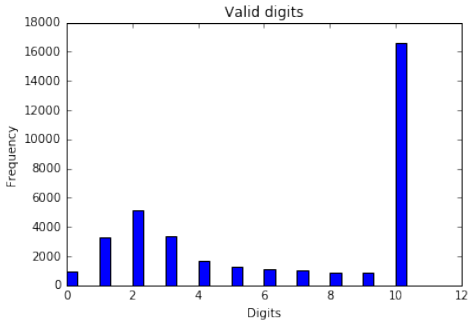
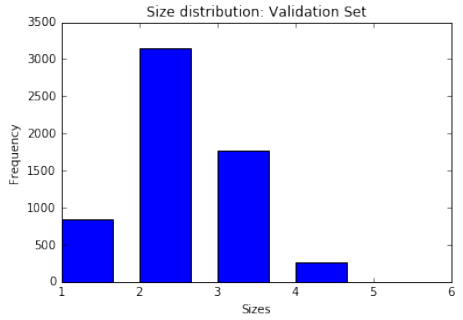
There were a few decision decisions to be taken with regards to this model. One could go with an LSTM approach or a Convolutional Network approach. In the end, I decided to go with the approach outlined in this paper by Goodfellow, et al. This helped to compare our performance against the current benchmarks to help determine future improvements I need to make (and skills I need to learn in this area).

The metadata information contains the bounding box information for the digits on these images. Using this information, the images are cropped so that all the digits were present within the cropped image and this cropped image was reduced to a 32x32 size. In addition, these images were converted to greyscale.

Similar to how the cropped images dataset were used to generate train, validate and test sets, the train and extra sets from this dataset were also split and recombined to create the following train and validate sets

- train set: 229731 images (28390 from original train, 201341 from extra)
- validation set: 6023 images (5011 from original train, 1012 from extra)

These sets were then randomized.

	Digits	Labels
Training Set		
Validation Set		

As can be seen, there are very few examples of a higher number of digits in these sets and this poses a serial challenge when it comes to decoding these digits.

Model training

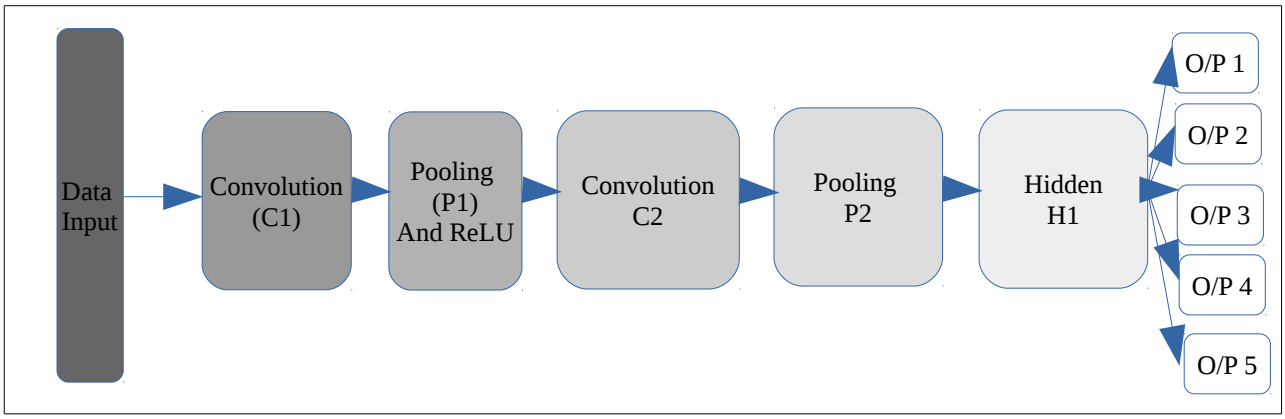
Models were built using Google's tensorflow library. A very useful resource in learning more about this toolkit (and Deep Learning) is [this course from Udacity](#).

The model for the single digits which worked quite well had the following architecture

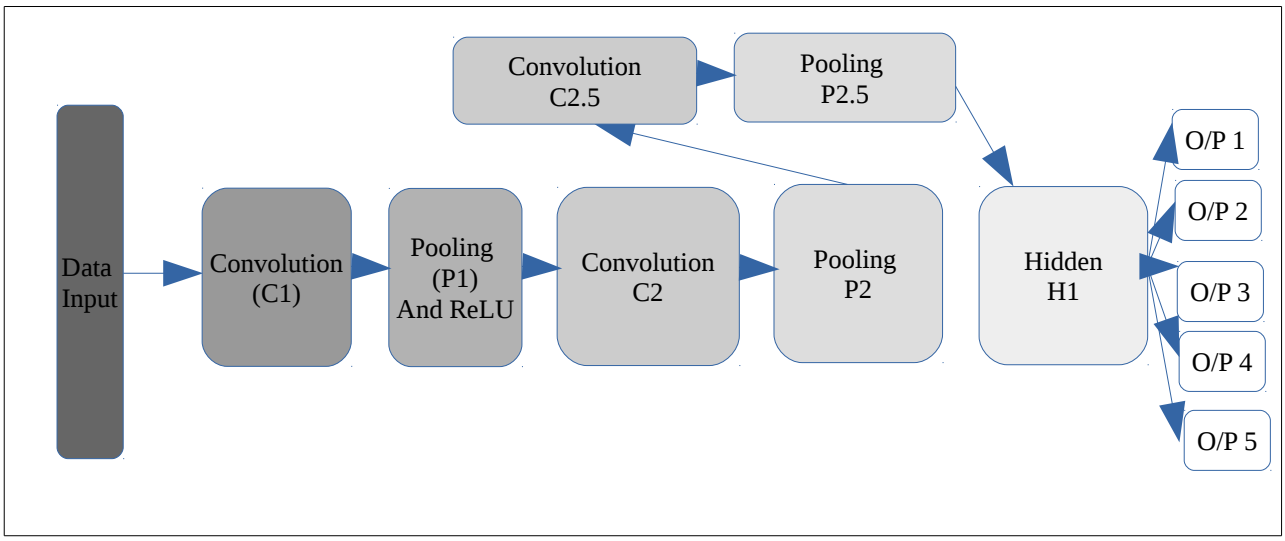
- Convolution Layer 1 with a patch size of 5x5 and a depth of 16: This alters the images to dimensions of 28x28x16
- Pooling Layer 1 with a patch size of 2x 2 and a stride of 2 which alters the images to dimensions of 14x14x16
- Activation Function (ReLU)
- Convolution Layer 2 with a patch size of 5x5 and a target depth of 32. This alters the image to size 10x10x32

- Pooling Layer 2 with a patch size of 2x2 and a stride of 2 which alters the image to the size of 5x5x32
- Activation Function (ReLU)
- Hidden Layer of size 32x64 (using ReLU activation)
- Dropout, for regularization
- Output Layer of size 64 x 10 (softmax layer generated probabilities of the output belonging to various digit classes)

This was adapted for the multi digit case with the following change: instead of one output layer, 5 output layers were used (of size 64 x 11) to correspond to the 5 digits. The output space of 11 was augmented for the blank space which is present in the case of variable string lengths.



This was found to work quite well. However, this merely achieved an accuracy of 80.2% in getting the whole string right. Thus, an additional layer was added in order to boost performance. The new network has an additional Convolution, Pool, ReLU sequence



The architecture for this network was as follows

- Convolution Layer 1 with a patch size of 5x5 and a depth of 16: This alters the images to dimensions of 28x28x16
- Pooling Layer 1 with a patch size of 2x 2 and a stride of 2 which alters the images to dimensions of 14x14x16
- Activation Function (ReLU)
- Convolution Layer 2 with a patch size of 5x5 and a target depth of 32. This alters the image to size 10x10x32
- Pooling Layer 2 with a patch size of 2x2 and a stride of 2 which alters the image to the size of 5x5x32
- Activation Function (ReLU)
- Convolution Layer 2 with a patch size of 3x3 and a target depth of 64. This alters the image to size 3x3x64
- Pooling Layer 2 with a patch size of 3x3 and a stride of 3 which alters the image to the size of 1x1x64
- Activation Function (ReLU)
- Hidden Layer of size 64x64 (using ReLU activation)
- Output Layer of size 64 x 10 (softmax layer generated probabilities of the output belonging to various digit classes)

This increased the performance further, getting us closer to the goal. At this stage an additional hidden layer was added to see if this would get us over the line but this actually reduced the performance by 1%.

Results

All these tasks were carried out on a machine with the following specifications

- CPU: Intel i7-4720HQ CPU
- RAM: 32 GB
- Sufficient Secondary Storage
- OS: Ubuntu 16.04
- GPU: Nvidia GeForce GTX 980M

As previously described, several variations were tried with respect to the number of convolutional layers and the depths of these layers. These are present in the .ipynb files submitted along with this report. A discussion on these is present in the Conclusions section below.

The best accuracy for the Single/Cropped Images dataset was **94.0%.**

The F1-scores for each digit were as follows

Digit	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
F1	0.934	0.954	0.958	0.922	0.950	0.936	0.926	0.941	0.912	0.917

Leading to an overall unweighted F1-score of **0.936** and a weighted F1-score of **0.940**

This model was then adapted to create a model that could decode all digits at once. The best performance on the complete image dataset was **94.7**% accuracy in identification. However, it is for only **80.2%** of the images that all the digits in that image were correctly recognized.

The addition of the additional Convolutional Layer led to the accuracy improving to **82.1%** though the additional of the second hidden layer dropped this back down to **81.1%**. However, the per class F1 score went to almost 0.85 for all classes.

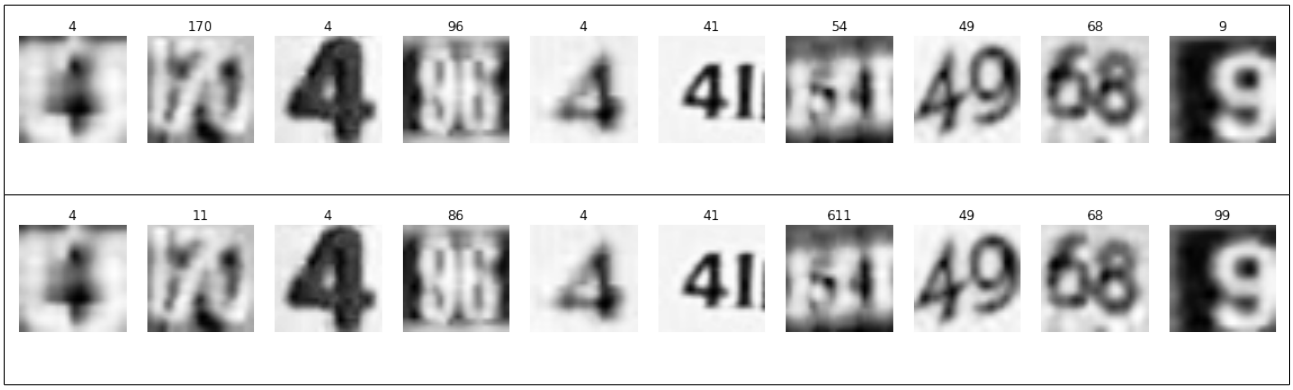
The weighted F1 score over the whole set was 0.95 while the unweighted score was 0.89. Even after removing the blank label, the weighted average was .889 and the unweighted average was .881.

The F1-scores for each digit were as follows

Digit	0	1	2	3	4	5	6	7	8	9	Blank (10)
F1	0.884	0.912	0.911	0.865	0.912	0.886	0.850	0.885	0.849	0.852	.991

The goal of reaching 0.9 F1 score on the entire set was reached but the accuracy of 0.85 on getting the entire string correct is below the desired goal. This will constitute further investigation as discussed in the Future Work section.

Shown below are some random samples from the test set for the complete images and the results for these images.



As can be seen, the model does a decent job in most cases but in some cases (for example, the 7th image above), it imagines it is seeing 611 instead of 54. The future work section below discusses thoughts on tackling this.

Conclusions

Neural Networks are very powerful tools and the ease with which models can be built will only lead to a wider adoption of these methods. The model in this project is as follows

- The data is first prepared from the .mat files
- It is preprocessed only minimally - the colour channels are removed in favour of a greyscale image and the arrays are transposed to be Python compliant (row major)
- It is fed to the a Convolutional Neural Net with the following architecture

For a non-trivial challenge such as reading characters from natural images, this network is able to achieve 89% accuracy straightaway. With some additional tweaks, this reached 94.8% accuracy. Although, these tweaks are non-trivial and, for some researchers, they nevertheless show the promise of this area. This will especially be true as GPU capacity increases and as it becomes easier to chain multiple GPUs together.

This is clearly exemplified by my own personal experience. When I started this project, I had to resort to using the CPU due to a library mismatch issue on my machine (cuDNN and TensorFlow). After the first submission, I took the time to upgrade the OS and libraries so that I could use the GPU for this. The difference in performance is startling. This productivity gain meant that I could spend more time on the validation routines and on improving the code.

The key findings/learnings from this project were

- performance increases as number of convolutional layers increase
- initialization plays a huge role in the training of the network. The xavier initialization for the convolutional layers boosted performance immensely.
- the choice of hyperparameters is important too. Setting a large learning rate can lead to exploding gradient problem which essentially kills the learning. Similarly, setting too small a learning rate leads to extremely slow performance gains.

On a personal note, I enjoyed learning the Pooling operation in greater detail and using TensorFlow for this project. Although, TensorFlow is considered a lower level library (than, say, Keras), getting to implement operations using TensorFlow is a good way to gain a better understanding of the way Neural Networks work. While this was quite challenging in the beginning and there was a

temptation to use other libraries, I am glad I stuck with process.

Future work

Obviously, there is considerable scope for improvement. There are two main areas in which these efforts will be focussed

- improving the digit recognition in the multiple digit case:
- learning long sequences of digits to progress towards a generic OCR tool. This involves the use of LSTMs for this goal.

These two aspects might be related. Thinking about humans decode this information leads to some interesting insights. The network tends to perform alright for digits that are clearly visible. Thus, tricks that humans use in difficult situations might apply to networks as well. Font, colour and spacing between digits all play a crucial role in determining the shapes when humans have to squint. All of this constitutes state information within an image and thus LSTMs should, thus, do a better job at identifying this information. In addition, this paper by Sermanet, et al, which was consulted in order to build a better validation set, discusses Lp pooling and is definitely worth experimenting with.

I am also keen on implementing a network that learns the number of digits/characters in an image. This has been previously explored by Goodfellow, et al. The interesting aspect of this is that the target variable would be continuous. I would then like to use this network to

- fix the prediction of the 5 digit strings in this dataset
- learn how the input image can be auto-segmented/chunked so as to read a fixed number of characters at a time.