Multi-digit recognition on StreetView images using Deep Convolutional Neural Networks

Aditya Jadhav aaj2146, Louis Massera lm3287, Sai Pavan Kumar Unnam su2216
Columbia University

Abstract

In this paper we address the problem of recognizing arbitrary multi-digit street numbers from unconstrained Google StreetView imagery. To this end, we design and train a Deep Convolutional Neural Network consisting of 10 hidden layers. To train and evaluate the model we use the publicly available SVHN dataset. To compare performance we use [2] as a benchmark. We achieved an accuracy of 96.4% and 93.5% on the test and independent validation datasets. The accuracy is defined as follows: An instance of a test image is considered to be a success only if all the characters in the image are correctly recognized. Also, along with test and validation accuracy values, we devised a measure of 'confidence' in the model's ability to beat a human at multidigit recognition using the concept of dataset 'coverage' that will be defined later in the paper. Our methodology was as follows: Compare accuracy for models of two different depths, define coverage and then outline our ways of increasing accuracy that was tending to saturate.

1. Introduction

Street View House Numbers (SVHN) [1] is an open source dataset made available for digit recognition machine learning models. It contains close to 200,000 cropped images of street numbers, sourced from Google's StreetView photographs amounting to a total of about 600,000 digits. The idea is to automate the process of transcribing address numbers on geo-tagged street view images as against to manually doing the same. Thus, an important milestone for the model is to achieve human level accuracy in multi digit recognition, which is believed to be 98% [2]. With this model, we were able to achieve 93.5% and 96.4% accuracy on test and validation sets respectively. Therefore, to devise a metric of comparison with a human operator, we defined a 'coverage' which was touched upon in the original work [2] but nothing concrete was outlined. Here in this project we have devised and implemented our own coverage metric based on a probabilistic score that first evaluates the quality of the prediction and then .

One of the main challenges in this project is that of handling the massive SVHN dataset. As mentioned before, the dataset comprises of 200,000 uniques images.

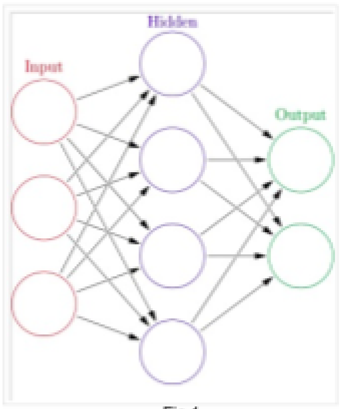


Fig 1
Typical Architecture of a CNN. Here we illustrate with a one hidden layer network

Training with a deep convolutional neural network on such a large set can be slow and tedious even with GPUs. Also, another challenge is to facilitate faster training over such a large dataset. Google trained their network for 6 days on state of the art resources that are available at their disposal. We overcame this challenge by making effective use of the 'extra' dataset given in [1] to train the model. Apart from that for saturations of the accuracy, we tried certain modifications to the learning rate while training. Next we shall explore the methodology of the original work followed by parallels to that with our implementation with an added focus on the coverage metric and comparison with training times

2. Summary of the Original Paper

2.1 Methodology of the Original Paper

In this work, a unified approach that integrates detection, segmentation and classification steps to recognizing multi digit address numbers is outlined. They have proposed an output layer which provides a conditional probabilistic sequence model for a string of digits. For a given input image X and predicted sequence S (S is a random vector of S_1, S_2,..., S_n where n is the sequence length and each S_i is a random variable that represents a digit) they aim learn a model of P(S|X) by maximizing ln(P(S|X)) on the training set. If L is the random variable that accounts for the uncertainty in the sequence length,

$$P(S=s|X)=P(L=n|X)\prod_{i=1}^{n}P(Si=si\mid X)$$

They have used a softmax classifier that receives input features extracted from X by a CNN. The best accuracy reported in the paper was obtained from a model that consisted of 8 convolutional hidden layers, 1 locally connected hidden layer and 2 fully connected hidden layers. Thus amounting to a total of 11 hidden layers. A visualization of the architecture is presented below

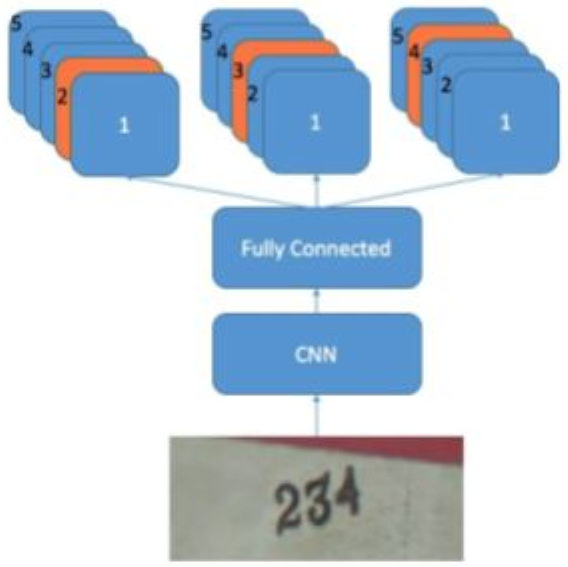


Fig 2.

Apart from this, the paper also delves into character recognition from the reCAPTCHA database. CAPTCHA is used by web services to distinguish humans from bots. The problem is similar to multidigit recognition with the added complexity of greater number of possible outputs digits and a variable length of the sequence. However, for the purpose of this project we will choose to focus on first accuracy vs depth of network followed by the our methodology to reach the accuracy of the original work with lesser resources and smaller training time

2.2 Key Results of the Original Paper

One of the key results in this work is getting 96% accuracy without any attempt at segmentation or cropping. Also, they explored the possibility of deeper networks leading to greater accuracy, as shown in fig 3.

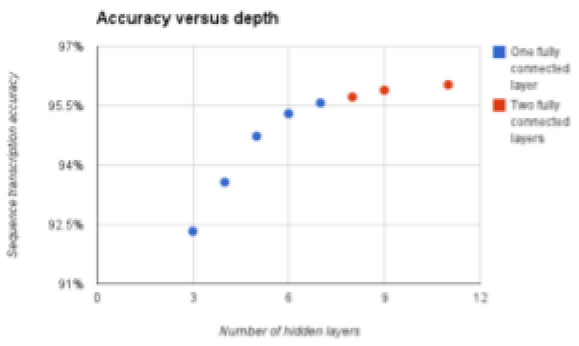


Fig.3

In [2], they plot a curve of accuracy vs depth of model. It seems to saturate beyond 10 hidden layers

3. Methodology

First we provide a comparison of accuracy vs depth as explored in the original work [2]. Followed by an analysis of how to reach the benchmark accuracy by modifying a fixed set of critical hyperparameters like learning rate. Finally we devise the coverage metric and plot the variation of performance with human level accuracy as a threshold.

3.1. Objectives and Technical Challenges

Our objective is to design a robust deep neural network that performs the task of digit recognition with sufficiently high accuracy and coverage at a moderate computation time and using easily available computational resources.

The challenges to this are the large dataset, unavailability of Google's hidden dataset to test the model against. Delving deeper into these problems, we see that: Tuning the model with many hyper parameters is difficult as training with one set of hyperparameters takes a huge amount of time. This limits our ability to tune several hyperparameters simultaneously. Also, predicting the output for the external images other than the given test data [1] is difficult as the digit structure mat file is not available to get the bounding box attributes.

3.2. Problem Formulation and Design

Coverage study

After training the model, we reached an accuracy of 93.5% on the test set. To get closer from the 98% accuracy goal (human performance), one solution is to discard before the evaluation some data from the set : the ones that are the least probable of being well interpreted by the model. In theory, it would reduce the set size and keep the correct prediction number constant, leading to an accuracy increase.

In order to do that, we need to consider closely the image score matrix. At the end of the training, each image is allocated an image score matrix of shape (11,5). At position (i,j), there is the score of the j-th figure to be an i (10 being the absence of figure). We should also recall that they are five positions because that is the maximum length allowed by the model. Thanks to softmax function, we can turn the scores obtained by each figures at each position into a probability.

For instance, if for a particular image, at position number one, the scores were : [-1.3, 2.0, 0.8, -1.1, 0.9, 2.2, 2.4,-0.8, -1.5, -0.9, -3.9], we obtain : [8.1e-03, 2.2e-01, 6.7e-02, 9.5e-03, 7.4e-02, 2.7e-01, 3.2e-01, 1.3e-02, 6.3e-03, 1.1e-02, 6.1e-04], which is a vector of sum 1. Then we consider only the highest probability in it, because that is the figure our model will choose. We take the mean of highest probabilities for the five positions to obtain the probability of the whole number. It appears that the least probable of being well interpreted are assigned small probabilities thanks to this metric. Choosing an adequate metric is crucial, it would be a pity to discard too many potential good results. Other metrics are worth of being considered, such as taking the minimum probability among the five positions, but we decided to focus on the mean, because of the results it provided.

In order to determine the minimum probability an image should have to make it into the final set, we evaluated our model for several values of it : [0.8, 0.85, 0.9, 0.925, 0.95, 0.975]. We computed the resulting accuracy and the resulting coverage size for the training set, the validation set and the test set. Here are the results :

Let's recall that with full sets, our model has an accuracy of 0.964% on training set, 0.96% on validation set and 0.93% on test set. The final objective is to get 98% accuracy on those sets (red line in the accuracy plot), especially on the test set. Let's not forget the coverage size they were able to reach in the paper : 95.64% (red line in the coverage plot).

We can see in the accuracy plot that increasing the minimum probability required increases the accuracy. At first (minimum probability = 0.8), there is a sizable gap between the test set accuracy and the other sets, but thanks to more selective coverage, the test curve is able to catch up. We were able to reach 98% of accuracy on the test set while keeping 91,5% of the images, with a minimum probability of 0.95.

On the coverage plot, we can observe that the test set is the less sensitive to high minimum probability, which is a good sign for the overall strength of our model.

On the test plot, we can observe the trade-off between the coverage size of the test set and its accuracy. It indicates that over 0.95, the trade-off is clearly unfavorable for the coverage size. It also tells us that our ideal value lies somewhere between 0.925 and 0.95.

By defining a simple metric, we were able to improve our accuracy to a human level without discarding more than 8.5% of the images.stated earlier.

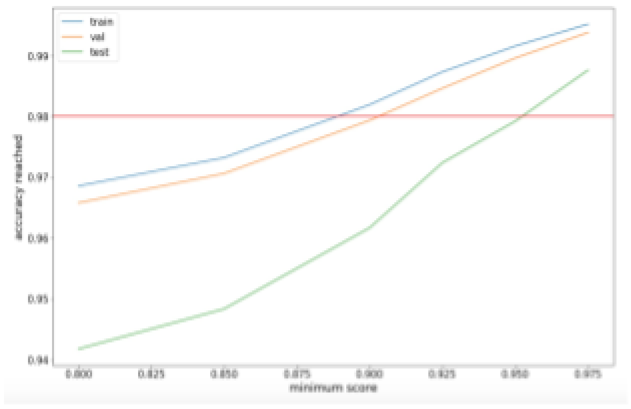


Fig. 4
Accuracy Plot

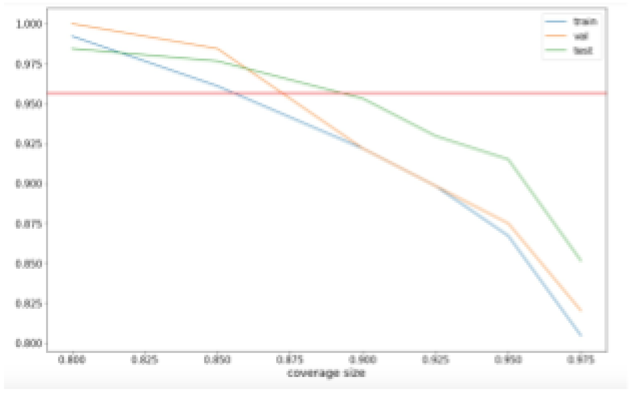


Fig.5
Loss Plot

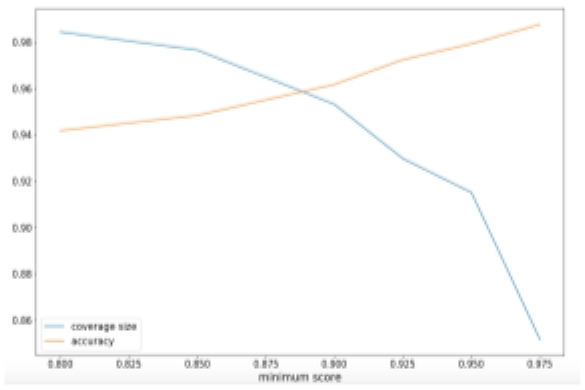


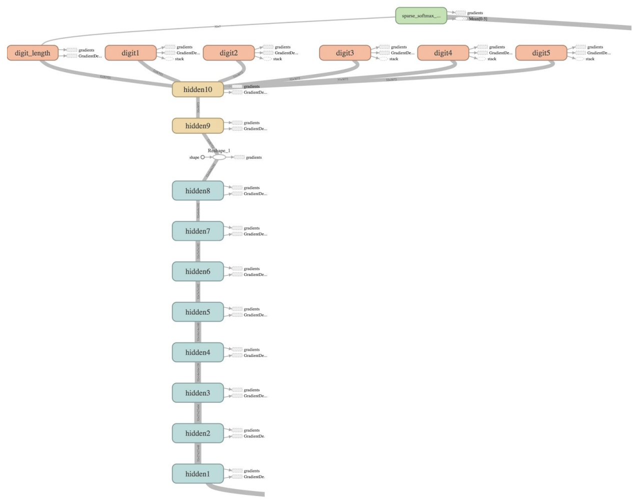
Fig.6

Test Plot

4. Implementation

We implemented the model in Tensorflow on Python 3.5. Also, the model is very similar to the best architecture presented in the paper barring minor differences like the use of max out at the input and one less convolutional layer.. The code organization is presented in the attached jupyter notebook

4.1. Deep Learning Network



4.2. Software Design

Following is the outline of the structure of our code and what each function achieves:
Create TfRecords:
1. Get the paths of the train, validation, and test data
2. Get the attributes of the image data from the digit structure mat file
3. Get the bounding boxes of the images from the attributes
4. Crop and resize the image to 64 * 64 by random cropping with box as the center of the image
5. Write the resized images to tfrecords file
6. Create TfRecords meta file which stores the length of examples for training, validation and test data

Data Utils:
1. Fetch batch size number of images from the TfRecords file in shuffled manner if specified.
2. Fetch the labeled values for batch images from the TfRecords file.
3. Convert image type and resize the image to 54 * 54* 3.
4. Return the batch images and their labelled values.

Model_accuracy:
1. Get the TfRecords of the train, validation, and test data.
2. Get the latest checkpoint model.
3. Fetch images and their correct labels in batch using Data Utils file.
4. Get the inference(predicted) values of the batch images from the model.
5. Check the accuracy of the predicted values.
6. Accuracy measure: Predicted value is correct only if all the digits and length of the digits are matched.
7. Prints the total accuracy value of the train, validation, and test data.

CNNModel:
1. Contains the layout of the model and the layers model is using.
2. Model is using total 10 hidden layers with 8 convolutional layers, and 2 dense layers.
3. Softmax layer for each digit and length of the digits after dense layers
3. It also calculates the softmax cross entropy loss of the feedforward output.

DataMeta:
1. Save function saves the number of examples each train, validation and test dataset has.
2. Load function gets the number of examples in the train, validation, and test dataset from the meta file.

Model_Train:
1. Make the Patience level as 100
2. Get the train, and validation data and their attributes from the TfRecords file
3. By using Batch utils fetch the batch images and their correct labels.
4. Get the inference of the data from the model.
5. Calculate the cross entropy loss of the output.
6. Train the model using gradient descent optimizer by exponentially decaying the learning rate.
7. Calculate the validation accuracy and check with

previously trained accuracy.
8. If there is a gain in accuracy, then save the model, else patience level decreases by 1.
9. Train the model until the accuracy gets saturated or the patience level of the model falls below significant level.

Check test images:
1. Fetch the batch images and their labelled values of the test data from the TfRecords file.
2. Preprocess the images using data utils function and predict the values using the latest model.

5. Results

5.1. Project Results

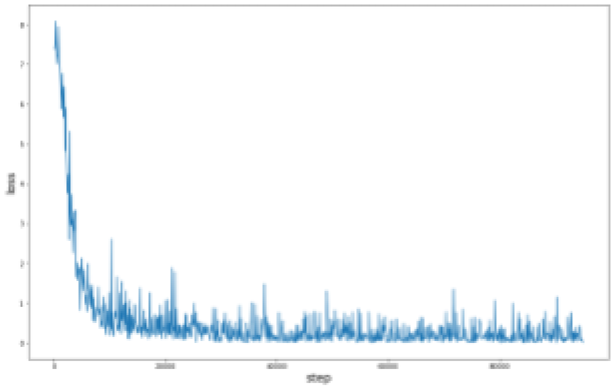


Fig. 7

Loss curve for 8-hidden layer model. Number of iterations is 103,000

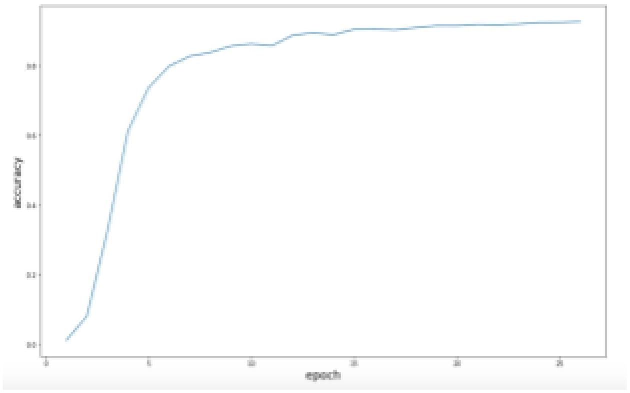


Fig. 8

Validation Accuracy curve for 8-hidden layer model. Number of iterations is 103,000 and final value is 94.01%

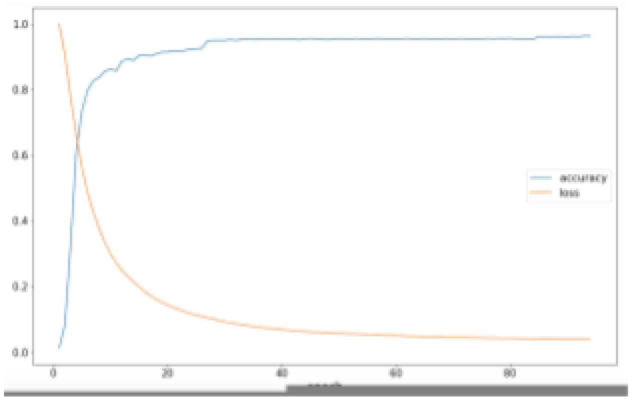


Fig. 9

A comparative visualization of loss and accuracy for 10 layer network. Validation accuracy = 96.4%

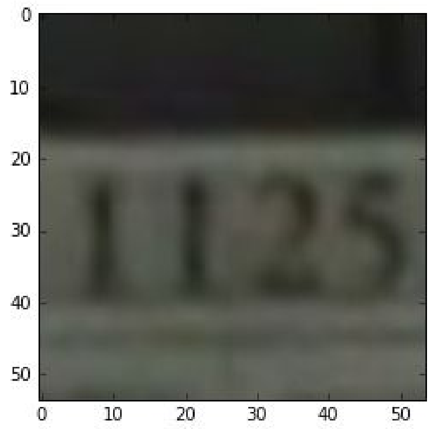
5.2. Comparison of Results

Model	Train Accuracy	Validation Accuracy	Test Accuracy	Coverage for 98%	Training time
From Paper	-	-	96.03	95.64	6 days
Our best model	96.4	96.1	93.5	95.3	14 hours
8 hidden layers model	94.01	94.5	91.07	91.4	9 hours

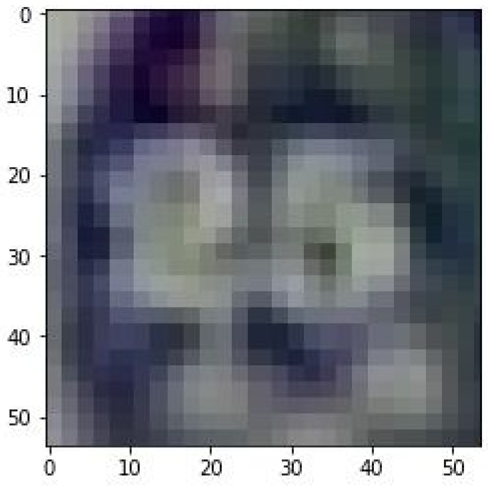
As we can see above the paper achieved a coverage of 95.64% for a test accuracy of 98% while we achieved 95.3% coverage. It is important to note that the coverage metric is not clearly defined in the paper. We have built on what was outlined and filled in the details with our own understanding of such a confidence measure. Below we have shown the results of our model on actual StreetView images of buildings in and around Columbia University.

Since we did not have access to Google’s hidden test data of StreetView images we sampled a few from buildings around the university to see how the model performs with real life examples. The findings of this experiment are documented below.

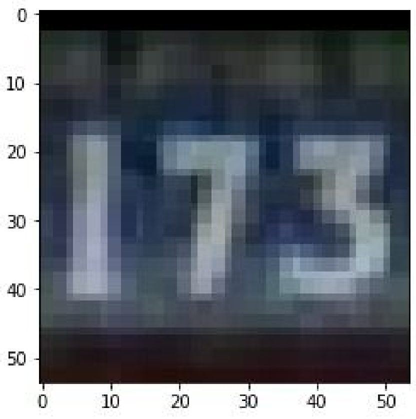
```
length: 4
digits: 1125
```



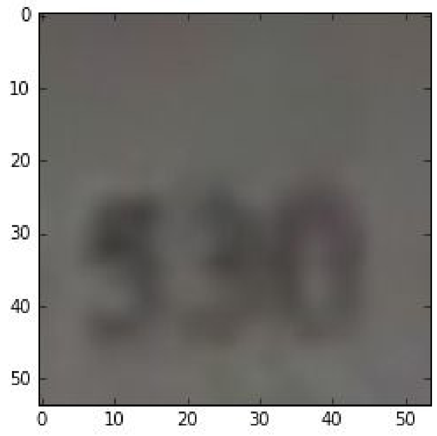
```
length: 2
digits: 25
```



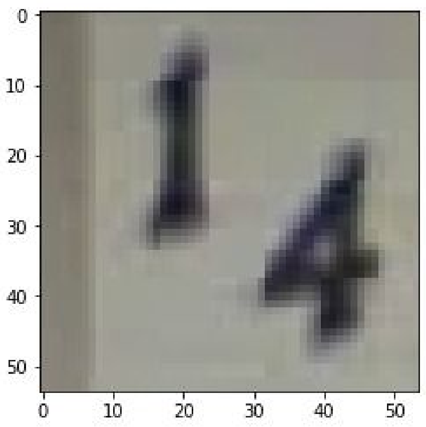
```
length: 3
digits: 173
```



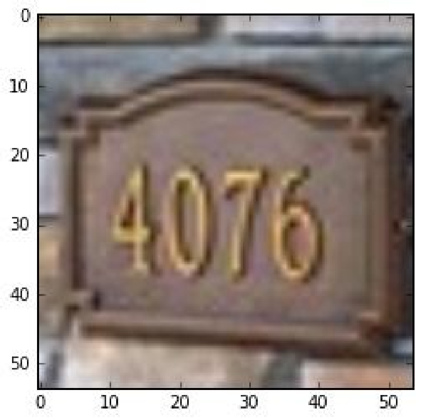
```
length: 3
digits: 530
```



```
length: 2
digits: 14
```



```
length: 4
digits: 1008
```



5.3. Discussion of Insights Gained

Multi-digit recognition is a complex problem that raises several issues. Answering those problems gave us precious insights.

- How to deal with a complex and heavy dataset in tensorflow ?

To make our computation running times shorter, we used TfRecords technique, that store the data in binary file to get rid of the time-consuming opening operation. What's more, we've trained our model on a GPU to make it even faster. The combination of both helped us dealing with such complex and heavy data sets.

- How to spot for the numbers in Google Street View images ?

To spot the numbers in our images, we generated bounding boxes around each figure, and it gave excellent results.

- Why is it important for a model to reach at least a human accuracy and how to achieve it ?

The notion of coverage is fundamental in the paper, because this task should be performed with a human accuracy. Indeed, predicting a wrong number is very problematic when you are building a map. So it is necessary to design a cogent metric to get rid of the data that are hard to predict by the model.

- How to overcome saturation of the accuracy curve ?

We overcame this problem by retraining the model on different machines using a different seed value for the learning rate and its decay.

6. Conclusion

Using easily accessible resources and a relatively small training time we were able to a test accuracy of 93.5% and a coverage of 95.3%. We trained the model for 14 hours as against to the 6 day training that the authors of [2] subjected their model to. Thus we were able to devise a confidence metric for our robust digit recognition system. In the future, we could extend this system for generalized character recognition to include alphabets.

6. Acknowledgement

We would like to thank Prof. Kostic and the Deep Learning TAs for a fantastic course that was structured well and aided greatly in our understanding of neural networks and deep learning. Prof. Kostic's classroom illustrations and visualizations along with the helpfulness of the TAs led to a solid understanding of various nuances of CNNs which eventually helped us figure out the insights we gained from a practical implementation like this.

7. References

Include all references - papers, code, links, books.

[1] https://louismassera@bitbucket.org/ecbm4040/2017_assignment2_lm3287.git

[2] Goodfellow, Ian J., et al. "Multi-digit number recognition from street view imagery using deep convolutional neural networks." *arXiv preprint arXiv:1312.6082* (2013).

[3] The Street View House Numbers Dataset. URL retrieved from: http://ufldl.stanford.edu/housenumbers/

[4] Street View House Numbers Recognition Using ConvNets, Ryn Ng. URL received from: https://ryannng.github.io/2016/12/20/Street-View-House-Numbers-Recognition-Using-ConvNets/

[5] Tfrecords Guide from : http://warmspringwinds.github.io/tensorflow/tf-slim/2016/12/21/tfrecords-guide/

8. Appendix

8.1 Individual student contributions - table

	aaj2146	lm33287	su2216
Last Name	Jadhav	Massera	Unnam
Fraction of (useful) total contribution	1/3	1/3	1/3
What I did 1	Report writing and proof reading	Report writing and figure compilation	Report writing and code comments
What I did 2	Evaluating function and test inferences code	Model and training code	Data preprocessing code
What I did 3			

.