Digit Recognition Capstone Project Report

MIchael Amundson

July 5, 2017

I. Definition

Project Overview

The goal of this project was to train a neural network to classify numerical digits from images of those digits. The challenge in doing this comes from the fact that the same digit can look quite different depending on the size, font and color.

Fig. 1 Examples of three different 3's

The task of recognising numbers is a subset of the larger task of optical character recognition. This is an important technology for converting images into characters that are recognized by a computer. This technology has a wide range of applications, from recognizing the characters in images of scanned documents to the new google translate app that can use a phone's camera to read written text and then translate it into a different language. Character recognition has been attempted with machine learning techniques such as linear classifiers, Support Vector Machines (SVM), K-Nearest Neighbor regression (KNN), and Neural Networks (NN) (LeCun 1998).

Some of the best results for character recognition have come with Convolutional Neural Networks (CNN). The LeNet architecture by Yann LeCun was trained on Mixed National Institute of Standards and Technology (MNIST) data which is a collection of individual handwritten characters and he was able to achieve 99.3% accuracy on individual digits. Ian Goodfellow trained a deep CNN on Google Street View House Number (SVHN) data (Goodfellow 2013) which contain a sequence of digits up to five digits long and was able to achieve 96% sequence accuracy. This project will focus on classifying images from the MNIST and SVHN datasets using a CNN.

Problem Statement

I want to build a classifier that can classify the digits in a given image and output their location. This program will need to determine the location of up to five digits in an image and classify each digit (0-9). This classifier will also need to be computationally cheap enough to train on my laptop in a reasonable amount of time and make predictions quickly. To achieve this I will start by analyzing MNIST data.

MNIST Data workflow

1. Generate sequences by concatenating one to five images together. By generating sequences this way it will be easier for the CNN to locate digits since each digit in the sequence will be in the same 28x28 pixel block. Also this is single channel data which will allow me to test more architectures in a shorter amount of time.
2. Test multiple CNN architectures to see which performs best on MNIST data classification. I will test architectures that are a smaller version of the Goodfellow architecture (Fig 2.)

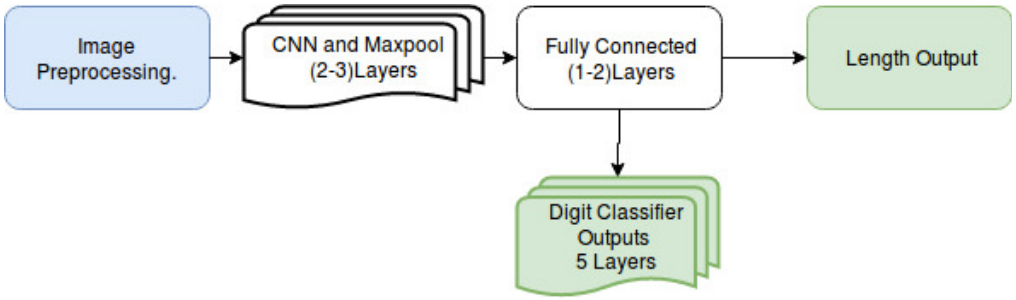


Fig. 2 CNN with less hidden layers but similar output to the Goodfellow architecture.

I will also test an architecture that contains an Inception Module (Fig 3.)(Szegedy, 2015).

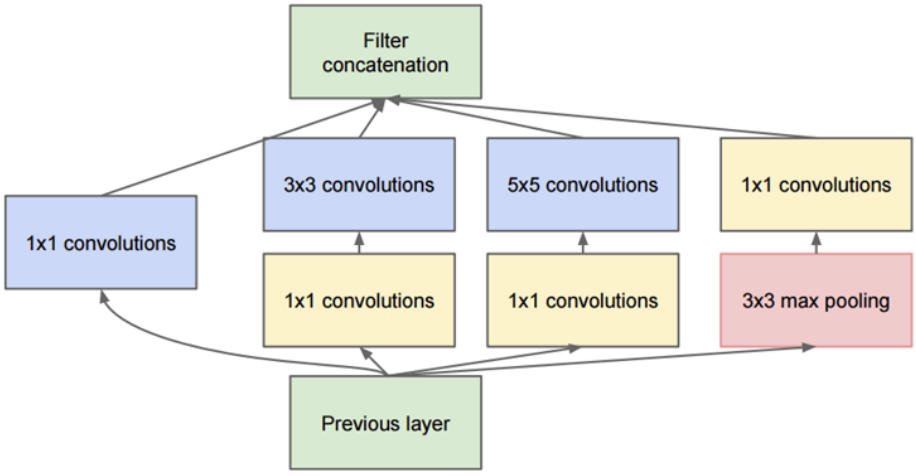


Fig. 3 Example Inception Module

SVHN Data workflow

1. Crop the images so that only the region which contains bounding boxes for all of the digits remains.
2. Scale the images so that they are all the same size and can be fed into a CNN.
3. Build and train a CNN based on the model I generated using MNIST data. This CNN will have an additional output layer that performs bounding box regression for each digit.
4. Optimize CNN based on validation accuracy and Intersection Over Union (IOU).

Metrics

I evaluated the SVHN networks using two metrics; sequence accuracy and IOU. I will evaluate my model's classification abilities based on the sequence accuracy defined as

$\frac{\#\ of\ images\ where\ every\ digit\ in\ image\ is\ classified\ correctly}{\#\ of\ total\ images}$. I split the training set and used ninety percent of the training data for training and saved ten percent as a validation set that I can view performance of and use to test different architectures and hyperparameters. To evaluate the localization abilities of my model I used average Intersection Over Union (IOU).

$IOU = \frac{area\ of\ intersection}{area\ af\ union}$ In this case IOU is the intersection of the predicted bounding box and the true bounding box divided by the union of the predicted bounding box and the true bounding box. A prediction perfectly matching ground truth would have an IOU of 1 while a prediction that completely missed the object would have a IOU of 0. I was not able to find a published benchmark for this dataset but I did find the heuristic that IOU's greater than 0.5 were generally considered good (Rosebrock 2016). My goal is to achieve average IOU's above 0.5 for all of the non-null digits.

II. Analysis

Data Exploration and Visualization

For this project I trained on both MNIST and SVHN data. I started on the MNIST data because it was simpler and I wanted to get a CNN that could do digit classification before I tried to add localization. I imported the sample MNIST dataset from Keras which consists of 60,000 training characters and 10,000 test characters. Each image is single channel and 28x28 pixels. Since MNIST is made up of single characters I concatenated three to five randomly selected characters in a row to form a sequence. For sequences shorter than five characters I filled the remaining space with null and labeled that space 10. This formed an input to the neural net that was 28x140. I generated 40,000 training sequences and 4,000 test sequences.

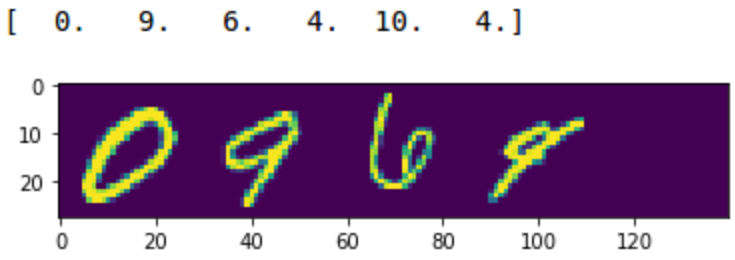


Fig. 4 Sample MNIST Sequence and label. The the first five labels are digit classifiers and the last label is the length of the sequence. A classifier label of 10 signifies a blank digit.

To develop the ability to classify color images that are not lined up in a row and are not all the same size I also trained on the Google SVHN dataset (Netzer 2011). The data is split into three collections of images, the training data is made up of 73,257 digits contained in 33,402 images, the testing data has 26,032 digits in 13,068 images and there are 531,131 digits in 202,353 images that were available to use as extra training data. The images in this dataset have three channels and are photos taken of house numbers with bounding boxes that give the location of the digits. This bounding box data allowed me to add an additional output layer to my CNN that provided localization with a bounding box regression and recognized where a digit was within an image.

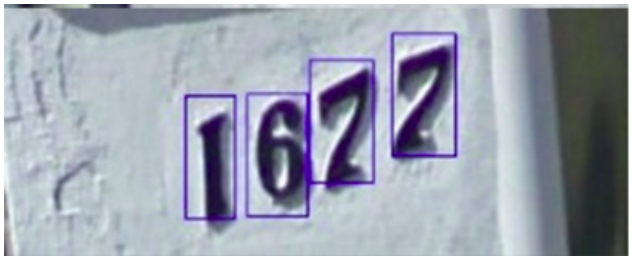


Fig. 5 Sample SVHN image showing bounding boxes.

Analysis of SVHN digits.

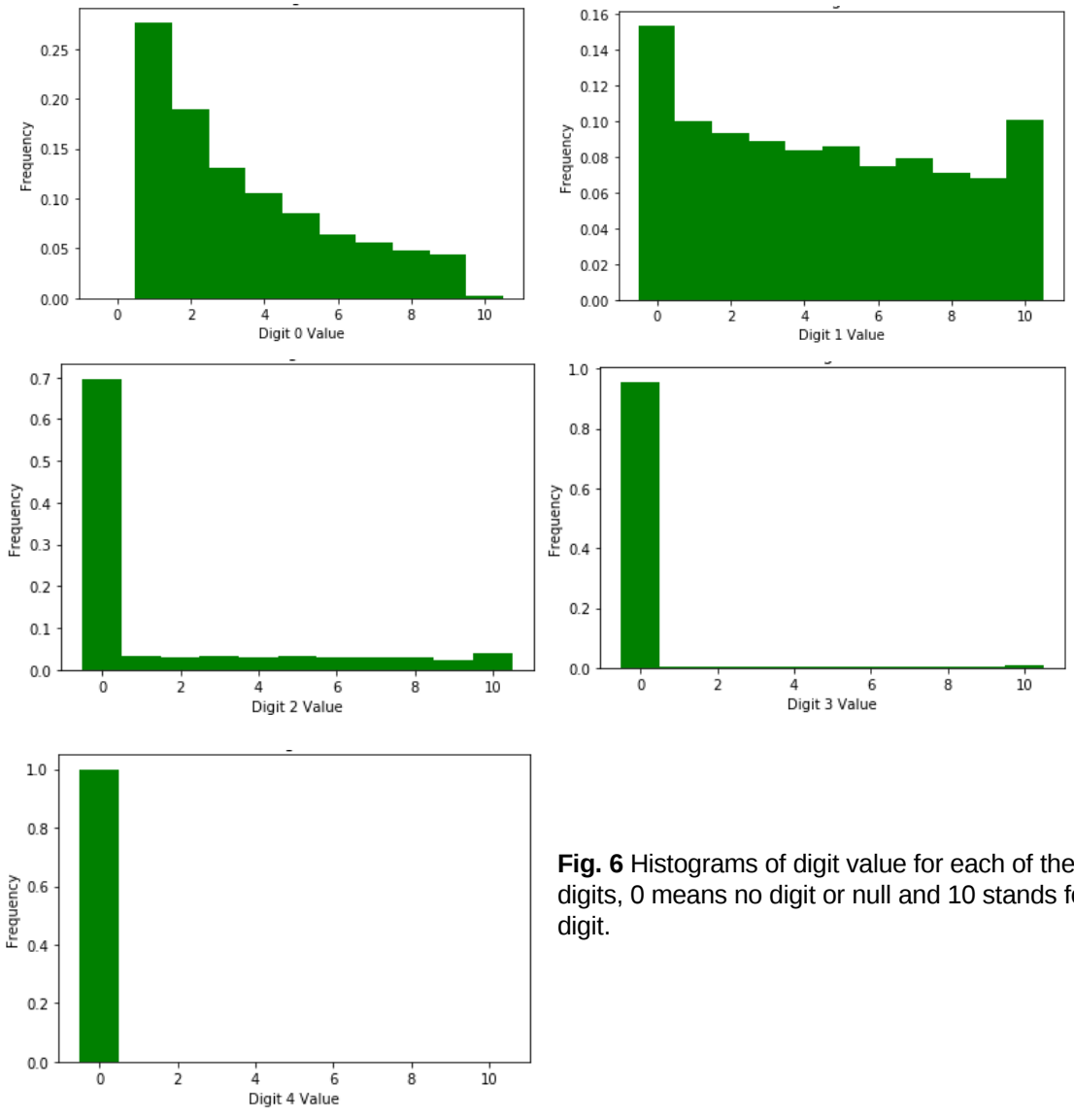


Fig. 6 Histograms of digit value for each of the five digits, 0 means no digit or null and 10 stands for a 0 digit.

I generated histograms of the digit values and was surprised to find that when we looked at digit zero we see that lower digits are overrepresented significantly. This is actually an example of Benford's Law which states "in many naturally occurring collections of numbers, the leading significant digit is likely to be small" (En.wikipedia.org, 2017). Looking at the second digit we see that the none category is already the predominant classification followed by zero. The last three digits are all heavily dominated by null. I had some concern that the last three digits would not perform well since they are so heavily weighted towards null. I tried applying weights to the loss function for the last three digits however this hurt overall accuracy and so is not part of the final model.

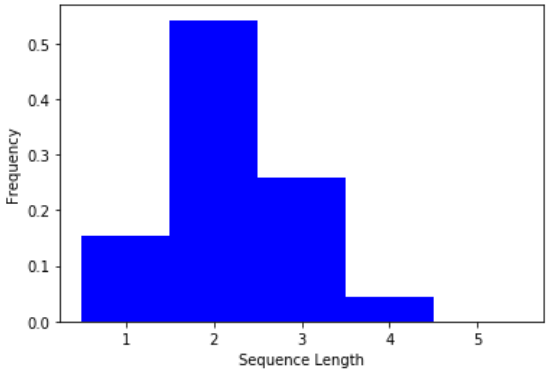


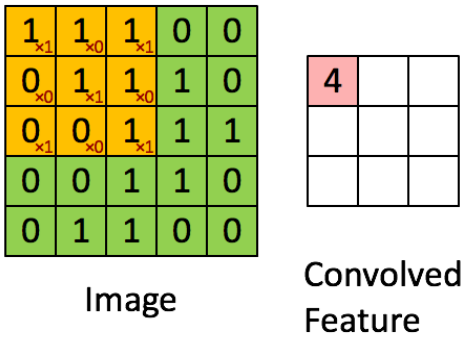
Fig. 6 Histogram of sequence length.
Looking at a histogram of sequence length we see that 2 house numbers is the most common length of sequence followed by 3 and then by one. There are hardly any sequences with a full five digits so the ability of our network to classify the fifth digit will likely not be as accurate.

I wanted to look at the width to height ratio for the images before they were resized because when I resized the images to a square I noticed that some of the images were becoming distorted. I cropped the image down to only the rectangular location that contains all of the bounding boxes before I resized the image. I looked at the width to height ratio of the image after cropping. Since these ratios were very close to one I felt justified in using a square input into the neural net.

Dataset	**Mean width-to-height ratio**
Standard training data	0.99
Extra Training data	1.23
Test Data	1.02

Fig. 7 Mean width-to-height ratio for the three datasets.

Algorithms and Techniques



Convolutions

A convolutional layer in a neural net is a layer that uses a shared set of weights for multiple regions of the image.

Fig. 8 A simplified convolution where the green matrix is the image where 0 is black or 1 is white. The 3x3 yellow window is a filter or kernel that passes over the image and outputs the result of an element wise multiplication of the weights of the filter and the pixel values at each location. (Image from Stanford Deep Learning wiki)

In an actual image classifier there will be three channels for the input data (Red, Green, and Blue). There will also be multiple filters used so that different features can be captured from the data.

ReLU

ReLU stands for Rectified Linear Unit. This type of unit was used as an activation function for the convolutional and fully connected layers. The main purpose of the ReLU is to provide non-linearity to the network and it does it in a very computationally efficient way. ReLUs also have sparse activation, since for a random initialization half of the units will be inactive. They have efficient gradient propagation as they avoid vanishing or exploding gradients since the derivative of the ReLU will always be 0 or 1.

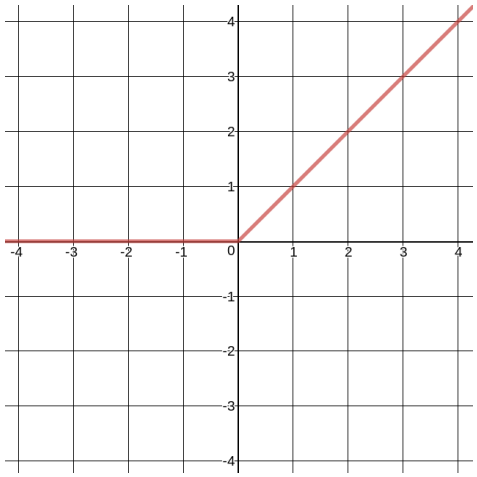
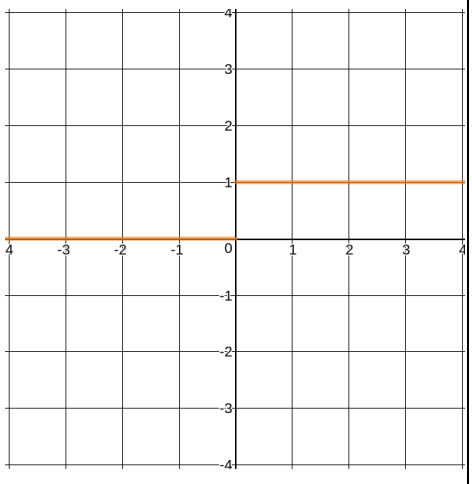
ReLU	**Derivative of ReLU**
f(x) = {x≤0 : 0, x>0 : x}	f'(x) = {x≤0 : 0, x>0 : 1}
	

Fig. 9 Graphs of ReLU activation function.

Max Pooling

Max Pooling is a technique used to downsample the neural net. This reduces the number of parameters that the network has to learn making the network computationally cheaper and less likely to overfit. The max pooling operation simply takes the maximum from a given area of interest. In my network all of the max pooling draws from a 2x2 pool. An example of max pooling is shown below which down samples from 16 units to 4.

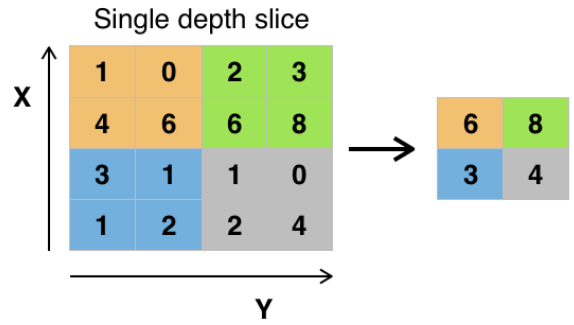


Fig. 10 Example of 2x2 max-pooling (Image from aphex34 on wikipedia)

Batch Normalization

Batch normalization is a technique used to prevent a few units from getting very large and overwhelming the rest of the network. The keras batch normalization layer normalizes the activations of the previous layer so that they have a mean close to 0 and a standard deviation close to 1. Since the output of the ReLU layer will always have a positive mean this means the batch normalization will shift the distribution mean back down to zero.

Dropout

Dropout is a NN training technique where the units in a layer of the NN are set to zero (dropped) based on the assigned dropout probability *p*. The remaining units have their output boosted by a factor of Error. When it is time to make validation or test predictions the dropout is removed and all the units are used with unboosted values. Dropout prevents overfitting by preventing overly complex representation of the data forming in the NN. It also causes the network to learn redundant representations of the input data for better prediction of output.

Softmax

Softmax is used as the activation on the final output layer. Softmax is defined by the equation in figure 10 where **z** is the input vector.

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

Fig. 11 Equation for calculating softmax.

Softmax puts most weight on the max probability outcome and minimizes weight on less likely values. This means that it will not change the prediction of the network but if the network has

made a correct prediction it will minimize the loss and if the network has made an incorrect prediction it will maximize loss which is exactly what we want to do.

Loss Functions

Categorical Cross Entropy (CCE) was used to calculate the loss in the classification output. For a single datapoint CCE is calculated using the equation to the left where p_i is the true probability and q_i is the predicted probability distribution. To minimize loss you want maximize the probability placed on the correct category and minimize the probability placed on the incorrect answer.

$$H(p,q) = -\sum_i p_i \log q_i$$

Fig. 11 Equation for calculating Categorical-Cross-Entropy.

Mean Square Error(MSE) is the average of the squares of all the error. MSE was used to calculate loss for the bounding box regression layer of the network.

Benchmark

The Goodfellow model will be the benchmark. Their model was a CNN that recognized the digits in SVHN data. Their model consisted of eight convolutional layers, one locally connected layer and two fully connected layers. They used six softmax output layers, one for each digit and one for the length of the sequence. They did not feed the location of the images to the network so the network had to come up with its own spatial representation. The model by Goodfellow et al achieved 97.84% character accuracy and a 96.03% sequence accuracy on the SVHN test set. I will use a smaller network due to limited computing abilities however I had a goal of achieving greater than 90% digit and greater than 80% sequence accuracy.

III. Methodology

Data Preprocessing

The MNIST data required minimal preprocessing. Each individual character was 28x28 pixels so I simply concatenated up to 5 characters to form an image that was 28x140. I also changed the range of the data from 0-255 to 0-1 in an effort to improve numerical stability of the CNN in training.

The SVHN data required more preprocessing.
1. Crop the image to the smallest rectangle that still contained all of the bounding boxes of the individual digits.
2. Resize the image to 54x54 pixels. I used scipy.misc.imresize to resize the image with bicubic interpolation as that seemed to give the best quality resized images.
3. Scale the bounding box coordinates from their position on the original image to the appropriate position on the 54x54 image.
4. One-hot encode the labels.

5. Scale the color values from 0-255 to -.5 to .5 again to improve stability of the network.

Implementation

One of the most difficult parts of this project was selecting the correct architecture for the CNN. The best performing architecture on the MNIST data that I was able to find was the one in figure 11. This architecture was able to achieve a digit accuracy of 0.9785 and a sequence accuracy of 0.89625. All convolutional layers have stride 1 and pooling layers have stride 2. All convolutional and fully connected hidden layers have ReLU activation.

1	Input 1x28x140 Image
2	5x5 30 Filter Convolution
3	2x2 Max Pooling
4	3x3 15 Filter Convolution
5	2x2 Max Pooling
6	Dropout 40% of Data
7	128 Node Fully Connected
8	128 Node Fully Connected
9	6-11 node outputs with softmax activation.

Fig 12. Best performing CNN architecture on MNIST data.

I also attempted to implement a sIngle layer of the Inception Architecture (Szegedy 2015). This did not match the performance of the simple convolutional architecture with a digit accuracy 0.94815 sequence accuracy 0.76775.

1.	Input 1x28x140 Image		
2.	1x1 64 Filter Convolution	1x1 64 Filter Convolution	3x3 Max Pooling
3.	3x3 64 Filter Convolution	5x5 64 Filter Convolution	1x1 64 Filter Convolution
4.	Concatenation of 3 branches above		
5.	30% Dropout		
6.	128 Node Fully Connected Layer		
7.	6-11 node outputs with softmax activation.		

Fig 13. Inception-like architecture.

To improve the performance of the inception layer I tried adding some more convolutional and fully connected layers but only gained a slight improvement to a digit accuracy of 0.95325 and a sequence accuracy 0.7875.

When I moved on to the SVHN data I kept a similar architecture to the highest performing MNIST network however I added batch normalization and a third convolution. I also increased the size of the first two convolutional windows and added an output layer that was trained on the bounding box coordinates of each digit. I assigned null digits negative coordinates during training so that the network would learn to put them outside of the normal viewing range.

1	Input 3x54x54 Image	
2	9x9 32 Filter Convolution	
3	2x2 Max Pooling and Batch Normalization	
4	5x5 32 Filter Convolution	
5	2x2 Max Pooling and Batch Normalization	
6	Dropout 40% of Data	
7	3x3 64 Filter Convolution and Batch Normalization	
8	256 Node Fully Connected	
9	128 Node Fully Connected	
10	Dropout 40% of Data	
11	6-11 node outputs with softmax activation.	20 node output for bounding box regression

Fig 14. Best performing architecture on SVHN data.

Refinement

My first SVHN network was like my final MNIST network in that it contained only digit classification and length of sequence output layers. In order to achieve better results on the SVHN data I added another convolutional layer to the network.

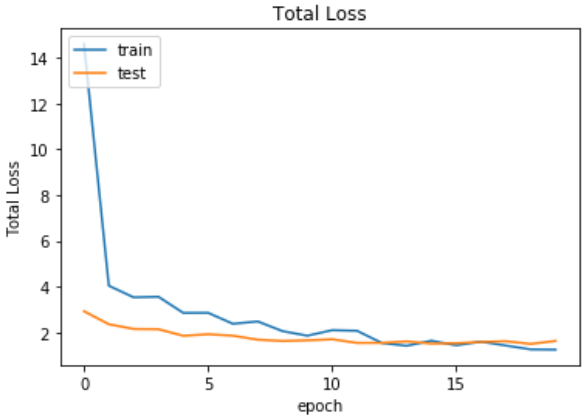
After running my CNN with all equally weighted layers I noticed that the contribution of the loss from the bounding box output layer was significantly smaller than the loss contribution of the classification layers. This corresponded with poor localization performance as seen in the table. I was able to increase the weight on the loss from the bounding box layer and achieve better localization. As I changed the weight for the bounding box loss the weight for losses on all other output layers was kept constant at 1.0.

Bounding box weight	Sequence accuracy	IOU
1.0	0.82	0.584
2.0	0.81	.609
10.0	0.81	0.685
20.0	0.74	0.688
50.0	.185	0.690

Fig 15. Table showing model performance while changing the weighting given to localization.

IV. Results

Model Evaluation and Validation



I limited the training to 20 epochs since the last 6 epochs didn't show significant improvement in the metrics I was interested in. I did not look into how to implement learning rate decay in keras but I think something like that might have allowed me to train longer and end up with better results.

Fig 16. Total Training and Validation Loss for the network.

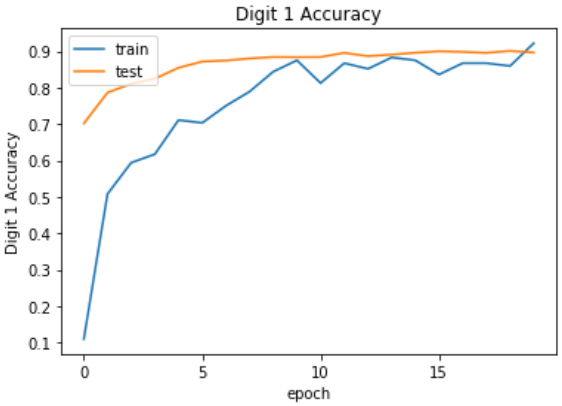


Fig 17. Training and Validation accuracy for the first digit.

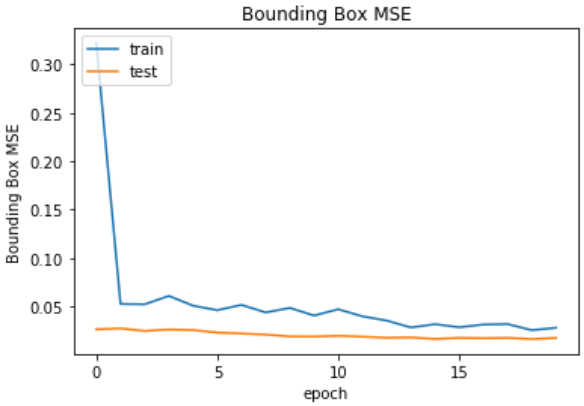


Fig 18. Training and Validation MSE for bounding box coordinate output layer.

After training I made predictions on the first 10,000 training samples and the test data. The classifier actually performed better on the test data than on the training data so we can say that the classifier generalizes well to unseen data and that we did not significantly overfit.

Dataset	Digit Accuracy	Sequence Accuracy	IOU
First 10,000 training samples.	94.7%	80.3%	.660
Test Data	96.2%	84.9%	.706

Fig 18. Comparison of results on training and test data.

Justification

My results of 84.9% sequence accuracy did not match Goodfellow's 96.03% however this network also provided localisation output with a respectable IOU of .706 and this was all done on a network with less than 900,000 parameters while Goodfellow's best network had over 50,000,000 parameters.

V. Conclusion

Free-Form Visualization

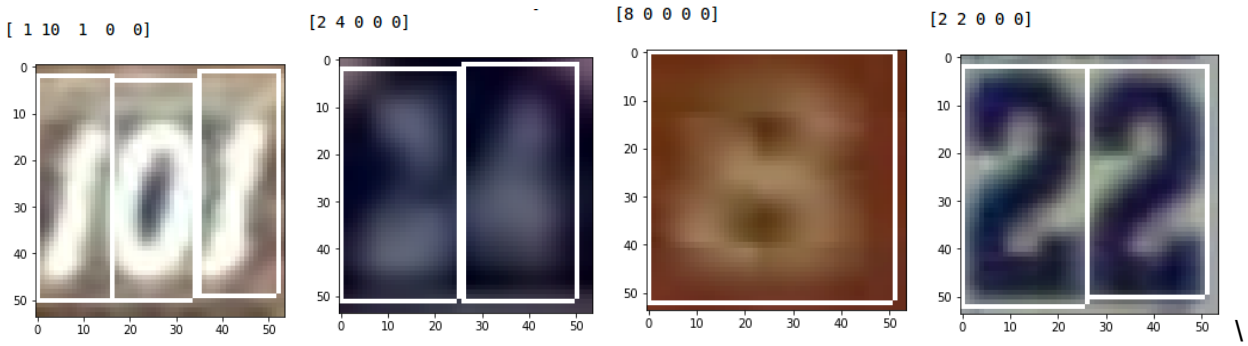


Fig 19. Four examples of correct digit classification and bounding box prediction. Predictions are in [ ] and zero represents a null digit while 10 represents a digit with value 0.

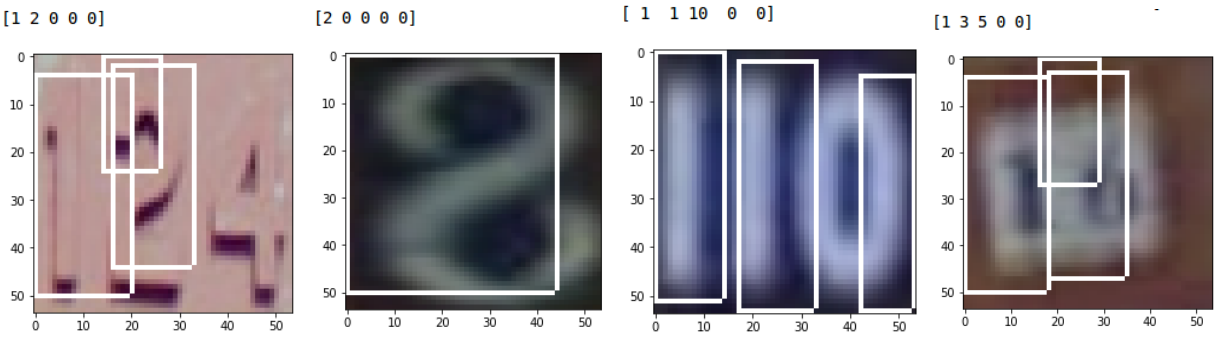


Fig 20. Four examples of mistakes made by the network.

The network tended to struggle more with images that had less contrast between the numbers and the background (First Example). In some cases the classification was correct but the

bounding box regression failed to put the box in the correct place (Third Example). Lines in the image that were not part of a number also cause the network to struggle (Fourth example).

Reflection

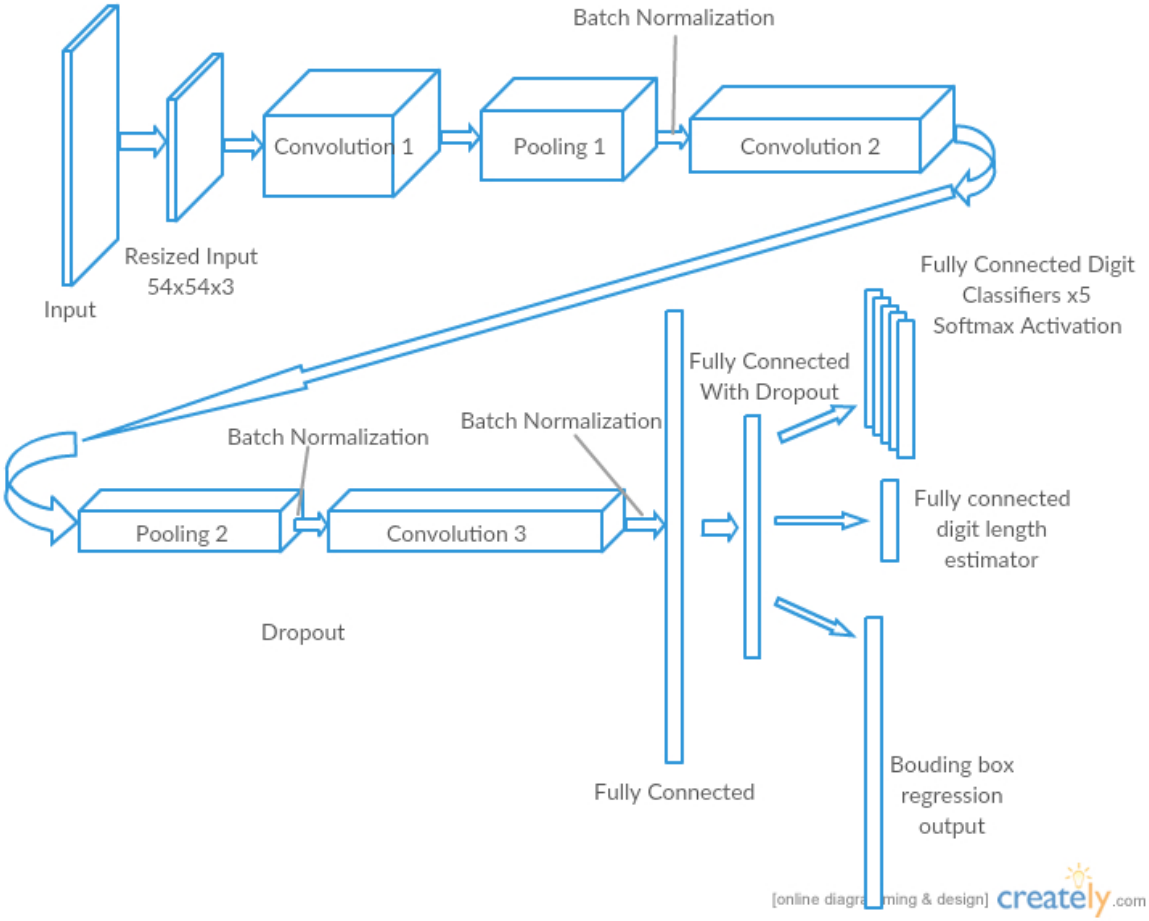


Fig 20. Diagram of the final architecture I used to provide classification and localization of digits.

There were lots of examples online of models that did classification but adding on a localization layer was new and a very interesting and challenging experience. Learning where I could get improvement in the model with better tuning was also very interesting.

Difficult parts of the project included figuring out how to get the data into a format I could use and keras would accept. Training the large network on my laptop and finding out the hard way how big of a model my machine could fit into memory. Installing all the necessary software was also a challenge as I tried to install on Windows but ended up converting my machine to Linux for easier installation and better performance with Tensorflow.

Improvement

On a project of this size there are many potential areas for improvement.

- Better parameter tuning - There are so many hyperparameters to tweak like number and size of layers, size of filters and pooling windows etc. that it is difficult to be able to test them all. Implementing a Gridsearch algorithm would be helpful in this regard.
- Image scaling - Many of the pictures were blurry after being shrunk to 54x54 so a better resizing algorithm would help the performance of the network.
- Data augmentation - Goodfellow used several random shifts of each image to increase the size of the data set and this would likely give improved performance.
- Bigger network - There was a tradeoff between weighting for localization and weighting the network for classification. When I increased the size of the network it was able to do both more effectively so an even bigger network would likely perform better.

References

Ian J. Goodfellow, I. Bulatov, Y. Ibarz, J. Arnoud, S. Shet, V. (2013). Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks. Technical Report, arXiv:1312.6082

Y. LeCun, L. Bottou, Y. Bengio and P. Haffner: Gradient-Based Learning Applied to Document Recognition, Proceedings of the IEEE, 86(11):2278-2324, November 1998

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

Rosebrock, Adrian. "Intersection over Union (IoU) for Object Detection."PyImageSearch. N.p., 27 Sept. 2016. Web. 27 Apr. 2017.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich Going Deeper with Convolutions, IEEE Explore, 2015

En.wikipedia.org. (2017). *Benford's law*. [online] Available at: https://en.wikipedia.org/wiki/Benford%27s_law [Accessed 27 Jun. 2017].

"Feature Extraction Using Convolution." Unsupervised Feature Learning and Deep Learning Tutorial. N.p., n.d. Web. 04 July 2017.