Convolutional Neural Network Image Recognition and Bound Box Prediction

1.0 DEFINITION

1.1 Project Overview

Image recognition, put simply, is the process of a computer being able to extract information from a digital image that is fed into its learning algorithm. There are myriad real world applications where image recognition can be seen making an impact: cancer screening, location services, home decoration, video conferencing, home security, self driving cars, just to name a few. As graphical and computational processing power continues to increase the uses for image recognition software will continue to expand, and become more refined as the field of machine learning continues to matures. One of the most effective methods for image recognition are convolutional neural networks (CNNs). CNNs operate much like human brain does when an image from the eye is transmitted to the brain. The whole image is broken up into smaller, overlapping images that the brain analyzes for visual cues that it uses to make the best prediction of what object is its field of view. The information derived from this process is the image recognition. Learning happens by the brain remembering what details in these small images are important to identify a certain object so in the future a particular object will be more easily identified. CNNs have an advantage compared to other image recognition algorithms as they are relatively simple in their construction and have shorter run times comparatively. This advantaged coupled with the increase in graphical processing power and access to large image databases makes CNNs an ideal method for image recognition.

1.2 Problem Statement

Design and implement a CNN algorithm that learns to recognize sequences of digits and predicted their corresponding bounding boxes. This will be demonstrated by training the model on a combination of the MNIST and SVHN datasets in the following steps:

1. Download the SVHN and MNIST datasets
2. Create a synthetic dataset using the MNIST data similar in structure of that to the SVHN dataset
3. Preprocess the synthetic MNIST and SVHN datasets
4. Construct and train a CNN classifier capable of identifying varied sequences of digits
5. Test the CNN classifier on test images to verifying digit recognition capability
6. Use SVHN dataset with ground truth bounding boxes to train CNN of matching architecture
7. Use newly captured images to test the validity of the model’s bounding box prediction

1.3 Metrics

Accuracy will be determined using predicted digits corresponding to their ground truth labels, and measured every 100 steps for the training batch and validation dataset, and once at the end of the simulation measuring the testing dataset. The accuracy metric was chosen as it is the most comprehensive measure a model's performance when predicting digit and bounding box recognition, and is the common metric used in the vast majority of similar studies of this nature. Accuracy is measured by taking the max probability of each digit prediction from sparse softmax cross entropy, comparing these digits to the true value, and taking the mean of the correct predictions over the whole batch or dataset. The calculation itself is:

$$Accuracy = \frac{True\ Positives + True\ Negatives}{Total\ Samples\ of\ Batch\ or\ Dataset}$$

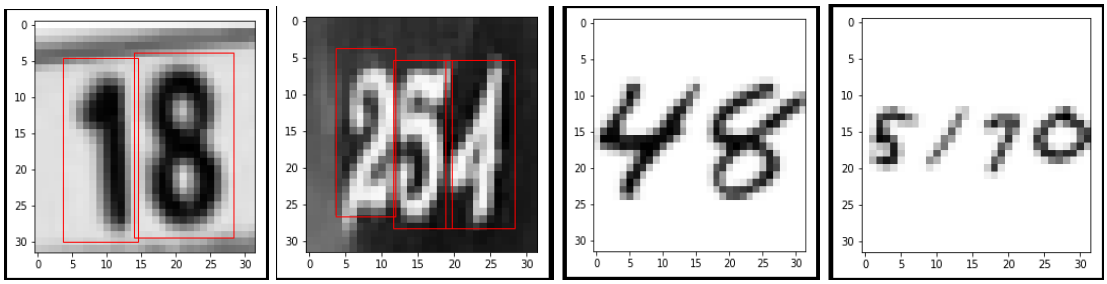
True positives and true negatives in this case refer to the model correctly identifying the digits when they are present in the image and not identifying a digit when it is not. A false positive refers to a digit incorrectly classified or acknowledging a digit that is not actually in the image at all, and a false negative refers to the model ignoring the presence of digits that are actually present in the image.
The bounding box accuracy will be determined by comparing if the predicted bounding box area is localized within 50% of the area of the ground truth bounding box. This equation is called intersection-over-union (IOU):

$$IOU = \frac{Area\ of\ Overlap\ between\ ground\ truth\ and\ predicted\ bounding\ box\ areas}{Area\ of\ Union\ of\ combined\ ground\ truth\ and\ predictied\ bounding\ box\ areas}$$

2.0 ANALYSIS

2.1 Dataset Exploration

In this study the MNIST and SVHN datasets were used to create a combined dataset of hand drawn digits and house numbers in groupings of 1-5 digits. There are a total of roughly 320k images: 280k training images, 15k validation images, and 23k testing images. The images are 32x32x1 grayscale format with 32 representing the pixel width and height and 1 representing the gray color dimension. Each image has a corresponding label which lists the numbers of digits in the image and digit themselves, including a label representing the absence of a digit in cases where there are less than 5 digits (the maximum number of digits in an image). The SVHN dataset also includes bounding box information which will be used in the second half of the project to determine digit location.

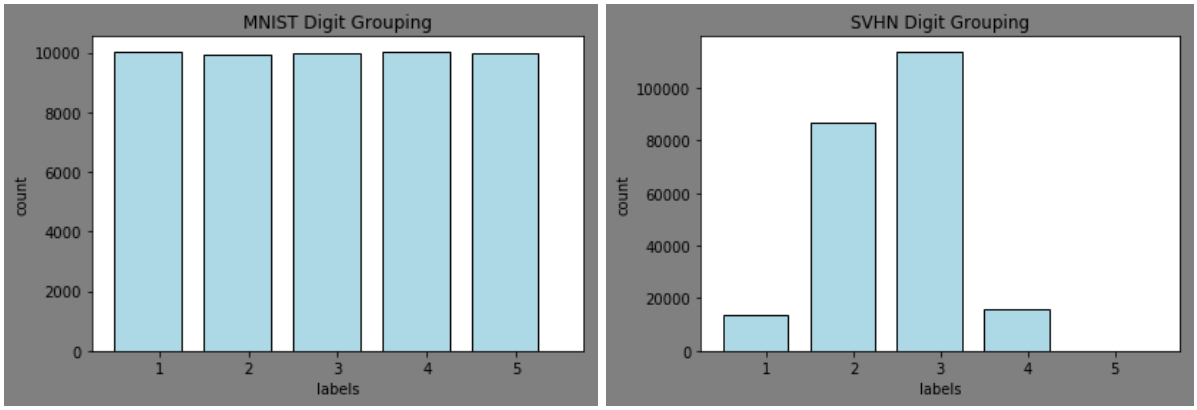


(a) (b) (c) (d)

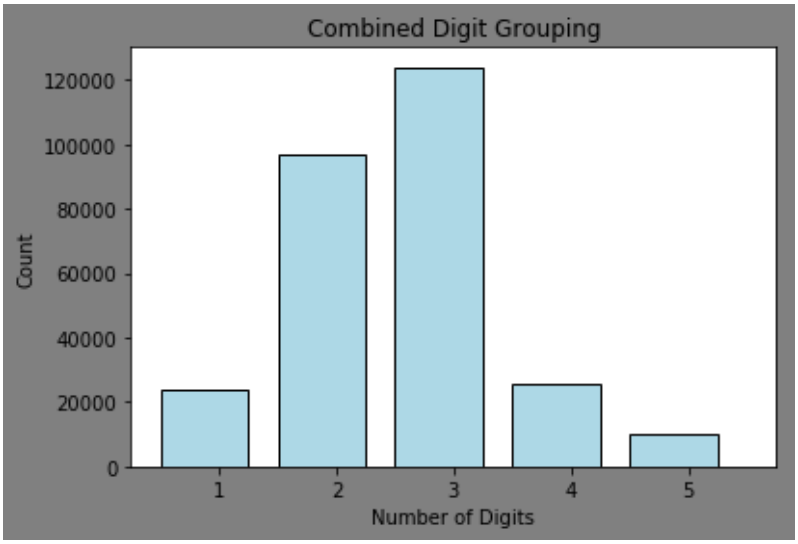
Fig 1. (a) SVHN 18 bounding box (b) SVHN 254 bounding box (c) MNIST 48 (d) MNIST 5170

2.2 Data Exploration

The images created from the MNIST dataset are uniformly distributed amongst the five digit lengths with roughly 10k images per class. The SVHN dataset on the other hand relies heavily on images with 2 and 3 digits accounting for roughly 180k, while 3 and 4 digits have a count of roughly 15k a piece, and 5 accounting for only 122. When the datasets are combined the distribution resembles the SVHN distribution, but with a better representation of images containing 5 digits.



(a) (b)



(c)

Fig 2. (a) MNIST digit count (b) SVHN digit count (c) Combine dataset digit count

2.3 Algorithms and Techniques

As stated in the introduction, a convolutional neural network was used as the classifier for digit recognition and bounding box prediction. The CNN consisted of input, convolutional, ReLu, pooling, fully connected, dropout and output layers.

- **Input layer** - Where the raw, 2D, grayscale images are initially fed into the network.
- **Convolutional layers** - The backbone of the CNN. They are used to filter the image using a pixel window that is size **Height**x**Width,** as seen in *fig 3,* with a predetermined filter depth. As this window is slid over the image, the dot product is computed for the region being convolved and produces a 2D map for each layer of depth or filter. Each filter is looking for a different feature that will help classify the image later on in the network
- **ReLu (rectified linear unit)** - An activation function that applies a nonlinear operation, $\frac{1}{1+e^x}$, to the linear elements resulting from convolution, and is implemented in conjunction with the its corresponding convolution layer. ReLu is preferred over tanh or sigmoid activation functions as it allows a larger range of value, max(0, x), which allows backpropagation to have a more significant impact on the values throughout the network, and decreases the influence of the vanishing gradient problem.
- **Pooling layer** - A layer which uses a process called downsampling to reduce the amount of features in the network by selecting the max value amongst a group of neurons using a sliding window of **Height**x**Width** much like the convolutional layer. This reduction reduces the computation needed to evaluate the images while leaving the filter depth unchanged, and is implemented in conjunction with its corresponding convolutional layer.

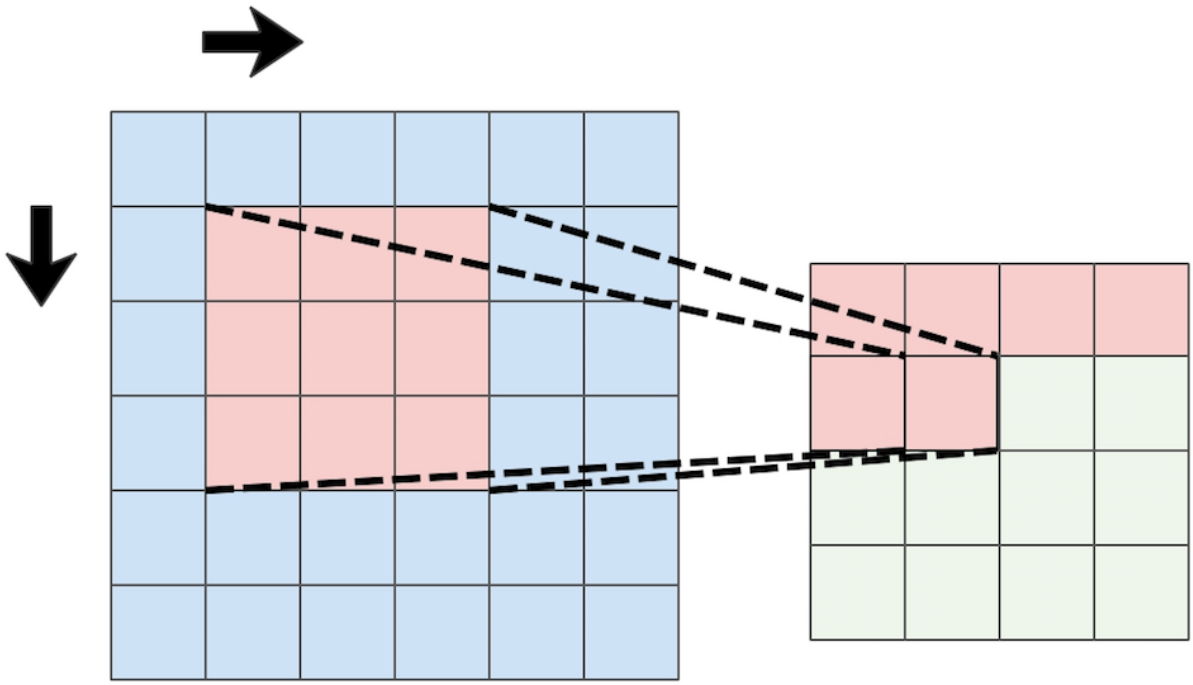


Fig 3. A rudimentary diagram of the sliding window used in convolutional and pooling layers

- **Fully Connected layer -** Connected to all the neurons from the previous convolutional layers representing high level features that will be used to predict individual digits.
- **Dropout layer** - A form of network regularization. It randomly removes activations from the network so it is forced to train with a reduced amount of inputs, thus reducing overfitting by never training on exactly the same data.
- **Output layer** - Where the classification of the digits are actually predicted using a sparse softmax algorithm to determine the best probability for the digits in sequence. As the loss is calculated at the end of each training session the network adjusts the weights to minimize loss by using backpropagation, which is a method of implementing the derivation chain rule. The derivation of the loss works its way backwards through the network calculating both the error of the previous layer and the cumulated weight change of the current layer by subtracting the product of the weight change and the learning rate by the current weight. This process continues until it propagates through the entire network and all weight are adjusted.

Listed below are the training parameters that were used to fine tune the classifier:

- Layer depth of convolutional and fully connected layers
    - The number of filters that will be used to analyze the data from the previous layer
- Size of the convolutional and pooling windows
    - This is the size of window that will scan the image from the previous layer

- Stride of the convolutional and pooling layers
    - The amount of pixels the window shifts when analyzing an image
- Padding of convolutional and pooling layers
    - Either '**VALID'** or **'SAME'**, this option determines whether the scanning window stops at the edges of the image (**VALID**) or is padded by zeros outside the image in order to retain the image's original dimensionality (**SAME**)
- Learning rate
    - A variable that determines how quickly or slowly the the gradient descent converges. An ideal learning rate will be of a size that will train the model in a reasonable amount of time and achieve good results on convergence. A popular method of achieving this outcome is using a decaying learning rate that has a high value when training begins and is reduced as training continues.
- Optimizer algorithm
    - An algorithm that helps to minimize the loss function
- Training length
    - The amount of epochs ran during training
- Training batch size
    - Number of images viewed during a single training step
- Training dropout probability
    - The percentage of neurons deactivated at the dropout layer during training

2.4 Benchmark

The initial benchmark goals is set at <5% prediction error based on similar CNN models that have classified the MNIST and SVHN datasets. Similar project have achieved as high as 99% digit recognition, but I am doubtful I will be able to reach numbers that high with my computer's performance capabilities. The bounding box benchmark is set at <15% prediction error. This will hopefully be high enough to accurately encompass the digits in most predictions, but if not, the number will be adjusted until a decent working model is designed.

3.0 METHODOLOGY

3.1 Data Preprocessing

1. Created the file *pickle_work_around.py* to save and load pickle files larger than 2 GB
2. Randomized the MNIST dataset by creating sequences of random digit from 1-5
    a. Trim 4 pixels off the right and left edge for more readable and compact images
    b. Join images filling empty space with zeros when necessary to maintain digit aspect ratio
    c. Resize into 32x32 images
    d. Create corresponding label dataset simultaneously
    e. Create 10k samples of each digit range for a total of 50k

3. Generate datasets with bounding boxes from SVHN dataset
    a. Expand bounding box width and height by 15% in each direction
    b. Account for outside bounding boxes of non-digits in samples where there is less than 5 digits
    c. Change from 3 dimensional RGB to 1 dimensional grayscale
    d. Resize into 32x32 images
    e. Create new training and valid datasets by mixing original training and valid datasets
4. Combine and randomized order of synthetic MNIST and SVHN datasets

3.2 Implementation

The steps to implement the digit recognition and bounding box CNN are as follows:

1. Load the training, validation and testing sets into memory
2. Display a random image from the training class to confirm images were loaded properly
3. Create methods for the input, convolutional, pooling, fully connected, and output layers
4. Determine the depths of the filters and the strides of the convolutional and pool layers
5. Create a convolutional model
6. Build the neural network, this includes:
    a. Inputs
    b. Outputs
    c. Loss
    d. Accuracy
    e. Learning rate and optimizer
    f. Training epochs
    g. Training batch size
    h. Training dropout layer keep probability
7. Train the model
    a. Compile training and validation loss and accuracy at predetermined intervals
    b. Determine accuracy of testing dataset at the end of training
    c. Save the model

3.3 Refinement

Steps were taken to refine the algorithm and techniques are listed below:

- Reducing the images to 32x32 from 64x64 sped up the computational process exponentially as well as increased the accuracy by 1.6%
- Training with an exponentially decaying learning rate rather than the default learning rate increased testing accuracy by 3.6%
- Increasing the batch size from 128 to 256 samples increased performance by 0.7% with a negligible effect on the computational speed.
- Increasing the dropout rate from 0.5 to 0.9375 increased the testing accuracy by 2.2%

- Lowering the dropout rate back down to 0.7 increased the performance by an additional 0.9% (This action was not reflected in the image recognition CNN, as it caused the accuracy of the bounding box predictor to behave erratically)

4.0 RESULT

4.1 Model Evaluation and Validation

The list below details the parameters used in the CNN's final architecture shown in *figure 4.* A validation set was used to determine the model's accuracy and loss during training, and a separate test set was used after training concluded to verify the validation results.

- The images received from the combined MNIST/SVHN dataset were 32x32x1
- The first convolutional layer filter was size 3x3x16
- The second convolution layers filter was size 3x3x32
- The third convolutional layer filter was size 3x3x64
- All convolutional layers used 'SAME' padding with a stride of 1
- All convolutional layers were followed by a x(0, max) relu activation function and a pooling layer with 2x2 window, SAME padding, and stride of 2
- The first fully connected layer reduced the samples down to 128
- The second fully connected layer reduced the samples down to 64
- A dropout layer with a 0.7 keep probability was included after the second fully connected layer in the image recognition CNN
- A dropout layer with a 0.9375 keep probability was included after the second fully connected layer in the bounding box CNN
- The output layer finally reduced the samples down to 11 in preparation for classification
- A sparse softmax classifier was used to determine the loss in the image recognition network since the labels were not one-hot encoded
- An AdagradOptimer was used as the optimization algorithm as it was designed to be used with sparse datasets
- An exponentially decaying learning rate was used with a learning rate of 0.5 decaying after 10000 steps at a rate of 0.95
- The training ran for 90k epochs
- The training batch size was 256 samples

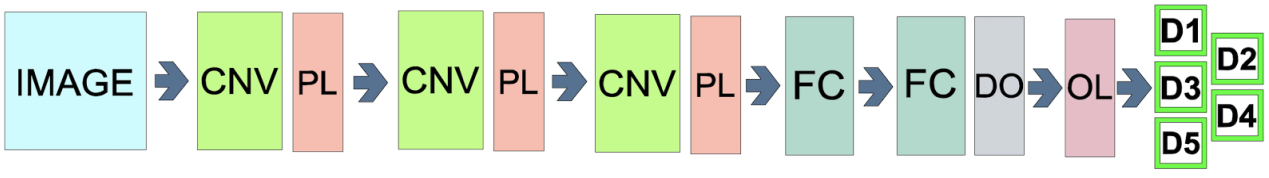


Fig 4. Final architecture of the Convolutional Neural Network

4.2 Justification

While the project's testing accuracy was only over slightly over the benchmark at 95.1% once testing was concluded as seen in *figure 5*, the model showed promise given the variability amongst the samples from the SVHN and MNIST datasets. The bounding box accuracy required a substantial amount of parameter tweaking to get above the the threshold of 85%, but it climbed to as high as 90% by the final iteration. I believe this number is artificially high due to an easy-to-read dataset. The validation set never got higher than 65%, which is most likely due to the arrangement of the housing numbers themselves. As can be seen in *figure 5* below in the freeform visualization, numbers with tight spacing, over exaggerated calligraphy, or something as simple as an random color can confuse the bounding box prediction by assuming there are two zeros stacked on top of one another.

The results from the model are robust enough to solve the problem defined in section 1.2 since the model itself was trained on two dataset whose digits are comprised of two dissimilar datasets which met expectation. However, if this model was to be used in any sort of commercial or professional manner, performance of the model would have to be increased substantially as human recognition is slightly above 98% recognition in regards to image recognition. The digit bounding box prediction performed well enough to reach the desired benchmark in the testing set, but the model is obviously more suited for image recognition than it is for bounding box prediction. As can be observed in *figure 7* below, specific images with minimal background noise perform much better than the latter.

The model lives up the solution's expectations regarding image recognition and bounding box prediction, but a more powerful computer capable of handling a CNN with additional layers, increased depths, and longer run times could elevate the model from just meeting expectations to a fully reliable application.

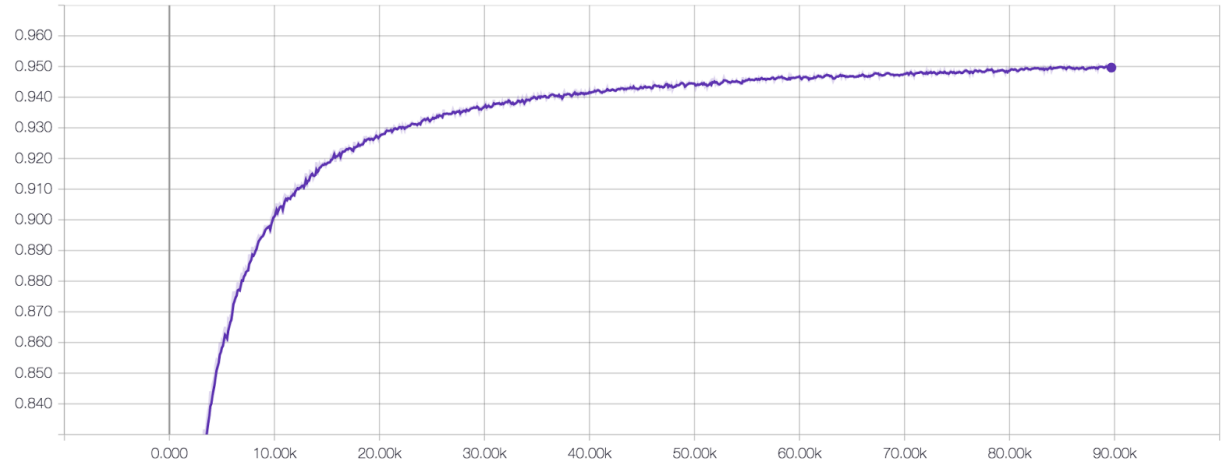
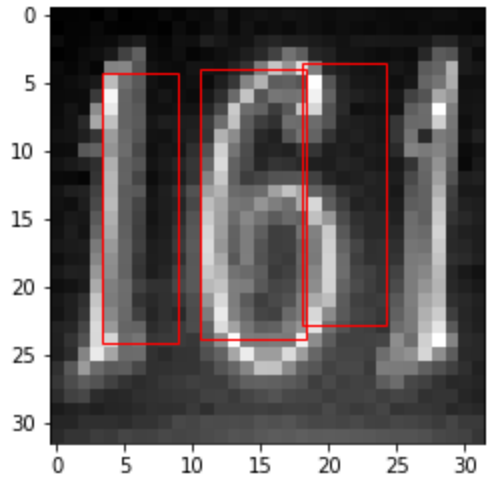


Fig 5. Digit recognition accuracy plot of validation dataset over 90k iterations

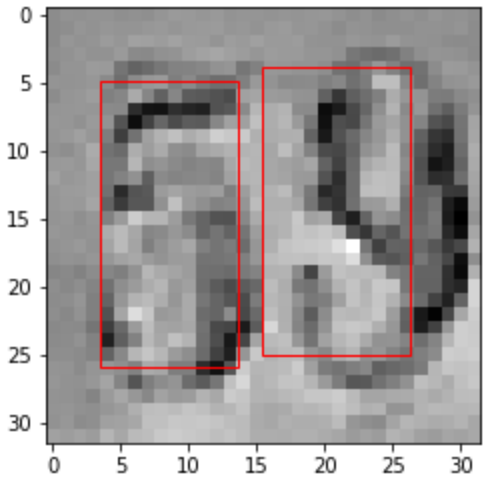
5.0 CONCLUSION

5.1 Free-Form Visualization

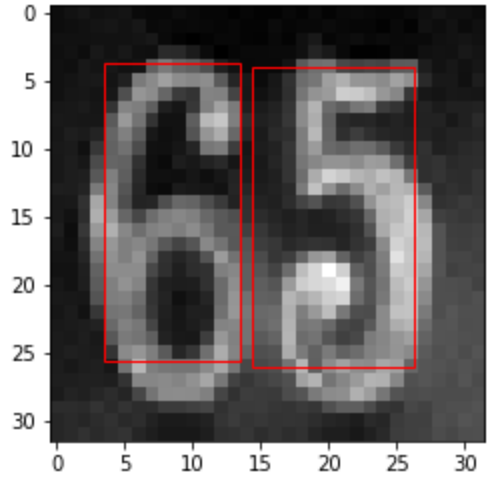
The free-form images are newly captured images that were reduced from a 3 dimensional RGB vector down to a 1 dimensional grayscale format for consistency with the MNIST/SVHN combine dataset that trained the model. Not all of the images were housing numbers in order to test how well the bounding box could perform on outliers in the dataset.



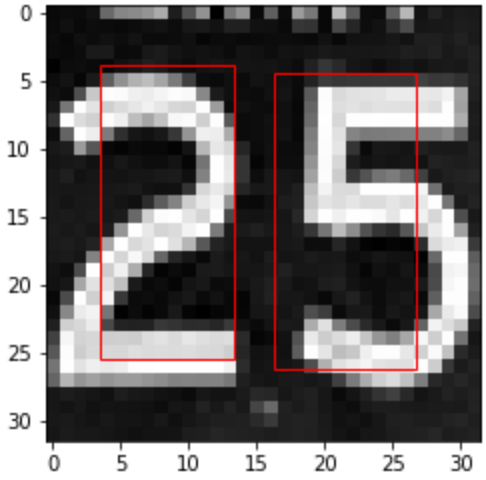
(a)



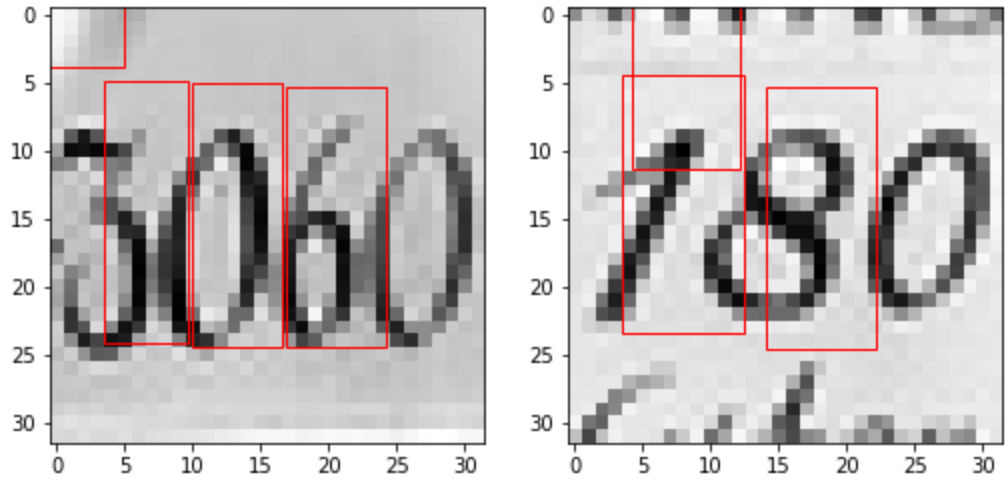
(b)



(c)



(d)



(e) (f)

Fig 6. (a) - (c) are house numbers, (d) is a speed limit sign, (e) - (f) are hand drawn signs

5.2 Reflection

1. Proposed a project that was relevant to deep learning, and that I had not seen attempted before
2. Suggested benchmarks based on previous projects that used data similar to my mine
3. Created a multi-digit dataset from the single digit MNIST dataset
4. Preprocessed the MNIST and SVHN dataset in one master set
5. Experimented with different CNN architectures using portions of the data to test the builds and gain insight into what results I could expect when using the entire combined dataset in regards to digit recognition
6. Repeated step 4 this time in regards to bounding box prediction
7. Finalized the CNN architecture and began image recognition and bounding box training, tweaking the network's parameter through several iterations in order to find a setup that would well with both types of detection (the only difference between the two was the dropout ratio)
8. Once the image recognition benchmark was achieved, a CNN with matching architecture was trained on the SVHN dataset with ground truth bounding boxes around each digit

Building the randomized MNIST dataset was a small project in and of itself. Learning how to manipulate, bind, and resize images while maintaining their original aspect ratio and legibility consumed a fair amount of time. I had a similar experience altering the size and images of the boxes in the SVHN dataset.

Tweaking the network's parameter to improve the bounding box recognition was the most frustrating part of the project. A small change in the dropout layers or batch size could dramatically affect the loss and accuracy of the predictions. After observing how the predictor performed on randomly captured images it became clear how temperamental predicting bounding boxes from ground truth images could be.

Getting to really understand the ins-and-outs of tensorflow was also a stressful, but enjoyable part of the project. Each time I needed to implement something new, such as saving statistics to run in tensorboard, it seemed inevitable that something in my system wouldn't quite work the way it was suppose to, which led to a lot of unintended learning from message boards, quora, and stackoverflow.
This entire project was really a great learning experience because it expanded on all the deep learning concepts taught in the Udacity's course. This forced me to dig deeper into the core concepts of deep learning and further my understanding of the mathematics and theorems in order to explain what was happening concise language.

4.3 Improvement

There is no doubt that having a more powerful machine, or the ability to run the project using a GPU could improve the performance of the network. I ran into problems trying to increase my batch size over 256, adding additional convolutional layers, and training for additional epochs all of which have been proven to increase performance. The bounding box accuracy could have benefited from a deeper network and larger dataset. Perhaps higher resolution images and longer run time could contribute as well, although I'm uncertain how much these actions would help prediction accuracy based on the results obtained from this study.