Multi-Digit Recognizer on The Street View House Numbers (SVHN) Dataset

Group 3: Sahil Bambulkar, Rami Elzibawi and Jacob White

Abstract

With the prevalence of image and video data on the internet, image classification and detection is a very relevant problem. A subset of this problem involves character recognition, and while this has been largely solved for documents, this is a very limited domain and so the larger problem of detecting numbers still remains to be solved. This project will aim to build off of Goodfellow et al.'s paper to use a Convolutional Neural Network (CNN) model to predict a sequence of multi-digit numbers from street signs and house numbers. The data comes from the The Street View House Number (SVHN) Dataset - Format 1 which contains 150,000 labeled sample images. While the problem outlined by the paper aims to train the CNN for numbers up to 5 digits, our approach will branch slightly. This data was used to train the CNN for numbers up to 4 digits. Another difference from the paper was in the calculation of accuracy, as the original paper does not consider partial accuracy and only looks at complete accuracy of a number (all valid digits and sequence length). Both of these were changes inherited from our branching of a previous implementation of this paper by Georgia Institute of Technology scholar Binu Enchakalody. Due to limitations in time, we slightly branched our implementation off of his but relied on the same preprocessing, CNN architecture and detection. Some preprocessing of the data done such as using bounding box dimensions to crop a 30% padded region and resizing each image to 48 x 48 pixels in both BGR and Grayscale. The training set had 150,000 samples (including 30,000 true negatives) and the test set had 13,000 samples. We also used a CNN architecture, using Adam as our optimizer, a batch size of 64, 25 epochs, Sparse Categorical Cross Entropy as our loss function and with an input of 48 x 48 x 3. Our implementation calculates a sequence accuracy metric that takes into consideration partial accuracy for instances when full numbers were not detected. We achieved Training Loss of 1.536, Test Loss of 2.5314, Validation Loss of 1.567 Training Sequence Accuracy of 82.46% Validation Sequence Accuracy of 82.22% and Test Sequence Accuracy of 86.10%.

Introduction

The problem we are working on is correctly classifying multi-digit numbers in street level photographs coming from house numbers or Street View House Numbers (SVHN). In this case, a multi-digit number is a sequence of consecutive integer digits from 0-9. When determining accuracy, we will not only be concerned with the recognizer's ability to accurately predict every digit, but also partial accuracy of some of the digits. As house numbers rarely go above lengths of 5 digits. The dataset we will be working on reflects this with very few training images with more than 5 or more digits. So the sequences can be considered of bounded length and the model should not return output in those cases, and in our case, omitted from the sample data.

Similar problems have been solved, such as optical character recognition (OCR) on constrained domains like document processing as they have many working solutions that deliver predictions with human comparable accuracy. On the other hand, arbitrary multi-character text recognition in disparate photographs has still not been mastered. The issues arise because of the variability of fonts, colors, styles, orientations and character arrangements. As well environmental factors like lighting, shadows, and resolution, motion and focus-blur.

Why it is Important

A multi-digit number recognition in street view images is extremely useful in our current world of digital cartography with the widespread use of online mapping applications like Google Maps and Apple Maps. With global maps from every nation on earth, the ability to set an exact address from an image's pixels, combined with the geographic location of each photograph, could be extremely valuable in pinpointing the addresses of collected images.

Another application is in CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart). CAPTCHAs are challenges whose aim is to limit access to non-human users by administering tests easily passable by humans but difficult for computers. One common CAPTCHA test is a series of malformed or slanted numbers and a text box asking users to correctly identify the number. If the multi-digit recognizer is able ability to correctly identify these numbers and bypass this, at this point prevalent, internet security measure it is of concern and should motivate the updating of these tests.

Brief Overview of Results

After preprocessing, training our Convolutional Neural Network (CNN) model and applying detection to new data, we achieved Training Loss of 1.536, Test Loss of 2.5314, Validation Loss of 1.567 Training Sequence Accuracy of 82.46% Validation Sequence Accuracy of 82.22% and Test Sequence Accuracy of 86.10%. The method to achieve these results, an analysis on these results and a comparison of these results will be explored in this report.

Limitations

Due to changes in the TensorFlow library almost every code implementation of the Goodfellow et al.'s paper had been outdated and needed small to substantial restructuring to run. Another issue was acquiring the hardware necessary to train the model, the paper implementation used a distributed DisetBelief implementation running on over 100 computers over six days and most amateur implementations ran the training over ten hours on powerful GPUs. Using Google Collab, we had our training take up to twenty hours and often run over its memory allocation and crash. These two issues compounded, as many times we would find out a certain function call was outdated only after a long training period meaning wasted productivity. Due to these limitations, we did not create this Convolutional Neural Network (CNN) implementation from scratch, rather we altered version of a previously made implementation made by Georgia Institute of Technology scholar Binu Enchakalody. We made small changes to this code such as adjusting hyperparameters (epochs and batch size), fixed code issues and used it to train a model and used it to test unseen data.

Related Work

Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks - Goodfellow et al.

The primary work of our project was Goodfellow et al.'s paper on Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks. This paper presented a solution to solving the problem of multi-digit number recognition from street view images. The paper proposed a unified approach to operate directly on the image pixels for the localization, segmentation and recognition steps. They employed the DistBelief implementation of neural networks, referenced in Dean et al.'s 2012 paper, to train distributed neural networks on images. On the SVHN dataset they were able to achieve 96% accuracy in recognizing complete street numbers.

Our work was directly drawn from this paper with our CNN model architecture being inspired from this, with a few changes that will be detailed below. Their best performing architecture had eight convolutional hidden layers, one locally connected hidden layer, and two densely connected hidden layers. Besides the first hidden

layer which contained hideout units, the rest contained rectifier units. At each spatial location in each layer the number of units were 48, 64, 128, 160 for the first four layers and 192 for all other local connected layers. The fully connected layer contained 3,072 and the convolutional layer included max pooling with a window size of 2x2 and subtractive normalization. The convolution kernels were 5 x 5 and dropout was applied to all hidden layers excluding the input.

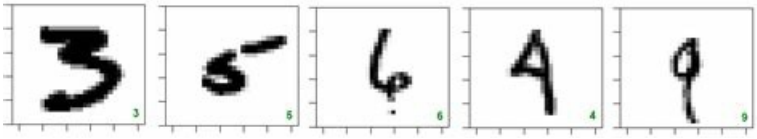
The biggest limitation from mimicing the results of the model was the lack of resources, as they were able to train their model on a DistBelief distributed implementation for 6 days with 10 replicas. We did not have access to those kind of computing resources and thus had to train our model with far less. Another factor we could not truly replicate was their exact preprocessing and detection as the paper do not go into implementation details and something our project struggled with.

Resources

These other learning resources were also useful while completing this project

- Udemy Course: Deep Learning Convolutional Neural Networks in Python by Matthew Gregg
- MIT 6.S191: Convolutional Neural Networks Lecture
- Stanford Lecture Series on Convolutional Neural Networks for Visual Recognition
- Evolution Of Object Detection Networks Course

Data



The most elementary version of digit recognizing problem is classifying the handwritten digits in the Modified National Institute of Standards and Technology (MINHST) dataset. The MINHST handwritten dataset found at http://yann.lecun.com/exdb/mnist/ consists of 60,000 black and white, centered handwritten digits ranging from 0-9 within 28 pixel by 28 pixel images. This is considered a relatively simple and straightforward image classification problem because the handwritten numbers are black and white, centered, straight and not cluttered with multiple digits.

A more advanced version of this problem was introduced with the The Street View House Numbers (SVHN) Dataset found at http://ufldl.stanford.edu/housenumbers which came from Stanford University. Images from the SVHN dataset are street sign pictures taken from Google Street View images. The SVHN dataset consists of 73257 digits for training and 26032 digits for testing as well as 531131 for extra training. *SVHN - Format 2 Dataset* consists of 32 by 32 pixel fully cropped individual (MNIST) digits. This dataset contains cropped, centered and cleaned and labeled images from street signs with digits ranging from 0 - 9. This problem is very similar to the previous dataset, but is slightly more challenging due to the addition of colors and fonts. Images also contain distracting digits that add some issues as well.

Finally we come to the dataset that was used for this problem. An undertanding of the previous comparatively easier datasets provides good background on why this dataset was chosen for this problem. The *SVHN - Format 1 Dataset* contains multi-digit labeled street images, in color, with numbers with 0 to 5 length whose individual digits ranged in value from 0 to 9. As the two preceding examples demonstrated, this is a substantially harder problem for a number of reasons. To begin with, the multiple digits per number within the image adds exponentially more possible classes as previously there were only 0 - 9 with a single digit context. Additionally, the images are not centered, each can be rotated, have blurring and have lighting differences. Each image is also of variable size. For our purposes, approximately 150,000 images were used for the training set which includes 30,000 true negatives. The test set contains 13,000 samples and a validation set is randomly chosen from 10% of the training set.

Preprocessing

Image Normalization

For the reasons outlined, a degree of preprocessing was required to be applied to make these images suitable for best performance within the CNN. For image normalization, mean subtraction was done for all training and test samples followed by feature normalization using the mean and standard deviation. This occurs as the mean for each feature is subtracted from the respective set and then divided by the standard deviation. This applies, with the same mean and standard deviation, for the training, test and validation set.

General Data Preparation and Premade Bounding Boxes

The SVHN - Format 1 Dataset provides bounding box information for each digit of its sample files. Before inputting into the preprocessing, for ease of use this data was combined into one file. This was done as the test, training and validation data were combined with each of their respective images, bounding box information and label information into an easily accessible file rather than load and referenced each of them separately. Also of note, bounding box dimensions were used to crop a 30% padded region around each box. This is also applies to post-detection images.

Additional Data Preparation includes:

- Resized to 48 x 48 pixel BGR and grayscale (although our model only used BGR)
- Binary Digit classifer was created to detect if it was a digit or not for detection purposes
- 5+ digits were omitted for lack of data and becasue they rarely occur in real life.

Detection algorithm

Premade bounding boxes are obviously not available for non-SVHN input data and thus we also include scaling, localization and sliding windows. localization is done as Image gradients and magnitude are used to create a binary mask which ignores any parts of the image without major gradient changes and avoids image elements above/below a certain size. Scaling occurs as the binary image / image is scaled ten levels by 80% each time (compared to the previous image). A 48 x 48 sliding window runs through the sample image at steps of up to 5 and ignores windows without relevant gradients. The classifier additionally removes window candidates with high likelihood of not being a digit. A final check is done for the overall confidence of it being a digit. If this is less than 70%, it is again ignored. If all of these requirements are satisifed, a local bounding box (along with liklihood) are saved.

The preceding process of locazliaton and sliding windows often causes multiple hits so a probability mask is created using the saved bounding boxes and their predictions. One issue, espescially when working with digits, is that it will fail when close to each other. Thus, we should use non-maxima suppression to remove invalid boxes based on overlap scores. The final rsized versions of these boxes are passed to the model and if the confidence is over 80% a bounding box and digit are drawn onto the final image.

Methods

CNN Model Explained

To solve our problem of the SVHN dataset we used a CNN (convolutional neural network). There are many reasons we used CNN for our problem, firstly CNN's take advantage of inputs being images. Images are seen as a 3D structure with those dimensions being the weight, height, and depth, depth being the RGB color channels. An advantage of having our input space being 3 dimensional is that we do not have to flatten our image pixels resulting in more information.

Secondly CNN's use shared weights and neurons property resulting in a smaller number of weights. Lastly CNNs have been proven to give the best results in terms of accuracy when it comes to image recognition problems which was perfect in our scenario because the SVHN problem is an image recognition problem.

Now that we know what the best model to use, we should explain it. CNN's input is the raw pixel values of the images. Like we mentioned before, the third dimension in CNN is the depth which were our colors, that means that grey scale images which do not have any color are 2D because they will have no depth and colored images will be 3D.

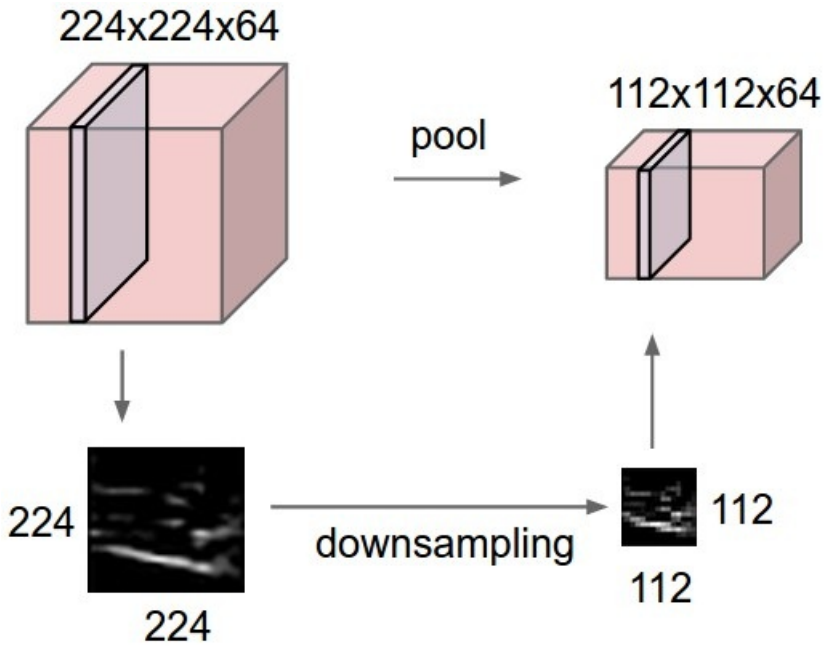
Now that we know about the input it takes we can talk about the layers. CNNs have four types of layers. Convolutional layers are the building blocks of CNNs. A convolution is an application of a filter to an input that results in an activation, repeating the application of the same filter will result in a feature map which summarizes the presence of detected features in that input.

Convolutional layer

Convolutional layers are the building blocks of CNNs. A convolution is an application of a filter to an input that results in an activation, repeating the application of the same filter will result in a feature map which summarizes the presence of detected features in that input.

Pooling layer

The function of the pooling layer is to reduce spatial size inturn reducing the amount of parameters and computation on the machine or network, it accomplishes this using the MAX function. This layer operates on each feature map independently, It is used for dimension reduction. This layer operates on depths independently so it takes the depths slice by slice. The figure below illustrates this very well

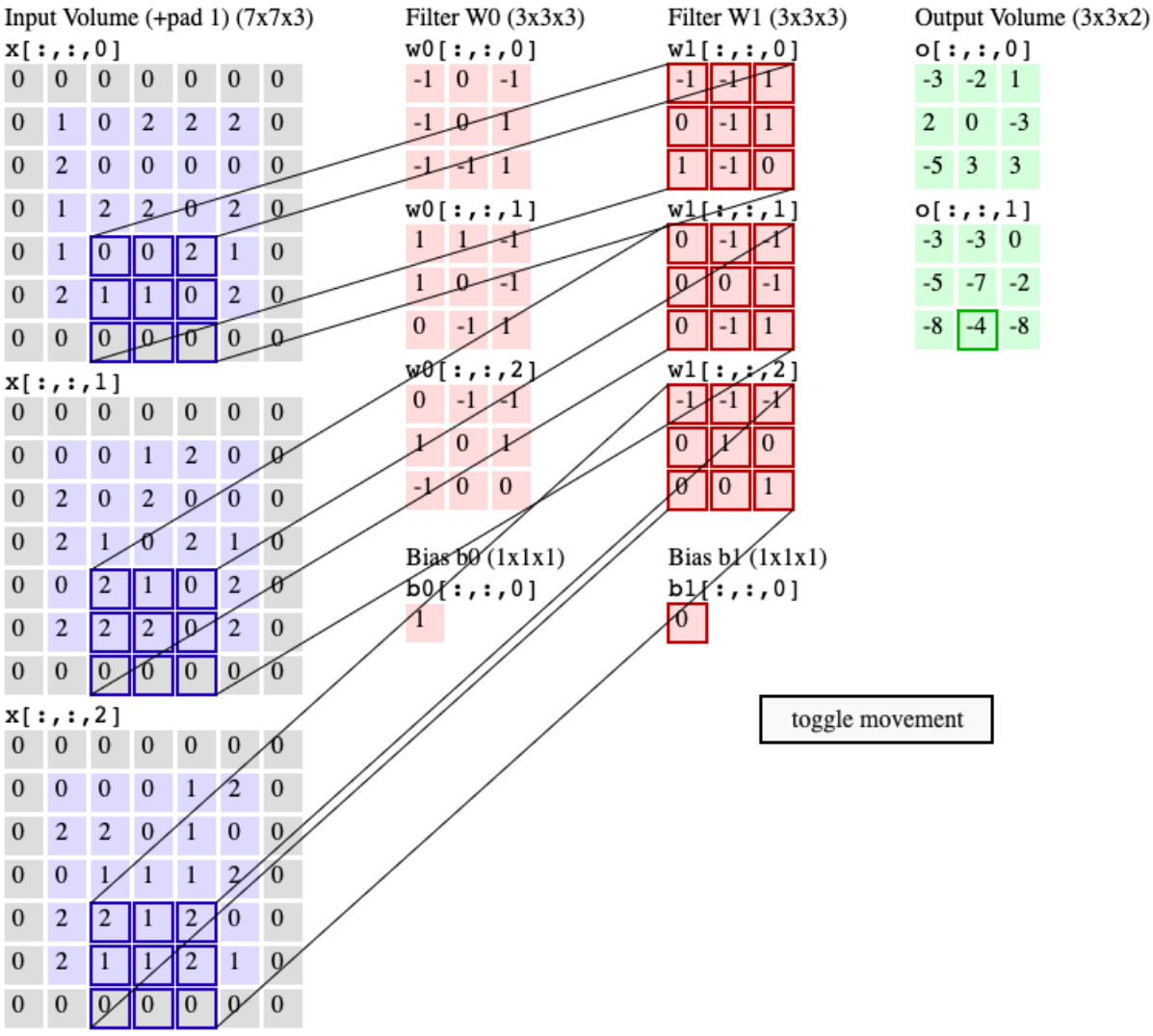


Activation layer

The activation function is a node that is put at the end of or in between Neural Networks. They help to decide if the neuron would fire or not. In most cases and in ours we use ReLu (the rectified linear activation function). This is a piecewise linear function that will output the input directly if it is positive and if it is not it will instead output 0, the purpose of using this layer is to increase the non linearity in our images.

Fully Connected Layer

The final layer for CNN's is the fully connected layer, this layer is simply feedforward neural networks. The input to these networks is the output from the final pooling or convolutional layer which is then flattened and fed into the fully connected layer, after passing through the fully connected layers, the final layer uses softmax activation function which we use to get the probabilities of the input belonging to a certain class which is the goal of our CNN.



The figure above is a screenshot of a running demo of a CONV layer taken from the Stanford University Convolutional Networks courses page, it will help us better describe filters.

While using CNNs multiple filters are taken to slice through an image and map them one by one in order to learn different portions of the image. That's why the figure above is so important because it helps us visualize this. Its as if we are scanning our input volume reading in all the regions.

Our layers and filters

For layer 1 in our designed model we have 2 x (16 filters 3x3), we can easily break this down by looking at the picture we have provided above. We can think of the number outside the parentheses as the amount of filters that are being used for this layer in our current layer , layer 1 we have a two in that place so it's actually exactly like in our figure above we would have filters W0, and W1 as well but if we take another layer in our model for example layer 4 instead of a 2 we have a 3 so we would instead have 3 filters W0, W1, and W2. The output volume is actually the dot product of all these filters so for layer 4 it would be the dot product of W0,W1,and W2.

Filters Per Layer

- Layer's 1,2,3,6,8 = 2 filters
- Layer's 4,5,7 = 3 filters

Now that we understand what the first digit represents we can now move into what's in the parentheses. If you look at the figure again you can see that we have a column for the filters and in that column we have a label W0[: , : , 0] and it goes until we have W0[: , : , 2] this last number the 2 represents our first number in the parentheses so in our situation for layer 1 we have 16 filters so we would run from 0 to 16, so unlike the figure above we would only have 2 rows for our filters we would instead have 16. In our model we used many different filter sizes for our layers. I believe this is our depth.

Depth of filters

- Layer 1 = 16
- Layer 2 = 32
- Layer 3 = 48
- Layer 4 = 64
- Layer 5 = 128
- Layer's 6,7 = 256
- Layer 8 = 512

We have almost completely broken down filters and convolution layer operations, the last thing that we have to talk about is the filter sizes. In our figure we have a filter size of 3 by 3 meaning we have 3 rows and 3 columns, just like we have in layer 1 of our model which is also 3x3. The figure does a good job of demonstrating how the filters read in from the input volume.

Row and Column Size

- Layer 1,2,3,4 = 3x3
- Layers 5,6,7,8 = 5x5

Our CNN Model implementation

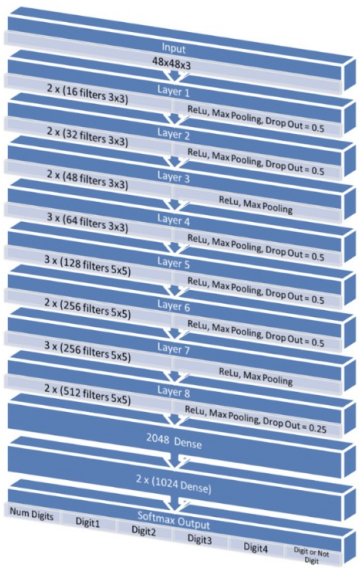


Figure 1 Designed Architecture for SVHN

This is our project's CNN model (architecture originally designed by Georgia Institute of Technology scholar Binu Enchakalody):

- Activation for all layers except the final dense layer was ReLu
- We have six softmax outputs all connected to the final layer , the first represents the number of digits in the picture, for example the number 369 would have a number of digits of 3.
- Our next four outputs would be the digits themselves since our max is 4 for the amount of digits we can identify, and finally our final output is a binary digit or not output for catching true negatives.

- When compiling our model we used sparse categorical cross entropy for our loss.

- Mini batch sizes used are 64

- Adaptive Momentum Estimation (Adam) is used in all 3 models. Adam is a very effective optimizer which is a combination of both momentum (exponential weighted average) and Root Mean Square (RMS) prop.
- The hyper parameters for learning rate (α), momentum (β1) and RMS(β2) are recommended to be used at their default values of 0.001, 0.9 and 0.999.
- Weights for each layer were randomly generated by Keras
- Overfitting was accounted for by Dropout Layers which randomly drops hidden units and measures their performance
- Early stopping was implemented if the loss does not effectively move for more than 5 epochs
- Each layer has batch normalization to help the converage (applied before each pooling layer)

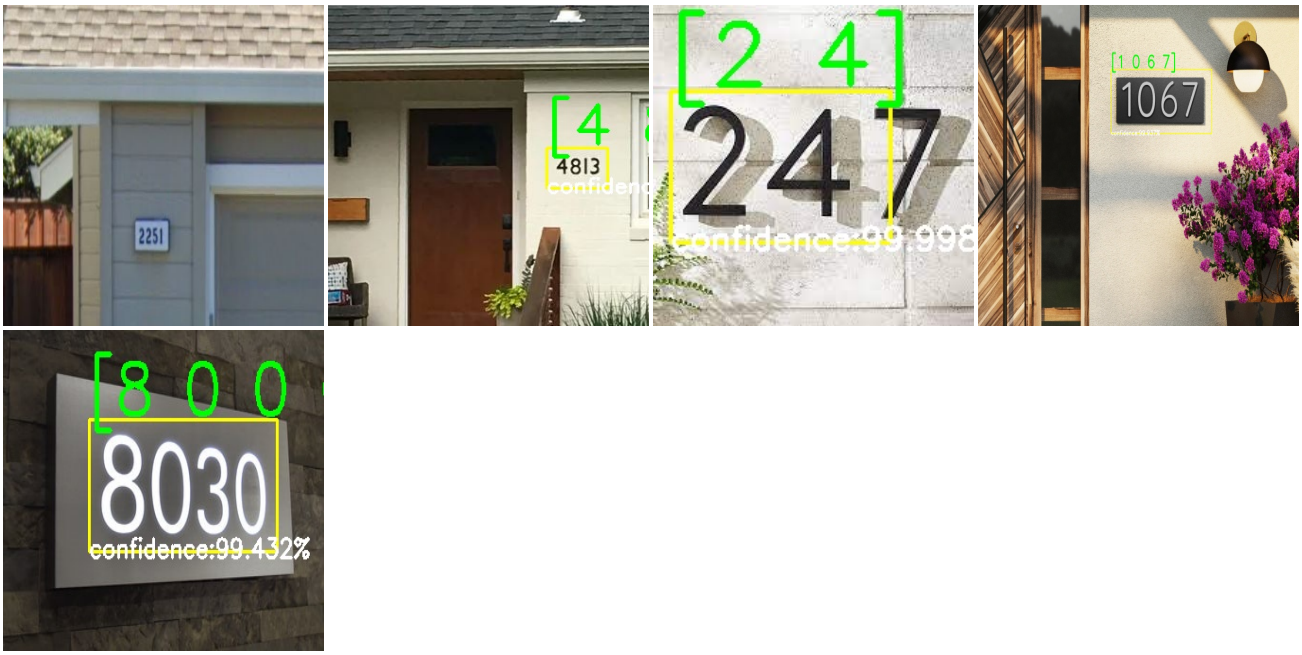
Experiments

We trained our model through 25 epochs and with a batch size of 64. We achieved Training Loss of 1.536, Test Loss of 2.5314, Validation Loss of 1.567 Training Sequence Accuracy of 82.46% Validation Sequence Accuracy of 82.22% and Test Sequence Accuracy of 86.10%. We then compared our model and metrics with a pre-trained VGG model that was run on the same data. The VGG model ended up having better metrics compared to our model by a significant margin. After we trained our model we ran a set of street view house numbers through our model and the VGG model. Both models did not correctly guess every number, and had problems in some images detecting digits as well as wrongly classifying digits.

The hardest part for the model was finding where to put the bounding box, which determines where the digits are located. This is challenging for the model when it is introduced to new images where it doesn't know where the digits are located, and has to determine where to put the bounding box. In the training data, the bounding boxes were provided within the SVHN dataset. The model can easily be confused with objects that resemble numbers and put the bounding boxes in areas with no numbers, also it can detect where the digits are but only put a bounding box around a few digits missing some.

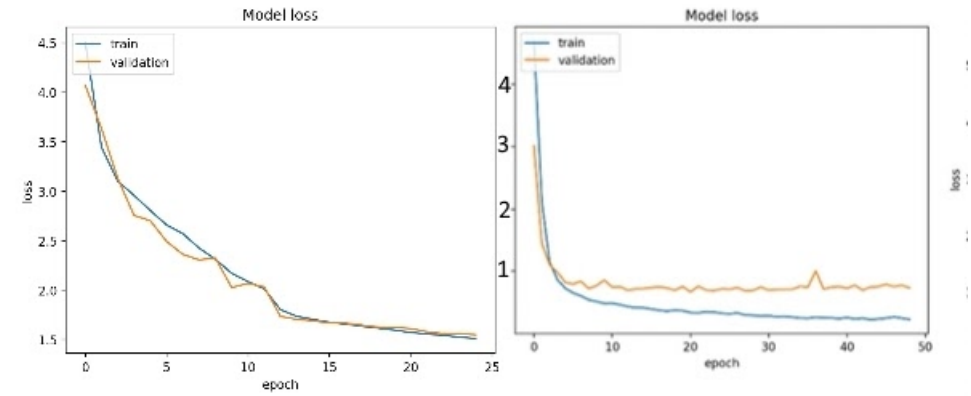
Examples from Our Model

Examples from VGG Model

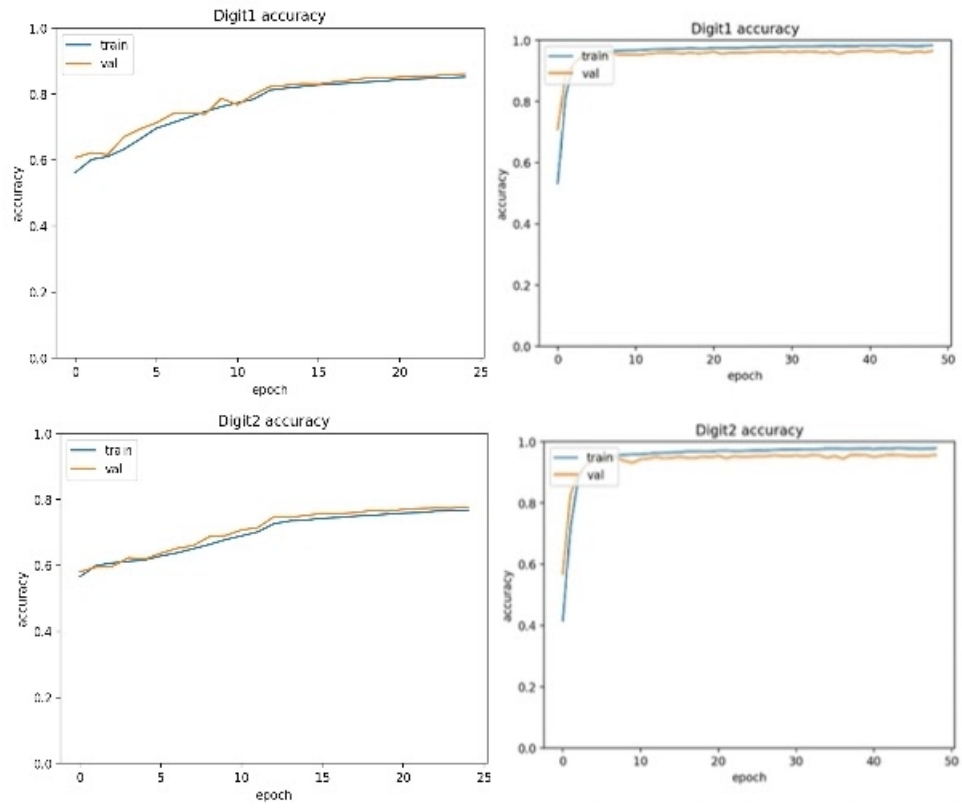


Our model ended up performing worse but still was able to classify some digits. The VGG model also made mistakes but has overall better accuracy.

Both models struggled with the localization, segmentation and detection issue in finding where to put the bounding box, this is where most of the incorrect classifications happened. When our model was able to correctly place the bounding box it was able to classify digits fairly accurately, however it performed worse in placing the bounding box compared to the VGG model. However the VGG model also wasn't perfect and made mistakes in placing the bounding box.



Our model *VGG16- Pre Trained*



From the graphs you can see that the VGG model had better accuracy and loss, although interestingly our model's validation loss was closer to the train loss compared to the VGG model. As all the labeled data had bounding boxes included, one major source of issues was training with unseen data where detection must be done from scratch. This would mean work on the detection algorithm and refinement in our our localization and segmentation.

Model	Train Loss	Test Loss	Val Loss	Train Seq. Accuracy	Val Seq. Accuracy	Test Seq. Accuracy
VGG Pre Train	0.17	0.76	0.88	96.62	87.91	91.24
Our Model	1.536	2.5314	1.567	82.46	82.22	86.10

Again seeing the loss values you see the VGG model performing better. Putting it in perspective their model was run on much more images and also included more images with vertical number orientations which both models have trouble classifying correctly. If we ran the VGG model on the same data as our model it may have performed better than ours, which we saw when we ran 15 sample images through both models. The biggest factor holding our model back was our limited compute power and time. Due to these limitations, we only ran 25 epochs compared to the VGG models 75 epochs. The VGG model was also trained on a much larger number of images, which due to our limited memory and computing power we could not replicate. The main issue for our model was placing the bounding box in new images we fed it, which due to our smaller training data set it was not as good in placing the bounding box. Although our model's performance was not as good, taking into consideration the vast difference in computing power our model still performed well in our experiments.

Conclusion

While we achieved considerable progress in our goal of developing a multi-digit recognizer on The Street View House Numbers (SVHN) Dataset, our results leave a lot to be desired. We were not able to achieve test Sequence accuracy comparable to our comparable model with our model reaching 86.10% and the VGG Pre Trained model achieving 91.24%. This was likely due to lack of training time stemming from inadequate hardware available. Another more significant issue was our failures with detection, as we found in many of our sample images that our classifier was not even being presented digits to classify. One potential mistake was to only rely on BGR images instead of running it on BW images with higher contrast. These were prepared in preprocessing but not used. Given more time and resources we would be able to further develop both the preprocessing, detection and classification to optimize this project. All in all, it was a great learning experience that introduced our group to implementing a convolutional neural network. Even more so, the fact that it was developed to tackle a still unsolved problem that has major impact on digital mapping by reliably assigning street numbers to digital street view images is inspiring. Another application for this idea would be to apply it to CAPTCHAs and see how it would perform on a dataset specifically designed to fool computers. While this first iteration would not fare well, perhaps a more refined version of this project could be able to tackle the problem and it seems like a natural extension of this project.