Machine Learning Nanodegree
Capstone Project

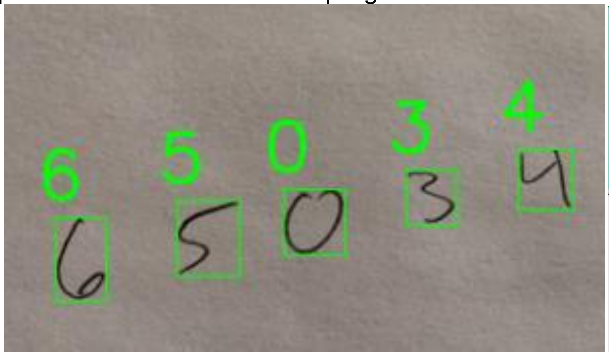
Dharmendra Kumar

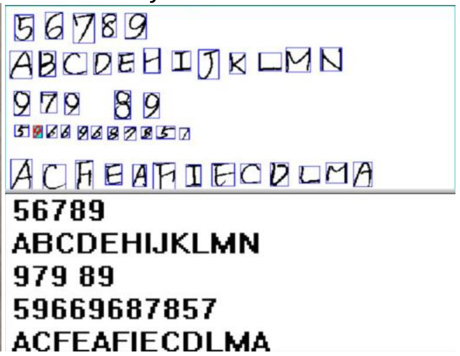
March 2017

I. Definition

Project Overview

Over the last few years, object recognition on images has piqued interest of both the leading industry and academic professionals. While some problems like single character recognition are easy to solve, recognizing a sequence of characters still are challenging. While Machine learning has been a known science since 1980s, availability of a huge amount of open data along with the use of GPUs to solve these computationally intensive problems have accelerated progress in this domain in the last 5 years.



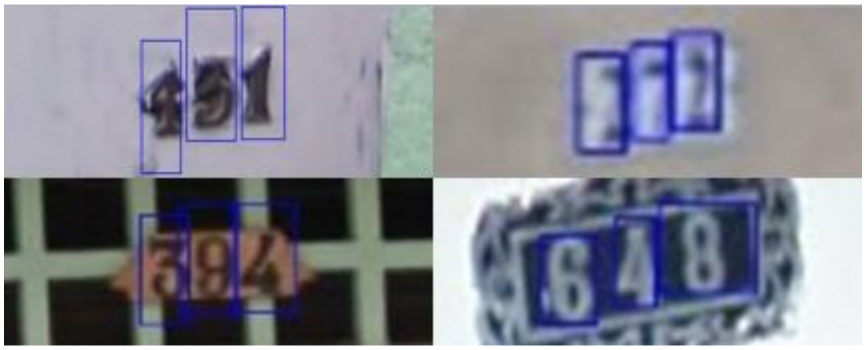


```
56789
ABCDEHIJKLMN
979 89
59669687857
ACFEAFIECDLMA
```

This project focuses on recognizing a house number, which is basically a sequence of digits. In particular, The objective here would be to develop an application, which recognizes house number from images. The application takes as input a bunch of images and provide a house number presented on each image. One of the challenges here might be that some house numbers are displayed in some specific order. For example, some might be displayed vertically or diagonally or in some other orientation. However, the goal here would be to build an application which recognizes numbers from given images with high accuracy.

Problem Statement

In this project, Street View House Numbers(SVHN)[1] dataset is used to train the application model to recognize numbers on images. SVHN is a real-world image dataset for developing machine learning and object recognition algorithms with minimal requirement on data preprocessing and formatting. The dataset is obtained from house numbers in Google Street View images. Image below represents several samples from this dataset.



Some of the images as shown above have quite bad quality, and would be challenging for even a human to recognize them accurately. Therefore, some error can be expeced from our model as well. Here the accuracy of a house number (label) would be to identity all digits in the house number is the correct order, which is a challenge as missing even a single digit from the number or its order would result in a failure.

Recognizing numbers on images can be accomplished with human help if the amount of images to recognize is not huge. However, if the amount of images is extremely large using people to solve this problem is not efficient. And this is the place when Machine Learning comes into play!

Evaluation Metric:

For this problem the accuracy is calculated by counting how many numbers were correctly recognized. Hence, all digits in the house number must match in the same order for to count as a success. Then we sum up all the successfully identified house numbers and divide it by a size of a dataset. Here is the formula:

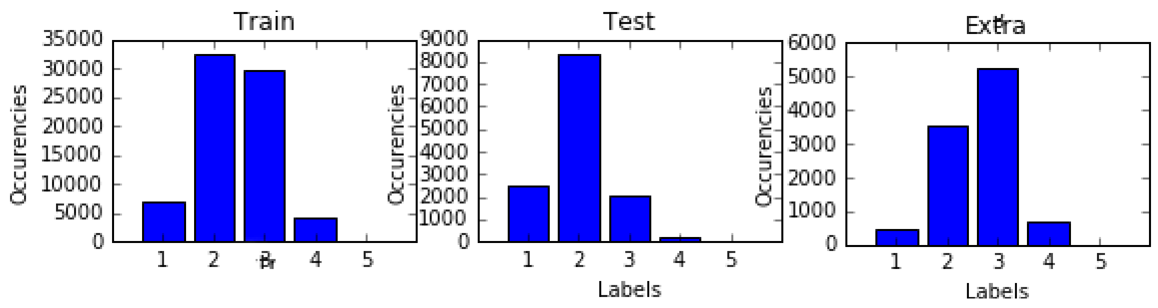
$$\Sigma(Correctly\ classified\ digits\#\ of\ digits\ in\ number)//\ \text{number of digits})/\text{number of images}$$

II. Analysis

Data Exploration

SVHN dataset consists of 3 groups. Training, testing and extra data. Training set has 33402 samples, testing set has 13068 and extra set has about 230000 samples. Extra set is used to extend our training set and perform validation during model training. There might be an infinite amount of labels, since there are no limitations about data for such an application. However, we are going to work with images which have up to five digits between 1 and 99.999 inclusive. Therefore, samples which have numbers with more than 5 digits are removed from the dataset during preprocessing. Beside the images, dataset contains also a digitStruct structure which contains some meta information about each image. This structure has same length as amount of images in a dataset (one entry per image). Here each entry has two fields: name which is a string containing the filename of the corresponding image and bbox which is a struct array that contains the position, size and label of each digit bounding box in the image. For instance, digitStruct(300).bbox(2).height gives height of the 2nd digit bounding box in the 300th image. We shall use this bounding box information to build another model to guess a bounding box around the labels.

Figure below illustrates class lengths distribution across several datasets. As we can see all datasets have Gaussian distributions. The relation of 1, 4 and 5 digits classes to the whole dataset almost equal for training, test and validation datasets. About 90% of training data has 2-3 digit labels. About 80% of test data has 2 digit labels and about 10% of tests data are numbers with 1 or 3 digits each. The majority of extra data are numbers with 2 and 3 digits in a number.



Now let’s review images sizes. First, during preprocessing we are going to crop images just near the numbers to simplify training effort for our neural network. Therefore, original image size does not matter. The table below show min, max and mean sizes of images height and width of train, validation and test datasets after they were cropped just near the numbers.

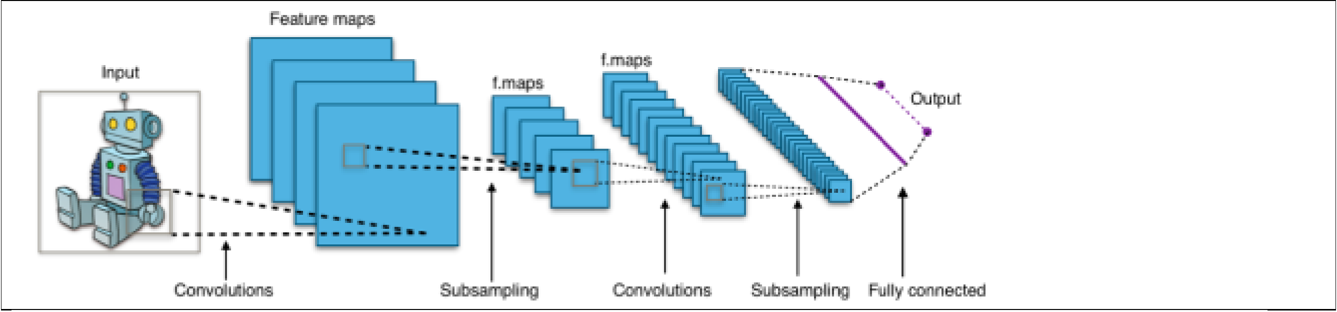
	Train	Test	Validation
Max Width	876	1083	668
Max Height	501	516	356
Min Width	25	31	22
Min Height	12	13	13
Mean Width	128.29	172.58	100.04
Mean Height	57.21	71.56	60.72

Since images sizes are different we are going to resize all of them to same size during preprocessing step which is described below.

Algorithms and Techniques

To solve this problem, we are going to use modern deep learning techniques which are provided by the Google’s TensorFlow library. In addition, a model using Keras library is built and the best model is further tuned to be finally used for our application. In the first part of a project we develop a multilayer Convolutional Neural Network which is trained to predict just a single digit.

Convolutional Neural Network (CNN, or ConvNet) is a type of feed-forward artificial neural network in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex, whose individual neurons are arranged in such a way that they respond to overlapping regions tiling the visual field. Convolutional networks were inspired by biological processes and are variations of multilayer perceptrons designed to use minimal amounts of preprocessing. They have wide applications in image and video recognition, recommender systems and natural language processing.



A ConvNet model represents a stack of various layers that transform an input to the output. The most common types of layers are:

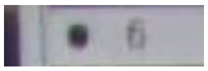
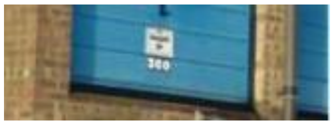
Convolutional Layer - The Conv layer is the core building block of a Convolutional Network that does most of the computational heavy lifting. The convolutional layer parameters consist of a set of learnable filters. The network will intuitively learn which filters to activate when they see some type of pattern. The input to the convolutional layer usually called input feature map and output is an output feature map.

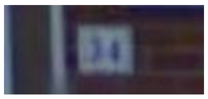
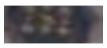
Pooling Layer - progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control over-fitting. The Pooling layer operates independently on every depth slice of the input and resizes it spatially, by using some function. In our application MAX operation will be used, as the most common one.

Fully Connected Layer - Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset. See the Neural Network section of the notes for more information.

Loss Layer - The loss layer specifies how the network training penalizes the deviation between the predicted and true labels and is normally the last layer in the network. Various loss functions appropriate for different tasks may be used there. We use Softmax loss function for predicting a single class of K mutually exclusive classes.

After basic version is designed, we tune network’s parameters trying to find configuration with lowest error rate. This network should be able to recognize numbers with up to five digits with low error rate. It will be able to take an image with house number presented and return the number which is presented on an image. We evaluate this approach on test dataset and achieve over 90% accuracy in recognizing complete street numbers as SVHN dataset contains many samples which is hard to recognize even for a human.

As you can see images above are very different. They have different size or quality or all together, they can contain some additional symbols nearby. Digits could be written diagonally. All such data is hard to recognize even for a human, so we expect that neural network may also face with some problems here. Moreover, due to a slow hardware, the neural network which we build is going to be quite simple and we don’t use all available data, since it is hard to load all it to the RAM. Therefore, our solution would target an accuracy of 80% where the numbers are correctly identified from the images from the testing set.

Benchmark

There have been a few studies performed on this problem. However, the research paper written by Ian Goodfellow, Yaroslav Bulatov, Julian Ibarz,Sacha Arnoud, Vinay Shet on Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks is widely regarded as the benchmark for this problem. Their state-of-the-art model achieved 97.84% accuracy in recognizing complete street numbers, which is simply outstanding.

III. Methodology

Data Preprocessing

1. Crop each image just near the numbers to simplified effort required for model fitting.
2. Resize cropped images to 32x32 size since after cropping all images have different sizes.
3. Convert all input images into a grayscale. This is a very common technique in image preprocessing to help classifiers as digit’s color has little or no significance in its identification.

This is achieved by multiplying red channel by a factor of 0.2989, green by 0.5870 and a blue one by 0.1140.

4. Apply local contrast normalization on images to help our optimizer converge faster and accurately.
5. Replace ‘0’ digit class from 10 to 0 to be consistent with other digits and classes. Instead, use class 10 to label not-existing digit resulting in a total of 11 classes for identified digit labels.
6. Split each sample class by digit. If number has size less than 5 digits, then add labels 10 at the end. It means that number 567 will be labeled as: [‘5’, ‘6’, ‘7’,’10’,’10’].

Implementation

Following models were built for this project (See following URL for the associated code and results)
https://github.com/dkumar95120/machine_learning_dk/blob/master/projects/digit_recognition/digit_recognition.ipynb

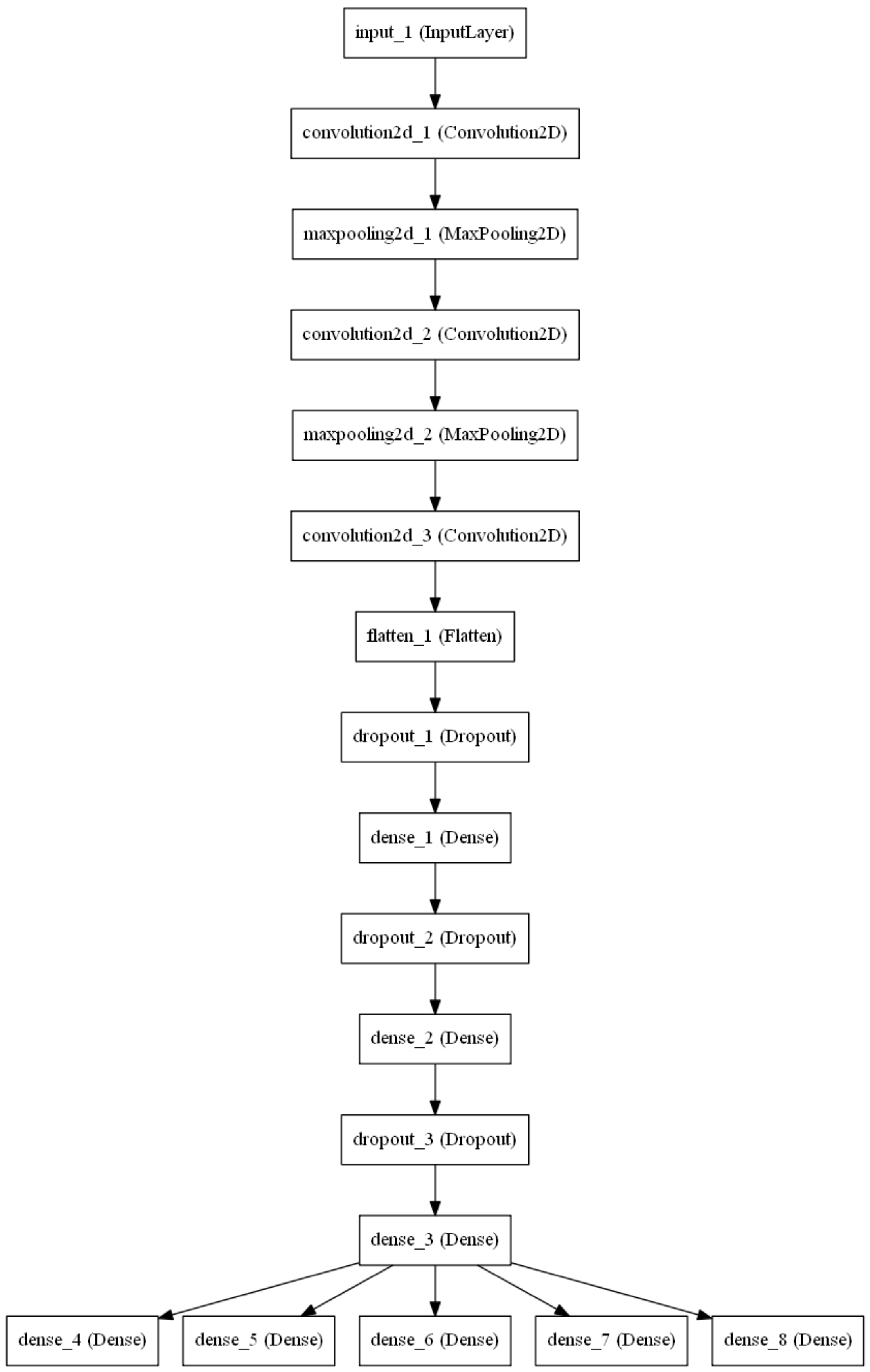
TensorFlow CNN Model

1. Convolutional layer with 8 feature maps, which with the size of 5×5 and a rectifier activation function. This is the input layer, expecting images with the structure outline above [pixels][width][height].
2. A pooling layer that takes the max called MaxPooling2D. It is configured with a pool size of 2×2.
3. Convolutional layer with 16 feature maps of size 3×3.
4. Pooling layer taking the max over 2*2 patches.
5. Convolutional layer with 32 feature maps of size 3×3.
6. Next is a layer that converts the 2D matrix data to a vector called Flatten. It allows the output to be processed by standard fully connected layers.
7. Next a fully connected layer with 1024 neurons and rectifier activation function.
8. Fully connected layer with 1024 neurons and rectifier activation.
9. Dropout layer with a probability of 20%.
10. Fully connected layer with 256 neurons and rectifier activation.
11. Dropout layer with a probability of 20%.
12. Fully connected layer with 32 neurons and rectifier activation.
13. Output layer with 11 nodes for 11 classes (including blank) with a softmax activation function to output probability-like predictions for each class.
14.

Keras CNN Model

1. Convolutional layer with 8 feature maps of size 5×5.
2. Pooling layer taking the max over 2*2 patches.
3. Convolutional layer with 16 feature maps of size 3×3.
4. Pooling layer taking the max over 2*2 patches.
5. Convolutional layer with 32 feature maps of size 3×3.
6. Flatten layer.
7. Dropout layer with a probability of 20%.
8. Fully connected layer with 1024 neurons and rectifier activation.
9. Dropout layer with a probability of 20%.
10. Fully connected layer with 256 neurons and rectifier activation.
11. Dropout layer with a probability of 20%.
12. Fully connected layer with 64 neurons and rectifier activation.
13. Output layer with 11 nodes with a softmax activation to output probability like predictions

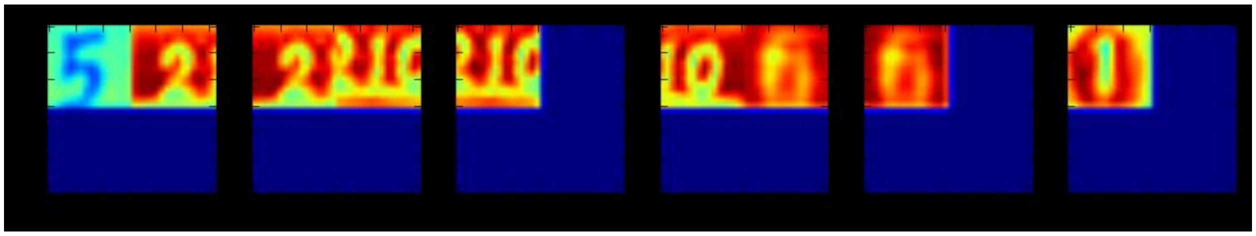
(see figure below describing the Keras model).


input_1 (InputLayer)
convolution2d_1 (Convolution2D)
maxpooling2d_1 (MaxPooling2D)
convolution2d_2 (Convolution2D)
maxpooling2d_2 (MaxPooling2D)
convolution2d_3 (Convolution2D)
flatten_1 (Flatten)
dropout_1 (Dropout)
dense_1 (Dense)
dropout_2 (Dropout)
dense_2 (Dense)
dropout_3 (Dropout)
dense_3 (Dense)
dense_4 (Dense)
dense_5 (Dense)
dense_6 (Dense)
dense_7 (Dense)
dense_8 (Dense)

IV. Results

Model Evaluation and Validation

In the first part of a project I trained each model on a synthetic data set comprised of the individual digit SVHN dataset in a 2x2 grid to keep aspect ratio of each digit intact and following the dataset preprocessing steps outline earlier (see get_image.py file for the related code). See figure below for a sample of synthetic images.



Their respective performance was as follows:

Model type	Digit Accuracy	Label Accuracy	Run Time
TensorFlow CNN	63%	23%	2225sec
Keras CNN	82%	51%	2704sec

The Keras CNN model demonstrated better performance over the TensorFlow model, and was chosen for further tuning for the remainder of the project.

Both models were also trained and tested using the real SVHN dataset that was also preprocessed following the same steps as for the synthetic data (except no image concatenation). I created load_digits_dataset() function to load either the single or multi-digit SVHN or synthetic datasets. I used 90% of the training data for training while remaining for validation. The testing data was used only to verify model accuracy to make sure no overfitting happened. The model performed quite well on the real dataset as shown in the table below:

The CNN model with TensorFlow performed much better on the real dataset than the synthetic one. This model had independent weights for each digit in the label. However, its accuracy was still quite bad (just 56%) for the label, i.e. when all the digits in the label were identified correctly in the right order. The Keras CNN model worked much better as the layers till the output layer were shared for all output digits. It had to be tuned to improve its accuracy though mostly by running the model for more epochs. The runtime of the Keras model also had a factor of 4 advantage over the TensorFlow model.

DataSet	Model type	Digit Accuracy	Label Accuracy	Run Time
Synthetic	TensorFlow CNN	63%	23%	2225sec
	Keras CNN	83%	51%	2667sec
Real SVHN	TensorFlow CNN	88%	56%	4972sec
	Keras CNN	92%	71%	1200sec

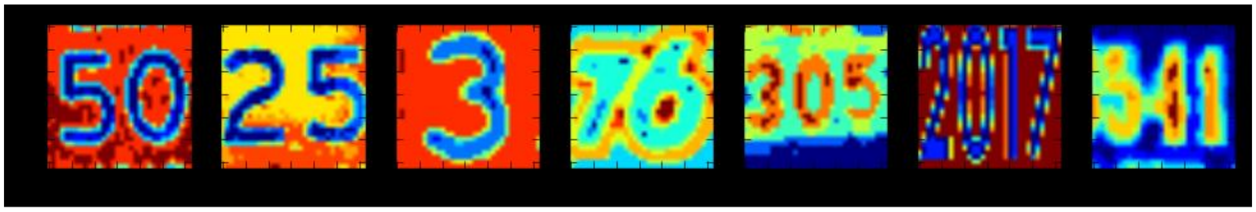
The CNN model using Keras performed much better than the TensorFlow model, and both models performed much better on the real dataset as compared to the synthetic dataset. The table above shows the accuracies of each model for synthetic and real dataset. I believe these results could be improved by increasing depth of the convolution layers. However, I was limited by the memory and cpu on my desktop as higher depth models kept crashing python on my desktop. Besides, each epoch took over 20 mins to complete when it was not crashing. Therefore, with the current limitation of my computing resources I believe I got good results with these models.

The Keras CNN model was further tuned using the extra SVHN dataset with 202353 samples with following adjustments:

1) Epochs were increased to 20
2) Batch size was increased to 128
3) Trained on 182118 samples, and validated on 20235 samples

The resulting model was able to achieve 95% accuracy with each digit and 82% accuracy for the entire label!

The model was also attempted on real images (shown below along with their prediction)



Predicted:50 25 3 76 305 2017 541

As evident from the answer above, the model was able to predict 7 out 7 numbers correctly, which is excellent for a model which was restricted due to computing resources. The model seems to have difficulty when the numbers are small as we are limited to a 32x32 image which needs to cram all digits.

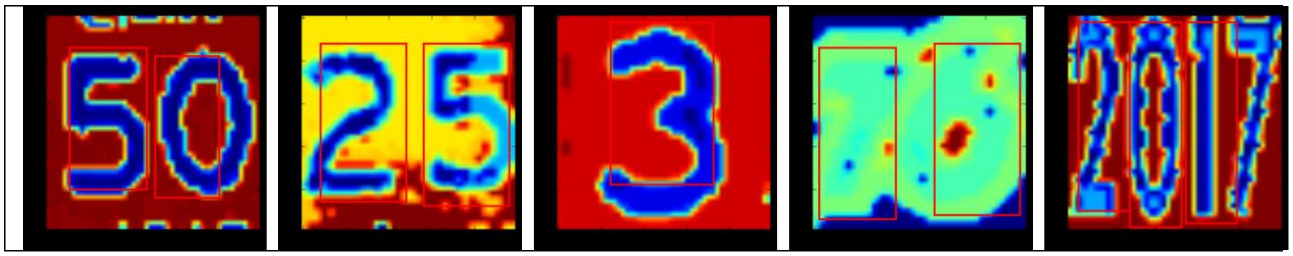
Bounding Box Prediction Model

In addition, the same model was trained on the bounding boxes provided for these images in the SVHN dataset. For this the dataset was preprocessed as follows:

1) Assumed a maximum of five bounding boxes per image (as we expect a max of 5 digits in each sample)
2) Each bounding box was extended by 10% in both x and y directions
3) Corresponding image was resized to 50x50 dimension (to save computation time)

4) Four dimensions per bounding box (x, y, w, h) were saved, hence a total of 20 output variables were prepared for each image.

This model took nearly 4 hours to regress on the extra dataset but achieved only 58% accuracy, which is largely due to small image size of 50x50. The same sample images were tested for determining the bounding boxes that are shown below:



Justification

Python Number Reading Application

An independent python application is developed leveraging the Keras CNN model developed with the aforementioned design and used to recognize digits along with bounding boxes around the number labels in any random image of any size and uses OpenCV for visualization. The sample usage of the application would be as follows:

Read_house_numbers image1 image2 …

1. First, the program loads the machine learning Keras CNN model for classification of digits

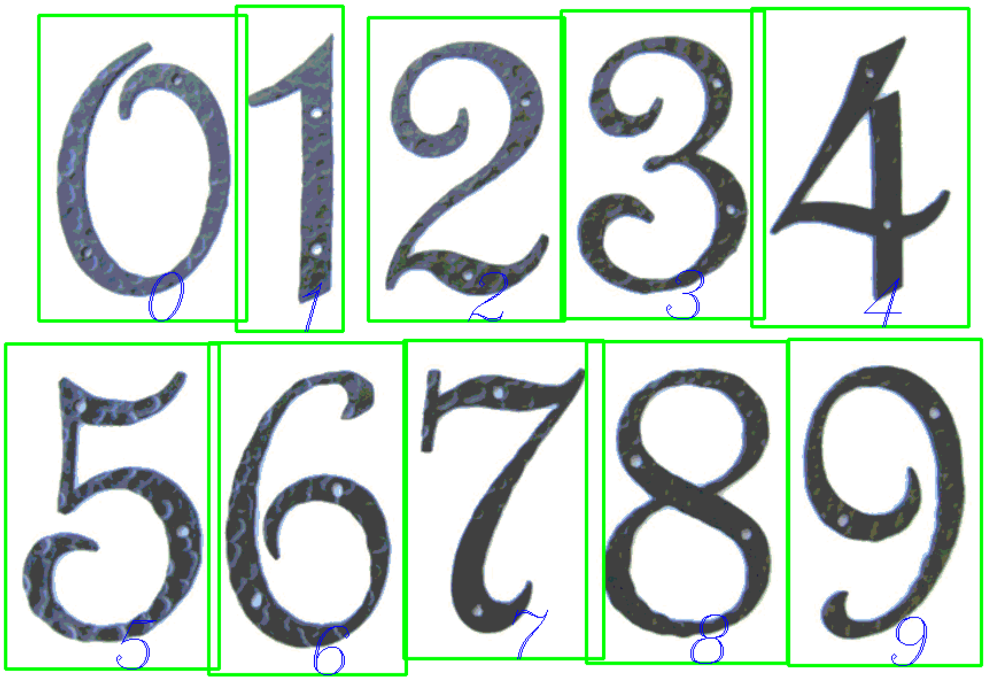
The python program reads all specified images (one at a time) and finds the bounding boxes first using the opencv findContours module. For this the image needs to be preprocessed as follows:
1. Converted to grayscale image using cvtColor module using cv2.COLOR_BGR2GRAY filter
2. Applies Gaussian filtering using opencv GaussianBlur module
3. Establishes threshold for finding contours in the image using adaptiveThreshold module
4. Finds outermost contours in the image using findContours module using computed threshold and cv2.RETR_EXTERNAL mode
5. Collects bounding boxes for each contour and saves them for further processing and look up
6. Extends bounding box dimensions to by 10% in both x and y directions
7. Adjusts bounding box dimensions so that the bounding box is inside the specified image
8. Creats a new image by capturing image contents within the bounding box
9. Resizes image to 32x32 size for classification
10. Dialates image to improve recognition of digits
11. Normalizes pixel values within [-1,1] to improve their classification
12. Stores normalized image in the array of images

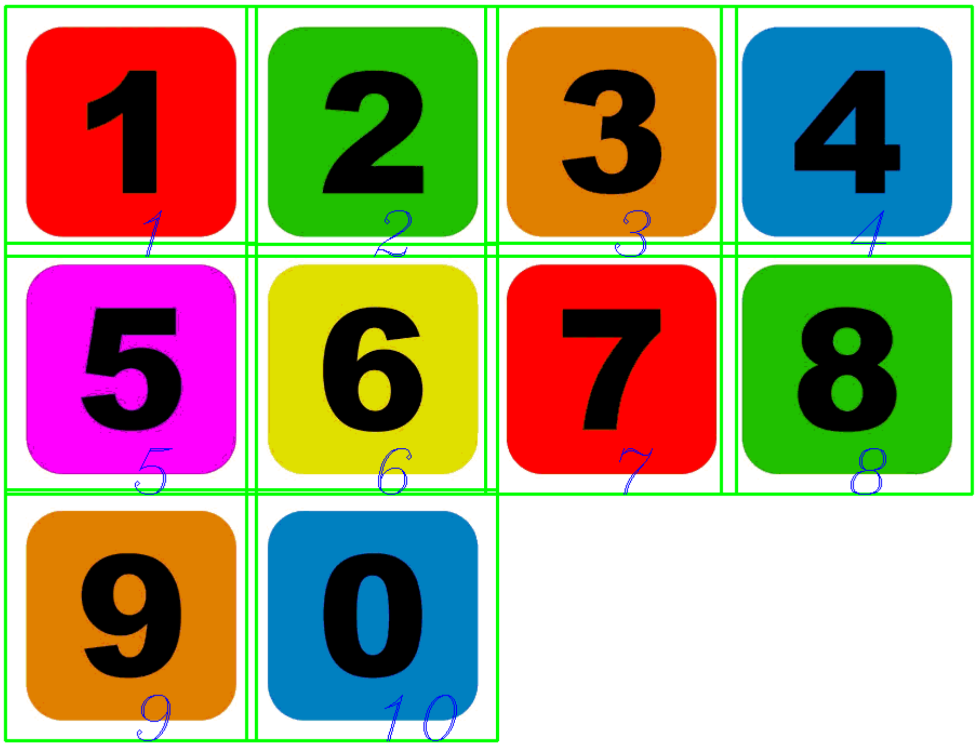
The following operations are then performed on the prepared sub-images found
1. Predicts digits using the loaded model given the preprocessed images above
2. Gathers predicted digits by performing argmax function on the predictions and taking transform

For each identified digit in the image:

1. Draws bounding box rectangle using the bounding box dimensions
2. Builds text label for each number found in the image
3. Draws text around the bounding box at its bottom middle section

Finally, it shows the resulting image as the output. A few samples of images that were attempted using this program are shown below:





V. Conclusion

In this project, a simple convolutional neural network was trained to recognize numbers on images from SVHN dataset. I employed a stricter label accuracy which requires matching all digits in the label in the given order to consider a success. Otherwise, the digit accuracy rate of this model is about 95% which is very good. However, the label accuracy was 82% which is above the 80% goal that was set for the project, and is good considering the constraints on available computing and human resources. Moreover, a lot of images are hard to recognize even for a human. Therefore, the result is quite encouraging and even exceeded my expectations.

The test accuracy and the accuracy with real image samples suggest that the model is not over fitted and robust to unseen data. The fact that all datasets have very close accuracy value means that our model is ready to deal with any values from the SVHN dataset and not only those which we have used for training.

The standalone python application was also mostly accurate to identify numbers in the random image. However, the contour finding algorithm from opencv tends to find each digit by itself, hence we are unable to test the sequence of digits with real images.

Finally, with everything said, there is clearly a lot of room for improvement. Neural Network can be expanded to use more features and layers, which could be performed with more powerful hardware with GPU that are better equipped to process matrix calculations. We really need to improve accuracy of such an application before putting them in real world applications such as in the postal services.