Classification and Detection with Convolutional Neural Networks

Sivagami Subramanian Nambi
snambi3@gatech.edu

ABSTRACT

Automatic Detection of Numbers and/or digit is a task which has shown a lot of promise in Deep Learning and Computer Vision. Convolutional Neural Networks (CNNs) are used widely in Computer Vision to perform Recognition tasks because CNNs makes use of spatial information. The goal of this project is to predict the sequence of numbers on Street View House Number (SVHN) [4] dataset using CNN.

EXISTING METHODS:

One of the first paper to address that CNNs can be used on image for classification task is "ImageNet Classification with Deep Convolutional Neural Networks " by Krizhevsky, 2012 [1] which showed that CNN can perform classification task and was better than supervised learning algorithms on ImageNet dataset.

"Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks" by Ian Goodfellow, 2014 [2] got 96.03% accuracy on SVHN dataset by using an output layer providing a conditional probabilistic model of sequence with a deeper architecture.

"Very Deep Convolutional Networks for Large-Scale Image Recognition" by Simoniyan and Zisserman, 2015 [3], introduces the CNN architecture called VGG16 (16 layers deep) and demonstrated that the representation depth is beneficial for classification accuracy and showed a positive effect on ImageNet dataset. This project is based on modifying the existing VGG16 architecture and applying it to classifying SVHN dataset [4]

IMPLEMENTATION

PREPROCESSING

Dataset Preprocessing: SVHN dataset contains

Format1: Full Numbers along with bounding box values of each digit

Format2: Cropped Digits.

Both the formats contain 73257 digits for training, 26032 digits for testing with 10 classes as output across different font styles.

I used both the images of both the formats. Format 2 contained better quality pictures, which is necessary for classifying digits in real-world images but had this shortcoming,

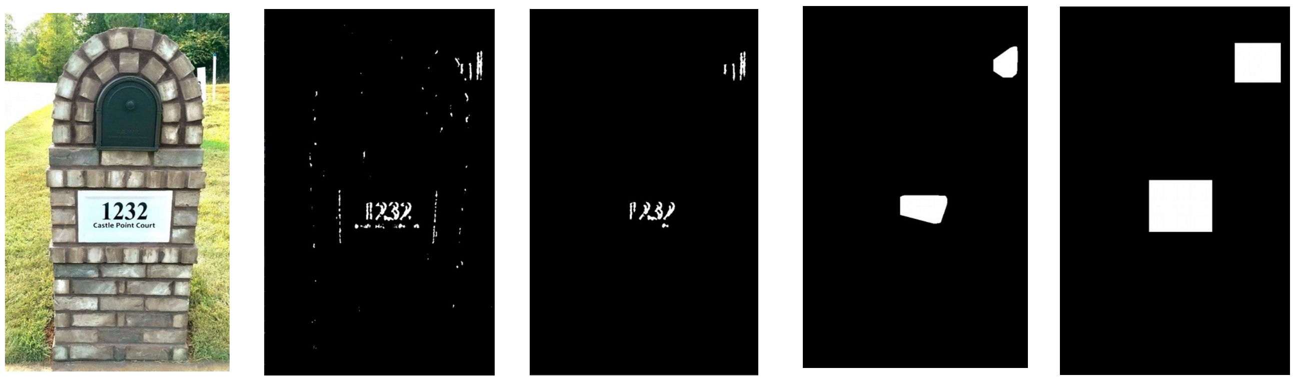
Images like these contain more than one digit but were associated with a single label. So I also used Format 1 dataset and extracted individual digits using bounding box value.

To distinguish between digit and non-digit, I had to add non-digit samples to the dataset. For this, I used the original Format 1 of SVHN and cropped 48x48 patch which is not included in the bounding box (~14k samples) as well as added randomly cropped patches from Houses-dataset[5] (~7k samples) and labelled them as 0 (non-digit). So the dataset now contains 11 output classes.

I had to resize these input data to 48x48 as this was the minimum required size of VGG16. Other preprocessing steps like RGB mean subtraction and normalization were taken care of by inbuilt function in TensorFlow.

NOISE MANAGEMENT:

The input image may contain noise, different lighting condition or any other distracting objects in addition to the house numbers. The input image is first passed on to the Gaussian filter to filter out the noise and smoothen the image. Then it is passed to Sobel operator which detect edges with a gradient above certain magnitude. These edges are then passed to Connected components for eliminating elements with a smaller number of pixels. After which, Convex hull operation is applied to the binary image to join the pixel elements which are within a radius of 50 px. Bounding filled rectangular box is found for each convex hull and the bounding box is expanded by a small percentage to provide padding.



HANDLING LOCATION INVARIANCE

Sliding window technique was employed. This involved sliding a sliding window of fixed size 48x48 across length and breadth of the image. If the current location had a binary mask whose mean is less than a certain value the current location is skipped and the window is moved to the adjacent location. If the binary mask condition is satisfied a classifier was run at the current location and label was obtained. If the label is greater than 0 (a digit) with a confidence probability of above 0.95, then the labels along with bounding box values are saved.

HANDLING SCALE AND FONT INVARIANCE

To meet scale invariance Image Pyramid was used along with sliding window. With a fixed window size, the image size was decreased so that it can detect digits of a larger scale. Image Pyramid of up to 5 levels is used, the corresponding bounding box was resized and saved if a digit is found.

This sliding window along with bounding box introduced one problem of overlapping digit bounding boxes. To solve this challenge non-maximum suppression based on IoC (Intersection over union) and score (confidence probability) of the prediction is used to choose bounding boxes over a certain threshold. This method correctly classifies the digit with one bounding box per digit, but sometimes wrong sized bounding box is selected. This can be improved.

PERFORMANCE CONSIDERATION

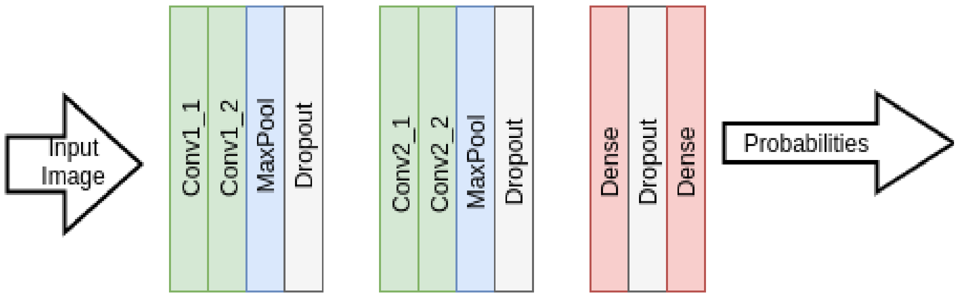
Moving the sliding window with stride 1, proved to be very expensive. So in my implementation, the sliding window moves with stride 4. This has improved the performance by (2^16) although has a risk of not detecting smaller digits. Also instead of using the classifier (forward pass) at every location of the window, using the binary mask that I have implemented above proved to be very efficient as the classifier is used only when the mean of the binary image is above a certain threshold.

CLASSIFICATION

MODEL VARIATION

I used 3 approaches for designing the classifier. First was to build my own CNN, next was to use the existing VGG16 architecture and train it from scratch. The third was to use the pre-trained VGG16 weights.

In order to understand concepts behind CNN I used the help of external resources [6], [7] and to implement took the help of Keras documentation [8]. I designed my architecture as the diagram below with a 3x3 filter and dense output layer with Softmax that outputs 11 classes (1-10 digit and 0 non-digit)



The next approach was to use VGG16 and train it from scratch. I used the model available in Tensorflow with weight as None, which left me with the architecture alone initialised with random values. I modified the last layer to output 11 classes (same as before). I trained in the architecture for my training data.

Finally, I experimented with pre-trained weights of VGG 16 on ImageNet dataset that had 1000 class output with seemingly unrelated data such as animals and vehicles as they are not related to digits in SVHN dataset. I used Transfer Learning as generalizing a model trained for one problem to another has proven to be effective. I froze the initial 8 layers of VGG16, [9] that is the weights in the first 8 layers will not be modified, as they are used to identify edges and basic shapes. The other layers were initialised with the pre-trained weights but were further trained according to my dataset. As usual, modified the output layer to 11 output classes. This fine-tuning method proved to be beneficial.

TRAINING VARIATION

For all the above models there are a set of parameters that can be varied. I varied Loss Function, Optimizer, batch size, Learning rate to choose the combination that best worked for me.

Loss Function: Cost/Loss function quantifies the error between predicted values and expected values and presents it in the form of a single real number. For this problem, I chose Categorical Cross-Entropy loss function as it is a multiclass classification problem

Optimizer: Optimizers are used to minimize/maximize the loss function according to the need. I experimented with Adam optimizer and Stochastic Gradient Descent (SGD). SGD turned out to be better. SGD is an iterative method which is used to find the parameters (weight and bias) of a function that minimizes the loss function as much as possible. To use SGD to its best other hyperparameters should be tuned. They are learning rate and batch size.

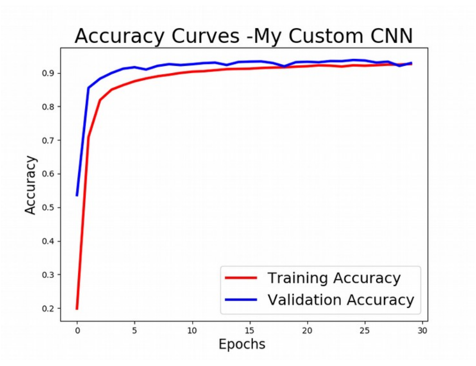
Learning rate: Learning rate is a hyperparameter that controls how much to change the model in response to the estimated error each time the model weights are updated. It is the most important hyperparameter as a very small value may lead to a larger number of epochs to converge while a relatively larger value may cause overshooting. As learning rate is a small positive parameter I experimented with 0.000001, 0.00001, 0.001, 0.1. I found 0.001 to be the most optimal choice.

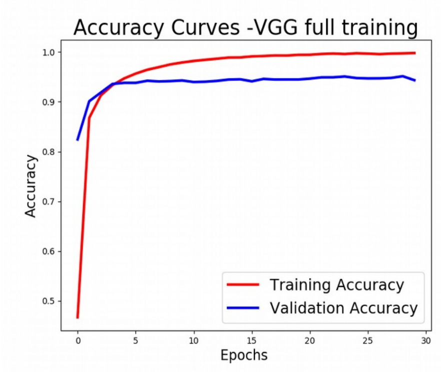
Batch Size: It refers to the number of training examples that are used in one iteration. Batch size typically can be 32, 64, 128. Batch size of 100 worked the best for me. Batch size mainly depends on RAM. I used a laptop with 16GB RAM.

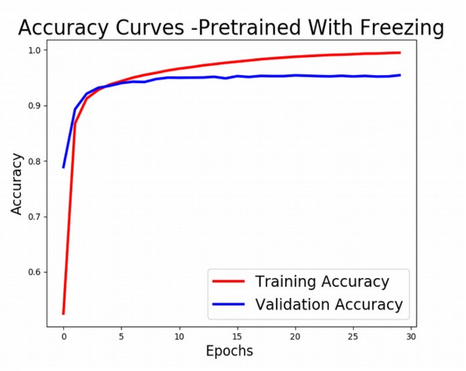
The tuning of hyperparameters is based on how the model is performing on validation data using validation accuracy and validation loss. This is used to prevent the model from overfitting the training data. Validation data is 10% data from train data. The method of early stopping employed which stops the training process, when the validation loss stops to improve in 3 consecutive iterations. The benefits of this method are i) prevents the model from overfitting ii) Reduces the training epochs even if the model doesn't overfit.

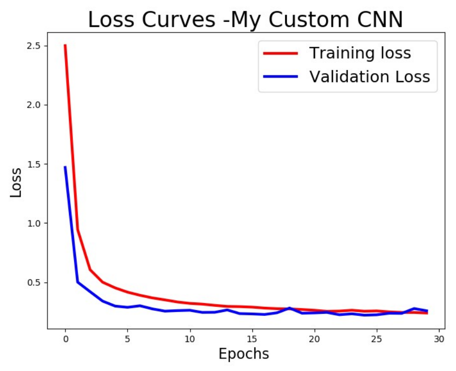
EVALUATING PERFORMANCE

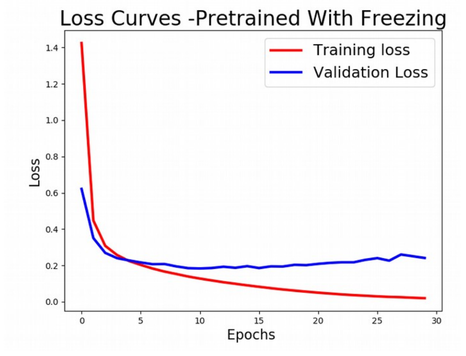
The following graphs were generated without early stopping and training for 30 epochs

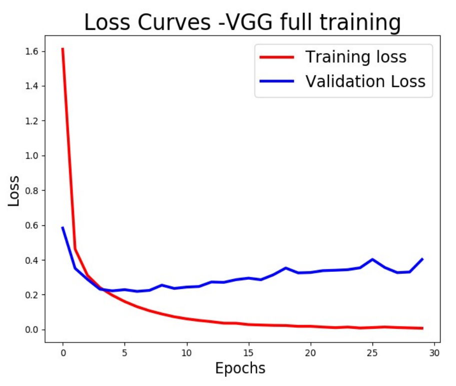










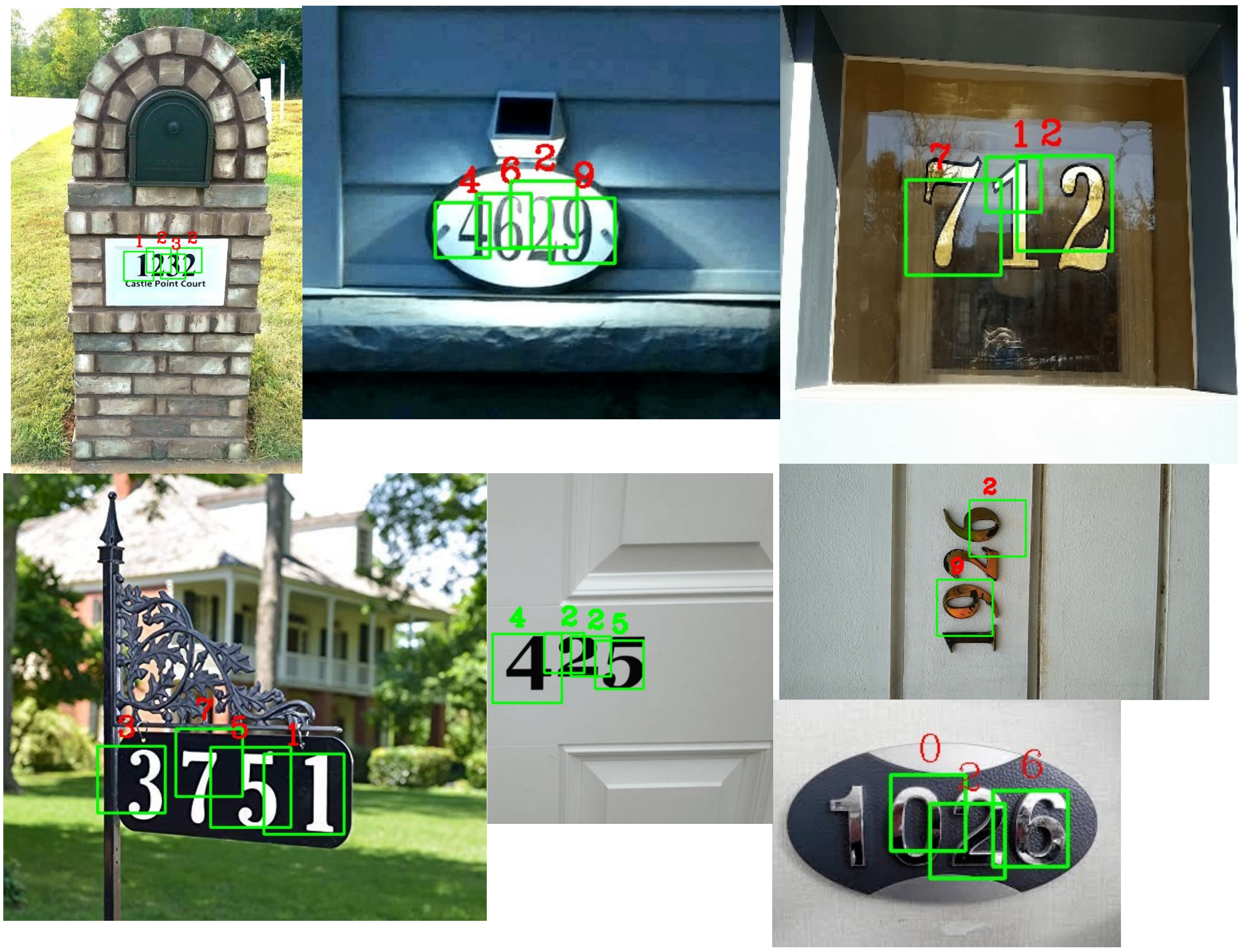


From the graphs above, both the training and validation accuracy sharply increases in the initial 5 iterations and tends to plateau or overfit after that. So early stopping is necessary for this situation. This is the result after early stopping.

Model Name	Train		Validation		Test	
	Accuracy	Loss	Accuracy	Loss	Accuracy	Loss
My CNN	0.90	0.31	0.92	0.26	0.90	0.34
VGG16 Pretrained	0.97	0.097	0.95	0.19	0.94	0.22
VGG16 Scratch	0.96	0.13	0.94	0.22	0.92	0.31

Clearly, VGG16 with pre-trained weights performs better and this model is used for real-world images.

RESULTS



The results of images that work and fails when taken on different scales, location, lighting and orientation. Improving the performance of with stride value >1 as well using image pyramid led to non-tight bounding box in which sometimes digits are ignored or counted twice. Images are trained with small orientation differences. So the model does not detect 90 degree rotated images. Also identifying 1 as a digit is not very robust as most of the non-digit contained vertical edges of buildings which is assumed to be either 1 or non-digit

DISCUSSION

Training the model was not straight forward as small changes in filters, activation and pooling showed great variations in convergence. Hyperparameter tuning was one of the most time-consuming jobs and including limited hardware that posed a constraint. Techniques used to improve detection performance required a lot of fine-tuning to become robust. Especially non-digits were classified as 1 or vice-versa.
The CNN models using individual Softmax model for each digit of sequence proposed by Goodfellow, 2013 [2] reported ~96% accuracy. I am happy VGG16 pre-trained version has ~94% accuracy on test data and my custom CNN has 90% accuracy in testa data.

IMPROVEMENTS

More training data should be used especially to differentiate 1 and non-digit. More samples including alphabet should be included as non-digit. The model is variant to rotation over a certain angle which can be improved by adding more rotated data. Also, bounding box size and accuracy can be improved with further tuning.

LINKS TO VIDEOS

Demo Video: https://youtu.be/WoRBkeaG8Ac

Presentation: https://youtu.be/1EjcUxjSevA

REFRENCES

1. A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks,"

2. Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks, Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, Vinay Shet 2014

3. Very Deep Convolutional Networks for Large-Scale Image Recognition, Karen Simonyan, Andrew Zisserman 2015

4. 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*

5."House price estimation from visual and textual features", Eman H Ahmed, Mohamed N.

6. Michael A. Nielson, " Neural Networks and Deep Learning", Determination Press 2015

7. CS231n: Convolution Neural Networks for Visual Recognition, Spring 2019

8. TensorFlow 2.0 Keras https://www.tensorflow.org/api_docs/python/tf/keras

9. Transfer Learning and the art of using Pretrained models in Deep Learning , Dishashree Gupta
https://www.analyticsvidhya.com/blog/2017/06/transfer-learning-the-art-of-fine-tuning-a-pre-trained-model/