Final Project: Classification and Detection with Convolutional Neural Networks

Rupal Gupta

rgupta338@gatech.edu

Abstract

Object Detection in images is a difficult problem because the algorithm must not only find all objects in an image but also their locations. In this project, I will be addressing both object detection and classification in real images and develop CNN models to identify any sequence of upto 5 digits. I analyzed three different CNN models: a custom CNN model, a VGG-16 model with pre-trained weights (ImageNet dataset) and a VGG-16 model (trained from scratch). Each of these gave test accuracies of 86.72%, 83.72% and 78.42% respectively over entire digit sequence. For multi-digit detection and evaluation of real images I used the custom CNN model. To handle digits at different scales, I used sliding window algorithm and Non Maxima Suppression with image pyramids.

Description of Recent Methods:

This section briefly describes some of the recent work published in Computer Vision research. Starting from 2013, **Overfeat Sliding Window Method** performed object detection by classifying different image patches. Each high probability patch gives a class of that region (dog/person) and its location. This method runs classification on all the sub-windows formed by sliding different sized patches all through the image. My algorithm utilizes a similar sliding window method to identify the location of sequence of digits in an image. In 2015, **R-CNN (Optimized variants like Fast R-CNN (2015) and Faster R-CNN (2015))** [11] was introduced which breaks down the object detection into two tasks: object proposal and region classification. The object proposal task uses algorithms that group super-pixels (like selective search) or algorithms based on sliding window (EdgeBoxes). It uses a technique at test time called Non Maximal Suppression (NMS) which is used to merge highly overlapping regions of same predicted class. I used this algorithm in my custom CNN implementation. Year 2016 saw the advent of **YOLO** [4], You Look Only Once, which is still a state of the art real time object detection method. It handles detection as a regression problem, using only one CNN, where every image is divided into grid to get the class probabilities with spatially separated bounding boxes. Variants like Fast YOLO were even able to outperform other methods like R-CNN. In 2016, an improvement upon YOLO was introduced named **SSD** (Single Shot MultiBox Detector) [11]. It uses multiple sized convolutional feature maps on top of YOLO. It uses anchor boxes at a different aspect ratios and learns the off-set to a certain extent than learning the box itself. Then in 2017, came **Mask R-CNN** which extends the R-CNN functionality to solve the instance segmentation problem by carrying out pixel level segmentation. Mask R-CNN adds a branch to Faster R-CNN that outputs a binary mask which tells whether or not a given pixel is part of an object. Apart from these methods, several architectures have also emerged after AlexNet

(2012) and VGGNet (2014). These modern network architectures include **GoogLeNet** (2014), **ResNet** (2015), **Neural Architecture Search (NAS) Net** (2017), **RetinaNet** (Feb 2018) [18] etc.

Description of implemented method:

Data Analysis and Preparation: To solve the **multi-digit classification problem**, I used the full format data has ~33401 training images and ~13000 test images. I created the negative images from the training set (~34176 images) using the area outside the bounding box. For positive training examples, I consolidated per digit bounding box information to a single bounding box for all the digits. This helped in training model to detect the entire digit sequence at once. Most of the full format images have 3 digits; there are only 5% images with 4 digits, 25 images with 5 digits and only 1 image with 6 digits. I removed the 6 digit image from dataset and assumed digit sequence length (N) of 5 (as per Goodfellow et. al). Image normalization has been performed using mean and standard deviation of the train dataset all throughout. The image patches are resized to (64, 64) and I used normalized grayscale images for custom CNN model and normalized RGB images for both the pretrained VGG and scratch VGG.

Custom CNN Architecture: All three models are developed in Keras using Tensorflow as backend and Python 3.6.7. The models were trained on AWS EC2 p2.xlarge instance. It had a single Tesla K80 GPU and RAM of 61GB. The Functional API of Keras was used (instead of Sequential) because it allows additional flexibility to have multiple inputs and outputs. Before finalizing the architecture of custom CNN, I experimented with different number of layers, kernel size, batch size, optimizers, loss function etc. Finally I used two CNNs in my pipeline, first one performs the task of object recognition/detection and second one does multi-digit classification. The structure of the main 'sequence based digit classifier' CNN is shown in Figure 1. It uses 10 Convolution Layers, 5 Batch Normalization Layers (to provide inputs with zero mean and unit variance) and 5 Max Pool Layers. The Max Pooling layer reduces the image dimensionality and extracts the sharpest features for the model to learn thereby adding some level of translation invariance. This is followed by 2 dense layers with ReLu as the activation function and 1 dense layer with Softmax activation function. The ReLu activation function helps in avoiding the vanishing gradients issue with the sigmoid or tanh activations. The digit recognizer CNN has a similar but slightly different structure and it helps to determine areas where the digits is present. The advantage of using two CNNs in conjunction is that both of them get a chance to filter out false positives.

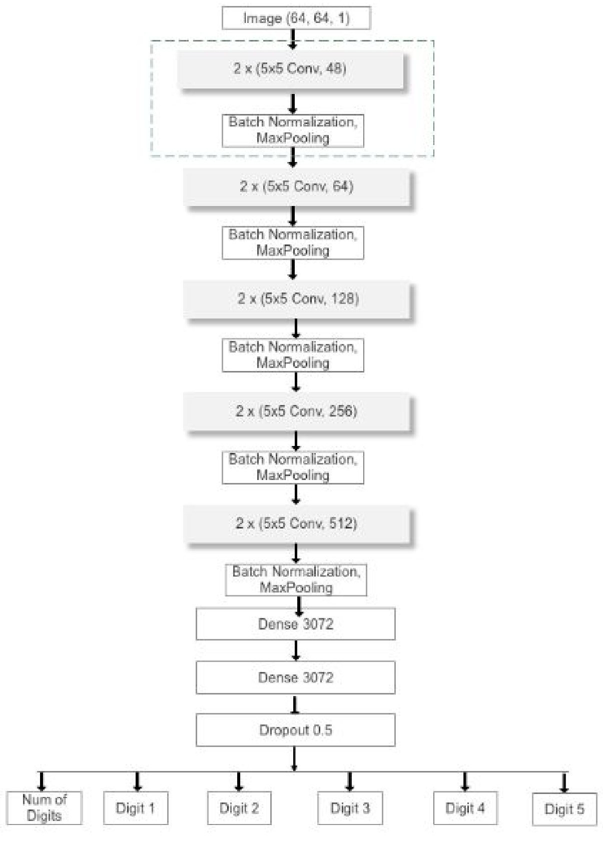

Figure 1. Custom CNN Architecture

VGG16 Architecture: To use the VGG 16 implementation and train it from scratch, I loaded a VGG model in Keras without the top layer (which consists of fully connected layers) and I used weights=None to initialize the weights randomly. For the second model with pre-trained weights, the network simply uses weights='imagenet'. Both these models use the same architecture as the VGG16 model but I added two dense layers with 3072 units and 6 final dense layers with softmax activation function where the first layer returns the total number of digits detected in an image and the other 5 layers return the actual digits detected at each of the 5 places.

Performance Considerations: To get the results from sliding window in a reasonable time, the digit recognizer CNN passes only those windows to the digit classifier CNN which have a confidence over threshold of 88% and discards the remaining windows. Moreover the sliding window method uses different step sizes dynamically at certain image pyramid level, so that a digit is never missed. The digit classifier CNN then processes only these selected windows and returns the bounding boxes and image patches where the confidence of having number of digits is greater than 98%. Finally all the boxes that surround the digit are passed through Non Maxima Suppression and a bounding box around the house number is plotted on the image.

CNN Parameters and other details: Optimizer: I experimented with SGD optimizer but observed very low training accuracies and slower convergence rate. On switching to Adam optimizer, I observed significantly lower training and validation loss. I used the default values for learning rate (0.001), beta_1 (0.9) and beta_2 (0.999) but varied LR and AMSGrad (True/False) as required.
Loss Function: For multi-class classification, I used Sparse Categorical Cross Entropy. Cross entropy is used to measure the dissimilarity between observed class labels and predicted probabilities for integer values of class labels. Instead of dense one-hot coding of the class labels, I used integer value from 0 to 5 for every class label for number of digits and 0 to 10 for class labels for the actual digit value. This worked well with Sparse Categorical Cross Entropy.
Batch Size: Batch size is defined as the number of samples that will be propagated through the network before a gradient descent step is taken. Using a mini-batch size helps because it takes lesser memory to calculate gradients and helps the model trains faster because weights are updated after each propagation. After experimenting with most commonly used batch sizes are 32, 64, 128, 256, I finally used 64.
Learning Rate: Learning Rate (LR) is a hyper-parameter that controls how much the network weights are adjusted with respect to the loss gradient. It was observed that decreasing the LR as training progressed had significant impact on improving the loss of the model as it is able to reach the 'minimum'. The LR was initially set to the default value of Adam optimizer (10^{-3}) and then decreased by a factor of 10 when the validation loss stopped improving. This was achieved by using the 'ReduceLROnPlateau' callback provided by Keras.
When to Stop training: I stopped training the network once validation accuracy started to decrease (loss stopped improving) which suggests overfitting. I used the Early Stopping callback in Keras to monitor loss for 5 epochs and if no improvement observed, I stopped the training right away.
Image Augmentation: I Using sliding windows with Image pyramids fulfils the classifier requirements of being **scale invariant**. To get the right bounding box at different scales I used non-maxima suppression. Furthermore, to make the classifier **invariant to location, rotation/pose, font** I used ImageDataGenerator class provided by Keras.

Results and analysis of the algorithm applied to images:

Images that work well: The algorithm I used for real image evaluation resembles the Overfeat sliding window approach. My actual results on my images and video are shared at links below:

Images: Images Office 365 Drive Link or Images Google Drive Link

Video: Video Office 365 Drive Link or Video Google Drive Link

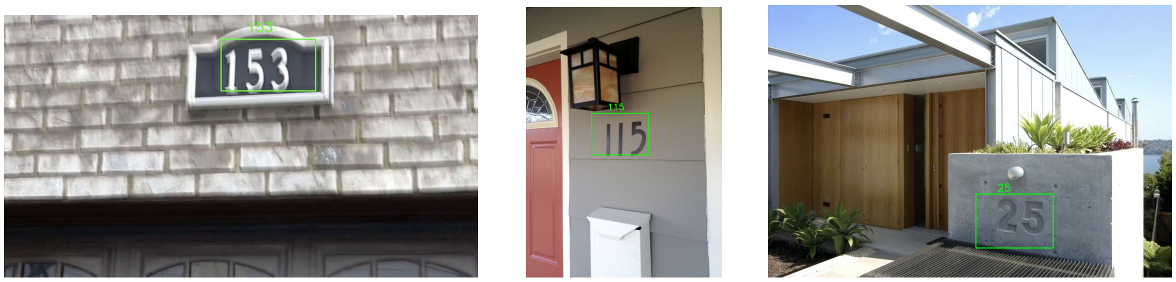

Figure 2: The model predicts the location and digit correctly at different locations and scales in image, at different orientations, at different lightning and texture conditions

Images that do not work well: As evident from the following images, the algorithm does not work when the orientation of digits is different from the ones the model was trained on. Normally the house numbers are written from left to right in a horizontal manner but in the following images, we can see that the model has difficulty predicting digits which are horizontally flipped (mirror images of digits), images with 90 degree rotated digits, vertically stacked digit sequences and any such patterns on which the model has not been trained sufficiently.

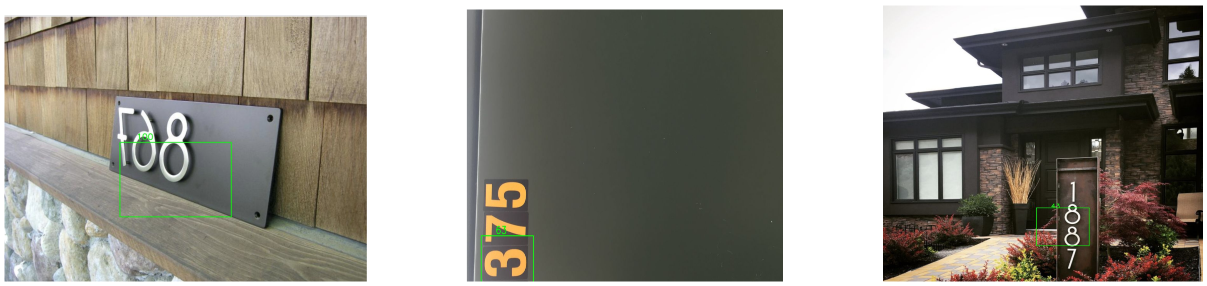

Figure 3: Model returns 100 as output as that is the only pattern it could understand from this image patch. In the second image, the image is flipped counterclockwise by 90 degrees. The rightmost image has digits stacked vertically which the model was never trained on.

Performance Statistics Analysis:

I saved the model weights only at the epochs where lowest loss was reported. I generated plots for loss, error and accuracy for each of the 5 digits, num_of_digits and model loss for all the three CNN models which resulted in total 57 plots. Due to the limited space, I am presenting here the model loss plots, digit 1 error plots and digit 2 accuracy plots for all the three models. All of the plots generated can be accessed at: All_loss_error_and_accuracy_plots

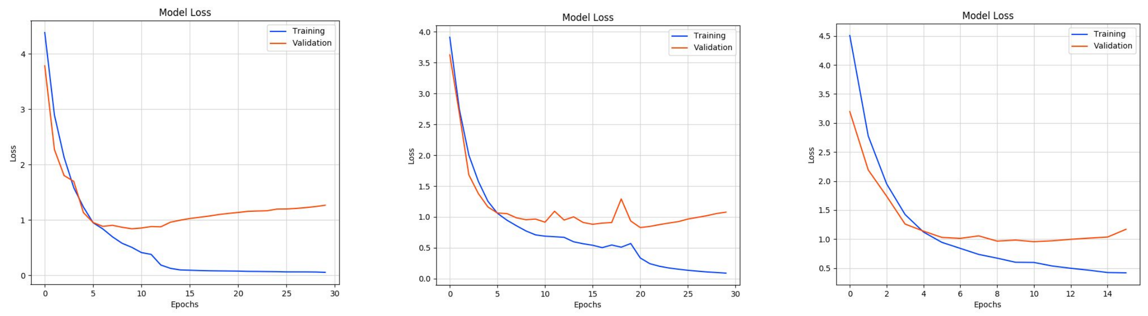

Figure 4. Model Loss for Custom model, VGG16 pretrained weights, VGG 16 scratch (left to right)

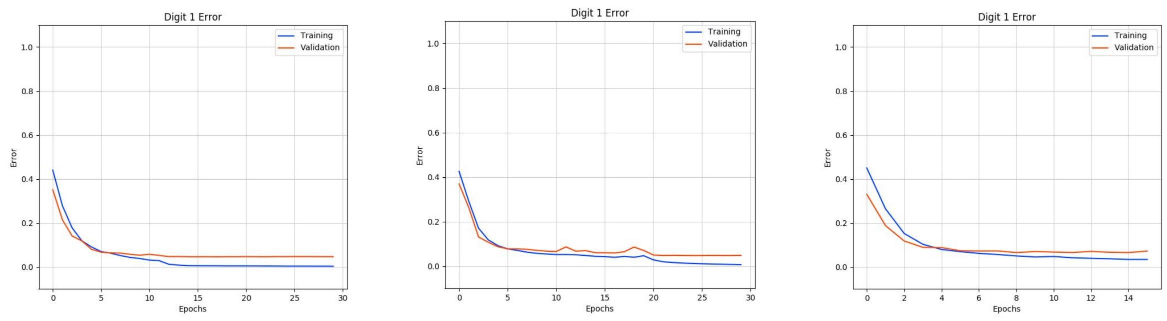

Figure 5. Digit_1 Error plot for Custom model, VGG16 pretrained weights, VGG 16 scratch (left to right)

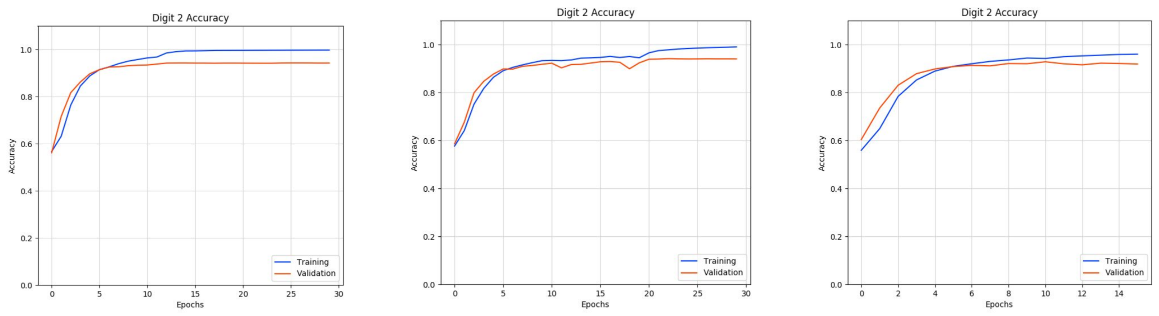

Figure 6. Digit_2 Accuracy plot for Custom model, VGG16 pretrained weights, VGG 16 scratch (left to right)

Comparison of Models: The following table contains the accuracies obtained for each of the model on the entire training set (positive + negative examples) and the entire test set. Since my custom model generates the entire sequence of digits at once, I am only comparing the entire sequence accuracies for all the three models. As evident from the accuracy metrics, the custom CNN is performing better than any of the VGG16 models, hence, I have used my custom CNN model in my final pipeline:

Model	**Overall Train Sequence Percentage Accuracy**	**Overall Test Sequence Percentage Accuracy**
VGG 16 Scratch	85.89%	78.42%
VGG 16 pre-trained weights	88.52%	83.72%
Custom CNN	92.16%	86.72%

Table 1: Comparing training and test digit sequence accuracy for all three models.

Comparison to the State of the Art methods:

Deciding the architecture of the custom CNN took many different iterations of experimentation of tuning hyper-parameters such as choice of optimizer, batch size, learning rate, sliding window operations, thresholds, loss function, layers to use, filter and kernel sizes, normalization etc. Despite all the challenges, my custom model has **86%** accuracy on the SVHN test dataset, **in less than 30 epochs**. Training each of the three models took about 2 hours on the GPU. I believe if I could employ the latest techniques and train for longer, I would have achieved higher accuracies. The state of the art methods mentioned before used deeper networks, had an availability of vast amount of training data and took several days to train. The accuracy reported by Goodfellow et. al (2013) [13] using the multi digit classification on SVHN dataset is **96%**, although they trained their model using millions of non-SVHN street number data and then tested on the SVHN dataset. Taylor et. al (2017) [5] reported the **test set error as 1.6%** on the SVHN test dataset, using WideResNet, per channel mean and standard deviation normalization, and epochs higher than 120.

Improvements and Future Work:

This project was my first attempt with Deep Learning and Convolutional Neural Networks. There are certain limitations to this model a) there are only 4% images in the training set that have 4 digits or more, which reduces the overall accuracy of the model to detect and classify four digits. b) Despite all the efforts to suppress false positives, there are still some cases where false positives cannot be fully removed. Both the issues can be addressed if ample amount of negative and related training data is available while training the model. To improve upon my existing architecture, I could use the following strategies: 1) To improve the speed of sliding window method, I can replace the fully connected layer in ConvNet by 1x1 convolution layers [3]. Also, using the faster state of the srt techniques like YOLO, R-CNN etc would speed up the model. Sliding window also suffers from the drawback that an object might not always fit in the sliding window created, it could be wider or heightened. Hence many times sliding window approach fails to detect such features or only partially detects them. 2)The models could be trained on different orientations of digits such as vertically stacked digits to make it more robust. It could also be trained with many more positive and negative examples so that it could learn the feature representations well.

Presentation Links: Presentation Slides and Presentation Video

References:

1. 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.* (PDF)
2. Nitish Shirish Keskar and Richard Socher (2017) Improving Generalization Performance by switching from Adam to SGD https://arxiv.org/pdf/1712.07628.pdf
3. Convolutional Neural Networks (by Andrew Ng) https://www.coursera.org/learn/convolutional-neural-networks
4. Joseph Redmon, Santosh Divvala, Ross Girshick , Ali Farhadi (2016), You Look Only Once (Unified, Real Time Object Detection) https://arxiv.org/pdf/1506.02640.pdf
5. Terrance DeVries and Graham W. Taylor (2017), *Improved Regularization of Convolutional Neural Networks with Cutout* https://arxiv.org/pdf/1708.04552.pdf
6. Jan Hosang, Rodrigo Benenson, Bernt Schiele (2017) *Learning non-maximum suppression* https://arxiv.org/pdf/1705.02950.pdf
7. Included some negative images from: Image Geo-localization based on Multiple Nearest Neighbor Feature Matching using Generalized Graphs. Amir Roshan Zamir and Mubarak Shah. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 2014. http://crcv.ucf.edu/data/GMCP_Geolocalization/
8. Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollar (2018) *Focal Loss for Dense Object Detection* https://arxiv.org/pdf/1708.02002.pdf
9. Nitish Shirish Keskar, Richard Socher (2017) *Improving Generalization Performance by Switching from Adam to SGD* https://arxiv.org/pdf/1712.07628.pdf
10. Ross Girshick, Jeff Donahue, Trevor Darrell and Jitendra Malik (2014) *Rich feature hierarchies for accurate object detection and semantic segmentation* https://arxiv.org/pdf/1311.2524.pdf
11. Wei Liu, Scott Reed , Cheng-Yang Fu , Alexander C. Berg et. al SSD: *Single Shot MultiBox Detector* https://arxiv.org/pdf/1512.02325.pdf
12. Spyros Gidaris, Nikos Komodakis (2015), *Object detection via a multi-region & semantic segmentation-aware CNN model* https://arxiv.org/pdf/1505.01749.pdf
13. Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, Vinay Shet *Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks* https://arxiv.org/pdf/1312.6082.pdf