CLASSIFICATION AND DETECTION WITH CONVOLUTIONAL NEURAL NETWORKS

SCOTT LILLEBOE - SCOTT.LILLEBOE@GMAIL.COM

OVERVIEW

The project required the recognition and detection of housing numbers using Convolutional Neural Networks and the use of image pyramids with non-maxima suppression. The dataset that was used for the project was the SVHN dataset 2 (32x32 images) (Yuval Netzer, 2011). A short video presentation can be found here:

- YouTube: https://youtu.be/w7ldLmji-lw

NOISE MANAGEMENT

A Gaussian blur was utilized to suppress random noise caused by both poor image quality and image artifacts like subtle texture differences in objects like building materials (stucco, bricks, cement, asphalt, etc.). I found some lighting differences caused by shadows were slightly suppressed by blurring causing less false positives.

LOCATION AND SCALE INVARIANCE

A sliding window at various image scales was implemented to gather an array of image patches to be fed to the CNN. This allowed a digit to be identified across the image space.

PERFORMANCE CONSIDERATIONS

To mitigate the number of image patches that would be fed to the CNN, a step size parameter was used and determined based on the scaling of the original image, the larger the image the larger the step size. The CNN itself was trained with various augmentations that let it be more robust to scale variances. This helped bring down the number of scales that needed to be applied to the image. The result for each image patch set on a Windows 10 desktop with 4 cores (8 logical CPUs at 3.5 GHz), 16GB of ram, was approximately 50 seconds.

MODELS ANALYSIS AND PERFORMANCE STATISTICS

Three CNN algorithms were evaluated to determine which would be utilized for the final image and video processing. Each algorithm was tested with the following conditions:

- 10% of training, validation, testing data (randomly selected)
    - Approximately 30K training, 3K validation, 5k testing
- 50 epochs
- Mean and standard deviation normalized training, validation and testing data
- 4 optimizers with all the same parameters per optimizer

CUSTOM MODEL

The model was a CNN that consisted of 3 convolution blocks each consisting of three layers: Conv2d, BatchNormalization, and Relu activation. Those three convolution blocks were followed by a flattened layer, three fully connected layers and finally a fully connected softmax layer. The total number of parameters for this CNN was 947,755 with 946,859 as trainable. This model performed the best with the Nadam optimizer with regards to training accuracy per epoch and resulting test accuracy, Table 2. The input to the model were 32x32 pixel images.

VGG16

This model used the VGG16 as its backbone except the top three fully connected layers were removed. Those layers were removed to mitigate the memory usage, training and prediction times for the model. It reduced the total number of parameters from 138,368,555 to 14,720,331. A flattened layer, three full connected layers and a dense layer with a softmax activation were added that brought the total and trainable parameters to 15,211,595. The input to the model were 48x48 pixel images.

The model was further split into two main configurations, not trained and pre-trained with the ImageNet dataset (Karen Simonyan, 2015). The untrained model performed well for some for the optimizers; however, where it lagged the pre-trained model was the first 10 epochs. Figures 2 and 3 shows that after about the 10th epoch both models were in the 90% plus accuracy range for training, verification and testing values, but the pre-trained reached those values by the end of the second epoch's training. The pre-trained model was selected to further research based on the higher accuracy values which could have been influenced by its "head start" seen in the learning curves.

VGG16 pre-trained was further tested with the VGG16 layers frozen (non-trainable). This decreased the training time of the model as the backpropagation no longer needed to update most of the weight values. Table 2 displays the accuracy results of this method and it was shown that the accuracy values plateaued about at the 80% training value. It is suspected this was due to the lack of trainable parameters caused by the freezing.

Table 1

Optimizer	Learning Rate
AdaDelta	1.0
Adam	0.001
Nadam	0.002
SGD	.01

Table 1 displays various learning rates that were set for the optimizers that were applied to all the test models. For the VGG16 pre-trained model SGD was selected due to the resulting test accuracy was higher than the other optimizers. Adam and Nadam performed very well for the custom model, but they were poor for both VGG16 variations. The intuition was the poor performance was due to the learning rates being too low causing very slow learning and possibly getting stuck in local minima. What was also utilized was Keras's ReduceLROnPlateau which gradually decreased the learning rate by a factor of 0.2 when the validation loss value did not decrease for the previous 2 epochs. The thought behind this was to emulate simulated annealing by letting the backpropagation adjust the weight values quickly at first but gradually settle overtime as learning became more difficult with respect to decreasing the validation loss value.

Having selected the model, VGG16 with pre-trained weights, it was further trained beyond its initial dataset that used to filter out the various models and combinations of optimizers. Due to hardware limitations a system was coded that looped through 5 epochs on 20% of the total training, validation, testing data for 10 iterations. Each dataset contained 56K training, 8K validation and 10K testing samples that was randomly selected from each of their respective datasets. This method was chosen to keep an amount of previously viewed samples in each epoch iteration. Within these datasets a Keras ImageDataGenerator was utilized to further augment the data by introducing rotations to the data.

After 85 total epochs training was halted due to accuracy rates no longer seeing increases, Figure 4. There was not any distinct epoch level that started to show overfitting where the training and validation began to diverge, they just stopped significantly increasing. The decision was made to end after this epoch due to time constraints on the project and the belief that all the training data had sufficiently be utilized. It is suspected that the large training data set, the introduction of 25% dropout layers between each fully connected layer, and the randomization of training batches and data augmentation helped decrease the risk of overfitting. The model was accepted as completed with training and was fed into the pipeline and satisfactory results were found, see Figures 5-10 and the available video. The final model architecture can be seen at the bottom of Table 2.

Categorical cross entropy was the loss function that the CNN utilized due to a multi categorical output of the network (McCaffrey, 2014). The CNN had 11 classes, 10 for digits 0-9 and one for negative samples that didn't represent a digit.

Table 2

Model & Optimizer Accuracy Metrics				
Algorithm	Optimizer	Train Accuracy	Validation Accuracy	Test Accuracy
Custom	Nadam	.975	.979	.947
	Adam	.986	.976	.941
	Adadelta	.986	.972	.928
	SGD	.983	.982	.944
VGG16 Not Trained	Nadam	.398	.416	.302
	Adam	.465	.448	.341
	Adadelta	.994	.981	.950
	SGD	.966	.968	.902
VGG16 Pre-Trained	Nadam	.472	.469	.335
	Adam	.467	.488	.348
	Adadelta	.976	.978	.930
	SGD	.998	.978	.952
	Nadam w/Layers Frozen	.808	.789	.688
	Adam w/Layers Frozen	.785	.762	.666
	Adadelta w/Layers Frozen	.799	.755	.695
	SGD w/Layers Frozen	.770	.760	.683

Final Model Architecture		
Layer (type)	Output Shape	Param #
img_in (InputLayer)	(None, 48, 48, 3)	0
vgg16 (Model)	multiple	14,714,688
flatten_1 (Flatten)	(None, 512)	0
dense_1 (Dense)	(None, 384)	196,992
dropout_1 (Dropout)	(None, 384)	0
dense_2 (Dense)	(None, 384)	147,840
dropout_2 (Dropout)	(None, 384)	0
dense_3 (Dense)	(None, 384)	147,840
dropout_3 (Dropout)	(None, 384)	0
y (Dense)	(None, 11)	4,235
Total params: 15,211,595 Trainable params: 15,211,595 Non-trainable params: 0		

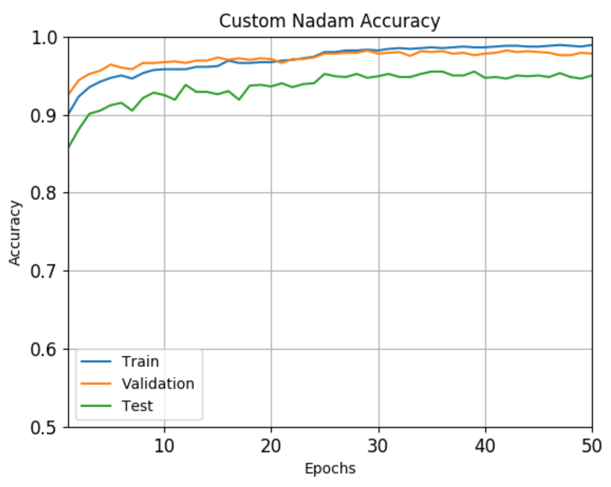

Figure 1

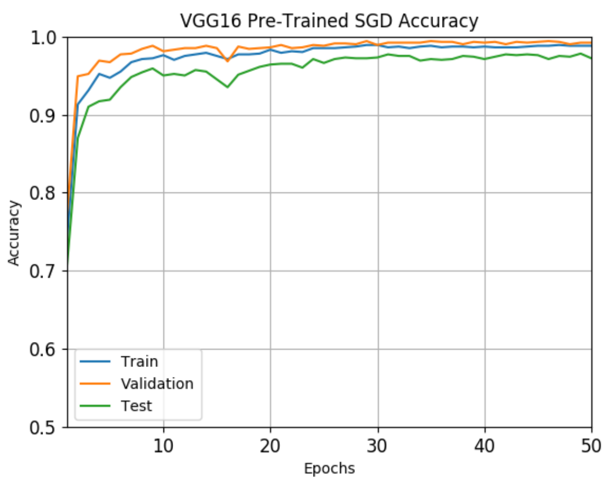

Figure 2

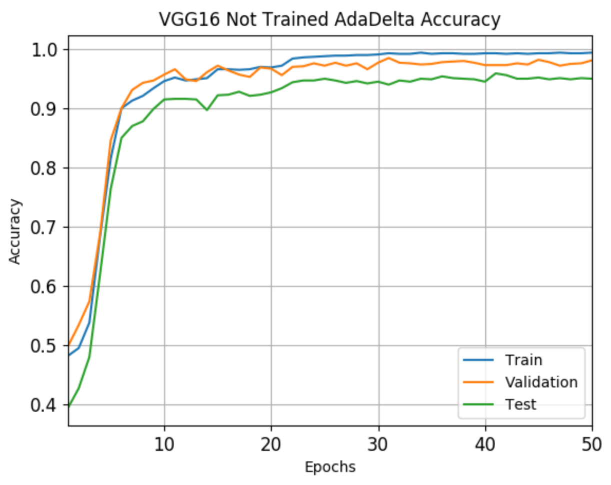

Figure 3

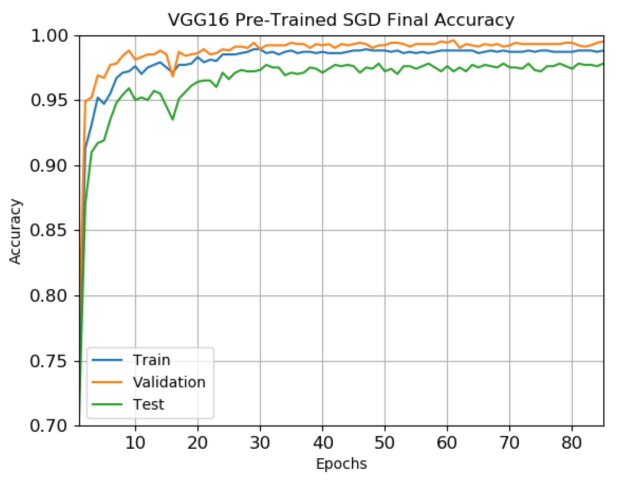

Figure 4

Table 3

RESULTS

Figures 5-10 and the link to the video display the final results obtained from the pipeline and CNN developed for the project. The chosen images show different orientations, scales, fonts, and lighting variations. Figure 10 is a screenshot from the video, the link can be found below the screenshot. The images presented correctly classified the digits; however, there are multiple images where false positives are found that either had highly complex scenes or contained a mixture of digits and letters. The final presentation video highlights one of those cases of poor recognition and detection by this model.

The number of false positives for the presented images and video were reduced by implement the following processes:

- Slightly blurring the input image through a 5x5 Gaussian blur.
- Increase CNN complexity (depth) which was provide by the VGG16 model
- Provide a large training set, approximately 300K training samples
- Augment the training set by utilizing Keras's image data generators
- Use different threshold values for each digit from CNN predict values
- Use non-maximum suppression to limit total number of bounding boxes

- Create custom algorithms to limit false positive bounding boxes.

The last point needs further clarification. Certain digits were seen in images that were false positives, especially the digit of 1. The first pass of eliminating artifacts is to find the mean centroid of all the found bounding boxes of the digits. With that mean value, all boxes with centroids that were four standard deviations from the mean were removed. The intuition behind this methodology was true digits would be found multiple times due to the multiple passes of the sliding window which would bias the total centroid mean towards the true values. This didn't eliminate false positives, so another step was required.

The next step was to find the average distance between all the remaining bounding box centroids. From this value, boxes that had their individual average centroid distance greater than 2.5 times the distance of the overall average were removed from consideration. The intuition behind this step was addresses are image sequences that are usually a set distance between each other. If a bounding box was found that was too far from the overall distance value had a higher probability of not being part of the true digit sequence.

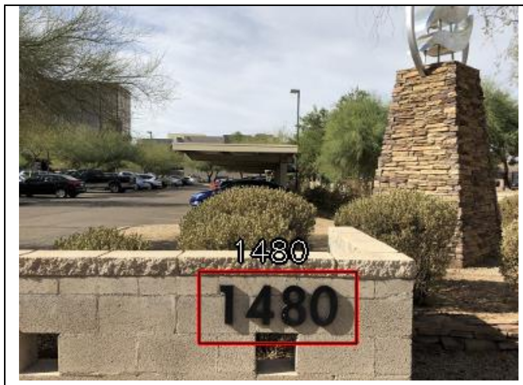

Figure 5

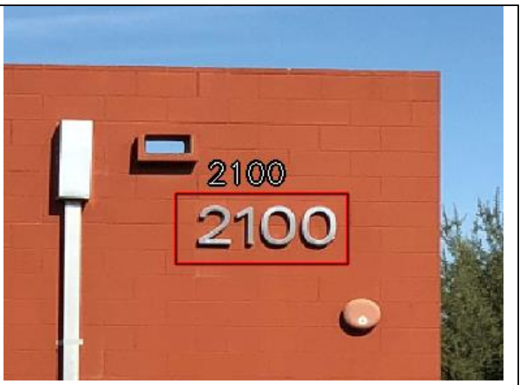

Figure 6

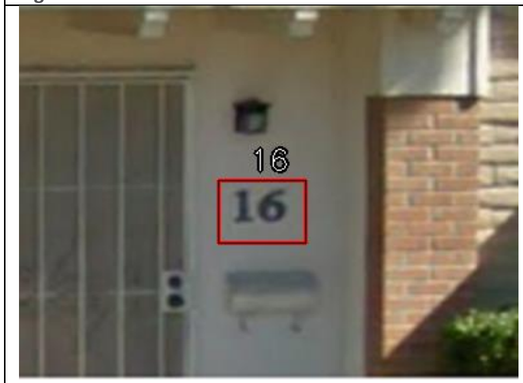

Figure 7

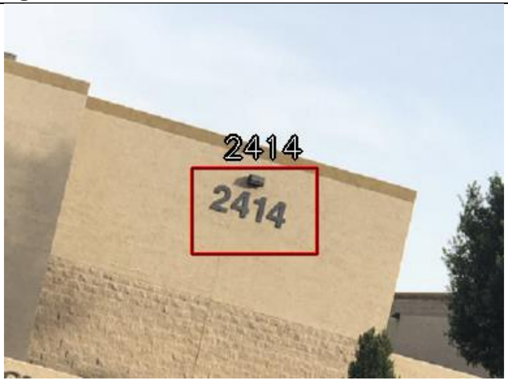

Figure 8

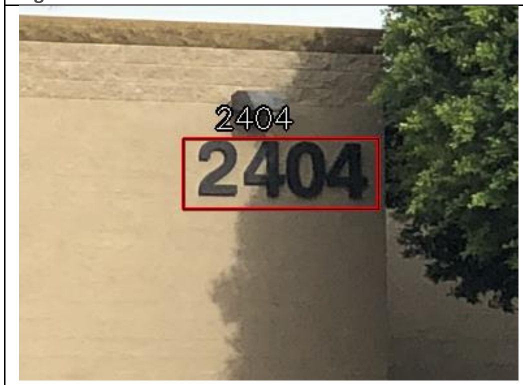

Figure 9

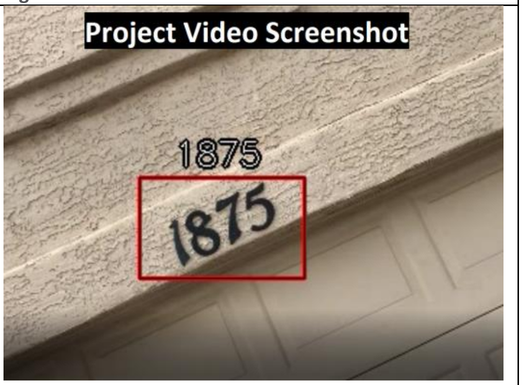

Figure 10

Video can be found at: https://www.youtube.com/watch?v=xWz1FvkK7Kc

For the selected images and video, the pre-trained VGG16 model performed well and the overall accuracy was acceptable; however, it was highly specialized for the given input data and project task. To increase the robustness of the model a larger amount of training data, especially for the negative data set would need to be acquired and applied. The current model is biased with respect to the negative training because of limited project time and hardware constraints. The original explored model used a full sequence classifier rather than the single digit that the model was finalized with. Initial results were promising, but there were issues classifying sequences longer than three. The suspected issue was caused by warping of the input data to fit a 48x48 pixel space. Further efforts might have corrected the mistakes, but time constraints resulted in moving the project in its finalized direction.

EXISTING METHODS COMPARISSON

The original custom model that was created was based on the REsNet50 architecture (Kaiming He, 2015). This approach was abandoned due to time and hardware constraints. The model was producing promising results albeit at a very slow rate. The YOLO model would have been an interesting one to purse for this project as it would have helped speed up the slowest portion of the pipeline, CNN prediction. Where this project used a sliding window to input an array of image slices YOLO uses a single neural network to predict bounding boxes and class probabilities for the full image over a single evaluation (Joseph Redmon S. D., 2016).

REFERENCES

Diederik Kingma, J. B. (2014, December 22). *Adam: A Method for Stochastic Optimization.* Retrieved from Cornell University Library: https://arxiv.org/abs/1412.6980v9

Joseph Redmon, A. F. (2016, December 25). *YOLO9000: Better, Faster, Stronger.* Retrieved from Cornell University Library: https://arxiv.org/abs/1612.08242

Joseph Redmon, S. D. (2016, May 9). *You Only Look Once: Unified, Real-Time Object Detection.* Retrieved from Cornell University Library: https://arxiv.org/abs/1506.02640v5

Kaiming He, X. Z. (2015, December 10). *Deep Residual Learning for Image Recognition.* Retrieved from Cornell University Library: https://arxiv.org/abs/1512.03385

Karen Simonyan, A. Z. (2015, April 10). *Very Deep Convolutional Networks for Large-Scale Image Recognition.* Retrieved from Cornell University Library: https://arxiv.org/abs/1409.1556v6

McCaffrey, J. (2014, April 22). *Neural Network Cross Entropy Error*. Retrieved from Visual Studios Magazine: https://visualstudiomagazine.com/articles/2014/04/01/neural-network-cross-entropy-error.aspx

Samuel L. Smith, P.-J. K. (2017, November 1). *Don't Decay the Learning Rate, Increase the Batch Size.* Retrieved from Cornell University Library: https://arxiv.org/abs/1711.00489v1

Yuval Netzer, T. W. (2011). *Reading Digits in Natural Images with Unsupervised Feature Learning - NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*. Retrieved from The Street View House Numbers (SVHN) Dataset: http://ufldl.stanford.edu/housenumbers/