ConvNets Applied to House Number detection

James Corwell

***Abstract*—This paper serves as an exploration into detection in Computer Vision utilizing Convolutional Neural Networks.**

I. Introduction

A convolutional neural network (CNN) is a type of feed-forward artificial neural network that has shown great promise in computer vision detection applications.

Introduced here are three different CNNs. Two are the popular VGG16 implementations, famous for receiving a record-breaking score on the ImageNet dataset. Also introduced will be a custom architecture CNN (CA-CNN).

Modern research has revolved significantly around CNNs. Goodfellow et al. [4] detail how a CNN trained on SVHN dataset can attain ~98% accuracy on single digits, and a 96% accuracy in recognizing entire street numbers. CNN research has become much more promising thanks to advances in GPU computing, and now I will attempt to achieve similar results on my 2015 macbook air.

II. Methods and Implementation

The implemented CA-CNN network uses 6 convolutional layers varying in size from 32x32 to 512x512 neurons, with batch normalization, a dropout of .3, and ReLU activation. ReLU is the most popular activation function to be used in CNN architecture. The final layer is a dense layer using softmax activation. This enables us an output of 10 different possible classifications (0-9) and an associated probability of what the CNN believes it is in each node. The sum of the probability nodes is guaranteed to equal to 1.

The networks were then trained on the SVHN dataset, image type 2 [1]. Which consisted of 73257 digits for training, and 26032 digits for testing. All images were 32x32 pixels, and all color channels were retained during training.

figure showing 4 different samples of training instance images

The CA-CNN used an adam optimizer. The CNN used a learning rate of .001 and a batch size of 256. It was originally set to train for 100 epochs, with an Early Stopping check set to stop training if improvement in the validation loss was not detected for two epochs. Overall the CA-CNN btained a training and validation accuracy >94% and trained in as few as 8 epochs. Accuracy was measured in MSE.

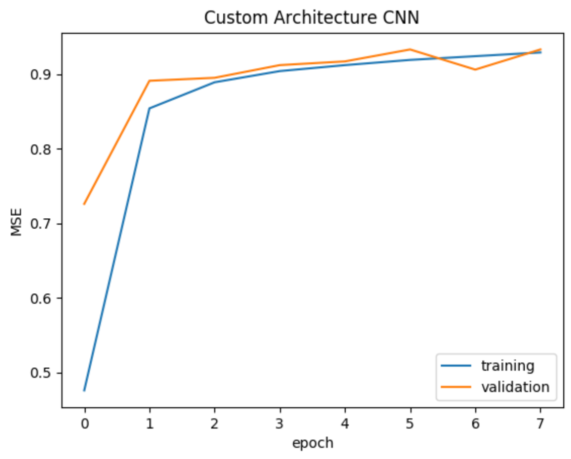


Figure depicting learning curve for CA-CNN

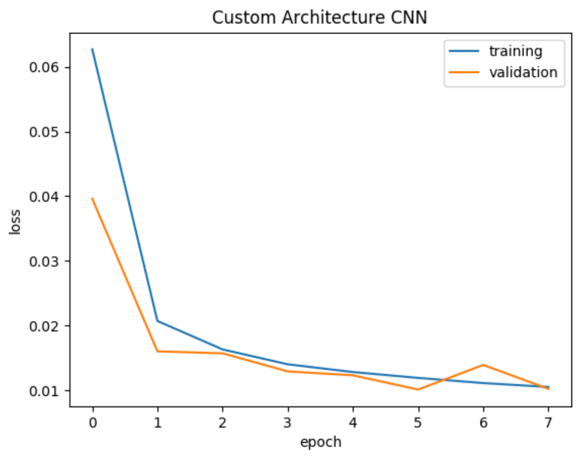


Figure depicting loss curve for CA-CNN

Training VGG16 was more difficult. The sheer density of these CNNs makes them more difficult to work with. According to the original VGG paper [2], the original VGG networks were designed with a batch-size of 256. They used two different learning rates, 10^-2, and 10^-3 after accuracy plateaued. In the experimental attempt shown here, a few different values were attempted in the learning rate, based off of this paper and online recommendations.

The VGG16 framework had to be slightly modified to work the SVHN data. Namely, the first layer accepting the images as input needed to be reshaped to accept 32x32 input. Then the outer layers needed to be reshaped to give a 10-classification output. Keras makes loading and manipulating the VGG networks simple, as they are preloaded into tensorflow. In this experiment adam optimizer was used instead of SGD

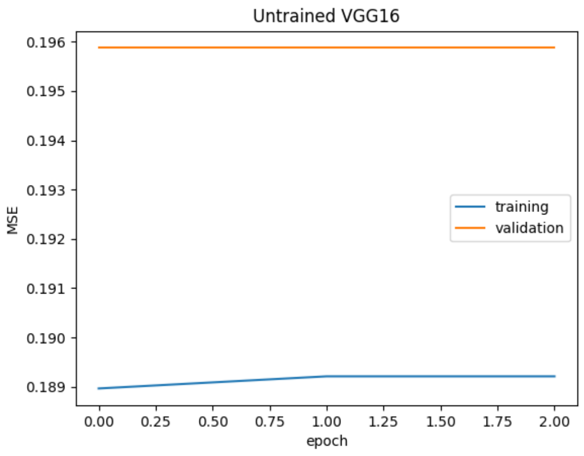


Figure depicting learning curve for VGG16 with random weights

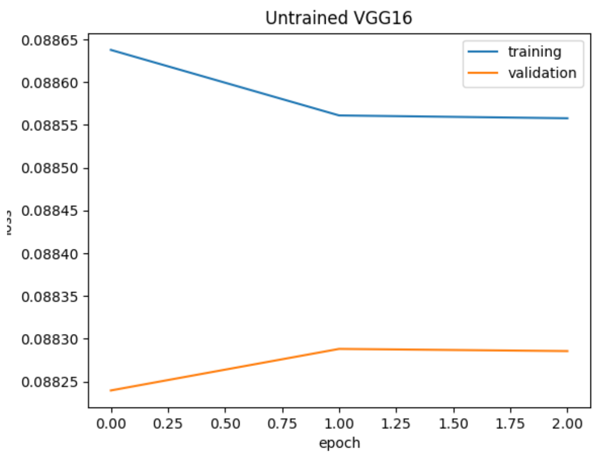


Figure depicting loss curve for VGG16 with random weights

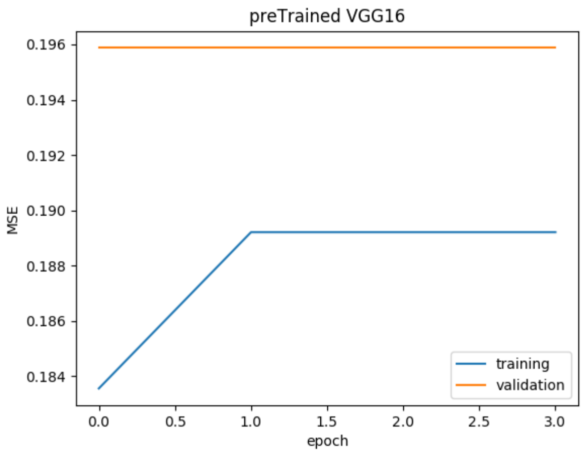


Figure depicting learning curve for VGG16 initialized with ImageNet weights

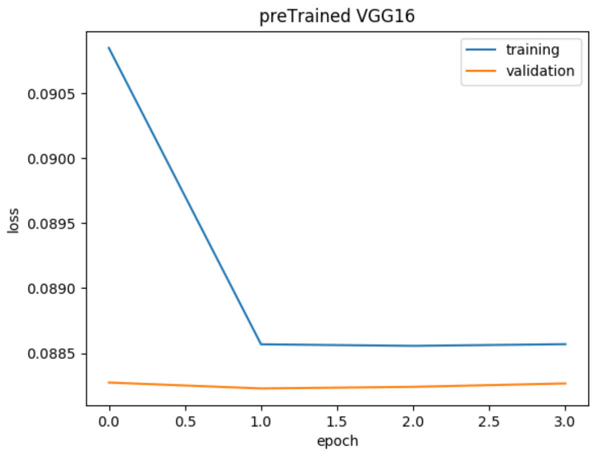


Figure depicting loss curve for VGG16 initialized with ImageNet weights

VGG16 was unable to break a 20% accuracy threshold. This was discussed heavily in the course project slack channel. Some were able to break the 20% threshold after several epochs, but unfortunately my machine was not able to train for very large ranges. I attempted different learning rates as well, without noticing any difference in performance.

After the classifier was trained to satisfaction. The next step in the process was to make a mechanism to scan an image searching for numbers. This was achieved with a Sliding scale window method. First you scan across the original image with a 32x32 pixel range and feed that window into the classifier. After parsing the whole image, you scale image down and parse it with a 32x32 window again. This designs a multiple-scale detection approach.

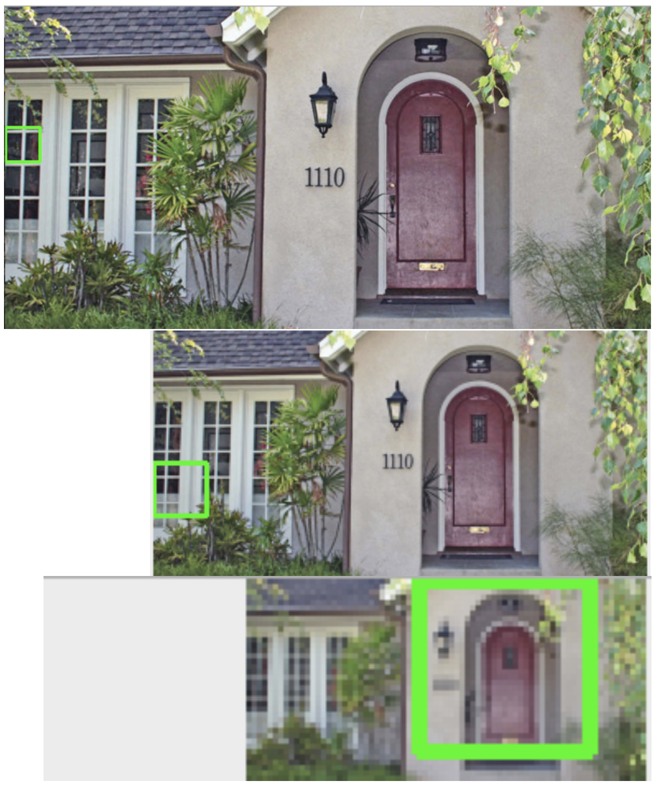


figure depicting a Gaussian pyramid with a 32x32 pixel range parsing the span of the file

The classifier is run for every instance of the 32x32 window moving across the image. The window moves at a horizontal velocity of 11 pixels per iteration.

III. Results

In order to demonstrate the results, one of the first images of a house with numbers was taken from google search [figure 1]. The numbers are on clear display in the center and there are many non-numeric items in the image.

figure of original test image.

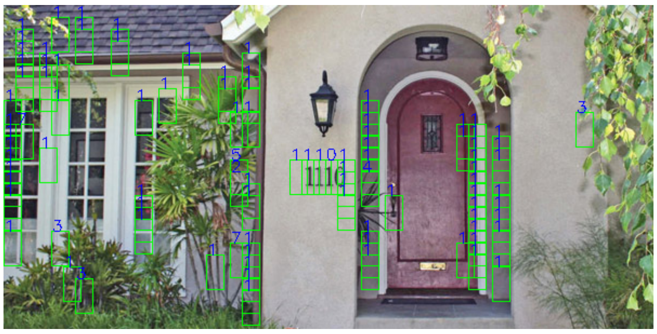


fig of the original image with classification

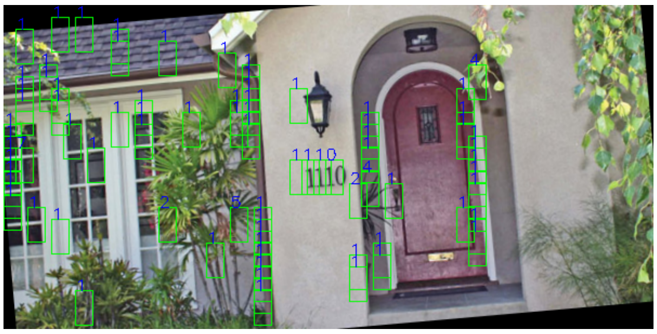


figure of measurement with rotation

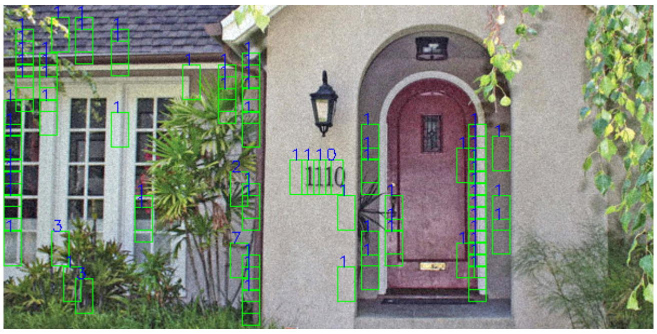


figure of measurement with Gaussian noise

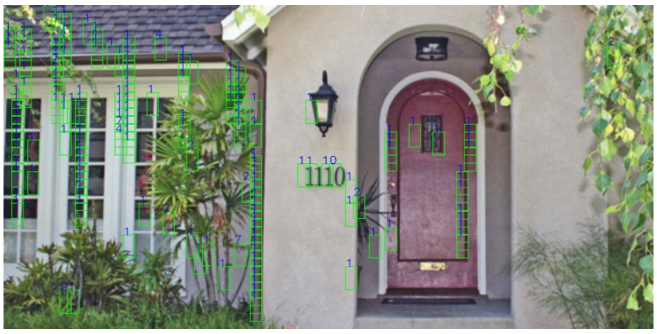


figure of the image with 2x the pixel count, (scale invariance)

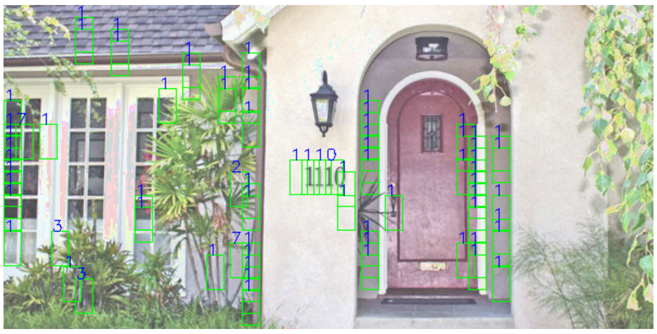


figure of image with increased luminosity

In all of these cases the CA-CNN was able to correctly classify the house numbers. There is a significant number of false positives scattering the image, however. This is probably due to the complexity of the background. This could be efficiently solved by having an area of measurement

In an example of a slight mis-categorization we can see when we attempt location independence [Appendix I]. Here we simply stack the image horizontally and see if it classifies each number correctly. It almost does, but falls slightly short. This mis-categorization is due to the velocity and/or widith of the moving window. If the classifier's window is awkwardly positioned, it can mislabel or miss instances.

IV. Conclusion/further

This implementation compares honorably to state of the art methods. Also the accuracy on individual digits was not as great as Goodfellow et al. [4], an accuracy of >94% is successful. The dark side to Deep Machine Learning projects is often the application of your trained classifier. As you can see in my results, there is still some work to do to filter the results.

The next steps for this project would be to remove the instances of classification of objects that are not numbers. To do this, I'd want to implement a form of pattern recognition. Firstly, integrating non-maxima suppression into the scanning process. Essentially, I could lower the bounds of probable detection, and if there are multiple measurements of a number in that area, then non-maxima suppression could be applied and the number classified [3]. This would work as a noise-dampener essentially, and prevent multiple identifications of the same point, as well as allowing me to lower my measurement certainty, as I am now allowing multiple measurements.

Other than that, I think sacrificing location variance would be a noble way to get rid of noise. If you make it a design requirement to have the address centered on the image, the detector will know where to look, and will not have as many false-positives. This is analogous to a variate 'center' such as when you tap your iPhone to determine where to give focus instead of relying on autofocus.

Another way to increase functional identification accuracy would be to train the CA-CNN classifier with an 11th class that essentially represents 'no numeric value found'. You would have to train the CA-CNN with extra data of noise, trees, etc. but these negative examples may provide the network with a robust counter-measure to unknown things.

There were many difficulties in implementing this experiment with limited hardware. On a 2015 MacBook Air, a single epoch of VGG16 could take more than 3 hours. It became quickly unfeasible to train and experiment with VGG16, especially considering the high performance on the CA-CNN. An AWS instance would be a noble way to circumvent this, but this was not available at the time.

Also, it is worth noting that the 'cv_proj.yml' environment provided for this project did not work for Mac OSX. The best attempts were made to recreate the environment however possible, and further details of the environment are located in the source code.

Overall, this project shows that you can train a classifier to great accuracy using a CNN, and then apply it in real Computer Vision facets.

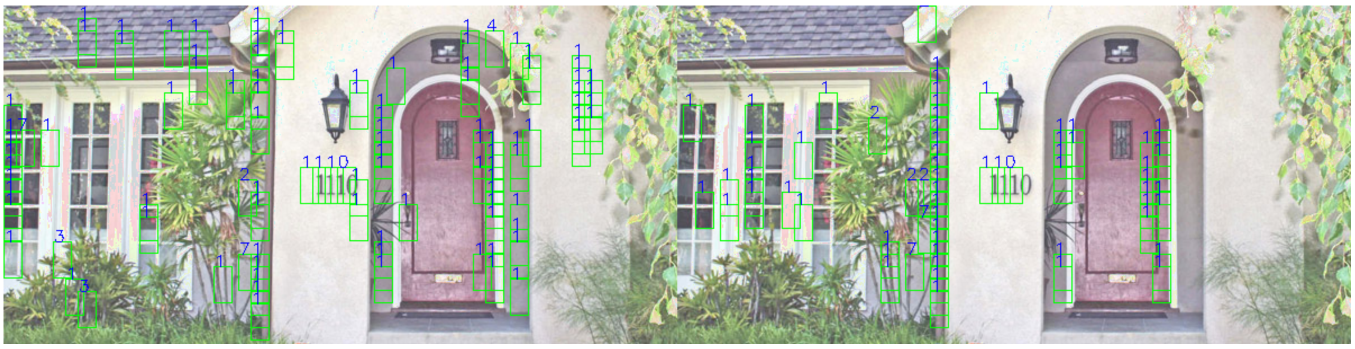
V. References

[4] I. Goodfellow, et al. "Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks" https://arxiv.org/abs/1312.6082

[1] Stanford SVHN data http://ufldl.stanford.edu/housenumbers/

[2] K. Simonyan, A Zissermana "Very Deep Convolutional Networks for Large-Scale Image Recognition" https://arxiv.org/pdf/1409.1556.pdf

[3] R. Rothe et al. "Non-Maximum Suppression for Object Detection by Passing Messages between Windows" https://www.vision.ee.ethz.ch/publications/papers/proceedings/eth_biwi_01126.pdf



Appendix I. Image stacked horizontally showing perfect number recognition on the left, alongside a slight mishap on the right, even though left and right are the same. A failure of location invariance.