Detection and Classification with Convolutional Neural Networks

Isak Diaz isak@gatech.edu
December 02, 2017

Detecting and classifying multiple digits on unconstrained natural photographs has been the subject of significant research efforts in the past 20 years. Goodfellow et al. Punlished a paper “Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks” in which the Street View House Numbers (SVHN) dataset was used to train a convolutional neural network that evaluated up to 5 digits in a single image. Prior to 2012, the paper states that all previous work had focused on classifying single digit images. Goodfellow instead cropped the bounding boxes and resized them to 64 x 64, then they took 54 x 54 windows of each box and trained on those. Their best model produced results of 96.03% on multi-digit transcription and 97.84% on single-digit transcription. Their model took 6 days to train.

The method I chose to implement required three steps. First the processing of the image to match the format of the training data, then the detection of the bounding box using a sliding window and pyramid scheme, and finally the classification of the digits by applying transformations to the bounding box and passing it through the model a second time. This produced robust results.

Training the Models

Custom Model

I created a custom model first which used 48 x 48 gray scale images. This model consisted of 6 blocks of convolution layers, each with it’s own batch normalization, 2D max pooling and dropout layers. Batch normalization allows our inputs to keep a 0 mean and unit variance which helps faster convergence as large weights within our network are avoided. 2D max pooling reduces the complexity of our inputs and also makes it more feature invariant which helps with the generalization of our model. Having dropout layers with 25% dropout rates prevents over fitting of training data by removing nodes during the training process.

VGG-16 from Scratch

The Visual Geometry Group of the university of Oxford created a convolution neural network model which they then released to the public after winning the ImageNet Large Scale Visual Recognition Competition (ILSVRC). Their 2014 version of the model is included in the Keras package. I implemented two versions of their model. The first being trained with randomly initialized weights. Since the Keras documentation read “It should have exactly 3 inputs channels, and width and height should be no smaller than 48”, the data was preprocessed to include 48x48 BGR images as inputs with the mean subtracted.

The key to adapting the VGG16 model to a various set of outputs was to remove the top 3 layers of the original implementation which consists of two fully connected 4096 dense node layers followed by a final classification block. ReLU activation is used for all layers except the last one which consists of softmax activation. These top layers were connected by flattening the output of the VGG model. The top layers consisted of 3 Dense layers of 512 nodes, ReLU activation and Dropout layers set to 25 percent. The output of these layers were input into each of the 5 classification layers which

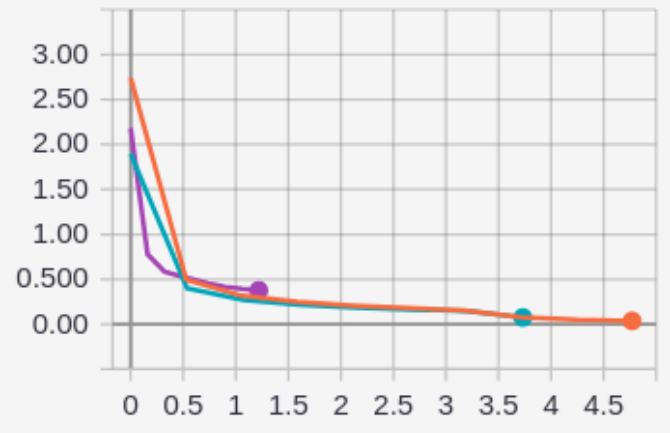
had soft-max activation. This model was then trained from scratch which means that the weights were randomly instantiated.

VGG-16 Pretrained Weights

The third model included pre-trained weights from ImageNet. Much of the online documentation as well as tutorials suggested that only the top layer be trained while freezing the VGG model. The result of which I found to be multi-digit testing accuracies of up to 20 percent. Leaving the weights frozen proved to be ineffective in producing viable results even when the top layers consisted of 4096 nodes each. Instead the model was initialized with the pre-trained weights and then trained until the validation loss stopped decreasing for 3 consecutive epochs. This was performed with Keras callback function named “Early_Stopping”.

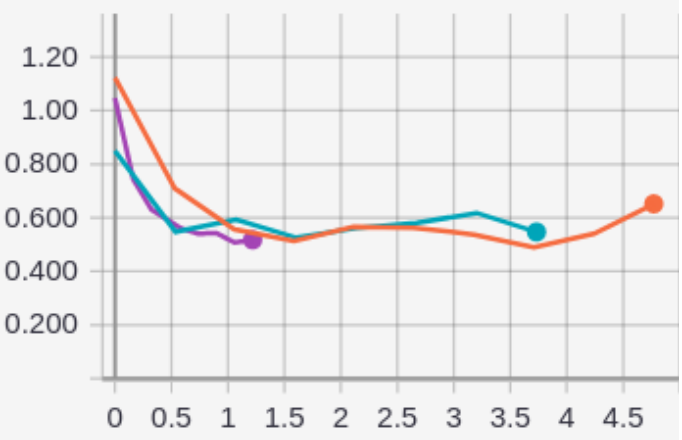
Results:

Training Loss



Time(Hours)

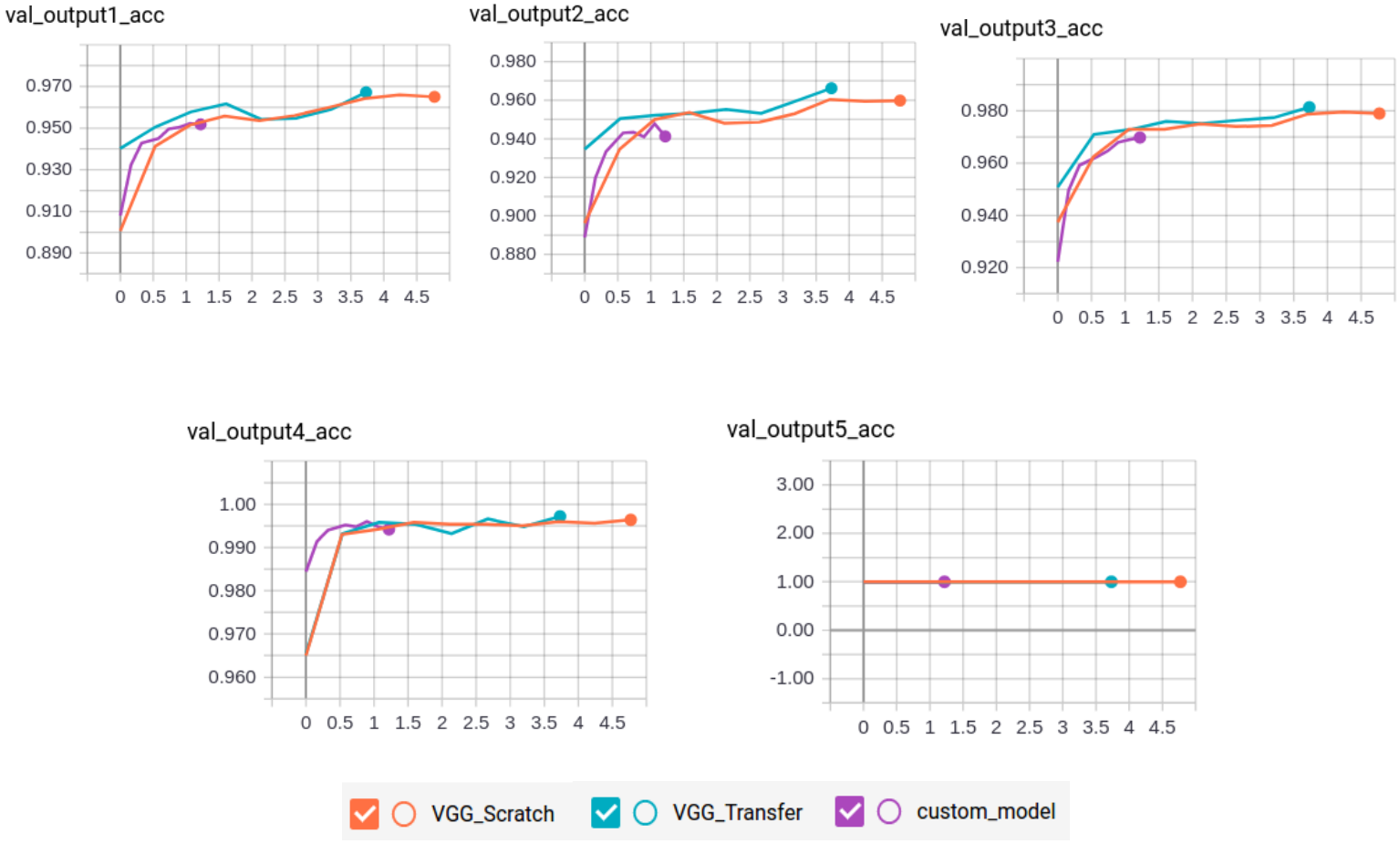
Validation Loss



Time(Hours)

The training data consisted of 230k multi-digit images and 78k negative examples. The validation set contained 5k positive samples and the testing set contained 13k positive samples. The models were all trained on 10 epochs with the possibility of early stopping when validation loss failed to improve. The custom model, VGG Transfer and VGG Scratch trained for 4, 7 and 9 epochs respectively. We see that the training loss consistently decreased while the validation loss was reduced less predictably. Since the VGG models contain 14 million parameters and our dataset is only 308k images, it is possible for our training loss to reach zero. In this case the validation losses bottomed at 0.546, 0.578 and 0.542 while the training losses went down to 0.46, 0.18 and 0.10 for Custom, Scratch and Transfer. We see that the transfer model starts with a significantly lower validation loss than it would have if the weights were randomly initialized. While it trained for less time, it still produced better results with the testing data.

Digit Accuracies versus Time in Hours



The digit accuracies for the validation set continue to increase with time which means that our model does not over-fit. The digit accuracies show that transfer learning does indeed save time. While all the weights were mutable, the starting position matters. The Transfer model produced digit accuracies comparable to or better than the model trained from scratch a whole hour sooner. This represented a 28% speedup. Starting with pre-trained weights produced tangible benefits during every step of the training process. At every epoch, the multi digit accuracies of the Transfer model would have performed better than the Scratch model with randomly initialized weights. The custom model however has 466k trainable parameters which allows it to fit the data faster in the early stages of training even with untrained weights. The following are the results of the model on the with-held testing set.

Testing Accuracies:

Custom Model: 89.86% **VGG16 Scratch:** 93.82 % **VGG16 Transfer:** 94.46%

The validation accuracies also tend to match the overall accuracy of our testing set. Goodfellow et al. produced 96.03% accuracy on sequence transcription and we can see that the VGG16 Transfer model produced a very comparable 94.46%! Their character level accuracies performed at 97.84% and our Transfer model performed at 97.05%. This number is reached if we average the last two digits which have near 100% classification. The average accuracy of the first three digits is 95.09% which is closer to what we see for our multi-digit classifier.

As we can see the pre-trained weights model found a better optima than training from scratch. For neural networks there can be many local optima and adjusting the learning rates is crucial to finding a nice balance between exploration and exploitation. For all three models the learning rates were reduced by a factor of 20% when a plateau in validation loss was reached using the Keras function “ReduceLROnPlateau”. The model training proved to be fickle with regards to the stochastic gradient descent learning rate and decay. Setting the learning rate correctly for each epoch proved to be a challenge. A high learning rate would not converge and a lower one would instead get stuck in local optima. This was solved by using an adaptive learning rate method called Adadelta. This is a dynamic learning rate that changes according the dimension data from the first order derivative. Some of the benefits include the following; “no manual setting of a learning rate., insensitive to hyperparameters, separate dynamic learning rate per-dimension, minimal computation over gradient descent, robust to large gradients, noise and architecture choice, and applicable in both local or distributed environments” (Zeiler). This system was developed by Matthew D. Zeiler in his paper “ADADELTA: AN ADAPTIVE LEARNING RATE METHOD” and proved invaluable in achieving the results in this paper.

Pipeline:

Classifying unconstrained natural photographs requires a pipeline that that pre-processes the images so that the neural network can classify the 48 by 48 BGR image correctly. First we must process the entire image by subtracting the mean and creating a pyramid. For each layer of the pyramid we divide the image by half. Three layers were created and then a 48 by 48 sliding window was applied to each. The stride was reduced on each subsequent layer since it represented a larger movement in the original image. The sliding window represented bounding boxes of sizes 48, 96 and 192 in the original image. These window images were passed to the CNN model of our choosing. In this case we used the VGG16 Transfer Model since it produced the best results. Below is the results of classifying an image with all of the possible window sizes.

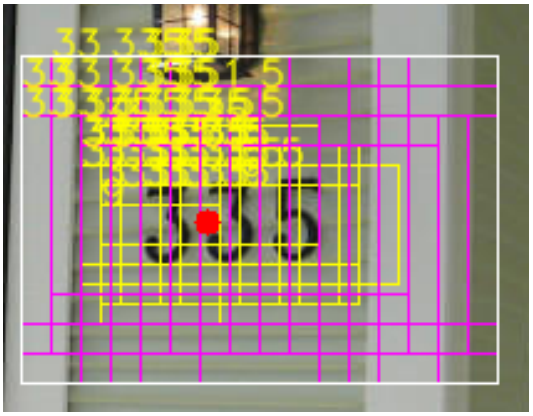


Figure 1. Bounding Box Size 96

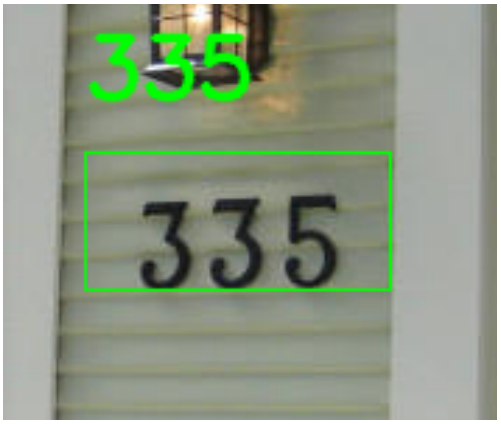


Figure 2. Bounding Box Size 48

The white box is the sum of the bounding boxes. My original attempt was to merge all rectangles to create one large one. The center of the box (marked by the red circle) was found by calculating the the sum of all rectangles and then finding the center. The scales were 48 pixels and 96 pixels in yellow and purple respectively. The large box produced by when searching for the center was too large and when rescaled down to the neural network size, it produced poor results. This can be seen as the white box in Figure 1 above. The second attempt consisted of picking the smallest pyramid size with which to make a bounding box. This produced the green box in Figure 2 which can also be seen as the sum of yellow squares in Figure 1. This created a much tighter grouping. With three possible bounding boxes, the smallest size was always chosen. This can cause problems when a box size does not capture all the digits as can be seen in the figure 3.

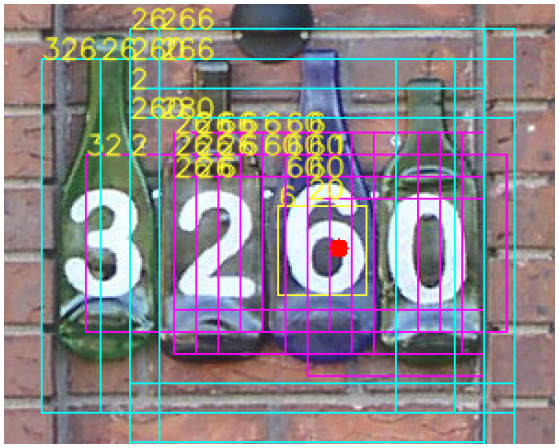


Figure 3. Small Bounding Box

Figure 4. Rotated Image

Figure 3 has three scales all color coded. The yellow box represents the smallest pyramid and the algorithm would have chosen it as the entire image. This would have produced a value of 6 because only one digit was captured. The second pyramid level would have properly classified 3 out of 4 of the digits since the first digit would have been partially cropped and the third size would have classified all four. In this scenario the the solution would have been to choose the second pyramid and scale each of the bounding boxes produced. In the pipeline implementation three scales were used; -10%, 0% and 10%.

Another issue was rotations. The neural network was not trained sufficiently on rotated images. One technique to make it more robust was to rotate all of the images in the training set. The approach I took was to rotate the bounding boxes between +/- 30, 45 and 60 degrees and pass it through the classifier. Then the algorithm found the predictions with the longest number length and returned the mode of those. This worked well for 4 or 5 digit numbers but produced some errors for one or two digit numbers since it was biased to choose any three digit number over a two digit number that was predicted multiple times. A change that could be made is to balance the mode of smaller numbers with the likelihood of larger numbers that may have been prematurely cropped. Finding the best possible prediction when the classifier returns multiple values is challenging. A more thorough implementation could have been created to decide which number was most correct. However, this decision process will result in some bias in the pipeline process. I chose to bias the predictions in favor of digit classifications of longer lengths.

The test video shows the classification of the number 12. In a few frames the number can be seen to change to 124, this is precisely because of this pipeline bias. The entire algorithm proved to be very robust as the video shows and overall the entire pipeline worked smoothly. Below are a few more correctly classified examples.

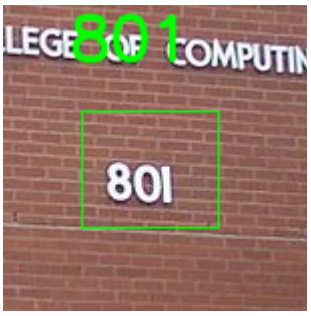


Figure 5.

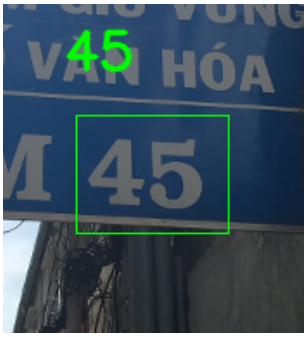


Figure 6.

Figure 7.

Test Video: https://youtu.be/7tW53Pm8fik

Works Cited:

Zeiler, Matthew D. (2012). "ADADELTA: An adaptive learning rate method". ArXiv:1212.5701 .

Goodfellow, Bulatov, Ibarz, Arnoud, Shet (2013). “Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks”, ArXiv:1312.6082