Classification and Detection with Convolutional Neural Networks using SVHN

Mingmei Niu
mniu30@gatech.edu

ABSTRACT:

Convolutional neural networks (ConvNets) were used to classify and detect multi-digit house numbers in photos using the Street View House Numbers (SVHN) dataset. VGG-like convolutional neural network layers were implemented using Keras with Tensorflow. SVHN photographs and house numbers were classified by using the trained model. Three classifiers were used to detect whether there were any numbers in the photos (training accuracy of 96% and testing accuracy of 95%), and then to classify the numbers from 0 to 9 (training accuracy around 94% and testing accuracy of 92%).

1. INTRODUCTION

Detecting and classifying multi-digit numbers from photos has been widely studied, due to its important applications in map-making, handwritten character recognition, vehicle plate recognition and more. A classic example is Google's street view photographs. Due to the large volume of images to classify, an automatic transcribing algorithm with high performance is needed to pinpoint the addresses of houses, stores, buildings, and more. Recognizing numbers from photographs is part of the optical character recognition (OCR) community. Even with all the extensive work done already, the arbitrary sequence of numbers from street views filled with all kinds of noise, lighting, angles, rotations, and occlusions make the task still quite challenging.

Convolutional networks (ConvNets) have been studied extensively in recent decades. CNN are neural networks with clusters of neurons having different parameters. There are multiple filtering layers within each layer applying a transformation to the vector input followed by an elementwise non-linearity. The CNN method uses discrete convolution rather than a fully general matrix multiplication, resulting in better computational efficiency and scalability. Image CNN normally use a pooling layer that summarizes the activation of adjacent filters with a single response. Pooling layers summarizes the response of groups of units with a function like maximum, mean, or L2 norm. The pooling layers improve the robustness of the networks to small translations of the input.

In this project, we are asked to detect and recognize the digits in a street view house image through 3 ConvNets methods: Convolutional Neural Network (CNN), VGG16 with pretrained weights, and

VGG16 from scratch. Provided as training data set, the Street View House Number (SVHN) data sets contains around 33,402 images as training data, and 13,068 images for testing.

2. METHODS

2.1 Data Processing

The SVHN dataset has two formats: original images with character level bounding boxes and MNIST-like 32-by-32 images centered around a single character. This project uses the second one for training plus the non-number image cropped from the original images. There are 73,257 digits for training, 26,032 digits for testing, and 11 labels, of which numbers are labeled as their values and non-numbers are labeled as 10.

Fig. 1: MNIST-like 32-by-32 images centered around a single character (many of the images do contain some distractors at the sides). Source: http://ufldl.stanford.edu/housenumbers/

2.2 Models

There are three different algorithms have been used for the digit recognition training process: a custom CNN, VGG16 with pretrained weight, and VGG16.

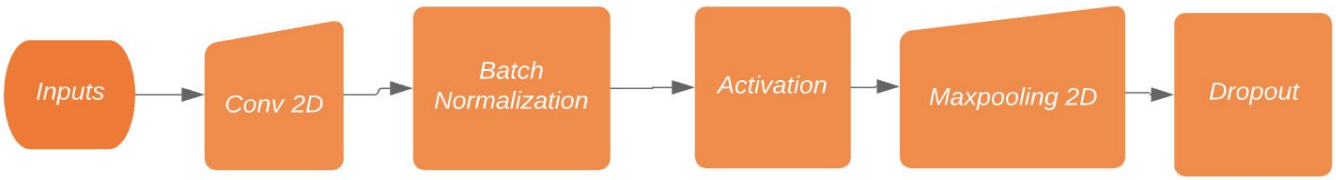

Figure 2: Structure of hidden layer for custom CNN model.

First, to create a custom CNN model, I created a model similar to the model in the paper by Goodfellow et al. [2] using the SVHN dataset. My model contains 7 convolutional hidden layers and 2 densely connected layers. As shown in Figure 2, the hidden layer contains a 2D convolution, a batch normalization, a rectified linear unit (ReLU) activation, a 2D max pooling process, and a dropout. The first 3 hidden layers have a filter size of 48 and a 3x3 kernel size for the 2D convolution. The max pooling has a 2x2 pool size and 1x1 strides, except that the first layer has 2x2 strides. The dropout rate for each hidden layer is 0.2. The 4th to 6th hidden layers have filer size 64, and every other parameters are the same except the 4th hidden layer has a 2x2 strides. The 7th hidden layer has filter size 128, and

2x2 strides. After the 7 hidden layer 2 dense layer of size 64 is added, and finally an output of 11 features using softmax as activation function.

Secondly, for the VGG-16 model, two size 256 dense layers are added after VGG, and then a dropout at 0.2 followed by the output of 11 features using softmax.

Lastly, for the VGG-16 model with pretrained weights, it is followed by two size 128 dense layers and an output of 11 features using softmax.

2.3 Detection Method

To detect the digits in the photo, different sizes of sliding windows are used to identify where there are numbers in the window. The detection contains three major stages: sliding windows with large steps to locate the sequence of digits to crop the image around the area, sliding window at smaller steps to find the potential digits and merging the boxes that contains the same digit.

At the first stage, the window size starts at half of the original image size, and it slides over the image with a step of half of the window size. If there is a window has confidence larger than 99% by the trained classifier, then it will crop the image around this area by extending the height twice and the width five times, enter into second stage; otherwise shrink the window size two thirds, this is repeated until the window size is less than 32 pixels. At the second stage, the window size starts at the size of the cropped image, it slides over the cropped image with a step that is a quarter of the window size. If there is a window with confidence larger than 85% by the trained classifier, then this window records it and then window size is shrunk by three quarters. This is repeated four times. At third stage, iterate through the recorded windows. If two windows have the same number, and are overlapping each other at least half of their area, then merge the two window to their average. This was repeated until there were no overlapping windows. After the above three stages, the numbers will be marked on the image.

4. RESULTS

Figure 3: examples of what did not work.

The detection and classification works and fails in different scenarios. Sometimes the numbers were not identified or they were identified incorrectly. Examples of what did not work and what did work are found in Figures 3 and 4.

Figure 4: Examples of what did work.

There are many false positives during actual detection, especially then the window happen to slide between two digits. Adding more non-number images to the data set should help with this issue. There are also issues with using images of different resolution, as the lower ones seem to have issues. Keeping the resolution consistent should help as well. Sometimes the digits that are set too closely together would cause issues, so in the future the window size can be reduced.

Next, I plotted the loss and accuracy against epochs for each model in Figures 5 and 6 below. The designed model and pre-trained VGG-16 have similar performance, while the VGG-16 has lower accuracy. The accuracy of the trained model looks to be promising (94%).

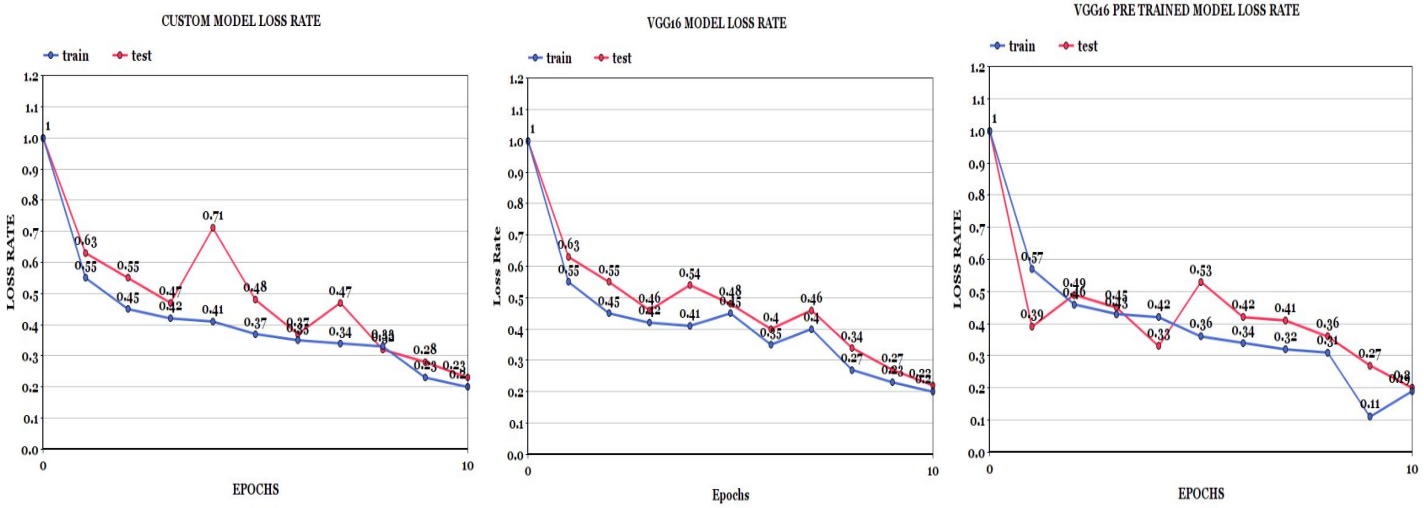

Figure 5: Loss rates for custom model (left), VGG16 Model (middle), and VGG16 pre-trained weights (right).

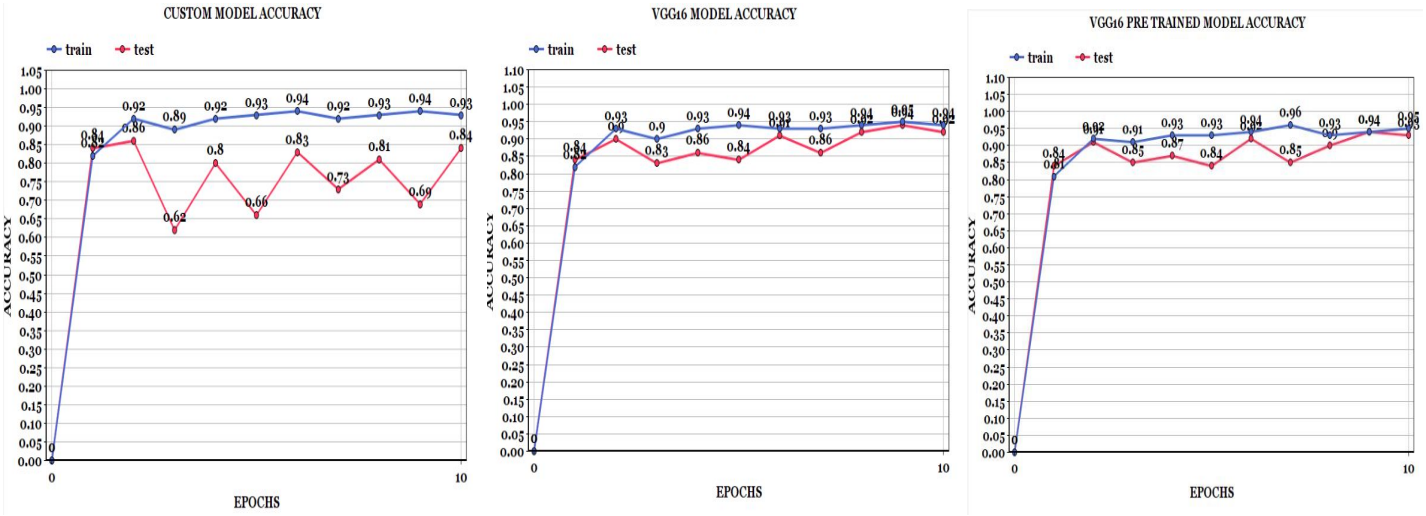

Figure 6: Accuracy rates for custom model (left), VGG16 Model (middle), and VGG16 pre-trained weights (right).

However, the actual result of the detection on real images suffers severely from too many false positives. The model seems to identify a lot more digits than there actually are, see Figure 3. Also, the letter "M" is often classified as a "4". I have tried different threshold in the detection algorithms, it turns out that if I want less false positives then there are many digits which will not be detected (true negatives). The model definitely has room for improvement.

The current parameters are the best in terms of balancing the false positives and true negatives. The sliding window method is working slow due to the algorithm needs to slide the window over an image multiple times. Even after the aggressive stop and cut method as described in the method section, the algorithm still take several seconds or even a minute to run on an image.

Video of digit detection can be found here: https://youtu.be/FvqIEYT7Dl0 (alternate link is https://www.dropbox.com/s/wyqhklcxwu5n74e/videoplaybackfinal.mp4?dl=0)

The video of the entire project can be found here: https://youtu.be/lhv3fSYLgOs (alternate link is https://www.dropbox.com/s/5g7cqnlyhjgpxmh/VID_20191204_124421.mp4?dl=0)

5 CONCLUSIONS AND FUTURE WORK

As mentioned previously, my CNN model shows 95.59% for the training data set and 89.98% for the test data set. While as published by Goodfellow et al. [2] using CNN, they get around 96% accuracy for the test data. And also Netzer et al. [4] show good performance in the same data set. Even though my model may not do a perfect job, but it still gives a reasonable prediction accuracy compared to the state of the art.

As for the future work, I would improve the methods with some more modification on data processing. By calculating the gradient of the component in the image, I may get a smaller range that may contain digits. After this, applying the additional VGG model I used in the current model for the accurate digit location detection. As for the digit value detection, I will get multiple resize results for the detection and

compare among each resized section, and only show the command digit detection. By modifying my current model, I'm positive it would give a better digit detection results.

The project may be improved with both classification and detection. For classification, a larger amount of data set, especially increasing the data for non-number image may significantly reduce the number of false positives. At detection, using segmentation instead of sliding window may help to reduce the running time. Another possible choice is to train a YOLO algorithm which is known to be fast at detecting objects in the image[7]. The model in Goodfellow's paper [2] for detecting a sequence of numbers has an accuracy of 96%. By trying to use their CNN model, my custom designed model achieved nearly 94% on the SVHN data set. However, the accuracy on real life image is much lower. This suggests that that the model may be overfitted. An unsupervised way of learning on the same task is done by paper[1], their best accuracy was 90.6% on the data set.

REFERENCES

1. K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics, 36(4):193–202*, 1980.
2. IJ Goodfellow, Y. Bulatov, J. Ibarz, and S. Arnoud. Multi-digit number recognition from street view imagery using deep convolutional neural networks. *arXiv preprint arXiv:1312.6082*, 2013.
3. Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.
4. Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.
5. Ciresan, D.C., Meier, U., Masci, J., Gambardella, L. M., and Schmidhuber, high performance convolutional neural networks for image classification. In *IJCAI, pp. 1237–1242*, 2011.
6. K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR, abs/1409.1556*, 2014.
7. Redmon J, Divvala S, Girshick R, et al. You only look once: Unified, real-time object detection[C]. *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2016: 779-788.