Classification and Detection with Convolutional Neural Networks

Chuqiao Dong
chdong@gatech.edu

1 Overview

The total work flow is shown in Figure 1.

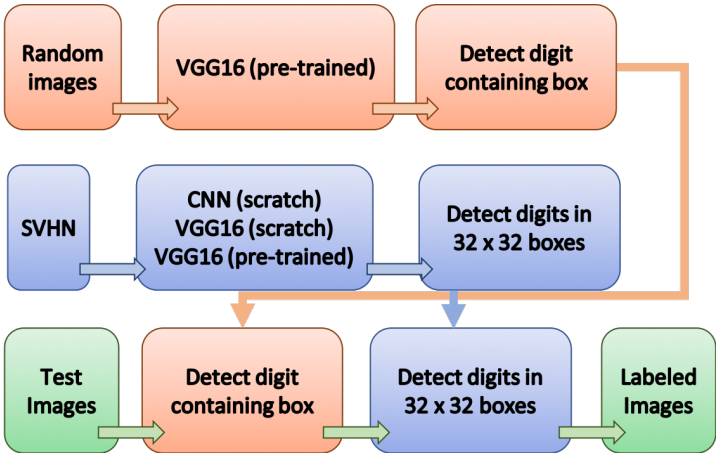

Figure 1: Total Flow Chart

In this work, we are asked to detect and recognize the digits in a street view house image through various Neural Network methods. In this work, I have inplemented three different algorithms for the digit recognition training process: Convolutional Neural Netowrk (CNN) [1, 2], VGG16 with pretrained weights [3], and VGG16 from scratch. Provided as training data set, the Stree View House Number (SVHN) contains around 73,257 as training data, and 26,032 for testing purpose. Data manipulate details will be stated in the Data Preparation section. By applying different algorithms, the prediction accuracy as 95.59% (CNN), 94.43% (VGG16 pre-trained), and 92.64% (VGG16 scratch). VGG16 with pre-trained weights performed best, thus this would be used for the real image detection. And in the real image detection process, I firstly use some random images with some containing digits and the rest not through another VGG16 model, so that my model would successively learn whether and where

the digits are in an image, then only within this digit appeared area, apply the pretrained VGG16 in first step for any single digit detection. Details about this part would be talked in the Model section. This process show a good prediction in the 5 images show in the Results section and mostly good results in the video.

2 Data Preparation

The initial training data set from SVHN contains digits 1-10 in 32 x 32 x 3 format. Since I would like to detect digit by digit, thus 10 is not suitable for this detection. I changed all the 10s to be 0s so that now I have 0-9 digits. And for the CNN algorithm only, I would like the color channel does not have influence on the detection, so that I changed all the images into gray scale. After changes the images into gray scale, I ploted the digit distribution among the training and test data set shown in Figure 2.

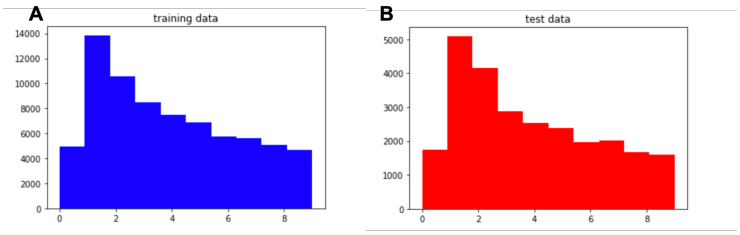

Figure 2: Digit Histogram for training (A) and test (B) data

From the histogram, it is clear the digit distribution is not too biased. In addition, normalization of the raw data would make the training and test data gives a better prediction. So after changing the three channeled image, I used the normalization mathod to get the normalized data for the training. And I also have the training and test data set shaffled before put into the training algorithms. In conclution, I have the normalized data set in gray scale for the CNN training and normalized data set in RGB channel for VGG16 related algorithms.

3 Model

This part has been shown in Figure 1 blue section:

There are three different algorithms have been used for the digit recognition traing process: CNN, VGG16 with pretrained weight, VGG16. In CNN, the architecture has been shown in Figure 3 A, and the architecture for VGG been shown in 3 B.

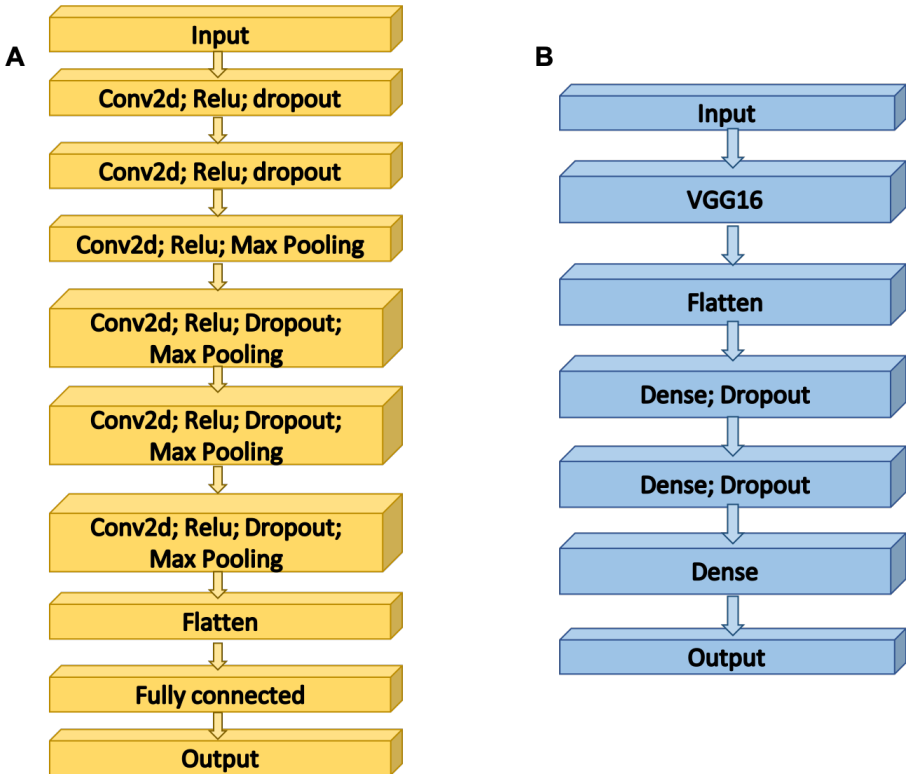

Figure 3: (A) Architecture for CNN. (B) Architecture for VGG.

Tensorflow (Session) was used to build the CNN architecture and Tensorflow (Keras) was used for VGG related models. Learning rate for CNN algorithm is chosen to be 0.0005. A smaller value like 0.0001 has been tried, but not much approve of either the accuracy or the loss has been shown. For the sake of effeciency, a learning rate of 0.0005 has been chosen. In the CNN algorithm, a 0.8/0.2 data seperation has been make to distinguish the training/validation data set. And the epoch number here is 50. As can be seen from Figure 4 A, when the epoch value reaches 10 and above, the accuracy of the validation data does not show any significant increase but the accuracy of training data climbing simultaneously. This shows when epoch reaches 10 and above, the model begin to tune itself fitting for the training data thus may become overfitting if the epoch number keep increasing.

As for the VGG16 related algorithm, as has been shown in 3 B, the main architecture has been applyied by the existing VGG16 model. The only difference between pretrain weighted VGG16 and scratched VGG16 model is whether to adopt the pretrained weights. Former one was applying the pretrained weight by setting 'weights='imagenet" and adding one flatten layer followed by three dense layers on top of the VGG16 model. In this way, the VGG16 model has been transfered to my digit detection situation. On the other hand, the

scratched VGG16 model also take the similar archtecture, but with the setting 'weights=None' to avoid adopting the pretrained weights. On the top of VGG16 architecture, I also add one flatten layer followed by three dense layers. Shown in Figure 4 B and C, it is clear that the VGG16 model transfered with pretrained weights makes better prediction than the one with random initial weights. This might because the training data set in this work may still not be enough to get a 'best' performance in finding the 'perfect' weights.

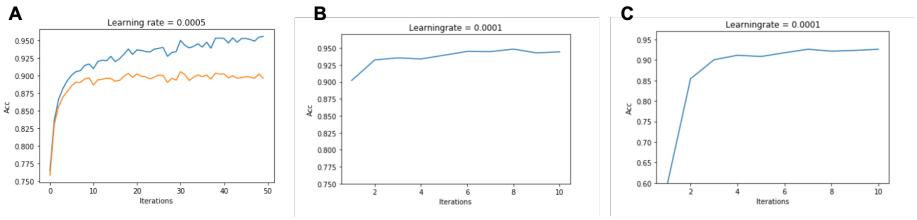

Figure 4: (A) Accuracy as a function of epoch for CNN. (B) Accuracy as a function of epoch for VGG with pretrained weights. (C) Accuracy as a function of epoch for VGG from scratch. Blue line for training data ang orange for test set.

This part has been shown in Figure 1 orange section:

In the real image detection problem, the input image may not be a certain 32 x 32 image with single digit in it. Thus, it is very important to train the model to learn where to find the digits. Thus, I used 10 randomly picked images (5 with digits and 5 not), sliced into 120 x 120 subimages and concatenate together getting 36,112 subimages totally. These images has been used to tell the model what is the digits look like so that when given a raw image, this new model can tell you where may contains the digits, then applying the previous trained digit detection model, I can get an idea which digits are shown in this image.

After training the data set for the detection of digits in any 32 x 32 images, I also applied VGG16 algorithm for the detection of the positions where may contains these digits (additional VGG model). By applying the similar architecture with base VGG16 model followed by one flatten layer and three dense ones, this model can do quit a good job in detecting the digit positions.

This part has been shown in Figure 1 green section:

When given a new image, the code would initially apply a convolution of a sliding window (100 x 100) through this image to detect where might contains the digits by calling model trained in orange section. Once getting the highest possibility of the digits location, another convolution of a small sliding window (32 x 32) will be applied to detect which digit this may belonging to by calling the model trained in blue section. And after this process, I would get a reasonable preduction of what are the digits in this image.

However, the digits may not always be of the same size among different images. Even though the additional VGG model (orange section) would do a good job in finding the digit locations, there is still cases the digits themselves

are larger than the sliding window of this additional VGG detection process. In this situations, I also applied the resize of the initial image to be 70% of the initial one so that a 100 x 100 sliding window might be large enough to detect the digit locations. If still not, I also include another layer of this resizing and detection to get a reasonable results.

4 Results

As has been stated in the Model section, the overall model was trained in two levels: detection of the digits position and detection of each digit. It shows a good detection of thefive images provided in Figure 5.

And the video detection can be find through link:
https://www.dropbox.com/s/w9oahxx9e0ev0vp/output.mp4?dl=0
Video presentation can be find through link:
https://www.dropbox.com/s/q5gf0xuawraeagk/presentation.mp4?dl=0

Figure 5: (I-V) Five images for the digit detection. (VI) Failure detection from video.

My method works pretty well in these five images shown previously, which contains different scales/orientation of the image, different location/font of the digits, and even contains lighting and noise. However, there are still some cases shown in the video that this detection failed. The main reason for the failure is that the detected digit location sometimes may not contain all of the digits in the images that may lead to some miss-detection of digits. And there are some other situations when the digit location is right but the digit itself is too large for the 32 x 32 sliding window size that may lead to the detection of partial of the digits. An example of this is that when read digit '5' only partially, it is

very possible to detect it as digit '1'. And this is exactly what happened in the failure images shown in the video.

As for the statistics analysis of my method, it did a good job in this point. During the 529 frames detection video, there are only around 25 frames show a failure (less than 5%) and my model is also very stable in the sake of repeatable performance for each time of use.

5 Discussion and Future Work

As I mentioned previously, my CNN structure show 95.59% for the training data set and 89.98% for the test data set. While as published by Goodfellow et al. [1] using CNN, they get around 96% of accuracy for the test data. And also Netzer et al. [2] show good performance in the same data set. Eventhough my model may not do a perfect job, but it still gives a reasonable prediction accuracy comapring to the state of art.

As for the future work, I would improve the methods with some more modification in the data prepossing part. 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 I used in the current model for the accurate digit location detection. As for the figit value detection, I'm planning to get multiple resize results for the detction and compare among each resized section, and only show the command digit detection. By modificate my current model, I'm positive it would give a better digit detection results.
For source code, please see attached source code file.

References

[1] Ian J Goodfellow et al. "Multi-digit number recognition from street view imagery using deep convolutional neural networks". In: *arXiv preprint arXiv:1312.6082* (2013).

[2] Yuval Netzer et al. "Reading digits in natural images with unsupervised feature learning". In: (2011).

[3] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).