Road Traffic Sign Recognition

Bo Peng

December 15, 2017

Abstract

Road traffic sign recognition plays an important role in autonomous driving and drivers assistance system. In this project, I designed an artificial neural network composed of three convolutional layers and two fully connected layers to classify different types of road traffic signs. Results showed that the neural network used in this project was able to achieve the accuracy of about 90% for testing data.

1 Introduction

Traffic sign recognition has been widely explored for applications including highway maintenance, driver support system, and intelligent autonomous vehicles. It is still challenging to develop efficient algorithms for accurate recognition of different traffic signs because traffic sign images are usually captured in various lighting conditions with different distortion, resolution, etc.

The problem I am working on is to design a robust neural network for automatic classification of traffic signs. More specifically, the major goal is to directly predict the type of any input traffic sign image using a pre-trained neural network without over-fitting. More specifically in this project, convolutional neural network was used at first to extract key features from traffic sign images after preprocessing. And then I flattened the output of the last convolutional layer as the input of the following fully connected layers, which is in fact a multi-layer perceptron network for pattern classification. It should be noted that we can try any other classifier that is applicable for solving non-linear classification problems (e.g., Support Vector Machine). Finally, we obtained the probability of the input image being each type of traffic signs. In order to test the adaptability of the developed neural network, two standard traffic sign datasets from Belgian and German were used for training and testing.

2 Data Preparation

In this project, standard Belgian Traffic Signs (BTS) and German Traffic Signs (GTS) were used.

The BTS dataset [1] consists of 4,575 training and 2,520 testing traffic sign images, both of which contain 62 types as shown in Figure 1. Images with the same type are put into

one folder with the folder name showing the corresponding label of each type of traffic signs. The size of each image varies a lot, composed of red, green, blue channels.

Similarly, the GTS dataset [2] contains 43 categories of traffic signs but with larger data volume. A total of 39,209 training images and 12,630 testing images were collected, which contribute to building a more robust neural network. Sample images of each type are shown in Figure 2.

Figure 1: BTS Sample traffic sign images for each class.

Figure 2: GTS Sample traffic sign images for each class.

3 Method

With available training and testing data, the method utilized in this project can be concluded into two steps, data preprocessing and neural network design.

3.1 Data Preprocessing

Data preprocessing is necessary since traffic sign images in both BTS and GTS dataset were acquired in different lighting environment with various distortion and resolution as demonstrated in Figure 1 and Figure 2.

Considering that neural network generally requires inputs with the same dimensions, all the images in BTS and GTS were resized into the same spatial dimension (e.g., 64×64). In addition, normalization was conducted over the greyscale of each image for training a more robust network since every image was captured in different lighting condition.

Finally, to avoid over-fitting, the training data were divided into two subsets, one training data subset and the other validation data subset. The validation data were mainly used for monitoring the training process, showing when the network would be over-fitted. The size of the validation data in this project was set to be 20% of the original training data. As a result, the remaining 80% were used as training data.

3.2 Neural Network Design

In this project, two types of neural networks were used for building the traffic sign recognition model, that is, convolutional neural network and the fully connected neural network [3].

At first, I designed 3 convolutioanl layers. More specifically, the filter size for all the 3 convolutional layers is 3×3. But I have different number of filters for each layer, the 1^{st} layer with 32 filters, the 2^{nd} one with 32 filters, and the 3^{rd} one with 64 filters. The strides for all the filters were set to be [1, 1, 1, 1] corresponding to each dimension, i.e., input image, row, column, and band. It should be noted that each convolutional layer is followed by a max-pooling layer with the size of 2×2 and a ReLu layer.

Second, between the convolutional layers and the fully connected layers, one more step is required to flatten the output of the last convolutional layer.

Finally, I used two fully connected layers with 128 neurons respectively performing as a non-linear classifier to output the probability of each input image being each type of traffic signs. Similarly, the ReLu layer was added to the output of the last fully connected layer.

The architecture of the developed neural network is illustrated in Figure 3.

To train this neural network, the initialization of weights and biases was implemented by taking normal distributions (with zero mean and small variance), which is very prevalent. Moreover, three indicators were used for training including training accuracy, validation accuracy, and validation loss. The accuracy is defined as the percentage of input images being correctly recognized. And the loss is computed as a result of the softmax cross entropy between truth labels and predictions. With respect to the optimization method, Adam Optimizer was used for computing the gradients and optimizing the weights and biases. The goal of the optimization is to minimize the loss with certain learning rate (e.g., 0.01).

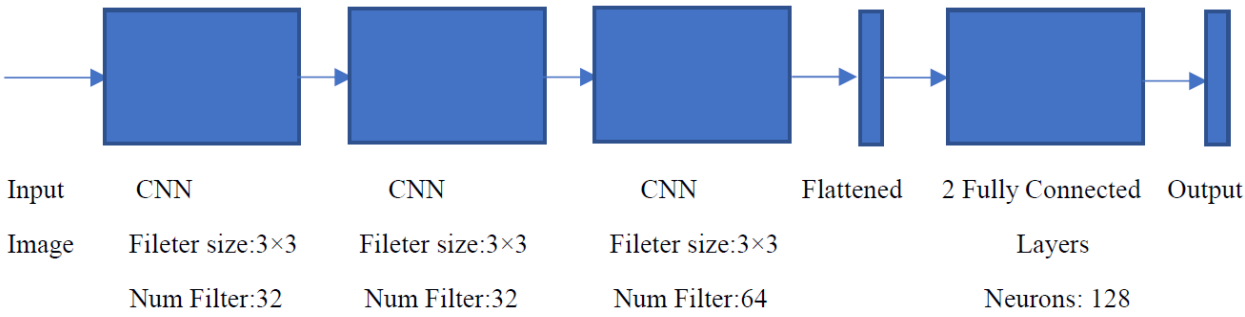

Figure 3: Architecture of the neural network.

It is worth noting that training with mini-batch as input for this network would achieve better performance [4].

4 Experimental Results and Discussion

The experiments were implemented with Python and Tensorflow. The computing environment is Ubuntu 16.04 LTS with Intel Core i5-7300 CPU of 2.6 GHz × 4. Both BTS and GTS datasets were used for neural network training and testing. In order to get better performance, the learning rate for training was set to a small value, 0.001.

4.1 BTS Dataset

Images in BTS dataset were resized to 64×64 and normalized before feeding into the neural network. In order to evaluate the impact of the batch size on testing accuracy, batch size of 16, 32, and 64 were tested respectively. Taking batch size of 16 as an example, the training process is shown as Figure 4. This training process is monitored and controlled

```
Epoch: 1...Train acc:  68.8%...Validation acc:  81.2%...Validation loss: 1.037
Epoch: 2...Train acc:  87.5%...Validation acc:  68.8%...Validation loss: 1.011
Epoch: 3...Train acc:  81.2%...Validation acc:  81.2%...Validation loss: 0.759
Epoch: 4...Train acc: 100.0%...Validation acc:  87.5%...Validation loss: 0.584
Epoch: 5...Train acc: 100.0%...Validation acc:  87.5%...Validation loss: 1.069
Epoch: 6...Train acc:  93.8%...Validation acc:  93.8%...Validation loss: 0.372
```

Figure 4: Training process.

by three indicators including training accuracy, validation accuracy, and validation loss. If one of these indicators continues going down, the model is likely to be over-fitted. As can be learned from Figure 4, the 6^{th} epoch seems to be over-fitting, which means the training process should be stopped.

Figure 5 to Figure 7 show the change of training accuracy, validation accuracy, and testing accuracy during the training process epoch by epoch when using different batch sizes (i.e., 16, 32, and 64).

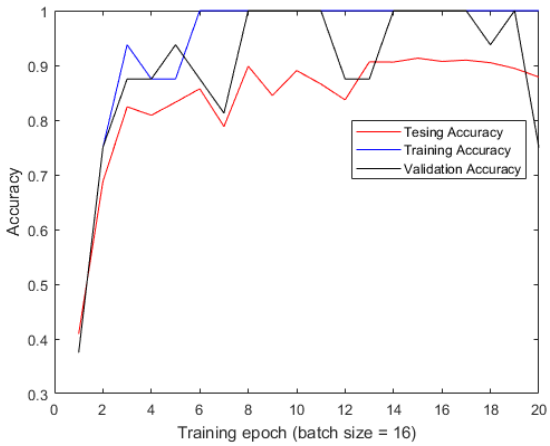

Figure 5: Accuracy during the training process (Batch size = 16).

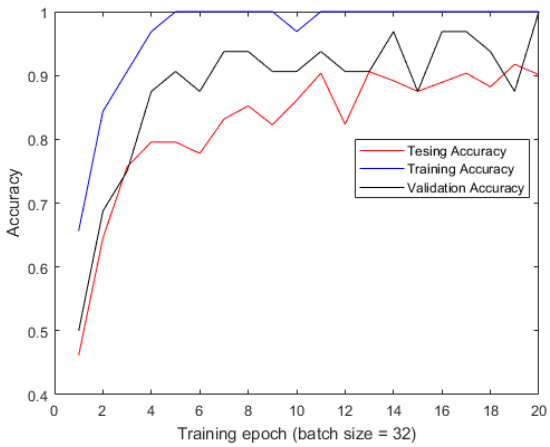

Figure 6: Accuracy during the training process (Batch size = 32).

As shown in Figure 5 to Figure 7, when both training and validation accuracy increase or approach to 100%, the model could acquire higher testing accuracy. However, if only training accuracy increases while validation decreases, the testing accuracy would probably go down due to the issue of over-fitting. As a result, it is a good practice to control the training process by monitoring the validation accuracy and loss at the same time.

In addition, the relationship between testing accuracy and the batch size is illustrated in Figure 8. The testing accuracy is relatively higher if the network is trained with smaller batch size. However, the time consumed for training the network with smaller batch size would be greater than that with a larger batch size.

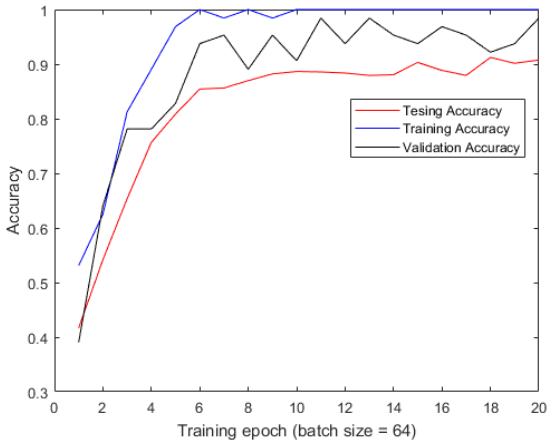

Figure 7: Accuracy during the training process (Batch size = 64).

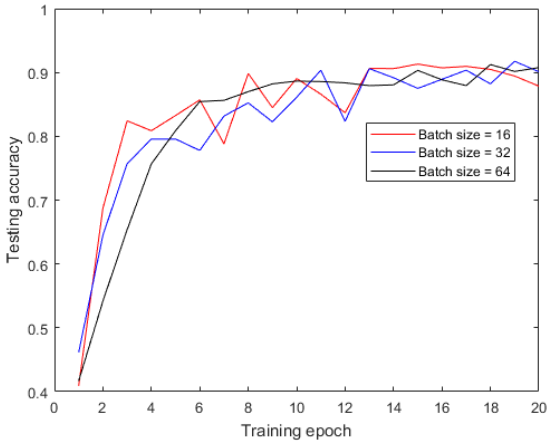

Figure 8: Testing accuracy with different batch size for BTS dataset.

Figure 9 shows the prediction of some sample testing images, with green color corresponding to correct recognition and red color indicating false prediction.

4.2 GTS Dataset

In analogy with experiments conducted on BTS dataset, images from GTS dataset with reshaped size of 32×32 were used for training and testing the developed neural network. Figure 10 shows how testing accuracy changes with training epochs and batch sizes. We can draw similar conclusion as the one corresponding to BTS dataset. That is, smaller batch size contributes to higher testing accuracy and better network adaptability.

It is worth noting that the curves in Figure 10 are much smoother than that in Figure 8. Due to the fact that GTS data are much bigger than BTS data, the neural network was trained much more in each epoch. Thus the neural network trained by GTS data is more

Figure 9: Prediction on sample traffic sign images in BTS testing data.

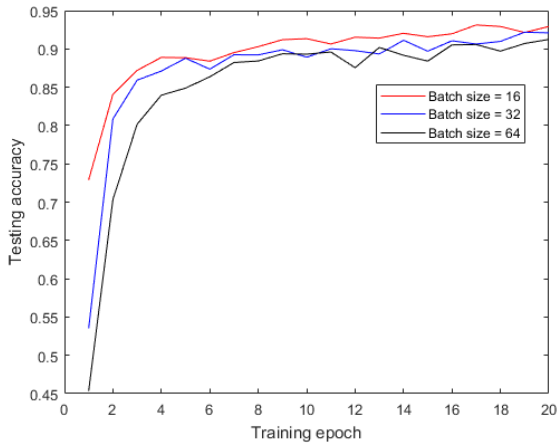

Figure 10: Testing accuracy with different batch size for GTS dataset.

robust and stable than that by BTS data. That is why we would like to use big data to train a neural network for better performance.

5 Conclusion

Based on the experimental results, following conclusions can be drawn:

First of all, data preprocessing plays an important role in neural network training. To get a well trained network, we need to know more about the data, and then decide what specific preprocessing method can be applicable, such as image normalization and transformation in this project.

Second, big data contributes to improving the stability of the neural network. In order to get a well trained and robust neural network, the first step is to collect a large number of data and clean those data. In addition, mini-batch input fed into the neural network is good for optimization, and thus improving the testing accuracy.

Third, it is very import to monitor the validation accuracy and loss after each epoch. This is a good practice since it helps control the training process and thus avoid over-fitting of the neural network.

References

[1] M. Markus, et al. Traffic sign recognitionHow far are we from the solution? *Neural Networks (IJCNN), The 2013 International Joint Conference on. IEEE*, 2013.

[2] German Traffic Signs (GTS) dataset. http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset

[3] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, *MIT Press*, 2016.

[4] A. Geron, Hands-On Machine Learning with Scikit-Learn and Tensorflow, *OReilly Media Pub.*, March 2017.