German Traffic Sign ClassifierProject

Author: Aneeq Mahmood

Email: Aneeq.sdc@gmail.com

Subject: Writeup for CarND- Term1- P2

The goal of this work has been to identify German traffic signs suing a convolutional neural network (CNN) approach. The python code for this project is available on github at: https://github.com/amahd/German-Traffic-Sign-Classifier, which contains the jupyter notebook containing all the source code

The subgoals of this work include:

- Load the data set available online
- Explore, summarize and visualize the data set
- Design, train and test a model architecture
- Use the model to make predictions on new images
- Analyze the softmax probabilities of the new images
- Summarize the results with a written report.

These subgoals are discussed individual below.

German Traffic Signs- Data Visualization

-

Figure 1: German traffic Signs

As a start, the image set of the German traffic signs (GTSs) is available online at http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset.

However, a modified 32 by 32 bit RGB collection of images has been provided in the project repository as a pickled dataset. In the first cell of the project jupyter book. The data is ‘unpickled’ and three datasets namely training, validation and test sets are obtained. There are a total of 51389 images from 43 classes. The image breakdown is:

Number of training samples = 34799

Number of validation samples = 4410

Number of testing samples = 12630

Each image is RGB with dimensions (32, 32, 3).

Number of classes = 43.

Figure 2 shows a random image from the training simple, and also shows the histogram of all the image types. The latter shows how many images per class are available in the training set. This data visualization has been done in cell 2 of the jupyter book.

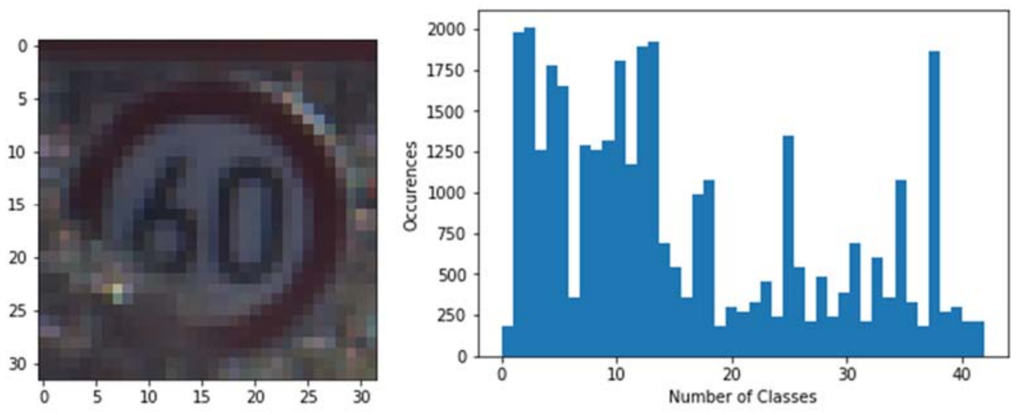


Figure 2: (a) A sample image from the training data (b) Histogram of the training data

The histogram clearly shows an imbalance of number of imager per class. The direct influence of this on the CNN performance will be that images with less frequency will not be correctly detected as compared to the images with high frequency. This will lead to an overall drop in accuracy performance of the CNN.

Data Pre-processing (Cell 3a):

As this training dataset (TD) contains RGB image, there is a case to convert it to grayscale, as it is easier to detect edges as observed in Project 1 of the term. However, traffic signs' color often give a strong indication and training the CNN to detect colors will be beneficial. Hence, the TD is left as RGB. There is a case to change the dataset to also contain a grayscale layer to aid training on both edges and colors, which will change the image dimensions to (32,32,4). However, this has not been implemented. In fact, the only pre-processing done to the images is to limit pixel values each of R,G, and B channel between -1 and 1. This is done by subtracting 128 from each pixel and dividing by 128. Figure 3 shows before and after effects of such a normalization.

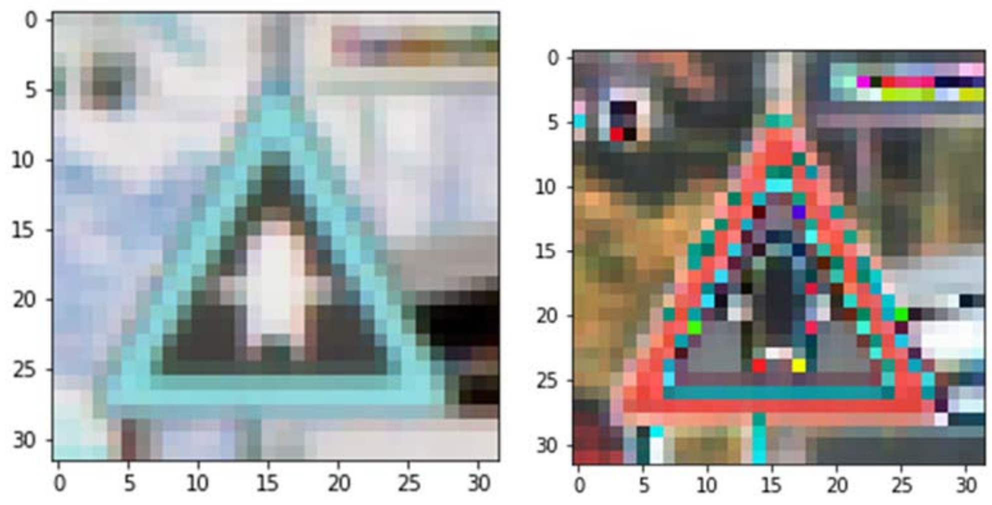


Figure 3: Before and after normalization

The CNN Model (Cell 3c)

The model used to for training GTS is the famous LeNet model [1]. The reason for choosing this model has been its simplicity, and also the fact that it has worked well with other datasets such MNIST. Simpler models based on deep neural networks with one or two layers might have resulted in more hit and trial which would have increased the hyper-parameters list. Deeper networks such as AlexNet [2] would have been too complicated for TD with relatively few classes. The architecture of the LeNet model is shown in Table 1.

Table 1: CNN architecture

Layer 1 Convolutional	
Input	32x32x3 RGB
Kernel	5 x 5 x 6
Output of filtering	28 x28 x 6
Activation	Relu
Pooling	2 x2 with stride of 2
Layer 2 - Convolutional	
Input	14x14 x 6
Kernel	5 x 5 x 16
Output of filtering	10 x10 x 16
Activation	Relu
Pooling	2 x2 with stride of 2
Layer 3- Connected	
Input	5 x 5 x 16 (flattened to 400)
Output	120
Activation	Relu
Layer 3- Connected	
Input	120
Output	84
Activation	Relu
Output Layer	
Output	10
Activation	Relu

Training the Model (Cell 3f)

Choice of optimizer: Adam optimizer

This choice has been made to reduce the hyperparameters as learning rate for the Adam optimizer can be left untouched. Other optimizers such as gradient descent or stochastic gradient descent would affect performance with learning rate and hence will require more frequent iterations over the TD.

The number of epochs has been set to 30. This has been done to reduce the whole training time. Through a hit and trial of training, it has been shown that these number of epochs give a reasonable idea of convergence, and as shown later can also indicate idea about overfitting. Batch sizes has been left open to investigate at this moment. Figure 4, 5, 6 show the accuracy and loss for training and validation datasets, using batch sizes of 64,128, and 256. What can be seen form these figures it that all three batch sizes lead to a validation accuracy above 95%. However, there are obvious signs of overfitting, as accuracies for all batch sizes have sharp decreases and fluctuations.

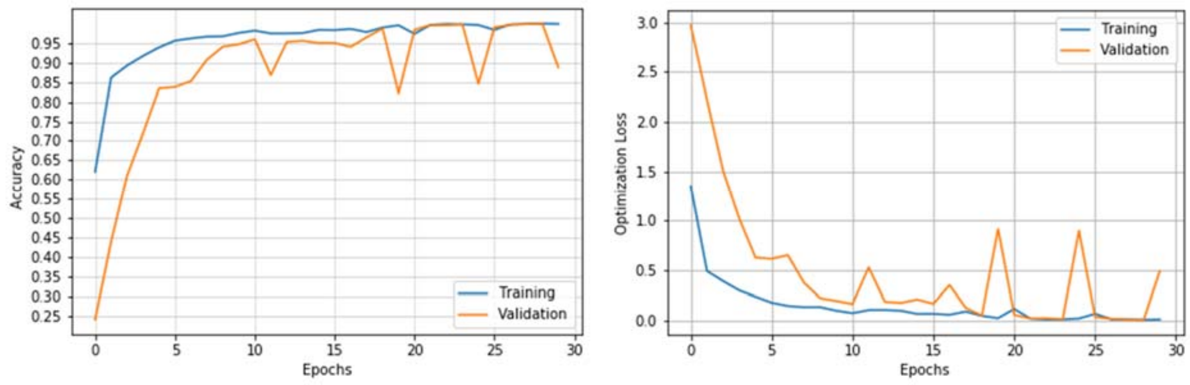


Figure 4: Accuracy and loss plots for batch size 64

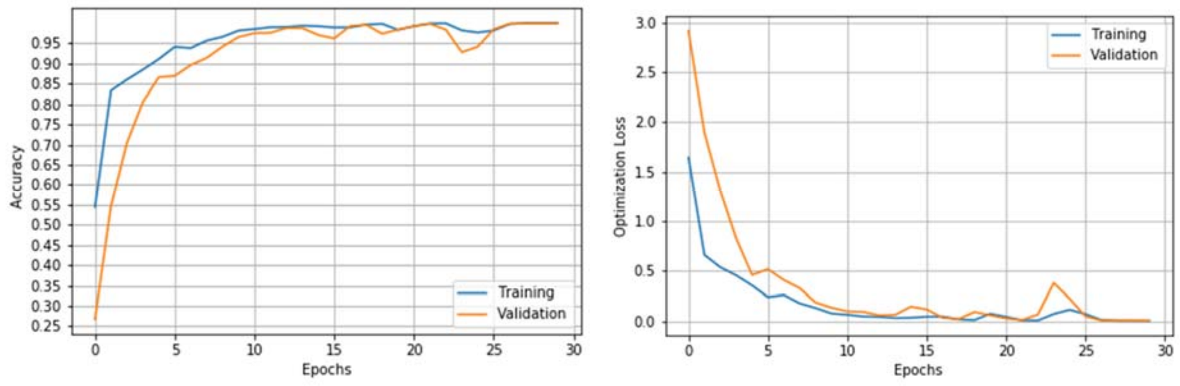


Figure 5: Accuracy and loss plots for batch size 128

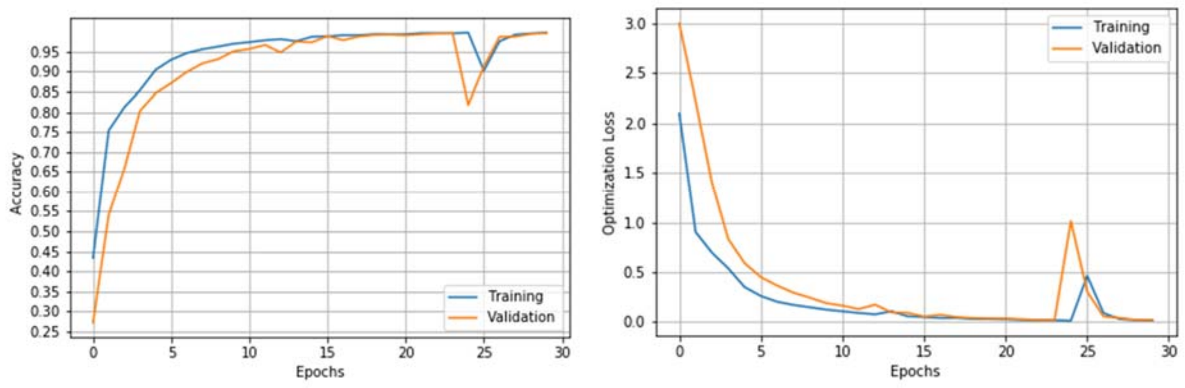
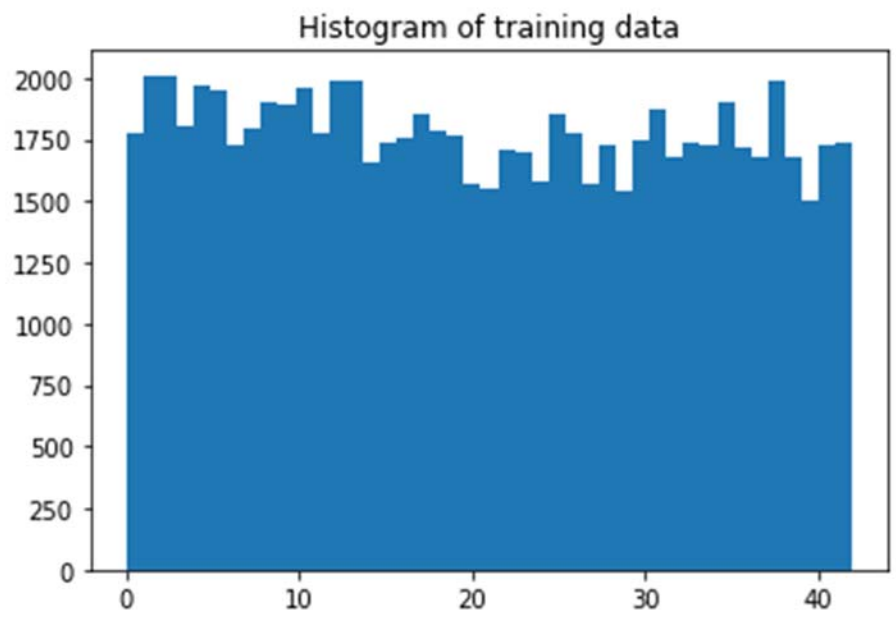


Figure 6: Accuracy and loss plots for batch size 256

To avoid overfitting, two techniques have been employed. Firstly data augmentation is employed to increase number of images for training. Moreover, dropoff of 50% is introduced on the last two fully connected layers.



Data Augmentation:

Various libraries are present on the internet which can augment images. For this work, the image augmentation library at [3] is used. Any mage to be augmented is first affect by random Gaussian blur. Afterwards random brightness, sharpness and contrast normalization is done on them. The code for this section is in cell 4a.

To account for different number of images per class in TD, in the cell 4b, at first it is determined that how many new images per class should be added. Then data augmentation is applied on randomly selected images of a class. The overall number of training data is extended from 34799 to 76500 approximately. The new histogram for images from all classes is shown below

Training the Model on Augmented Data (Cell 5)

The augmented and original TD are combined, pre-processed, and then fed through the model. A final validation accuracy of over 95% is achieved. Similarly, 96% accuracy on test images is achieved (Cell 5b).

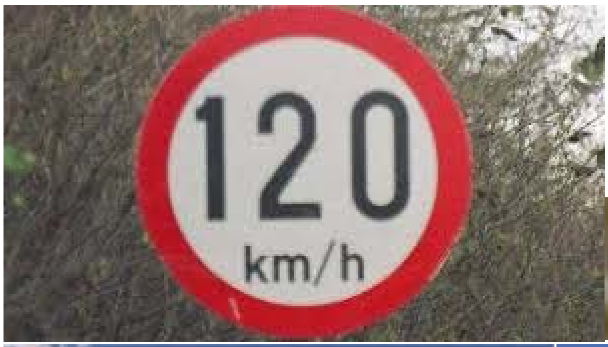
Final results after 30 epochs have been

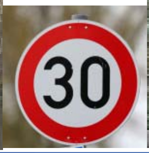
```
Training Loss = 0.142
Validation Loss = 0.036
Training Accuracy = 0.962
Validation Accuracy = 0.992
```

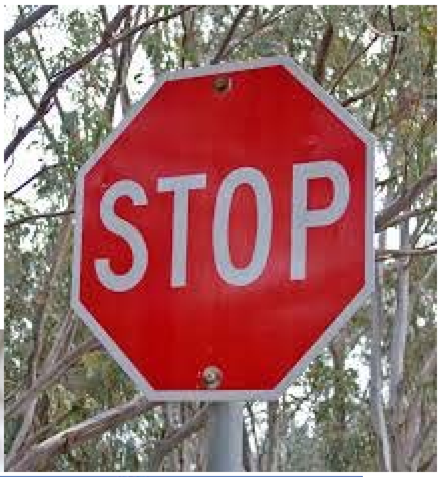
Cell 5a also shows the final smooth plot.

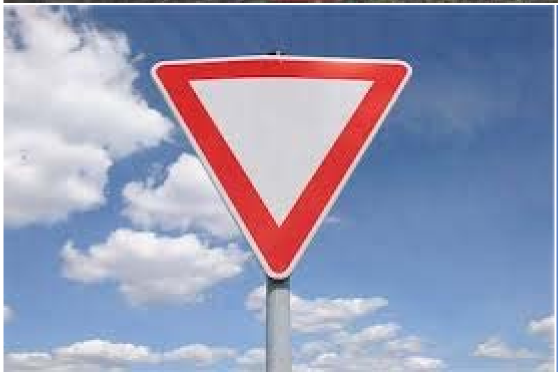
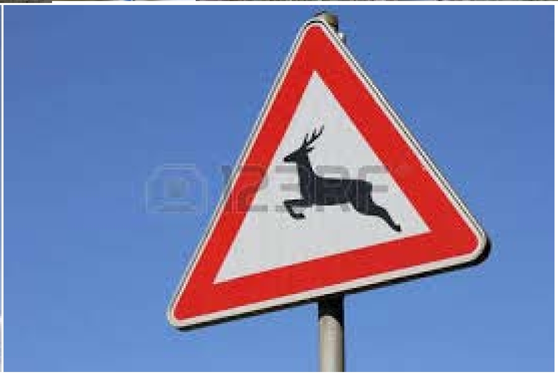
Adding new images from Web (Cell 6)

5 new images are added from Internet which are shown below







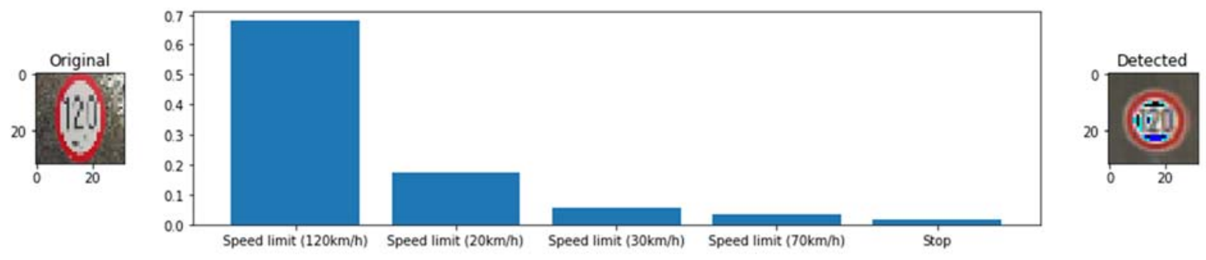



Introduction of New Images and their Identification (Cell 6a and 6b)

The accuracy on these images and the predicted probabilities are summarized inside the jupyter notebook and the html file.

It has been expected that accuracy on new images would be higher, sighting the previous high value of accuracy on test images. However, 60 % accuracy on 5 new images is observed. First wrong image detected was the Stop sign, which was detected as a speed sign. The second highest probability was in fact for the Stop sign. Thus, the reason for false detection can be that Stop sign is taken from such an angle, that it looks more round and resemble more like a speed limitation sign. The other wrongly detected images was yield sign, and it was not even detected within the first 5 images. Hence, that can be deemed as a curious case.

A sample figure showing prediction probabilities in bar chart, along with original and detected images, is given below



Visualization of layers in CNN

The code for visualizing layers of the CNN is shown in Cell 7a and 7b. Figure 7 shows the input to layer 1 of the CNN while figures 8 and 9 are the outputs from the convolutions layers having depths of 6 and 16 each. Here it can be seen that the first layer can detect different edges in the figure while the batch of filters in the next layer collect more features about the sign, and pass it onto the next layers which flattens the information and eventually leads to image detection.

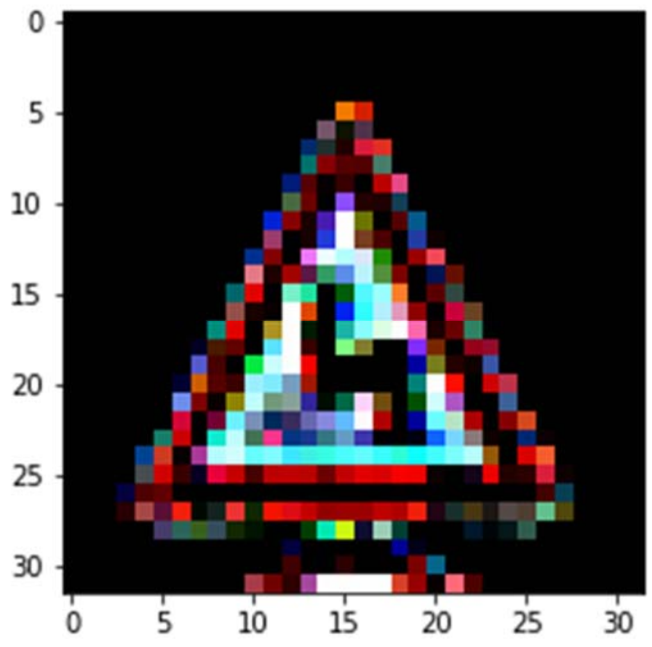


Figure 7: Sample image from training data

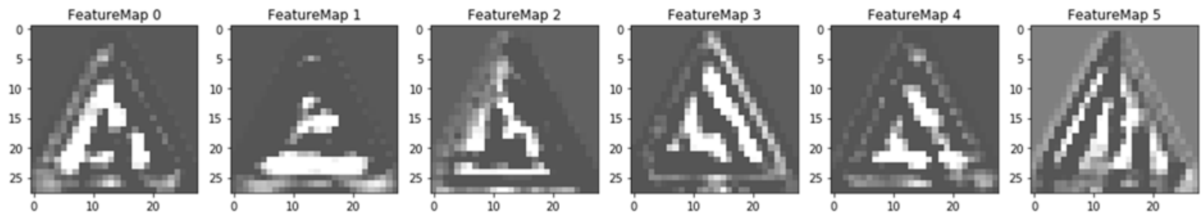


Figure 8: Output after the first activation layer

:

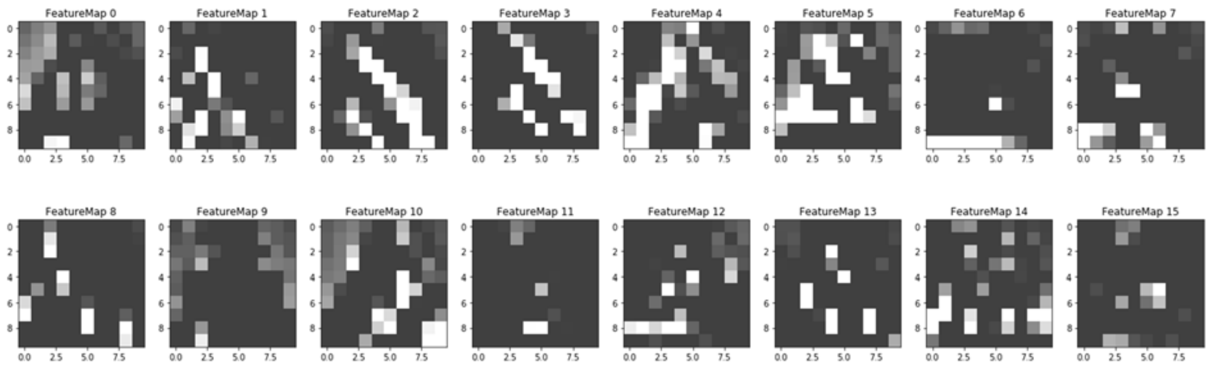


Figure 9: Output after the second activation layer

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition.*Proceedings of the IEEE*, November 1998.

[2]Alex Krizhevsky , Ilya Sutskever , Geoffrey E. Hinton, “Imagenet classification with deep convolutional neural networks”, 2012

[3] https://github.com/aleju/imgaug