Traffic Sign Recognition

Udacity Student

Udacity
Self-driving car Nanodegree Program
Project 2

Contents

1 Introduction

This document serves as the project writeup for the Udacity Self-driving car Nanodegree Program, term 1, project 2, traffic sign classification. The project code, as well as the present document can be retrieved at https://github.com/kacl780tr/blob/master/carnd_project_2.

2 Project structure

It should be noted that in this project a number of useful functions and classes have been gathered into three companion files, which are imported into the main notebook file, rather than implementing everything directly within the notebook. This allows the notebook itself to be maintained in a clean and uncluttered state. The files in question are:

traffic_data.py contains function related primarily to loading, handling and visualizing data.

train_tools.py contains functions related to training and testing, such as evaluation of accuracy measures, loading/restoration of dumped model checkpoints, etc.

html_table.py contains a simple class to make formatting of html tables within python notebooks easy.

3 Dataset characteristics

The provided data was previously divided into training, validation and testing sets. A few relevant summary statistics are

- The training set contains 34799 examples
- The size of the validation set is 4410
- The size of test set is 12630
- The shape of a traffic sign image is (32, 32, 3) *(height, width, channel)*
- The number of unique classes/labels in the data set is 43

A random sample of each of the 43 labels is shown in figure 1.

Unfortunately, not all classes of traffic signs are represented equally. It is not known if the composition of the data sets reflects some estimate of the actual frequency of occurence of the signs on real roadways. In any case, a plot of the composition of the three datasets is presented in figure 2. On the graph scale, however, the disparity between classes looks much worse than it seems to be in practice.

3.1 Preprocessing

Data normalization is well-known technique in machine learning. It makes the scales of the various dimensions much more equal, and thus makes the task of optimization less dependent on specific directions or starting points.

For our purposes, we have chosen to normalize the image data such that all values lie in the range (-1, +1), which makes the image symmetric about zero. In addition, we chose to apply this normalization to each color channel independently. This has the effect of giving each image channel an intensity much closer to that of the other channels when fed to the network. Thus an image which is predominantly red, say, would have its blue and green channels amplified relative to the red channel. The hypothesis is that this makes the overall intensity information more like that of a grayscale image, without removing the color-specific cues that may assist the neural network in identifying features.

Note also that the images are provided as unsigned byte arrays, and must be converted to floating point arrays to avoid overflow issues with the normalization.

In our implementation, the preprocessing step is applied on a "just-in-time" basis as the training process loops over batches. This is done by the **Data** class, which can be found in the file **traffic_data.py**. This class also has the ability to shuffle its contents.

3.2 Augmentation

For supervised learning problems such as the one here, more data is almost always better. It was therefore decided to apply data augmentation techniques as a means of increasing the amount of data available, in order to

- Increase the variety of features available to the network.

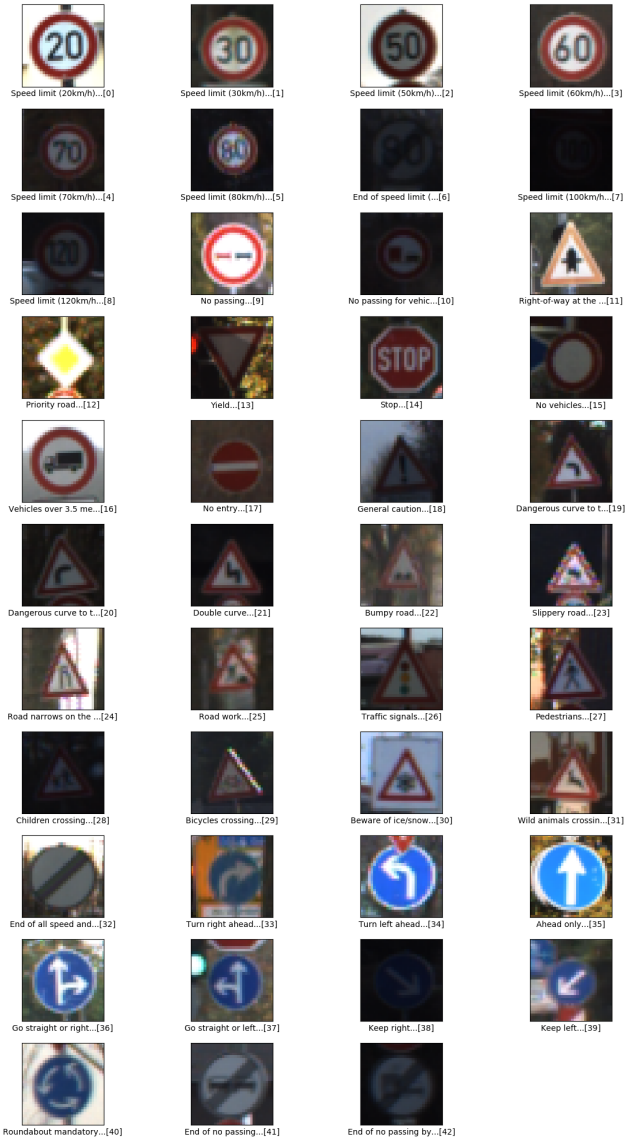

Figure 1: Random selection of each image class

- Decrease the probability of overfitting.

The specific augmentation techniques which have been applied here include:

Rotation Images are rotated by a random angle θ in the range (-35, +35) degrees. If $|\theta| \leq 2$, then $\theta = 0$

Reinterpolation With a probability of $\frac{1}{2}$, the image is upscaled by a factor of 2 with bicubic interpolation, followed by downscaling back to the original size using bilinear interpolation

Shifting Random shifts of up to ± 2 pixels are independently applied to the horizontal and vertical directions. Probability of zero shift $= \frac{1}{3}$

One can imagine that rotations correspond to leaning sign poles, hilly terrain, etc., reinterpolation might be interpreted as foggy weather, while shifting might be due to changes in lane selection.

We do *not* include either vertical or horizontal flipping of images for the simple reason that a "left turn" sign, once flipped, is no longer a "left turn", but rather a "right turn". Similarly, vertical flips would also result in nonsensical samples. Such flips would serve only to reduce the overall data quality.

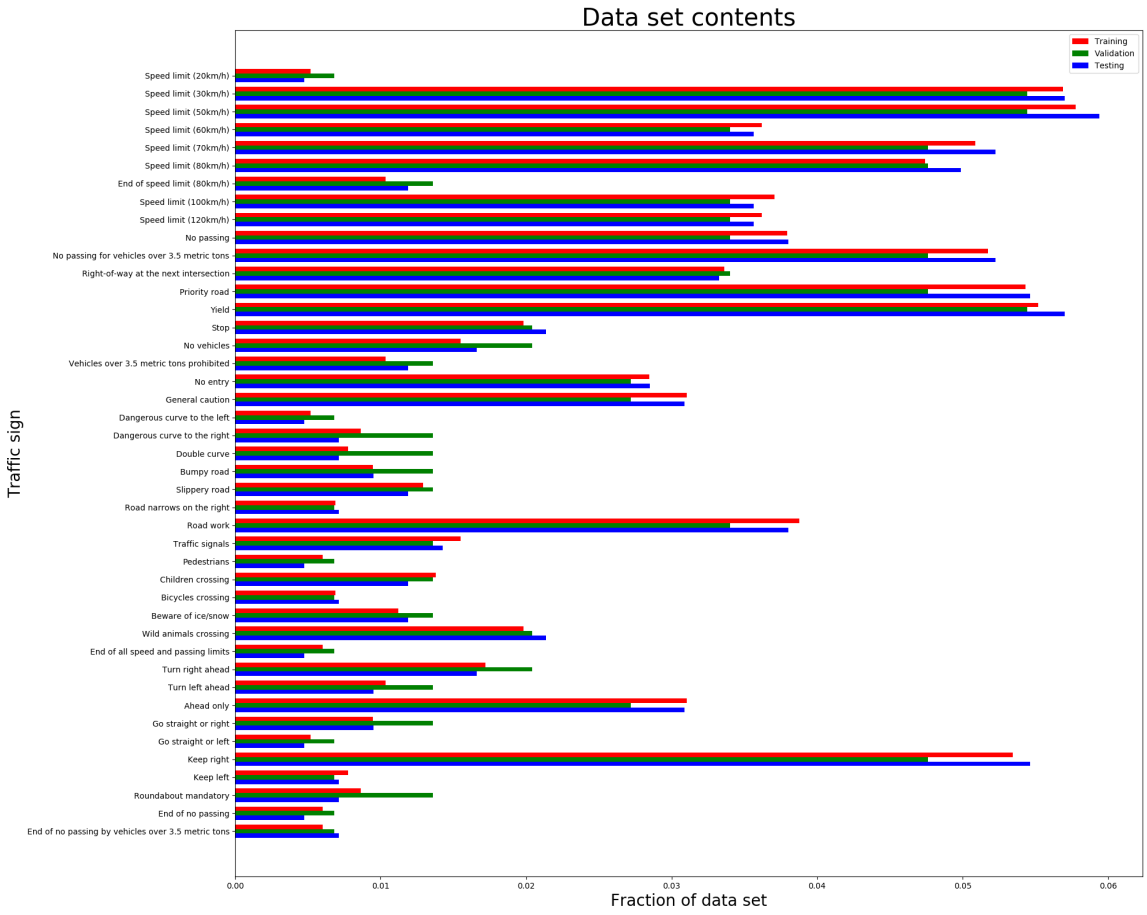

Figure 2: Composition of training, validation and testing data set by class label

In order to maximize the amount of training data, two additional images were generated using these techniques for each of the original training images. No augmented images were generated from the validation or test data sets, thus these crucial sets remain completely unknown to the model until it is called upon to make the necessary predictions. The overall composition of the training data is thus left unchanged, we simply have more data from which to extract relevant features.

A sample of the results of the data augmentation process employed here can be seen in figure 3. Note that these augmented batches were generated once, and stored to disk to save processing time, and to allow different architectures to be directly compared, having been trained on the same data.

4 Model

4.1 Architecture

The final architecture was chosen from amongst several alternatives. All were essentially derivatives of the LeNet architecture, which seems to perform very well for this application. The final candidate is essentially a version of LeNet, with extended feature maps, and the addition of dropout and batch normalization. A detailed description of the architecture is provided in table 1.

The loss measure used for gradient descent is the cross-entropy softmax loss applied to the logits output by the network. The accuracy measure is the straightforward equality of ground-truth and argmax of the logits.

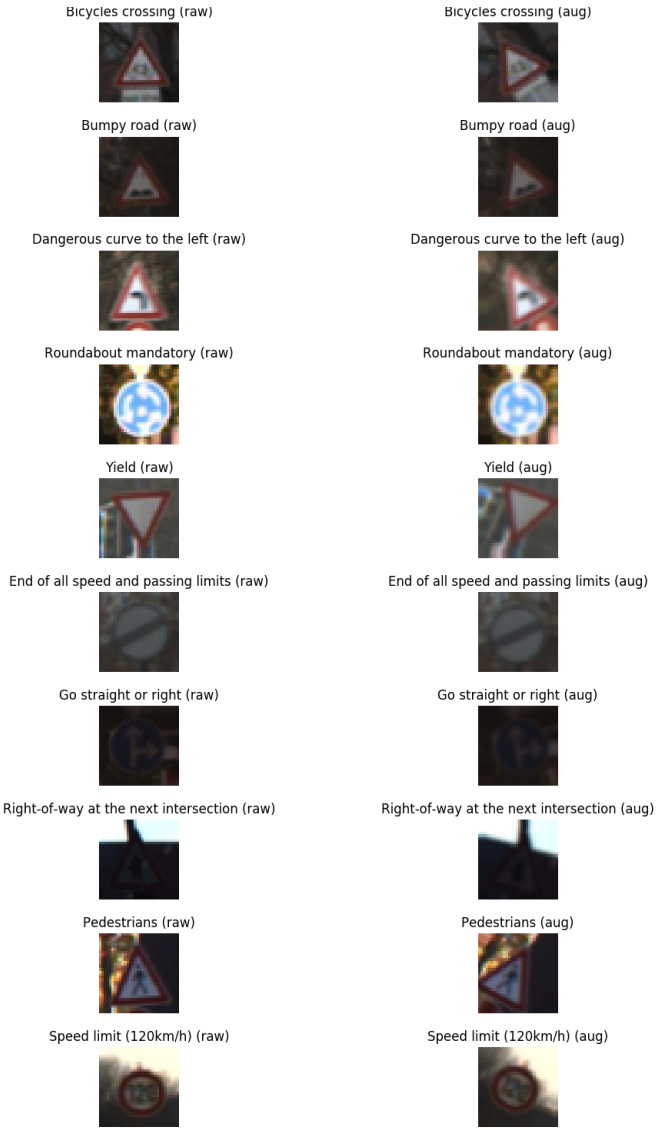

Figure 3: Random selection of data augmentation comparisons

4.2 Training procedure

The model was trained using the Adam optimizer, with all parameters except the learning rate left at their default values. Learning rate decay was employed such that the learning rate was reduced by a factor of 0.975 each 5 epochs.

To facilitate collection of results, a method of tracking validation accuracy was devised such that each globally improved accuracy level (beyond a threshold) resulted in the model variables being saved as a checkpoint. Insufficient progress toward a new global maximum accuracy resulted in early termination of the run.

The attainable accuracy level was initially benchmarked with the basic LeNet architecture with no modifications. It was found that validation accuracy of approximately 0.90 was attainable. Data augmentation was explored as the first means of increasing the accuracy levels, the preference being to obtain maximum performance out of smaller, faster, models when real-time processing is likely to be a factor.

The basic approach followed here was that of making incremental adjustments to an architecture which

Layer	In	Out	Notes
Input	-	$32 \cdot 32 \cdot 3$	Placeholder for RGB image
Convolution $(5 \cdot 5)$	$32 \cdot 32 \cdot 3$	$28 \cdot 28 \cdot 12$	Stride 1, padding = valid
Batch normalization	$28 \cdot 28 \cdot 12$	$28 \cdot 28 \cdot 12$	-
ReLU activation	$28 \cdot 28 \cdot 12$	$28 \cdot 28 \cdot 12$	-
Max pooling $(2 \cdot 2)$	$28 \cdot 28 \cdot 12$	$14 \cdot 14 \cdot 12$	Stride 2, padding = valid
Convolution $(5 \cdot 5)$	$14 \cdot 14 \cdot 12$	$10 \cdot 10 \cdot 32$	Stride 1, padding = valid
Batch normalization	$10 \cdot 10 \cdot 32$	$10 \cdot 10 \cdot 32$	-
ReLU activation	$10 \cdot 10 \cdot 32$	$10 \cdot 10 \cdot 32$	-
Max pooling $(2 \cdot 2)$	$10 \cdot 10 \cdot 32$	$5 \cdot 5 \cdot 32$	Stride 2, padding = valid
Flatten	$5 \cdot 5 \cdot 32$	$1 \cdot 800$	-
Fully connected	$1 \cdot 800$	$1 \cdot 120$	-
Batch normalization	$1 \cdot 120$	$1 \cdot 120$	-
ReLU activation	$1 \cdot 120$	$1 \cdot 120$	-
Dropout regularization	$1 \cdot 120$	$1 \cdot 120$	Retain rate as tuneable parameter
Fully connected	$1 \cdot 120$	$1 \cdot 84$	-
ReLU activation	$1 \cdot 84$	$1 \cdot 84$	-
Dropout regularization	$1 \cdot 84$	$1 \cdot 84$	Retain rate (shared)
Fully connected	$1 \cdot 84$	$1 \cdot 43$	- Final output logits

Table 1: Model layer architecture

had demonstrated capability. Here, these adjustments were

1. the addition of dropout,
2. batch normalization,
3. depth increase in convolution feature maps.

5 Results

At completion of training, the model presented the following results:

- training accuracy $\gtrapprox 0.99$
- validation accuracy of 0.983
- testing accuracy of 0.967

These results provide evidence that the model is fitting well, without overfitting, and is capable of generalizing to unseen data in a reasonable manner.

However, accuracy is not the whole story. If there were a small number of stop signs in the validation or test sets, for example, then we might achieve a very high accuracy even when missing all of them. This would likely be considered a fatal flaw in the model, given its intended purpose.

As a further check of the test results, the confusion matrix was computed for the test data, and the precision, recall and F-score calculated for each class. The results are encouraging, if not completely satisfactory.

The F-score results show that

- The minimum F-score value (0.654) was for the "Pedestrians" class. This may be considered problematic, given the importance of identifying pedestrians for autonomous vehicle applications.
- 38 of 43 classes have F-score values above 0.90.
- 4 classes have an F-score of 1.0.

5.1 Web test images

A total of 14 images were found on the internet. Two of these images were not actual pictures, but rather logos, and were included out of interest. Deliberately included in this set are some of the more complex sign variants.

The web test images, along with their ground-truth labels, and the top-5 prediction probabilities from the model are shown in figure 4.

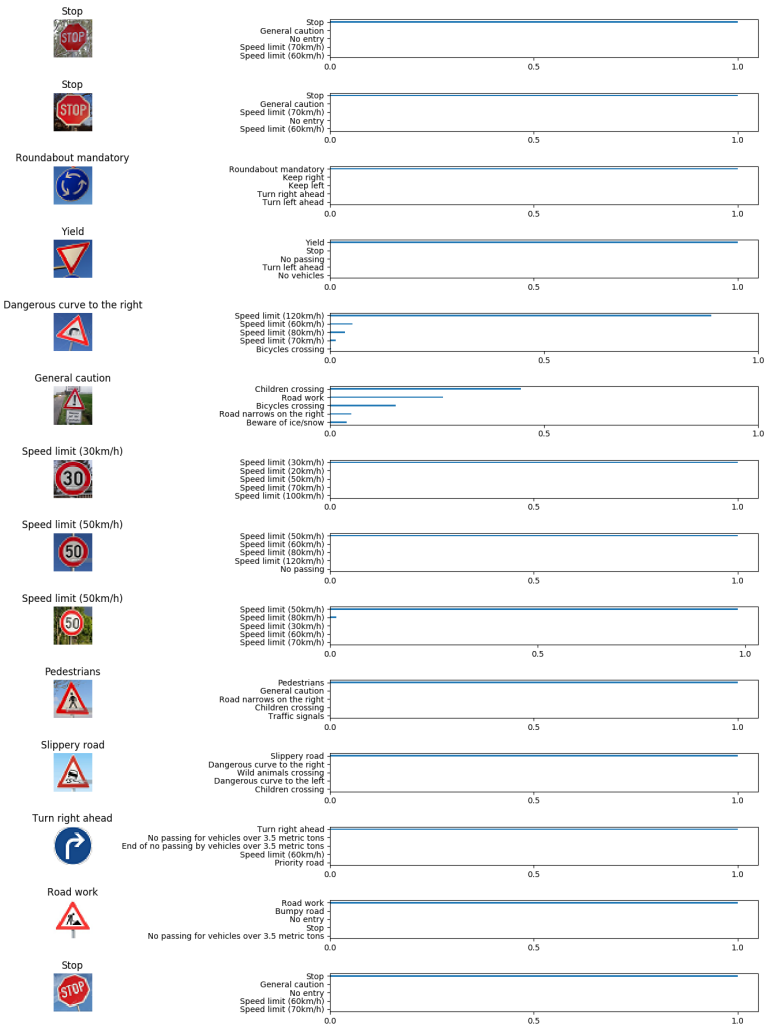

Figure 4: Web test images with ground-truth and top 5 model predictions.

One can see that the model is, in most cases, extremely confident in its predictions. The accuracy of 0.857 on the web test images is somewhat lower than the performance on the test set. This is very likely due to small-sample size, as well as the fact that when composing the web set a deliberate choice of several images which might be considered both more complex, as well as under-represented in the data sets, was made.

6 Conclusion

The model described here represents a good start on the problem of traffic sign recognition. The accuracy results are encouraging, but examination of the per-class results provides some indications about how to do better.

The first thing to do would be to acquire more data, particularly for the less-well represented images, and in proportion to estimates of the consequences of mis-identification. In our web test set, for example, "Dangerous curve to the right" was mis-identified as "Speed limit (120 km/h)". Taking a dangerous curve at 120 km/h is probably exactly the wrong thing to do.

Data augmentation could be further refined, but it is likely unwise to rely on this alone to improve the scores of problematic classes.

As always, refinement of the architecture, as well as fine-tuning of training procedures are also available.

Classname	Precision	Recall	F-score
Ahead only	0.99458	0.94103	0.96706
Beware of ice/snow	0.80952	0.68000	0.73913
Bicycles crossing	0.95652	0.97778	0.96703
Bumpy road	1.00000	0.85833	0.92377
Children crossing	0.85965	0.98000	0.91589
Dangerous curve to the left	1.00000	0.98333	0.99160
Dangerous curve to the right	0.94737	1.0	0.97297
Double curve	0.86905	0.81111	0.83908
End of all speed and passing limits	0.96774	1.0	0.98361
End of no passing	1.0	0.85	0.91892
End of no passing by vehicles over 3.5 metric tons	1.0	1.0	1.0
End of speed limit (80km/h)	1.0	0.98667	0.99329
General caution	0.99723	0.92308	0.95872
Go straight or left	1.0	1.0	1.0
Go straight or right	0.95283	0.84167	0.89381
Keep left	1.0	0.95556	0.97727
Keep right	0.97278	0.98406	0.97839
No entry	1.0	0.99722	0.99861
No passing	0.9732	0.98333	0.97824
No passing for vehicles over 3.5 metric tons	0.99543	0.99091	0.99317
No vehicles	0.96744	0.99048	0.97882
Pedestrians	0.74468	0.58333	0.65421
Priority road	0.99412	0.97971	0.98686
Right-of-way at the next intersection	0.9351	0.92619	0.93062
Road narrows on the right	0.95604	0.96667	0.96133
Road work	0.92731	0.98333	0.9545
Roundabout mandatory	0.95556	0.95556	0.95556
Slippery road	0.77202	0.99333	0.8688
Speed limit (100km/h)	0.98575	0.92222	0.95293
Speed limit (120km/h)	0.93939	0.96444	0.95175
Speed limit (20km/h)	1.0	0.8	0.88889
Speed limit (30km/h)	0.99444	0.99306	0.99375
Speed limit (50km/h)	0.98406	0.988	0.98603
Speed limit (60km/h)	0.92473	0.95556	0.93989
Speed limit (70km/h)	0.9688	0.98788	0.97824
Speed limit (80km/h)	0.94163	0.97302	0.95706
Stop	0.99631	1.0	0.99815
Traffic signals	0.95213	0.99444	0.97283
Turn left ahead	1.0	1.0	1.0
Turn right ahead	0.99526	1.0	0.99762
Vehicles over 3.5 metric tons prohibited	1.0	1.0	1.0
Wild animals crossing	0.96691	0.97407	0.97048
Yield	0.99032	0.99444	0.99238

Table 2: Precision, Recall and F-score for the Test data set