Udacity Self-Driving Car Engineer Nanodegree

Traffic Sign Classifier

Introduction

The goals / steps of this project are the following:

- Load the data set (see below for links to the project data set)
- 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

Rubric points

Here I will consider the rubric points (https://review.udacity.com/#!/rubrics/481/view) individually and describe how I addressed each point in my implementation.

Writeup

This report is the project writeup.

Data Set Summary & Exploration

1. Provide a basic summary of the data set. In the code, the analysis should be done using python, numpy and/or pandas methods rather than hard-coding results manually.

I used the pandas library and other simple techniques to calculate summary statistics of the traffic signs data set:

- Number of training examples = 34799
- Number of validation examples = 4410

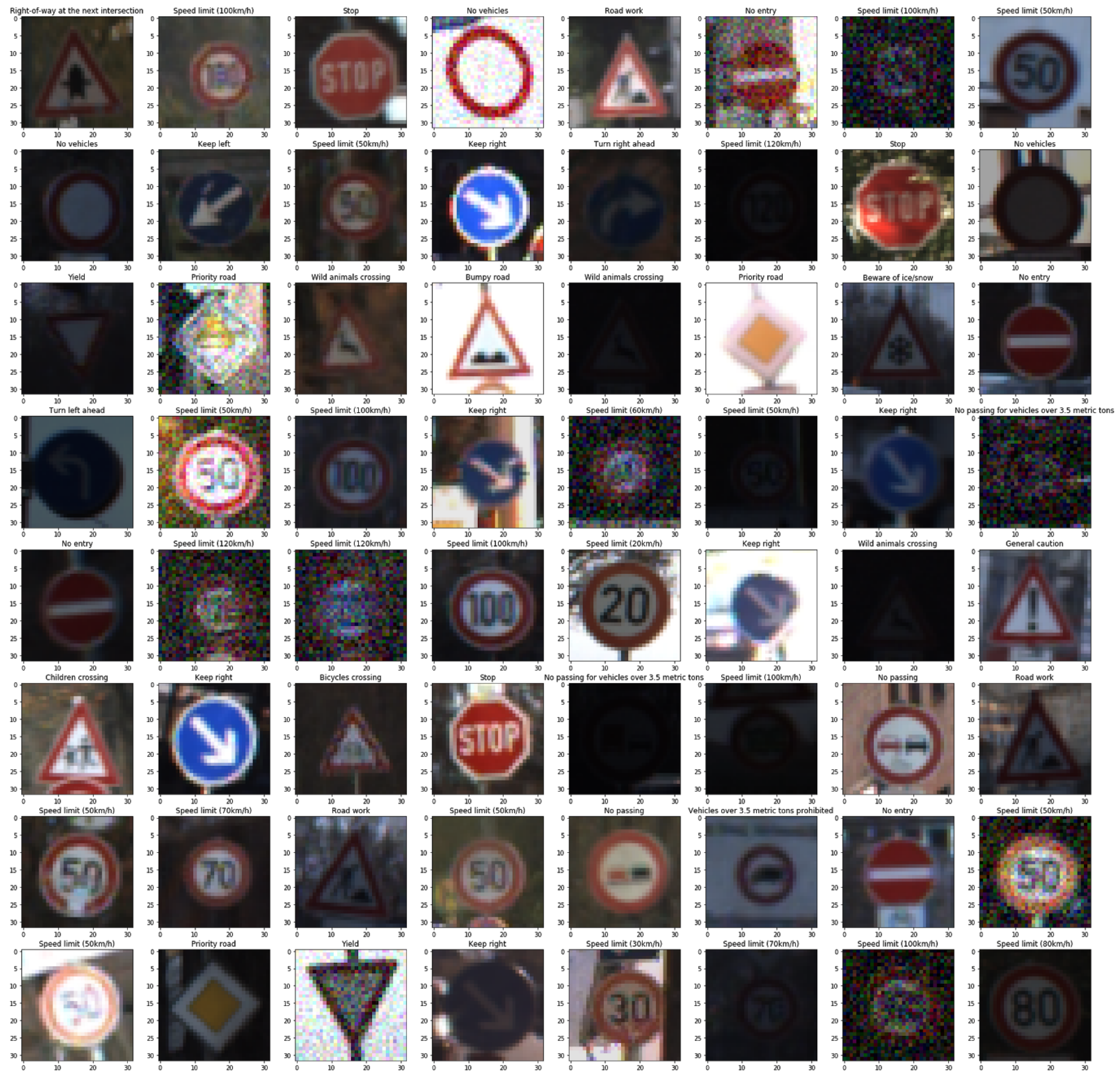
- Number of testing examples = 12630
- Image data shape = (32, 32)
- Number of classes = 43

After I augmented the training set, the same statistics were:

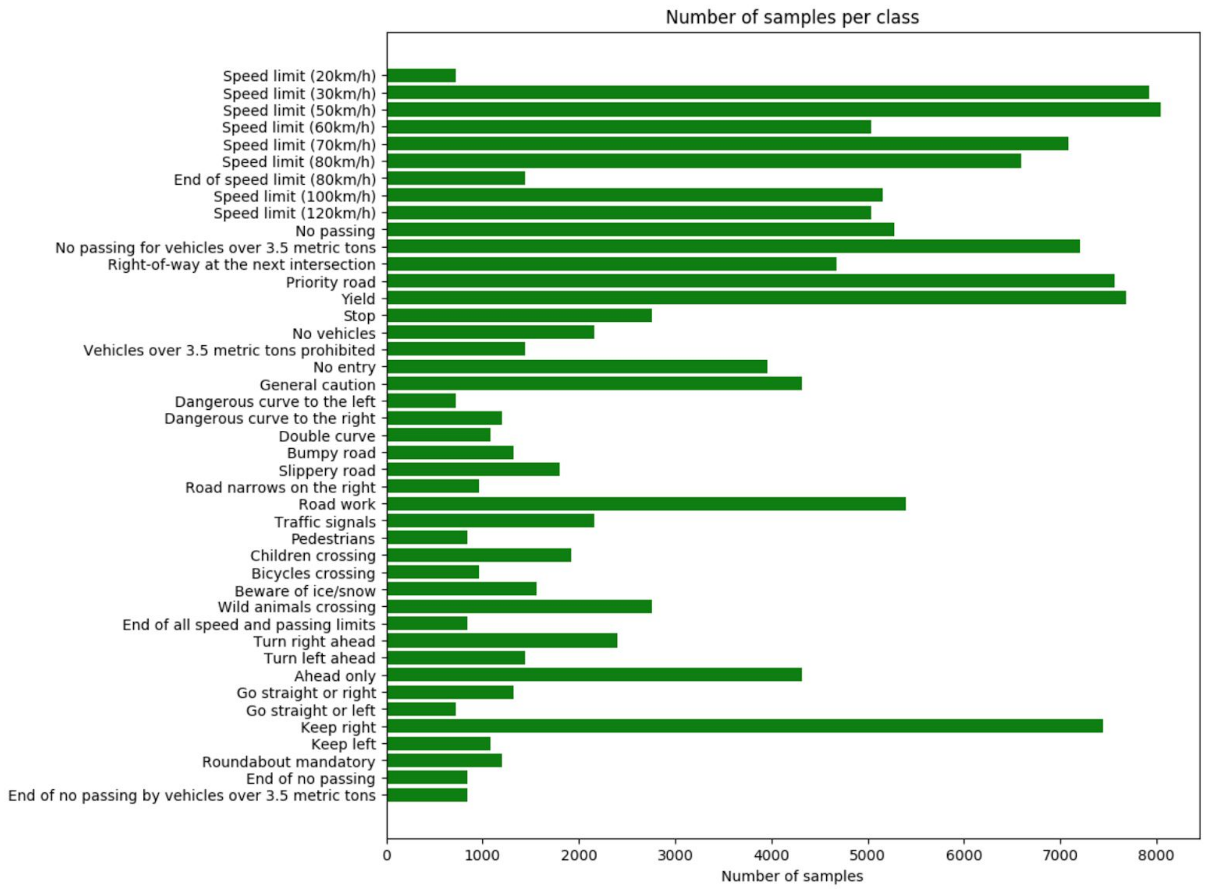
- Number of training examples = 139196
- Number of validation examples = 4410
- Number of testing examples = 12630
- Image data shape = (32, 32)
- Number of classes = 43

2. Include an exploratory visualization of the dataset.

I visualized 64 randomly chosen images, including also augmented images.



I also visualized the number of images per class as a bar chart. This chart also includes the augmented data. The smallest classes now have around 750 examples, which should be adequate.



Design and Test a Model Architecture

1. Describe how you preprocessed the image data. What techniques were chosen and why did you choose these techniques? Consider including images showing the output of each preprocessing technique. Pre-processing refers to techniques such as converting to grayscale, normalization, etc.

I experimented with multiple preprocessing techniques: normalization of pixels to zero mean value, conversion to grayscale and two different image sharpening methods. As testing the model performance with the different preprocessing techniques, I ended up choosing a combination of image sharpening and normalization. I normalized each RGB channel separately. The image sharpening technique I used was to subtract a Gaussian blurred image from the original.

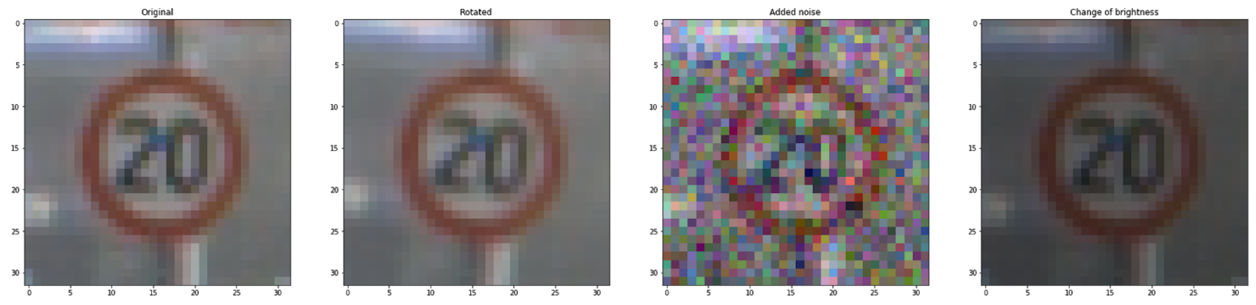
1b. As described in the "Stand Out Suggestions" part of the rubric, if you generated additional data for training, describe why you decided to generate additional data, how you generated the data, and provide example images of the additional data. Then describe the characteristics of the augmented training set like number of images in the set, number of images for each class, etc.

As the smallest traffic sign classes in the training data sets were originally quite small (approx. 250 images), I decided to augment the training data set by generating additional images.

I used three techniques: adding random noise, random rotation between -15 to 15 degrees, and changing image brightness randomly (to brighter or to darker). Each of the augmentation techniques has a randomization element to avoid introducing artificial patterns to the data, and also for greater variety.

I applied each data generation technique to each training set image and, therefore, the resulting training set had 4x images compared to the original. I experimented with Tensorflow's image augmentation libraries but had performance problems with them, and used sci-kit image libraries instead.

Here is an example of an original image to which each data generation technique has been applied to.



The data augmentation code is located in a separate notebook: Data Augmentation.ipynb

2. Describe what your final model architecture looks like including model type, layers, layer sizes, connectivity, etc.) Consider including a diagram and/or table describing the final model.

I chose a convolutional neural networks (CNN) based solution because CNNs they are often successfully used for image classification problems, and because of the availability of GPU resources. I chose a modified version of one of the classic LeNet-5 architecture for a starting point for its relative simplicity. Find the description of my final model in the table below. Because of overfitting issues, I also added two dropout layers.

Layer	**Description**
Input	32x32x3 RGB image
Convolution 1: 5x5x6	1x1 stride, valid padding, outputs 28x28x6
RELU activation	
Max pooling 2x2	2x2 stride, valid padding, outputs 14x14x6
Convolution 2: 5x5x16	1x1 stride, valid padding, outputs 10x10x16
RELU activation	
Max pooling 2x2	2x2 stride, valid padding, outputs 5x5x16
Flatten	Unwrap 5x5x16 into 400x1
Fully Connected 1	Outputs 120
RELU activation	
Dropout	Keep probability 0.425
Fully Connected 2	Outputs 84
RELU activation	
Dropout	Keep probability 0.425
Fully Connected 3	Outputs 43

Finally, argmax is applied to the final fully connected layer for predictions.

3. Describe how you trained your model. The discussion can include the type of optimizer, the batch size, number of epochs and any hyperparameters such as learning rate.

The hyperparameters I used for training the model were as in the table below. I experimented with most of these values.

Hyperparameter	Value	Comment
Optimizer	AdamOptimizer	One of the common modern optimizers
Learning rate	0.003	Training was slightly faster than with smaller values but the results were equally accurate
Batch size	512	Find as large value as the computing platform’s memory allows
Loss function	Cross-entropy	One of the typical loss functions
Number of epochs	15	Training started converge around this number and did not improve accuracy anymore

4. Describe the approach taken for finding a solution and getting the validation set accuracy to be at least 0.93. Include in the discussion the results on the training, validation and test sets and where in the code these were calculated. Your approach may have been an iterative process, in which case, outline the steps you took to get to the final solution and why you chose those steps. Perhaps your solution involved an already well known implementation or architecture. In this case, discuss why you think the architecture is suitable for the current problem.

While I was experimenting with the model and its hyperparameters, an important tool was to evaluate the accuracy for training, validation and test sets for each new attempt. My final model results were:

- Training set accuracy of 0.973
- Validation set accuracy of 0.951
- Test set accuracy of 0.951

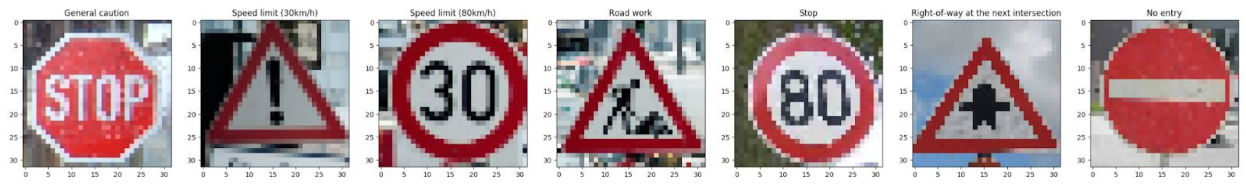
How I got there was an iterative process. On high level my journey was the following:

- Using the LeNet-5 architecture as a starting point. Modification was needed to make the model support 43 different traffic sign types.
- Experimented feeding grayscale images instead for RGB but this reduced accuracy.
- Noticed overfitting between the training and the validation set, and decided to add dropout layers for two of the fully connected layers. This improved accuracy for the validation set.
- From this point on, the further improvements were achieved through image pre-processing and hyperparameter tuning. I spent more time on this than experimenting with the model itself.

Test a Model on New Images

1. Choose five German traffic signs found on the web and provide them in the report. For each image, discuss what quality or qualities might be difficult to classify.

Here are five German traffic signs that I found on the web:



All the images are of very good quality and I assumed very accurate predictions.

2. Discuss the model's predictions on these new traffic signs and compare the results to predicting on the test set. At a minimum, discuss what the predictions were, the accuracy on these new predictions, and compare the accuracy to the accuracy on the test set.

In the test run included with my project submission, all other signs are identified correctly but the 80 km/h sign is incorrectly identified as a 30 km/h sign. I suspect this is because the numbers 3 and 8 are quite similar, but the failure is actually surprising because this test set is such good quality images in good lighting conditions unlike the training, validation and test sets. The resulting accuracy is only 0.857, compared to the accuracy of 0.951 on the official test set. However, most of the intermediate models achieved an accuracy of 1.0 on these test images.

3. Describe how certain the model is when predicting on each of the five new images by looking at the softmax probabilities for each prediction. Provide the top 5 softmax probabilities for each image along with the sign type of each probability.

For all other images except for number 6, the prediction is relatively certain in a sense that the margin to the next best matching class is quite large. Both speed limit images have only got other speed limit classes as top-5 predictions. In terms of an absolute value, the prediction of image 4 is most certain.

Image no	Prediction and correctness	Top-5 softmax probabilities
1	CORRECT: No entry	43.6 No entry 25.9 Stop 8.9 Priority road 8.2 No passing 4.9 Yield
2	CORRECT: Right-of-way at the next intersection	46.6 Right-of-way at the next intersection 25.8 Beware of ice/snow 16.1 Double curve 2.0 Turn left ahead 1.4 Pedestrians
3	CORRECT: Speed limit (30km/h)	41.5 Speed limit (30km/h 11.1 Speed limit (70km/h) 10.4 Speed limit (50km/h) 4.2 Speed limit (80km/h) 1.8 Speed limit (20km/h)
4	CORRECT: Road work	49.7 Road work 12.7 General caution 6.0 Yield 1.6 Bumpy road 0.2 No passing for vehicles over 3.5 metric tons
5	CORRECT: General caution	38.0 General caution 22.2 Pedestrians 21.3 Traffic signals 8.3 Road narrows on the right

		3.5 Go straight or left
6	INCORRECT: Speed limit (30km/h) -- should be Speed limit (80km/h)	11.4 Speed limit (30km/h) 11.0 Speed limit (80km/h) 10.8 Speed limit (50km/h) 7.3 Speed limit (60km/h) 6.1 Speed limit (70km/h)
7	CORRECT: Stop	30.1 Stop 11.2 No entry 6.8 Yield 2.8 Speed limit (30km/h) 2.0 Priority road

Improvement ideas

To get the model to predict on 0.90-0.92 accuracy level was relatively easy, but from there on improvements required more work. Despite the different hyperparameters and image pre-processing techniques applied, I was not able to get past 0.95 accuracy for the test set.

For further improvements, I would first try conducting error analysis and see what type of images get mis-classified. I suspect fully dark images is a major error category, and pre-processing techniques could help to process those images better.

Secondly, I would try a more advanced CNN model, such as adding inception modules that Sermanent and LeCun have used in their paper *Traffic Sign Recognition with Multi-Scale Convolutional Networks*.