Traffic Sign Recognition

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

Writeup / README

The code for my project is placed along with this file as Traffic_Sign_Classifier.ipynb.

Data Set Summary & Exploration

1. Provide a basic summary of the data set and identify where in your code the summary was done.

The code for this step is contained in the second code cell of the IPython notebook.

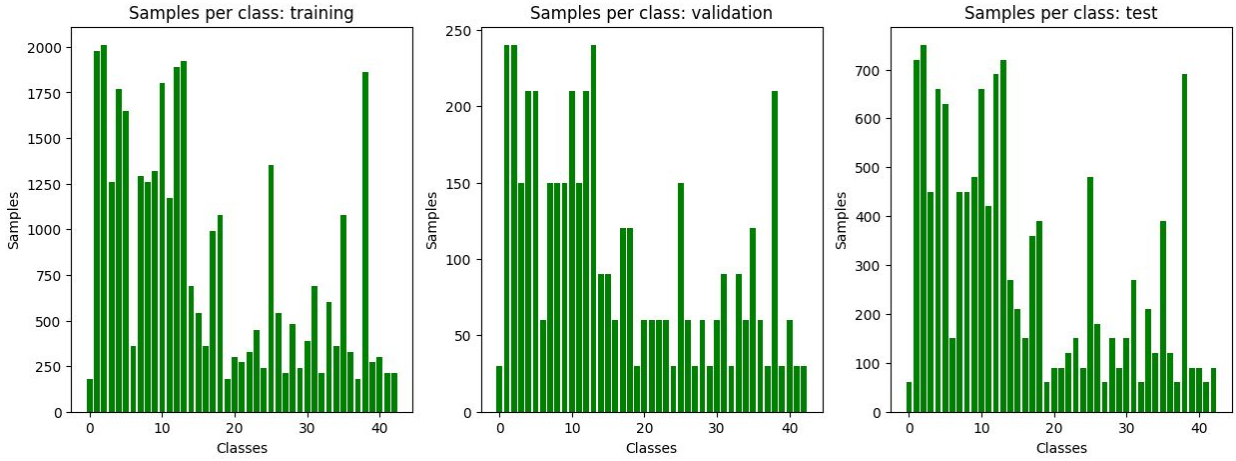
I used the pickle library to load the data of the traffic signs data set:

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

2. Include an exploratory visualization of the dataset and identify where the code is in your code file.

The code for this step is contained in the 3, 4, 5, 6 code cells of the IPython notebook.

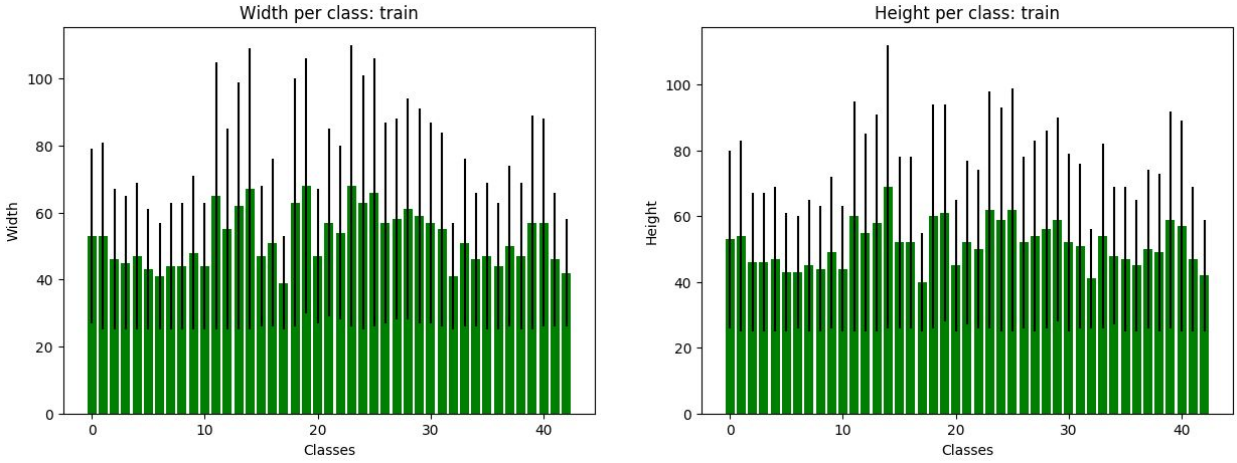
The chart below represents the distribution of the samples per sign class for training, validation, and test data sets:

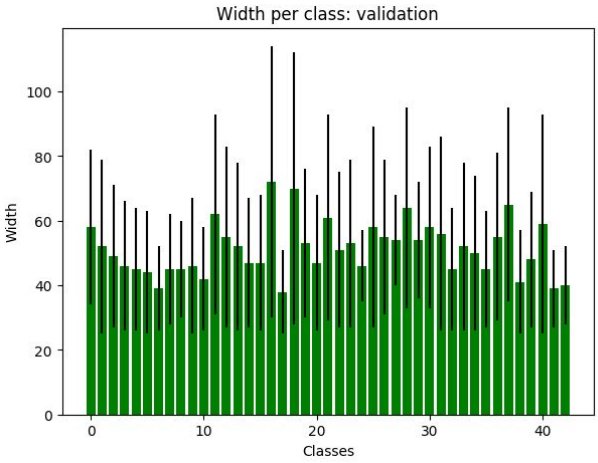


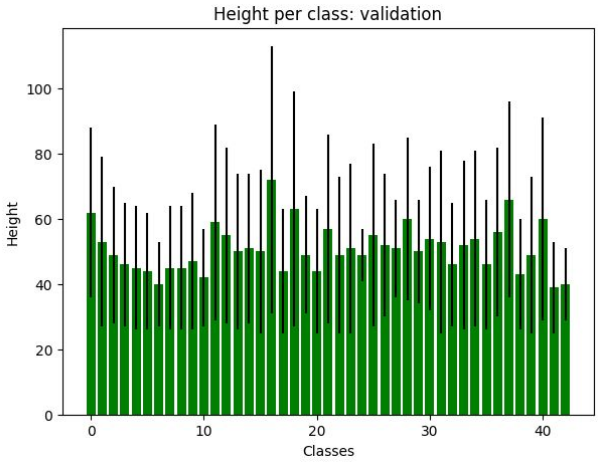
Two conclusions may be done based on this chart:

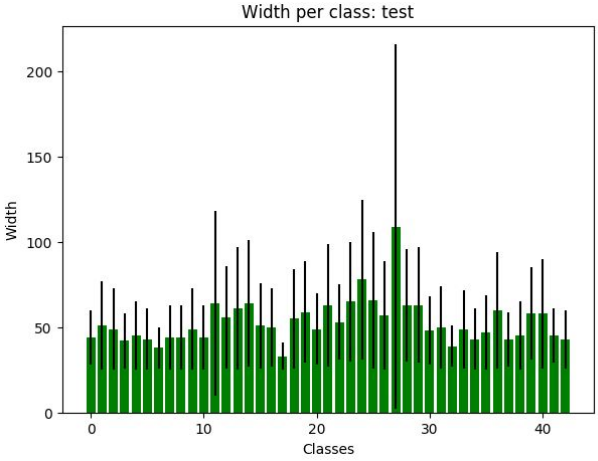
- All three data sets have similar distribution of the represented sign classes
- There is significant variation in the representation of the classes in the data set

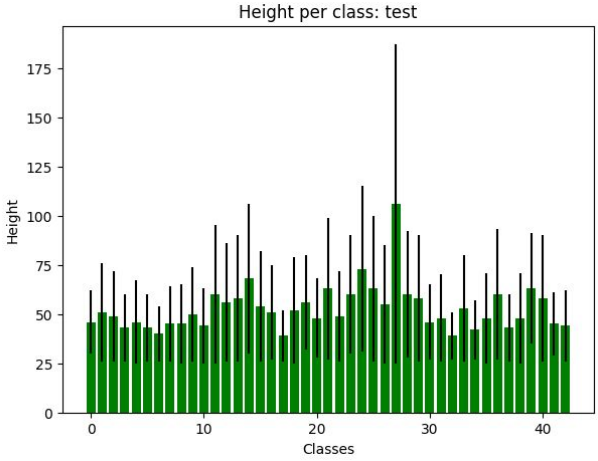
The chart below represents distribution of the original width and height (average and max/min) of the images per class for all the data sets:











The conclusion from this visualization is that there is no dramatic variation in the sizes of the original images between training, validation, and test data sets. There is also no dramatic difference in the sizes of the original images between images representing different classes

Overall, the visualization of the dataset did not reveal dramatic differences in the way the data were collected between classes and between data subsets, which allows to use the data for training. The only unfortunate fact regarding the data set is that some of the classes are significantly underrepresented in the dataset compared to other classes.

Design and Test a Model Architecture

1. Image preprocessing:

I have performed two kinds of data preprocessing:

- Data augmentation (function augmentDataSet and friends)

In order to increase the data set for the training purposes I have applied the following five transformations for each image in the training set:

- Identity
- Affine transformation using two different matrices
- Rotations around the center of the image 15 degrees both clockwise and counter-clockwise

Below is example of the sign before and after augmentation (note, that the last image in the row is grayscale transformation, which was not used for the final training)

As the result of augmentation the training data set has been increased five-fold.

- Data normalization (function applyNormalization)

I have used the recommendation from the class lectures and individually for each layer in the image applied the following normalization:

X = (X - mean) / mean

I have not performed transformation to grayscale, mostly because of my original assumption that color of the sign should have important information regarding the sign itself. So, I have not seen obvious reason to limit the number of potential features.

2. Preparation of training/validation/testing data sets:

I have not performed any extra split on the training/validation/testing data sets beyond what was already provided in the original data set.

As the result my final training set consisted of 173,955 (34799 * 5) images. My validation set consisted of 4410 images. Finally, my test set consisted of 12630 images. Retrospectively, I probably should have augment validation set as well, and/or randomly dedicate part of the training set to the cross-validation purposes.

3. Model description:

The code for my final model is located in the “Model architecture” section of the ipython notebook and implemented by “CNNNet” function.

I have started with the LeNet model described in the class materials. I have then applied “multi-scaled feature” described at Sermanet and Yann LeCun paper, feeding both outputs from first and second convolutional levels into the fully-connected third layer. I have also applied dropouts with keep probability of 0.9 to every layer.

code for my final model is located in the “Model architecture” section of the ipython notebook and implemented by “CNNNet” function.

My final model consisted of the following layers:

Layer	Description
Input	32x32x3 RGB image, normalized as described above
Convolution1 5x5	1x1 stride, valid padding, outputs 28x28x6
RELU	
Max pooling	2x2 stride, outputs 14x14x6
Dropout	Keep_probability = 0.9
Convolution2 5x5	1x1 stride, valid padding, outputs 10x10x16
RELU	
Max pooling	2x2 stride, outputs 5x5x16
Dropout	Keep_probability = 0.9

Fully connected1	Input: convolution1 concatenated with convolution2 after dropout steps Output size: 43 * 4
RELU	
Fully connected2	Output size: 43*2
RELU	
Fully connected3	Output size: 43
Softmax	

4. Training procedure:

The code for training the model is located in the “Train, Validate and Test the Model” cell of the ipython notebook.

To train the model, I used tensorflow framework running on my local workstation equipped with GTX 1070 Nvidia GPU.

I have run the training cycle in 300 epochs using the complete training set split into batches of 8192 images. I have tried to use maximum possible batch size. Every 10 epochs I was evaluating the model on the complete training and validation subsets. I have started with learning rate of 0.001 and decreased it to 0.0001 after 200 iteration of epochs.

The training was reaching the accuracy 1.0 after approx. 190 epochs. However, I continued training because the accuracy of the validation was still improving.

5. The rationale behind the choice of the model architecture:

I have started with LeNet model from the lectures. The model had a tendency to overfitting, leading to only 88% accuracy over the test data set. I have then added multi-select feature recommended at Sermanet and Yann LeCun paper, and added dropout levels to avoid the overfitting. The model produced better results.
I have also experimented by adding more convolutional levels and/or increasing the number of the filters in each convolutional filter. However, those experiments had negative effect in my original experiments - the training usually stopped around 97%, producing around 88% percent over the test set.

So, I have decided to come back to the model working for me.

The code for calculating the accuracy of the model is located in the “Train, Validate and Test the Model” section of the Ipython notebook - “accuracy_operation”

My final model results were:

- training set accuracy of 1.00000
- validation set accuracy of 0.96191
- test set accuracy of 0.95186

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 six German traffic signs that I found on the web:

Class: 0: Speed limit (20km/h)

Class: 13: Yield

Class: 21: Double curve

Class: 34: Turn left ahead

Class: 38: Keep right

Class: 40: Roundabout mandatory

My main concern were the classes 21 and 40 from this small set, because those classes were originally significantly underrepresented in the training data set.

2. Model predictions for new images:

Discuss the model's predictions on these new traffic signs and compare the results to predicting on the test set. Identify where in your code predictions were made. 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 (OPTIONAL: Discuss the results in more detail as described in the "Stand Out Suggestions" part of the rubric).

The code for prediction is at “Predict the Sign Type for Each Image” section of the Ipython notebook

Here are the results of the prediction:

Image	Prediction
Speed limit (20km/h)	Speed limit (20km/h)
Yield	Yield
Double curve	Right-of-way at the next intersection
Turn left ahead	Turn left ahead
Keep right	Keep right
Roundabout mandatory	Roundabout mandatory

The model was able to correctly guess 5 of the 6 traffic signs, which gives an accuracy of 0.83333%. I do not think it is possible to directly compare the results with the accuracy over the test set, because of extremely small number of test samples here.

3. Analysis of model predictions for the downloaded images.

The code for making predictions on my final model is located in the “Output Top 5 Softmax Probabilities For Each Image Found on the Web” of the Ipython notebook.

The top 5 predictions for each sample are:

[9.09472525e-01, 9.05273408e-02, 1.00598818e-07, 2.78746093e-08, 1.68915182e-09]
[**0**, 1, 5, 16, 2]

[1.00000000e+00, 5.67351606e-08, 7.80538882e-13, 3.73052080e-14, 1.10085696e-14],
[**34,** 35, 33, 40, 38]

[9.52687442e-01, 4.61670980e-02, 1.00169191e-03, 1.02722333e-04, 4.02699370e-05],
[11, **21**, 23, 25, 24]

[1.00000000e+00, 2.69040478e-17, 1.58684013e-18, 2.20687309e-22, 7.27828956e-26],
[**38**, 36, 34, 40, 39]

[1.00000000e+00, 2.58587777e-16, 5.10509378e-20, 1.14651143e-20, 1.34825310e-21]
[**40**, 33, 37, 2, 11]

[1.00000000e+00, 8.94058777e-15, 9.79277786e-18, 3.13628088e-19, 1.16772105e-19]
[**13,** 15, 9, 12, 3]

In 4 out of 6 cases the prediction is done with the probability of almost 1.0. In one case the proper prediction is done with the probability of 0.9 vs. next best chance of 0.09. Interestingly enough, the model has very strong prediction of 0.95 for the wrong case. The right class is the next best choice with the probability of 0.046.

Conclusion

Even simple CNN models have huge potential of producing strong results in the classification of the traffic signs. In this study I have been able to produce a model, which produced over 95% accuracy over the training set. I strongly believe that the main factor which limited the outcome of the project was not enough augmentation of the training data. I have been able to train the model to the very high accuracy on the training data, but the lack of enough training data probably limited my ability to train it better for test set.