Traffic Sign Recognition

The goals / steps of this project are the following:

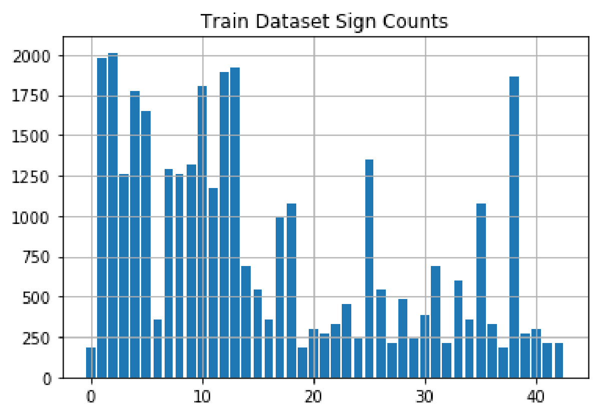
- Load the data set (see below for links to the project data set)
- Explore, summarize and visualize the data set
- Design, train and test a model architecture
- Use the model to make predictions on new images
- Analyze the softmax probabilities of the new images
- Summarize the results with a written report

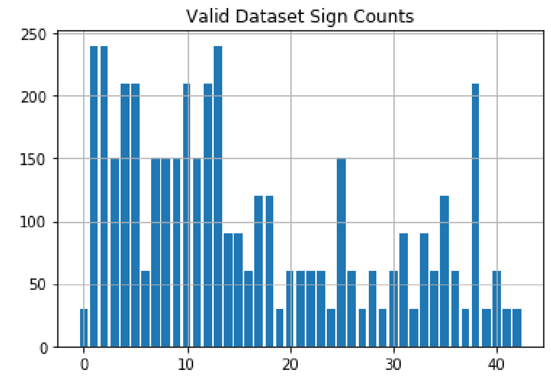
Data Set Summary & Exploration

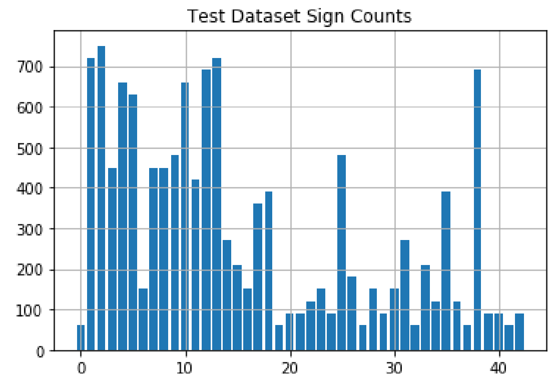
- The size of training set is 34799.
- The size of the validation set is 4410.
- The size of test set is 12630.
- The shape of a traffic sign image is 32x32x3
- The number of unique classes/labels in the data set is 43

Visualization of the dataset.

Here is an exploratory visualization of the data set. It is a bar chart showing how the data are distributed for each class.







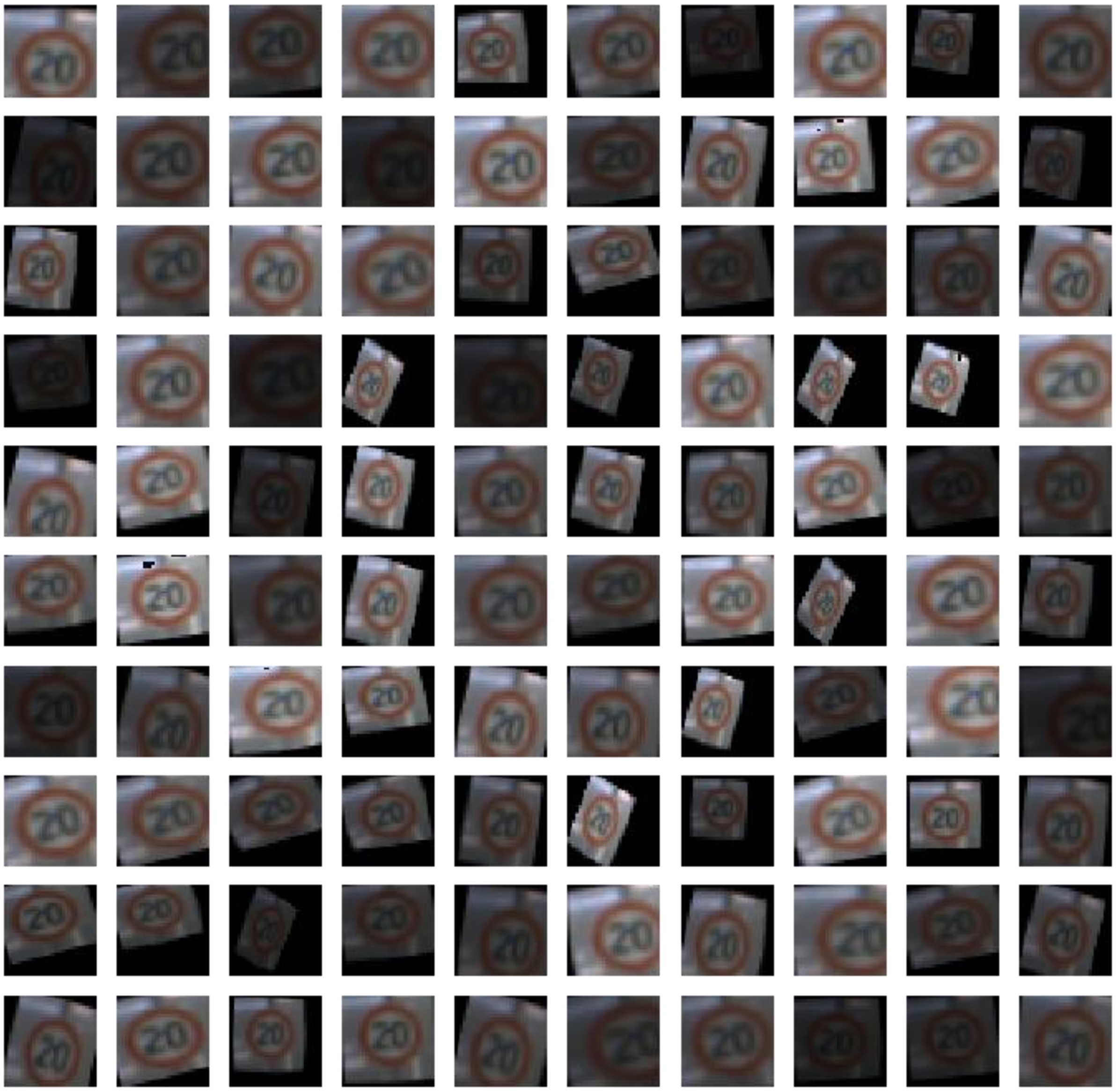
Data Preprocessing and Augmentation

Initially, I preprocessed data in two steps by converting the images to grayscale and normalizing them. I proceeded with this initial preprocessing and developed my CNN architecture. I was able to get the validation accuracy of 96% with this data set.

According to the histogram of data shown above, the distribution of data is not balanced. For some classes, the number of samples is around 2000 and for some other, it's less than 200 samples. In order to improve the accuracy further, I decided to generate some additional data. Data augmentation was performed by translation, rotation, affine transformation and changing the brightness of images randomly. I used the code in the following reference to augment my data.

https://chatbotslife.com/german-sign-classification-using-deep-learning-neural-networks-98-8-solution-d05656bf51ad

The following figure shows an example of transformation of an image for 100 different cases. Each transformation is a random combination of translation, rotation, affine transform and brightness change, with randomly selected parameters.



Neural Network Architecture

I started with original LeNet architecture and trained the network and was initially able to get the accuracy of around 93%. Then I modified the architecture and added two dropout layers in the fully connected part of the network. I was able to bump the accuracy to 96%. I tried adding more layers including convolution, fully connected and dropout layers but none of these led to a further improvement of the accuracy. My final model consisted of the following layers:

1. 5x5 convolution (32x32x1 in, 28x28x6 out)
2. ReLU
3. 2x2 max pool (28x28x6 in, 14x14x6 out)
4. 5x5 convolution (14x14x6 in, 10x10x16 out)
5. ReLU
6. 2x2 max pool (10x10x16 in, 5x5x16 out)
7. Flatten layers (5x5x16 -> 400)
8. Fully connected layer (Input = 400. Output = 200)
9. ReLU
10. Dropout layer
11. Fully connected layer (Input = 200. Output = 100)
12. ReLU
13. Dropout layer
14. Fully connected layer (Input = 100. Output = 43)

Describe how you trained your model

To train the model, I used "Adam optimizer" and the following hyper parameters for the network.

Batch size = 100
Epochs = 20
Learning rate = 0.002
keep_prob = 0.7

Describe the approach taken for finding a solution

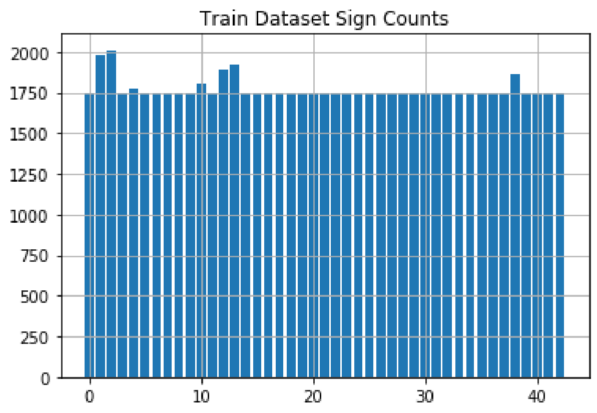
I converted the data to grayscale and normalized them in the beginning. Then used the original LeNet architecture and trained my network with the following parameters.

Batch size=100
Epochs=50
Learning rate=0.001
No Dropout

I obtained a validation accuracy of 0.93 using this configuration.
Next, I added dropout layers to the two fully connected layers of the network. With a keep_prob value of 0.5 I was able to reach the validation accuracy to 0.94. I tried different values of dropout, and obtained an accuracy of 0.96 with keep_prob=0.7.

Next, I tried to change the other hyper parameters such as batch size and number of epochs but was not able to reach any higher accuracy.

In order to further improve the accuracy, I augmented data and created a minimum of 1750 samples for every class. But I was not able to increase the accuracy using augmented data. Here is how the histogram of augmented data looks like.



I also tried modifying the model architecture by

- adding another layer of convolution;
- adding another fully connected layer;
- and adding dropout layers after the convolution layers;

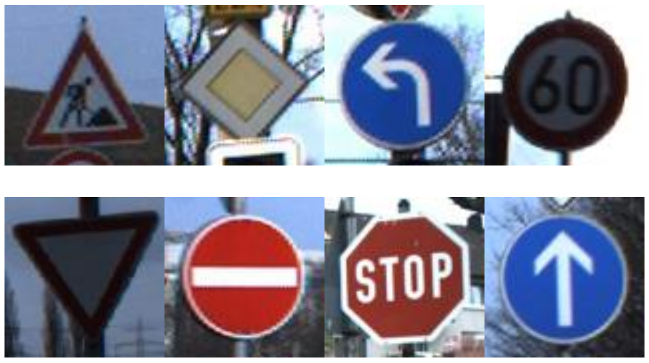
however, none of these changes really improved the accuracy of my model.

My final model results without augmented data were:

- **Training set accuracy:** 0.999
- **Validation set accuracy:** 0.965
- **Test set accuracy:** 0.943

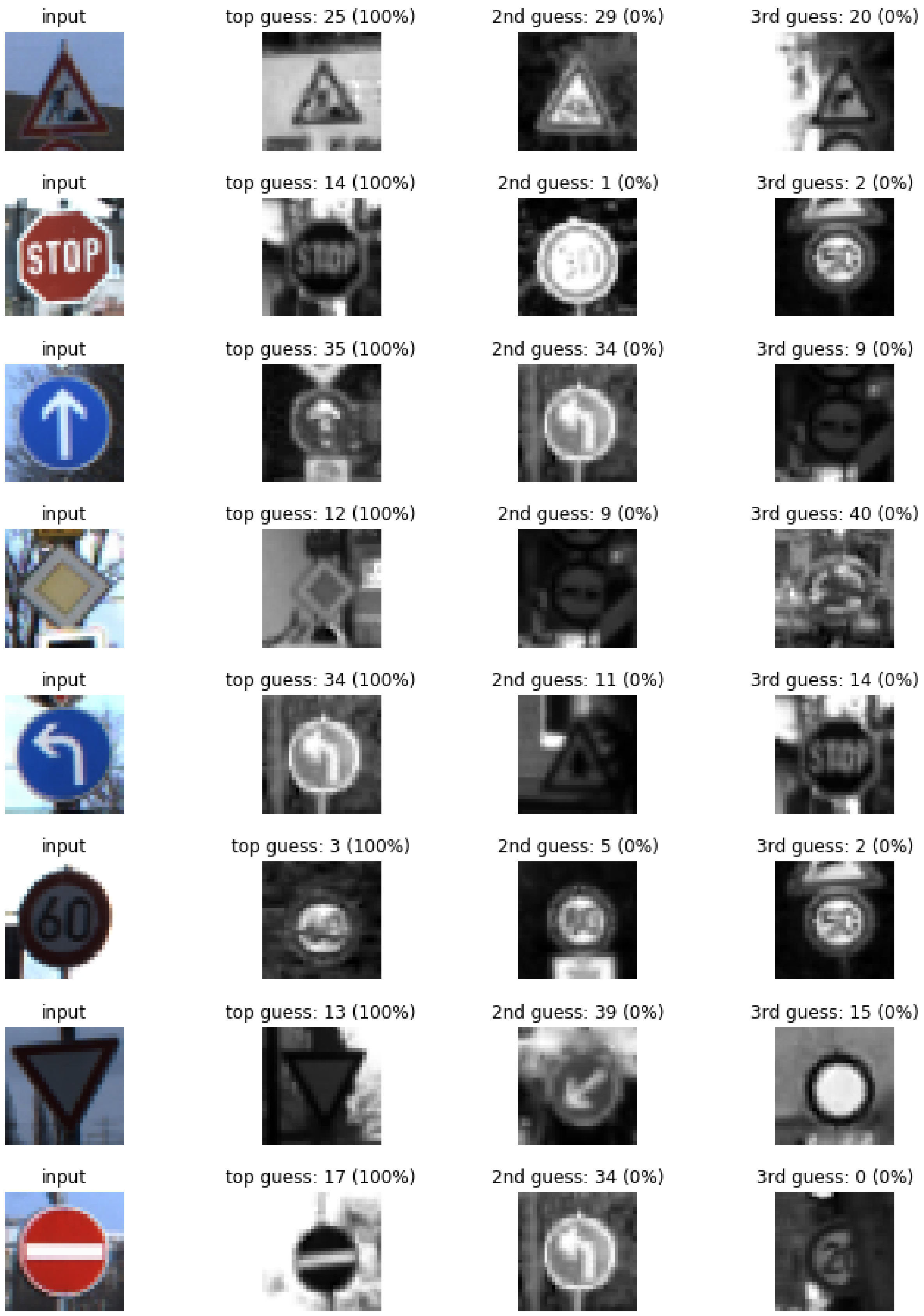
Test the Model on New Images

I found the following eight German traffic signs on the web:



The corresponding class labels for the selected images are **labels = [25, 14 , 35 , 12 , 34 , 3 , 13 , 17]** and the model was able to predict all classes correctly resulting in an accuracy of 1.00.

The following figure shows the top three selected classes for each image along with their probabilities. According to this figure, for all eight images, the correct class has been predicted with the probability of 100% and the probability of other classes is 0%, showing the high accuracy of our trained model.


input
top guess: 25 (100%)
2nd guess: 29 (0%)
3rd guess: 20 (0%)
input
top guess: 14 (100%)
2nd guess: 1 (0%)
3rd guess: 2 (0%)
input
top guess: 35 (100%)
2nd guess: 34 (0%)
3rd guess: 9 (0%)
input
top guess: 12 (100%)
2nd guess: 9 (0%)
3rd guess: 40 (0%)
input
top guess: 34 (100%)
2nd guess: 11 (0%)
3rd guess: 14 (0%)
input
top guess: 3 (100%)
2nd guess: 5 (0%)
3rd guess: 2 (0%)
input
top guess: 13 (100%)
2nd guess: 39 (0%)
3rd guess: 15 (0%)
input
top guess: 17 (100%)
2nd guess: 34 (0%)
3rd guess: 0 (0%)