**Traffic Sign Recognition**

**Build a Traffic Sign Recognition Project**

**Data Set Summary & Exploration**

The size of training set is 34799

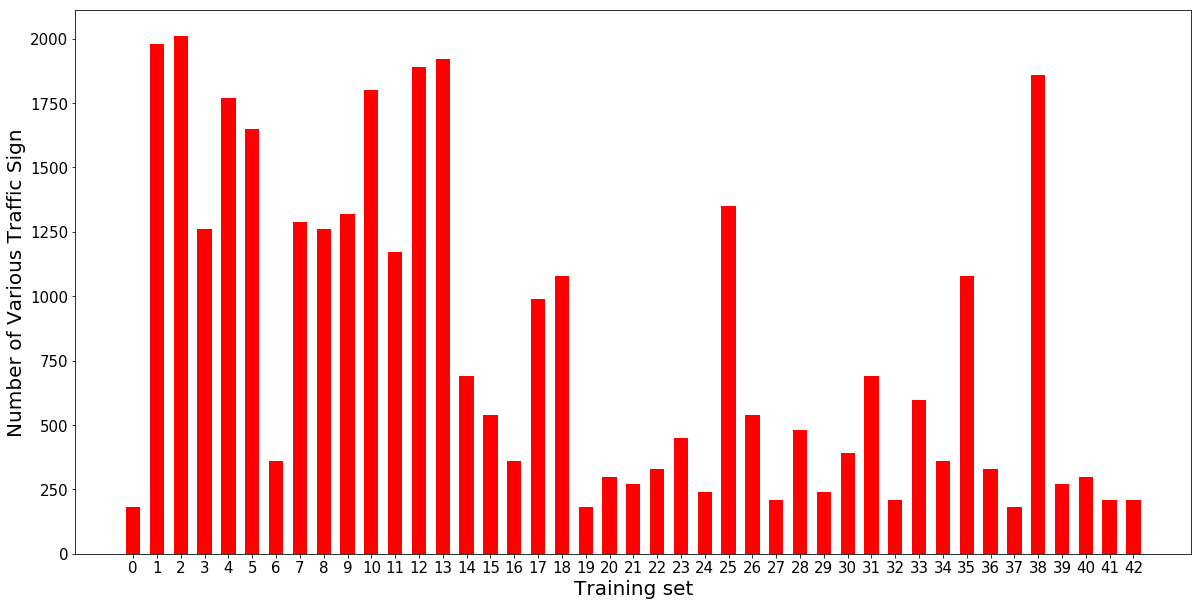
The size of the validation set is 4410

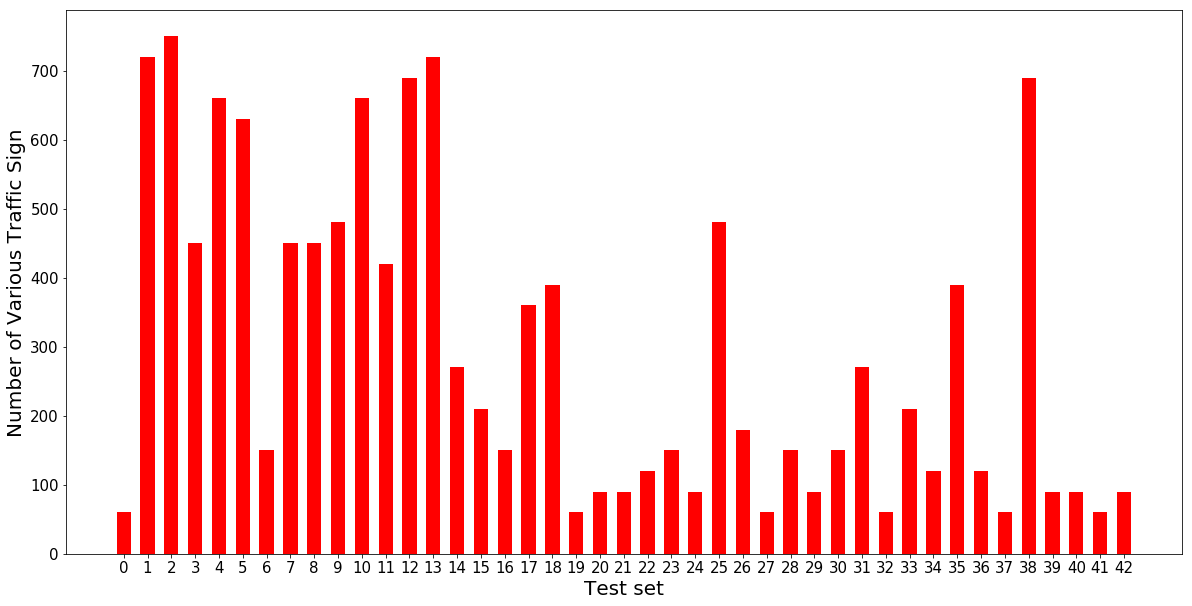
The size of test set is 12630

The shape of a traffic sign image is 32\*32\*3

The number of unique classes/labels in the data set is 43

Here is a bar chart showing how the data distribution. We can see the data is not balanced. Some traffic signs have more example than others. But training set has the same distribution as the test set.





**Design and Test a Model Architecture**

1.Preprocess data.

As a first step, I decided to normalized the image data because it will accelerate the training of my network. And then I one-hoted the labels.

I don't add more data to my data set , because as it seems to me the existed data set have get some augmented image as extra data and I got 95% accuracy on test set with this original data set.

2. Model Architecture.

My final model consisted of the following layers:

|  |  |
| --- | --- |
| Layer | Description |
| Input | 32x32x3 RGB image |
| Convolution 3x3 | 1x1 stride, same padding, outputs 32x32x16 |
| RELU |  |
| Convolution 3x3 | 1x1 stride, same padding, outputs 32x32x32 |
| RELU |  |
| Max pooling 2x2 | 2x2 stride, outputs 16x16x32 |
| Convolution 3x3 | 1x1 stride, same padding, outputs 16x16x64 |
| RELU |  |
| Convolution 3x3 | 1x1 stride, same padding, outputs 16x16x128 |
| RELU |  |
| Max pooling 2x2 | 2x2 stride, outputs 8x8x128 |
| Flatten layer | Outputs:8192 |
| Fully connected | Outputs:4096, Activation:Relu |
| Drop out | Keep prob = 0.7 |
| Fully connected | Outputs:1024, Activation:Relu |
| Drop out | Keep prob = 0.7 |
| Softmax | Outputs:43 |

3.Model training and Summary.

To train the model, I used adam as optimizer. I set 0.001 as learnning rate. As the cost stopping drop in epoch5, I train the model for epochs of 5. I was using a AWS server with K-80 GPU so I set the batch size pretty big as 512 to accelerate the training.

However the result is not very good at the beginning. My network is overfitting. So added drop out layer and results became better.

My final model results were:

My network is a simplified VGG network. Because VGG is working so good on various image classification problem, it's not a bad idea to copy some of it's architecture. As our data image is only 32\*32\*3, quite small, I think it's enough to use a shallower network. In fact, I achieved XXX accuracy on validation data set and 95% accuracy on test set.

**Model Testing**

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

5.png4.png3.png2.png1.png

All five images have some sort of distortion. Because when I cut them for net image, I’m using rectangle. But I have to convert the image into square 32\*32 so my network could be working. And there is shadow on some of the signs but it’s not that obviously after I shrink the images.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Image | 1 | 2 | 3 | 4 | 5 |
| Correct label | 13 | 25 | 29 | 27 | 23 |
| Correct Sign names | Yield | Road work | Bicycles Crossing | Pedestrians | Slippery road |
| Predicted label | 13 | 25 | 25 | 27 | 17 |
| Predicted Sign names | Yield | Road work | Road work | Pedestrians | No vehicles |

The model was able to correctly guess 3 of the 5 traffic signs, which gives an accuracy of 60%. I don’t understand why it gets me the wrong answer on image3. The wrong sign is quite different from the right one. But error on image 5 is forgivable. 2 sign looks pretty close due to the image is quite small.

I visualized the softmax probability for all 5 images.

As we can see, the network is pretty sure of it’s predicting. 2 of 5 images got probability almost close to 1 on their answers. Rest 3 got probability than 0.8 on their answers.

