Traffic Sign Recognition Project

The goals / steps of this project are the following:

- Load the German traffic signs image data set
- Explore, summarize and visualize the data set
- Preprocess and Augment the data set (if required)
- 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

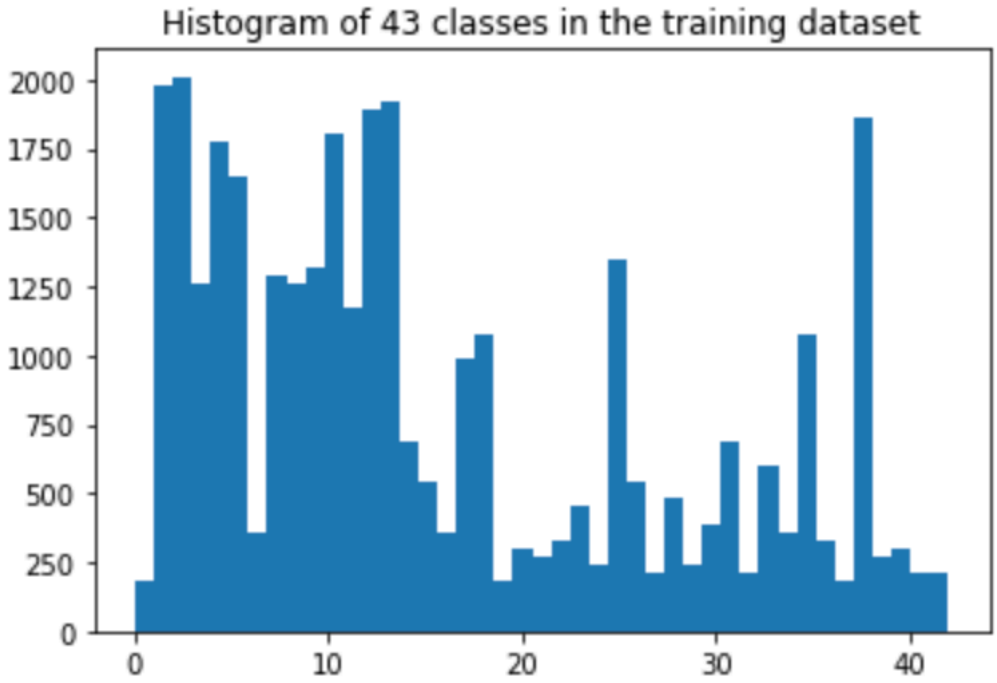
1. Provide a basic summary of the data set. In the code, the analysis should be done using python, numpy and/or pandas methods rather than hardcoding results manually.

I used the picke library to load and calculate summary statistics of the traffic signs data set:

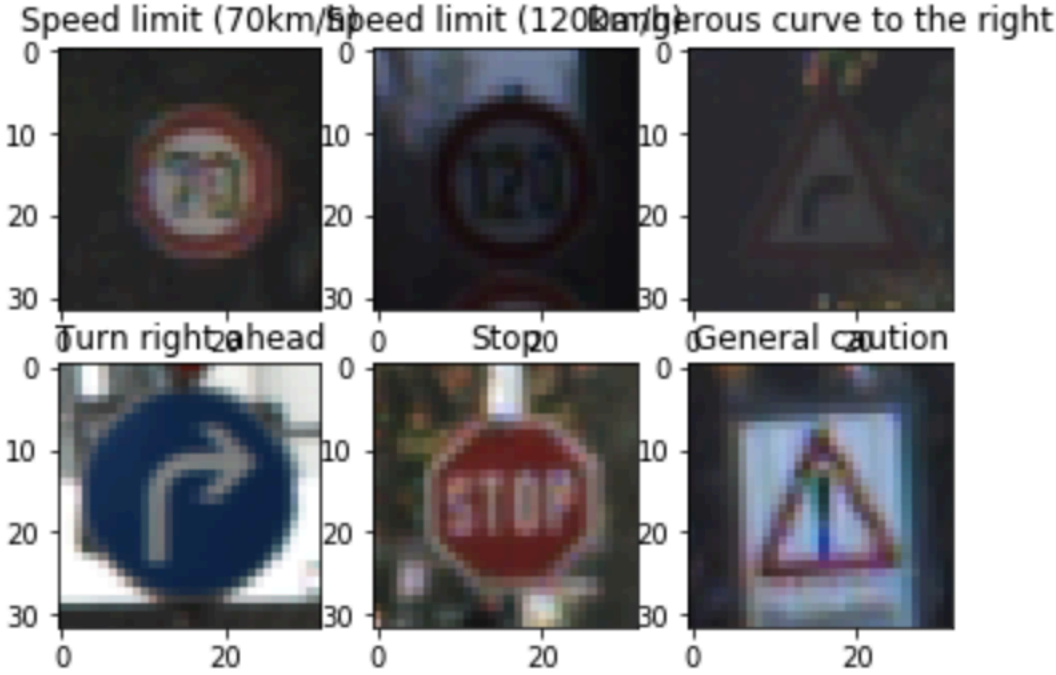
- The size of training set is 34799
- The size of the validation set is 12630
- The size of test set is 4410
- The shape of a traffic sign image is 32x32x3
- The number of unique classes/labels in the data set is 43

2. Include an exploratory visualization of the dataset.

Here is an exploratory visualization of the data set. It is a bar chart showing the number of images for each class.



Some images present in test dataset:

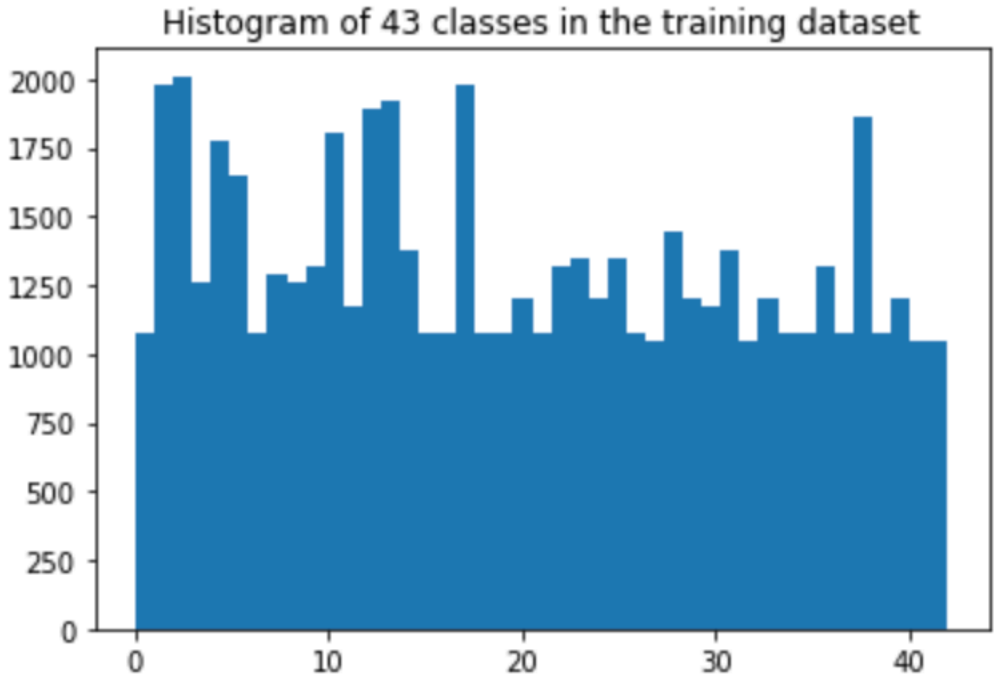


Design and Test a Model Architecture

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. (OPTIONAL: 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 a first step, I decided to generate more images using the same data set, as the data distribution across the different labels was not proper and because of which the model was not performing well for those labels, so I used the spicy library to generate new images by rotating them with a random angle between 0 and 30 and then appended all the generated images in the original data set.

Here is an exploratory visualization of the augmented data set. It is a bar chart showing the number of images for each class.



Then I used cv2 library to convert the images to gray-scale. As in traffic sign signals, shapes are more important than the color, therefore I choose to convert the colored images to the gray-scale images. Changing all the images to greyscale helped my model to learn better and faster.

Data Normalization: Then I normalized the gray-scaled data between 0.1 and 0.9. This prevents the overfitting. The value between 0.1 and 0.9 also avoids any potential problems incurred by allowing the data to be zero.

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.

My final model consisted of the following layers:

Layer	Description
Input	**32x32x1** gray-scale image
Convolution 5x5	1x1 stride, valid padding, outputs **28x28x20**
RELU	
Max pooling	2x2 stride, outputs **14x14x20**
Dropout	0.5
Convolution 3x3	1x1 stride, valid padding, outputs **12x12x40**
RELU	
Convolution 3x3	1x1 stride, valid padding, outputs **10x10x80**
RELU	

Layer	Description
Max pooling	2x2 stride, outputs **5x5x80**
Flatten	output - **2000**
Fully Connected 2000x120	output **120**
Dropout	0.7
Fully Connected 120x84	output **84**
Fully Connected 84x43	**output 43**
Softmax	

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.

To train the model, I used three convolution layers, three ReLU activation functions and two dropouts, one after second convolution layer and one after first fully-connected layer with keep_prob as 0.5 and 0.7 respectively.
Also, I used Adam optimizer with learning rate as **0.01**, batch size = **128** and **50** epochs to train the model.
For the hyper-parameters, I used a mean of **0** and standard deviation of **0.1**

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.

My final model results were:

- Validation set accuracy: **96.80%**
- Training set accuracy: **100%**
- Test set accuracy: **98.00%**

```
EPOCH 49 ...
Validation Accuracy = 0.961

EPOCH 50 ...
Validation Accuracy = 0.968

Model saved
```

```
import tensorflow as tf
saver = tf.train.Saver()
with tf.Session() as sess:
    saver.restore(sess, './traffic_sign_classifier_project')
    test_accuracy = evaluate(X_test, y_test)
    print("Test Accuracy = {:.3f}".format(test_accuracy))
```

```
Test Accuracy = 0.980
```

```
import tensorflow as tf
saver = tf.train.Saver()
with tf.Session() as sess:
    saver.restore(sess, './traffic_sign_classifier_project')
    train_accuracy = evaluate(X_train, y_train)
    print("Train Accuracy = {:.3f}".format(train_accuracy))
```

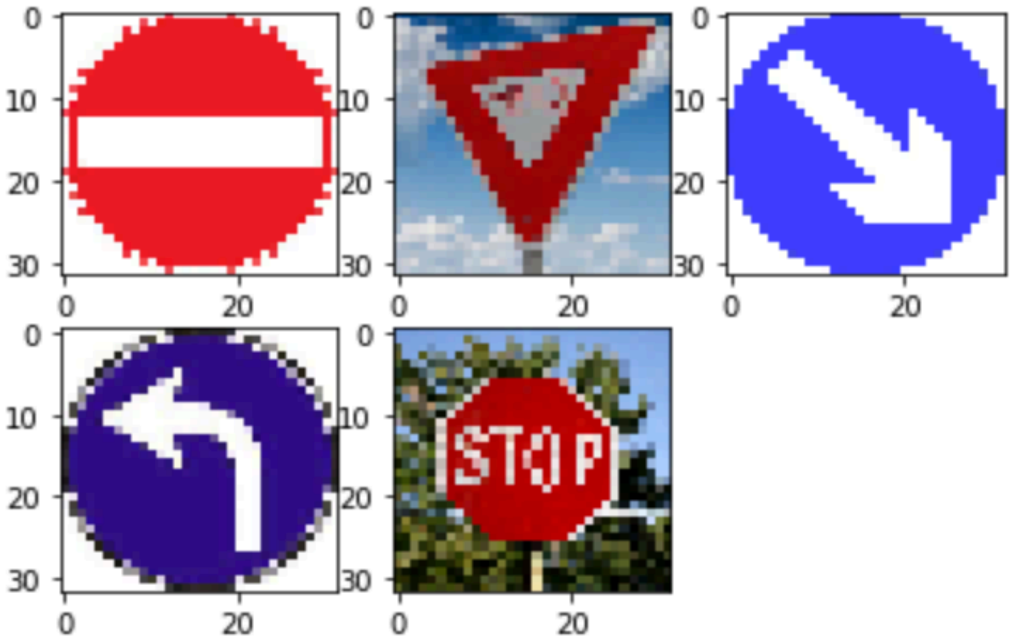
```
Train Accuracy = 1.000
```

To achieve this, First I used the LeNet Architecture as a base model. But I was getting accuracy of around 89% in that model after 50 epochs.
Then I augmented the data set and preprocessed the images by converting all the images to gray-scale and by normalizing those images to prevent overfitting.
With this architecture, I was getting the validation accuracy of around 92-93%. So then I decided to add one more convolution layer and one more fully connected layer in the model with 2 dropout layers, one with keep_prob as 0.5 and another with keep_prob as 0.7. And finally I was able to achieve the validation accuracy of 96.80%

If an iterative approach was chosen:
I would have added some more convolution layers in the model. Also I would have augmented the images better by zooming randomly at some points, cropping the corners, rotation, flipping and would have let the model to learn the colored images instead of greyscale. As we cannot determine if a traffic sign was red, yellow or green from grayscale image.

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:



I feel it would be difficult for the model to classify image 2 and 5 as there is a lot of background noise in these 2 images.

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 (OPTIONAL: Discuss the results in more detail as described in the "Stand Out Suggestions" part of the rubric).

Here are the results of the prediction:

Image	Prediction
No Entry	No Entry
Yield	Yield
Keep Right	Keep Right
Turn left ahead	Turn left ahead
Stop	Stop

In [75]:
```
prediction = tf.nn.softmax(logits)
with tf.Session() as sess:
    saver.restore(sess, tf.train.latest_checkpoint('.'))
    new_figs_class = sess.run(prediction, feed_dict={x: test_figures, keep_prob1 : 1.0, keep_prob2 : 1.0})
    for i in range(5):
        predict_label = np.argmax(new_figs_class[i])
        print('Image', i, 'prediction:',  predict_label, ', the true label is', true_label[i],'.')
```

```
Image 0 prediction: 17 , the true label is 17 .
Image 1 prediction: 13 , the true label is 13 .
Image 2 prediction: 38 , the true label is 38 .
Image 3 prediction: 34 , the true label is 34 .
Image 4 prediction: 14 , the true label is 14 .
```

```
with tf.Session() as sess:
    saver.restore(sess, tf.train.latest_checkpoint('.'))
    accuracy = evaluate(test_figures, true_label)
    percentage = accuracy * 100
    print("Prediction Accuracy for additional images is {:.3f} %".format(percentage))
```

```
Prediction Accuracy for additional images is 100.000 %
```

The model was able to correctly guess 5 of the 5 traffic signs, which gives an accuracy of 100%.

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.

The code for making predictions on my final model is located in the last cell of the Ipython notebook.

```
Image 0 probabilities: [9.99997377e-01   1.68161296e-06   7.87049771e-07   1.513681
13e-07
   7.33346255e-08] and predicted classes: [17 12 13 14 9]

Image 1 probabilities: [1.00000000e+00   1.16237196e-12   1.11976270e-12   9.045687
78e-14
   9.46824334e-16] and predicted classes: [13 32 1 12 17]

Image 2 probabilities: [1.00000000e+00   1.30109736e-12   5.29353308e-13   3.734053
81e-13
   9.26544675e-14] and predicted classes: [38 31 13 5 23]

Image 3 probabilities: [9.99942422e-01   2.13429757e-05   1.67836388e-05   8.315652
85e-06
   4.65215726e-06] and predicted classes: [34 35 23 38 28]

Image 4 probabilities: [9.99999046e-01   8.01656370e-07   1.19343440e-07   1.137143
80e-08
   7.78278597e-09] and predicted classes: [14 38 17 12 36]
```

For the first image, the top five soft max probabilities were:

Probability	Prediction
.99	No Entry
1.68161296e-06	Priority Road
. 7.87049771e-07	Yield
1.51368113e-07	Stop
7.33346255e-08	No Passing

Similarly, we can see from the above output, predicted classes is the top 5 soft max probabilities for each image.