Build a Traffic Sign Recognition Project

Goals:

The goals / steps of this project are the following:

- Load the 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 individually and describe how I addressed each point in my implementation.

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one.

- You're reading it!

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 pandas library to calculate summary statistics of the traffic signs data set:
    1. The size of training set is 34799
    2. The size of the validation set is 4410
    3. The size of test set is 12630
    4. The shape of a traffic sign image is (32, 32, 3)
    5. The number of unique classes/labels in the data set is 43

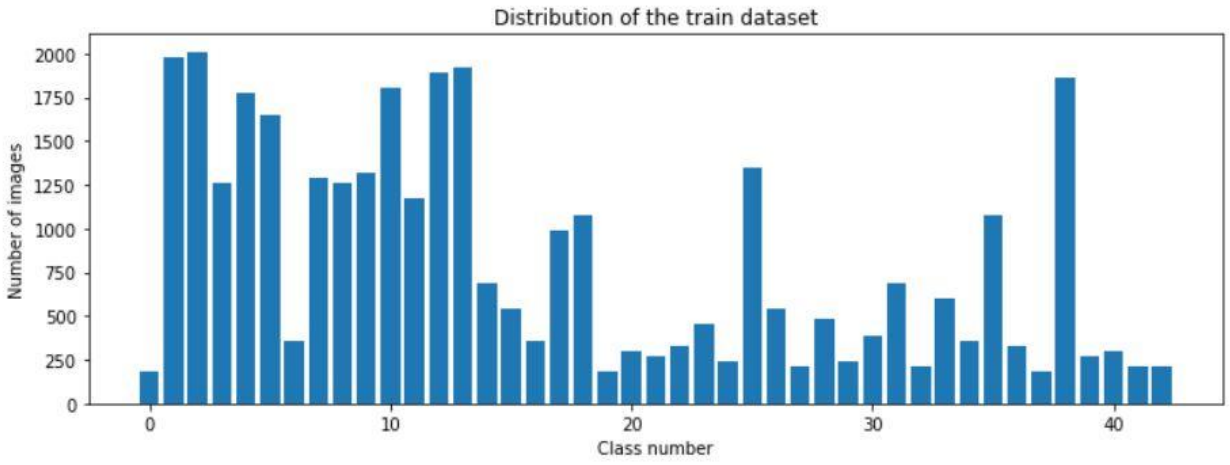
Provide a Basic Summary of the Data Set Using Python, Numpy and/or Pandas

In [2]:
```
### Replace each question mark with the appropriate value.
### Use python, pandas or numpy methods rather than hard coding the results

# TODO: Number of training examples
n_train = x_train.shape[0]

# TODO: Number of validation examples
n_validation = x_valid.shape[0]

# TODO: Number of testing examples.
n_test = x_test.shape[0]

# TODO: What's the shape of an traffic sign image?
image_shape = x_train.shape[1:]

# TODO: How many unique classes/labels there are in the dataset.
n_classes = len(set(y_train))

print("Number of training examples =", n_train)
print("Number of valid examples=", n_validation)
print("Number of testing examples =", n_test)
print("Image data shape =", image_shape)
print("Number of classes =", n_classes)
```

```
Number of training examples = 34799
Number of valid examples= 4410
Number of testing examples = 12630
Image data shape = (32, 32, 3)
Number of classes = 43
```

2. Include an exploratory visualization of the dataset.

Here is an exploratory visualization of the data set. It is a bar chart showing how the data –



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 performed three preprocess steps for my data. They are –
  a. Gray scaling data
  b. Equalizing data
  c. Normalizing data

For this I created three functions, each to perform the above task mentioned.

```
In [8]: def grayscale(image):
            return cv2.split(cv2.cvtColor(image, cv2.COLOR_RGB2YUV))[0]
```

```
In [9]: def equalize(image):
            return cv2.equalizeHist(image)
```

```
In [10]: def normalize_image(image):
             mini, maxi = np.min(image), np.max(image)
             return (image - mini) / (maxi - mini) * 2 - 1
```

Then I created a combined function to perform all these operations on a single image.

```
In [11]: def preprocess_image(image):
             return np.expand_dims(normalize_image(equalize(grayscale(image))), axis=2)
```

The Finally I created a function to operate these preprocessing over the entire dataset.

```
In [12]: def preprocess(dataset):
             return np.array([preprocess_image(im) for im in dataset])
```

The result of these preprocessing is as follows.

Then I performed **data augmentation.** The reason for this is, the number of training samples is relatively low. Moreover there are 43 classes, so the number per classes in the training data is very small which can make our network week. After augmentation the number of training examples were 215000.

In [19]:
```
unique, counts = np.unique(y_train, return_counts=True)

target = 5000
x_augmented = []
y_augmented = []

for cls, count in tqdm(list(zip(unique, counts)), 'Augmenting training dataset'):
    diff = target - count
    x_augmented += generate(x_train_n[np.where(y_train == cls)[0]], diff)
    y_augmented += [cls for _ in range(diff)]
```

```
Augmenting training dataset: 100%|| 43/43 [00:14<00:00,  3.05it/
s]
```

In [20]:
```
x_train_n = np.concatenate([x_train_n, np.array(x_augmented)])
y_train = np.concatenate([y_train, np.array(y_augmented)])
n_train = y_train.size
print('Final number of training examples', n_train)
```

```
Final number of training examples 215000
```

The resulted images were also good.

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:

```
In [25]: def LeNet(x, mu=0, sigma=0.1):
             # 3x3 convolution with ReLU activation
             conv1 = conv2d(x, 3, 1, 12)
             conv1 = tf.nn.relu(conv1)

             # 3x3 convolution with ReLU activation
             conv1 = conv2d(conv1, 3, 12, 24)
             conv1 = tf.nn.relu(conv1)
             pool1 = max_pool(conv1, 2)

             # 5x5 convolution with ReLU activation
             conv2 = conv2d(pool1, 5, 24, 36)
             conv2 = tf.nn.relu(conv2)

             # 5x5 convolution with ReLU activation
             conv2 = conv2d(conv2, 5, 36, 48)
             conv2 = tf.nn.relu(conv2)
             pool2 = max_pool(conv2, 2)

             # Flatten and Concatenate
             flat1 = tf.contrib.layers.flatten(pool1)
             flat2 = tf.contrib.layers.flatten(pool2)
             flattened = tf.concat([flat1, flat2], axis=1)

             # First fully connected with 512 neurons and dropout to reduce variance
             fully1 = fully(flattened, 5136, 512)
             fully1 = tf.nn.relu(fully1)
             fully1 = tf.nn.dropout(fully1, keep_prob)

             # Second fully connected with 256 neurons and dropout to reduce variance
             fully2 = fully(fully1, 512, 256)
             fully2 = tf.nn.relu(fully2)
             fully2 = tf.nn.dropout(fully2, keep_prob)

             # Output layer
             return fully(fully2, 256, n_classes), conv1, conv2
```

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 the following configuration -

batch_size = 32,

keep_prob = .5,

epochs = 100,

patience = 3,

modelname = 'LeNet'

Also for learning I used the following:

learning_rate = tf.train.exponential_decay(

learning_rate=0.0005,

global_step=global_step,

decay_steps=ceil(n_train / batch_size), # Decay every epoch

decay_rate=0.95,

staircase=True)

I used checkpoint in my training, so that it will stop if the validation accuracy does not increase for three iterations.

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:

- training set accuracy of 99.9%
- validation set accuracy of 99.3%
- test set accuracy of 97.047%

This training accuracy was obtained after the 2nd epoch and it did not improve for three more epochs, so, after 5th epoch my training terminated which eliminated the risk of overfitting.

The model I used was similar to LeNet5 model. It is already a well-known model for image classification and it proved again.

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.

- I actually used eight images from web.

All images are of size 32x32 and clean. There labels are 11,1,12,38,34,18,25

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:

Actual	Predicted
11	11
1	1
12	12
38	38
34	34
18	18
25	25

So, I get an 100% accuracy on new data!!

```
INFO:tensorflow:Restoring parameters from .\traffic_net.ckpt
```

input

top guess: 11 (100%)

2nd guess: 0 (0%)

3rd guess: 1 (0%)

input

top guess: 1 (100%)

2nd guess: 0 (0%)

3rd guess: 2 (0%)

input

top guess: 12 (100%)

2nd guess: 0 (0%)

3rd guess: 1 (0%)

input

top guess: 38 (100%)

2nd guess: 0 (0%)

3rd guess: 1 (0%)

input

top guess: 34 (100%)

2nd guess: 0 (0%)

3rd guess: 1 (0%)

input

top guess: 18 (100%)

2nd guess: 0 (0%)

3rd guess: 1 (0%)

input

top guess: 25 (100%)

2nd guess: 0 (0%)

3rd guess: 1 (0%)

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. (OPTIONAL: as described in the "Stand Out Suggestions" part of the rubric, visualizations can also be provided such as bar charts)

The code for making predictions on my final model is located in the 11th cell of the Ipython notebook.

For the first image,

Predictions	Probability
11	1.0
0	0.0
1	0.0
2	0.0
3	0.0

For the second image,

Predictions	Probability
1	1.0
0	0.0
2	0.0
3	0.0
4	0.0

For the third image,

Predictions	Probability
12	1.0
0	0.0
1	0.0
2	0.0
3	0.0

For the fourth image,

Predictions	Probability
38	1.0
0	0.0
1	0.0
2	0.0
3	0.0

For the fifth image,

Predictions	Probability
34	1.0
0	0.0
1	0.0
2	0.0
3	0.0

For the sixth image,

Predictions	Probability
18	1.0
0	0.0
1	0.0
2	0.0
3	0.0

For the seventh image,

Predictions	Probability
18	1.0
0	0.0
1	0.0
2	0.0
3	0.0

```
print(top5_prediction)
```

```
TopKV2(values=array([[1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.]], dtype=float32), indices=array([[11,  0,  1,  2,  3],
       [ 1,  0,  2,  3,  4],
       [12,  0,  1,  2,  3],
       [38,  0,  1,  2,  3],
       [34,  0,  1,  2,  3],
       [18,  0,  1,  2,  3],
       [25,  0,  1,  2,  3]]))
```