Traffic Sign Recognition

The goals / steps of this project are the following:

```
1. Load the data set (see below for links to the project data set
)
2. Explore, summarize and visualize the data set
3. Design, train and test a model architecture
4. Use the model to make predictions on new images
5. Analyze the softmax probabilities of the new images
6. 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.

The code for this step is contained in the IPython notebook under the heading "Step 1: Dataset Summary & Exploration" and subsection of "Provide a Basic Summary of the Data Set Using Python, Numpy and/or Pandas"

I used the numpy 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 **12630**
3. The size of test set is **4410**
4. The shape of a traffic sign image is **(32, 32)** and number of channels is **3** (color)
5. The number of unique classes/labels in the data set is **43**

2. Include an exploratory visualization of the dataset.

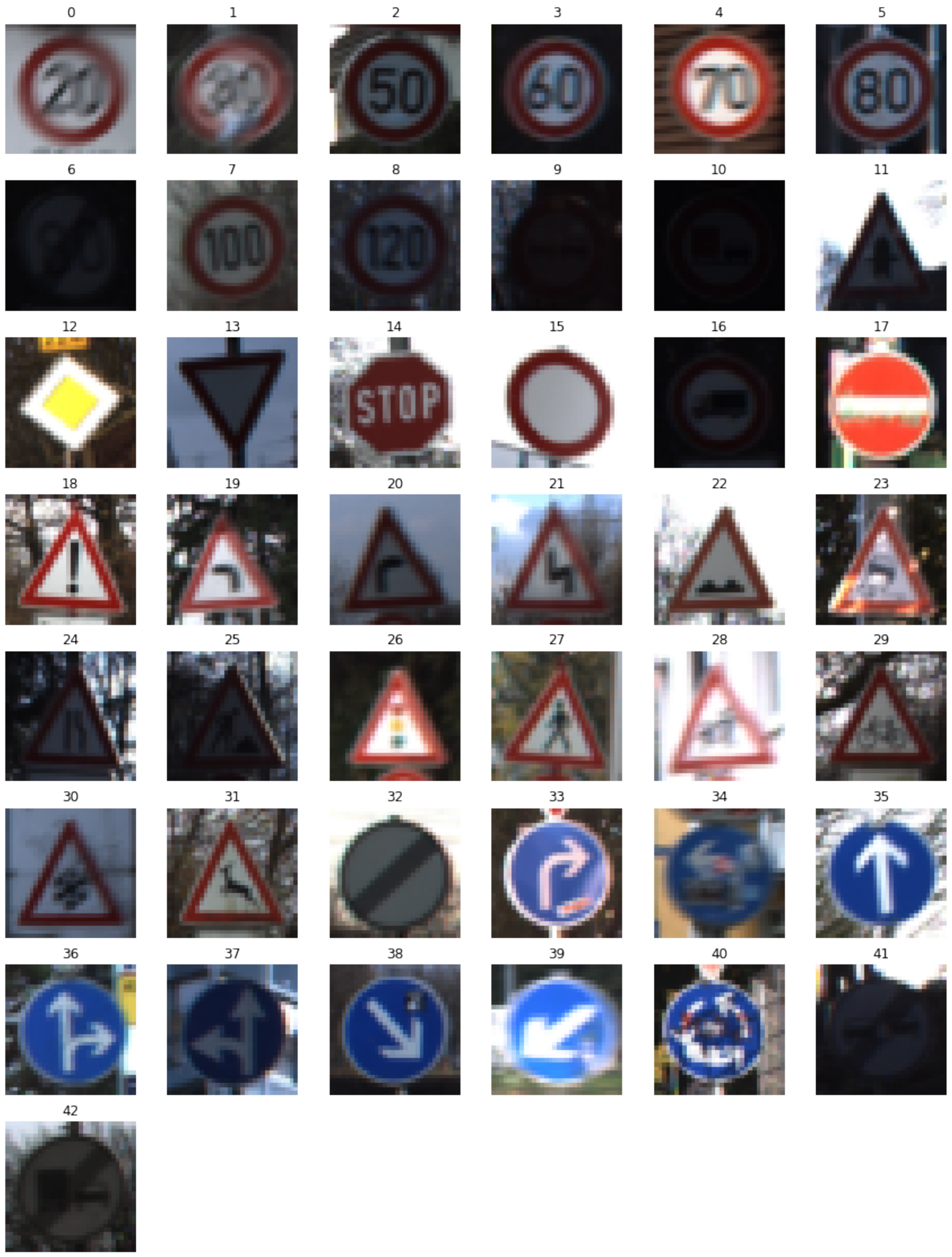
The code for this step is contained in the IPython notebook under the heading "Include an exploratory visualization of the dataset"

Here is the list of classes and their indices:

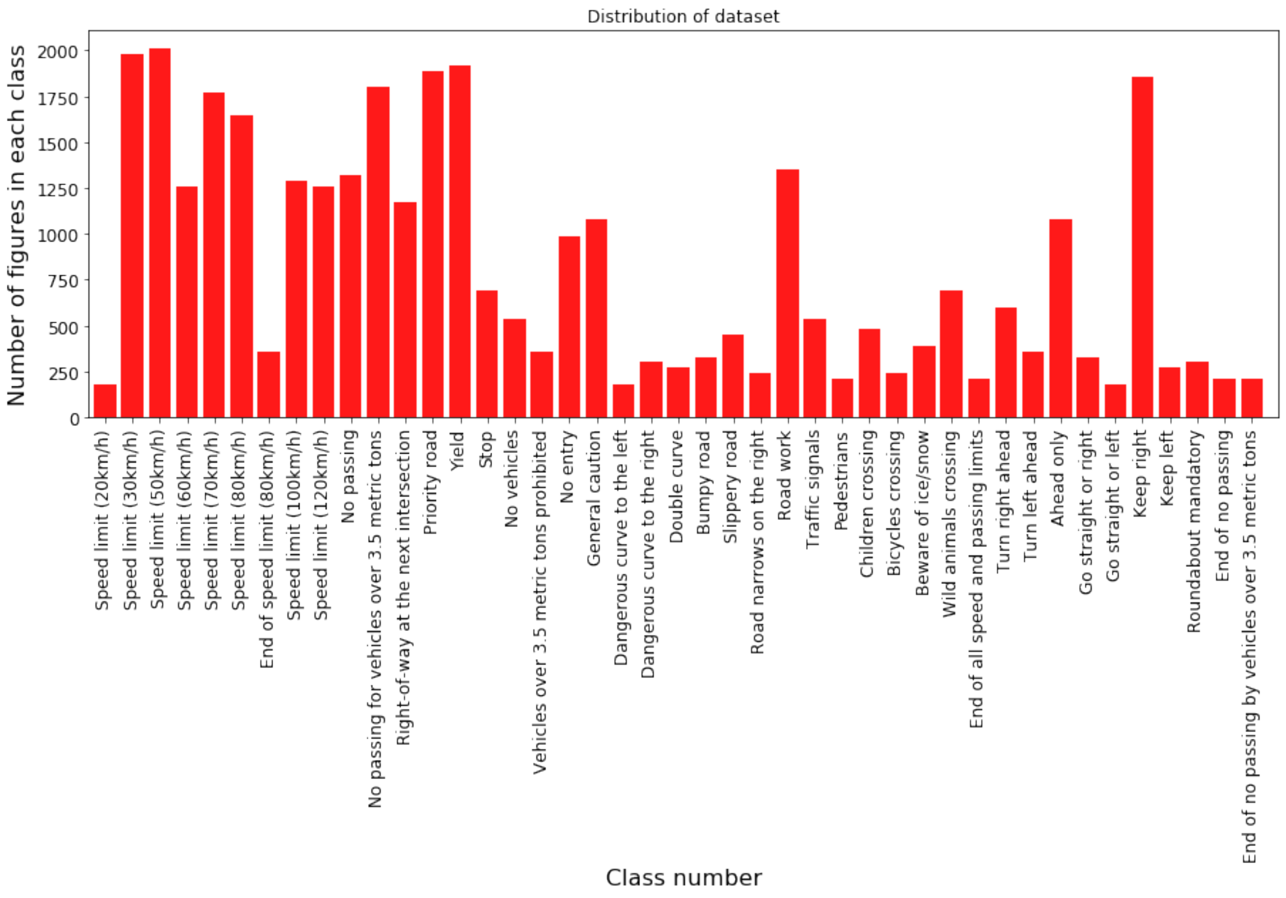
```
    0                                       Speed limit (20km/h)
    1                                       Speed limit (30km/h)
```

```
2                                   Speed limit (50km/h)
3                                   Speed limit (60km/h)
4                                   Speed limit (70km/h)
5                                   Speed limit (80km/h)
6                            End of speed limit (80km/h)
7                                  Speed limit (100km/h)
8                                  Speed limit (120km/h)
9                                             No passing
10          No passing for vehicles over 3.5 metric tons
11                 Right-of-way at the next intersection
12                                         Priority road
13                                                 Yield
14                                                  Stop
15                                           No vehicles
16              Vehicles over 3.5 metric tons prohibited
17                                              No entry
18                                       General caution
19                           Dangerous curve to the left
20                          Dangerous curve to the right
21                                          Double curve
22                                            Bumpy road
23                                         Slippery road
24                             Road narrows on the right
25                                             Road work
26                                       Traffic signals
27                                           Pedestrians
28                                     Children crossing
29                                     Bicycles crossing
30                                    Beware of ice/snow
31                                 Wild animals crossing
32                   End of all speed and passing limits
33                                      Turn right ahead
34                                       Turn left ahead
35                                            Ahead only
36                                  Go straight or right
37                                   Go straight or left
38                                            Keep right
39                                             Keep left
40                                  Roundabout mandatory
41                                     End of no passing
42     End of no passing by vehicles over 3.5 metric ...
```

Here I draw the last figure in each class



and here is the distribution of dataset: Number of images in each class



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.)

1.1 Data Augmentation:

As it is obvious from figure above, there is a large variatoin in distribution of images per classes, some classes such as "go straight or left" have less than 200 images where others such as "speed limit(30km/h) have close to 2000, so images clearly do not have a fair distribution of all types. I did augmentation, which added extra samples to the training set by applying different forms of transformations, including random rotation, x and y translation to the original image. At the end we have same number of images per classes(2000 images per class).

1.2 Convert to gray Scale

I converted images from color scale to gray scale.

1.3 Normalization

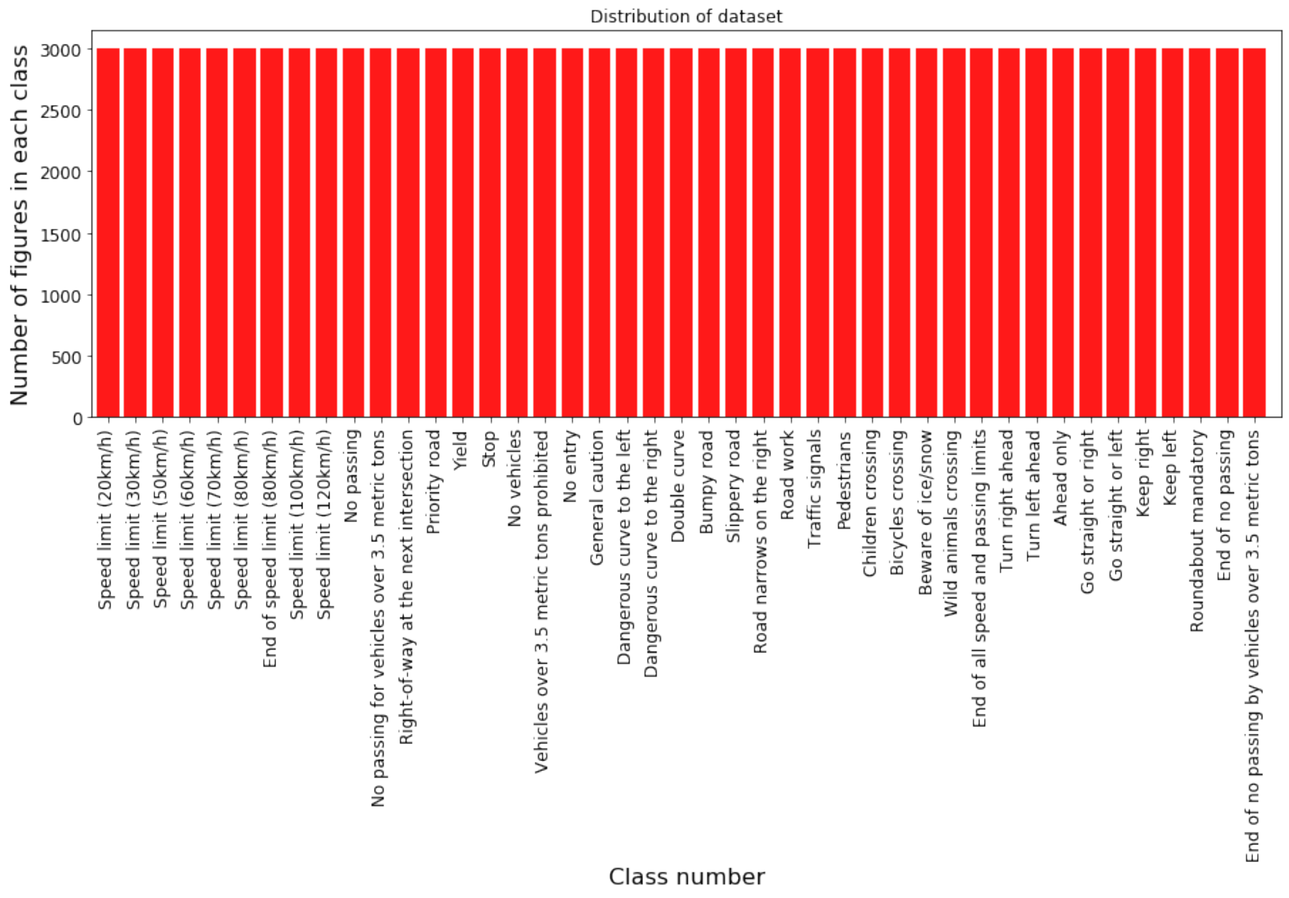
I normalized the image data to remove differences of contrast and brightness between images. Here normalizing is standardizing which is making the data zero-centered by subtracting its mean and normalizing it by its standard deviation to have a unit variance.

Now, augmented images are generated using described pre-processed data(gray scale and normalized). In data augmentation, I used pre-defined and built-in functions in OpenCV (cv2.getRotationMatrix2D and cv2.warpAffine)

After applying all type of pre-processing and data augmentaation, here is the summary of result:

```
Training Set:   129000 samples
Validation Set: 4410 samples
Test Set:       12630 samples
```

now here is the distribution of dataset after preprocessing and augmentation: Number of images in each class



As you can see in above figure there is a fairly distribution among images in each classes

2. Model Architecture

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
Convolutional Layer1(5x5)         1x1 stride, Valid padding, Output =
28x28x6
Activation 1                      RELU
Max Pooling 1                     2x2 stride, Output = 14x14x6
-----------------------------------------------------------------
-------
Convolutional Layer2(5x5)         1x1 stride, Valid padding, Output =
10x10x16
Activation 2                      RELU
Max Pooling 2                     2x2 stride, Output = 5x5x16
-----------------------------------------------------------------
-------
Flatten                           Output = 400
Fully connected Layer1            Output = 120
Activation 3                      RELU
Dropout 1                         Keep Probability = 0.7
-----------------------------------------------------------------
-------
Fully connected Layer2            Output = 84
Activation 4                      RELU
Dropout 2                         Keep Probability = 0.7
-----------------------------------------------------------------
-------
Fully connected Layer 3           Output = 43
```

3. Training Model

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.

In order to tune the hyper parameters including:

```
1. Adam gradient optimizer learning rate: 0.0009
2. Regularization strength
3. Number of epochs: 60
4. Batch size: 128
5. Dropout probability: 0.7
```

I used an Adam optimizer.

4. Approach taken for finding a solution

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.

BASELINE Architecture (LeNet):

I started the project (as mentioned by udacity) by running LeNet architecture on the German Traffic Sign Dataset. In order to do that I only modified the followings:

```
Changing the X_train shape such that it has 1 channel
Changing the number of classes from 10 in the MNIST dataset to 43
which is the length of the test set.
```

data is already splitted in to validation and training sets, so there is no need to manually doing this by using sklearn.model library. I assigned 20% of the data for validation and 80% is left for training.

I chose the LeNet structure as the baseline of my network to classify traffic signs. It is simple and small enough to be trained quickly for making decisions regarding the image pre-processing and data augmentations.

"Here is the log of training:"

Training...

EPOCH 1 ... Validation Accuracy = 0.822

```
EPOCH 2 ... Validation Accuracy = 0.869

EPOCH 3 ... Validation Accuracy = 0.890

EPOCH 4 ... Validation Accuracy = 0.897

EPOCH 5 ... Validation Accuracy = 0.910

EPOCH 6 ... Validation Accuracy = 0.908

EPOCH 7 ... Validation Accuracy = 0.912

EPOCH 8 ... Validation Accuracy = 0.904

EPOCH 9 ... Validation Accuracy = 0.921

EPOCH 10 ... Validation Accuracy = 0.921

EPOCH 11 ... Validation Accuracy = 0.923

EPOCH 12 ... Validation Accuracy = 0.935

EPOCH 13 ... Validation Accuracy = 0.933

EPOCH 14 ... Validation Accuracy = 0.930

EPOCH 15 ... Validation Accuracy = 0.939

EPOCH 16 ... Validation Accuracy = 0.940

EPOCH 17 ... Validation Accuracy = 0.921

EPOCH 18 ... Validation Accuracy = 0.933

EPOCH 19 ... Validation Accuracy = 0.939

EPOCH 20 ... Validation Accuracy = 0.940

EPOCH 21 ... Validation Accuracy = 0.930

EPOCH 22 ... Validation Accuracy = 0.935

EPOCH 23 ... Validation Accuracy = 0.934

EPOCH 24 ... Validation Accuracy = 0.943

EPOCH 25 ... Validation Accuracy = 0.940

EPOCH 26 ... Validation Accuracy = 0.942
EPOCH 27 ... Validation Accuracy = 0.941

EPOCH 28 ... Validation Accuracy = 0.946

EPOCH 29 ... Validation Accuracy = 0.952

EPOCH 30 ... Validation Accuracy = 0.952

EPOCH 31 ... Validation Accuracy = 0.948

EPOCH 32 ... Validation Accuracy = 0.946

EPOCH 33 ... Validation Accuracy = 0.937

EPOCH 34 ... Validation Accuracy = 0.949

EPOCH 35 ... Validation Accuracy = 0.949

EPOCH 36 ... Validation Accuracy = 0.943

EPOCH 37 ... Validation Accuracy = 0.957

EPOCH 38 ... Validation Accuracy = 0.960

EPOCH 39 ... Validation Accuracy = 0.954

EPOCH 40 ... Validation Accuracy = 0.946

EPOCH 41 ... Validation Accuracy = 0.947

EPOCH 42 ... Validation Accuracy = 0.942

EPOCH 43 ... Validation Accuracy = 0.955

EPOCH 44 ... Validation Accuracy = 0.959

EPOCH 45 ... Validation Accuracy = 0.946

EPOCH 46 ... Validation Accuracy = 0.955

EPOCH 47 ... Validation Accuracy = 0.939

EPOCH 48 ... Validation Accuracy = 0.959

EPOCH 49 ... Validation Accuracy = 0.956

EPOCH 50 ... Validation Accuracy = 0.949

EPOCH 51 ... Validation Accuracy = 0.952
```

EPOCH 52 ... Validation Accuracy = 0.944

EPOCH 53 ... Validation Accuracy = 0.949

EPOCH 54 ... Validation Accuracy = 0.952

EPOCH 55 ... Validation Accuracy = 0.956

EPOCH 56 ... Validation Accuracy = 0.947

EPOCH 57 ... Validation Accuracy = 0.944

EPOCH 58 ... Validation Accuracy = 0.949

EPOCH 59 ... Validation Accuracy = 0.939

EPOCH 60 ... Validation Accuracy = 0.953

Model saved In [26]:

My final model results were:

```
training set accuracy of 99.9%
validation set accuracy of 95.3%
test set accuracy of 93.1%
```

If an iterative approach was chosen: What was the first architecture that was tried and why was it chosen?

I chose the LeNet structure as the baseline of my network to classify traffic signs. It is simple and small enough to be trained quickly for making decisions regarding the image pre-processing and data augmentations.

What were some problems with the initial architecture?

The training data was able to achieve very good accuracy but validation data accuracy was 85%. So, it had overfitting problem.

How was the architecture adjusted and why was it adjusted? Typical adjustments could include choosing a different model architecture, adding or taking away layers (pooling, dropout, convolution, etc.), using an activation function or changing the activation function. One common justification for adjusting an architecture would be due to over fitting or under fitting. A high accuracy on the training set but low accuracy on the validation set indicates over fitting; a low accuracy on both sets indicates under fitting.

To overcome over-fitting, I have added dropouts to the model at various layer and experimented different keep probabilities. The best result seem to be placing two dropouts between the three fully connected layer and keep probability of 0.7.

Which parameters were tuned? How were they adjusted and why?

At first I chose a large value for learning rate to increase the speed of the training but the result was not good enough, so I lowered to small value. The keep_prob = 0.7 and was the best option, for data augmentation, I used two transformation of rotation and translation. At the end, I generated 3000 images for each class.

What are some of the important design choices and why were they chosen? For example, why might a convolution layer work well with this problem? How might a dropout layer help with creating a successful model?

I used the dropout layers in fully connected layers to help it to reduce over-fitting of the data

If a well known architecture was chosen: What architecture was chosen?

The LeNet model with dropout layers is great.

Why did you believe it would be relevant to the traffic sign application?

It has great and promising results on predicting 32x32x3 images on letters and numbers. I beleive recognise objects in similar size as street signs.

How does the final model's accuracy on the training, validation and test set provide evidence that the model is working well?

the accuracy on test data set is 95.3%. It is higher than the minimum requirement of what udacity asked(93%) for the validation set. However, the model is still over-fitting.

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.

first image:

The first image is a Priority road sign. The image is slightly blurry but through the program, the image will be clear and the model should work well.

second image:

The second image is a Turn left ahead sign. The image is slightly blurry and tilted.

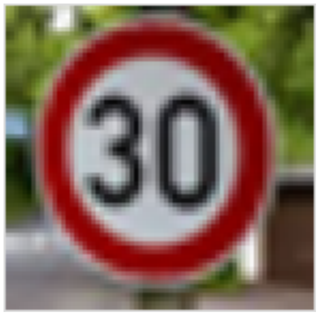
third image:

The third image is a Speed limit (60km/h) sign. The lighting is certainly different.

fourth image:

The fourth image is a Right-of-way at the next intersection sign. The lighting is certainly different.

fifth image:



The fifth image is a Speed limit (20km/h) sign.

I wrote some lines code to capture the class id of images as an input to my model so the accuracy is 100%.

2. 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).

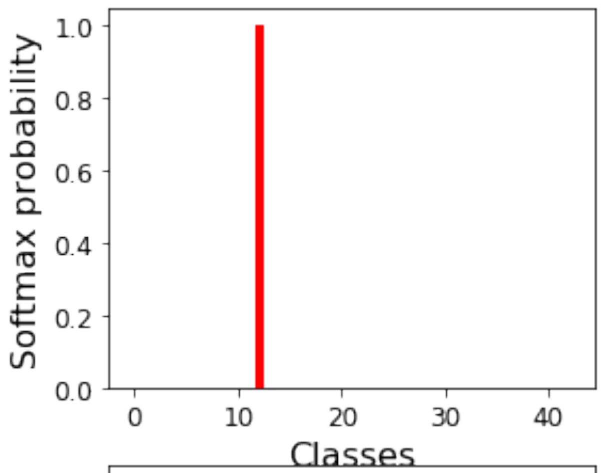
The code for making predictions on my final model is in IPython notebook under the heading "Predict the Sign Type" for Each Image

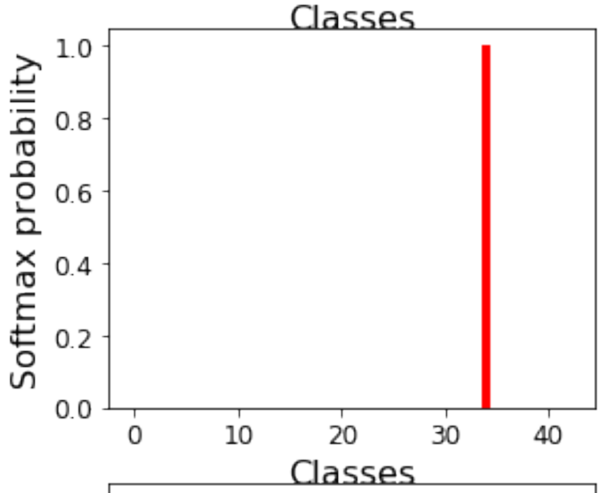
Here are my results for prediction:

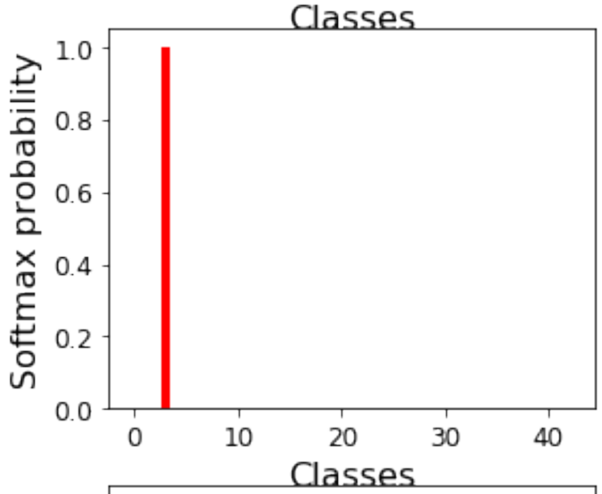
```
image                                       Prediction
-----------------------------------------------------------------------
------------
Priority road                               Priority road
Turn left ahead                             Turn left ahead
Speed limit (60km/h)                        Speed limit (60km/h)
Right-of-way at the next intersection       Right-of-way at the next
intersection
Speed limit (20km/h)                        Speed limit (20km/h)
```

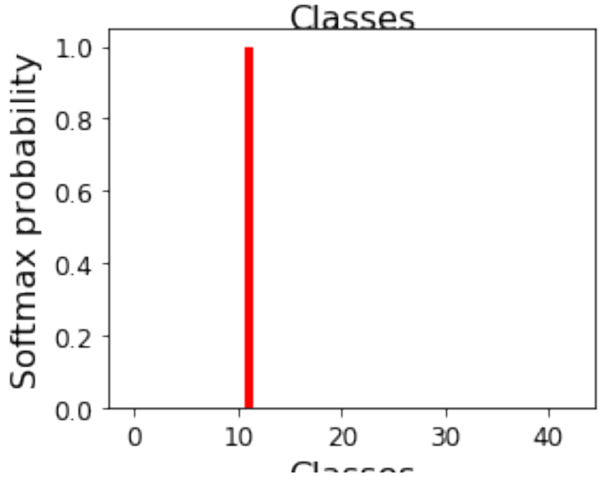
my model wcould predict correctly all five images of traffic signs.

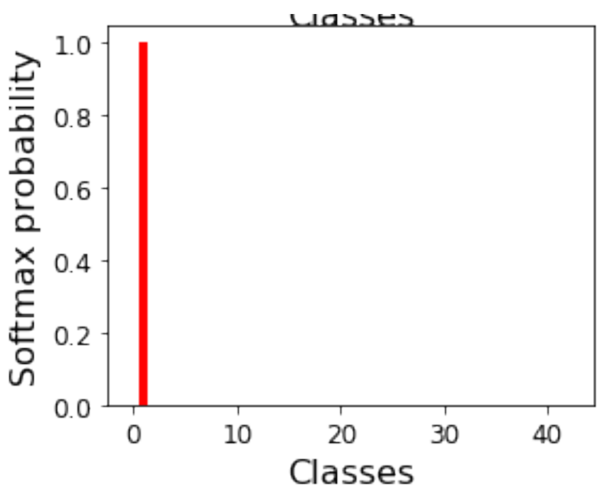
3. Describe how certain the model is when predicting on each of the five new images by looking at the softmax probabilities for each prediction and identify where in your code softmax probabilities were outputted. Provide the top 5 softmax probabilities for each image along with the sign type of each probability. (OPTIONAL: as described in the "Stand Out Suggestions" part of the rubric, visualizations can also be provided such as bar charts)


Softmax probability
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
Classes


Softmax probability
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
Classes


Softmax probability
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
Classes


Softmax probability
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40



for all of these five images, the model could get the probability of 1 for correct class

Required:

You provide a good analysis of the results on the newly acquired images but forgot to compare the accuracy on the new set of images to that on the old test set and tell if your model is overfitting or underfitting.

The accuracy on the captured images(5 images that I fed to model) is 100% while it was:

```
1. training set accuracy of 99.9%
2. validation set accuracy of 95.3%
3. test set accuracy of 93.1%
```

therefore It seems the model is working well on chosen images, Since the test accuracy(93.1) is less than validation accuracy(95.3%) and the result I got for these images(100%) is higher, so the model can be improved in the future by adding more layers and tunning better hyperparameters and adding regularization, so it seems the model is slightly overfitting.

Just a little thing missing : You should clearly discuss how certain or uncertain your model is of its prediction.

I run model on five images that I chose, and here is the result of

my_top_k = sess.run(top_k, feed_dict={x: images_test, keep_prob: 1.})

```
TopKV2(values=array([
[  1.00000000e+00,   0.00000000e+00,   0.00000000e+00, 0.00000000
e+00,   0.00000000e+00],
[  1.00000000e+00,   1.12621469e-15,   2.65482870e-21, 3.89836245
e-32,   4.56915729e-38],
[  1.00000000e+00,   3.79276299e-09,   9.29154739e-11, 6.39640355
e-11,   8.53514850e-13],
[  1.00000000e+00,   2.14781445e-21,   0.00000000e+00, 0.00000000
e+00,   0.00000000e+00],
[  1.00000000e+00,   3.67713319e-19,   4.75852056e-22, 2.57694976
e-36,   0.00000000e+00]], dtype=float32),

indices=array([[12,  0,  1,  2,  3],
               [34, 38, 36, 13, 40],
               [ 3,  5,  2, 25, 10],
               [11, 30,  0,  1,  2],
               [ 1,  5,  2, 38,  0]], dtype=int32))
```

Looking just at the first row we get [1.00000000e+00,0.00000000e+00,0.00000000e+00, 0.00000000e+00,0.00000000e+00], we can confirm these are the 5 largest probabilities. You'll also notice [12, 0, 1, 2, 3] are the corresponding indices. As it is clear, for all of these five images, the highest probability is 1, so that is why it gave me accuracy of 100%.

```
In [ ]:
```