Traffic Sign Recognition

Writeup

Build a Traffic Sign Recognition Project

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

Step 1: Dataset Summary & Exploration

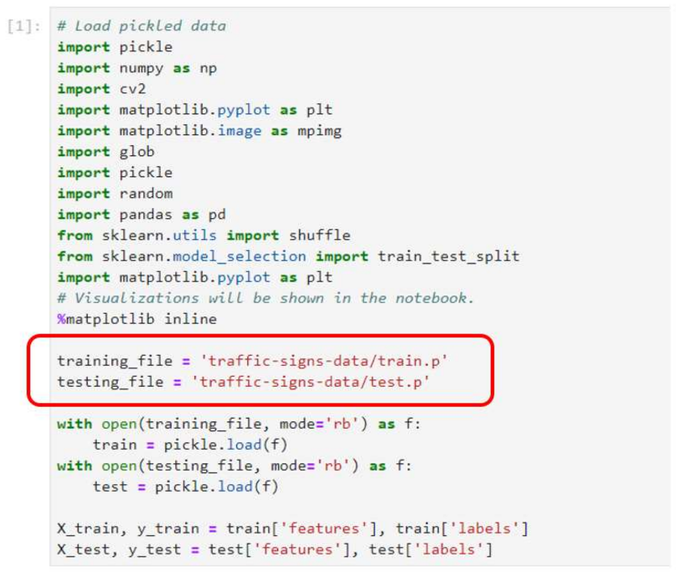
I used the pandas library to calculate summary statistics of the traffic signs data set:

The pickled data is a dictionary with 4 key/value pairs:

- 'features' is a 4D array containing raw pixel data of the traffic sign images, (num examples, width, height, channels).
- 'labels' is a 1D array containing the label/class id of the traffic sign. The file signnames.csv contains id -> name mappings for each id.
- 'sizes' is a list containing tuples, (width, height) representing the original width and height the image.
- 'coords' is a list containing tuples, (x1, y1, x2, y2) representing coordinates of a bounding box around the sign in the image.

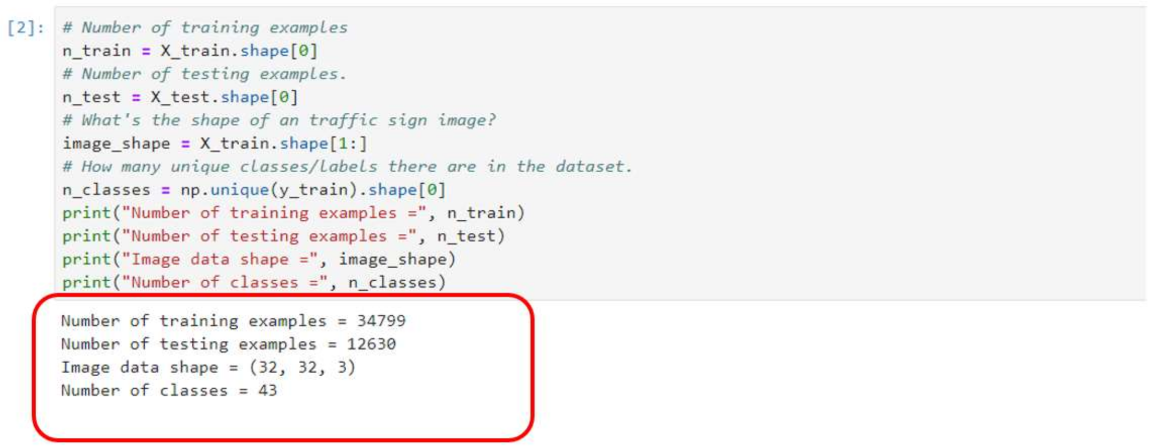
  THESE COORDINATES ASSUME THE ORIGINAL IMAGE. THE PICKLED DATA CONTAINS RESIZED VERSIONS (32 by 32) OF THESE IMAGES

Step 0: Load The Data



```
[1]: # Load pickled data
import pickle
import numpy as np
import cv2
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import glob
import pickle
import random
import pandas as pd
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
# Visualizations will be shown in the notebook.
%matplotlib inline

training_file = 'traffic-signs-data/train.p'
testing_file = 'traffic-signs-data/test.p'

with open(training_file, mode='rb') as f:
    train = pickle.load(f)
with open(testing_file, mode='rb') as f:
    test = pickle.load(f)

X_train, y_train = train['features'], train['labels']
X_test, y_test = test['features'], test['labels']
```

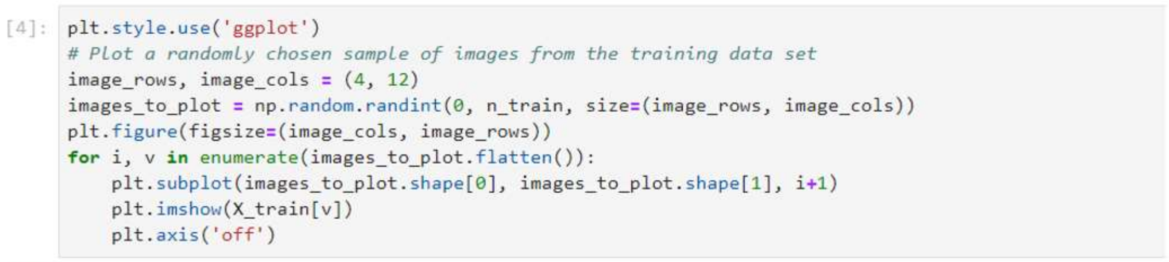
Provide a Basic Summary of the Data Set Using Python, Numpy and/or Pandas



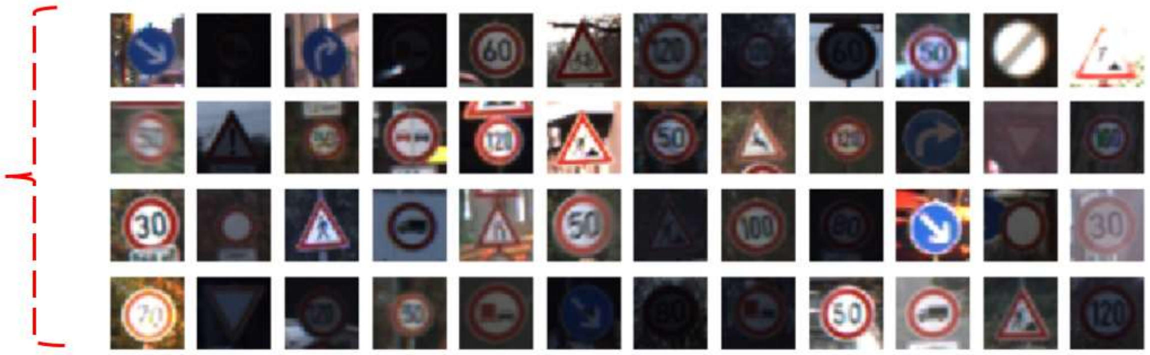
```
[2]: # Number of training examples
n_train = X_train.shape[0]
# Number of testing examples.
n_test = X_test.shape[0]
# What's the shape of an traffic sign image?
image_shape = X_train.shape[1:]
# How many unique classes/labels there are in the dataset.
n_classes = np.unique(y_train).shape[0]
print("Number of training examples =", n_train)
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 testing examples = 12630
Image data shape = (32, 32, 3)
Number of classes = 43
```

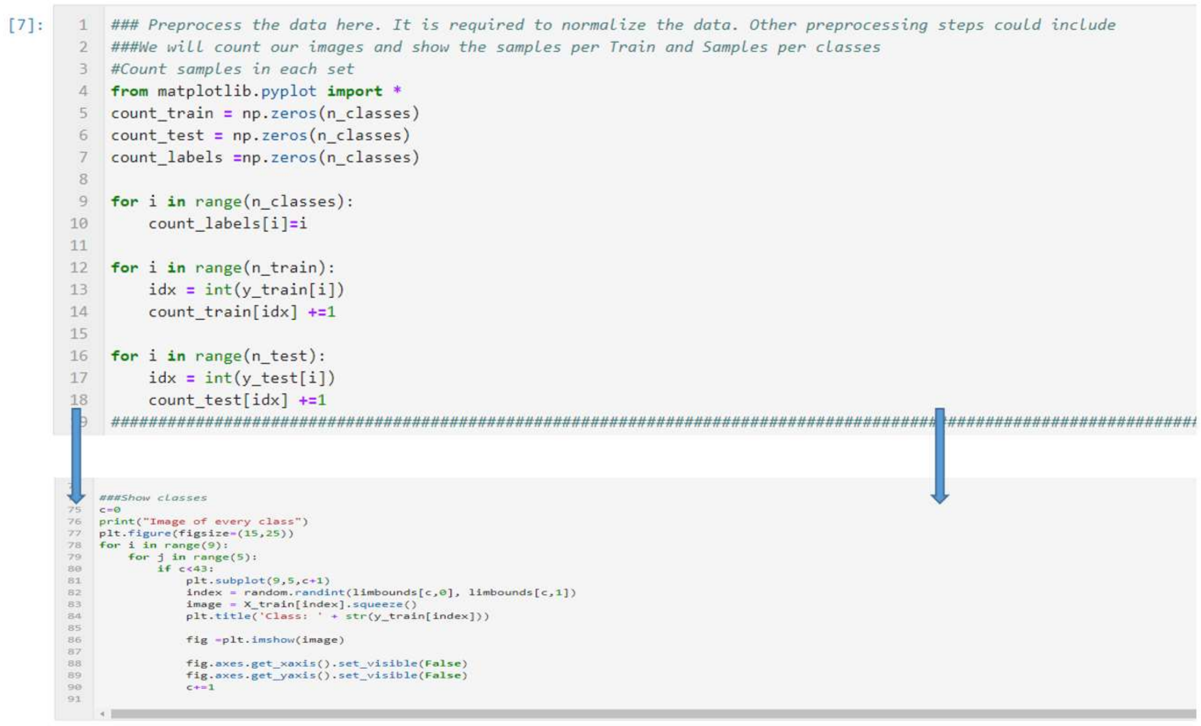
2. Include an exploratory visualization of the dataset.



```
[4]: plt.style.use('ggplot')
     # Plot a randomly chosen sample of images from the training data set
     image_rows, image_cols = (4, 12)
     images_to_plot = np.random.randint(0, n_train, size=(image_rows, image_cols))
     plt.figure(figsize=(image_cols, image_rows))
     for i, v in enumerate(images_to_plot.flatten()):
         plt.subplot(images_to_plot.shape[0], images_to_plot.shape[1], i+1)
         plt.imshow(X_train[v])
         plt.axis('off')
```

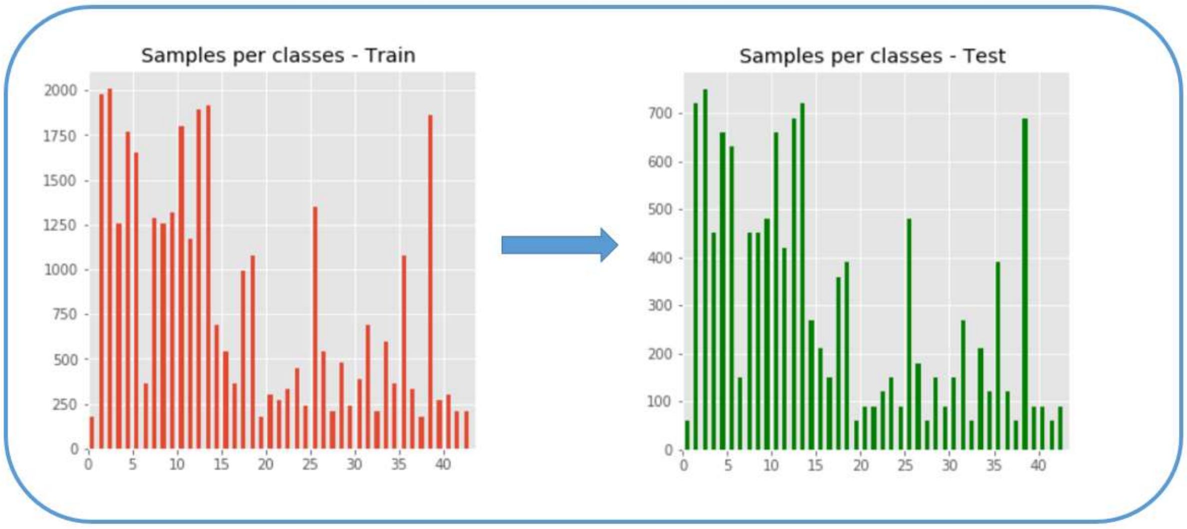


Samples per class and Samples per train

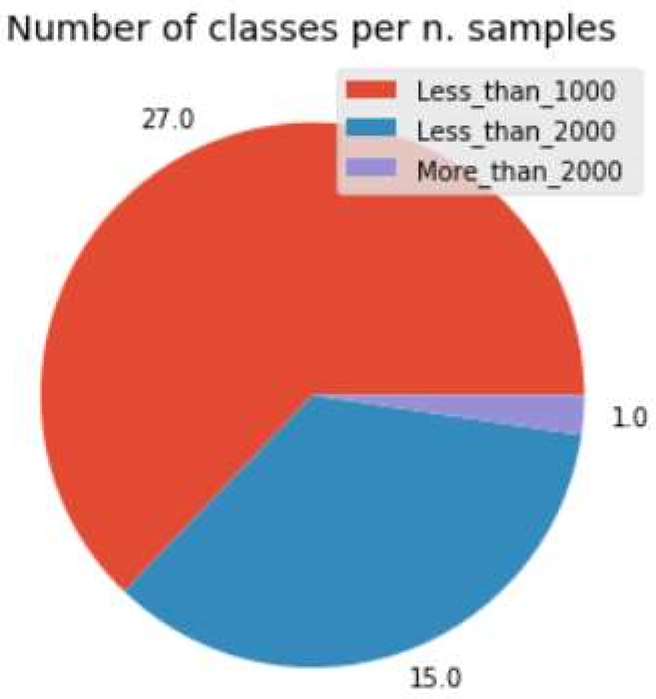


```
[7]: ### Preprocess the data here. It is required to normalize the data. Other preprocessing steps could include
     ###We will count our images and show the samples per Train and Samples per classes
     #Count samples in each set
     from matplotlib.pyplot import *
     count_train = np.zeros(n_classes)
     count_test = np.zeros(n_classes)
     count_labels =np.zeros(n_classes)

     for i in range(n_classes):
         count_labels[i]=i

     for i in range(n_train):
         idx = int(y_train[i])
         count_train[idx] +=1

     for i in range(n_test):
         idx = int(y_test[i])
         count_test[idx] +=1
     ######################################################################################################################
```

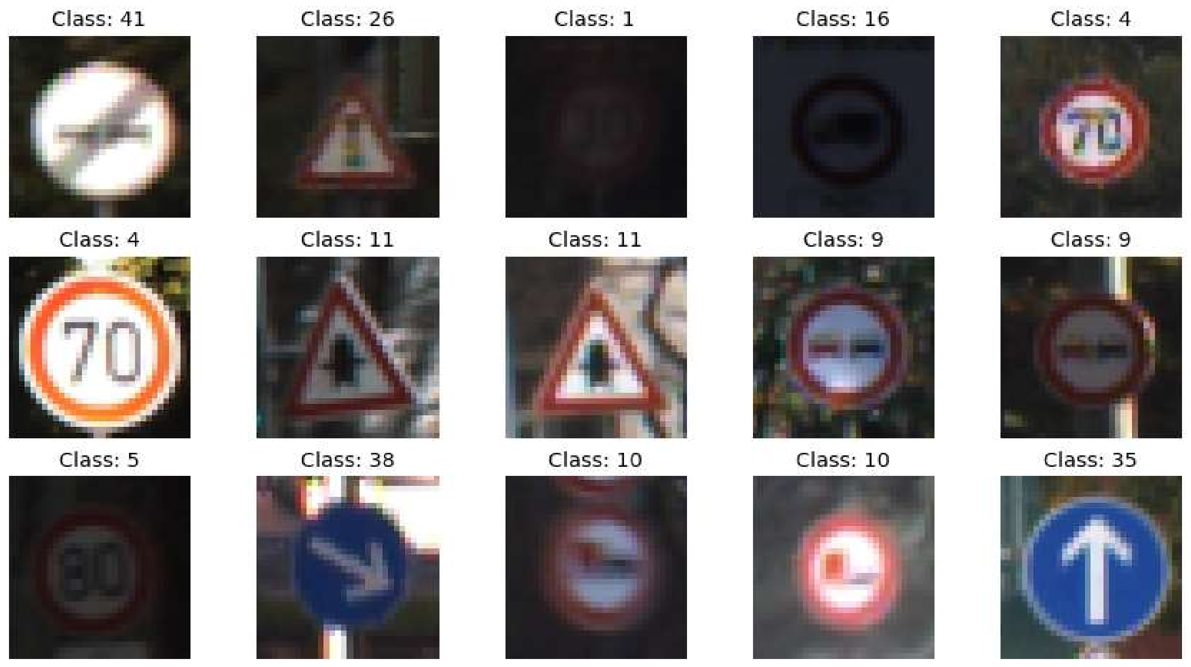
```
###Show classes
c=0
print("Image of every class")
plt.figure(figsize=(15,25))
for i in range(9):
    for j in range(5):
        if c<43:
            plt.subplot(9,5,c+1)
            index = random.randint(limbounds[c,0], limbounds[c,1])
            image = X_train[index].squeeze()
            plt.title('Class: ' + str(y_train[index]))

            fig =plt.imshow(image)

            fig.axes.get_xaxis().set_visible(False)
            fig.axes.get_yaxis().set_visible(False)
            c+=1

```

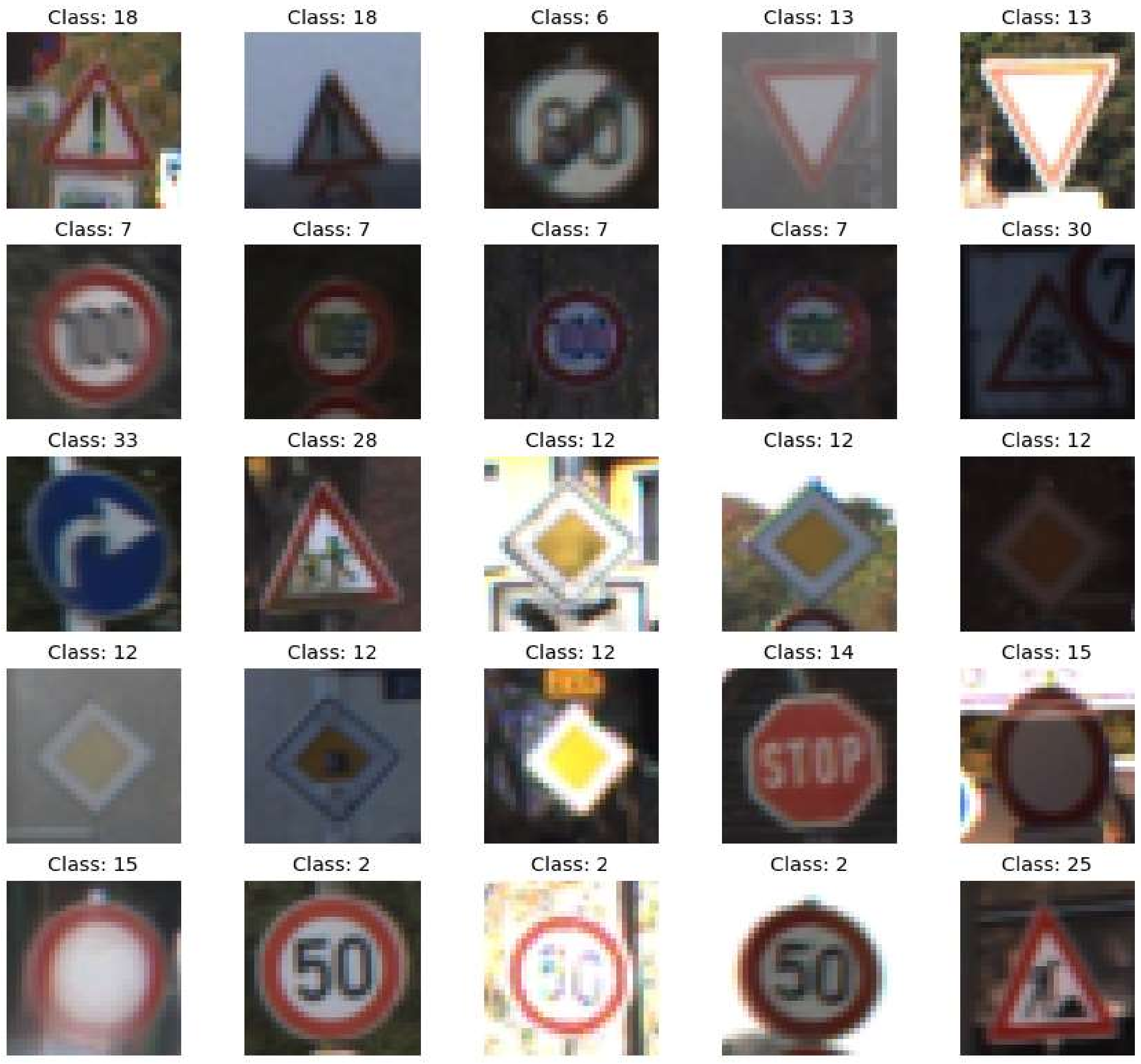


```
[7]: <matplotlib.legend.Legend at 0x2a576b81780>
```



From the 43 classes, 26 have less than 1000 samples 11 have more equal than 1000 samples and less than 2000 6 have more than 2000 samples.

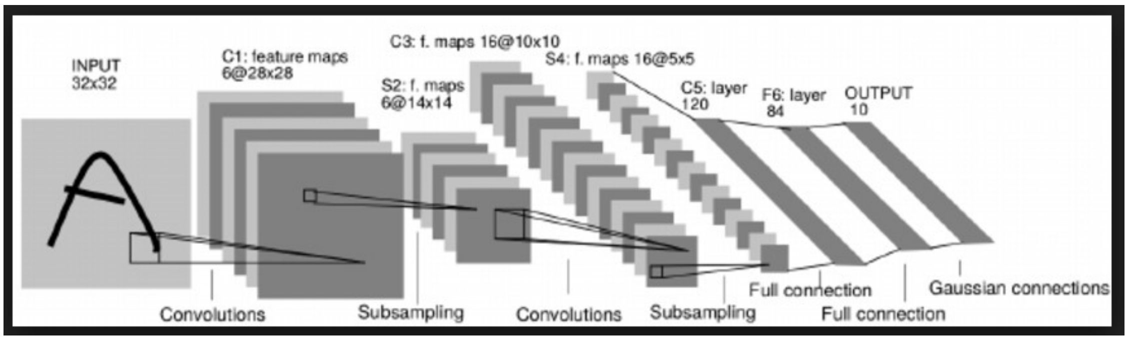

Class: 41
Class: 26
Class: 1
Class: 16
Class: 4
Class: 4
Class: 11
Class: 11
Class: 9
Class: 9
Class: 5
Class: 38
Class: 10
Class: 10
Class: 35


Class: 18
Class: 18
Class: 6
Class: 13
Class: 13
Class: 7
Class: 7
Class: 7
Class: 7
Class: 30
Class: 33
Class: 28
Class: 12
Class: 12
Class: 12
Class: 12
Class: 12
Class: 12
Class: 14
Class: 15
Class: 15
Class: 2
Class: 2
Class: 2
Class: 25

There are a lot of classes subsampled, while other have a lot of samples. We will add much more train values on the subsampled classes.

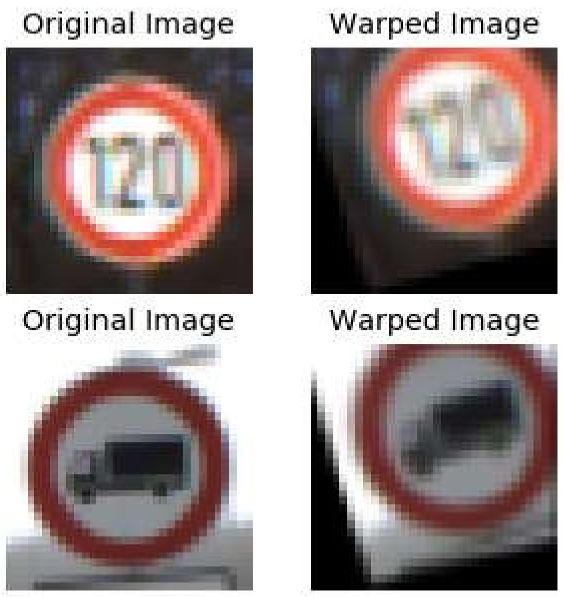
Design and Test a Model Architecture

We will use LeNET-5 Model.



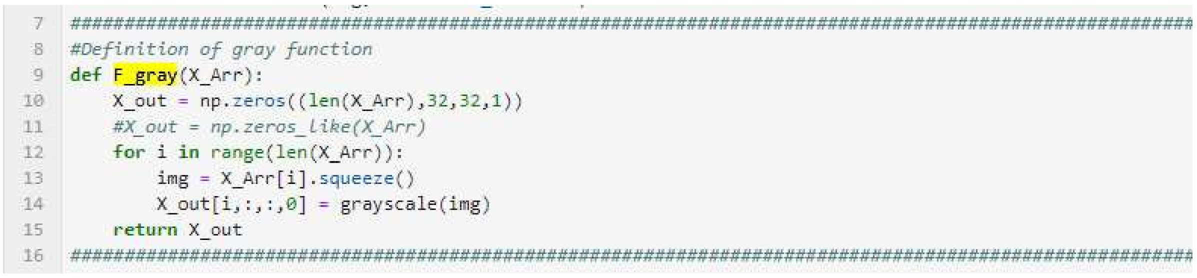
The first step is Wropping the images. Bacaouse of perspective I need to warped these images.

```
[8]: <matplotlib.image.AxesImage at 0x1e94610b1d0>
```

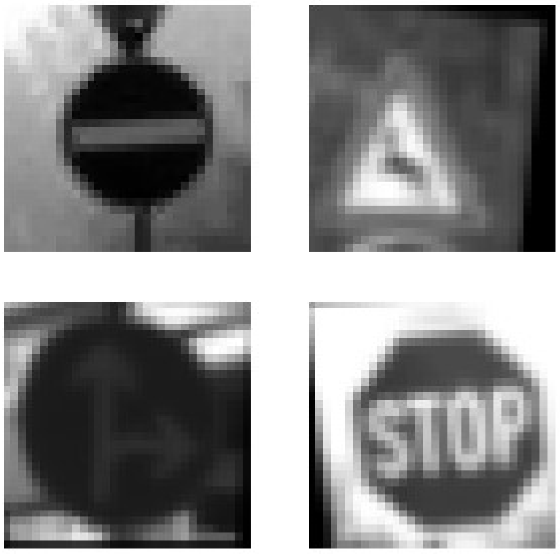


After this, we will prepare our images with grayscaling to remove one dept from image.It will be the second step.

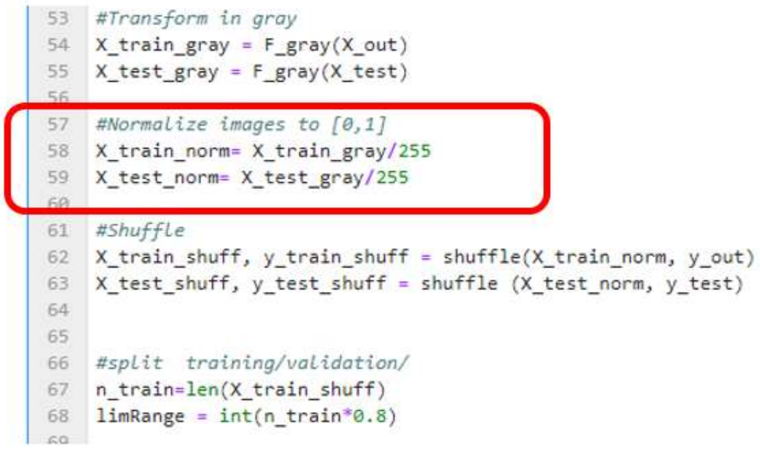
Here is an example of a traffic sign image before and after grayscaling.



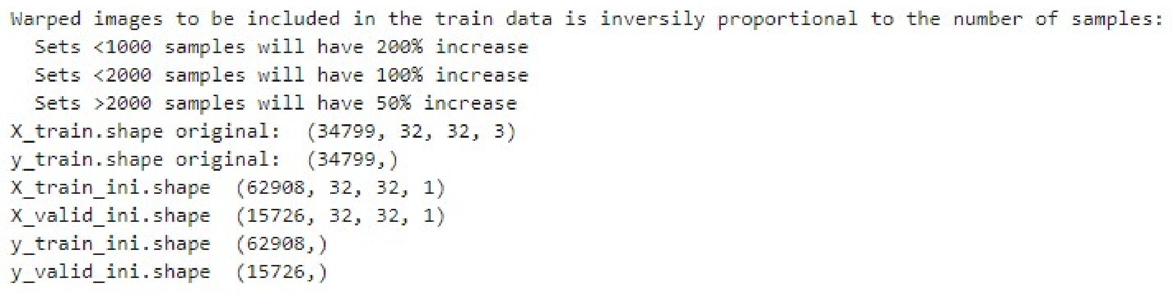
```
####################################################################################################
#Definition of gray function
def F_gray(X_Arr):
    X_out = np.zeros((len(X_Arr),32,32,1))
    #X_out = np.zeros_like(X_Arr)
    for i in range(len(X_Arr)):
        img = X_Arr[i].squeeze()
        X_out[i,:,:,0] = grayscale(img)
    return X_out
####################################################################################################
```



And also we will normalize the image,



```
#Transform in gray
X_train_gray = F_gray(X_out)
X_test_gray = F_gray(X_test)

#Normalize images to [0,1]
X_train_norm= X_train_gray/255
X_test_norm= X_test_gray/255

#Shuffle
X_train_shuff, y_train_shuff = shuffle(X_train_norm, y_out)
X_test_shuff, y_test_shuff = shuffle (X_test_norm, y_test)


#split  training/validation/
n_train=len(X_train_shuff)
limRange = int(n_train*0.8)
```



```
Warped images to be included in the train data is inversily proportional to the number of samples:
  Sets <1000 samples will have 200% increase
  Sets <2000 samples will have 100% increase
  Sets >2000 samples will have 50% increase
X_train.shape original:  (34799, 32, 32, 3)
y_train.shape original:  (34799,)
X_train_ini.shape  (62908, 32, 32, 1)
X_valid_ini.shape  (15726, 32, 32, 1)
y_train_ini.shape  (62908,)
y_valid_ini.shape  (15726,)
```

As a last step, I normalized the image data because, Normalization can help improve robustness and convergence of the learning process as it ensures that the feature dimensions, in this case pixel intensities, are on the same scale.

The difference between the original data set and the augmented data set is the following ...

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:

The final architecture is based on LeNet lab exercise. Feed forward neural networks, five layers. Two convolutional layers and 3 fully connected layers

Layer 1: Convolutional. Input = 32x32x1. Output = 28x28x20.
    Activation: tanh
    Pooling: Input = 28x28x20. Output = 14x14x20.
Layer 2: Convolutional. Output = 10x10x48.
    Activation: tanh
    Pooling: Input = 10x10x48. Output = 5x5x48.
    Flatten. Input = 5x5x48. Output = 1200.
Layer 3: Fully Connected. Input = 1200. Output = 120.
    Activation: tanh
Layer 4: Fully Connected. Input = 120. Output = 84.
    Activation: tanh
Layer 5: Fully Connected. Input = 84. Output = 43.

Additional comments after project revision:

The baseline architecture used for this project was LeNet.

The first three layers are convolutional layers. It is a 2D filter because we're working with an image, two dimensions (and I used grayscale, to ignore the third dimension).

In my interpretation of how convolution works. It is like a filter sliding every position of the image. Because it is a linear filter, the resulting value will be greater when there is an exact match between the image and the filter weights, and value zero if not correlated at all. It is equivalent to try to find small features (or better, kernels) of the image. Try to find one piece of a puzzle per time: the eyes, nose, ears, hair, and so on.

The original LeCun paper even cites even that he used different sizes of the kernels. Using the analogy of the puzzle, one type of filter looked for greater pieces, and other, smaller pieces.

The first layer extract these direct features, while second and third layers are levels of abstraction over the first layer. First I try to identify each piece of the puzzle, then group each piece of the puzzle to form a greater piece of the puzzle, then group these group of puzzles.

And finally, there is a transformation of this image in a single code of information, by Flatten layer. The dense layers can work as a usual neural network from now on.

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 an epochs,optimizer and batch.

Epochs, optimizer, batch

Baseline

My baseline was exactly the same Lenet model in previous class laboratory. The only changes were on sizes of input and output.

There were some steps to evaluate the parameters, described below.

Step 1: Epochs

I did an evalution of some different epochs, resulting in:

Name	1	2	3	4	5
EPOCHS	5	10	15	20	25
Validation	0.910	0.956	0.969	0.974	0.969

There are improvement in validation till 20 - 25 Epochs.

Step 2: Optimizer

I tested three optimization functions: AdamOptimizer, Grad Descent, Momentum Each of the tests were made varying the step among 5 values, according to tables. The Epoch was 25.

AdamOptimizer:

Name	1	2	3	4	5
Step	0.005	0.01	0.025	0.04	0.05
Validation	0.954	0.978	0.978	0.981	0.979
Test	0.875	0.898	0.899	0.893	0.902

The Adagrad is a very good choice.

Grad Descent:

Name	1	2	3	4	5
Step	0.005	0.01	0.025	0.04	0.05
Validation	0.066	0.057	0.055	0.058	0.063
Test	0.072	0.065	0.059	0.062	0.063

The Grad Descent didn't even converged to a reasonable value.

Momentum:

Name	1	2	3	4	5
Step	0.005	0.01	0.025	0.04	0.05
Validation	0.695	0.887	0.890	0.849	0.861
Test	0.641	0.802	0.801	0.750	0.769

The momentum optimizer didn't performed well also. I kept the AdamOptimizer.

Step 3: Grayscale

I applied to grayscale to every training and test images.

There were only a small difference to baseline without grayscale. But, the processing time (since grayscales uses one layer instead of 3) was better. So, I kept the grayscale filter from now on.

Name	1	2	3	4	5
Step	0.005	0.01	0.025	0.04	0.05
Validation	0.925	0.973	0.982	0.979	0.978
Test	0.844	0.892	0.903	0.892	0.896

Step 4: Dropout layer

I tried to insert one dropout layer (60%) after each convolutional layer. It performed worse than without dropout. Thinking well, the pooling layer does the function of dropout layer, so it doesn't make sense at all to have both pooling and dropout.

Since the TanH performed better, I used it as activation function.

Step 6: Wider Network¶

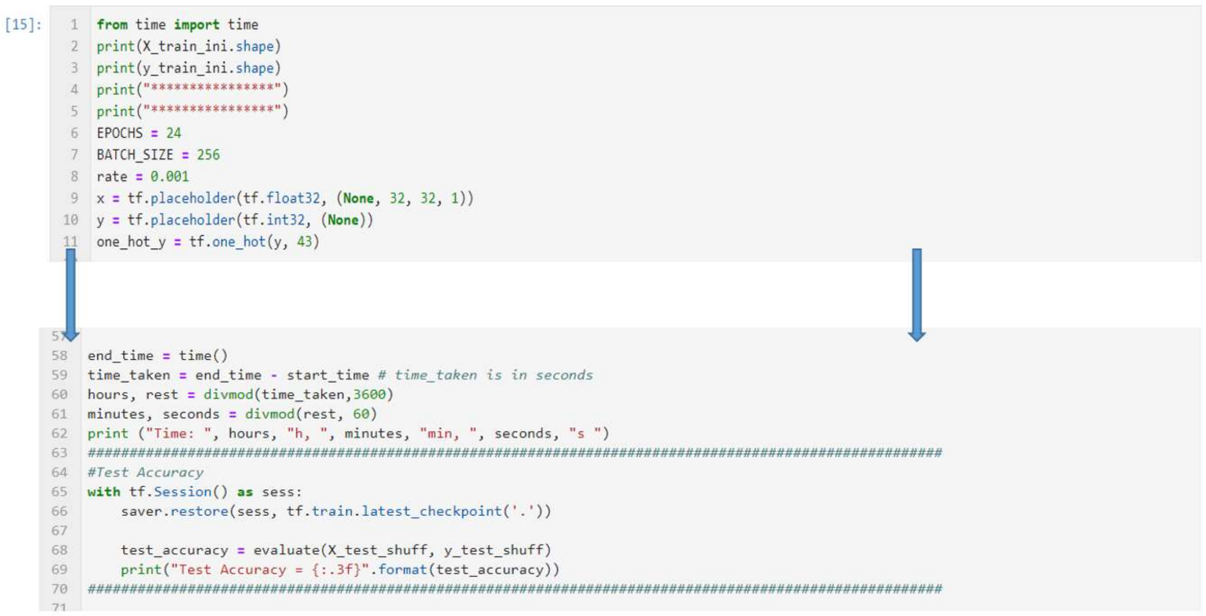
To try to reach bigger accuracy, i tested some wider Networks. Basically, I changed the number of neurons in the first two convolutional layers.

The value on the table refers to the number of neurons on corresponding layer.

Name	**1**	**2**	**3**	**4**
Layer 1	12	20	30	40
Layer 2	32	48	60	70
Validation	0.936	0.925	0.940	0.942
Test	0.915	0.919	0.936	0.934

The layer with 30 and 60 neurons (case 3 above) was the best one. And it's Test acuracy is of 93,4%.

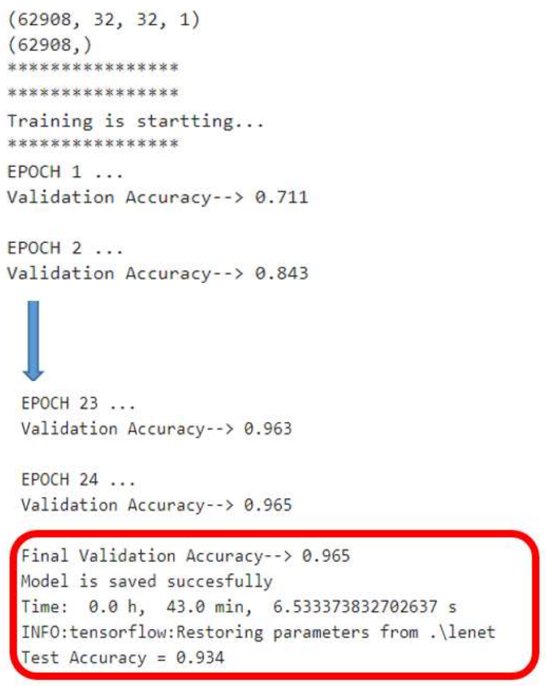
4. Describe the approach taken for finding a solution and getting the validation set accuracy to be at least 0.93. Include in the discussion the results on the training, validation and test sets and where in the code these were calculated. Your approach may have been an iterative process, in which case, 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.



[15]:

```
from time import time
print(X_train_ini.shape)
print(y_train_ini.shape)
print("****************")
print("****************")
EPOCHS = 24
BATCH_SIZE = 256
rate = 0.001
x = tf.placeholder(tf.float32, (None, 32, 32, 1))
y = tf.placeholder(tf.int32, (None))
one_hot_y = tf.one_hot(y, 43)
```

```
end_time = time()
time_taken = end_time - start_time # time_taken is in seconds
hours, rest = divmod(time_taken,3600)
minutes, seconds = divmod(rest, 60)
print ("Time: ", hours, "h, ", minutes, "min, ", seconds, "s ")
##########################################################################################################
#Test Accuracy
with tf.Session() as sess:
    saver.restore(sess, tf.train.latest_checkpoint('.'))

    test_accuracy = evaluate(X_test_shuff, y_test_shuff)
    print("Test Accuracy = {:.3f}".format(test_accuracy))
##########################################################################################################
```



```
(62908, 32, 32, 1)
(62908,)
****************
****************
Training is startting...
****************
EPOCH 1 ...
Validation Accuracy--> 0.711

EPOCH 2 ...
Validation Accuracy--> 0.843
```

```
EPOCH 23 ...
Validation Accuracy--> 0.963

EPOCH 24 ...
Validation Accuracy--> 0.965

Final Validation Accuracy--> 0.965
Model is saved succesfully
Time:  0.0 h,  43.0 min,  6.533373832702637 s
INFO:tensorflow:Restoring parameters from .\lenet
Test Accuracy = 0.934
```

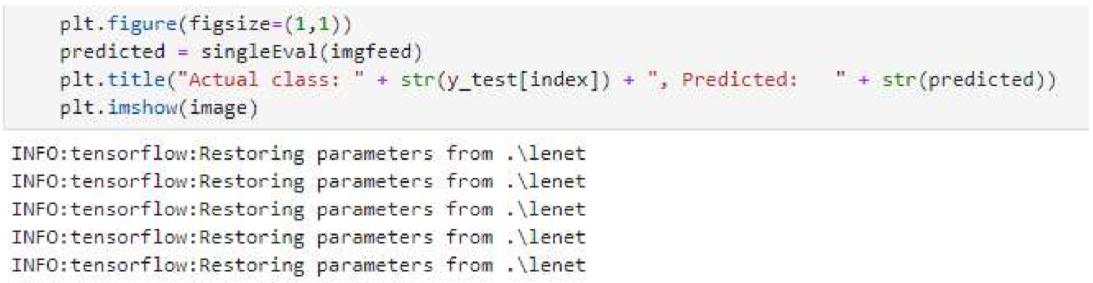
I started from LeNet. I tested and discarded dropout layer, used grayscale, and tanH as activation function. Epochs = 24 in the final evaluation. Final step = 0.001, batch = 256.
And a wider neural network with 30 and 60 neurons in convolutional layers 1 and 2.

The final model had validation accuracy of **96,5%** and test accuracy of **93,4%.**

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.**

From training set:

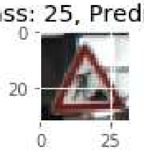


```
    plt.figure(figsize=(1,1))
    predicted = singleEval(imgfeed)
    plt.title("Actual class: " + str(y_test[index]) + ", Predicted:   " + str(predicted))
    plt.imshow(image)
```

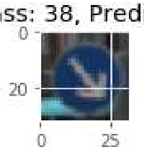
```
INFO:tensorflow:Restoring parameters from .\lenet
INFO:tensorflow:Restoring parameters from .\lenet
INFO:tensorflow:Restoring parameters from .\lenet
INFO:tensorflow:Restoring parameters from .\lenet
INFO:tensorflow:Restoring parameters from .\lenet
```

Actual class: 34, Predicted: [34]

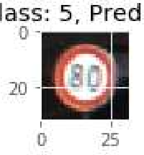
Actual class: 25, Predicted: [25]



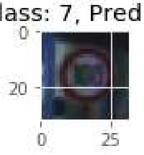
Actual class: 38, Predicted: [38]



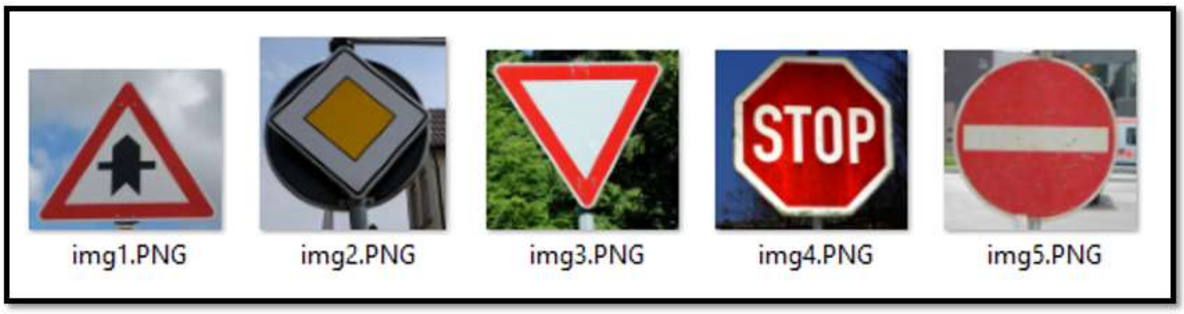
Actual class: 5, Predicted: [1]



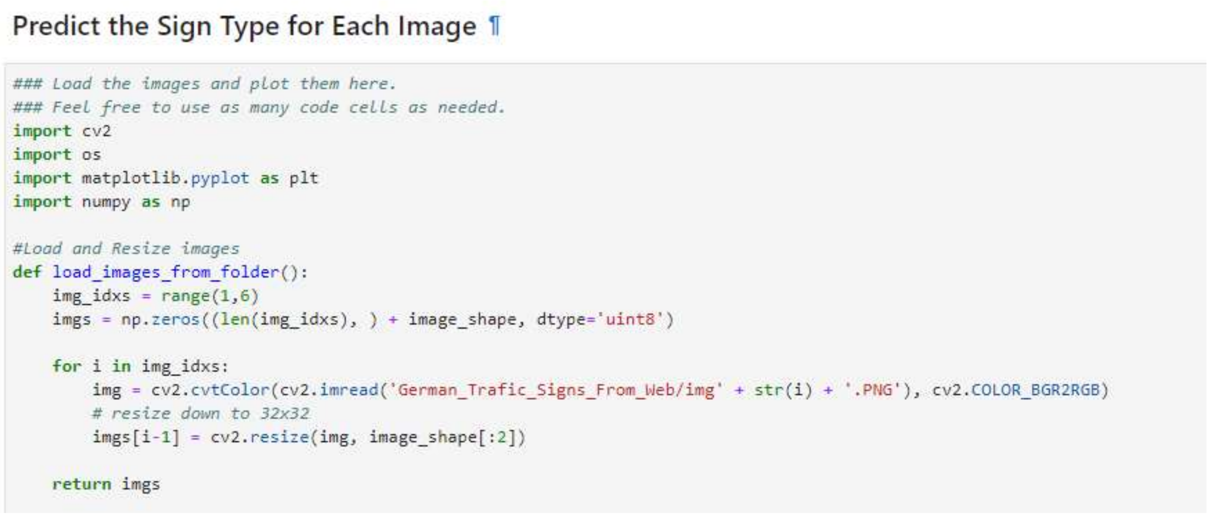
Actual class: 7, Predicted: [1]



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



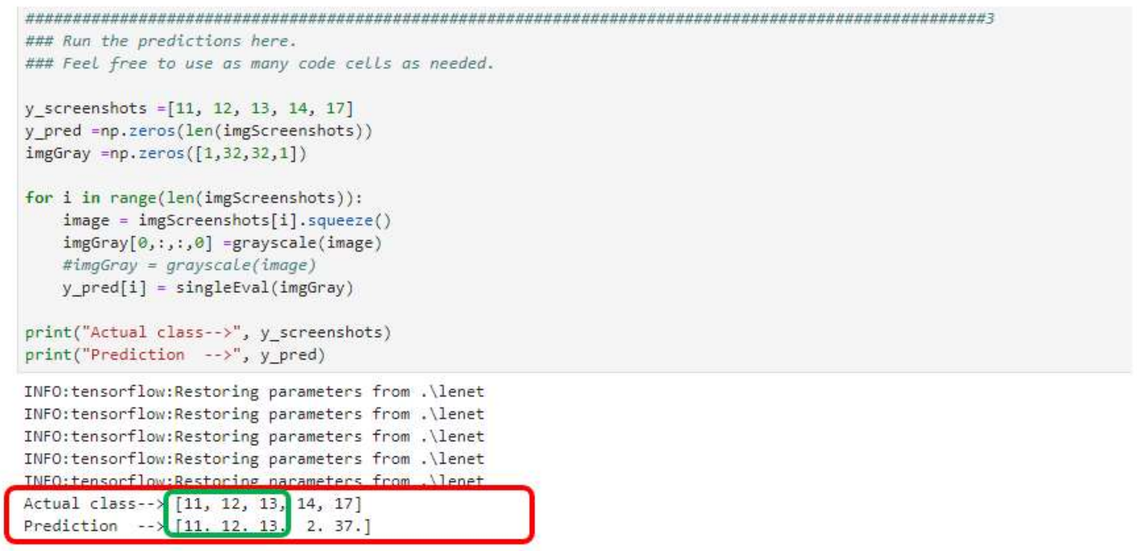
First af all , I resized their sizes to 32*32.



Predict the Sign Type for Each Image ¶

```
### Load the images and plot them here.
### Feel free to use as many code cells as needed.
import cv2
import os
import matplotlib.pyplot as plt
import numpy as np

#Load and Resize images
def load_images_from_folder():
    img_idxs = range(1,6)
    imgs = np.zeros((len(img_idxs), ) + image_shape, dtype='uint8')

    for i in img_idxs:
        img = cv2.cvtColor(cv2.imread('German_Trafic_Signs_From_Web/img' + str(i) + '.PNG'), cv2.COLOR_BGR2RGB)
        # resize down to 32x32
        imgs[i-1] = cv2.resize(img, image_shape[:2])

    return imgs
```

Result:



```
##################################################################################################3
### Run the predictions here.
### Feel free to use as many code cells as needed.

y_screenshots =[11, 12, 13, 14, 17]
y_pred =np.zeros(len(imgScreenshots))
imgGray =np.zeros([1,32,32,1])

for i in range(len(imgScreenshots)):
    image = imgScreenshots[i].squeeze()
    imgGray[0,:,:,0] =grayscale(image)
    #imgGray = grayscale(image)
    y_pred[i] = singleEval(imgGray)

print("Actual class-->", y_screenshots)
print("Prediction  -->", y_pred)
```

```
INFO:tensorflow:Restoring parameters from .\lenet
INFO:tensorflow:Restoring parameters from .\lenet
INFO:tensorflow:Restoring parameters from .\lenet
INFO:tensorflow:Restoring parameters from .\lenet
INFO:tensorflow:Restoring parameters from .\lenet
Actual class--> [11, 12, 13, 14, 17]
Prediction  --> [11. 12. 13.  2. 37.]
```

I have success with first 3 pictures, but I have fail with last 2 pictures.

-

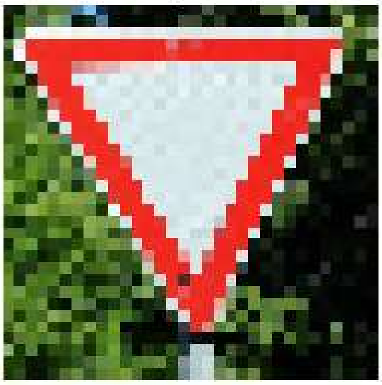
Img-->0

Img-->3

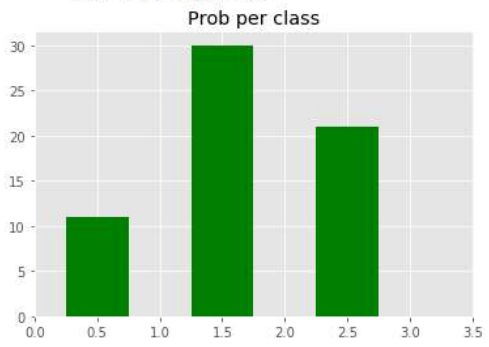
Img-->1

Img-->4

Img-->2



```
INFO:tensorflow:Restoring parameters from .\lenet
INFO:tensorflow:Restoring parameters from .\lenet
INFO:tensorflow:Restoring parameters from .\lenet
INFO:tensorflow:Restoring parameters from .\lenet
INFO:tensorflow:Restoring parameters from .\lenet
[array([[11, 30, 21]]), array([[12, 40, 38]]), array([[13, 25, 33]]), array([[ 2,  3, 14]]), array([[37, 34, 33]])]
Img   Top_k          Actual
0   [[11 30 21]]       11
1   [[12 40 38]]       12
2   [[13 25 33]]       13
3   [[ 2  3 14]]       14
4   [[37 34 33]]       17
*******Plot of probabilities******
```



The second two images needs a specific training on these signals.

As a result , it was not easy to success these difficult steps about deep learning wtih convolution. Some images has very good result after trainig, but the others not. I think I should make much more image processing before training dataset. Day by day with this UDACITY great course I will learn and implement better image processing techniques.

On the other hand I am very happy to learn deep learnig issues with this course.Thank you very much UDACITY team.