Traffic Sign Recognition Classifier - Writeup/README

This is the write up. All rubric points will be addressed in the cells below.

Dataset Summary & Exploration

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

I used Numpy and Python. np.unique came in handy for this. Code is located in cell 2 of the notebook.

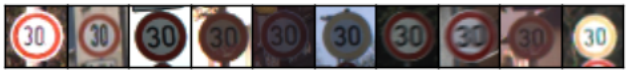
```
Number of training examples = 39209
Number of testing examples = 12630
Image data shape = (32, 32, 3)
Number of classes = 43
```

2) Include an exploratory visualization of the dataset

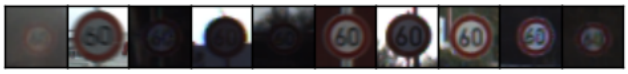
DATA VISUALIZATION The code for the data exploration is in code cells 3 and 4 of the notebook. I wanted to look at samples of all the data to get a feel for the data set. The last two plots show that there is a highly variable number of training examples for each class. Therefore we would expect that we might expect to do poorly on the classes that are under represented. So that means that data augmentation might improve accuracy. We will need to examine the distribution of errors to see for sure if the under represented classes are causing large error or not. The pictures seem to have very little rotation or angle changes in them as well. The data also helps to explain why grayscale worked better in the published paper by Lecun. The color seems poor in these images. So plan will be to use grayscale and then experiment with some types of data augmentation to improve results if needed. The last two plots can be compared to make sure that the distribution of classes is the same for training and test data, which it is.

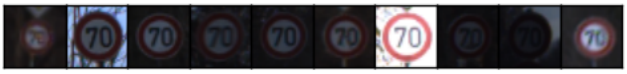
In [3]:
```
### Data exploration visualization code goes here.
```

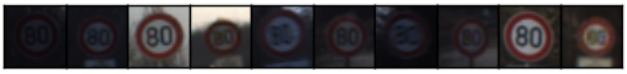

Class 0: Speed limit (20km/h) samples= 210


Class 1: Speed limit (30km/h) samples= 2220


Class 2: Speed limit (50km/h) samples= 2250

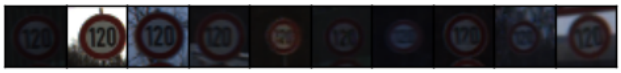

Class 3: Speed limit (60km/h) samples= 1410


Class 4: Speed limit (70km/h) samples= 1980

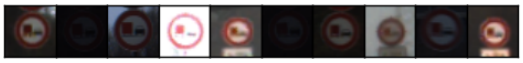

Class 5: Speed limit (80km/h) samples= 1860


Class 6: End of speed limit (80km/h) samples= 420


Class 7: Speed limit (100km/h) samples= 1440


Class 8: Speed limit (120km/h) samples= 1410


Class 9: No passing samples= 1470


Class 10: No passing for vehicles over 3.5 metric tons samples= 2010

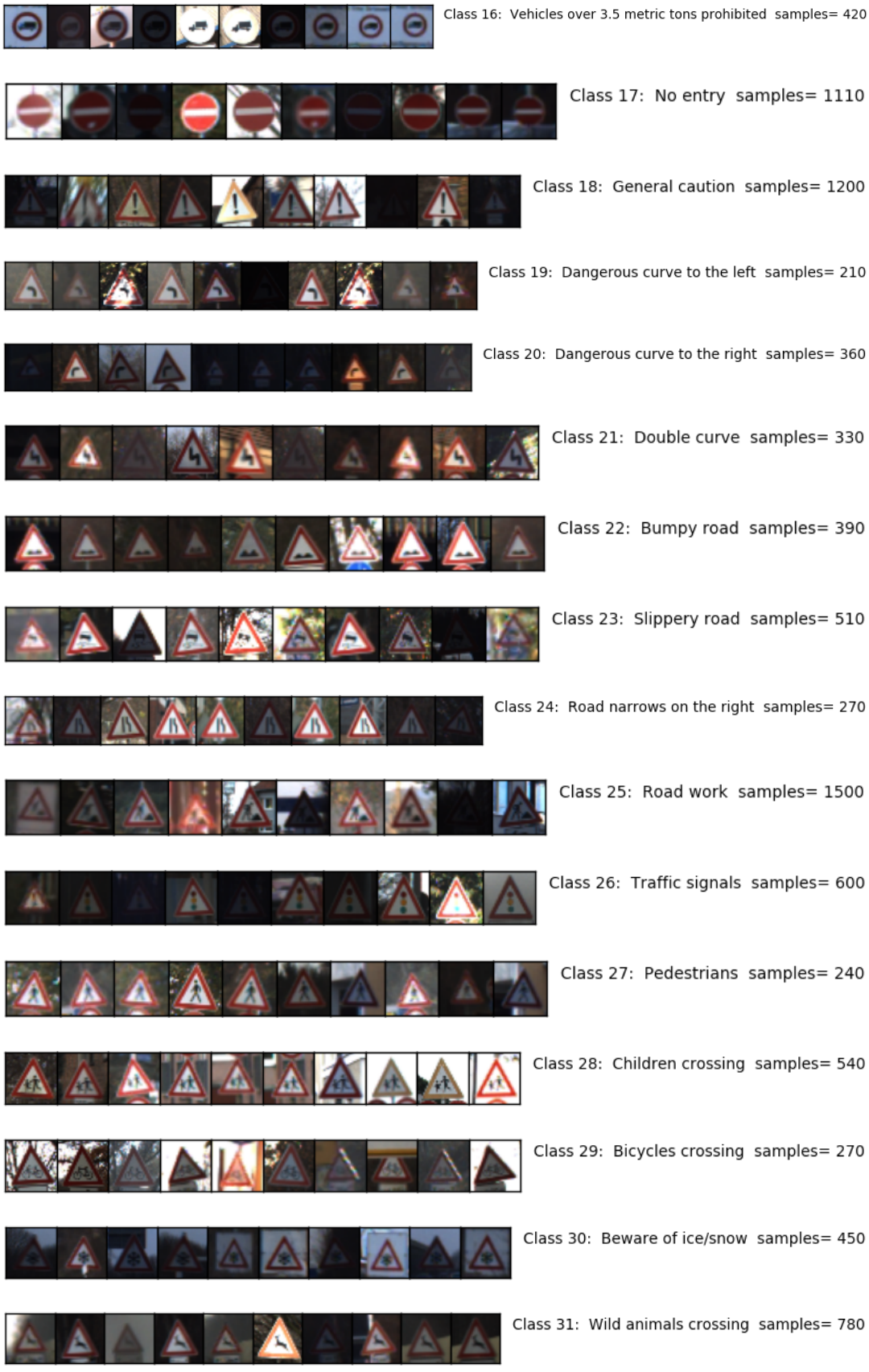

Class 11: Right-of-way at the next intersection samples= 1320


Class 12: Priority road samples= 2100


Class 13: Yield samples= 2160


Class 14: Stop samples= 780


Class 15: No vehicles samples= 630


Class 16: Vehicles over 3.5 metric tons prohibited samples= 420
Class 17: No entry samples= 1110
Class 18: General caution samples= 1200
Class 19: Dangerous curve to the left samples= 210
Class 20: Dangerous curve to the right samples= 360
Class 21: Double curve samples= 330
Class 22: Bumpy road samples= 390
Class 23: Slippery road samples= 510
Class 24: Road narrows on the right samples= 270
Class 25: Road work samples= 1500
Class 26: Traffic signals samples= 600
Class 27: Pedestrians samples= 240
Class 28: Children crossing samples= 540
Class 29: Bicycles crossing samples= 270
Class 30: Beware of ice/snow samples= 450
Class 31: Wild animals crossing samples= 780


Class 32: End of all speed and passing limits samples= 240


Class 33: Turn right ahead samples= 689


Class 34: Turn left ahead samples= 420


Class 35: Ahead only samples= 1200


Class 36: Go straight or right samples= 390


Class 37: Go straight or left samples= 210

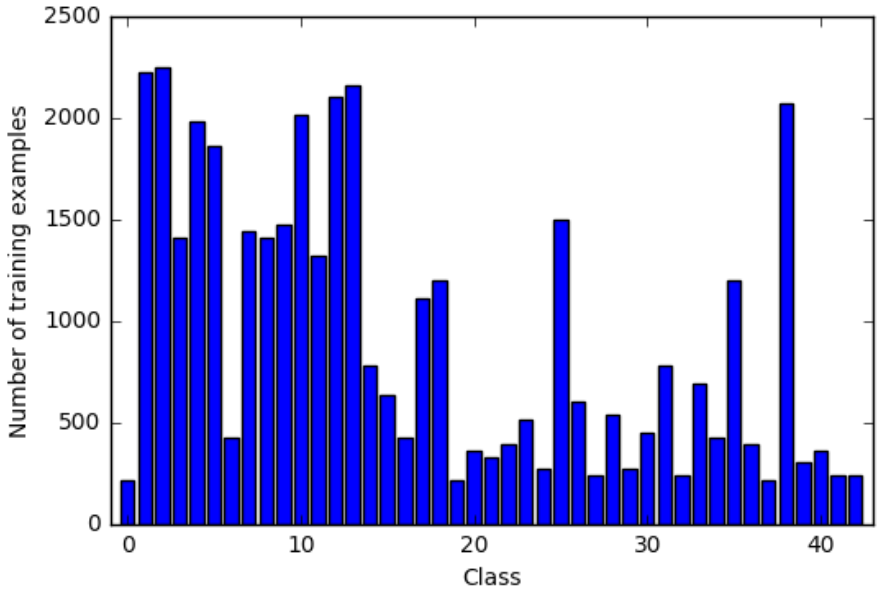

Class 38: Keep right samples= 2070

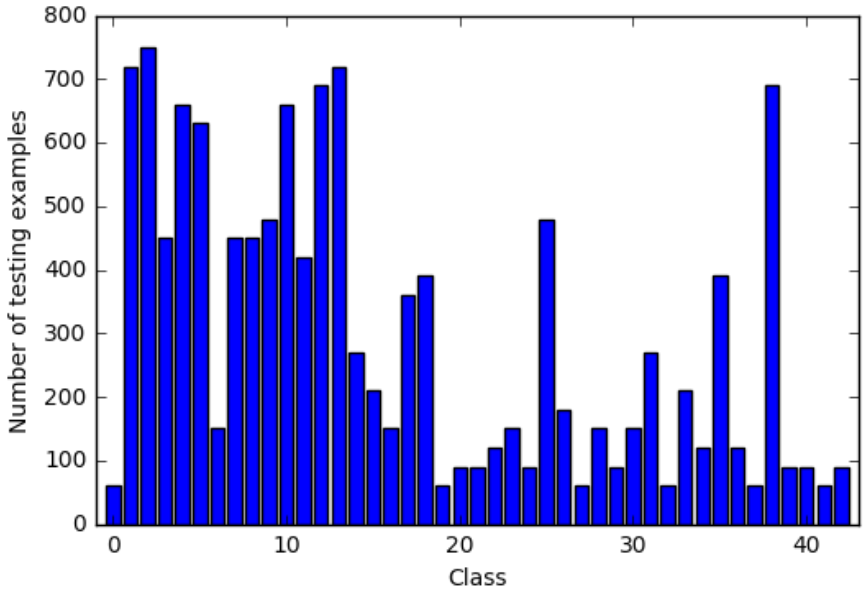

Class 39: Keep left samples= 300


Class 40: Roundabout mandatory samples= 360


Class 41: End of no passing samples= 240

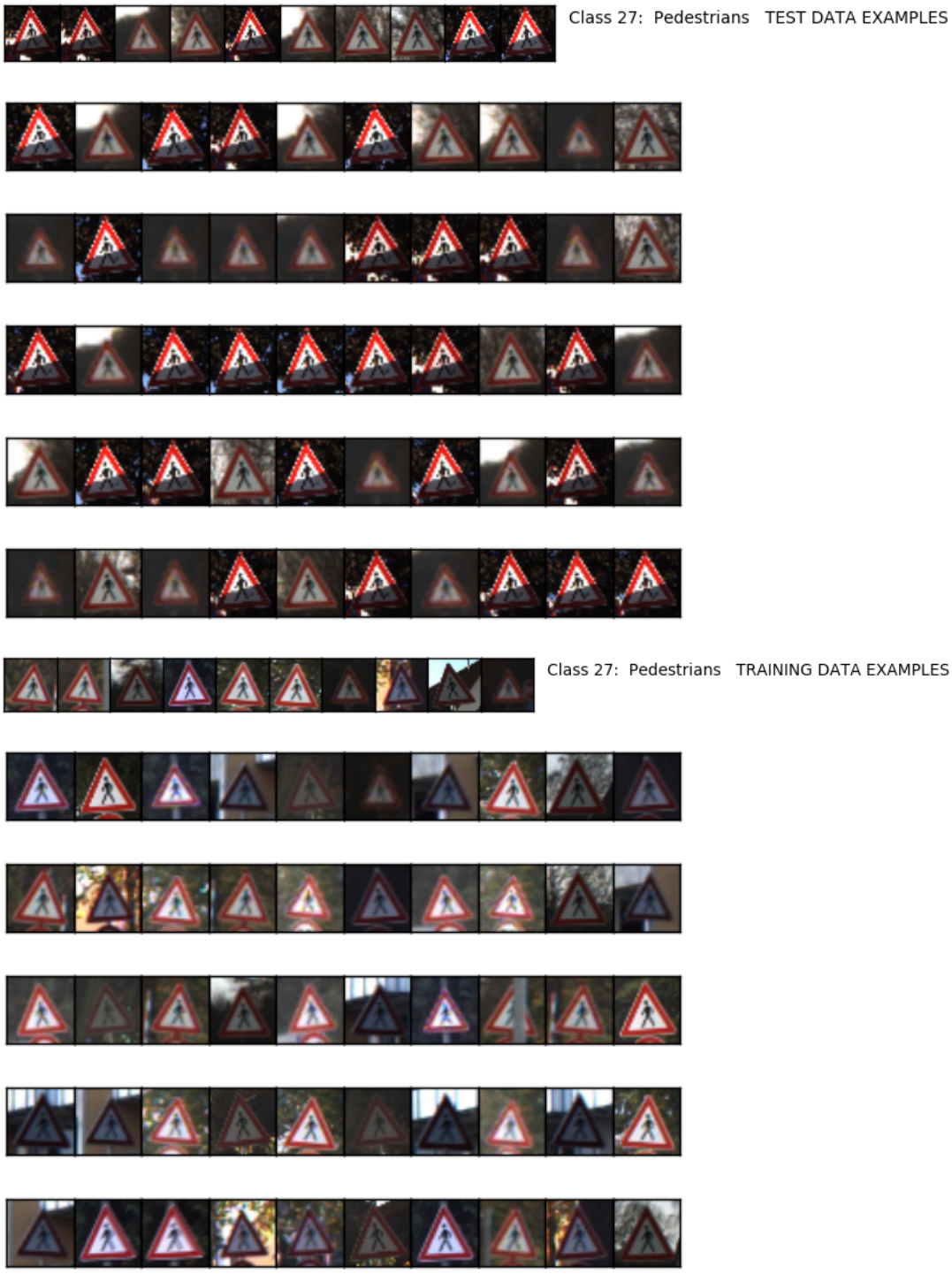

Class 42: End of no passing by vehicles over 3.5 metric tons samples= 240





DATA EXPLORATION PART 2 Decided to dig deeper into a specific class (27-Pedestrians) since my Test results were poor in this class. I also tried to improve results by data augmenting just this class (see next section). The data augmentation I performed was based on rotating training images to produce more data. This did not help. I believe the results below explain why. The test data set is extremely biased. It looks like it was generated from only a handful of original images and then used data augmentation to generate variations of the same images. This seems like very poor practice for a competition data set? Almost all of the test images are skewed in the same direction. So if we augment with enough variations, we likely could improve results. There is no substitute for visual inspection of data. This makes me very leary of future data sets. Also implies that the only way to improve results on this data set is by lots of data augmentation. I am out of time now, so that will have to wait for another project. The the code is located in the 4th code cell in the notebook.

In [4]:
```
### More data exploration
```



Design and Test a Model Architecture

1) Pre-process the Data Set (normalization, grayscale, augmentation)

Use the code cell (or multiple code cells, if necessary) to implement the first step of your project.

DATA AUGMENTATION Experimented here with some limited data augmentation. Running low on time so am going to just try to augment a single class that has low representation to see if it impacts results or not. Any data augmentation of this data set requires some care because some signs if flipped or rotated mean something different. The recommended way to do augmentation is to have it always on so you are always training on a 'new' set of data. I am simply going to try to increase the Pedestrian class case since it had poor test results and also was a fail case in my own image set. This simple data augmentation did not improve results. See previous section for discussion of why. I used skimage.transform.rotate to generate rotated images to add to the training set. I randomly chose images from the training set to rotate so that a different image was rotated each time. I used a uniform distribution between -15 and +15 degrees. I added 800 images of just this class. One needs to be careful when using skimage as the functions often change the dtype of the input in the output. I show examples of ten original images followed by the same ten images rotated in the plots below. The code for this is in code cells 5 and 6 (labeled In[9]) of the notebook.

```
In [9]: ### Visualize the rotations
```

LeCun paper showed grayscale worked best so going to use that info as starting point. Here is a stack exchange link on converting RGB to grayscale: http://stackoverflow.com/questions/687261/converting-rgb-to-grayscale-intensity (http://stackoverflow.com/questions/687261/converting-rgb-to-grayscale-intensity) that gives coefficients commonly used and some supporting wiki document at https://en.wikipedia.org/wiki/Luma_(video (https://en.wikipedia.org/wiki/Luma_(video)). Common practice is to use Y′ = 0.299 R′ + 0.587 G′ + 0.114 B′ where the prime denotes previous scaling of pixel values to be between 0 and 1. (Gamma compression). The code for this is in code cell 7 (labeled In[10])

Visualize grayscale Below plots show examples of the grayscale conversion. I wrote my own code for this in code cell 7. See previous cell for details on background. I chose grayscale because the Lecun paper showed that it worked the best. When looking at data, it makes sense as many of the images are underexposed. Also many of the signs that mean diffent things have the same color. It makes sense that the classifier would use shapes and edges more than color with this data set and problem. I saw a jump in validation accuracy of about 6-8% when I applied grayscale.

```
In [13]: ### Visualize grayscale here
```

Experimented with rescaling intensity of images. This actually made the validation accuracy worse, so I dropped it. I did not have time to experiment further. I think that the grayscale had already done the heavy lifting and this step likely over exposed some images washing out details. The code for this cell is commented out but included here for reference.

In [ ]:
```
### Experiment with rescaling intensity of images, thought was that this might
 improve
### over grayscale, but it actually made the validation accuracy worse. There
might be settings
### here that could improve it, but out of time to continue on this. Comment o
ut for now.
'''
import numpy as np
from skimage import exposure

for i in range(len(X_train)):
    image = X_train[i]
    p1, p9 = np.percentile(image, (1, 90))
    image = exposure.rescale_intensity(image.squeeze(), in_range=(p1, p9))
    X_train[i]=np.reshape(image,(32,32,1))
for i in range(len(X_test)):
    image = X_test[i]
    p1, p9 = np.percentile(image, (1, 90))
    image = exposure.rescale_intensity(image.squeeze(), in_range=(p1, p9))
    X_test[i]=np.reshape(image,(32,32,1))
'''
```

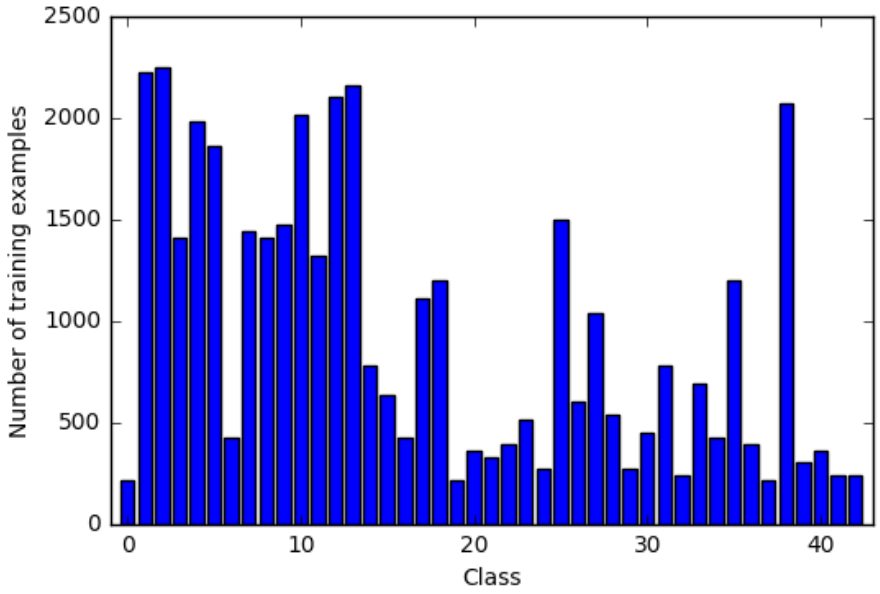
2) Split Data into Training, Validation and Testing Sets

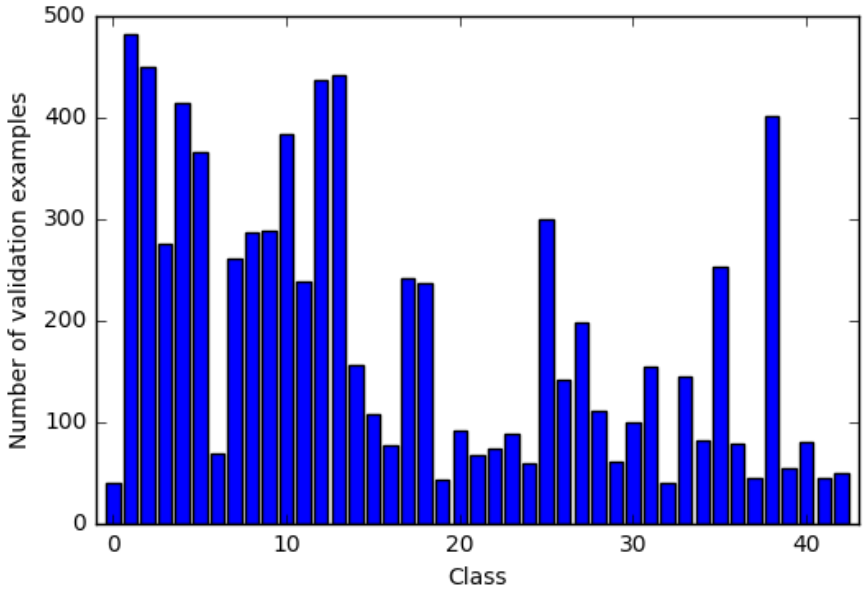
Added plots to make sure the distribution of classes after splitting was similar to starting distribution. This sanity check showed that the split keeps the original distributions as it should. I used sklearn train_test_split function to split the test data into training and validation sets. I used 20% for this split. The code for this is in code cell 12 (labeled In[15]). I described the data augmentation in previous cell. A of the split is shown below.

```
summary
X_train shape =  (32007, 32, 32, 1) y_train shape =  (32007,)
X_test shape =  (12630, 32, 32, 1) y_test shape =  (12630,)
X_validation shape =  (8002, 32, 32, 1) y_validation shape =  (8002,)
```

In [15]:
```
### Split the data into training/validation/testing sets here.
```

```
X_train shape =  (32007, 32, 32, 1) y_train shape =  (32007,)
X_test shape =  (12630, 32, 32, 1) y_test shape =  (12630,)
X_validation shape =  (8002, 32, 32, 1) y_validation shape =  (8002,)
```





3) Model Architecture

The code for my model is in the 13th code cell (labeled In[15]). I used a modified version of the LeNet architecture since it performed quite well from the start. I decided that the convolutional layers should be deeper so implemented that. I experimented with a third convolutional layer but it did not improve results so I dropped it. I also experimented some with drop out but again did not see much improvement so left it out. I believe that both of these experiments would have different results if I had done a lot more data augmentation. The layers are shown in the table below.

Layers	**Input Size**	**Output size**
Conv 1 , 5x5, S=1	32x32x1	28x28x32
Relu	28x28x32	28x28x32
Max Pool, 2x2, S=2	28x28x32	14x14x32
Conv 2, 5x5, S=1	14x14x32	10x10x64
Relu	10x10x64	10x10x64
Max Pool, 2x2, S=2	10x10x64	5x5x64
Flatten	5x5x64	1600
Fully Connected	1600	1024
Relu	1024	1024
Fully Connected	1024	43

4) Train, Validate and Test the Model

The training code cells for training,validation and testing are in cells 15,16,and 17 respectively (In[18],In[19],In[20]). To train the model I used softmax with cross entropy as a loss function. I used the Adam optimizer with defaults and learning rate set to 0.001. I used 20 Epochs and Batch size of 128.

Model Evaluation

Evaluate how well the loss and accuracy of the model for a given dataset.

Train the Model

5) Aproach Taken for finding a Solution

I started with the LeCun model from the last section. It got about 84% validation accuracy which is not bad. I first experimented with just preprocessing using grayscale and this jumped the performance up to close to 90% by itself. I did some searching for solutions on the internet and saw that most solutions getting high test accuracy used more and deeper layers. I first simply made the original Lecun model deeper in the first two convolutional layers and also removed a fully connected layer so resulting in two fully connected layers at the end. I experimented with drop out some but did not see immediate benefit so left it out. I did experiment with adding a third convolutional layer and removing pooling in the first layer. This also did not perform better than the simpler model so I kept the 2 conv model.

I also experimented with looking at training accuracy vs validation accuracy. I noted that my validation accuracy seemed to be tracking my training accuracy closely. I ran for longer training to see if the validation accuracy would get worse, but it did not. This made me think that my model might be underfit or possibly just right. I tested under fit by adding convolutional layer and removing some pooling. This did not improve or change the behaviour of validation accuracy vs training accuracy. I also tried running a very small training set of 500 samples to see if I could overfit and then see validation accuracy go down. This experiment had poor validation accuracy but I still did not see validation accuracy decrease over time. At this point, I decided that my results were being mostly influenced by the data set itself and not really the model.

I experimented with data augmentation as I suspected that I could not get better results without it. The published works all do this. I wanted to see if I could just increase data for one class that was under represented in the data and improve results on that class. I understand that at this point, I am using test results to try to improve the model which is a no no, but I did this as an experiment to understand what was happening. The results were very interesting as I believe I found a problem with the original data set. The class 27 (pedestrians) in the test set is not diverse at all. See previous cell for deeper discussion and plots. I am confident that if I ran a lot of data augmentation including rotations, blur, view angle, cropping, etc. that I could improve results further. However, I stopped short of doing that extra work for time considerations.

Training accuracy was 100%, validation accuracy 99.5%, Test accuracy 95.4%.

Evaluate Test results The code in cell 19 (In[22]) breaks down the test results by class. This was useful to verify if the under represented classes did poorly or not. It was interesting that the under represented classes sometimes did worse relative to higher represented classes but it was not always the case. Again, I believe these results are highly dependent on the test data and therefore hard to interpret without doing a lot more data augmentation.

In [22]:
```
## test cell to evaluate test results
```

```
count of class:  0  =  [52, 60]
count of class:  1  =  [713, 720]
count of class:  2  =  [736, 750]
count of class:  3  =  [438, 450]
count of class:  4  =  [628, 660]
count of class:  5  =  [603, 630]
count of class:  6  =  [123, 150]
count of class:  7  =  [425, 450]
count of class:  8  =  [444, 450]
count of class:  9  =  [473, 480]
count of class:  10  =  [645, 660]
count of class:  11  =  [408, 420]
count of class:  12  =  [676, 690]
count of class:  13  =  [714, 720]
count of class:  14  =  [254, 270]
count of class:  15  =  [209, 210]
count of class:  16  =  [149, 150]
count of class:  17  =  [348, 360]
count of class:  18  =  [322, 390]
count of class:  19  =  [60, 60]
count of class:  20  =  [89, 90]
count of class:  21  =  [71, 90]
count of class:  22  =  [110, 120]
count of class:  23  =  [138, 150]
count of class:  24  =  [77, 90]
count of class:  25  =  [453, 480]
count of class:  26  =  [150, 180]
count of class:  27  =  [30, 60]
count of class:  28  =  [144, 150]
count of class:  29  =  [90, 90]
count of class:  30  =  [117, 150]
count of class:  31  =  [267, 270]
count of class:  32  =  [57, 60]
count of class:  33  =  [207, 210]
count of class:  34  =  [119, 120]
count of class:  35  =  [377, 390]
count of class:  36  =  [108, 120]
count of class:  37  =  [55, 60]
count of class:  38  =  [672, 690]
count of class:  39  =  [85, 90]
count of class:  40  =  [73, 90]
count of class:  41  =  [46, 60]
count of class:  42  =  [88, 90]
0.9535233570863024
```

Test a Model on New Images

Below are 9 images I obtained from the web. All but the 60km sign came from Google street view captures in Germany. The 60km was a stock photo found in general searching on the web. I used some built in ubuntu image editing to crop the images similar to the original data. I saved the files with names that contained the class label. I then ran the code in cell 20 (In[23]) using skimage and scipy to get each image, resize it to 32,32,3, store images in an array, and create a label array.

Load and Output the Images

In [23]:
```
### Load the images and plot them here.
```

Predict the Sign Type for Each Image

The code in cell 21 (In[24]) does all the same pre-processing and uses the same trained weights as used on the original data. The results are shown along with plots of each image below. The accuracy was pretty good with only missing 1/9 images. Given the test accuracy results on the original data set of 95%, these results seem reasonable. Also the one sign it missed was also a difficult class for the original test data (Pedestrians - 27). These signs were all pretty similar in quality to the training set other than the one stock photo (60km). So these results compare favorable with the test results.

In [24]:
```
### Run the predictions here and use the model to output the prediction for ea
ch image.
```

WRONG ! --> Class 11: Right-of-way at the next intersection CORRECT ==> Class 27: Pedestrians

CORRECT! --> Class 3: Speed limit (60km/h)

CORRECT! --> Class 34: Turn left ahead

CORRECT! --> Class 12: Priority road

CORRECT! --> Class 36: Go straight or right

CORRECT! --> Class 25: Road work

CORRECT! --> Class 36: Go straight or right

CORRECT! --> Class 13: Yield

CORRECT! --> Class 17: No entry

Analyze Performance

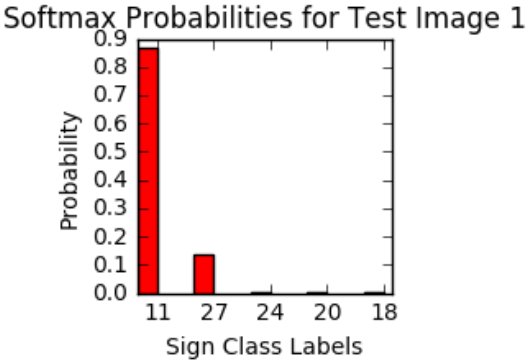
In [25]:
```
### Calculate the accuracy for these 5 new images.
```

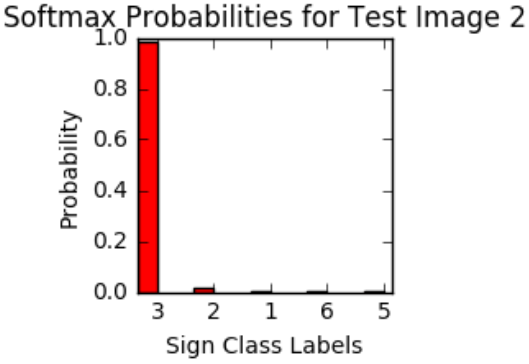
```
Accuracy for 9 images =  0.888888888889
```

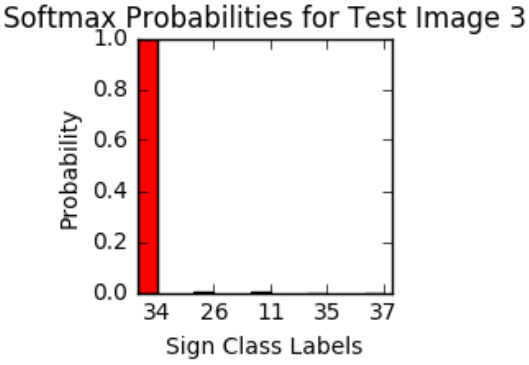
Output Top 5 Softmax Probabilities For Each Image Found on the Web

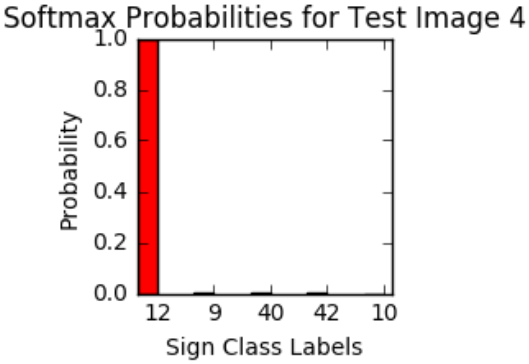
The results shown below in bar graphs show that all but the first image (Pedestrians), had very high confidence with the chosen class being very close to 100% and all other classes extremely low. The one that it got wrong, however, had approximately 87% for the wrong class and only 13% for the correct class. The class that it should have been (11) looks very similar to the pedestrian case as the border of the sign is the same and there is a black figure in the center. So it is not surprising that the net might make this mistake without enough training data. The code for this cell is in code cell 23 (In[26]).

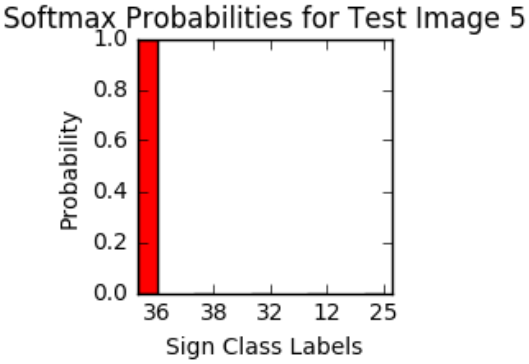
In [26]:
```
### Print out the top five softmax probabilities for the predictions on the Ge
rman traffic sign images found on the web.
```

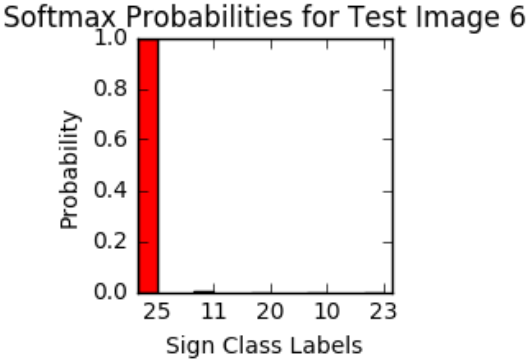

Softmax Probabilities for Test Image 1
Probability
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
11
27
24
20
18
Sign Class Labels

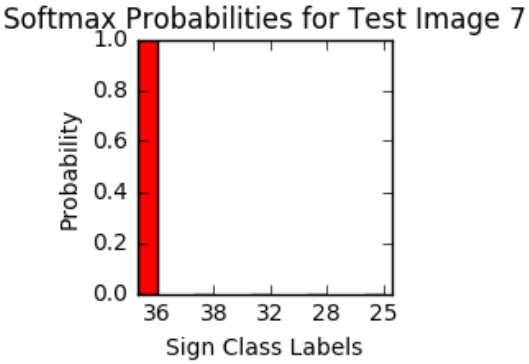

Softmax Probabilities for Test Image 2
Probability
0.0
0.2
0.4
0.6
0.8
1.0
3
2
1
6
5
Sign Class Labels

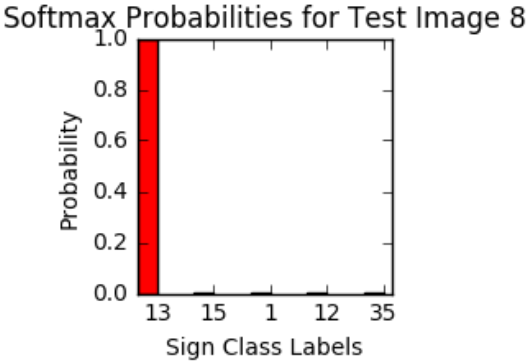

Softmax Probabilities for Test Image 3
Probability
0.0
0.2
0.4
0.6
0.8
1.0
34
26
11
35
37
Sign Class Labels

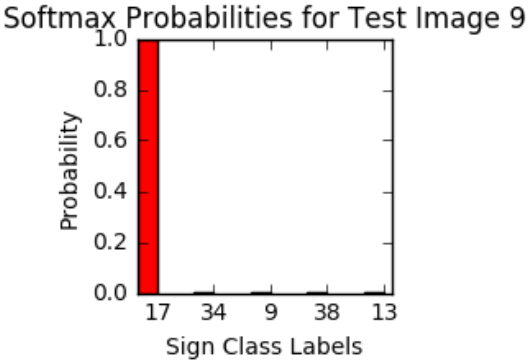

Softmax Probabilities for Test Image 4
Probability
0.0
0.2
0.4
0.6
0.8
1.0
12
9
40
42
10
Sign Class Labels


Softmax Probabilities for Test Image 5
Probability
1.0
0.8
0.6
0.4
0.2
0.0
36 38 32 12 25
Sign Class Labels


Softmax Probabilities for Test Image 6
Probability
1.0
0.8
0.6
0.4
0.2
0.0
25 11 20 10 23
Sign Class Labels


Softmax Probabilities for Test Image 7
Probability
1.0
0.8
0.6
0.4
0.2
0.0
36 38 32 28 25
Sign Class Labels


Softmax Probabilities for Test Image 8
Probability
1.0
0.8
0.6
0.4
0.2
0.0
13 15 1 12 35
Sign Class Labels


Softmax Probabilities for Test Image 9
Probability
1.0
0.8
0.6
0.4
0.2
0.0
17
34
9
38
13
Sign Class Labels