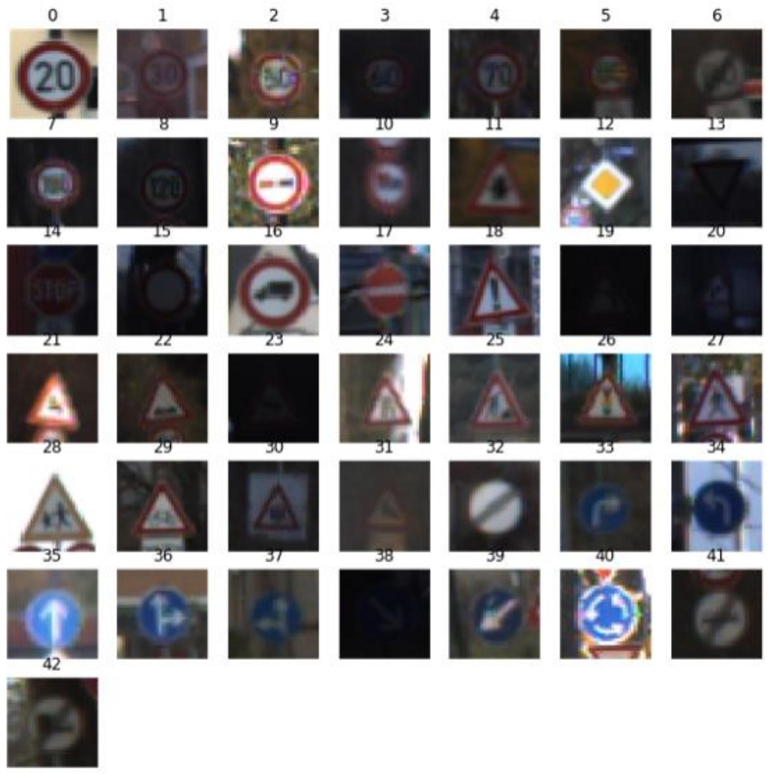
Project: Traffic Sign Recognition Classifier

Steps to complete the project:

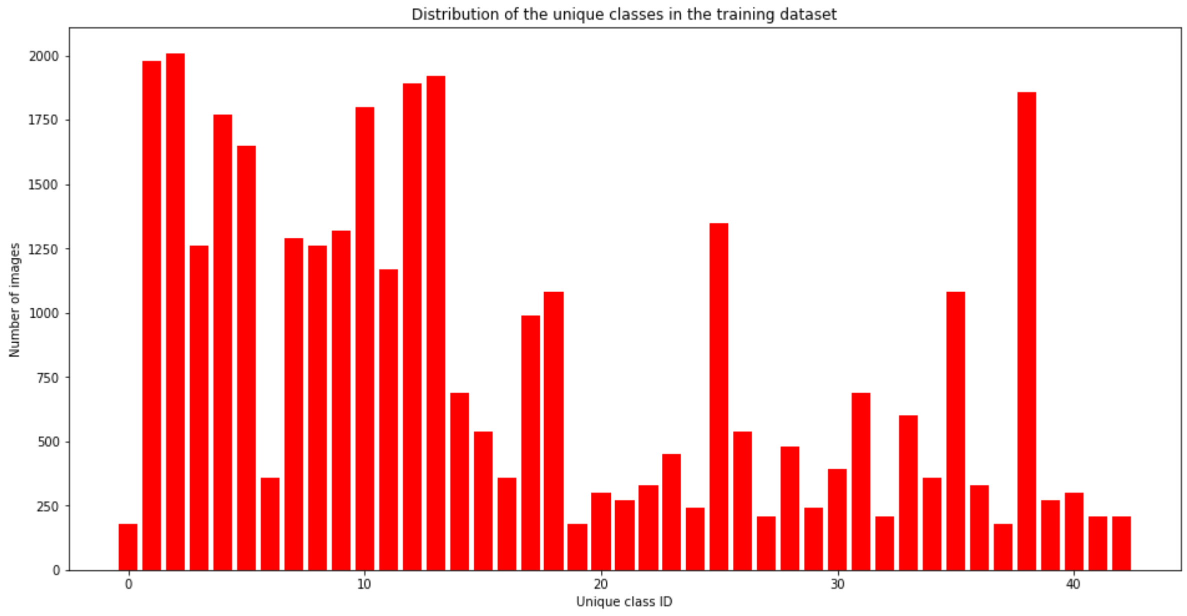
Setup the environment

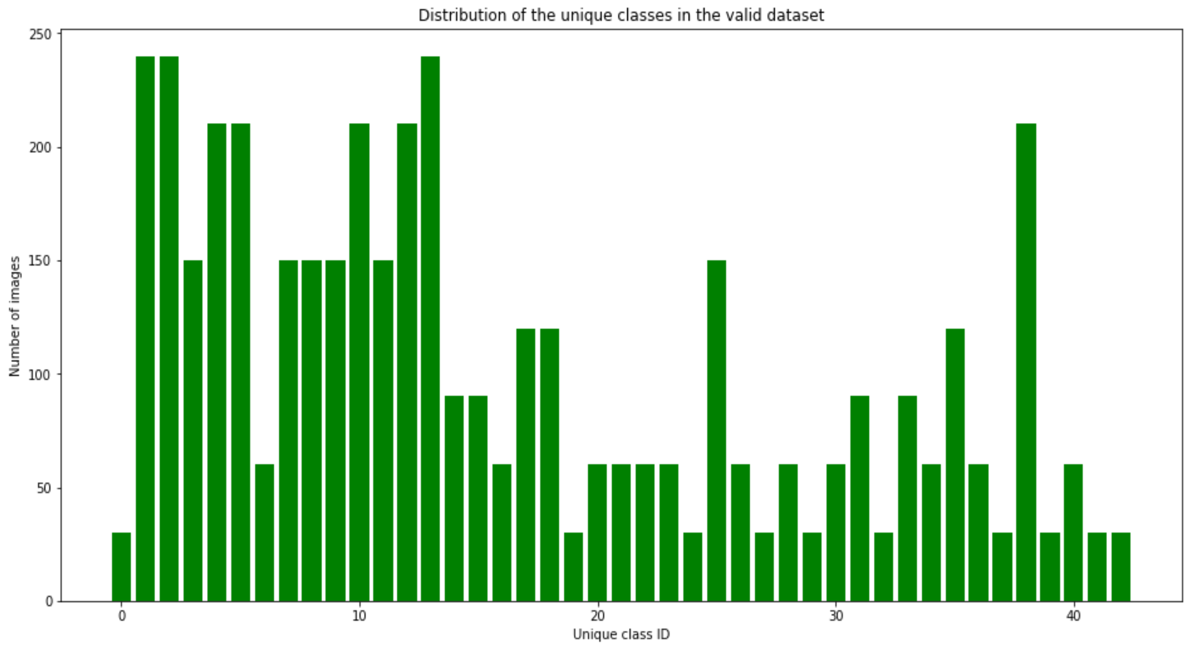
a. Clone the project; download the dataset
b. Activate carnd-term1 environment

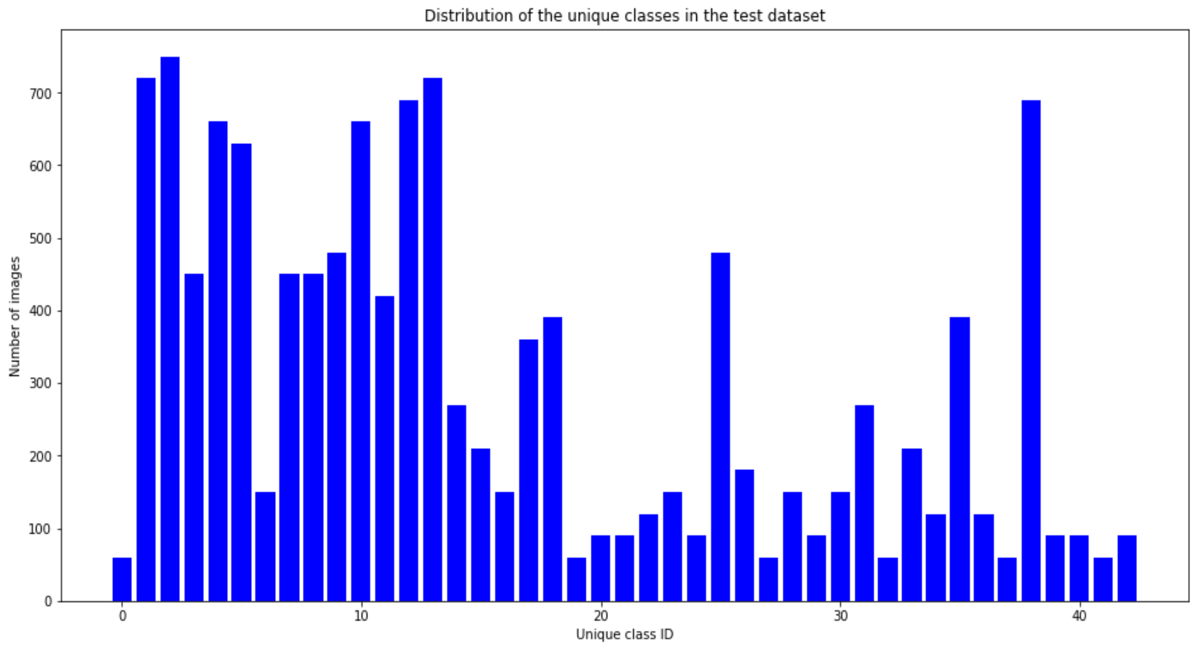
1. **Step 0: Load The Data**
   a. Loaded the data – training, validation and test sets.
2. **Step 1: Dataset Summary and Exploration**
   a. In the next notebook cell, I provided some basic summary of the of the dataset. The number of unique classes in the training, validation and test sets must be equal, and indeed they are, 43 unique classes to classify.
      i. Image Shape: (32, 32, 3)
      ii. Number of training examples = 34799
      iii. Number of testing examples = 12630
      iv. Number of train classes = Number of valid classes = Number of test class es = 43
   b. As for data visualization I plotted the 1st sample of each unique categories of the training dataset



c. In order to see how the number of samples in each category relate to each other I plotted the distribution of them.



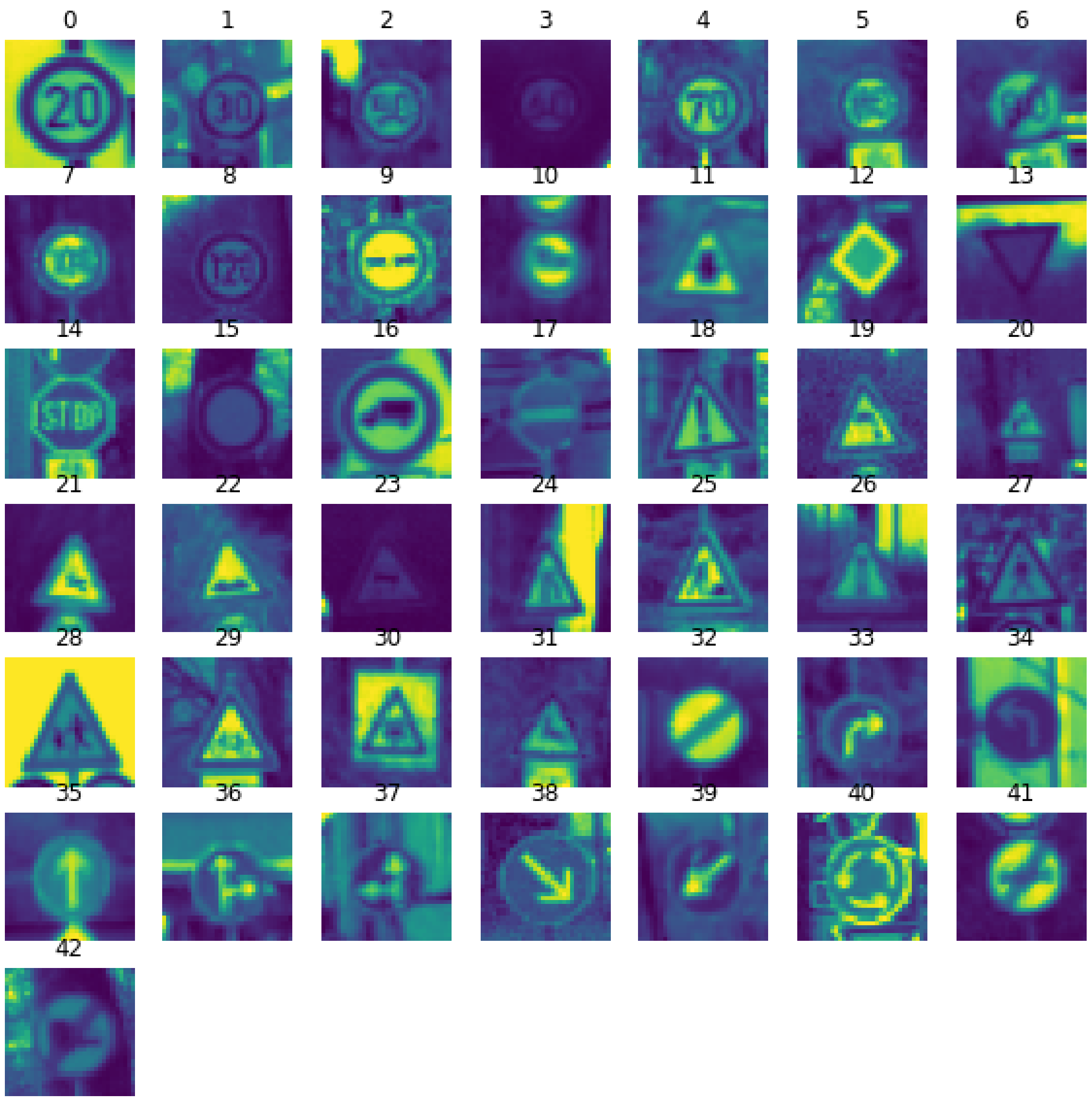




i. A few observations we can make of the plotted data:
    1. The categories with the most samples in the training set:
        a. 30 Km/h speed limit
        b. 50 Km/h speed limit
        c. No passing for vehicles over 3.5 metric tons
        d. Priority road
        e. Yield
        f. Keep right
    2. The categories with the least samples in the training set
        a. 20 Km/h speed limit
        b. Dangerous curve to the left
        c. Pedestrians
        d. End of all speed and passing limits
        e. Go straight or left

3. Design and Test the Model Architecture

a. As a preprocessing step I normalized my data so that the data has mean zero and equal variance. The 32*32 image size remains while the images get transformed to gray.



b. As it is pointed out in the project template the achievable validation accuracy with the LeNet-5 implementation is not more than 89%. In order to be able to reach the minimum 93 % val. acc. I added a few more convolutional layers, increased the depth of the filters and added dropout after the 1st and 2nd Fully connected layers. My network structure is shown below:
   i. Layer 1: Convolutional. Input = 32x32x1. Output = 32x32x6.
   ii. Activation.
   iii. Layer 2: Convolutional. Input = 32x32x6. Output = 32x32x12.
   iv. Activation.
   v. Pooling. Input = 32x32x12. Output = 16x16x12.
   vi. Layer 3: Convolutional. Output = 16x16x16.
   vii. Activation.
   viii. Layer 4: Convolutional. Output = 16x16x32.

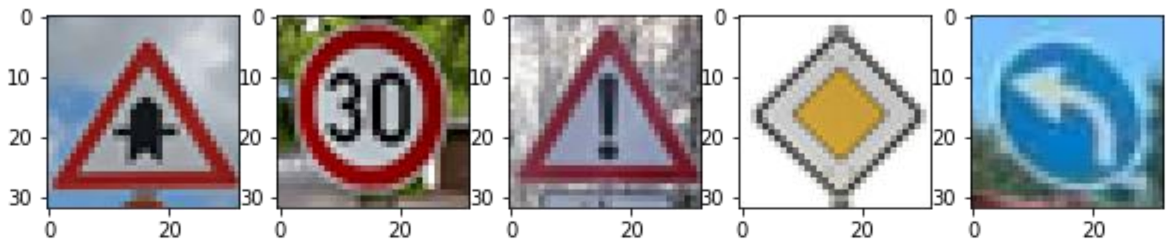
ix. Activation
x. Pooling. Input = 16x16x32. Output = 8x8x32.
xi. Flatten. Input = 8x8x32. Output = 3072.
xii. Layer 5: Fully Connected. Input = 3072. Output = 240.
xiii. Activation
xiv. Dropout
xv. Layer 6: Fully Connected. Input = 240. Output = 120.
xvi. Activation
xvii. Dropout
xviii. Layer 7: Fully Connected. Input = 120. Output = 43.

c. Create placeholders for the batch of input images as well as for the labels and the keep probability hyperparameter for the dropout
d. One-hot encode the labels
e. Set training pipeline hyperparameters
f. At training I reach >95 % validation accuracy

With learning rate set to 0.001, Nr. Of EPOCHS to 30 and batch size 256 with 0.5 keeping probability I reach the required accuracy. Having trained the net with grayscale images the training runs much faster than it would using color images.

4. Test Model on new Images

I found 32x32 size color images



I normalized them and ran the predictions and I got 100 % accuracy. The top 5 softmax probabilities can be found in my notebook.

To be fair I have to admit this test set is as “simple” as it gets, any other results other than 100 % accuracy would be inexcusable on this test set.