Project 2: Traffic Sign Classifier
Self-Driving Car Nanodegree
Lakshay Khatter

Identifying traffic signs is an important part of self driving cars. Knowing how fast you can go on a road, or knowing when to stop are important parts of a self driving car. In order to implement a solution like this we use a technique called deep learning. The idea of deep learning stems from how a brain operates using signals and synapses, neural networks.

A very famous implementation by Yan LeCunn was used in this project. The LeNet architecture uses a convolutional neural network to identify types of numbers in an image. This great part about this approach was that it also works well on classifying traffic signs of each type (43 in total).

Dataset and Summary and Exploration
The dataset came as a trio of files that were pickled for easier use when the time came. After loading these results the appropriate data structure matrix of shape 4-Dimensions. The images are un-rowed and therefore the input vector is 32*32*3, where three is the number of input channels.

A quick summary of the data yields the following output:

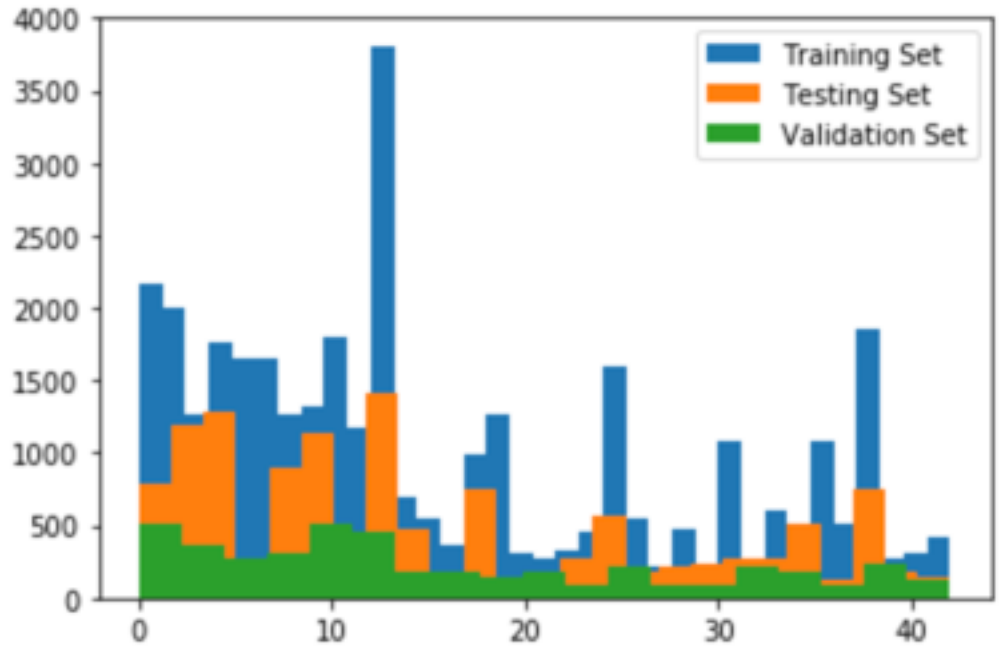
```
Number of training examples = 34799

Number of testing examples = 12630

Image data shape = (32, 32, 3)

Number of classes = 43
```

Exploratory visualization of the dataset



We can see how some classes have more examples than others and may affect results.

Design and Test a Model Architecture
A simple technique to improve our results is to increase the detests with the existing images. Data augmentation takes images and applies certain transformations to them, whether that is rotating, inverting, and discoloring to name a few techniques. There are numerous libraries that can help with this. The main techniques used were to turn the image grayscale. While simple it improved my model accuracy significantly.

The model architecture of LeNet contented no major changes. Some changes were necessary such as changing the number of input features. Originally the architecture was meant to take in 3-channel(RGB) 32x32 images. In our pre-processing phase we turned these photos into grayscale, meaning instead of three channels there was only one and thus reduce the size of the input vector (32*32 vs 32*32*3).

My final model look like something below:

Layer	Description
Input	32x32x6 Grayscale Image
Layer 1: Convolution 3x3	1x1 string, same padding, output: 28x28x6
RELU	Activation
Max Pooling	Output shape 10x10x16
Layer 2: Convolutional	2x2 String, Same padding, 10x10x16
Activation	Relu
Pooling	output: 5x5x16
Flatten	
Layer 3: Fully Connected	Output: 120
Relu	
Layer 4: Fully Connected	Output: 84
Relu	
Layer 5: Fully connected	Output: 10 layers

Describe how you trained your model. the discussion can include the type of optimizer, the batch size, number of epochs and any hyper-parameters such as learning rate.

To train this model I created the following modification:
1) Reduced the learning rate to 0.001
2) Increased the number of epochs to 100
3) Increased the batch size to 128

I achieved this special balance by taking an iterative approach. I found be reducing the learning rate and increasing the number of run throughs on the data set (increasing the number of epochs), the actual performance improved even through it took longer. It was clear that a larger learning rate would sway results significantly. It was clear that LeNet architecture works well because of its results not he MNIST dataset. By applying this architecture along with find tuning the hyper parameters can lead to some excitingly cool results.

Testing the model on a new image


```
[14 15 17 25  3]
```

The model was able to classify 5 out of 5 of the web downloaded images. This compares well due to the accuracy of 0.96 when compared to the test set.

The model in terms of SoftMax probabilities is also 100%.