Self Driving Car Engineer Nanodegree
Project 2: Traffic Sign Recognition
Mano Paul

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

Dataset Summary

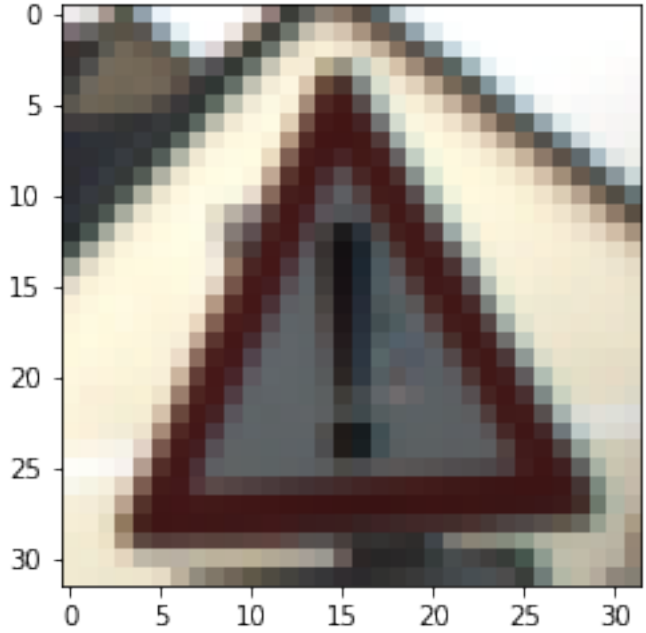
I used the regular python length functions to calculate summary statistics of the traffic signs data set:
• The size of training set is 34799
• The size of the validation set is 4410
• The size of test set is 12630
• The shape of a traffic sign image is 32 x 32 x 3
• The number of unique classes/labels in the data set is 43

```
Summary statistics of the traffic signs dataset

Number of training examples = 34799
Number of testing examples = 12630
Number of validation examples = 4410
Image data shape = (32, 32, 3)
Number of classes = 43
```

Dataset Exploratory visualization of the dataset.

Here is an exploratory visualization of the data set.
It is a random image from the dataset along with its index. The index maps to the label in the reference signnames file that is comma-separated.



In this example, the index was 18. The reference file has the class id of 18 maps to “General Caution”

16	14	Stop
17	15	No vehicles
18	16	Vehicles over 3.5 metric tons prohibited
19	17	No entry
20	18	General caution
21	19	Dangerous curve to the left
22	20	Dangerous curve to the right
23	21	Double curve

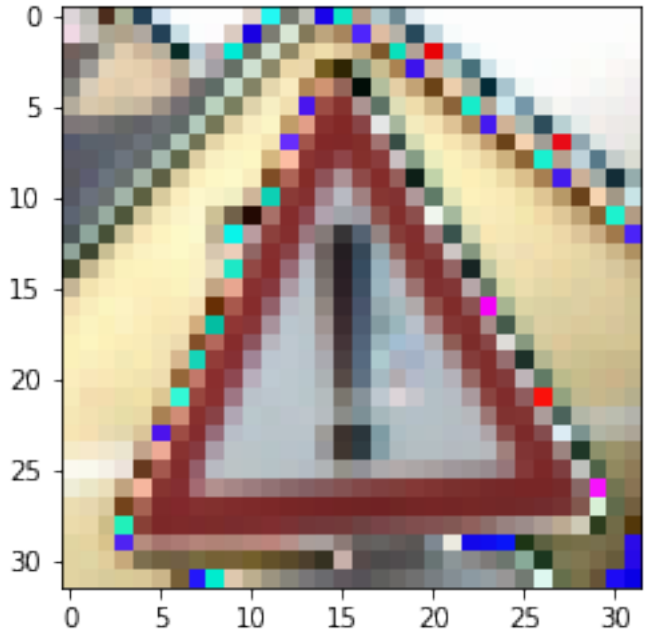
Design and Test a Model Architecture

Pre-processing Image Data

Conversion to grayscale reduced the channel layers from 3 (RGB) to 1 and this posed an issue with array dimensions not matching correctly when training the data using tensorflow. Need to reconcile this but in the interest of time, I resorted to normalization instead of grayscale conversion for preprocessing. The channel reduction would have optimized the training time.

I normalized the image data because this makes the mean value of the image to go toward zero and this helps in optimizing the training. For images since the pixel value is between 0 and 255, subtracting 128 from the pixel value and dividing by 128 can be used to normalize the data. However, I used the
normalized_grayscale_image = a + (((image_data - grayscale_min)*(b - a))/(grayscale_max - grayscale_min)) formula, where a=0.1, b=0.9, grayscale_min = 0 and grayscale_max=255

To test normalization, the random image selected earlier was normalized and the normalized image looked as shown below:



At the end of the preprocessing the training data was shuffled. Shuffling was accomplished using the function shuffle in the sklearn.utils library.

Model architecture

The architecture leveraged was the LeNet model which has the following sequence.
INPUT -> Convolution 1 -> RELU activation -> Pooling ->

Convolution 2 -> RELU activation -> Pooling -> Flatten -> Fully Connected -> Activation -> Fully Connected

The image below shows each layer and the shape of the tensor it outputs. In total, we have 2 convolutional layers and three fully connected layers that output a prediction of 43 logits for the 43 different classes of traffic signs.

My final model consisted of the following layers as shown in the image:

```
ConvNet 1:  Tensor("add:0", shape=(?, 28, 28, 6), dtype=float32)
ConvNet 2:  Tensor("add_1:0", shape=(?, 10, 10, 16), dtype=float32)
Fully Connected 0:  Tensor("Flatten/Reshape:0", shape=(?, 400), dtype=float32)
Fully Connected 1:  Tensor("Relu_2:0", shape=(?, 120), dtype=float32)
Fully Connected 2:  Tensor("Relu_3:0", shape=(?, 84), dtype=float32)
Logits:  Tensor("add_4:0", shape=(?, 43), dtype=float32)
```

Model Training

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 the following:
Model: LeNet
Optimizer: AdamOptimizer

Number of epochs: 50
Batch size: 125
Learning rate of 0.009
Mean: 0
Sigma: 0.1

Model accuracy validation

I had to repeatedly try different epochs, learning rates and batch sizes until the training set accuracy was determined to be greater than 93%

My final model results were:
- training set accuracy of 98.2
- test set accuracy of 89.7
- validation set accuracy of 90.0

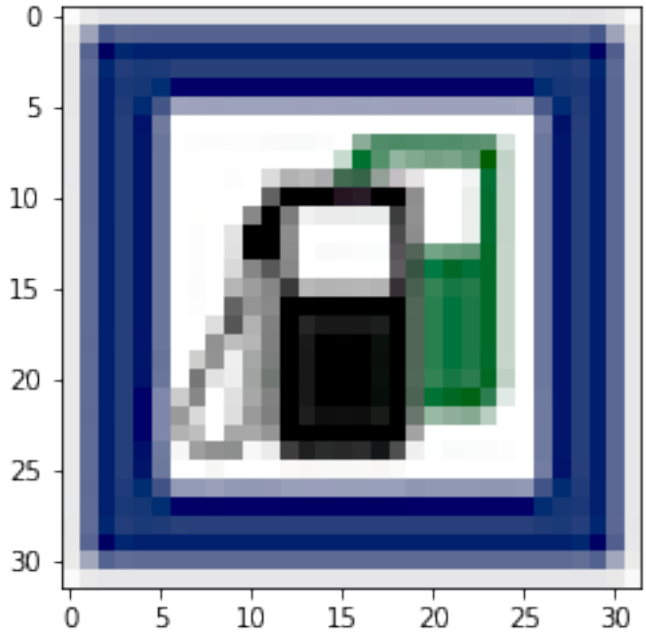
- The LeNet architecture was chosen for its reputation in the industry to be a robust architecture. Since the LeNet architecture takes a 32x32xC where C is the number of channels input and the images in the German Training Dataset is 32x32x3 (RGB) dataset, the LeNet architecture is a good starting architecture for traffic sign classification.
- The final model's validation accuracy was 98.2 and this provides evidence that the model is working well with just normalized data. If the dataset is converted to grayscale, it would speed up the training time.

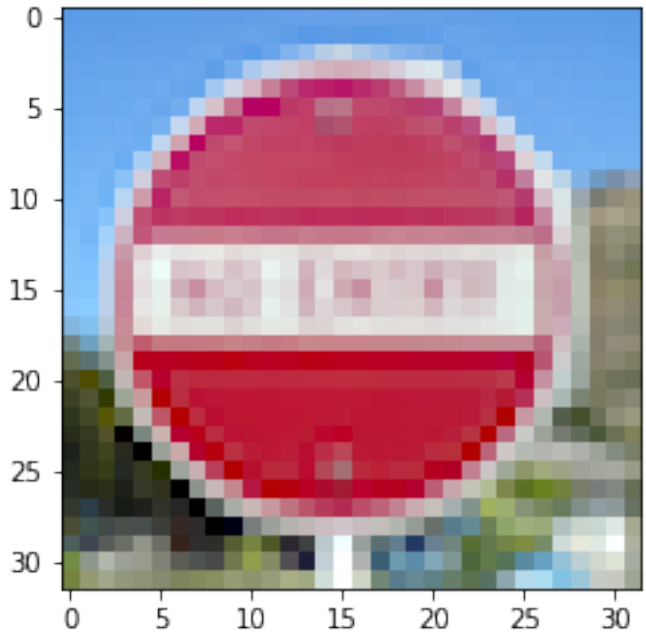
Testing the Model on new images

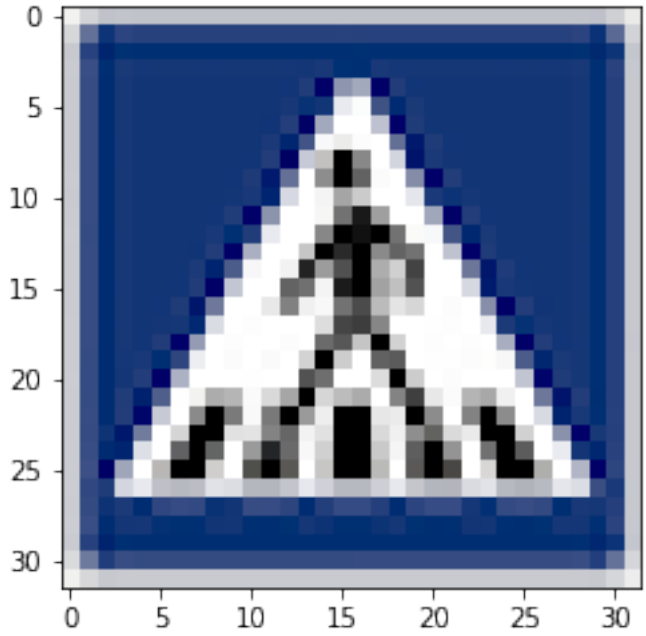
There were six German traffic signs that I found on the web.

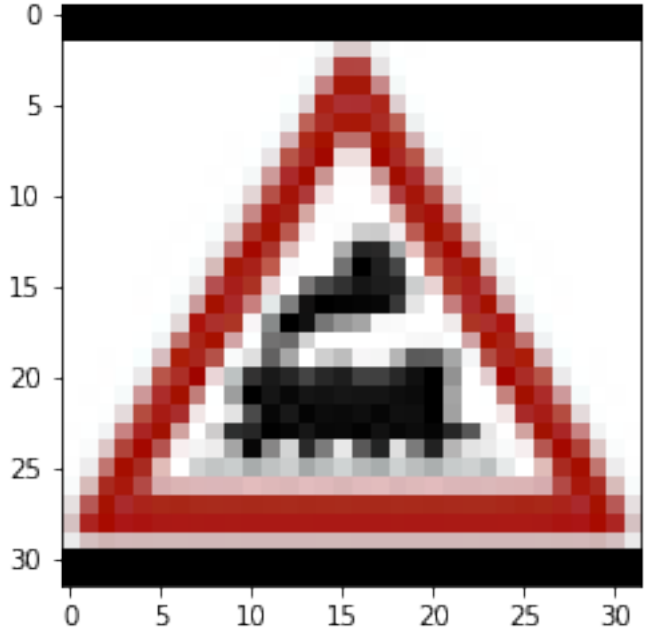
#	**Image**
1	Gasoline
2	No-entry sign
3	Pedestrian crossing
4	Railroad crossing
5	Speed Limit 30km/h
6	Stop

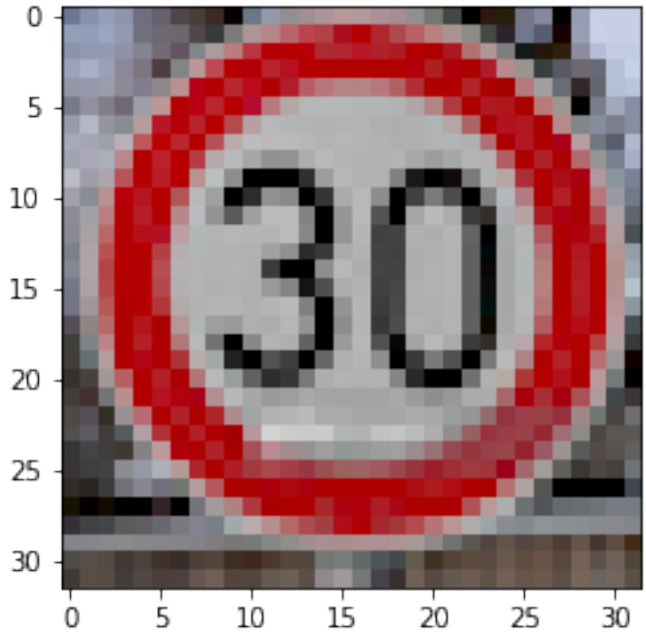
These images were stored into a local directory and loaded for processing.


0
5
10
15
20
25
30
0
5
10
15
20
25
30


0
5
10
15
20
25
30
0
5
10
15
20
25
30


0
5
10
15
20
25
30
0
5
10
15
20
25
30


0
5
10
15
20
25
30
0
5
10
15
20
25
30



After normalizing these images, they were tested against the previously trained LeNet neural network (nn) to see if the nn would predict/classify these images correctly.
The first (Gasoline) and fourth (Railroad Crossing) image might be difficult to classify because they are not there in the 43 classes of traffic signs.

Model's predictions on these new traffic signs

The Sign Names file that was read displayed

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

20 – Dangerous curve to the right

8 – Speed (120 km/h)

5 - Speed limit (80 km/h)

Here are the results of the prediction:
Each of the images were predicted as
Either a dangerous curve to the right (index – 20) or
Speed limit (120 km/h) (index – 8) or
Speed limit (80 km/h) (index – 5)
in that particular order.

Since there is only 1 speed limit sign (30 km/h), and the second most probable predictions was speed limit (120 km/h) and the third most probable prediction was speed limit 80 km/h), while we can say that there was 16.67% (1 of 6 images) accuracy of prediction of the speed limit, but since the prediction of the speed limit sign value (120 km/h and 80 km/h), the prediction can be considered erroneous. This error could have possibly due to the low quality and granulated images.

DID NOT USE THIS
Converting to Grayscale

Using a random image from the dataset, first I converted the image from color to grayscale using opencv library in python. Then I plotted the same color image in grayscale as shown here.
Reading the shape of the color image and the grayscale image indicated that the channel for color image was 3 while there was no value for the grayscale image. Assuming that the channel is 1 for grayscale image. The reason for converting to grayscale is because the training time on 1 channel would be faster than an image with 3 channels.

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