Traffic Sign Recognition

Ahsan Habib

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

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Writeup / README

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. You can use this template as a guide for writing the report. The submission includes the project code.

Data Set Summary & Exploration

1. Provide a basic summary of the data set. In the code, the analysis should be done using python, numpy and/or pandas methods rather than hardcoding results manually.

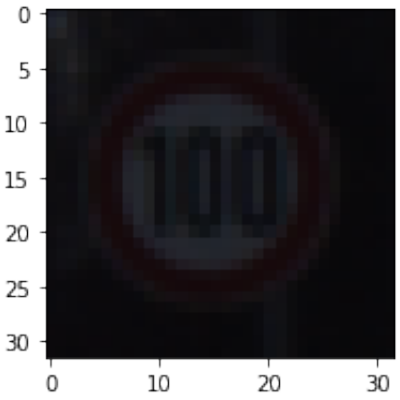
I used the python library to calculate summary statistics of the traffic signs data set:

- The size of training set = 34799

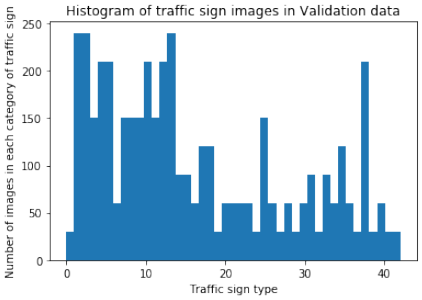
- The size of the validation set = 4410
- The size of test set = 12630
- The shape of a traffic sign image = (32, 32, 3)
- The number of unique classes/labels in the data set = 43

2. Include an exploratory visualization of the dataset.

First, I randomly visualize one image from the training data set. Here is an example:



Next, I draw a histogram of the training, validation and test data set to see the distribution of traffic sign images per traffic sign types. Here is an exploratory visualization of the data set.

		
Training Data	Validation Data	Test Data

Design and Test a Model Architecture

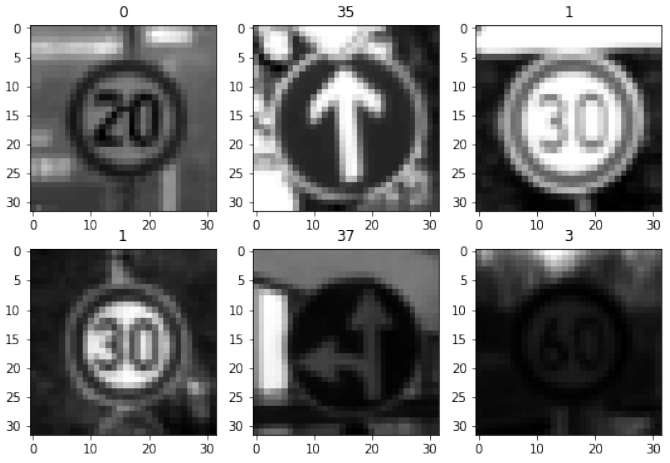
1. Describe how you preprocessed the image data. What techniques were chosen and why did you choose these techniques? Consider including images showing the output of each preprocessing technique. Pre-processing refers to techniques such as converting to grayscale, normalization, etc. (OPTIONAL: As described in the "Stand Out Suggestions" part of the rubric, if you generated additional data for training, describe why you decided

to generate additional data, how you generated the data, and provide example images of the additional data. Then describe the characteristics of the augmented training set like number of images in the set, number of images for each class, etc.)

At the beginning of the project, I build, validate the model without data processing. I used RGB images and train LeNet model with the train data. I was able to achive approximately 90-92% accuracy with this. When I tested this model with German signs, multiple images were failing.

Then, I started doing data normalization. The first step of data normalization was to convert RGB images into a gray-scale images. This reduces to color depth from 3 channels to one. As color is not an attribute of traffic sign, gray scale images reduces the dimension to train the model and it improves the performance of a neural network. It makes the training problem more tractable.

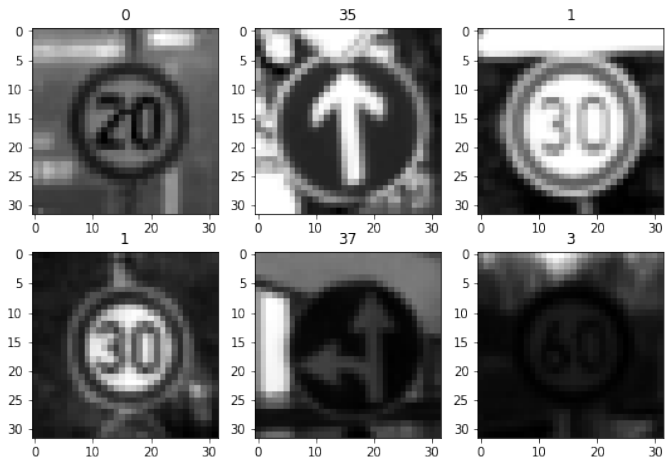
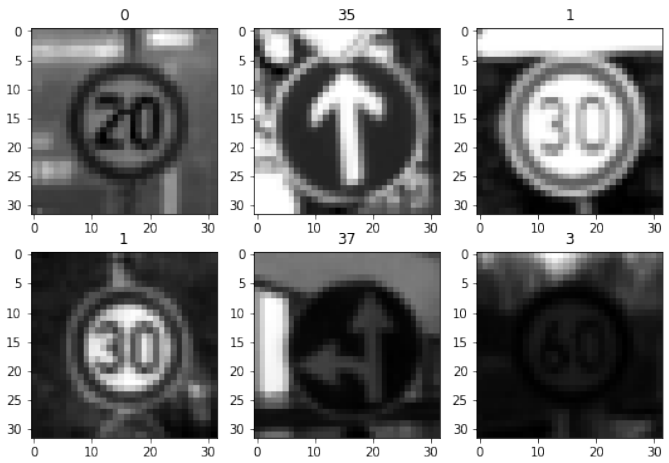
Here is an example of a traffic sign image before and after grayscaling.

	
A Set of Original Images	Grayscale conversion of the Image Set

As a last step, I normalized the image data because it ensures that each input parameter has a similar data distribution. This makes convergence faster while training the network. I have built a normzalization function to bring each pixel value between [0,1].

```
def normalize(x):
    return (x-128.0)/128
```

Here is an example of a set of grayscale images and normalized images:

	
Grayscale Images	Normalized Images

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.

Model Architecture

For the architecture, I mostly for follow LeNet. I have adjusted input and output of each layer to fit the requirements of traffic sign classifier. I have added dropout after activation at Layer 3 and Layer 5.

Here is the description of the architecture:

- Layer 1: Convolutional.
- Activation. RELU
- Max Pooling.
- Layer 2: Convolutional.
- Activation. RELU.
- Max Pooling.
- Flatten. Flatten the output shape of the final pooling layer such that it's 1D instead of 3D by using tf.contrib.layers.flatten.
- Layer 3: Fully Connected.
- Activation. RELU. Then dropout with probability `keep_prob`
- Layer 4: Fully Connected.
- Activation. RELU. Then dropout with probability `keep_prob` .
- Layer 5: Fully Connected (Logits).

- Output: Return the result of the 2nd fully connected layer.

My final model consisted of the following layers:

Layer	Description
Input	32x32x1 Grayscale image
Convolution	1x1 stride, VALID padding, outputs 28x28x6
Activation	RELU
Max pooling	2x2 stride, outputs 14x14x6
Convolution	1x1 stride, VALID padding, outputs 10x10x6
Activation	RELU
Max pooling	2x2 stride, outputs 14x14x6
Flatten connected	Input = 5x5x16. Output = 400
Fully connected	Input = 400. Output = 120
Activation	RELU
Fully connected	Input = 120. Output = 84
Activation	RELU
Softmax	Fully connected logits with 43 outputs

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.

I have use Adam optimizer and feed the loss that tries to reduce the mean of cross entroy of softmax function.

```
logits, L1_activation, L2_activation = LeNetS(x)
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=one_hot_y, logits=logi
ts)
loss_operation = tf.reduce_mean(cross_entropy)
optimizer = tf.train.AdamOptimizer(learning_rate = rate)
training_operation = optimizer.minimize(loss_operation)
```

I have played with the followings hyperparameters:

- learning rate. A higher learning rates improves validation accuracy of the model in the beginning, however, the accuracy plateaus quickly. I have tried 0.1, 0.01, 0.001 and several other number and finally chosen 0.001.
- EPOCHS. I have played with 20 - 50. After 30 the gain is not significant.
- BATCH_SIZE. I have tried with 128 and 256 and finally picked 256
- `keep_prob`. This is the input for dropout function after activation of L3 and L4 level. I have played with 0.45-0.95 and settled down with 0.65 for the final model

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.

First, I used unmodified LeNet without any preprocess of training data RGB data, LeNet --> 86% accuracy

Then, I played with learning rate, epochs, batchsize. It increases the accuracy to roughly 90%

I have modified LeNet to add dropout and with `keep_prob=0.65`, I was able to achieve accuracy close to 92-93%

Then, I have started pre-processing the data. I have converted the RGB image to grayscale and then normalized the data. Now, after playing with learning rate, epochs, batch size and `keep_prob`, I was able to reach 95-96% of accuracy for validation data set.

I have saved all models in the project workspace as a reference.

My final model results were:

- Validation Accuracy = 0.962
- Test Accuracy = 0.936

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.

I have downloaded 16 German traffic signs from http://benchmark.ini.rub.de/?

section=gtsrb&subsection=dataset#Annotationformat. Here are the original images:

When I test my first model based on RGB train data, It had problems properly identifying speed limit 100 mph, stop sign, round about and Vehicles over 3.5 metric tons prohibited.

With data normalization and right tuning of hyperparameters, the model was able to detect most images properly.

Finally, I was able to create a model (saved as lenet-final96) that identifies all 16 images correctly. However, if I

run the training multiple times and create different models, it may result to fail one or two signs. This may due to `keep_prob` that randomly drops some output.

The model mostly had trouble to properly identify Image 9 (Vehicles over 3.5 metric tons prohibited) and Image 3 (RoundAbout Mandatory). From histogram, it shows we have vary small data sample of these image types. Increasing the training data set of these image will improve the validation and test accuracy.

2. Discuss the model's predictions on these new traffic signs and compare the results to predicting on the test set. At a minimum, discuss what the predictions were, the accuracy on these new predictions, and compare the accuracy to the accuracy on the test set (OPTIONAL: Discuss the results in more detail as described in the "Stand Out Suggestions" part of the rubric).

Here are the results of the prediction:

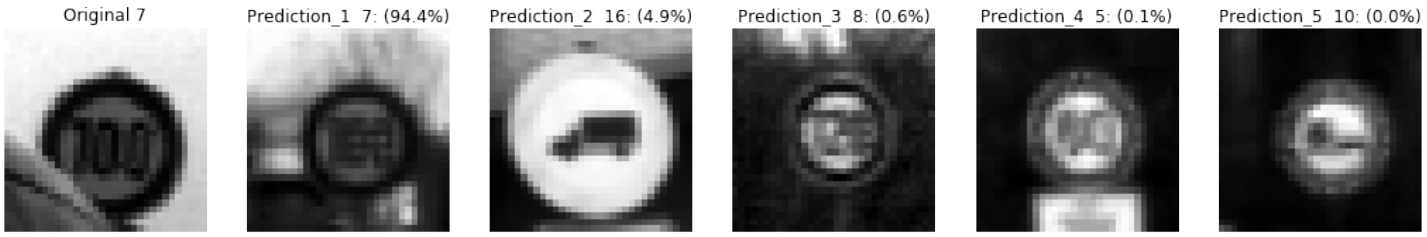
Image	Prediction
Right-of-way	Right-of-way
Traffic signals	Traffic signals
Roundabout Mandatory	Roundabout Mandatory
Bumpy Road	Bumpy Road
Speed limit (100 km/h)	Speed limit (100 km/h)
Yield	Yield
Priority Road	Priority Road
Yield	Yield
No Vehicles over 3.5 tons	No Vehicles over 3.5 tons
Slippery Road	Slippery Road
Speed limit (70 km/h)	Speed limit (70 km/h)
Ahead Only	Ahead Only
Stop sign	Stop Sign
No Entry	No Entry
Speed limit (30 km/h)	Speed limit (30 km/h)
General Caution	General Caution

The model was able to correctly guess 16 of the 16 traffic signs, which gives an accuracy of 100%. This is not always guaranteed. However, with the final model architecture, the detection sometimes fail to one image (accuracy 94%) and occasionally two images (87.5%).

3. Describe how certain the model is when predicting on each of the five new images by looking at the softmax probabilities for each prediction. Provide the top 5 softmax probabilities for each image along with the sign type of each probability. (OPTIONAL: as described in the "Stand Out Suggestions" part of the rubric, visualizations can also be provided such as bar charts)

The top five softmax probabilities for the image for `Speed limit 100 km/h` are:

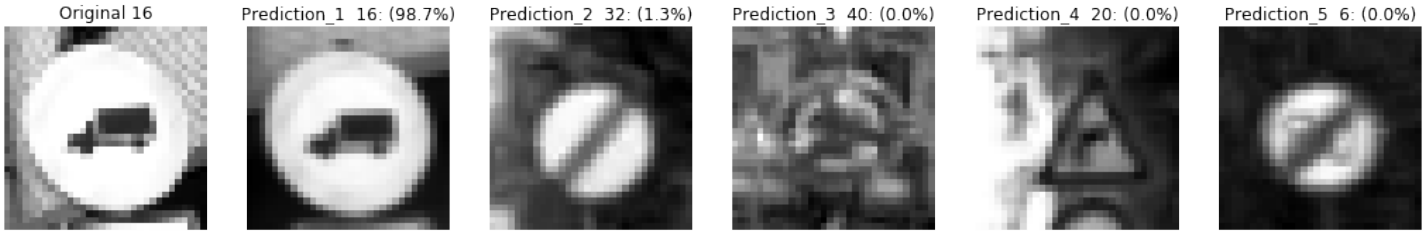
Probability	Prediction
94.4%	Speed limt (100 km/h)
4.9%	No Vehicles over 3.5 tons
0.6%	Speed limt (120 km/h)
0.1%	Speed limt (80 km/h)
0.0%	No passing vehicles over 3.5 tons



Top 5 probability for `Speed limit 100km/h`

The top five softmax probabilities for the image for `Speed Vehicle over 3.5 ton probihibited` are:

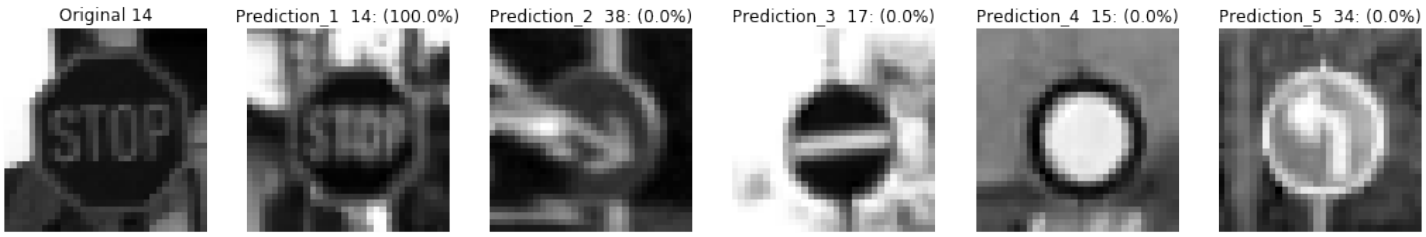
Probability	Prediction
98.7%	No Vehicles over 3.5 tons
1.3%	End of all speed and passing limits
0.0%	Roundabout mandatory
0.0%	Dangerous curve to the right
0.0%	End of speed limit (80km/h)



Top 5 probability for `No Vehicles over 3.5 tons` image.

The top five softmax probabilities for the image for `Stop Sign` are:

Probability	Prediction
100%	Stop Sign
0%	Keep right
0%	No Entry
0%	No Vehicles
0%	Turn left ahead



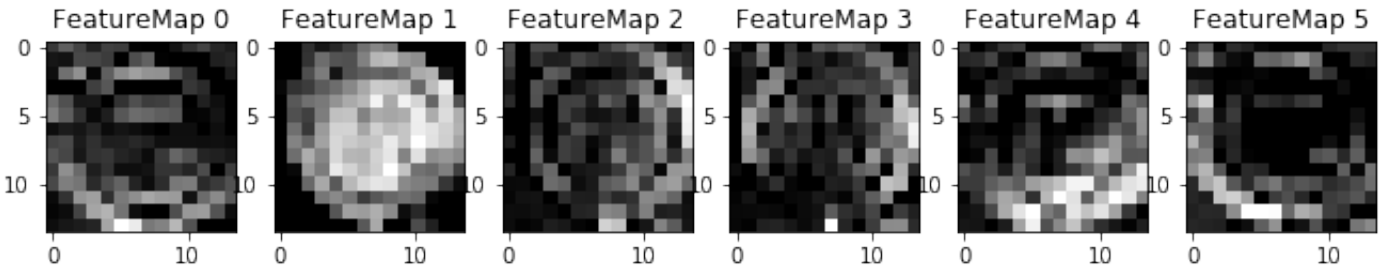
Top 5 probability for `No Vehicles over 3.5 tons` image.

Bar chart of softmax logits for each images are provided in the html file.

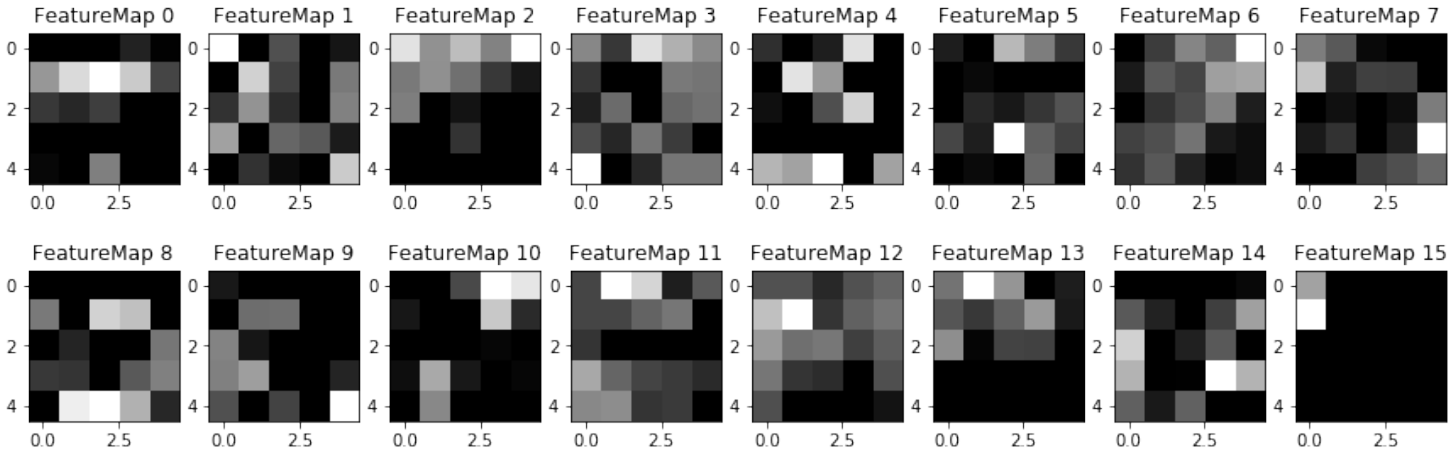
(Optional) Visualizing the Neural Network (See Step 4 of the Ipython notebook for more details)

1. Discuss the visual output of your trained network's feature maps. What characteristics did the neural network use to make classifications?

The LeNet model returns L1 and L2 activation output with the final logits of the model. The test image being fed into the network to produce the feature maps I have used the provided default value of activation contrast. Here are the feturemaps for L1 and L2 activation output feature maps:



L1 activation output feature maps



L2 activation output feature maps