Traffic Sign Recognition

Writeup

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

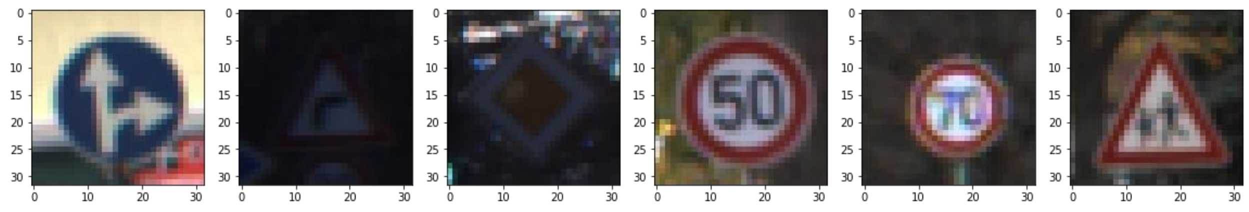
Data Set Summary & Exploration

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

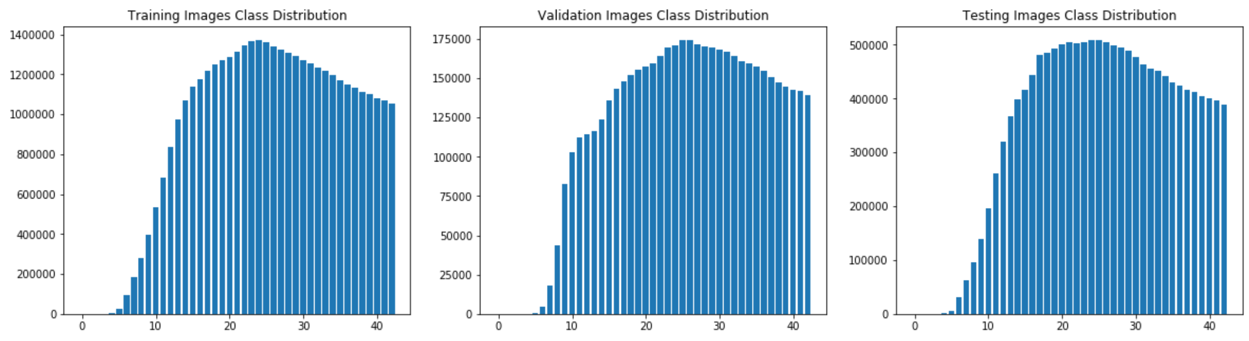
- The size of training set is **34,799** images
- The size of the validation set is **4,410** images
- The size of test set is 12,630 images
- The shape of a traffic sign image is **(32, 32, 3)**
- The number of unique classes/labels in the data set is **43**

Here are exploratory visualizations of the data set.

a) Sample images from the training set:



b) Image class distributions on the training, test, and validation sets:



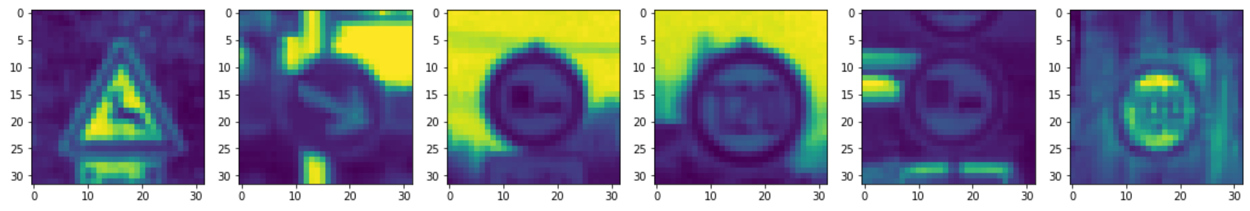
Design and Test a Model Architecture

Image Preprocessing

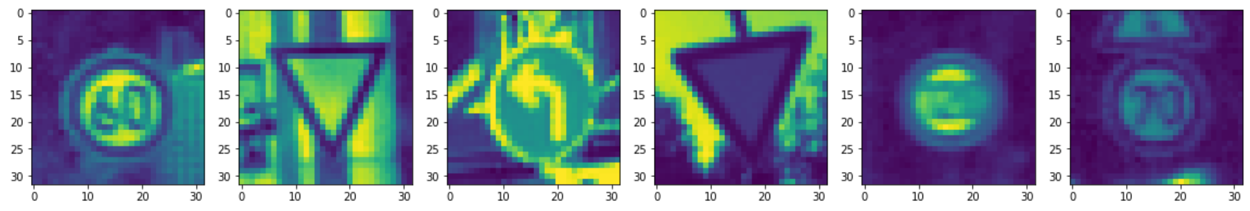
As this was a fairly simple machine learning problem with relatively clean and standard data, the only image preprocessing step that I deemed necessary for a successful deep neural network was to convert the images to grayscale. This is because the colors in the images are not an important element for the neural network to learn as features. It would be best for the neural network to train on the patterns of the traffic sign images with one color channel.

Below are sample images of the dataset in grayscale:

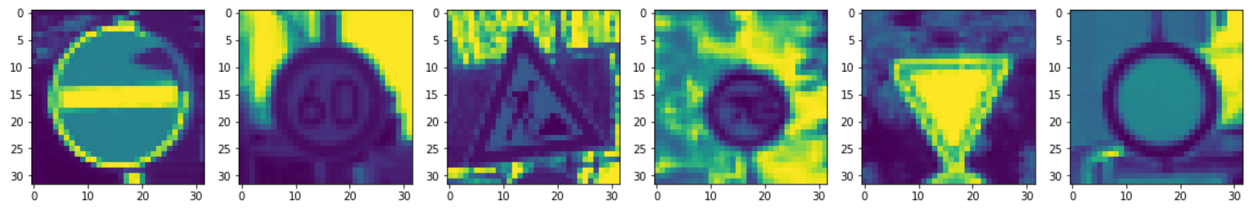
```
Sample Training Images
```



```
Sample Validation Images
```



```
Sample Testing Images
```



Model Architecture

My final model consisted of the following layers:

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_20 (Conv2D)           (None, 30, 30, 28)        280
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 15, 15, 28)        0
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 13, 13, 32)        8096
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 6, 6, 32)          0
_________________________________________________________________
flatten_10 (Flatten)         (None, 1152)              0
_________________________________________________________________
dense_46 (Dense)             (None, 150)               172950
_________________________________________________________________
dense_47 (Dense)             (None, 150)               22650
_________________________________________________________________
dense_48 (Dense)             (None, 150)               22650
_________________________________________________________________
dense_49 (Dense)             (None, 150)               22650
_________________________________________________________________
dense_50 (Dense)             (None, 512)               77312
_________________________________________________________________
dense_51 (Dense)             (None, 43)                22059
=================================================================
Total params: 348,647
Trainable params: 348,647
Non-trainable params: 0
```

In summary, my model consisted of two convolution layers that applied 28 and 32 filters respectively so as to identify as many features as possible from the images without overfitting. On both convolution layers, I applied a kernel size of 3*3 to maximize efficiency. Each convolution layer was also coupled with a Pooling layer of pool size of 2*2 so as to shrink the filter data by a factor of 2. I then applied a Flatten layer to prepare my data for entry into Dense layers. I applied 6 Dense layers to my model, the first four having 150 units with Rectified Linear Unit (ReLU) as the activation function. The fifth Dense layer had 512 units but also used ReLU as an activation function. The final layer had 43 units to match the classes in my dataset and used softmax as an activation function so as to use probabilities [0, 1] to make predictions.

To train the model, I settled on 20 epochs while implementing an epoch callback class that would halt training if the validation accuracy was above 0.9; a clear sign of overfitting. I then included my validation data so as to allow the model to track its progress. As this was a simpler model with little data, I found no need to train on batch sizes especially because I was training on a dedicated GPU instance.

The loss function I decided to use was sparse categorical cross-entropy as I was making multi-class predictions and I used the Adam optimizer as the data was not too complicated as to tweak the learning rate.

Analysis of Results

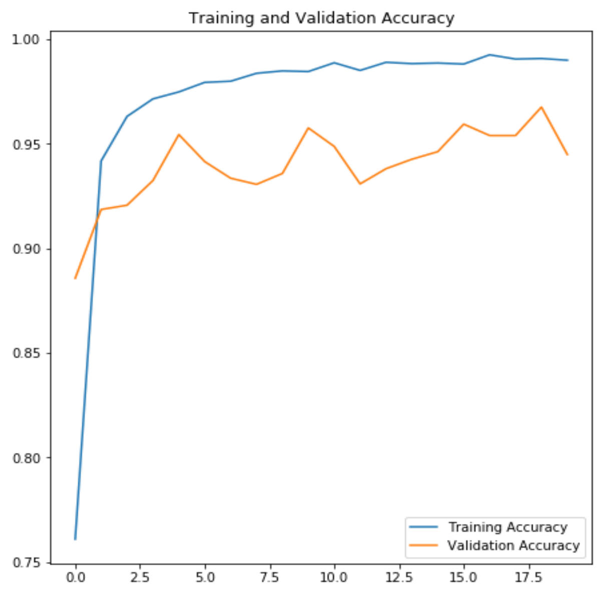
My final model results were:

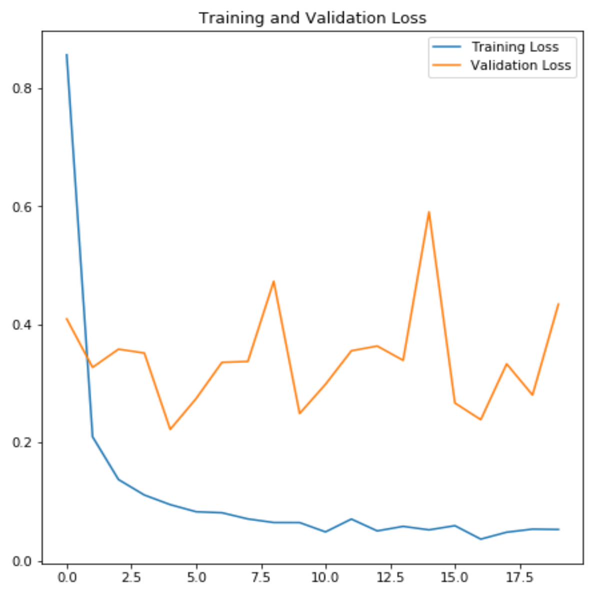
- Training set accuracy of 0.990
- validation set accuracy of 0.945
- test set accuracy of 0.930

To achieve the results above, I applied an iterative approach.

I began with a simple model with two convolution layers but only one dense layer. I then realized that when I increased the number of dense layers without increasing the input units, my validation accuracy would then increase as well. I finally settled on 6 dense layers for a final result of 0.945 on the validation set.

However, I also realized that the more I added dense layers, the more my model was tending towards overfitting as visualized below:

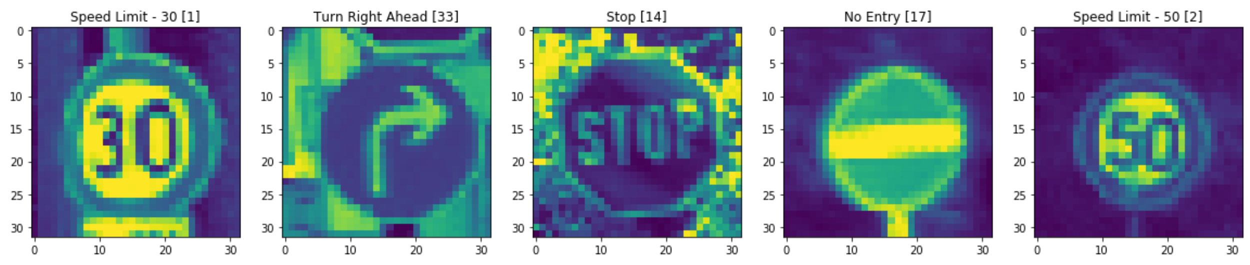




Despite of this, the validation accuracy increased at par with the training accuracy throughout the model thus there was no need of applying a dropout layer to the model.

Outsourced Images

Here are five German traffic signs that I found on the web:



All images sourced from the web were predicted correctly.

Here are the results of the prediction:

Image	Prediction
Speed Limit (30 km/hr.)	Speed Limit (30 km/hr.)
Turn Right Ahead	Turn Right Ahead
Stop	Stop
No Entry	No Entry
Speed Limit (50 km/hr.)	Speed Limit (50 km/hr.)

The model was able to correctly guess 5 of the 5 traffic signs, which gives an accuracy of 100%. This compares favorably to the accuracy on the test set of 93%.

For all images, the model is certain with a probability of 1.0 that the image is as it predicted.

Probability	Sign
1.00	Speed Limit (30 km/hr.)
1.00	Turn Right Ahead
1.00	Stop
1.00	No Entry
1.00	Speed Limit (50 km/hr.)