Traffic Sign Classification Project

Writeup

The goals of this project are the following:

1. Load the data set (see below for links to the project data set)
2. Explore, summarize and visualize the data set
3. Design, train and test a model architecture
4. Use the model to make predictions on new images
5. Analyze the softmax probabilities of the new images
6. Summarize the results with a written report

Data Set Summary & Exploration

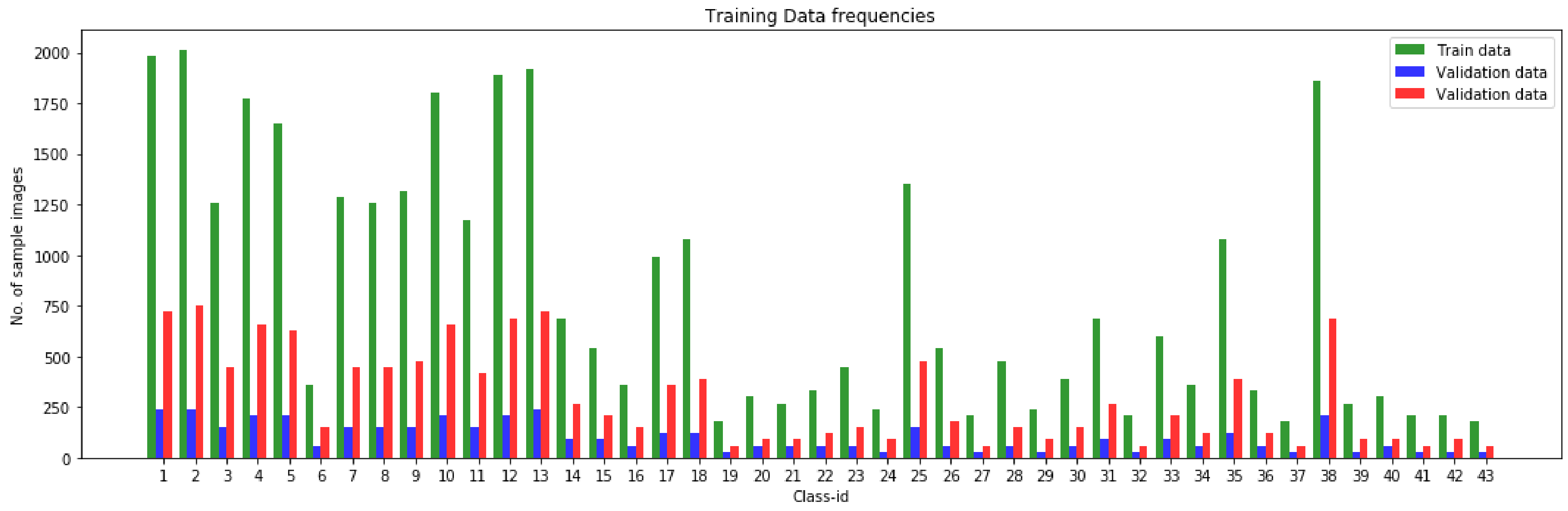
1. Providing the basic summary of the dataset.

In the code since all the **train_dataset**, **validation_dataset** and **test_dataset** were all numpy ndarray, therefore I used the shape attribute to get the required values.

- 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 pixels
- The number of unique classes/labels in the data set is 43

2. Including an exploratory visualization of the dataset

Here is an exploratory visualization of the dataset showing all the training, validation and test data, by plotting their frequencies against their respective class-ids.

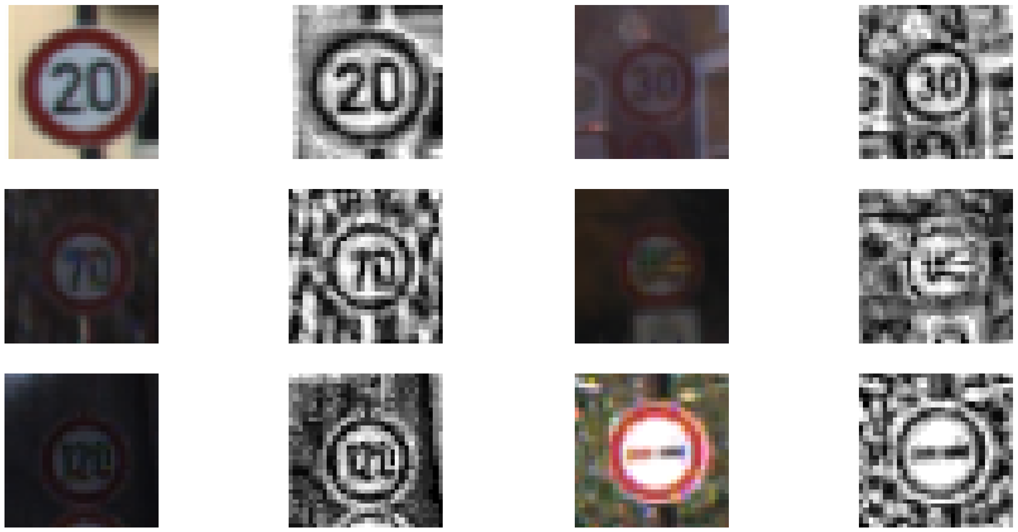


Design and Test a Model Architecture

1. First I decided to preprocess the colored images to grayscale (single channel). This helped as:

- From three channel RGB image I could use a single channel greyscale image which I can feed into my convolutional network
- In function preprocess_dataset (adapted from article in http://navoshta.com/traffic-signs-classification), the images are:
    - Converted to greyscale
    - Pixel values are scaled from [0, 255] to [0, 1]
    - Adaptive histogram equalization is applied, since it helps greatly in feature extraction

  Some examples are:



- Original Images and their preprocessed results, for all 43 classes are displayed in .ipynb file under visualization section

2. I have used ***LeNet***, with slight modification. The Model Architecture is as follows:

Layer	**Description**	
Input	32x32x1 Grayscale image	
Convolution (5x5)	Input: 32 x 32 x 1	Output: 28 x 28 x 6
RELU		
Max Pooling	Input: 28 x 28 x 6	Output: 14 x 14 x 6
Convolution (5x5)	Input: 14 x 14 x 6	Output: 10 x 10 x 16
RELU		
Max Pooling	Input: 10 x 10 x 16	Output: 5 x 5 x 16
Flatten	Input: 5 x 5 x 16	Output: 400
Fully connected layer	Input: 400	Output: 120
Dropouts	*during training keep_prob = 0.45	
Fully connected layer	Input: 120	Output: 84
Dropouts	*during training keep_prob = 0.5	
Fully Connected Layer	Input: 84	Output: 43

3. Training the model

To train the model I have used the following parameters:

a. **Batch Size**: 128
b. **EPOCHS**: 120
c. **Learning rate**: 0.0001

For optimized I used 'Adam Optimizer'. It uses Adam optimization algorithm which is an extension to stochastic gradient descent.

In local machine due to low computation power, I ran the training for only 25% of the training dataset with 50 EPOCHS, learning rate of 0.0001 and batch-size of 128. Using these parameters, the validation accuracy reached approximately 85% (since I was using a well-known architecture, LeNet). However I was interested in the Loss curve which I plotted for each epochs. After verifying that the learning rate was good, I moved the code to AWS.

4. Training on AWS

After moving the code to Amazon Web Services, which provided GPU instances, each epoch was quite quickly executed and analyzed. Since the LeNet Architecture performs great in classifying MNIST dataset of hand-written digits, it was expected to perform well in classification German traffic signs into 43 classes.

In the initial stages the parameters used were as follows:

Batch Size	Epochs	Learning Rate	Dropout's 'keep_prob'
128	50	0.001	0.85

This however yielded to validation accuracy of approx. 86%

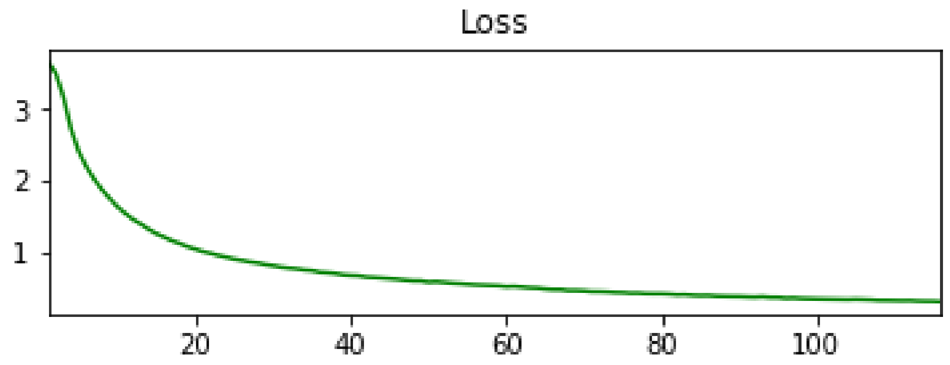
Next the number of epochs were increased to 150. But then the loss curved showed signs of over-fitting, however the validation accuracy reached around 91%

Next the Learning rate was decreased to 0.0001, the loss curve that was plotted now was better and showed a good learning rate.

After fine tuning the final parameters of the model were:

Batch Size	Epochs	Learning Rate	Dropout's 'keep_prob'
128	120	0.0001	0.5

The loss curve obtained



The final model's results were

Validation Accuracy	0.931
Loss	0.298764
Test Accuracy	0.9228

Testing the model on new images

Images downloaded from web

Right-of-way at the next intersection

Speed limit (60km/h)

Yield

Keep right

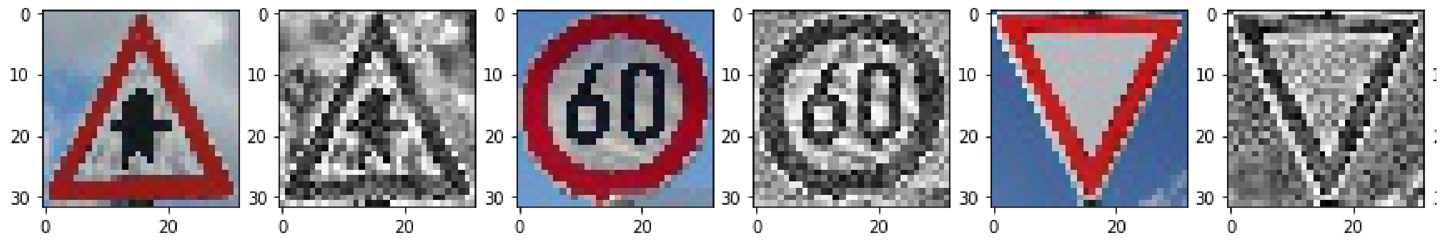
No entry

Road work

Children crossing

Stop

The images were resized to 32 x 32 pixels and preprocessed before testing with the model. Examples are:



The predictions of the model are:

Image	**Prediction**	**Result**
Right-of-way at the next intersection	Right-of-way at the next intersection	Correct
Speed limit (60km/h)	Speed limit (50km/h)	Wrong
Yield	Yield	Correct
Keep right	Keep right	Correct
No entry	No entry	Correct
Road work	Road work	Correct
Children crossing	Children crossing	Correct
Stop	Stop	Correct

The model predicted 7 out of 8 images correctly, with an accuracy of 0.875 or 87.5%.

Predictions in terms of Softmax probabilities

Let us analyze the top 5 softmax probabilities of the first and second images, since first one is correct prediction and second one is wrong prediction

First Image		**Second Image**	
Probability	*Prediction*	*Probability*	*Prediction*
Right-of-way at the next intersection	0.851	Speed limit (50km/h)	0.368
Beware of ice/snow	0.134	Speed limit (80km/h)	0.312
Children crossing	0.074	Speed limit (60km/h)	0.207
Roundabout mandatory	0.014	Speed limit (30km/h)	0.053
Bicycles crossing	0.012	End of speed limit (80km/h)	0.048

For the first image the model is certain that it is 'Right-of-way at the next intersection' sign. But for the second image we can see that probabilities of signs 'Speed limit (50km/h)', 'Speed limit (80km/h)' and 'Speed limit (60km/h)' are close (however Speed limit (50km/h) was the highest). This may be due to pattern in which the digits 5, 6, 8 are written.

The graphical representations of the top 5 predictions for all images are provided in .ipynb file