Traffic Sign Recognition

Udacity - Self-Driving Car Engineer Nanodegree

Sulabh Matele - March 26, 2017

Goal and steps of the project

The goal is to create a traffic sign recognition program using convolutional neural network.

The steps of this project are the following:

- Load the 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:

— Basic Summary of the Data Set

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

The code for this step is contained in the second code cell of the IPython notebook.
I used the 'numpy' library to calculate summary statistics of the traffic signs data set:

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

Number of input samples and shape was calculated using 'len()' and '.shape'.
Other than these, the 'np.unique' was used to get the number of unique class ids and also for getting their indices and repetitiveness to visualize the data more clearly and also that helped in mapping the data easily in next section.

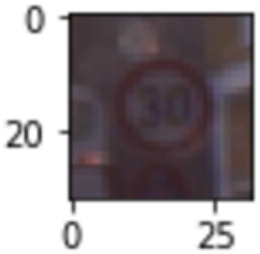
— Include an exploratory visualization of the dataset

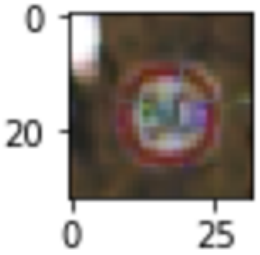
The code for this step is contained in the third code cell of the IPython notebook.

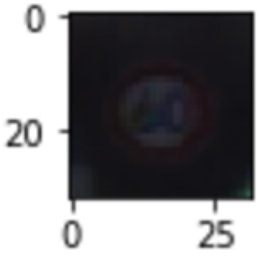
At this step initially all the unique class Id images' first occurrence is printed using the indices received in the last step, and also by using the repetitiveness for each class id, it was easy to find out how many samples are available per class id.

This visualization provides a good quick overview about, quality and visuals of the image and also it becomes clear that, some class ids has more samples than others.

Here is an exploratory visualization of the data set.



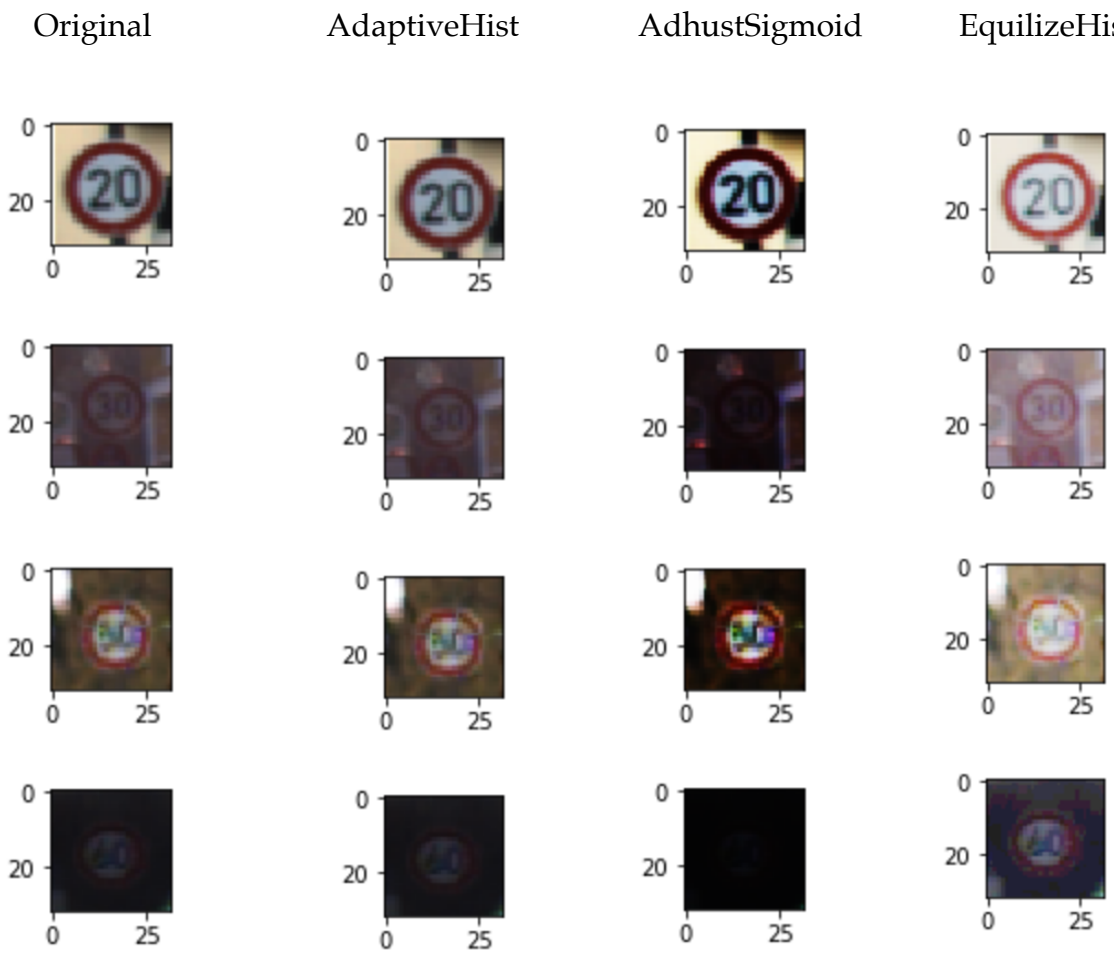




— Design and Test a Model Architecture

The code for this step is contained in the fourth code cell of the IPython notebook.
Since the images could be of various intensity and from different angles, it was required to find out a way to make images more understandable for the model.

I tried many preprocessing techniques and following are some of the results as compare to original images.



These techniques and their outputs were tried with the whole design and found that the "EquilizeHist" provides the best results. It is visually also very clear that, this techniques handles bright and dark images equally and adjust the Histogram in such a way that image becomes more clear.

— Model Architecture

The LeNet architecture is used as the base for this project.

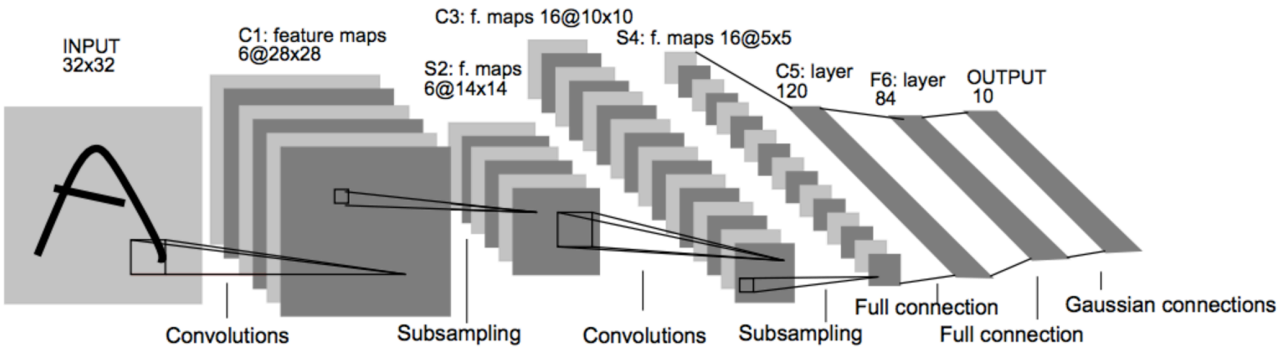


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Please find the configuration of layers used:

Layer	Functionality Type	Input	Output
Layer 1	Convolutional - relu activation	32x32x3	28x28x6
	Maxpool - Filter 2x2	28x28x6	14x14x6
Layer 2	Convolutional - relu activation	14x14x6	10x10x16
	Maxpool - Filter 2x2	10x10x16	5x5x16
	Flatten	5x5x16	400
Layer 3	Fully Connected - relu activation	400	120
Layer 4	Fully Connected - relu activation	120	84
Layer 5	Fully Connected - relu activation	84	43

— Train, Validate and Test the Model

As mentioned in the previous section about different adjustment and configuration of model hyper parameters, following were the parameters which worked well to increase the validation accuracy to 93%+.

Hyper Parameter/ configuration

Epchos - 10
Batch Size - 256
mu - 0, sigma - 0.15
Learn Rate - 0.006
Activation - relu
Padding - VALID
MaxPool Filter - 2x2
Optimizer - AdamOptimizer

During training the model, the values of training set are shuffled before creating a new batch to provide more robust training. The loss is calculated at the end of each training iteration optimized using "AdamOptimizer" which is most popular for adaptive optimization.

The trained model fed with validation samples to calculate the validation accuracy as well.

Final model results were:

Training set accuracy of **99.1%**
Validation set accuracy of **93.3%**
Test set accuracy of **91.4%**

Initially this architecture with default learning rate and other hyper parameter gives only approximate 89% of validation accuracy, then there were many iterations and techniques were tried to make the stable parameters to achieve 93%+ accuracy for trained network.

Some techniques / adjustments which were tried and adjusted:

- Adjustment of learning rate - then 0.006 worked well for me
- Different activation functions relu6, relu, elu, softplus were tried to compare the best performance of the network - then relu worked well for me.
- Increasing the number of 'epochs' which started causing the overfitting at some point, to avoid this the dropout and regularization were used - then with other parameter adjustment, it was not required to keep dropout and regularization.
- Tried adding more convolution layers after 1st and 2nd convolution, which did not help much in increasing the validation accuracy and then were removed.
- Different batch sizes - then 256 found worked well with 'epchos' value as 10.
- Different sigma values were tried - 0.15 worked well here.
- Valid and SAME padding techniques were tried.

— Test a Model on New Images

To test the trained network well, it was required to test the model with random internet images of the German traffic signs.

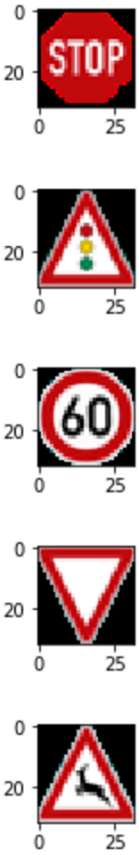
The downloaded images were of different sizes and with 4 color channels instead of 3, as it was used in our network to train. So, it was required to process the images and convert them to make them usable.

PIL.Image lib was used to resize image to 32x32 (with filter “Image.BICUBIC”) and also to convert from 4 channel to 3 channel color image.

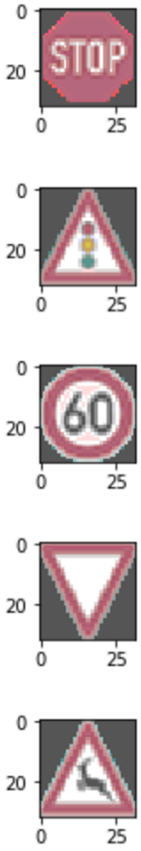
Before feeding the images to trained network it was also required to preprocess the images with the same processing as it was used on training and validation sampled. (For my network it was “EquilizeHist”)

Following are the processed images :

Resized Images



EqualizeHist



Stop Sign - Class id - 14
Traffic - Class id - 26
SpeedLimit - Class id - 3
Yield Sign - Class id - 13
Animal Sign - Class id - 31

The result of test:

Expected O/p = [14 26 3 13 31]
NW Predictions = [14 26 23 13 31]

Network was able to predict the new signs with 80% accuracy.

For understanding more about, why the network was not able to predict 1 image properly, we need to know, if the correct class id was even a probable response ?

For understanding this, we print the softmax probabilities for top 5 predictions and also the top 5 probable class ids:

Top 5 probable predicted classes:

14 —>	[14, 17, 3, 12, 7],
26 —>	[26, 18, 27, 11, 20],
3 —>	[23, 38, 2, 3, 25],
13 —>	[13, 35, 12, 2, 9],
31 — >	[31, 21, 11, 23, 1]

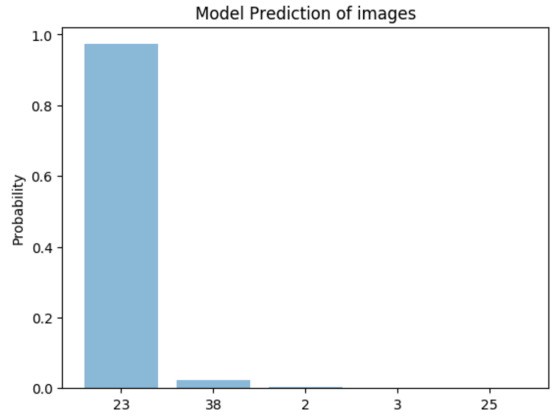
We will now plot the probability of each class to get predicted.

By printing the softmax probabilities for these predictions we get following top 5 probabilities, and possible predictions from the trained network.

Expected - 14

Expected - 26

Expected - 3



Expected - 13

Expected - 31

Analysis:

By carefully checking the probability plots, its evident that the trained network is quiet sure about the predictions. But still we received one wrong prediction.

Speed limit 60 sign (Class id 3) was detected as Slippery Road (Class id 23)

The possible causes for the wrong prediction could be:

- The network needs more careful training with better parameters.
- The image from the internet which was processed further to fed to trained network needs more processing.
- The network could be trained with better preprocessed images, like gray scale etc to make color and intensity independent.