UDACITY Self Driving Car Engineering

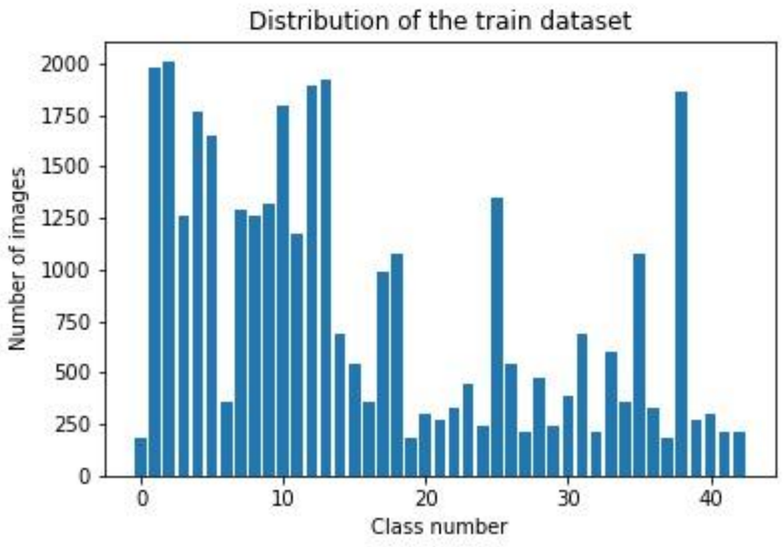
Project #2: Traffic Sign Classifier

Pipeline:

1- Explore the input data

- 32*32 RGB images
- 43 classes, uneven distribution

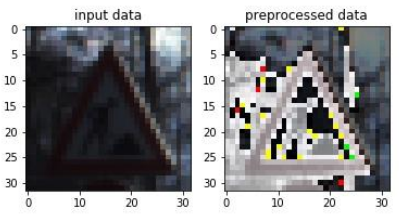
As seen below, some of the classes are underrepresented. Thus, it is required to augment the data, especially those who have low number of samples.



2- Preprocessing

To standardize the input set,

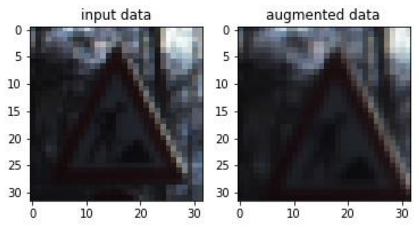
- RGB images are normalized to -0.5 +0.5 range
- Histogram equalization is performed



3- Augment the data

To even the distribution of the input data, augmented images added to those classes who have lower representations. Augmentation includes,

- (random) translation of the image on both x and y axis
- (random) scaling of the image



I implemented the augmentation in such a way that each class will have at least 300 samples.

*The test and validation data were not augmented.

4- Model Architecture

Firstly, the images are trained as RGB, NOT grayscale. I tried different architectures but the lenet gave the best performance for my model.

- 1 - Conv 5*5*3*6, relu, maxpool
- 2- Conv 5*5*6*16, relu, maxpool
- 3- Dense, inp:400, out: 120, relu, dropout
- 4- Dense, inp:120 out:84, relu, dropout
- 5- Final layer, inp:84 out: 43, softmax

*** Below is added after the first review feedback***

First architecture I have tried is as such:

- 1 - Conv 5*5*3*6, relu
- 2- Conv 5*5*6*12, relu
- 3- Conv 5*5*12*24, relu
- 4- Dense, inp:96, out: 1024, relu, dropout
- 4.5-(Optinal alternative) Dense, inp:1024 out:256, relu, dropout
- 5- Final layer, inp:256 out: 43, softmax

It did not succeed because I think 3 convolutional layer on a 32*32 input is overkill. It might still perform better but tuning it is harder since there is more layers and more parameters to be tuned. As instructors at Udacity also mentined it, the model architecture is more of an experimental thing, rather than analytically calculate how many layers are needed etc. Thus, in my case, the second model (the one that is used in this submission) performed much better and it was easier to tune.

I tuned model sizes and hyperparameters mainly from the losses of training and validation sets.

- If the training loss is saturated, ie do not go below some level after a while, that means the epoch number should be reduced.
- If in the end the training loss is much more smaller than the validation loss, then the model is overfitting and vice versa. Then I updated the parameters to reduce the overfit / underfit.

To sum up, I tuned parameters experimentally, but by analyzing the behaviors of training and validation losses.

I think most importing design choice is starting from a proven model and further improving it to fit your needs. It is not a copying, in contrast, it is not inventing the wheel again. Once I had the basic model, I know how to improve it by analyzing its feedbacks as I improve the model which I describe above how I did it.

*** Above is added after the first review feedback***

5- Training

NVidia Gtx860m is used for training, and it took less then hour (of course countless hours of experimenting is not included☺)

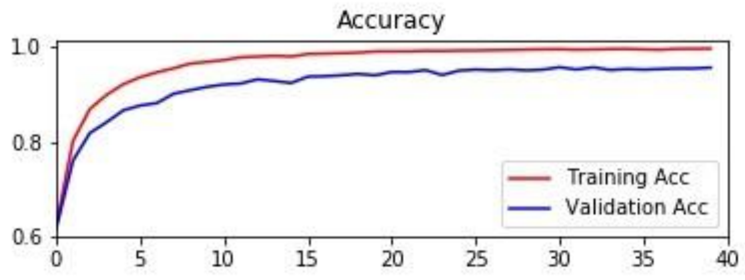
- Static (constant) learning rate of 0.001
- 35 epochs

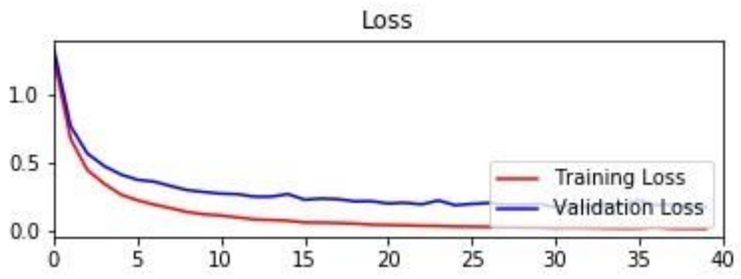
At each epoch, accuracy and loss curves of both training and validation set is logged.

6- Results

The model did really good job on learning the traffic signs,

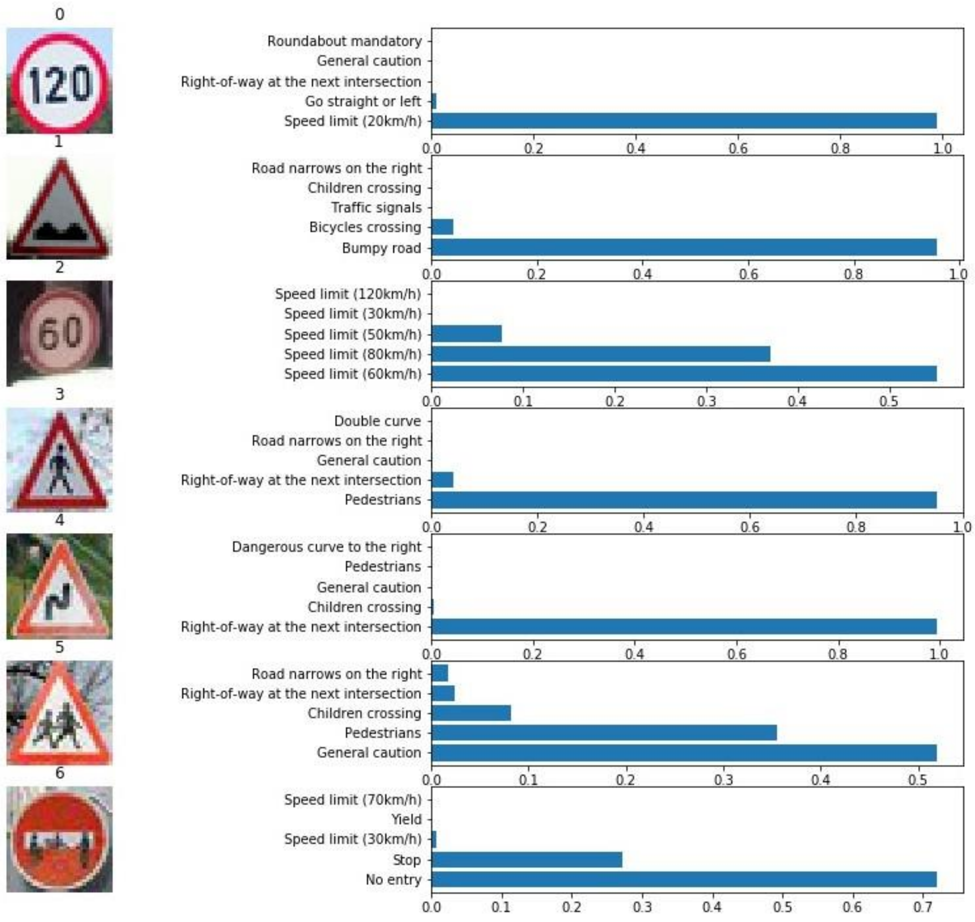
- 99.7% training
- 95.7% validation
- 94.2% test accuracy





7- New images

The new images are found on web, some of them are highly distorted, and the model predicted them as seen below, the accuracy on the new images is 5/7 =71%



It mispredicts 120 kmh sign to 20 kmh, which is extremely similar (1 digit away)

Also, it predicts Pedestrians sign with 35%, but 55% of "General Caution" outweights it.

Finally, it correctly predicts the "highly distorted" no entry sign correctly☺ The advantages of using RGB can be shown here, red color helps the model identify the sign.

*** Below is added after the first review feedback***

Even though some predictions are wrong but extremely close, if we analyze it as a binary True prediction – False prediction, the models accuracy on the new images is 5/7 or 71.4%.

- 99.7% training
- 95.7% validation
- 94.2% test accuracy
- 71.4% accuracy on new images

As clearly seen, the model is overfitting since it performs extremely well on training images and not so well on the new test images.

To reduce overfitting further, l2 regularization, higher rates of dropout and further modifying the model architecture are some of the possible solutions.

*** Above is added after the first review feedback***

Possible Future Improvements

- The model could be altered, ie adding inception layers, experimenting with sizes of conv layers etc.
- The augmentation might be performed thoroughly, like brightness changing, blurring / sharpening, warping etc.

Rubic Criterias

Submission Files: All the necessary files were uploaded.

Dataset Summary & Exploratory Visualization: 1 sample of each class and distribution of class sizes are given graphically.

Preprocessing: The preprocessing techniques discussed above.
Model architecture & training: The architecture and training details are given above.
Solution approach: The accuracy on the validation set is 95.7% and on the test set 94.2s%.

Acquiring New Images & Model Certainty: Top 5 softmax probabilities of new images and corresponding details are given above.