Traffic Sign Recognition Project

Introduction

The goal of this project is to develop a software pipeline that identifies the traffic sign recognition though deep learning training and evaluation from traffic sign data.

Environment

- Linux 18.04, Python3.6, OpneCV, Tensorflow, Numpy

File

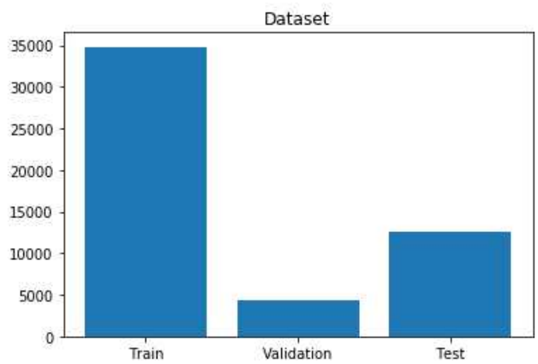
- Traffic_Sign_Classifier.ipynb : main code (train and evaluation)
- Checkpoints : Train model results (best weight) folder.

Step of this project are the following

- *step 1 : Load the data set (see below for links to the project data set)*
- *step 2 : Explore, summarize and visualize the data set*
- *step 3 : Design, train and test a model architecture*
- *step 4 : Use the model to make predictions on new images*
- *step 5 Analyze the softmax probabilities of the new images*

step 1 : Load data set

- containing "train data", "verification data", and "test data" and read from the pickle file. Later, the images and labels are split and stored in variables.
- Get the amount of each data and the number of label types for the size of the image.




```
Number of training examples = 34799
Number of validation examples = 4410
Number of testing examples = 12630
Image data shape = (32, 32, 3)
Number of classes = 43
```

[Image 1. Dataset Visualization]

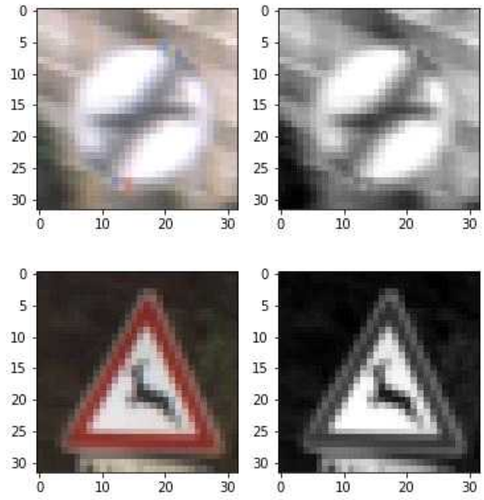
step 2 : Explore

- Read the pre-prepared “signname.csv” file and save the label name to the labels_set to match the index.
- Define the "explore" function to bring a few sample images and match them to the label names stored in the "labs_set" to make them visual.

[Image 2. Explore_train_images]

step 3 : Pre-process the Data Set

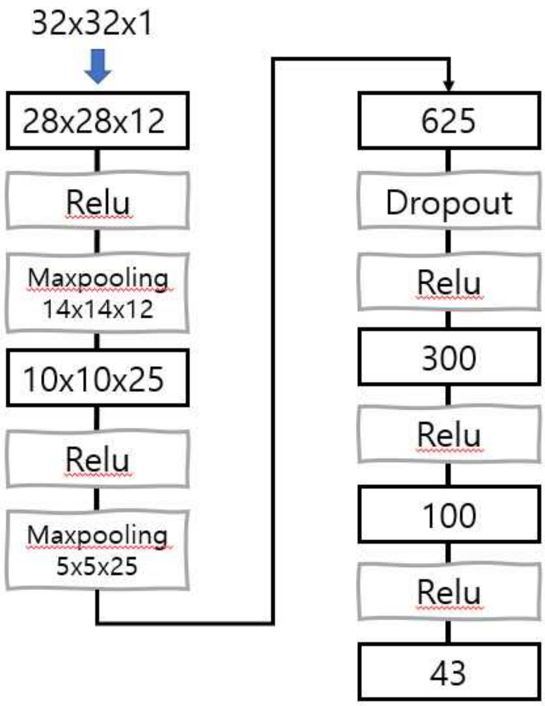
- Preprocess the data here. Data needs to be normalized. It also converts the image to gray scale and resizes it to fit the input size of the network.



[Image 3. Grayscale Image]

step 4 : Model architecture

- This step can be divided into three main steps
- the first function is conv() which determines the size stride of the filter and calculates Weight and bias.
- The second function creates a layer that classifies the class
- Finally, we design the structure of the network by defining the classifier function, which is the overall core structure.
- I designed a model that downsamples network structure in Layer 1 and Layer 2 with three layers and categorizes it in Layer 3.



step 5 : Train, Validate and Test the Model

- The train and validation code was written by storing only the weight of the highest model of 20 epoch.
- As shown in the figure below, the model was not stored afterwards because the highest accuracy was achieved at 10 width.

```
[epoch 1/20] == train_acc = 0.908 validation_acc = 0.837
[epoch 2/20] == train_acc = 0.969 validation_acc = 0.907
[epoch 3/20] == train_acc = 0.985 validation_acc = 0.923
[epoch 4/20] == train_acc = 0.988 validation_acc = 0.944
[epoch 5/20] == train_acc = 0.992 validation_acc = 0.934
[epoch 6/20] == train_acc = 0.988 validation_acc = 0.925
[epoch 7/20] == train_acc = 0.997 validation_acc = 0.939
[epoch 8/20] == train_acc = 0.996 validation_acc = 0.951
model saved into {8}_./Checkpoints/epoch_8_Checkpoint
[epoch 9/20] == train_acc = 0.995 validation_acc = 0.946
[epoch 10/20] == train_acc = 0.998 validation_acc = 0.959
model saved into {10}_./Checkpoints/epoch_10_Checkpoint
[epoch 11/20] == train_acc = 0.998 validation_acc = 0.948
[epoch 12/20] == train_acc = 0.998 validation_acc = 0.949
[epoch 13/20] == train_acc = 0.998 validation_acc = 0.944
[epoch 14/20] == train_acc = 0.999 validation_acc = 0.947
[epoch 15/20] == train_acc = 0.999 validation_acc = 0.949
[epoch 16/20] == train_acc = 0.998 validation_acc = 0.951
[epoch 17/20] == train_acc = 0.999 validation_acc = 0.948
[epoch 18/20] == train_acc = 0.999 validation_acc = 0.944
[epoch 19/20] == train_acc = 0.999 validation_acc = 0.953
[epoch 20/20] == train_acc = 0.999 validation_acc = 0.956
```

- Test dataset evaluation results

```
INFO:tensorflow:Restoring parameters from ./Checkpoints
accuracy in test set:  0.944101346002
```

step 6 : Test a Model on New Image

- The model was able to correctly guess 2 of the 5 traffic signs, which gives an accuracy of 40%. The accuracy on the captured images is 40% while it was 93.3% on the testing set thus It seems the model is overfitting and can further be tuned to better fit.

step 7 : Analyze the softmax probabilities of the new images