Traffic Sign Classsification

Gadde Venkata Sai Kumar

June 2017

1 Introduction

Traffic sign classification is very important for self driving cars, as the traffic signs play a very important role in the path planning. Self driving car is about understanding the local environment and planning the movements accordingly. Path planning like identifying the shortest path between the two destinations is a higher level problem for the car. So, to understand the local environment, the car has to understand the traffic signs.

Use cases for the traffic signs:

- When there is some work happening on the road, then the car has to identify it and need to take an alternative route. This real time information may not be updated into the google maps or any other navigation systems available for the vehicle.

2 Dataset Summary.

The dataset has imbalance of classes. The dataset is skewed. One of the important approach which can be taken when the dataset is skewed is to balance the classes. When the data is not about the images, class weights are given so that a particular sample is weighed more or considered multiple times by the factor of weight while training the machine learning model. But in case of images, image translation, image rotation and image affine can be done to augment the dataset.
Note- Visualization of the dataset: The visualization of the dataset is included in the python notebook.

2.1 Preprocessing techniques:

- The preprocessing techniques make the model more robust.
- ***Translation:*** The image can be moved to left, right, up or down so that the traffic sign remains the same, but in a different position.

```
def method_to_translate (X,y,limit=100000):
    rows,cols, _ = X[0].shape
    j=0
    X2 = np.empty([len(X),32,32,1])
    y2 = np.empty([len(X)])
    for c in range(43):
        x = X[np.where(y==c)]
        x_array = np.empty([ len(x) ,32, 32 , 1])
        y_array = np.empty([len(x)])
        if len(x)<limit:
            for i in range(len(x)):
                M_translation = np.float32([[1,0,np.random.uniform(-5,5)],[0,1,np
                dst_translation = warpAffine(x[i,:,:,:], M_translation, (cols,row
                dst_translation = np.reshape(dst_translation, (1,32,32,1))
                x_array[i] = dst_translation
```

```
            y_array[i] = c
        X2[j:j+len(x),:,:,:] = x_array
        y2[j:j+len(x)]   = y_array
        j += len(x)
return X2[:j,:,:,:],y2[:j]
```

- ***Rotation:*** It is not sure that the image is always upright when viewed by the camera. The image can be tilted to the right or left when viewed from different angle positions. The method implemented is in the python notebook.
- ***Affine:*** This scenario occurs when the image is captured from a different angle. The car has to identify the traffic sign from different angles and also from different distances. The method implemented is in the notebook.

3 Model Architecture:

A basic LeNet architecture is chosen for the classification. To improve the accuracy of the model, an interesting idea is adopted, which is residual networks. Neurons from two hidden layers are flattened and fully connected to a hidden layer. A significant increase in accuracy was observed. The idea was taken from the Microsoft ResNets. Even though this model is not as deep as the Microsoft Res Nets, the approach has resulted in better accuracy of the model. Residual networks exhibit ensemble like behavior.

Characteristics:

- Number of Layers: 7 (3 Convolutional + 2 maxpool + 2 Fully Connected layers.)
- Layers:
  - input : 32 x 32 x 1 image as input
  - Layer1: 28 x 28 x 6 (Convolutional layer)
  - Layer2: 14 x 14 x 6 (Maxpool)
  - Layer3: 10 x 10 x 16 (Convolutional layer)
  - Layer4: 5 x 5 x 16 (Maxpool)
  - Layer5: 1 x 1 x 400 (Convolution)
  - Layer6: 800 (Fully Connected Layer. Activations from Layer 5 and Layer 4 are combined in this layer)
  - Layer7: 43 (Output Layer.)

4 Model Training:

The model is trained on my Macbook pro without any GPU.

- Epochs: 80
- Batch size: 128
- Optimizer used: AdamOptimizer
- Learning Rate:0.001

5 Solution Approach:

The basic LeNet was used initially and an accuracy of around 0.89 was attained. Data-set Augmentation was done which resulted in a better accuracy around 0.92. Then A residual net was used which resulted in an accuracy of 0.94. Rather than improving the efficiency by using more data and more computation, it could be achieved by using a Residual nets concept. One way to implement this is to do data augmentation. The pros include, increase in the amount of data which results in a better model. The other approach is through using Residual Learning. Similar to the Residual Networks, the activations from two layers in the LeNet are combined and connected to the final fully connected layer. This resulted in improvement of accuracy significantly.

Advantages of Residual Networks The benefit of residual nets is during the back prop. This may eliminate the Vanishing Gradient problem.

6 Acquiring new Images:

Five images are downloaded from the internet. They are converted into grayscale and are resized into 32 x32 images and are given as inputs to the Deep learning classification model. The images and the performance is present in the python notebook submitted.
The major problem observed with the new images is the resolution of the input images. As the resolution of the input images is high and decreasing them using opencv has led to the loss in the information present in the image. Pixelation might be the reason behind low performance of the trained model.