Traffic Sign Recognition

Build a Traffic Sign Recognition Project

The goals / steps of this project are the following:

- Load the data set (see below for links to the project data set)
- Explore, summarize and visualize the data set
- Design, train and test a model architecture modified from LeNET refer to Traffic Sign Recognition with Multi-Scale Convolutional Networks Pierre Sermanet and Yann LeCun Courant Institute of Mathematical Sciences, New York University.
- Use the model to make predictions on new images
- Analyze the softmax probabilities of the new images
- Summarize the results with a written report

Dataset Exploration

Dataset Summary

Question 1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. You can use this template as a guide for writing the report. The submission includes the project code.

Answer 1 : I used the numpy library, and pickle.load() function to open and read the files for training and test. And then I use .shape()function and numpy to calculate summary statistics of the traffic signs data set. Data's specification is as below.

- The size of training set is 34799
- The size of test set is 12630
- The shape of a traffic sign image is (32,32,3)
- The number of unique classes/labels in the data set is 43

Exploratory Visualization

Question 2. include an exploratory visualization of the dataset.

Answer 2: Based on the matplotlib library, I generated row 2, colume 5 pictures randomly. It was for confirming the label and figure match to each other. (fig1)

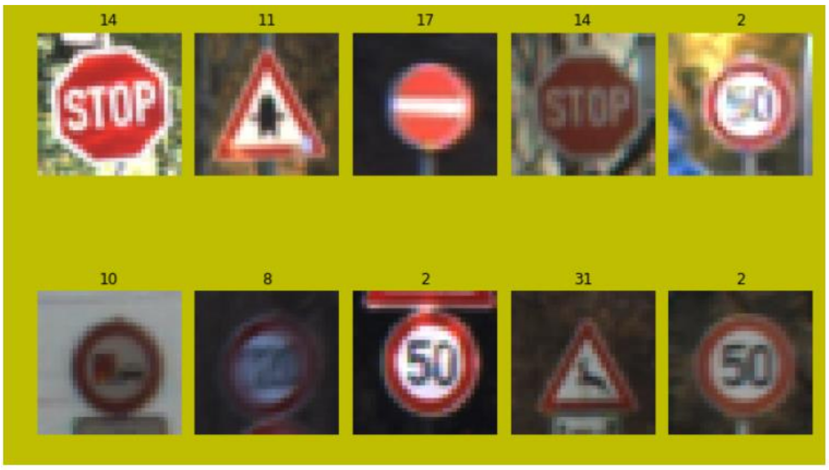


Fig 1. label and figure match

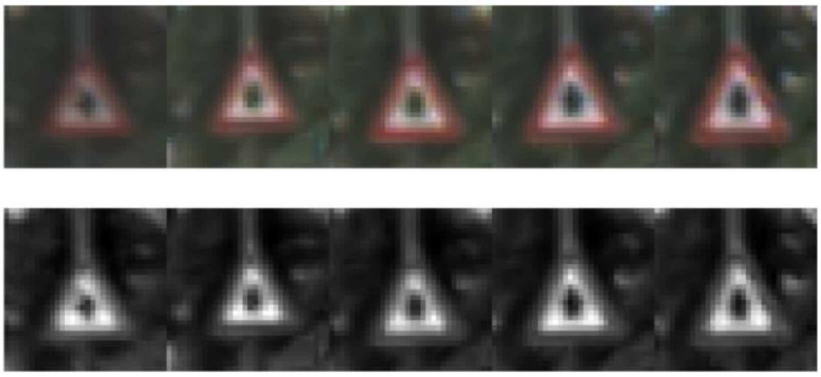


Fig 2. Color to grayscale

Figure below is the frequency histogram of the y_train parameter. This was used for confirm the frequency of the each labels.

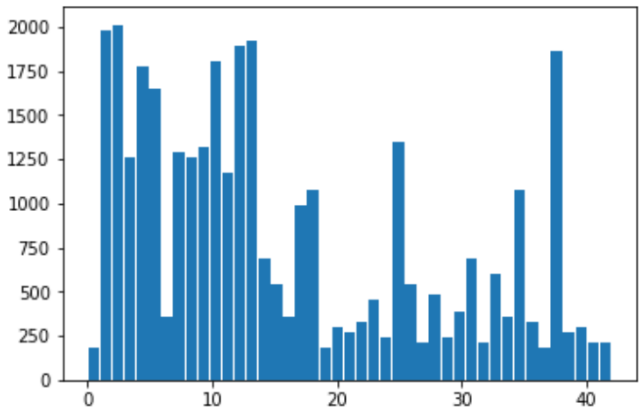


Fig 3. Frequency histogram of the y_train

Design and Test a Model Architecture

Question 3. Preprocessing data

Answer 3: preprocessing methods used. Because the origin training data is uniform and not pre-processed. So preprocessing images was needed. So the below methods were used to make the training data become more complex and not uniform.

- Image rgb type change to grayscale type
- Normalize the images to -1 and 1, this is for decrease the brightness and darkness of the images
- random_translate()
- random_scalling()
- random_wrap()
- random_brightness()
- image generate by rotate some angle.
    - But after doing this step, I found model became overfitting very easily. So I gave it up and delete. (still in my code)
- Shuffle the data

Model Architecture

The training model was made with lenet structure and modified refer to the paper(Traffic Sign Recognition with Multi-Scale Convolutional Networks, Pierre Sermanet and Yann LeCun Courant Institute of Mathematical Sciences, New York University)

Pieree's paper presented this method to training a model for traffic recognition. The architecture of the model is showed as figure 4. It has 2 maxpoling layers , 3 convolution layers and 3 steps of flatten layers. They considered the a number of difficult challenges due to real world variabilities such as viewpoint variations, lighting condition(saturations, low-contrast),motion-blur, occlusions, sun glare, physical damage, colors fading, graffiti, stickers and an input resolution as low as 15x15.

The traditional ConvNet architecture was modified by feeding 1st stage features in addition to 2nd stage features to the classifier. And their experiments conducted after phase 1 produced a new record of 99.17%. This good enough to train a model for traffic sign. What's more, it has 97.33% accuracy when feed with random features. This paper provided a good solution to architecture of model.

The model that I trained first time was general LeNET5 studied from udacity. However, it performance was not good enough. Training step couldn't get high accuracy and test accuracy was only 80%~90%. After used Pieree's model , the accuracy of validation step reached almost 99.4% and the test accuracy reached 94% . I thought overfitting still happened but the results was good enough to meet the specification.

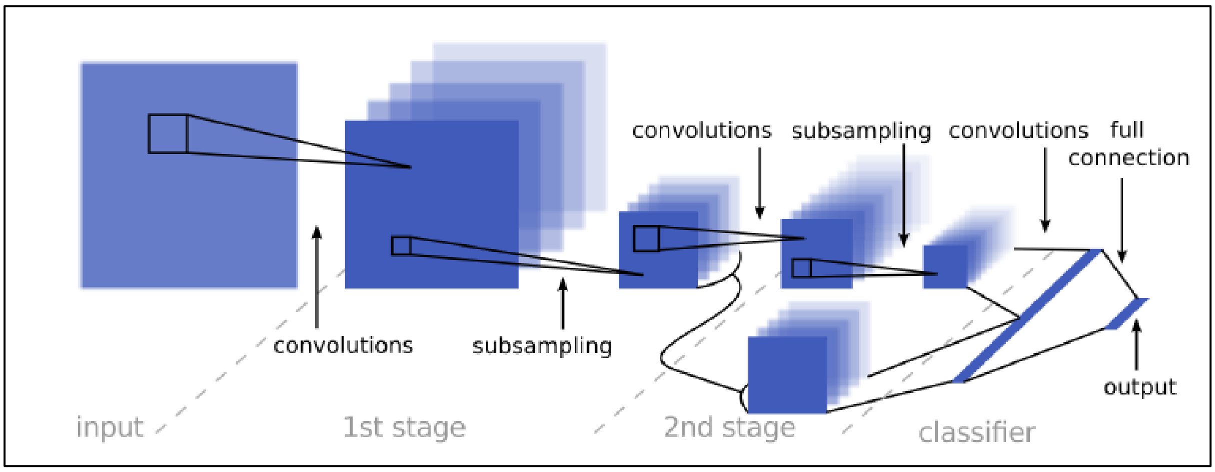


Fig 4. Architecture of multi-scale convolutional networks

- 1st layer : Input : 32x32x1, output: 28x28x6, valid padding, stride [1,1], filter size[5,5]
    - Almost same with LeNet
- Relu layer
- Max_pooling layer, stride [2,2], input: 28x28x6, output : 14x14x6
- 2nd layer: Input :14x14x6 , output : 10x10x16, filter size: [5,5], valid padding, stride[1,1]
- Relu layer
- Max_pooling layer, stride[2,2], input: 10x10x16, output: 5x5x16, filter size: [2,2], stride[2,2]
- Flatten layer(for 2nd stage which is direct to full connection), size : 400

- Convolution layer(2^{nd} stage subsampling)
- Flatten layer(for full connection), size : 400
- Concatenate layer of full connection , 400+400= 800
- Full connection to logis output . input: 800 output: 43

Model Training

The epochs, keep_prob, learning rate and batch size was decided by many training session. The by calculate the cross_entropy of each output, the loss_operation constructed. I use adamoptimizer as many other students did since it is the pretty good choice to do this project. And then decreased the loss function was executed by tensorflow.

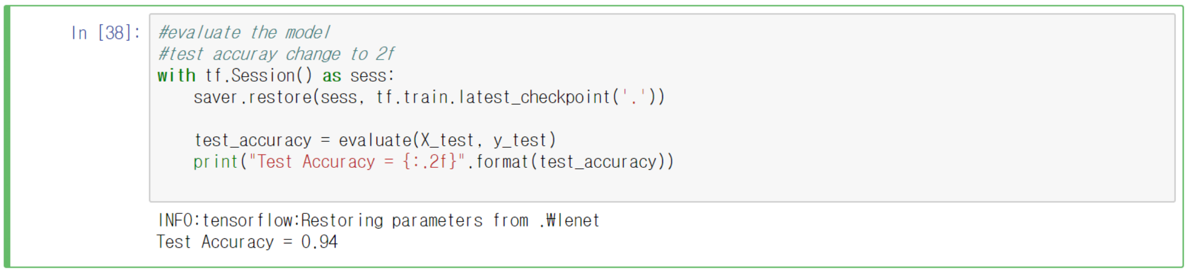
Batch_size : 100~110 (similar results)

Epochs : 60~65 (similar results)

Learning_rate : 0.0009 (it was very sensitive. Once if the value change into 0.001 or 0.0001, the model's performance changed a lot)

Solution Approach

After training, the model's accuracy was near 93%~94%.



```
In [38]: #evaluate the model
         #test accuray change to 2f
         with tf.Session() as sess:
             saver.restore(sess, tf.train.latest_checkpoint('.'))

             test_accuracy = evaluate(X_test, y_test)
             print("Test Accuracy = {:.2f}".format(test_accuracy))

         INFO:tensorflow:Restoring parameters from .₩lenet
         Test Accuracy = 0.94
```

Accuracy : 94%

Fig 5. Accuracy test

Test a Model on New Images

Acquiring New Images

I found some images for test in google by searched " german traffic signs". There were many images. I choose some clear images and can be exactly labeled. So I found images like below.

Some of them are with blue sky and some of them are totally a standard without any background. And some of them have green background. There are not only contain the important part of traffic sign but also contain disturbance to image recognition.

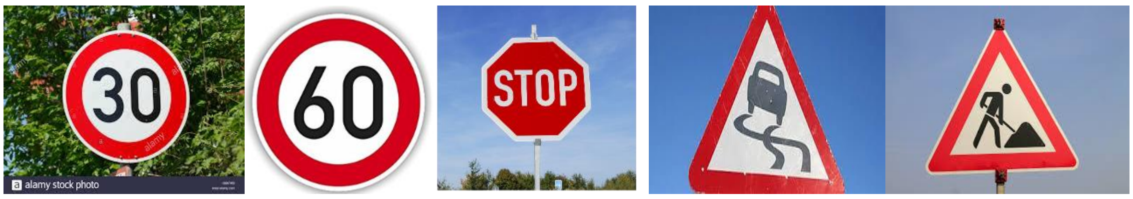


Fig. 6 German traffic images found in google

Performance on New Images

To confirm my model's performance with new images from download, I tested the model that I built. Of course the images were pre-processed by some functions. The labels of my download images [1,3,14,23,25]. However, accuracy of the trained model was only 60%. I think this happened because my model was overfitting or couldn't catch the features well.

Model Certainty - Softmax Probabilities

By using the tensorflow.nn.top_k(softmax_logits)function, I found the trained model's prediction was totally wrong. The 'stop' and 'slippy' labels prediction was totally wrong. So the model only get 60% accuracy. Especially the 'slippy' was 100% wrong. It seems like model needs more training to catch the features of images.

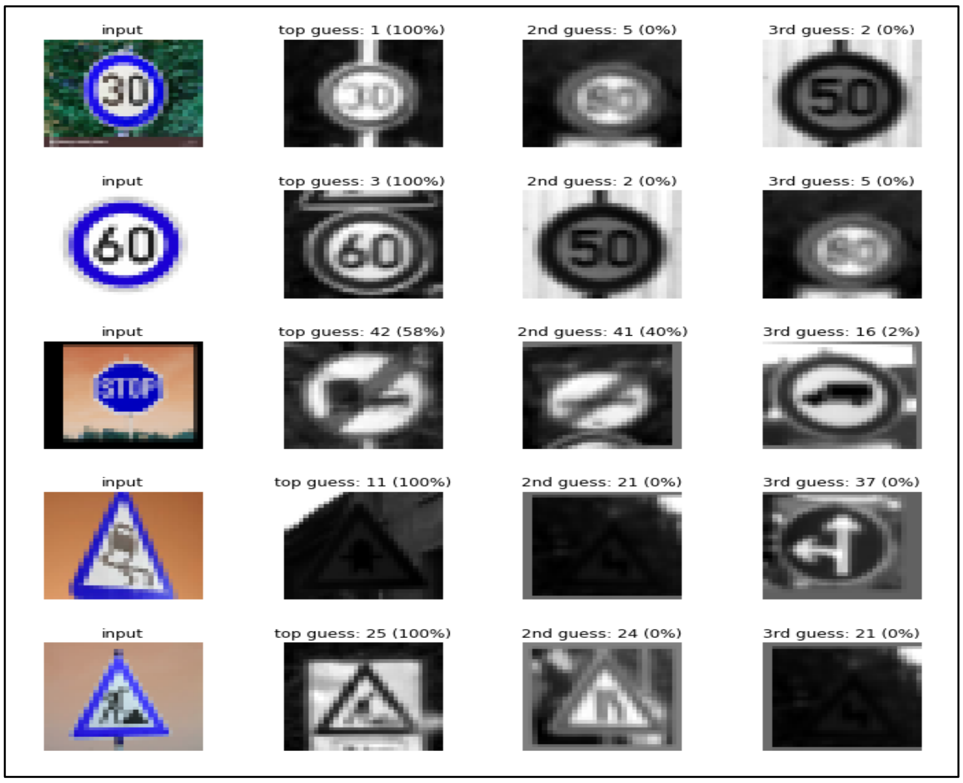


Fig 7. Top_k softmax

References

[1] Traffic Sign Recognition with Multi-Scale Convolutional Networks, Pierre Sermanet and Yann LeCun, Courant Institute of Mathematical Sciences, New York University

[2] jeremy-shannon, https://github.com/jeremy-shannon

[3] Param Aggarwal, https://medium.com/computer-car/intricacies-of-traffic-sign-classification-with-tensorflow-8f994b1c8ba

[4] MehdiSv, https://github.com/MehdiSv

[5] vxy10, https://github.com/vxy10