Abstract

This project shows how to classify german traffic signs using a modified LeNet neuronal network.

The 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
- 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

Submitted Files

1. Writeup of the summary of the submission
2. Jupyter notebook
3. HTML output of the notebook

Data Set summary and exploration

1. Dataset Summary

I used the numpy library to calculate summary statistics of the traffic signs data set:

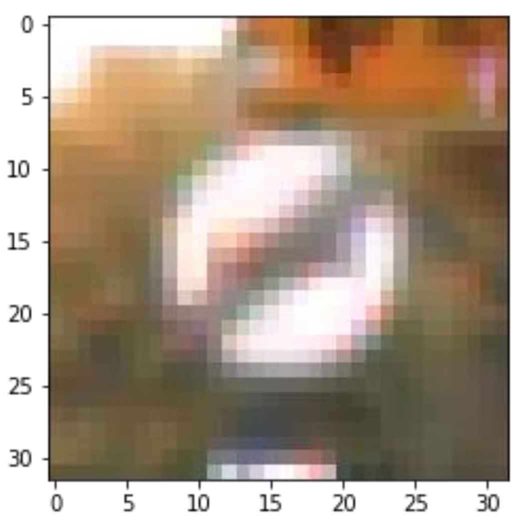
- The size of training set is 34799
- The size of the validation set is 4410
- The size of test set is 12630
- The shape of a traffic sign image is (32, 32, 3)
- The number of unique classes/labels in the data set is 43

2. Exploratory Visualization

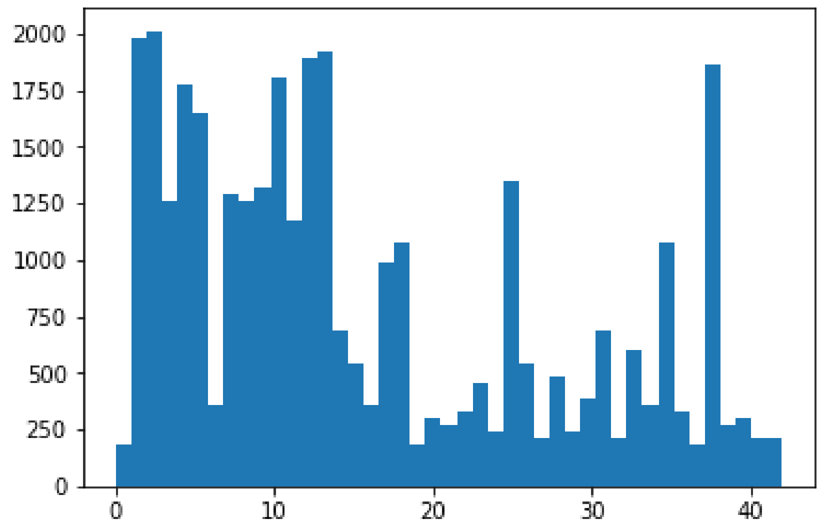
```
### Data exploration visualization code goes here.
### Feel free to use as many code cells as needed.
import matplotlib.pyplot as plt
# Visualizations will be shown in the notebook.
%matplotlib inline
plt.imshow(X_train[120,:])
plt.show()

plt.hist(y_train, bins = 43)
plt.show()
```

Sample output with label 41 – End of no passing



Histogram showing the frequency of each output in the training batch



Design and Test a Model Architecture

1. Preprocessing

The preprocessing step includes normalizing and shuffling the data

X_train = np.sum(X_train/3, axis=3, keepdims=True)
X_test = np.sum(X_test/3, axis=3, keepdims=True)
X_valid = np.sum(X_valid/3, axis=3, keepdims=True)
X_train = (X_train - 128)/128
X_test = (X_test - 128)/128
X_valid = (X_valid - 128)/128

from sklearn.utils import shuffle
X_train, y_train = shuffle(X_train, y_train)

2. Model Architecture

The model architecture used is the same LeNet architecture as described in the lecture. Further, I've added a dropout layer after the first fully connected layer to reduce over fitting and improve the validation accuracy.

```
### Define your architecture here.

from tensorflow.contrib.layers import flatten

def LeNet(x):
    # Arguments used for tf.truncated_normal, randomly defines variables for the weights and biases
for each layer
    mu = 0
    sigma = 0.1

    # SOLUTION: Layer 1: Convolutional. Input = 32x32x1. Output = 28x28x6.
    conv1_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 1, 6), mean = mu, stddev = sigma))
    conv1_b = tf.Variable(tf.zeros(6))
    conv1   = tf.nn.conv2d(x, conv1_W, strides=[1, 1, 1, 1], padding='VALID') + conv1_b

    # SOLUTION: Activation.
    conv1 = tf.nn.relu(conv1)

    # SOLUTION: Pooling. Input = 28x28x6. Output = 14x14x6.
    conv1 = tf.nn.max_pool(conv1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID')

    # SOLUTION: Layer 2: Convolutional. Output = 10x10x16.
    conv2_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 6, 16), mean = mu, stddev = sigma))
    conv2_b = tf.Variable(tf.zeros(16))
    conv2   = tf.nn.conv2d(conv1, conv2_W, strides=[1, 1, 1, 1], padding='VALID') + conv2_b

    # SOLUTION: Activation.
    conv2 = tf.nn.relu(conv2)

    # SOLUTION: Pooling. Input = 10x10x16. Output = 5x5x16.
    conv2 = tf.nn.max_pool(conv2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID')

    # SOLUTION: Flatten. Input = 5x5x16. Output = 400.
    fc0   = flatten(conv2)

    # SOLUTION: Layer 3: Fully Connected. Input = 400. Output = 120.
    fc1_W = tf.Variable(tf.truncated_normal(shape=(400, 120), mean = mu, stddev = sigma))
    fc1_b = tf.Variable(tf.zeros(120))
    fc1   = tf.matmul(fc0, fc1_W) + fc1_b

    # SOLUTION: Activation.
    fc1    = tf.nn.relu(fc1)

    #implementing a dropout
```

```
fc1 = tf.nn.dropout(fc1, keep_prob)

# SOLUTION: Layer 4: Fully Connected. Input = 120. Output = 100.
fc2_W  = tf.Variable(tf.truncated_normal(shape=(120, 100), mean = mu, stddev = sigma))
fc2_b  = tf.Variable(tf.zeros(100))
fc2    = tf.matmul(fc1, fc2_W) + fc2_b

# SOLUTION: Activation.
fc2    = tf.nn.relu(fc2)

#implementing another dropout

#fc2 = tf.nn.dropout(fc2, keep_prob)


# SOLUTION: Layer 5: Fully Connected. Input = 100. Output = 43.
fc3_W  = tf.Variable(tf.truncated_normal(shape=(100, 43), mean = mu, stddev = sigma))
fc3_b  = tf.Variable(tf.zeros(43))
logits = tf.matmul(fc2, fc3_W) + fc3_b

return logits
```

3. Model Training

To train the model, I used an Adam optimizer and the following hyperparameters:

- batch size: 128
- number of epochs: 50
- learning rate: 0.0009
- Variables were initialized using the truncated normal distribution with mu = 0.0 and sigma = 0.1
- keep probalbility of the dropout layer: 0.5

My final model results were:

- validation set accuracy of 95.6%
- test set accuracy of 93.7%

4. Solution Approach

To obtain the validation accuracy of better than 93%, I used the same LeNet architecture but with a dropout layer of probability 0.5. With the introduction of the dropout layer, I increased the number of epochs to 50 with a batch size of 128.

The introduction of the dropout layer slowed learning in the initial few epocs, but with further training, I was able to achieve validation accuracy of 95.6% with a test accuracy of 93.7%.

Test a Model on New Images

1. Acquiring new images

New German traffic signal images were downloaded from the internet to test the system performance. 8 images were downloaded with the respective label as per the csv file provided in the repository.

The images were preprocessed as per the training set.

```
import cv2
import glob
import matplotlib.pyplot as plt
import numpy as np
images = glob.glob('./test_image/*.JPG')

my_images = []

for fname in images:
    img = cv2.imread(fname)
    img = cv2.resize(img, (32,32))
    my_images.append(img)

my_images = np.asarray(my_images)
my_images = np.sum(my_images/3, axis=3, keepdims=True)
my_images = (my_images - 128)/128
```

Sample image before preprocessing –

Stop Sign – with label 14

2. Performance on new images

The model was able to identify all of the 8 images correctly with an accuracy of 100%.

3. Model Certainty - Softmax Probabilities

Softmax probability for the new images were derived. The model was able to determine the correct label with a very high percentage of accuracy (around 98% and more)

Possible future work

1. Data augmentation for improved training and performance
2. Implementation of deeper and more complex models such as VGG, AlexNet, ResNet etc.
3. Layer visualization to study the training architecture and further improve performance.