Self-Driving Car Engineer Nanodegree

Deep Learning

Project: Build a Traffic Sign Recognition Classifier

Submitted By: Ninad Ghike

Writeup / READMe

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.

Please find the link to my code: Project Code

Data Set Summary & Exploration

1. Provide a basic summary of the data set and identify where in your code the summary was done. In the code, the analysis should be done using python, numpy and/or pandas methods rather than hardcoding results manually.

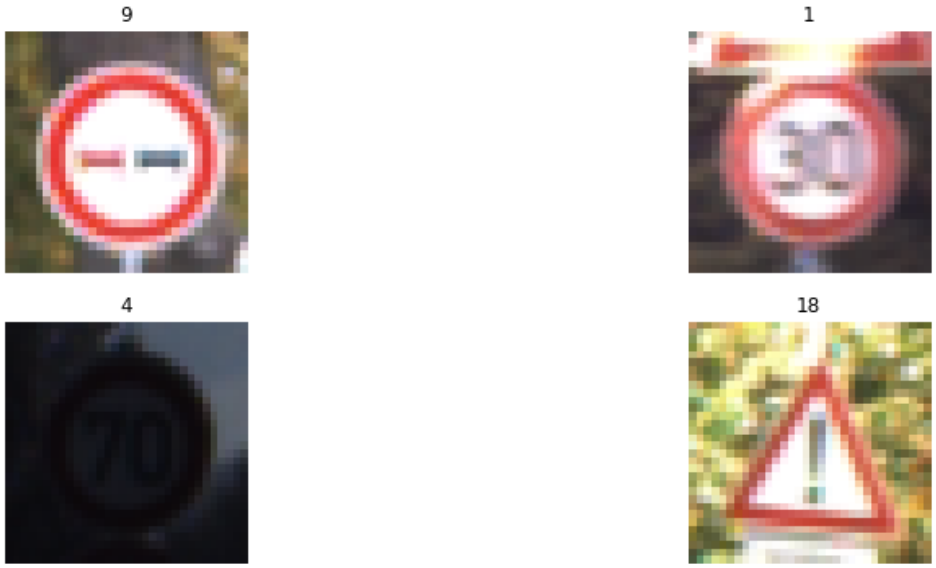
Numpy was used to calculate the summary statistics of the traffic signs data set.

```
# TODO: Number of training examples
n_train = y_train.shape[0]

# TODO: Number of testing examples.
n_test = y_test.shape[0]

# TODO: Number of validation examples.
n_valid = len(X_validation)


# TODO: What's the shape of an traffic sign image?
image_shape = X_train[0].shape

# TODO: How many unique classes/labels there are in the dataset.
n_classes = np.unique(y_train).shape[0]
```

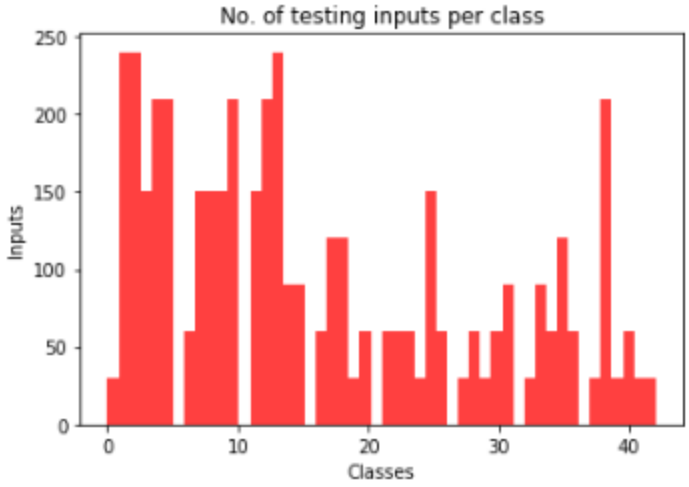
```
Number of training examples = 34799
Number of testing examples = 12630
Number of validation examples = 4410
Image data shape = (32, 32, 3)
Number of classes = 43
```

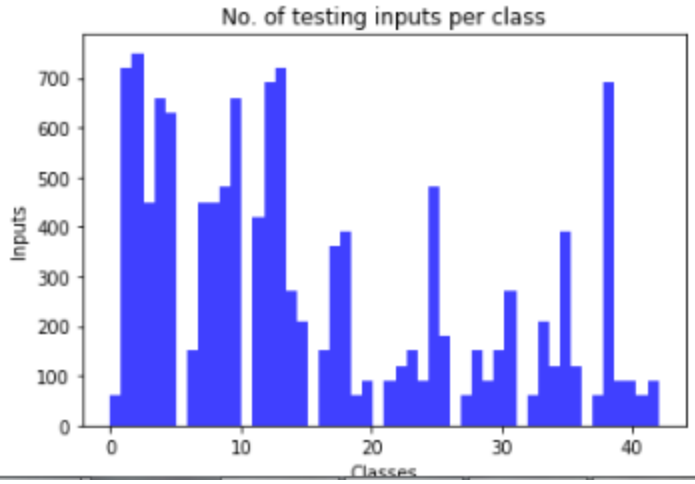
2. Include an exploratory visualization of the dataset and identify where the code is in your code file.

```
### Data exploration visualization goes here.
import matplotlib.pyplot as plt
import random
# Visualizations will be shown in the notebook.
%matplotlib inline

# show image of 10 random data points
fig, axs = plt.subplots(2,2, figsize=(15, 6))
fig.subplots_adjust(hspace = .2, wspace=.001)
axs = axs.ravel()
for i in range(4):
    index = random.randint(0, len(X_train))
    image = X_train[index]
    axs[i].axis('off')
    axs[i].imshow(image)
    axs[i].set_title(y_train[index])
```




```
#set up the plot and show the training data as histogram. Repeat for other
plt.figure(1)
n, bins, patches = plt.hist(y_train, 50, facecolor = 'green', alpha = 0.75
)
plt.xlabel('Classes')
plt.ylabel('Inputs')
plt.title('No. of Training inputs per class')
plt.show()
```





Correction: The Blue Graph indicates validation Dataset

Design and Test a Model Architecture

1. Describe how, and identify where in your code, you preprocessed the image data. What tecniques were chosen and why did you choose these techniques? Consider including images showing the output of each preprocessing technique. Pre-processing refers to techniques such as converting to grayscale, normalization, etc.

Pre – Processing

I used two different pre-processing techniques. Each provided and increase in accuracy as compared to un-pre-processed Images.

Technique 1 – Mean Equalization:

```
def preprocess(image):
    img = np.copy(image)
    img[:,:,0] = cv2.equalizeHist(img[:,:,0])
    img[:,:,1] = cv2.equalizeHist(img[:,:,1])
    img[:,:,2] = cv2.equalizeHist(img[:,:,2])
    img = img/255.-.5
```

Technique 2 – Normalization

```
def pre_pro(image):
    image = image/127.5-1.
    return image
```

Technique 3 – mean substraction

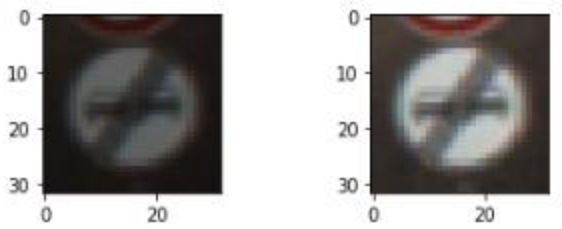
```
X_train = (X_train - X_train.mean()) / (np.max(X_train) - np.min(X_train))
X_test = (X_test - X_test.mean()) / (np.max(X_test) - np.min(X_test))
X_validation = (X_validation - X_validation.mean()) / (np.max(X_validation
) - np.min(X_validation))
```

In technique 2, the data was normalized between -1 and +1. This significantly improved the accuracy. The weights in the neural network are typically between -1 and +1. Therefore normalizing the input data to that range allowed the network to converge with better accuracy.

The idea of technique 1 and 3 was to center the data to 0. However this reduced the accuracy and hence was not used in the submitted code.

Grayscale was not used since color information plays a vital role in the recognition of signboards.

```
Preprocessing
Done preprocessing
```



2. Describe how, and identify where in your code, you set up training, validation and testing data. How much data was in each set? Explain what techniques were used to split the data into these sets. (OPTIONAL: As described in the "Stand Out Suggestions" part of the rubric, if you generated additional data for training, describe why you decided to generate additional data, how you generated the data, identify where in your code, and provide example images of the additional data)

Training, Validation and Testing data was loaded from the pickle files.

```
training_file = 'train.p'
validation_file = 'valid.p'
testing_file = 'test.p'

with open(training_file, mode='rb') as f:
    train = pickle.load(f)
with open(validation_file, mode='rb') as f:
    valid = pickle.load(f)
with open(testing_file, mode='rb') as f:
    test = pickle.load(f)

X_train, y_train = train['features'], train['labels']
X_valid, y_valid = valid['features'], valid['labels']
X_test, y_test = test['features'], test['labels']
```

3. Describe, and identify where in your code, what your final model architecture looks like including model type, layers, layer sizes, connectivity, etc.) Consider including a diagram and/or table describing the final model.

I tried using multiple architectures. Some of them are

1. 3-layer Convolutional Neural Network [Ref]
2. Le-Net without and with pre-processed Data
3. Le-Net with modified architecture [Ref]

Modified Le-Net seemed to provide better results as had many more layers introduces for feature extraction. But this approach was cost intensive as it took forever to train even for 10 epocs. Le-Net with pre processed data seemed to perform better than 3-layer Convolutional Neural Network

The architecture summary is as follows:

Layer	Description
Input	32x32x3 RGB image
Convolution 5x5	1x1 stride, valid padding, outputs 28x28x6
RELU	
Max pooling	2x2 stride, outputs 14x14x6
Convolution 5x5	1x1 stride, valid padding, outputs 10x10x16
RELU	
Max Pooling	2x2 stride, outputs 5x5x16
Flatten	output 400
Fully Connected	output 120
RELU	
Fully Connected	output 84
RELU	
Fully Connected	output 43

```
def LeNet(x):
    # Hyperparameters
    mu = 0
    sigma = 0.1

    # SOLUTION: Layer 1: Convolutional. Input = 32x32x1. Output = 28x28x6.
    conv1_W = tf.Variable(tf.truncated_normal(shape=(5, 5, 3, 6), mean = m
u, stddev = sigma))
    conv1_b = tf.Variable(tf.zeros(6))
    conv1   = tf.nn.conv2d(x, conv1_W, strides=[1, 1, 1, 1], padding='VALI
D') + 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, s
tddev = sigma))
    fc1_b = tf.Variable(tf.zeros(120))
    fc1   = tf.matmul(fc0, fc1_W) + fc1_b

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

    # SOLUTION: Layer 4: Fully Connected. Input = 120. Output = 84.
    fc2_W  = tf.Variable(tf.truncated_normal(shape=(120, 84), mean = mu, s
tddev = sigma))
    fc2_b  = tf.Variable(tf.zeros(84))
    fc2    = tf.matmul(fc1, fc2_W) + fc2_b

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

```
    # SOLUTION: Layer 5: Fully Connected. Input = 84. Output = 10.
    fc3_W  = tf.Variable(tf.truncated_normal(shape=(84, 43), mean = mu, st
ddev = sigma))
    fc3_b  = tf.Variable(tf.zeros(43))
    logits = tf.matmul(fc2, fc3_W) + fc3_b

    print(logits.get_shape().as_list())

    return logits
```

4. Describe how, and identify where in your code, you trained your model. The discussion can include the type of optimizer, the batch size, number of epochs and any hyperparameters such as learning rate.

```
### TRAINING PIPELINE
rate = 0.001
EPOCHS = 10
BATCH_SIZE = 128

logits = LeNet(x)
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits, one_hot_y)
loss_operation = tf.reduce_mean(cross_entropy)
optimizer = tf.train.AdamOptimizer(learning_rate = rate)
training_operation = optimizer.minimize(loss_operation)
```

The model was trained at an initial low rate of 0.001. This ensured a slow but stable learning. Also, the Adam Optimizer controls the rate. 10 epochs were made but accuracy did not improved after that. A batch size of 128 was optimal considering the memory size of the GPU used.

An Adam optimizer was used since it adaptively computes the learning rate.

5. Describe the approach taken for finding a solution. Include in the discussion the results on the training, validation and test sets and where in the code these were calculated. Your approach may have been an iterative process, in which case, outline the steps you took to get to the final solution and why you chose those steps. Perhaps your solution involved an already well known implementation or architecture. In this case, discuss why you think the architecture is suitable for the current problem.

The final model results were as follows:

Dataset	Accuracy (%)
Validation	94.3
Testing	93.6

The LeNet architecture was used. This architecture was chosen since it had worked successfully in the classification of numbers. It was interesting to see how an existing architecture can be modified to solve a problem which was similat but not the same as the original. In this case, the original problem had graysclae images and 10 output numbers. In this project the images used were color with 43 categories.

```
### TRAIN THE MODEL
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    num_examples = len(X_train)

    print("Training...")
    print()
    for i in range(EPOCHS):
        X_train, y_train = shuffle(X_train, y_train)
        for offset in range(0, num_examples, BATCH_SIZE):
            end = offset + BATCH_SIZE
            batch_x, batch_y = X_train[offset:end], y_train[offset:end]
            sess.run(training_operation, feed_dict={x: batch_x, y: batch_y
})

        validation_accuracy = evaluate(X_validation, y_validation)
        print("EPOCH {} ...".format(i+1))
        print("Validation Accuracy = {:.3f}".format(validation_accuracy))
        print()

    saver.save(sess, './lenet')
    print("Model saved")
```

The Validation accuracy was stuck at 89% for a very long time. After making multiple changes in the architecture and preprocessing it was found that the issue was with the number of epochs. 10 epochs was too less for the netwrok to converge. After increasing the epochs to 100, the validation accuracy went higher than 93%. But one thing I observed was that the accuracy actually started decreasing after around 70 epocs before it went back up. This seemed to me as a case of over fitting and hence I reduced the number of Epochs to 65.

```
Training...
```

```
EPOCH 1 ...
Validation Accuracy = 0.756

EPOCH 2 ...
Validation Accuracy = 0.859

EPOCH 3 ...
Validation Accuracy = 0.891

EPOCH 4 ...
Validation Accuracy = 0.885

EPOCH 5 ...
Validation Accuracy = 0.888

EPOCH 6 ...
Validation Accuracy = 0.885

EPOCH 7 ...
Validation Accuracy = 0.898

EPOCH 8 ...
.
.
.
.
.
.
.
.
EPOCH 60 ...
Validation Accuracy = 0.946

EPOCH 61 ...
Validation Accuracy = 0.944

EPOCH 62 ...
Validation Accuracy = 0.945

EPOCH 63 ...
Validation Accuracy = 0.944

EPOCH 64 ...
Validation Accuracy = 0.943

EPOCH 65 ...
Validation Accuracy = 0.943
```

I ran the model on Test Data and got accuracy above 93%

```
with tf.Session() as sess:
    saver.restore(sess, tf.train.latest_checkpoint('.'))

    test_accuracy = evaluate(X_test, y_test)
```

```
    print("Test Accuracy = {:.3f}".format(test_accuracy))
```

```
Test Accuracy = 0.936
```

I used MaxPooling to reduce the size of the input and also to prevent overfitting.

Test a Model on New Images

1. Choose five German traffic signs found on the web and provide them in the report. For each image, discuss what quality or qualities might be difficult to classify.

I decided to use following five signs I found on https://assets.publishing.service.gov.uk/media/58170307ed915d61c5000000/the-highway-code-traffic-signs.pdf. These images have no background in them thus reducing the noise factor.

Stop Sign is fairly easy to classify considering the shape and color.

The above two Images are a bit complex to classify as they have the same shape. If I would have converted them to gray scale, I would have lost the very important color information which these two signs provide. Also, the numbers look simillar

In the sign above for merge the line can be misinterpreted as a straight line and can cause error in classification.

These two signs above are same signs except that one is rotated form of other. This will cause an error in classification as my model did not account for the rotation. May be using rotated images during training would have helped it.

2. Discuss the model's predictions on these new traffic signs and compare the results to predicting on the test set. Identify where in your code predictions were made. At a minimum, discuss what the predictions were, the accuracy on these new predictions, and compare the accuracy to the accuracy on the test set (OPTIONAL: Discuss the results in more detail as described in the "Stand Out Suggestions" part of the rubric).

```
### Load the images and plot them here.

X_test_extra = []
y_test_extra = np.array([35,39,24,0,1,14])

extra = glob.glob('./test_images/Signs_no_Background/*.png')
for idx in range(6):
    img = cv2.cvtColor(cv2.imread(extra[idx]), cv2.COLOR_BGR2RGB)
    ## Make sure images are resized before appending
    img = cv2.resize(img,(32,32), interpolation = cv2.INTER_CUBIC)
    img = pre_pro(img)
    X_test_extra.append(img)

X_test_extra = np.asarray(X_test_extra)


### Run the predictions here and use the model to output the prediction fo
r each image.
### Make sure to pre-process the images with the same pre-processing pipel
ine used earlier.

with tf.Session() as sess:
    saver.restore(sess, tf.train.latest_checkpoint('.'))
    test_accuracy = evaluate(X_test_extra, y_test_extra)
```

```
    print("Test Accuracy = {:.3f}".format(test_accuracy))
    predict = sess.run(tf.nn.softmax(preds), feed_dict={x:X_test_extra})
    extra_top_k = sess.run(tf.nn.top_k(predict, 5))
    print(extra_top_k)



Test Accuracy = 0.500
TopKV2(values=array([[  1.00000000e+00,   1.10395457e-32,   0.00000000e+00
,
          0.00000000e+00,   0.00000000e+00],
       [  1.00000000e+00,   7.11453178e-22,   6.16860040e-25,
          1.90303391e-27,   3.39929813e-28],
       [  1.00000000e+00,   4.37592607e-26,   7.63698168e-27,
          1.41077491e-30,   1.87079720e-32],
       [  9.99432027e-01,   5.67785988e-04,   2.03307906e-07,
          8.08902506e-14,   1.06146860e-17],
       [  9.93793964e-01,   6.11506449e-03,   9.10192684e-05,
          9.41316469e-12,   7.82343686e-14],
       [  1.00000000e+00,   5.56322002e-11,   9.76419813e-12,
          5.37148278e-20,   8.36421203e-23]], dtype=float32), indices=arra
y([[35, 40,  0,  1,  2],
       [39, 37, 40,  2, 33],
       [18, 24, 27, 26, 11],
       [36,  1,  3, 35, 19],
       [18, 38, 34, 25, 37],
       [14,  1, 17, 25, 13]], dtype=int32))
```

Looking at the .csv file provided, we can summarize the predictions in table shown below:

Image	Prediction	Probability
Ahead Only (35)	35 - Ahead Only	100%
Keep Left (39)	39 - Keep Left	100%
Road Narrows on the Right (24)	18 - General Caution	100%
20 km/h (0)	36 - Child Crossing	99.94%
30 km/h (1)	18 - Turn Left Ahead	99.37%
Stop (14)	14 - Stop	100%

The model was able to correctly guess 3 of the 6 traffic signs, which gives an accuracy of 50%. The accuracy is lower than the test set since these are of different image quality. Even the colors are more saturated.

I also tried using the same model on images having some background noise in it.

The model seems to perform very poorly in that scenario. My submission has the result for these test Images.

I would like to take a moment and thank the reviewer for going through the project and my mentor Xuanzhe Wang.