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
- 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

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

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.

You're reading it! and here is a link to my project code

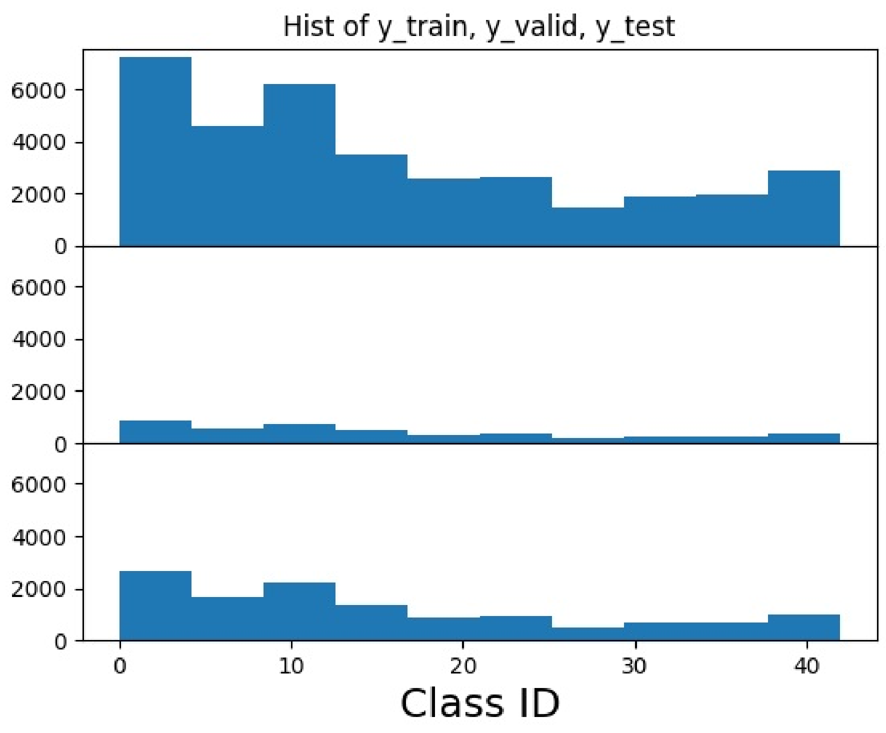
Data Set Summary & Exploration

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

I used the pandas 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 32x32x3
- The number of unique classes/labels in the data set is 43

2. Include an exploratory visualization of the dataset.



Here is an exploratory visualization of the data set. It is a histogram of train, validation, and test data set. From the figure below, we can see that

- all three datasets have similar distribution of traffic sign classes
- and the size of train dataset is the largest while validation dataset is the smallest
- from the distribution of the histogram, it also shows that the size of training images with class ID around 30 is smaller than others.

Because the size of images with class ID around 30 is smaller, our trained deep neural network performs worse with cooresponding test images.

Design and Test a Model Architecture

1. Describe how you preprocessed the image data. What techniques 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. (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, and provide example images of the additional data. Then describe the characteristics of the augmented training set like number of images in the set, number of images for each class, etc.)

First, I tried to use rgb color images because they contain more information than grayscale images(more model parameters to be trained as well). And it works and meets the validation rate of 93%. After a while, I found that colors were not important in recognizing traffic signs, as there are not different traffic signs with the same shape but different colors. It means that we can recognize traffic signs by their contour shapes. Therefore, I adoptted grayscale images in my final implementation. I rewrote a piece of code from stackoverflow to convert color images to grayscale:

```
def rgb2gray(rgb):
    '''
    convert one rgb image to grayscale image.
    Input: 3d numpy array
    output: 2d numpy array
    '''
    return np.dot(rgb[...,:3], [0.299, 0.587, 0.114])

def rgb2grayBatch(imgs):
    '''
    Convrt a batch of rgb images to grayscale images.
    Input: a batch of rgb images, 4d array
    Output: a batch of grayscale images, 4d array
    '''
    size_in = imgs.shape
    out = np.zeros((size_in[0], size_in[1], size_in[2]))
    for i in range(size_in[0]):
        out[i,:,:] = rgb2gray(np.squeeze(imgs[i,:,:]))

    return out[:,:,:,None]
```

Next, I subtracted each image by its mean pixel value such that images are shifted to the origin of 1024(32*32) dimensional space. When the input values are large, the neural network is more determined as the sigmoid function magnifies big values. Becase we want to train the DNN, it should be undetermined from the beginning. As a result, I normalized the image pixel value space to the section of [-1, 1]. The corresponding code is the following:

```
def preprocessImgsZeroMeanGray(gray_imgs):
    # rgb to graysacle
    size_in = gray_imgs.shape
    out = np.squeeze(gray_imgs)

    # zero-meaned
    means = np.mean(np.mean(out, axis=1), axis=1)
    out = out - means[:,None,None]

    # Get absolute values
    tmp_abs = np.absolute(out)
    # Get the max magnitude of pixels of each image
    tmp_max = np.amax(np.amax(tmp_abs, axis=1), axis=1)
    # If the image is pure black, use 1. Avoid 0/0.
    tmp_max = np.maximum(tmp_max, np.ones(tmp_max.shape))

    # Normalize the images
    out = out/tmp_max[:,None,None]

    return out[:,:,:,None]


def preprocessImgsZeroMean(imgs):
    '''imgs: rgb color images'''


    return preprocessImgsZeroMeanGray(rgb2grayBatch(imgs))
```

I decided to generate additional data because the test images of traffic sign could be rotated and noised.

To add more data to the the data set, I used the following techniques to rotate the training images:

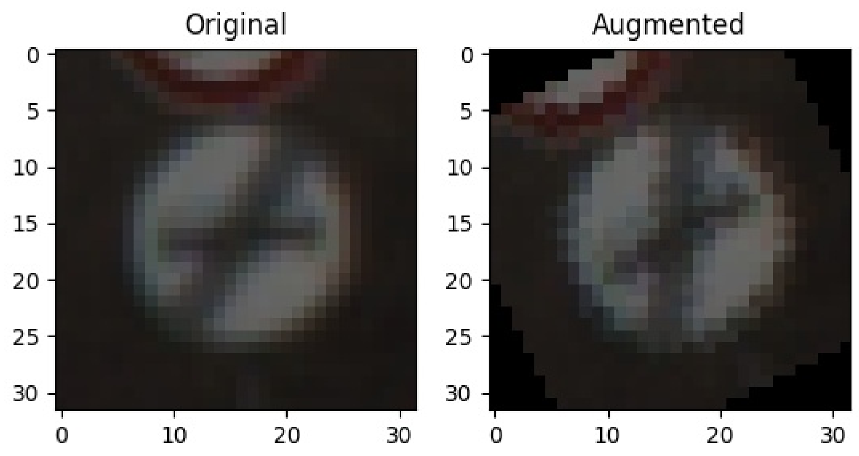
```
def augByRotation(gray_imgs, y_train, n_aug):
    '''
    Augment data by rotation.
    '''
    # Set upper and lower bounds of random variables
    lower, upper = -30, 30
    # Set the mean and standard deviation. Make the std small so that the ditribution is more uniformed
    mu, sigma = 0, 0.4
    # Get a random number generator
    gen_rand = stats.truncnorm(
        (lower - mu) / sigma, (upper - mu) / sigma, loc=mu, scale=sigma)

    size_ = gray_imgs.shape
    n_train = size_[0]

    X_aug = np.zeros((n_aug, size_[1], size_[2], 1))
    Y_aug = np.zeros((n_aug,))
    for i in range(n_aug1):
        r = np.random.randint(n_train)

        img = Image.fromarray(np.squeeze(gray_imgs[r,:,:,:]))

        X_aug[i,:,:,0] = img.rotate(gen_rand.rvs(1))
        Y_aug[i] = y_train[r]

    return X_aug, Y_aug


def augByNoise(gray_imgs, y_train, n_aug):
    '''
    Augment data by adding noise.
    '''

    # Noise generator
    lower, upper = -4, 4
    mu, sigma = 0, 0.4
    gen_rand = stats.truncnorm(
        (lower - mu) / sigma, (upper - mu) / sigma, loc=mu, scale=sigma)

    size_ = gray_imgs.shape
    n_train = size_[0]

    X_aug = np.zeros((n_aug, size_[1], size_[2], 1))
    Y_aug = np.zeros((n_aug,))

    for i in range(n_aug):
        noise = gen_rand.rvs((size_[1], size_[2]))
        #print(noise.shape)
        r = np.random.randint(n_train)
        X_aug[i,:,:,0] = np.maximum(np.squeeze(gray_imgs[r,:,:,:])+noise, np.zeros((32,32)))
        Y_aug[i] = y_train[r]

    return X_aug, Y_aug


def augDataZeroMean(rgb_images_train, y_train, n_aug_r, n_aug_n):
    '''
    Augment data.
    '''

    gray_imgs = rgb2grayBatch(rgb_images_train)

    X_aug1, Y_aug1 = augByRotation(gray_imgs, y_train, n_aug_r)
    X_aug2, Y_aug2 = augByNoise(gray_imgs, y_train, n_aug_n)

    X_train = np.concatenate((gray_imgs, X_aug1))
    y_train = np.concatenate((y_train, Y_aug1))
    X_train = np.concatenate((X_train, X_aug2))
    y_train = np.concatenate((y_train, Y_aug2))

    X_train = preprocessImgsZeroMeanGray(X_train)

    return X_train, y_train
```

As putted in the comments above, the traing image is rotated randomly between -30 degree and 30 degree.

```
n_aug1 = np.floor(0.4*n_train).astype(int)
n_aug2 = np.floor(0.4*n_train).astype(int)
X_train, y_train = augDataZeroMean(X_train, y_train, n_aug1, n_aug2)
```

The train data was augmented as above. The size of augmented dataset is `n_aug = np.floor(0.4*n_train).astype(int)`, i.e., 40% of the original dataset.

Here is an example of an original image and an augmented image:



2. Describe 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.

My final model consisted of the following layers:

Layer	Description
Input	32x32x3 RGB image
Convolution 5x5x3	1x1 stride, SAME padding, outputs 32x32x16
RELU	
Max pooling	2x2 stride, outputs 16x16x16
Convolution 5x5x9	1x1 stride, SAME pading, outputs 16x16x10
RELU	
Max pooling	2x2 stride, outputs 8x8x10
Fully connected	640x160, outputs 160
RELU	
Dropout	
Fully connected	160x84, outputs 84
RELU	
Dropout	
Softmax	etc.
Fully connected	84x43, outputs 43

3. Describe how 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.

To train the model, I used an AdamOptimizer:

- The initial learning rate is 0.001.
- EPOCHS = 100
- BATCH_SIZE = 128

4. Describe the approach taken for finding a solution and getting the validation set accuracy to be at least 0.93. 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.

My final model results were:

- training set accuracy of 0.999968
- validation set accuracy of 0.969161
- test set accuracy of 0.949169

An interative approach was adopted:

- First LeNet was choosen because it performs well on classifying grayscale images and is not very complicated;
- I used dropout layers to reduce overfitting effects. keep probability = 0.5 for training;

- Then the total number of prameters were increased by increasing depth of the first Conv2D from 6 to 16, and increased dimension of the first fully connected layer from 120 to 160. With more parameters, the model had more freedom to fit the data;
- Next I reduced the kernel size of Conv2D from 5 to 3, and adoptted SAME padding; As a rule of thumb, we could use small kernels with deepper depth insted of large kernels with less depth;
- The depth of the second Conv2D layer was reduced from 16 to 10 such that the total amount of parameters were similar to the original lenet;
- The `EPOCHS` was increased from 10 to 100 because the validation rate was stable after epoch 100.

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.

Here are five German traffic signs that I found on the web:

The first image might be difficult to classify because ...

2. Discuss the model's predictions on these new traffic signs and compare the results to predicting on the test set. 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).

Here are the results of the prediction:

Image	Prediction
Children crossing	Children crossing
Speed limit (70km/h)	Speed limit (70km/h)
Roundabout	Roundabout
Turn right ahead	Turn right ahead
Stop	Stop

The model was able to correctly guess 5 of the 5 traffic signs, which gives an accuracy of 100%. This compares favorably to the accuracy on the test set of 94.92%

3. Describe how certain the model is when predicting on each of the five new images by looking at the softmax probabilities for each prediction. Provide the top 5 softmax probabilities for each image along with the sign type of each probability. (OPTIONAL: as described in the "Stand Out Suggestions" part of the rubric, visualizations can also be provided such as bar charts)

The code for making predictions on my final model is located in the 21th cell of the Ipython notebook.

I pasted a part of the result here. As can be seen below, the model predicts all the five images quite accurately and the lowest probability is 99.48% of the third image(roundabout); and the scond predicted label of the third image is priority road.

```
      [[  9.99999404e-01,   5.90786669e-07,   3.07226822e-09,1.50683202e-10,   5.93126034e-11],
       [  1.00000000e+00,   4.62280457e-31,   1.01488139e-35, 0.00000000e+00,   0.00000000e+00],
       [  9.94755507e-01,   5.20829717e-03,   2.98061968e-05, 3.12066277e-06,   3.00460010e-06],
       [  1.00000000e+00,   1.81645952e-08,   8.92350582e-10, 2.10054868e-12,   3.65012789e-14],
       [  1.00000000e+00,   9.74361663e-28,   4.50242795e-28, 1.08726810e-29,   9.72845294e-30]],
indices=array([[28, 29, 22, 35, 24],
               [ 4,  1,  8,  0,  2],
               [40, 12,  7, 32,  8],
               [33, 14, 35, 13, 25],
               [14,  1, 38,  4, 13]]
```

(Optional) Visualizing the Neural Network (See Step 4 of the Ipython notebook for more details)

1. Discuss the visual output of your trained network's feature maps. What characteristics did the neural network use to make classifications?

The output images are shown in the 27th cell of the Ipython notebook. From the output characteristic maps of the first Conv2D, we can clealy see the contour of circles and straight lines of traffic sign 70km/h while straight lines of children crossing. Therefore, we know that the model has high activations for circles and straight lines.