Swapnil Daingade (sdaingad)

Traffic Sign Recognition

Writeup

Build a Traffic Sign Recognition Project

The goals / steps of this project are the following:

- Load the 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.

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.

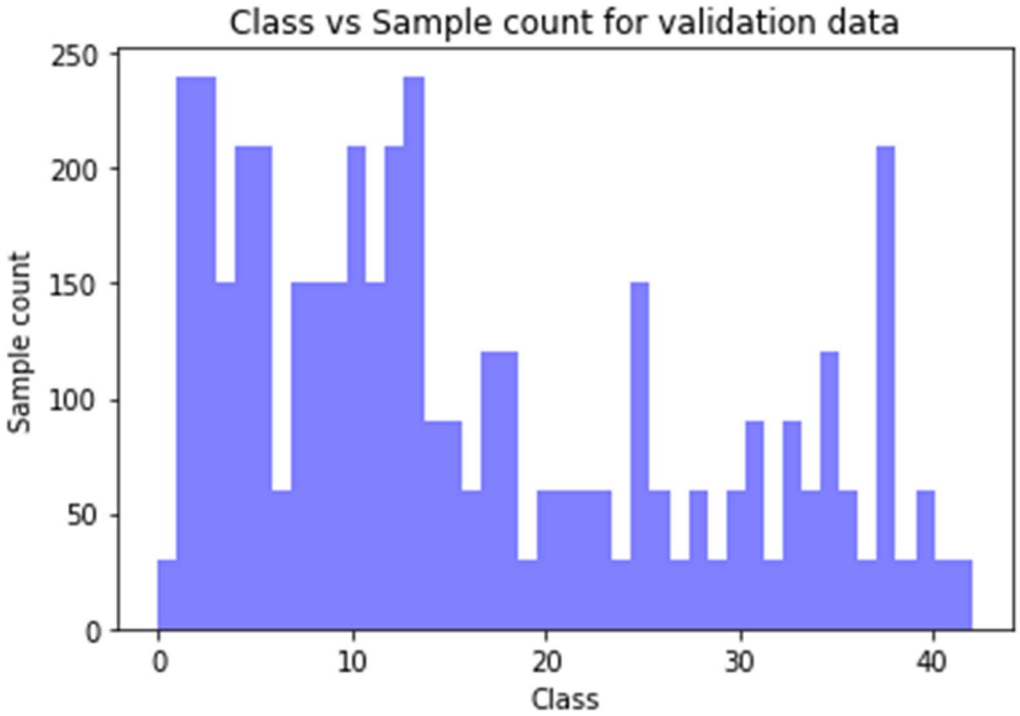
- The size of training set is 34799 (before data augmentation). After data augmentation it is > 400,000
- The size of the validation set is 4410
- The size of test set is 12630
- The shape of a traffic sign image is (32, 32, 3)
- The number of unique classes/labels in the data set is 43

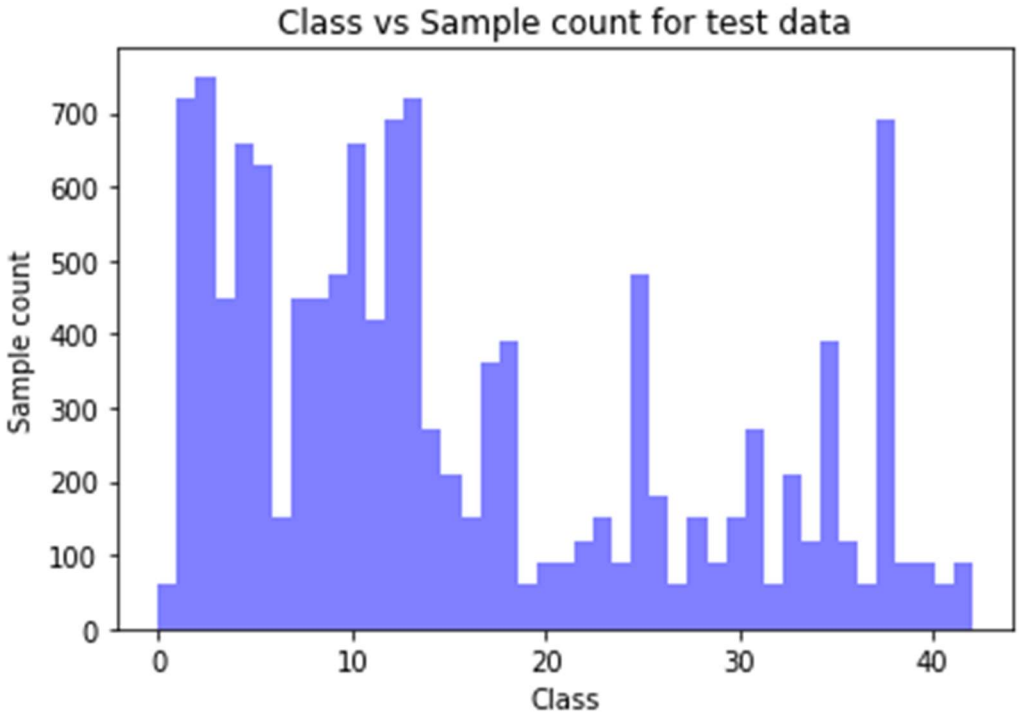
2. Include an exploratory visualization of the dataset.

Here is an exploratory visualization of the data set. It is a bar chart showing how the data samples are spread across class in training, validation and test dataset.

The first three graphs contain sample count per class **before data augmentation**

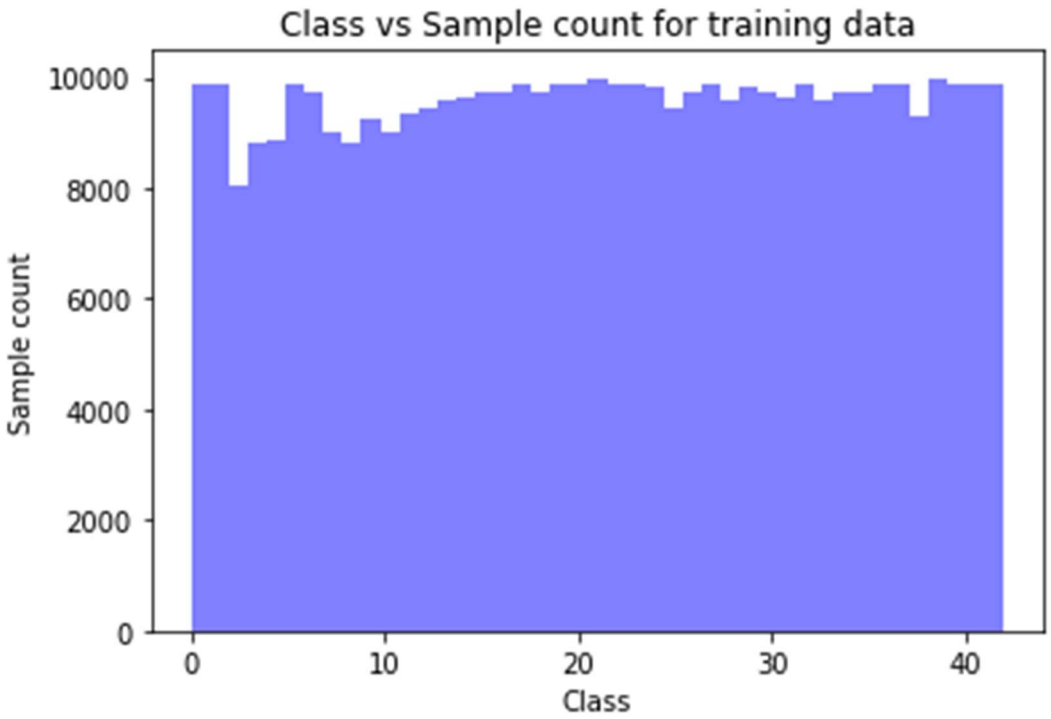
Sample count per class in training data (class 0 to class 42): [180, 1980, 2010, 1260, 1770, 1650, 360, 1290, 1260, 1320, 1800, 1170, 1890, 1920, 690, 540, 360, 990, 1080, 180, 300, 270, 330, 450, 240, 1350, 540, 210, 480, 240, 390, 690, 210, 599, 360, 1080, 330, 180, 1860, 270, 300, 210, 210]





Data augmentation is performed on the training dataset. We try to get around 10000 samples for each class (Data augmentation is described in detail as part of data preprocessing)

Graph of sample count for training data **after data augmentation** look like this



Sample count per class **after data augmentation (class 0 to class 42)**: [9900, 9900, 8040, 8820, 8850, 9900, 9720, 9030, 8820, 9240, 9000, 9360, 9450, 9600, 9660, 9720, 9720, 9900, 9720, 9900, 9900, 9990, 9900, 9900, 9840, 9450, 9720, 9870, 9600, 9840, 9750, 9660, 9870, 9584, 9720, 9720, 9900, 9900, 9300, 9990, 9900, 9870, 9870]

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.)

Pre-processing is done in three steps

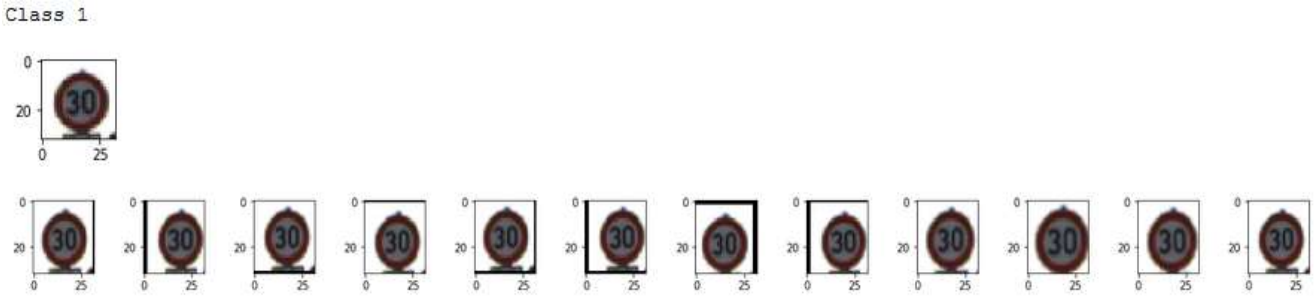
a. Data augmentation

I found that the distribution of training samples across classes was skewed. This led to lower validation accuracy for images belonging to those classes and lower validation accuracy overall.

To mitigate this problem, I do data augmentation by applying the following transformations to each image of each class

1. Shift image left (number of pixels to shift is chosen randomly)
2. Shift image right (number of pixels to shift is chosen randomly)
3. Shift image up (number of pixels to shift is chosen randomly)
4. Shift image down (number of pixels to shift is chosen randomly)
5. Shift image up and left (number of pixels to shift is chosen randomly)
6. Shift image up and right (number of pixels to shift is chosen randomly)
7. Shift image down and left (number of pixels to shift is chosen randomly)
8. Shift image down and right (number of pixels to shift is chosen randomly)
9. Zoom In/Out (randomly between 0.8x to 1.2x)
10. Zoom In/Out (randomly between 0.8x to 1.2x)
11. Zoom In/Out (randomly between 0.8x to 1.2x)
12. Zoom In/Out (randomly between 0.8x to 1.2x)

Here are these 12 transformations applied to one image in the training set to generate 12 new training images



b. Convert to grayscale

We use OpenCV's cv2.cvtColor(color_image, cv2.COLOR_BGR2GRAY) method to convert to grayscale

c. Normalize, so data has mean zero and equal variance

For each input pixel we calculate the mean value for that pixel across the dataset.
Each pixel value is normalized as follows (pixel_value – mean_value)/value_range
Normalization is applied after conversion to grayscale and the value_range is assumed to be 255.
The code to do this is captured below

```
def normalize_X_data(X_data):
    sum_X_data = np.sum(X_data, axis=0)
    avg_X_data = np.divide(sum_X_data, len(X_data))
    X_data = np.divide(X_data - avg_X_data, 255)
    return X_data
```

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	32x32x1 grayscale image
Convolution 5x5x1	Input 32x32x1, Output 28x28x24, filter 5x5x1, strides [1,1,1,1], valid padding
RELU	Activation function
Max pooling	Input 28x28x24, Output 14x14x24, ksize [1,2,2,1], strides [1, 2, 2, 1], padding 'VALID'
Convolution 5x5x24	Input 14x14x24, Output 10x10x64, filter 5x5x24, strides=[1, 1, 1, 1], padding 'VALID'
Max pooling	Input 10x10x64, Output 5x5x64, ksize [1,2,2,1], strides [1, 2, 2, 1], padding 'VALID'
Flatten	Input 5x5x64, Output 1600
Fully Connected	Input 1600, Output 480
RELU	Activation function
Fully Connected	Input 480, Output 336
RELU	Activation function
Fully Connected	Input 336, Output 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.

Here are the hyper parameters that I used to train my model

1. Learning Rate = 0.0005
   Initially started with 0.001, but decreased it to get better validation accuracy as the validation accuracy kept fluctuating with higher learning rate.
2. Optimizer = Adam
3. Batch size = 128

4. Epochs = 20
   Initially started with 10 but increased to consistently get > 93% validation accuracy.
   With 20 epochs, I have seen > 95% validation accuracy at times and > 93% validation accuracy almost always.
5. Dropout, keep probability = 0.4 , tried 0.5 and 0.3 as well but in the end settled at 0.4

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:

Validation set accuracy over 20 epochs (See Traffic_Sign_Classifier.html)

```
EPOCH 1 ...
Validation Accuracy = 0.898

EPOCH 2 ...
Validation Accuracy = 0.932

EPOCH 3 ...
Validation Accuracy = 0.950

EPOCH 4 ...
Validation Accuracy = 0.945

EPOCH 5 ...
Validation Accuracy = 0.933

EPOCH 6 ...
Validation Accuracy = 0.942

EPOCH 7 ...
Validation Accuracy = 0.945

EPOCH 8 ...
Validation Accuracy = 0.946

EPOCH 9 ...
Validation Accuracy = 0.928

EPOCH 10 ...
Validation Accuracy = 0.937

EPOCH 11 ...
Validation Accuracy = 0.939

EPOCH 12 ...
Validation Accuracy = 0.939

EPOCH 13 ...
Validation Accuracy = 0.930

EPOCH 14 ...
Validation Accuracy = 0.933
```

```
EPOCH 15 ...
Validation Accuracy = 0.937

EPOCH 16 ...
Validation Accuracy = 0.945

EPOCH 17 ...
Validation Accuracy = 0.936

EPOCH 18 ...
Validation Accuracy = 0.941

EPOCH 19 ...
Validation Accuracy = 0.943

EPOCH 20 ...
Validation Accuracy = 0.943
```

Validation accuracy is consistently over 93%

I also ran accuracy over train, validation and test for 10 epochs (See Traffic_Sign_Classifier_Training_Validation_and_Test.html)

```
Training...

EPOCH 1 ...
Training Accuracy = 0.986
Validation Accuracy = 0.921
Test Accuracy = 0.917

EPOCH 2 ...
Training Accuracy = 0.996
Validation Accuracy = 0.931
Test Accuracy = 0.937

EPOCH 3 ...
Training Accuracy = 0.998
Validation Accuracy = 0.936
Test Accuracy = 0.933

EPOCH 4 ...
Training Accuracy = 0.999
Validation Accuracy = 0.935
Test Accuracy = 0.932

EPOCH 5 ...
Training Accuracy = 0.999
Validation Accuracy = 0.933
Test Accuracy = 0.943

EPOCH 6 ...
Training Accuracy = 0.999
Validation Accuracy = 0.937
Test Accuracy = 0.946

EPOCH 7 ...
Training Accuracy = 1.000
Validation Accuracy = 0.939
Test Accuracy = 0.942
```

```
EPOCH 8 ...
Training Accuracy = 1.000
Validation Accuracy = 0.941
Test Accuracy = 0.943

EPOCH 9 ...
Training Accuracy = 1.000
Validation Accuracy = 0.948
Test Accuracy = 0.946

EPOCH 10 ...
Training Accuracy = 1.000
Validation Accuracy = 0.949
Test Accuracy = 0.944
```

Training accuracy is 100% while validation and test accuracy is consistently > 93%

If an iterative approach was chosen:

- What was the first architecture that was tried and why was it chosen?

Initially the LeNet-5 architecture was used without any preprocessing or data augmentation. This produced very low validation accuracy (~65%)

No dropout was added to prevent against overfitting

- What were some problems with the initial architecture?

There was no data preprocessing or data augmentation. The training accuracy was high (~95%) but validation accuracy would increase to ~65% and then keep fluctuating.

No dropout was added to prevent against overfitting.

Dataset was skewed, some classes has large number of samples while some had smaller numbers

- How was the architecture adjusted and why was it adjusted? Typical adjustments could include choosing a different model architecture, adding or taking away layers (pooling, dropout, convolution, etc), using an activation function or changing the activation function. One common justification for adjusting an architecture would be due to overfitting or underfitting. A high accuracy on the training set but low accuracy on the validation set indicates over fitting; a low accuracy on both sets indicates under fitting.

To increase accuracy following steps were taken

1. Data augmentation. Each class now had ~10000 samples in training set. Used shifting data in all directions and zooming in and out.
2. Color to grayscale conversion
3. Normalizing the data, Subtract mean and divide by range
4. Increase the depth of the convolutional layers in the CNNs so it can learn more features
5. Increase the number of neurons in the fully connected layers
6. Increase the number of epochs from 10 to 20
7. Decrease the learning rate from 0.001 to 0.0005
8. Add dropout with a keep probability of 0.4 (decreased from 0.5) to avoid overfitting

Various combinations of the above mentioned hyper parameters were

- Which parameters were tuned? How were they adjusted and why?

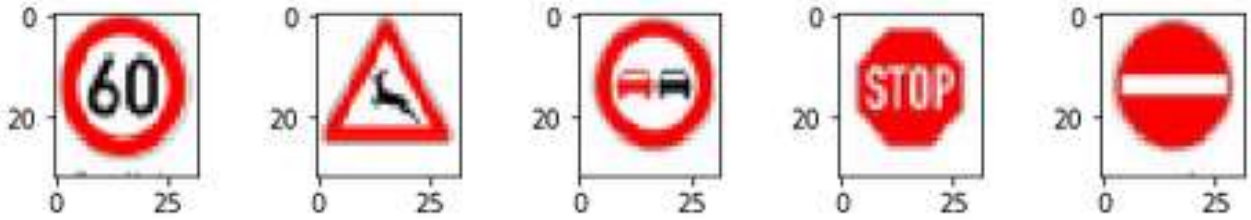
Here are the hyper parameters that I used to train my model

1. Learning Rate = 0.0005
   Initially started with 0.001, but decreased it to get better validation accuracy as the validation accuracy kept fluctuating with higher learning rate.
2. Optimizer = Adam
3. Batch size = 128
4. Epochs = 20
   Initially started with 10 but increased to consistently get > 93% validation accuracy.
   With 20 epochs, I have seen > 95% validation accuracy at times and > 93% validation accuracy almost always.
5. Dropout, keep probability = 0.4 , tried 0.5 and 0.3 as well but in the end settled at 0.4

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:



They were taken from this location on the web
https://adventuresindeutschlanddotcom.files.wordpress.com/2015/04/german-road-signs.png
These images are very different from the images in the test and validation set. They are traffic signs on a white background.
There are no other background features like trees, building etc. in these images. By picking these images I wanted to make sure
that the model I trained is trained to focus on the actual traffic signs and is not influenced by the noise in the background.

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
Speed limit (60km/h)	Speed limit (60km/h)
Wild animals crossing	Wild animals crossing
No passing	No passing
Stop	Stop

Image	Prediction
No entry	No entry

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 ~94.4%

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 cells 23 to 25 of the Ipython notebook.

Since we use softmax, all values are positive

Probability	Prediction
1.0	Speed limit (60km/h)
1.0	Wild animals crossing
1.0	No passing
1.0	Stop
1.0	No entry