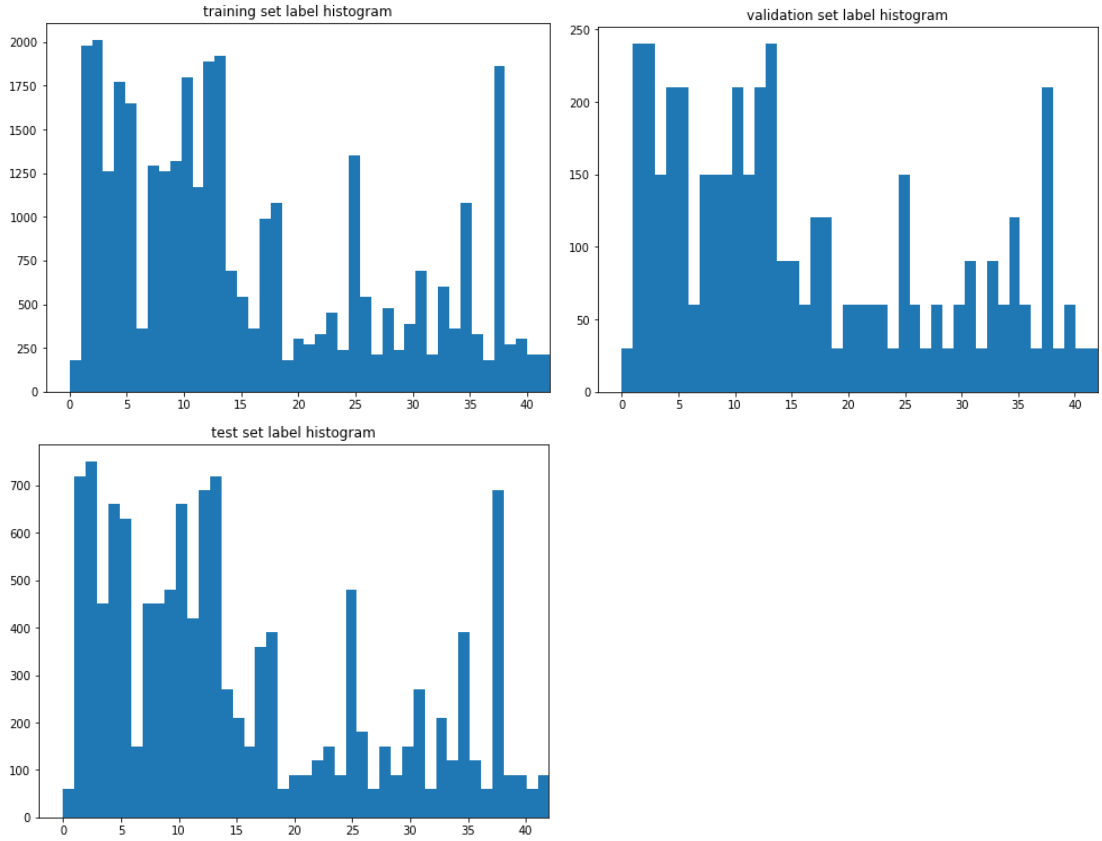
Writeup report for German traffic sign classifier project

Data Exploration

Data for this project is provided by Udacity and it has been already separated into training data, testing data, and validation data. There are 34,799 examples in training data, 4,410 examples in validation data, and 12,630 examples in testing data. The dimension of the provided image is 32x32x3, and there are 43 different classifications.

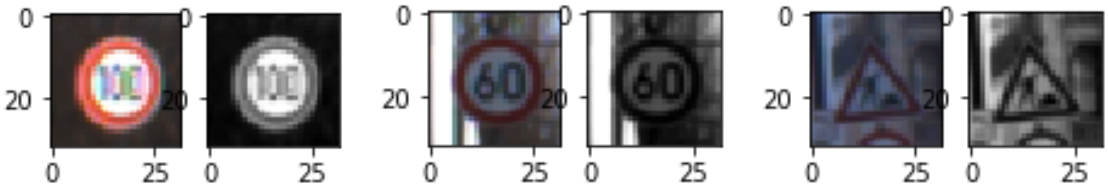
Below are sample traffic sign images. Images are traffic signs of various brightness, with some images being more easily identifiable to human eyes than others.

Number of images in the dataset varies greatly for different signs. For example, there are less than 200 images of signs with the label value of 1 in the training set. In contrast, there are nearly 2,000 images of signs with the label value of 2. The distributions of the labels seem to be similar for all three data set.



Data preprocessing

To preprocess the data, I first converted the image into grayscale and then normalized. Here are comparisons for a few RGB and normalized images.



One of the reasons for converting to grayscale is that it helps with the edge detection. Since signs have edges that need to be detected as well as the signs, converting to grayscale should aid in the training. In addition, it helps in reducing the training time, although this was a secondary concern since training time was not a hindrance when running on AWS GPU.

I also normalized the data with the mean of 0 and variance of 1, which makes the ranges of distributions of features to be similar. Since the gradient error vectors are multiplied by a single learning rate in back propagation, if the images were not normalized and had different distributions, corrections would widely vary for different images. Thus, normalizing helps with the training.

As discussed later, I did not augment the data, which would help improve training the model.

Model Architecture

Following model architecture was used to train the data.

Input

Images are grayscale and the input is 32x32x1 image.

Architecture

Layer 1: Convolutional. The output shape should be 28x28x48.

Activation. ReLU activation function. Dropouts of 85%

Pooling. Max pooling with output shape of 14x14x48.

Layer 2: Convolutional. The output shape should be 10x10x96.

Activation. ReLU of activation function. Dropouts of 85%

Pooling. Max pooling output shape of 5x5x96.

Layer 3: Convolutional. The output shape should be 3x3x172.

Activation. ReLU of activation function

Pooling. Max pooling output shape of 2x2x172.

Flatten. Flatten the output shape of the final pooling layer such that it's 1D instead of 3D. This should have 2x2x172 = 688 outputs.

Layer 4: Fully Connected. This should have 84 outputs.

Activation. ReLU activation function.

Layer 5: Fully Connected (Logits). This should have 43 outputs.

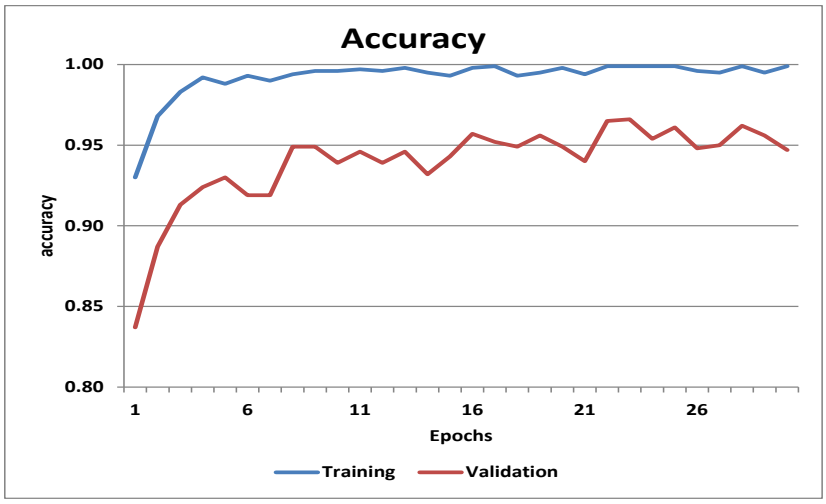
Output

Return the result of the 2nd fully connected layer.

Model Training

Batch size of 128 was used for 15 epochs with Adam optimizer to train the data. Learning rate of 0.001 was used. Mu was set as zero and sigma was 0.1. I chose the Adam optimizer because it was already implemented with the first model I had chosen and it was recommended to be used in lessons. This resulted in the test accuracy of 0.939. I am happy with the result of this particular model.

I limited epoch to 15 because training accuracy have topped almost to 1.000 by that point. Below chart shows the testing and validation accuracy when I ran it with 30 epochs. While validation accuracy increased slightly after epoch 15, training and testing accuracy did not show much improvement. Testing accuracy was 0.939 with 15 epochs and 0.943 with 30 epochs. Also, I did not see any evidence of overfitting even after I ran after 15 epochs as validation accuracy did not show any decline. This was probably aided by the dropouts in layer 1 and layer 2.



Before the final model architecture was chosen, I went through various iterations of models and hyper-parameters. Unfortunately, these iterations and their scores were not kept as records by me. I first started out with LeNet as suggested by the lesson, which gave a disappointing score. To improve the score, I increased the epochs and played around with higher and lower learning rates. I also added dropouts to reduce overfitting, especially in runs with larger epochs, and dropouts were kept until the final model.

Most of the iterations were trying different combinations of hyperparameters, particularly batch size, epochs and learning rate. However, I also added additional convolution layer and iterated with different filter size. Increasing filter size seemed to improve the score. Through numerous trials of various combinations of hyperparameters and model architecture, I arrived at my final model with good validation and testing accuracy.

Test a model on new images

I found five German sign images from the internet and tested on the trained model. Signs were obtained from various sources. One of the images was not a picture from the road, but rather was an image of a computer-generated sign. Here are the five images.

In addition, signs were of different initial shape. The initial shapes of the above five images are (957,1300,3), (262,300,3), (1024,768,3), (1024,768,3), and (1024,768,3). I would think that the model would have difficulty in identifying most of these images because the relative size of the sign in image is different than the testing images. Also, signs do not lie in the center of the images as the testing data set.

As expected, the model only predicted two of the five signs correctly (40% accuracy).

One way to improve the model accuracy is to augment the training data so images with different shapes and forms are trained. Some augmentations that can be done are rotation, translation, zoom and flips.

Below are the top five softmax probabilities of the predictions. First and second images had a correct predicted value. For these, model was very certain of its prediction with predicted value having over 90% probability. Third image was wrong, but the model was certain with nearly 100% probability. Model was also wrong on fourth and fifth images, but less certain with 60% and 78% probability, respectively.