Hardik J Patel
Traffic Sign Classifier project

Data Set Summary & Exploration

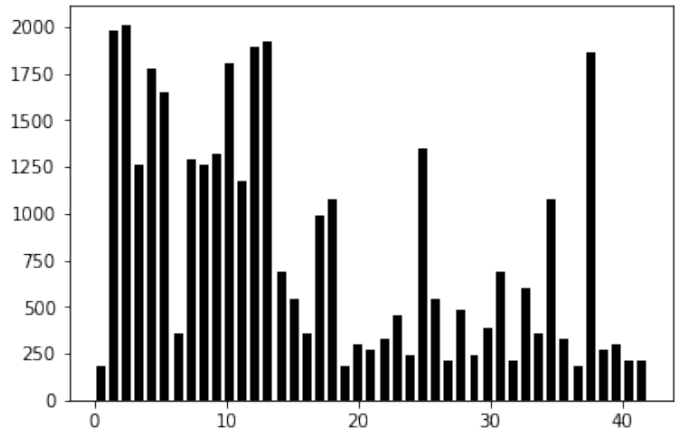
1. **Provide a basic summary of the data set. In the code, the analysis should be done using python, numpy and/or pandas methods rather than hardcoding results manually.**

I used the shape function in numpy to gather the sizes below:

- The size of training set is → 34799 images
- The size of the validation set is → 4410 images
- The size of test set is → 12630 images
- 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.**

This histogram shows the number of classes in the training data. We can see how some images are overrepresented in the training data. For example, image 1, which corresponds to "speed limit 30 km/hr" has approximately 2000 pitchers in the data, whereas, image19, corresponding to "dangerous curve to the left", has approximately 10% of image 1's share.



Design and Test a Model Architecture

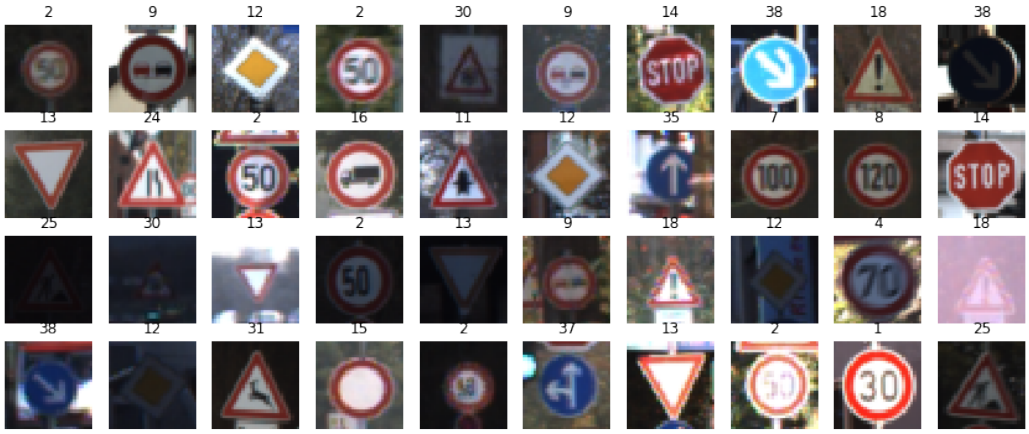
1. **Describe how you preprocessed the image data. What techniques were chosen and why did you choose these techniques? Consider including images showing the output of each preprocessing technique. Pre-processing refers to techniques such as converting to grayscale, normalization, etc. (OPTIONAL: As 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.)**

Preprocessing Techniques –

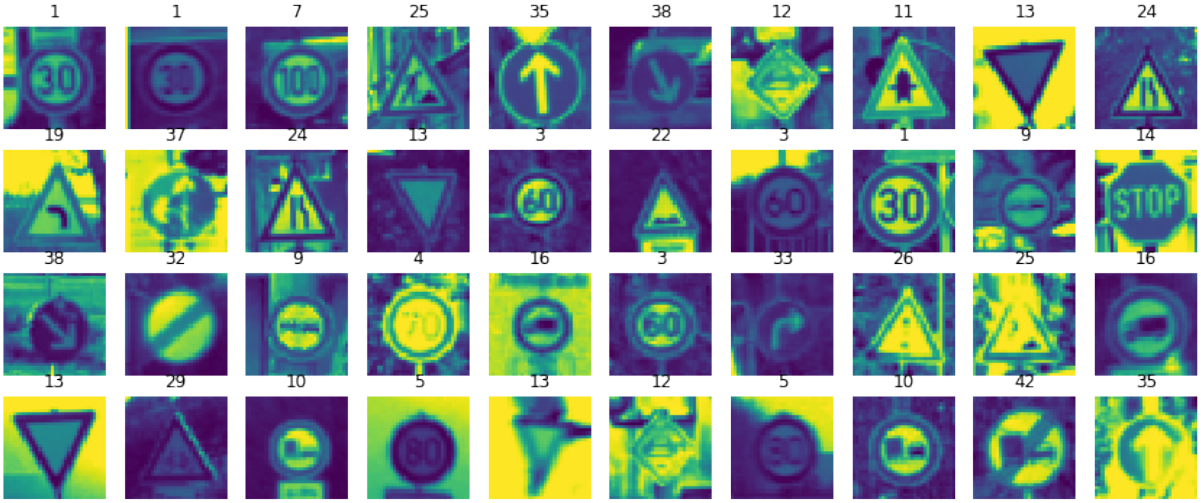
The two preprocessing techniques I used were normalization and grayscale conversion.

1. Normalization was used primarily because it was recommended. In practice, I found that normalization decreased the validation accuracy considerably. However, since it was recommended, I decided to keep the data normalized.
2. Grayscale conversion of the images improved validation accuracy of normalized images by 2 percentage points, however, I decided to keep it, because, it was less computationally heavy.
3. I also tried rotating the images.

Images before the grayscale conversion and normalization –



Images after the grayscale conversion and normalization –



The augmented data does not depend on color and is not susceptible to noise, and runs faster through the network.

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.**

The final model consisted of the following layers –

Layer	**Description**
Input Image	32*32*1 → 1 implies Grayscale image
Convolution	Output → 30*30*16 (Started with 16 filters), shape of filter = 3*3, Padding = Valid, Strides = 1*1
Rectified Linear Unit (ReLU)	Activation
Convolution	Output → 28*28*32 (Increase filters to 32) ,), shape of filter = 3*3, Padding = Valid, Strides = 1*1
Rectified Linear Unit (ReLU)	Activation
Max Pooling	Output → 14*14*32, Strides = 2*2 and pooling area = 2*2, Padding = Valid
Convolution	Output → 12*12*64 (Increase filters to 64), shape of filter = 3*3, Padding = Valid, Strides = 1*1
Rectified Linear Unit (ReLU)	Activation
Max Pooling	Output → 6*6*64, Strides = 2*2 and pooling area = 2*2, Padding = Valid
Regularization	Dropout → Prevent overfitting

Flatten Data	For input to the fully connected layer
Fully Connected Layer 1	Input → 1568, Output → 1024
Rectified Linear Unit (ReLU)	Activation
Regularization	Dropout → Prevent overfitting
Fully Connected Layer 2	Input → 1024, Output → 256
Rectified Linear Unit (ReLU)	Activation
Regularization	Dropout → Prevent overfitting
Fully Connected Layer 3	Input → 256, Output → 43
Return Logits	

Notes –

- Using average pooling did not affect the accuracy of the validation set in any meaningful way.
- Adding another convolution layer increased validation accuracy past the requirement of 0.93. However, adding a 4th convolution layer did not change the accuracy, past what the 3rd layer achieved.
- Adding dropouts also helped increase the accuracy.

3. **Describe how you trained your model. The discussion can include the type of optimizer, the batch size, number of epochs and any hyper parameters such as learning rate.**

The batch size was sill kept to 128, I didn't find any accuracy changes when I halved and double it. However, increasing the Epochs made the network able to train and better tune. Hence, I set the number of Epochs to 60. I tried the Keep_prob for dropouts to be 0.5 and 0.75, and with the probability of 0.75, the network performed better. The rate of learning was another parameter that I played with, setting it to 0.05, made overfitting a problem, and decreasing it further than 0.0005 required many more epochs to successfully train. Hence, I left the parameter to 0.001.

Another parameter was the filter size, I changed that from 5*5 to 2*3 as a smaller filter, even though was computationally expensive, performed better.

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.**

The table below specifies the changes and the way I approached the problem. There were other "Conditions" that I used, however, they were not very significant.

Condition	Accuracy
Without any normalization or conversion/before we make any changes	.86
Normalized data by subtracting and dividing by 128	.75
Converted images to grayscale	.77
Without normalization and with grayscale conversion	.88
Convert to grayscale, Dropout	.90
Grayscale, Dropout, Learn rate = .005, Epochs = 20, keep_prob = .5	.90
Similar values as above, except keep_prb = .75	.92
With another convolution layer and filters going up to 64	.95

A lot of this was performed in a trial and error basis. Changing a parameter and then changing another could perform worse than just changing one parameter.

If I tuned a parameter that greatly increased the accuracy, I didn't change it back.

Validation set accuracy → 0.936
Test set accuracy → 0.938

Test a Model on New Images

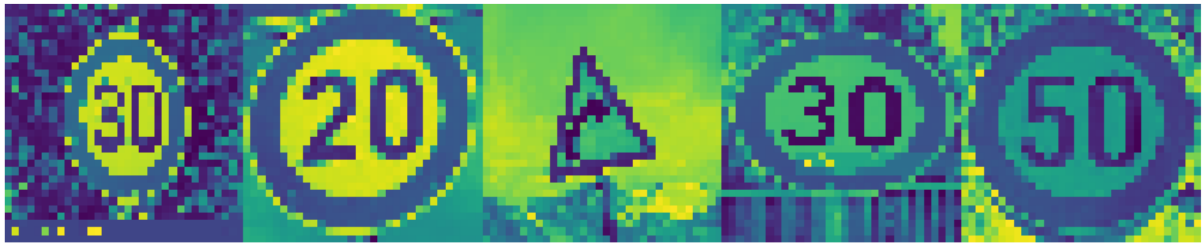
1. **Choose five German traffic signs found on the web and provide them in the report. For each image, discuss what quality or qualities might be difficult to classify.**

The images I chose are below.

The reason I chose two of the same images for the "30 km/hr" limit is because I wanted to see whether the network could identify two pictures when these were the same, with similar accuracy.

All pictures are very clear, and crisp. However, after I reshape them to 32*32 to be fed through the network, these become quire granular, as shown below -

And after the data augmentation process –



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**

The model was able to predict 3 out of 5 pictures, with an accuracy of 80%. The image with the problem is the turn right sign.

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.**

```
TopKV2(values=array ([[ 9.30411339e-01, 6.95787668e-02, 5.08265703e-06,
         2.66692223e-06, 9.05549769e-07],
       [ 9.87804055e-01, 1.21958787e-02, 4.09039637e-13,
         8.19734837e-14, 1.36120718e-18],
       [ 9.92401600e-01, 7.59264641e-03, 3.27135240e-06,
         2.02782417e-06, 4.07311802e-07],
       [ 9.95213389e-01, 4.78636939e-03, 2.21917901e-07,
         1.87000087e-08, 4.68579927e-13],
       [ 8.77376556e-01, 1.22621432e-01, 1.75323453e-06,
         9.29229174e-08, 4.70624855e-08]], dtype=float32), indices=array
([[ 5,  1,  3, 40,  0],
  [ 0,  1,  3, 28, 18],
  [20, 23, 30, 31, 22],
  [ 1,  5,  6, 14,  2],
  [ 1,  2,  5,  4,  6]], dtype=int32))
```

Above are the softmax outputs. The model seems to be sure of its predictions, however, it misses its mark in a few cases.