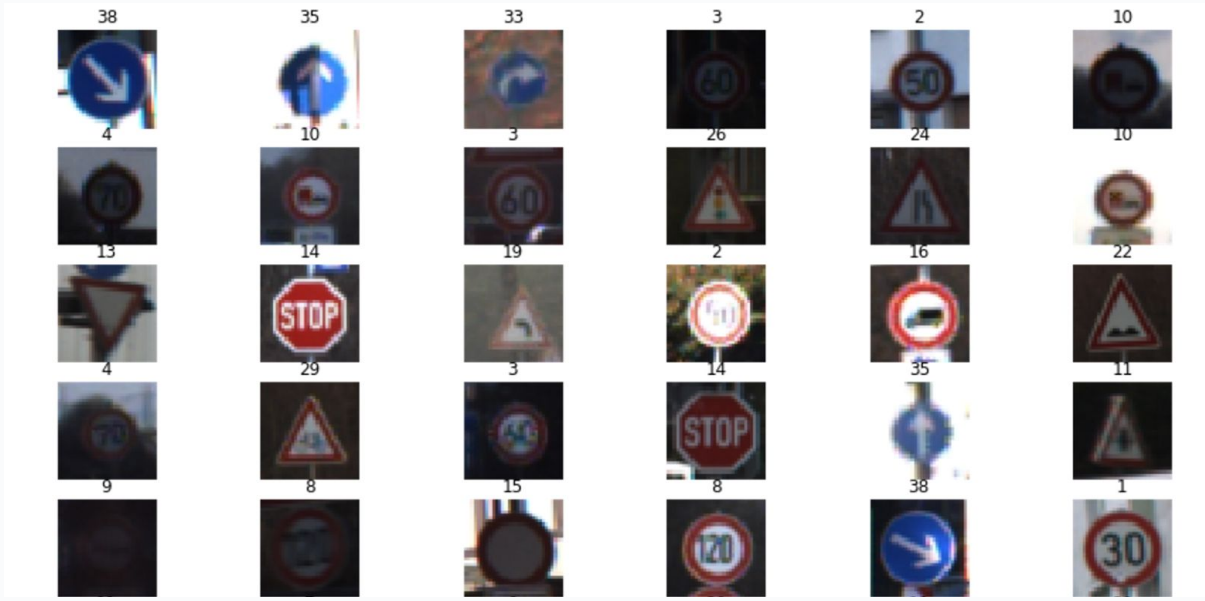
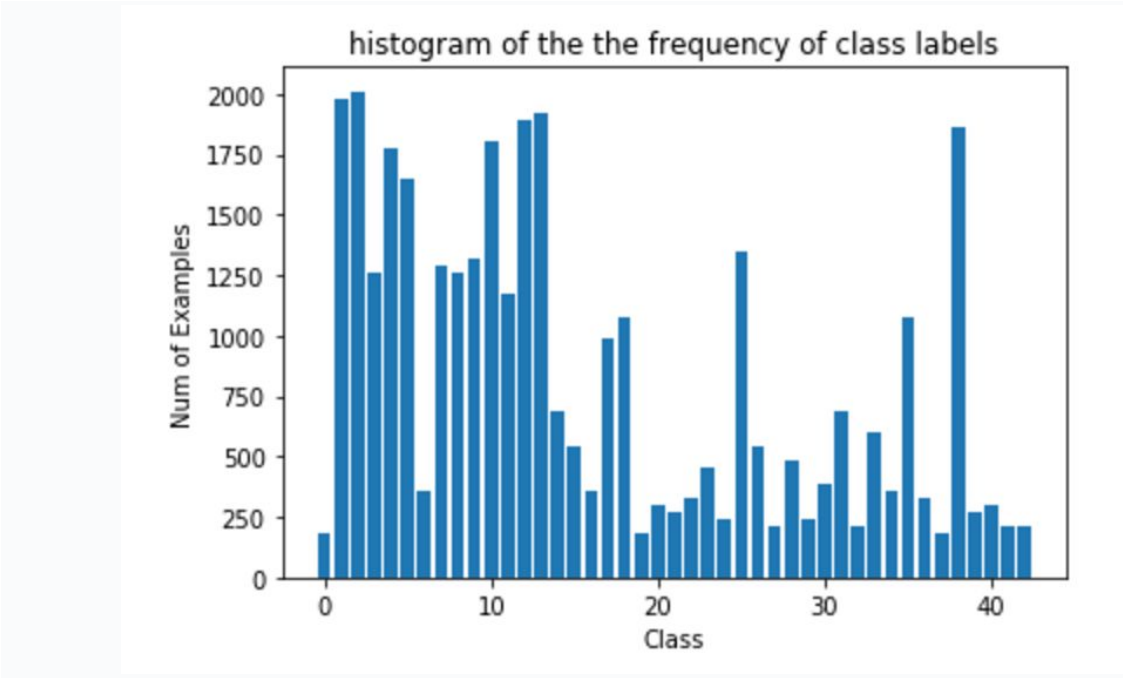
Dataset Exploration

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

Let's have a look at some of the sample training images.





As we can see that our data set is highly imbalanced. Some of the classes have only ~200 samples, while few other have >2000 samples. We shall address this problem by adding synthetic data to create for a relatively more balanced data set.

Preprocessing

I considered the following options for preprocessing:

(1) Gaussian blur
(2) RGB to gray conversion
(3) Normalization scheme 1: (RGB-128)/128
(4) Normalization scheme 2

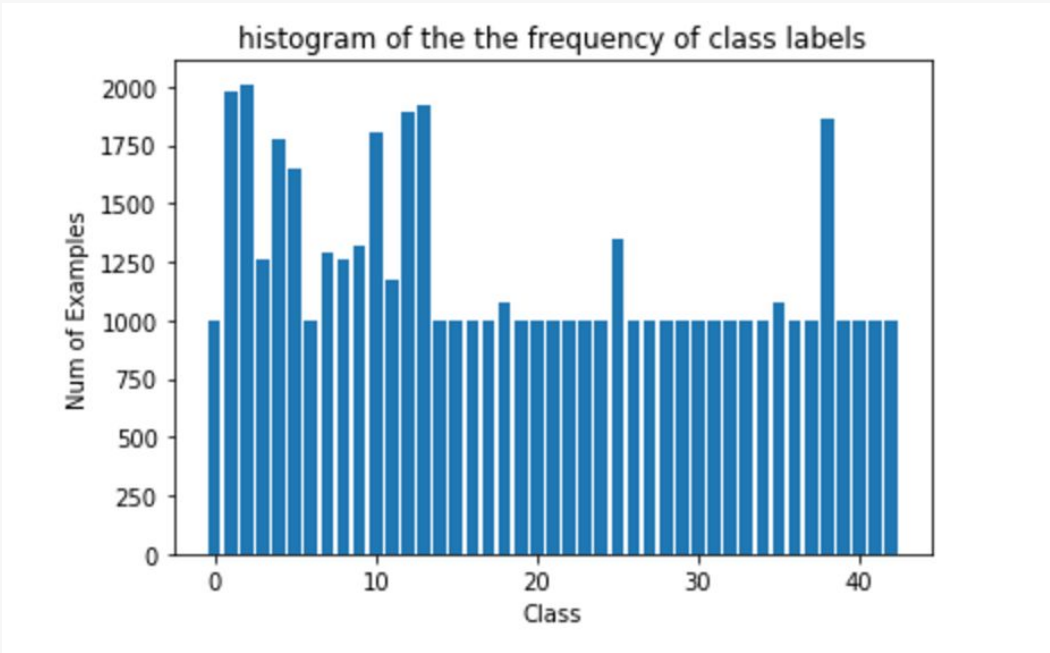
Out of these 4, I experimentally found that (2) and (3) improve the classification accuracy on validation and testing dataset. While, (1) and (4) decrease the performance. Therefore, I decided to use only (2) and (3) in my pipeline.

I added a function to transforms input images to generate new images. The function takes in following arguments:

1- Image
2- ang_range: Range of angles for rotation
3- shear_range: Range of values to apply affine transform to
4- trans_range: Range of values to apply translations over.

A Random uniform distribution is used to generate different parameters for transformation
 # Ref: https://github.com/vxy10/ImageAugmentation

All the classes with less than 1000 images were added with the synthetic data to make the samples count for the class to 1000.



Net Architecture

I used the standard LeNet architecture as my base. I added three 1x1 convolution layers immediately after the input which I expected to transform the 3 layer RGB input to a suitable color space. But, it only resulted in decreased accuracy. Therefore, I removed it.

The final architecture of my net was as following:

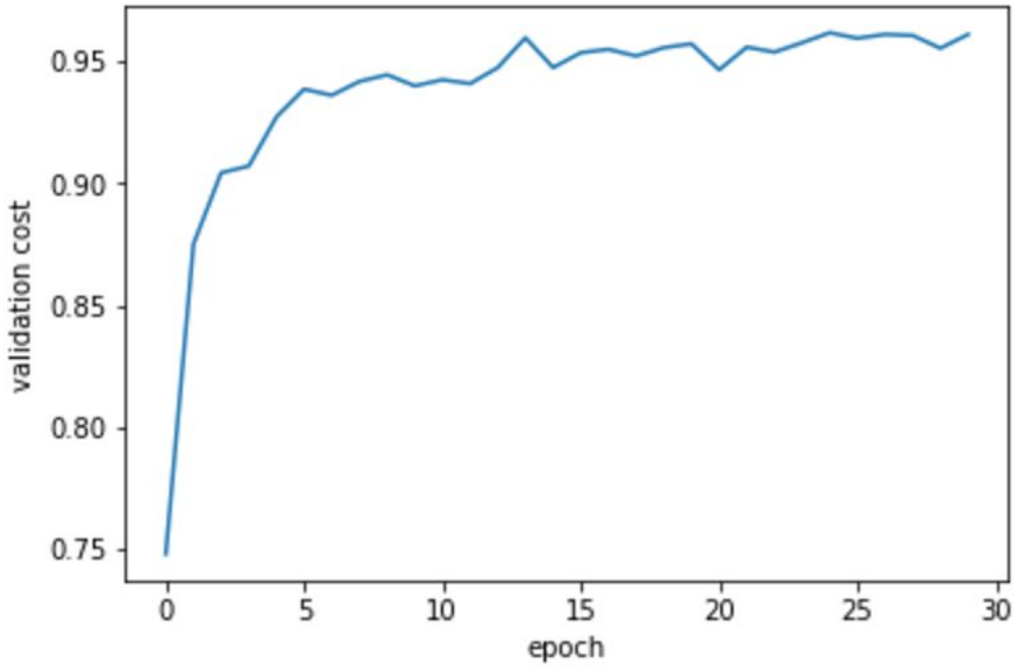
Layer Name	Input Size	Output Size
conv1	32x32x1	28x28x6
Relu1 & Local Response Norm	28x28x6	28x28x6
Max Pool1	28x28x6	14x14x6
conv2	14x14x6	10x10x16
Relu2 & Local Response Norm	10x10x16	10x10x16
Max Pool2	10x10x16	5x5x16
flatten	5x5x16	400
fully-connected1	400	120
Relu and Dropout	120	120
fully-connected2	120	84
Relu and Dropout	84	84
fully-connected3	84	43
softmax	43	43

Furthermore, I attempted the following changes into the architecture:

(1) Local Response Normalization
(2) Dropout
(3) L2 regularization

All of these 3 seems to be improving the performance on validation and test set. In fact Local-response-normalization is most effective of all. It improved the performance by almost 2%.

After 50 Epochs:
Validation Accuracy = 96.4%
Test Accuracy = 94.7%


validation cost
0.95
0.90
0.85
0.80
0.75
0
5
10
15
20
25
30
epoch

New Images

I manually downloaded these 5 test images using Google image search.

Image Name	Actual Label	Predicted Label
1.jpg	14	
2.jpg	2	
3.jpg	14	
4.jpg	39	
5.jpg	8	

=> Top 1 accuracy = 0%

Image Name	Certainty Level	True Sign present in Top 5 Predictions
1.jpg	18.7%	No
2.jpg	16.1%	No
3.jpg	18.0%	No
4.jpg	17.4%	No
5.jpg	16.4%	No

=> Top 5 accuracy = 0%

Softmax Probabilities for Top -5 candidates

Image Name	Softmax probabilities
1.jpg	[0.18738286, 0.13620333, 0.08324437, 0.06407724, 0.05702313]
2.jpg	[0.16108437, 0.13120322, 0.08417366, 0.06388941, 0.05764539]
3.jpg	[0.18056782, 0.1446635 , 0.09985326, 0.06637268, 0.06167481]
4.jpg	[0.1741706 , 0.14633104, 0.08404884, 0.0727352 , 0.06670956]
5.jpg	[0.16425672, 0.10036902, 0.07733455, 0.06124592, 0.04059792]

Looking at the above softmax values, the classifier does not seem to be very confident of its decisions, as even the max confidence value is ~18% only.

Filters from First Conv Layer