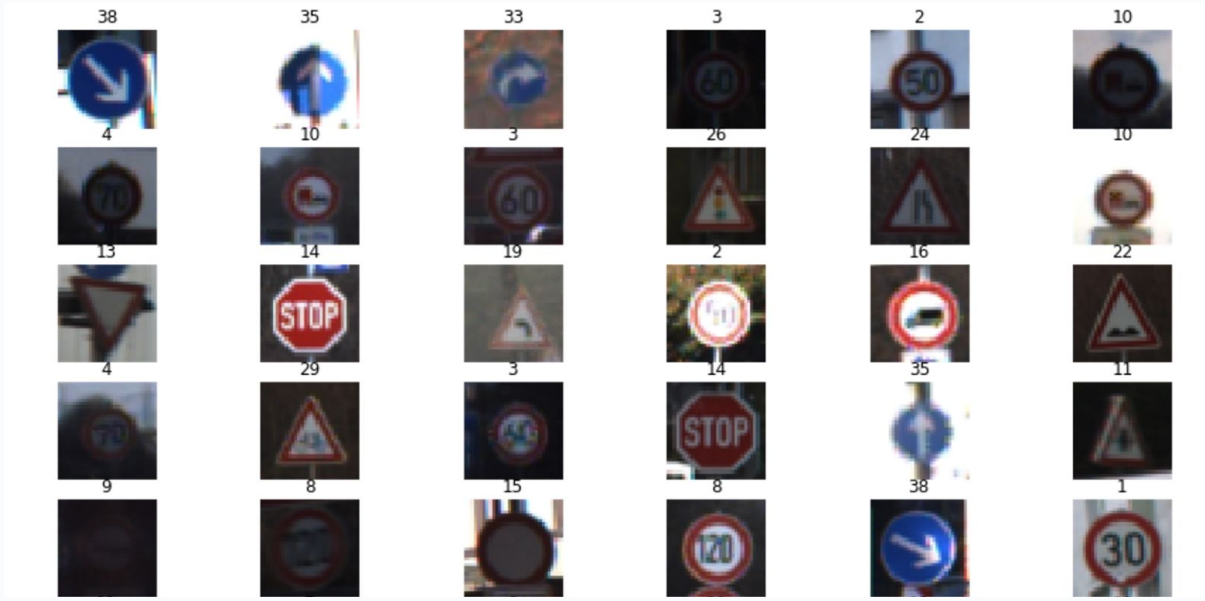
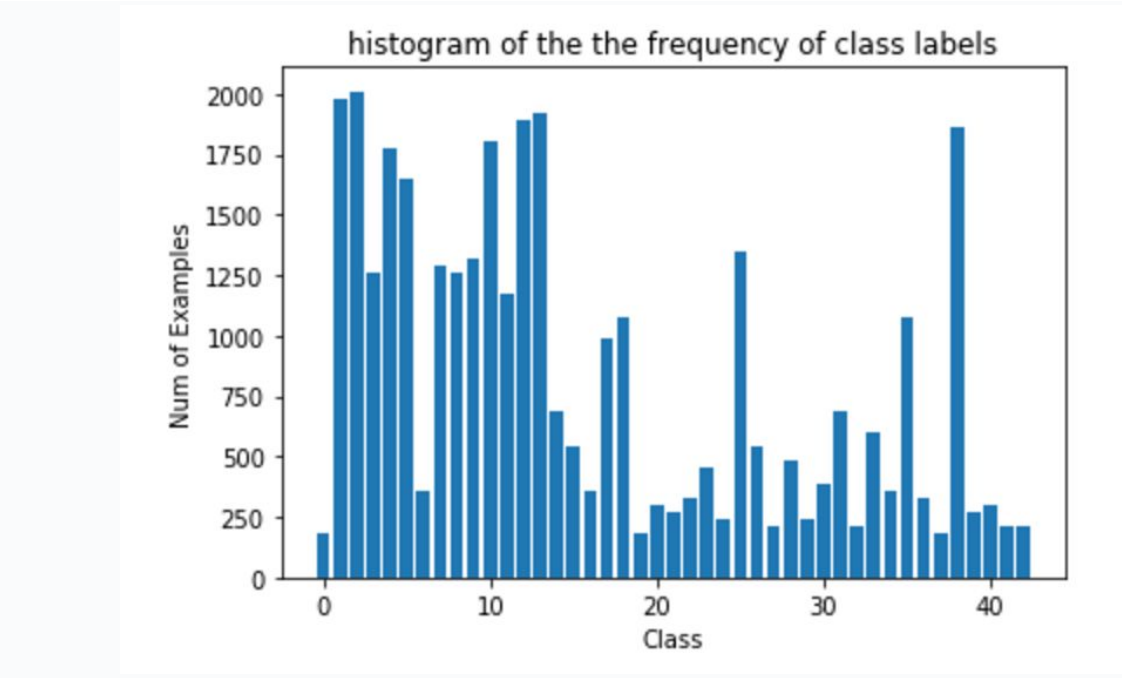
Dataset Exploration

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
(5) Adaptive Equalization (CLAHE)

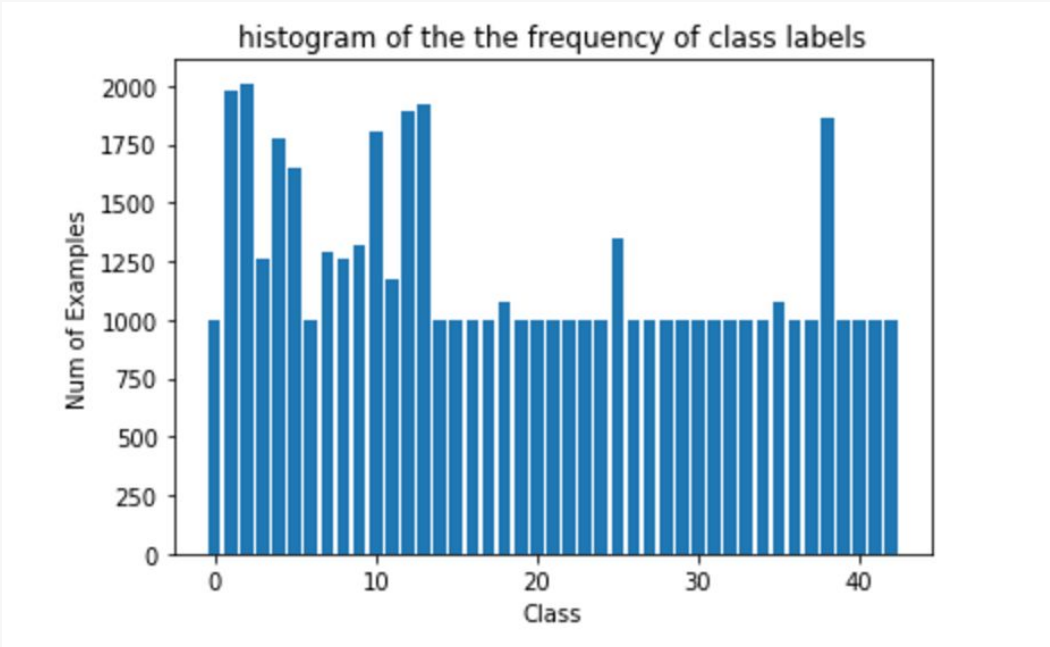
Out of these 5, I experimentally found that (4) improves the classification accuracy on validation and testing dataset. While, rest underperform in comparison. Therefore, I decided to use only (4) in my pipeline.

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.

Many of the early deepnet made use of a mixture of convolutional and fully connected layers. The convolutional layers work as filters on images and extract the features. The first convolution layer learns most basic features such as edges, lines, corners, etc. The next convolutional layer builds over these features and learn more advanced features. This continues to the following convolutional layers.

We put usually 2 to 3 fully-connected (FC) layers after convolutional ones. These layer work as classifiers on the features extracted by convolutional layers. Tradinally, 3 layer FC network has been trained in publications well. It does a good job of classification. A 4 layer FC network becomes too complex to train, because of excessive number of trainable parameters.

Pooling layer help with reducing the number of parameters in the networks. We have millions of parameters in the network and it helps to avoid overfitting. Similarly, Dropout is another trick which gives advantage of ensemble of classifiers within a single network. Since, we randomly shut down bunch of nodes in each training pass, the remaining nodes are encouraged to learn redundant information. This creates effect of combining bunch of classifiers for making a decision. Dropout works as a regularization technique. It prevents overfitting by reducing the number of parameters in each training pass (forward and backward).

I also used L2 regularization. This keeps the magnitude of weights and bias in check and does not let them grow too large. This also helps in avoiding overfitting, since too large coefficients result in a too complex model.

For activation, I used ReLU. It is demonstrated in Alexnet ‘2012 as an excellent non-linear activation function, with really each to compute derivatives.

I also used Local Response Normalization introduced in Alexnet ‘2012. It has been proved to improve the classification accuracy by locally normalizing the brightness in the input.

The final architecture of my net was as following:

Layer Name	**Input Size**	**Output Size**
conv1	32x32x3	28x28x6
Relu1 & Local Response Norm	28x28x6	28x28x6
Max Pool1	28x28x6	14x14x6
conv2	14x14x6	10x10x16

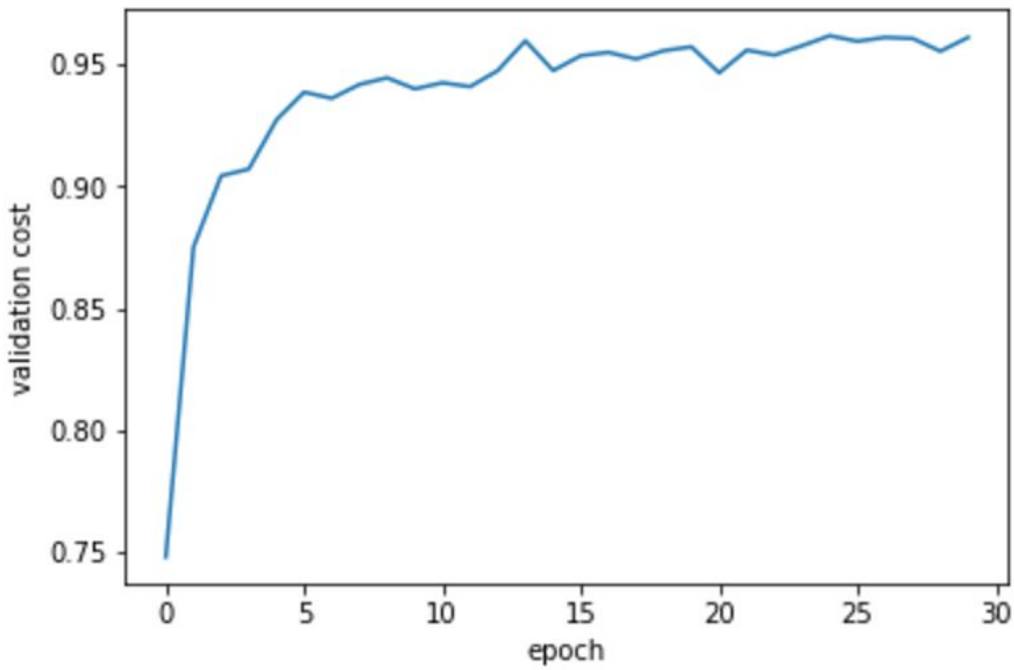
Relu2 & Local Response Norm	10x10x16	10x10x16
Max Pool2	10x10x16	5x5x16
flatten	5x5x16	400
fully-connected1	400	120
Relu and Dropout	120	120
fully-connected2	120	84
Relu and Dropout	84	84
fully-connected3	84	43
softmax	43	43

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 30 Epochs:
Validation Accuracy = 95.4%
Test Accuracy = 94.0%



New Images

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

Image Name	Actual Label	Predicted Label
1.jpg	14	14
2.jpg	2	1
3.jpg	14	14
4.jpg	39	39
5.jpg	8	0

=> Top 1 accuracy = 60%

Image Name	Certainty Level	Top 5 Predictions
1.jpg	18.7%	[**14**, 17, 12, 10, 9]
2.jpg	16.1%	[1, 0, **2**, 31, 4]
3.jpg	18.0%	[**14**, 17, 0, 9, 10]
4.jpg	17.4%	[**39**, 33, 37, 40, 6]
5.jpg	16.4%	[0, 1, 31, 16, 21]

=> Top 5 accuracy = 80%

Softmax Probabilities for Top -5 candidates

Image Name	Softmax probabilities
1.jpg	[**0.41140309**, 0.36877862, 0.20926622, 0.00894089, 0.00135778]
2.jpg	[0.94672465, 0.04030244, **0.00902292**, 0.00113126, 0.00102505]
3.jpg	[**0.990556**, 0.00348476670e-03, 2.89494940e-03, 2.16443324e-03, 4.71495616e-04]
4.jpg	[**0.998**, 0.00.31815, 0.000137785, 4.25545695e-05, 9.46409091e-06]
5.jpg	[0.84185314, 0.03713676, 0.0243352 , 0.01725472, 0.01607974]

Looking at the above softmax values, the classifier seem to be very confident of its decisions, even the wrong ones. In case of 5.jpg, clearly the classifier has all the wrong top 5 labels, but still it was pretty confident about them.

Filters from First Conv Layer