Traffic Sign Classifier

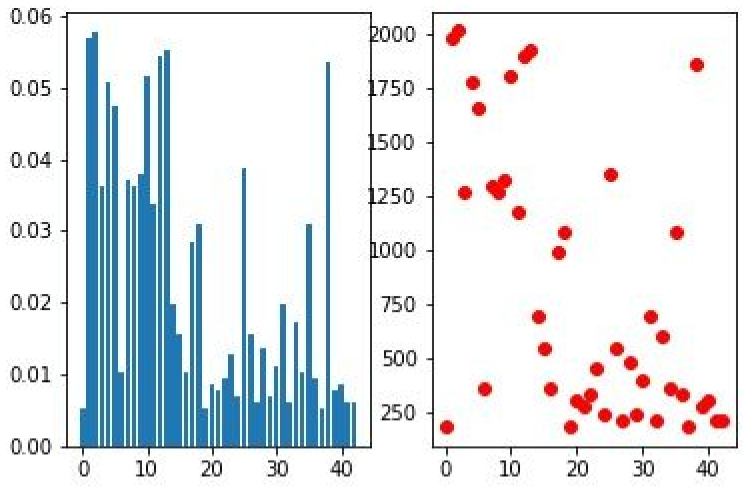
Mohammad Adil

Dataset Summary & Explanation

Size of training set	34799
Size of validation set	4410
Size of test set	12630
Shape of traffic sign image	32,32,3
Number of unique classes/labels	43

Exploratory Visualization

Visualization of the dataset was performed using a scatter plot and a bar chart. The scatter plot shows the exact number of images for each class while the bar chart shows the relative heights of each class. As we can see, the distribution is quite skewed towards a certain number of classes.



Design and Test Model Architecture

I first tried to implement Lenet-5 on the test images to see the accuracy. Converting the initial filters to grayscale and changing the fully connected layer at the end to 43 classes got the network to work on the new images. However, the accuracy was really bad. Performing image normalization got the network to work fine and produce an accuracy of around 87% or so. I then started to test several other methods used to improve training which are documented below

Increasing the width of layers

The easiest thing to do is to change the width of the layers. This means that adding more filters per each convolution layers. For the first two layers of Lenet-5, I tested different numbers including 32 and 64 channel layers and higher numbers like 200 and 400 channel layers. In the end I decided on 16 and 64 because increasing further didn't improve accuracy much. I also changed my fully connected layers to have more parameters. Testing different values, I finally settled at 400 -> 120 -> n_classes (for the output)

Adding a dropout layer

Dropout layers have been shown to improve accuracy and training performance significantly. I added a dropout layer after my first fully connected layer with a probability of 0.75. This improved accuracy every so slightly.

Removing max_pooling and replacing with strided convolution

In recent times, strided convolutions have shown to be more efficient and better than max_pooling. They do however increase the complexity of the network but strided convolutions have been implemented more efficiently in GPU recently. I replaced max pooling after the second convolutional layer with another layer with 5,5 kernel and 64 output filters. I changed my second convolution to have 32 filters instead of 64. Using a stride of [1,2,2,1], I could then get a 3,3,64 channel output from the third convolutional layer. After flattening, this becomes a total of 576 parameters decreasing the complexity of my fully connected layer (which previously had 1600 parameters). This gave me the highest advantage jumping my validation accuracy to well over 93% for most training runs. On average, I could get to a 94.5% validation accuracy over multiple runs.

Image manipulation

I tried to perform various other image manipulation routines like YCbCr conversion and grayscale conversion. However, either of these things didn't produce a significant impact in my training accuracy. Examples of grayscale and YCbCr images are shown below

Image and it's grayscale equivalent

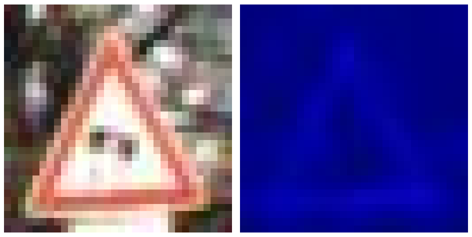


Image converted to YCbCr showing only the Y channel (on the right)

Decaying the learning rate

In order to stabilize the training after the first few epochs (when it starts to oscillate), I tried to perform decayed learning using global step and learning rate decay. However, that was really hard to get right often causing training to result in a lower final accuracy

Final Architecture

Layer	Description
Input	32x32x3 RGB
Convolution 5x5	1,1 stride, valid padding, outputs 28x28x16, relu
Max pooling	2,2 ksize and 2,2 stride, outputs 14,14,16
Convolution 5x5	1,1 stride, valid padding, outputs 10,10,32, relu
Convolution 5x5	2,2 stride, valid padding, outputs 3,3,64, relu
Flatten	Outputs 576 dimension vector
Fully connected	576->400, RELU activation

Dropout	Probability = 0.75
Fully connected	400->120, RELU activation
Fully connected	120->43, No activation (Linear)
Softmax	On output logits

Model Training

To train the model I simply used the same method as Lenet training but I tried additional things like using a decayed learning rate. I could have changed the epochs or batch size but I didn't have time to play with those.

Approach to get 93%+ accuracy

My basic approach (described in detail earlier) was to expand the architecture and then try doing some data pre-processing. I first did some initial normalization and then moved onto expanding the width of the output channels in my convolutional layers. This worked a little bit and gave me some increase in performance. I then proceeded to try increasing the hidden units in my fully connected layers. That worked a little bit but not much but increased the complexity immensely. I reduced it to a reasonable balance (400,120,43). At this point, during some runs, I could get to a 93% validation accuracy but not always

As discussed before, adding another convolutional layer and removing max pooling gave the most benefit and bumped my accuracy close to 95% on most runs. The test accuracy was also consistently close to 93% at this point. I could have expanded this more but I didn't have time so I stopped here.

Testing on new images

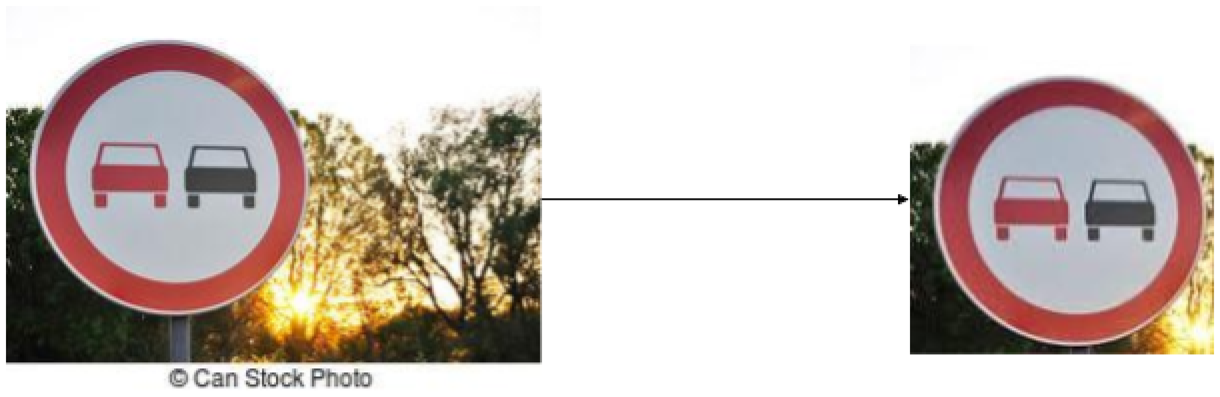
I chose 15 images from the internet in 6 categories including : "Pedestrians", "Bumpy road", "Keep right", "Yield", "No passing", "End of speed limit". Please check the 'web_examples' folder for the test images. The folders are named according to their IDs in signnames.cvs and these are used to identify their classes upon loading.

Examples from each class are given below

Classification of these 15 images resulted in a 60% accuracy. Some of the images were quite difficult. The following table shows accuracy of different classes

Road Sign Name	Number of images	Correctly predicted	Top softmax prob
Yield	3	100%	0.59
Bumpy Road	5	40%	0.276
Pedestrians	2	0%	0.136
Keep right	2	100%	0.587
End of speed limit	1	0%	0.666
No passing	2	100%	0.416

If we look at the examples, we can see that the yield sign images were really easy to pick up as there is no background etc in these images. The bumpy roads were very hard because a lot of them were rotated with backgrounds and our training set has not seen enough of these examples. Similarly, Pedestrians was hard because of no standard images found for that sign. Keep right was really easy as I took two very simple images off the internet. End of speed limit was a tilted sign and hence couldn't be detected. No passing was surprising as it performed well on one which was slightly rotated but had no background but performed badly on the other which had background. Upon seeing this, I went into the folder and cropped out the original image leaving only the sign part. Re-running the prediction resulted in a correct output.



Wrong Right

According to my analysis, the background plays a huge part. This is because the images are trained on tight crops of traffic signs.

Looking at the top softmax probabilities, it is a slightly confusing story. Many of the correct predictions are close to 0.5 top softmax probability. Most wrong predictions have very low probabilities ranging between 0.05-0.20. However, some correct predictions also have very small probabilities (0.18 in one case) despite being correct but this is not a common case. This just means that although the prediction is correct, the network is not fully sure as the image resembles other categories too.