Traffic Sign Classification Project

Data Exploration

Dataset Summary

The data used was from the pickled data provided by Udacity. Data was provided for training, validation, and testing.

File	# images
train.p	34799
test.p	12630

The image data consisted of images 32 x 32 in size and 3 color planes.

Images can be viewed in Python block In[7]

Exploratory Visualization

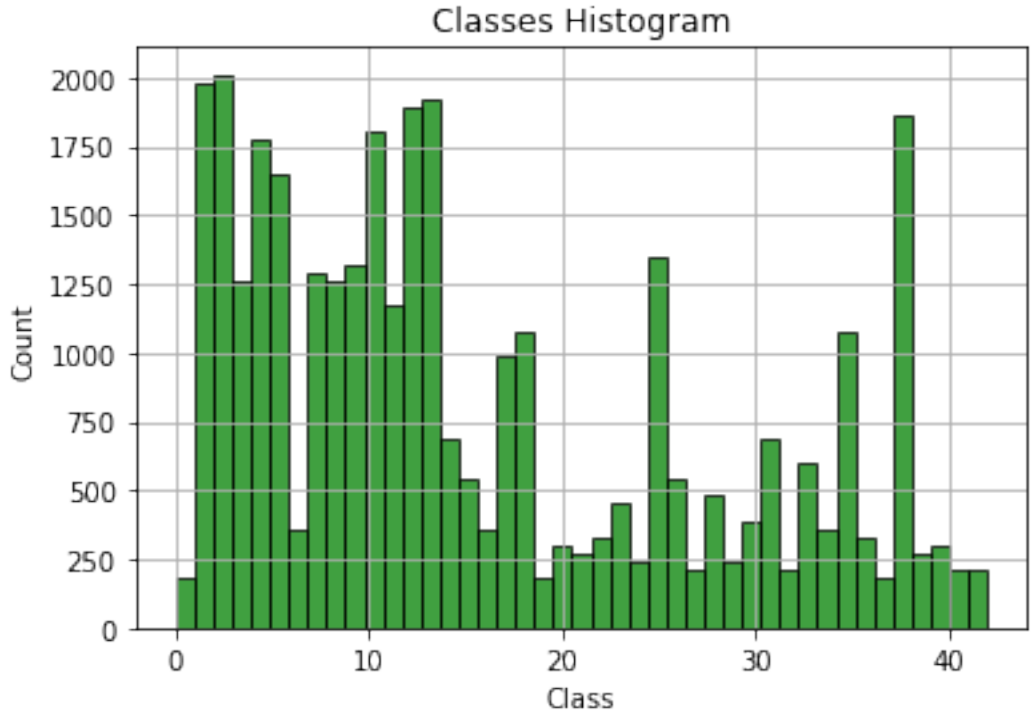

Figure 1 Class distribution Training Set

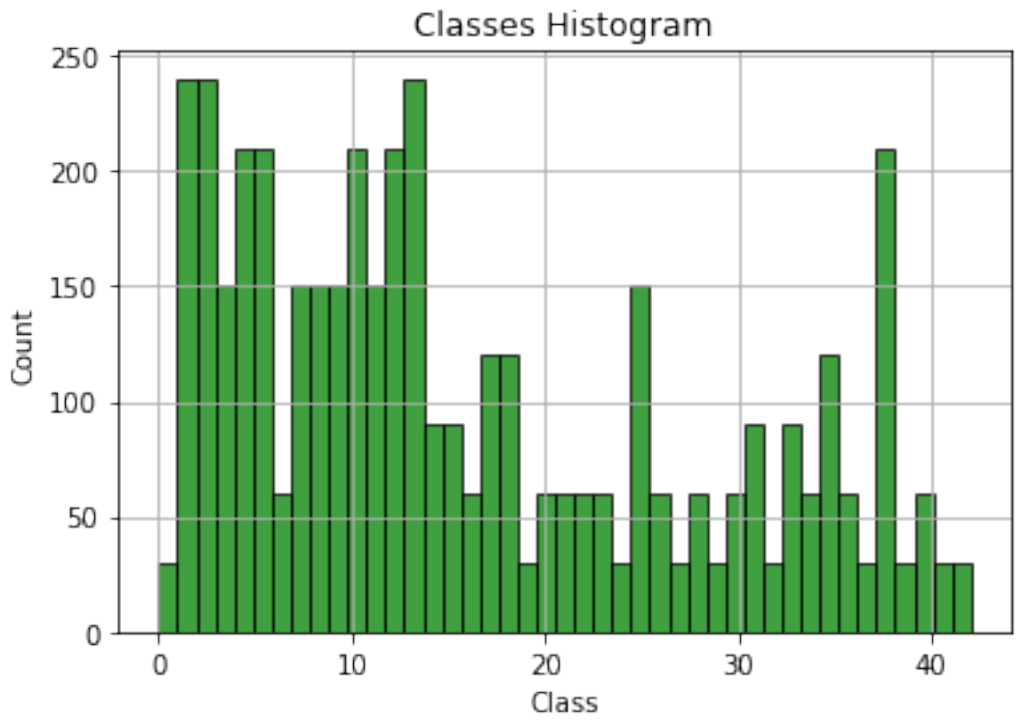

Figure 2Class distribution Validation Set

Model Architecture, Design and Test

Preprocessing

In general it is understood that when using a neural network it is often advantageous to preprocess the inputs in order to facilitate classification. I considered doing the following:

Converting the images to grayscale

Pierre Sermanet and Yann LeCun, in their paper "Traffic Sign Recognition with Multi-Scale Convolutional Networks," reported seeing greater accuracy when they converted images to grayscale. However, that was not the case during my experimentation. I found that my network trained faster with higher accuracy when using color images. This seems to fly in the face of the consensus I found on the Internet. Nevertheless, since that was what I was seeing, I went with my conclusions.

Increasing the image contrast

In addition to converting to grayscale, it was my belief that increasing the contrast of the image would help the network detect edges that would facilitate feature extraction by the convolutional layers. Nevertheless, I abandoned this as well because, since converting images to grayscale was not as effective as I wanted it to be, I thought it would be wise to hold off on increasing image contrast.

Augmenting the training set

Other ideas I read about included augmenting the training set to make the histogram more balanced. This could be done by making copies of images in underrepresented classes, and then modifying those copies with small affine transformations. This would broaden the training set in a way that would, at least in theory, help the network generalize better.

I found examples of these augmentation techniques in literature and code examples from former Udacity students. Nevertheless, I wanted to refine my architecture first before modifying the data.

Model Architecture

The approach I used is based off the LeNet architecture. The code implementing LeNet can be found at Python code block In[24].

I modified the LeNet architecture by changing activation functions and converting it into a multiscale convolutional network. The architecture is depicted in the table below:

Label	Description	SIZE	Hyperparameters Notes
Input	Input	32x32x3 RGB color	
C1	Convolutional Layer	In 32x32x3 Out 28x28x6	5x5x3 1x1 stride VALID padding
	Activation (TanH)	28x28x6	
	Subsample (Max Pool)	In 28x28x6 Out 14x14x6	Out to C2
	Subsample (Max Pool)	In 14x14x6 Out 7x7x6	2x2 stride Out to FC1
C2	Convolutional Layer	In 14x14x6, Out 10x10x16	5x5x6 1x1 stride VALID padding
	Activation (ReLu)	10x10x16	
	Subsample (Max Pool)	In 10x10x16 Out 5x5x16	2x2 stride
FC1	Fully connected layer	In 294 from C1 In 400 from C2, Out 120	All inputs are Flattened
	Activate (ReLu)	120	
FC2	Fully connected layer	In 120 Out 84	
	ReLu activation	84	
FC3	Fully connected layer	In 84 Out 43	Output layer

The code implementing the model architecture can be found in Python block In[26]

Model Training

The code for model training is in python code block 32 of the Jupyter notebook.

The hyper parameters are set in python code block 37. The model is configured to train with the following hyperparameters:

Learning rate	0.001

Epochs	100
Batch size	150

At times during the training, a particularly good accuracy level may be reached, only to be followed by a relatively poor one. Perhaps a different optimizer could have done a better job. Nevertheless, a check was put in to write to disk the state of the model every time the accuracy exceeded the last known highest accuracy. This way, the weights of the best epoch could be used even if subsequent epochs produced relatively poorer results.

Solution Approach

Modified LeNET

I started out using the LeNet architecture. This seemed like a good approach. I modified it to accept color images and output 43 classes. This actually gave results that were better than I expected but were still not good enough to satisfy that project requirements. I used a training rate of 0.001 and trained for up to 50 epochs. I was concerned about overfitting, so I tended to only train if the accuracy rates seemed to be trending upward without any random oscillations. In hindsight, I think it may have been ok to train longer. I adjusted the learning rate down to avoid the accuracy jumping back and forth as the accuracy approach anything above 90%.

Multi-Scale

After experimenting with LeNet, I decided to experiment with a Multi-Scale approach as outlined by Pierre Sermanent and Yann LeChan in "Traffic Sign Recognition with Multi-Scale Convolutional Networks". The primary change I made to my architecture was to pass the output of my first convolutional layer directly to the classifier in addition to passing it to the next convolutional layer, instead of simply passing it to the convolutional layer. This bypass does pass through another max pool layer so that it has a similar amount of subsampling as other inputs into that layer.

One other change I made was to experiment with a different activation function. I found changing the activation function used after the first convolution layer to tanh seemed to outperform the ReLu function. That is, in my testing the accuracy seemed to break past the 90% mark faster. Additionally, the accuracy results from the validation set seemed closer to that of the training set. This implied that there was less overfitting.

One last thing to note is that I did not see greater accuracy from using grayscale images verses color ones. This was a surprise since the article as well as anecdotal evidence would suggest grayscale images would perform better. Despite this however, I was never able to achieve the 99% accuracy reported in the paper.

Testing the Model on New Images

New Images (German Traffic Signs on the Web)

Per the instructions in the project assignment, I found 5 images on the web to test my conv net. In one of the pictures, the sign is slightly obscured by the face of a man in the picture. At the onset, I felt confident that my network would recognize all the signs. I was interested however to see how the

presence of the man's face would influence the recognition. As it would turn out that image presented no problem as all.

Before importing, I had to rescale the images down to 32 x 32 x 3 so they would be compatible with the architecture.

German traffic sign images from the web			
Original image	Image Class	Predicted Class	Top Softmax
	1	1	0.995
	25	22	0.825
	18	18	0.861
	18	18	0.999
	17	17	1.0
	22	22	1.0

Model's Performance

The performance of the model on the test set was not quite as good as the in the validation set in terms of raw accuracy. The network confused class 25 for class 22. Both classes are in the shape of a triangle. They are both red with a white inner part and they both have a black symbol in the center. What seemed odd however is that model selected a class with fewer training examples over the class with more.

The code that outputs the model predictions can be found at Python block In[49]

The code that outputs the model performance can be found at Python block In[52]

Model Certainty

The model had low certainty for the misclassified image. On the other hand, it had relatively high confidence for the others. As a point of interest, the model was a little unsure about the image with the man's face in it. That softmax probability was only 0.86

The code that outputs the softmax can be found at Python block In[53]