TRAFFIC SIGNS RECOGNITION

ANA-MARIA F.-B.

1. Goals of the Project

The goals / steps of this project are the following:

- Load the data set.
- Explore, summarize and visualize the data set.
- Design, train and test a model architecture.
- Use the model to make predictions on new images.
- Analyze the softmax probabilities of the new images.
- Summarize the results with a written report.

2. Data Set Summary and Exploration.

I used the numpy library to calculate summary statistics of the traffic signs data set:

- The size of training set is: 34799.
- The size of the validation set is: 4410.
- The size of test set is: 12630.
- The shape of a traffic sign image is: (32, 32).
- The number of unique classes/labels in the data set is: 43.

I used the pandas library to read the csv file which provides a dictionary between the number corresponding to each class and its actual name. This can be found in the code line 6 of the Iphyton Notebook.

To compute the distribution of the different types of signs in the data set I used the numpy method *np.unique* and counted the number of times each unique value comes up in the input array. I did this for the combined training, validation, and test set and for each of them separately. Finally I used mathplotlib to generate a plot of the obtained distributions seen in the code line 7 and 8 of the Iphyton Notebook respectively.

To visualize examples of each traffic sign in the data set I used the *numpy.where* method to find the first index where each traffic sign appears in the training set. The plot of this images can be found in the code line 11 of the Iphyton Notebook.

3. Design, Train and Test a Model Arhitecture.

The first step in this section is to preprocess the images in order to achieve consistency of the data set. I do this in two steps. First I convert the image to gray scale so that each pixel value represents only one type of color, a shade of gray. The color of each traffic sign is not giving us a lot of relevant information for its classification. It's rather the shapes and edges that are relevant. This is done by adding the pixel values of each color channel and divide by 3. Since we have 3 color channels (dimensions) with values between $[0, 255]$, the new obtained image has only one dimension with values between $[0, 255]$. Finally, we want the image data

to have mean zero and equal variance. This is achieved by subtracting 128 (the mean of the pixel values) from each pixel and then dividing by 128. The new pixel values lie in the interval $[-1, 1]$. This can be seen in the example in the code line 15 of the Jupiter Notebook. I also use *plt.imshow* to plot an example of the normalized gray scale image in line 14 of the Jupiter Notebook.

The model architecture I chose for this project is based on the LeNet architecture introduced in 1988 by Yann LeCun et. all. The following table summarizes the structure of the final architecture:

Layer	Description	Output
Input	32x32x1 Gray Scale Image	-
Convolutional 5x5	1x1 stride, valid padding	28x28x6
Activation	ReLU	28x28x6
Max pooling	2x2 strides, valid padding	14x14x6
Convolutional 5x5	1x1 stride, valid padding	10x10x16
Activation	ReLU	10x10x16
Convolutional 3x3	1x1 stride, valid padding	8x8x8
Activation	ReLU	8x8x8
Flatten	-	512
Fully connected	-	120
Activation	ReLU	120
Dropout	keep probability 0.7	120
Fully connected	-	84
Activation	ReLU	84
Dropout	keep probability 0.7	84
Fully connected	-	10

The starting LeNet Architecture had an accuracy of 0.89 on the validation set. In order to improve the accuracy I changed the initial architecture as follows:

- I removed the pooling layer after the second convolutional layer.
- I added a third convolutional layer of type 3x3.
- I used a ReLU activation function for the third convolutional layer.
- After the first and second fully connected layers I applied the dropout regularisation technique.

In the section 'Training the main architecture" of the Ipython notebook I trained the model with different values for hyperparameters in order to find the best choice of hyperparameters. I had 18 training sessions, each for 50 epochs, with all the permutations on the following set of hyperparameters:

Batch size	Dropout	Learning Rate
128	0.65	0.001
258	0.7	0.002
-	0.75	0.003

For the backpropagation step in the learning process I used the *Adam optimisation algorithm*, which is an extension of the stochastic gradient descent algorithm discussed in the lessons.

This is the part in charge of updating the weights of the model. The accuracy after training for 50 epochs in the different sessions is summarized in the following table:

1.	2.	3.	4.	5.	6.	7.	8.	9.
0.976	0.968	0.971	0.966	0.974	0.961	0.962	0.976	0.963

10.	11.	12.	13.	14.	15.	16.	17.	18.
0.963	0.969	0.963	0.969	0.972	0.952	0.962	0.972	0.964

Since the accuracy corresponding to each choice of hyperparameters was greater than 0.93, even achieving a maximum of 0.981 (see Ipython Notebook) any choice of hyperparameters could have worked. I decided to train the final model with the following values:

Batch size	Dropout	Learning Rate	Epochs
128	0.7	0.002	33

My final model results were:

Training Accuracy	Validation Accuracy	Testing Accuracy
0.998	0.968	0.948

The testing accuracy is very high and close to the validation accuracy. The validation accuracy is high and close to the training accuracy which implies we are neither overfitting nor underfitting. All these suggest that the model is performing well.

4. Test the model on new images.

In this section I downloaded 11 German traffic signs images from the web, crop them to focus on the actual traffic sign and resize them to obtain 32x32x3 images. The images are then plotted, see code line 26 in the Ipython Notebook. I use the same normalization pipeline as before and transform the data type to float32 as required for the input to my model. Finally I plot again the so obtained 32x32x1 gray-scale images in code line 28 of the Ipython Notebook.

I believed the model would have not done very well if the images wouldn't have been cropped and focused on the traffic sign. That is because most of the training data contained such images. Images with a lot of different objects in them, where the traffic sign is only a small part of the image, were not seen by the model in training and thus would be almost impossible to classify correctly. One way to work around this could be to use similar algorithms as in the advanced lane line finding project to process any image to have the characteristics of the training set.

The images I chose look very similar to the training set and I thus expect a high accuracy on this new data set. However, two images are problematic to classify, namely the first and third image in the plot. These show a "bicycles only" and a "pedestrian crossing" sign. These signs were not seen by the model during training and I was curious how they would be classified. The first sign indeed looks more similar to the circle shaped signs and this is how the classifier also interpreted it. For the "pedestrian crossing sign" I would have however expected that the classifier sees the similarity with the "pedestrian" or "children's crossing" signs in the training set and "interpolate" accordingly. However, this was not the case as seen in the prediction table in code line 32 of the Ipython notebook.

The model was able to correctly guess 9 out of 11 classes, which gives an accuracy of 0.818, as computed in line 33 of the Ipython notebook. This compares favourably to the accuracy

on the test set of 0.948. The two problematic images, the first and third in the plot, were classified incorrectly. The model predictions and the actual sign in each image can be seen in a pandas data frame in the code line 32 of the notebook. To look at how sure the model is about its prediction we compare the softmax probabilities for each prediction. The top 5 softmax probabilities can be found in the code line 34 of the Ipython notebook. The next table shows the first softmax probability for each prediction. The images are indexed as shown in the plot of the images in the Ipython notebook.

Img 1	Img 2	Img 3	Img 4	Img 5	Img 6.	Img 7	Img 8	Img 9	Img 10	Img 11
0.82	0.99	0.99	1.0	1.0	0.99	1.0	0.96	0.99	0.99	1.0

As one can see the model is very sure of its predictions, unfortunately also of the predictions which turned out to be false.

References

[1] Udacity, Self-Driving Car Engineer Nanodegree Program