#**Traffic Sign Recognition**

Build a Traffic Sign Recognition Project

The goals / steps of this project are the following:
* Load the data set (see below for links to the project 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

###Data Set Summary & Exploration

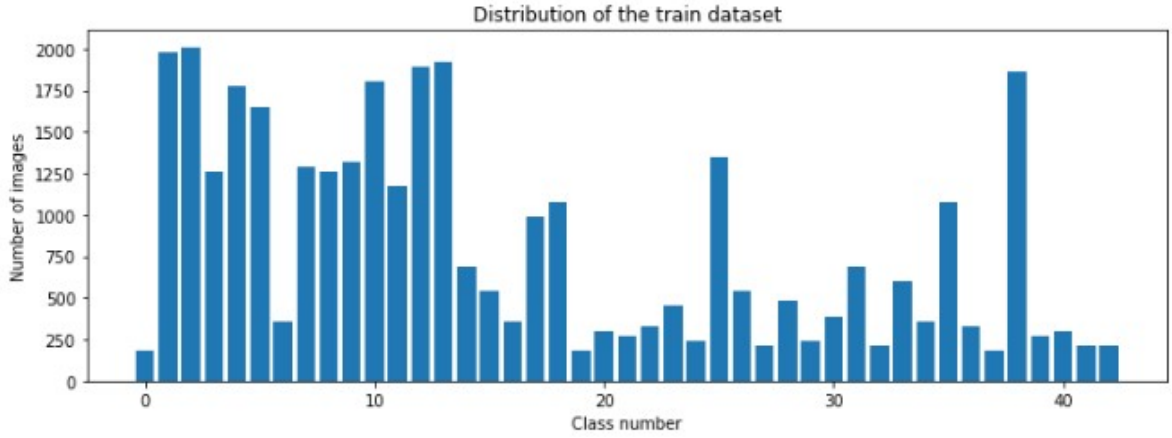
####1.

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 12630
* The size of test set is 4410
* The shape of a traffic sign image is (34799,32,32,3)
* The number of unique classes/labels in the data set is 43

####2. Include an exploratory visualization of the dataset.

Here is an exploratory visualization of the data set. It is a bar chart showing how the data are distributed per classes.



###Design and Test a Model Architecture

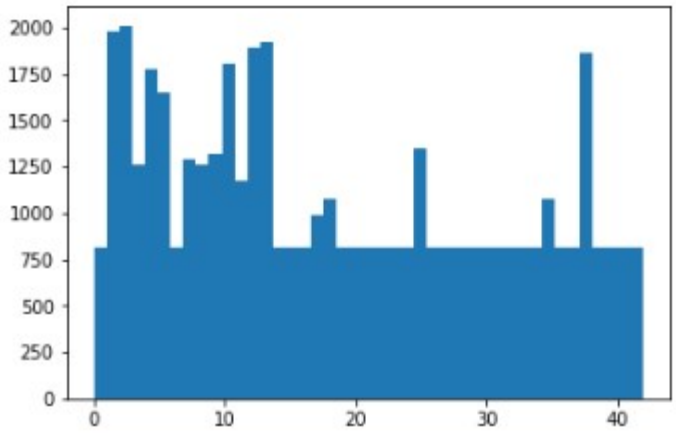
####1.
As first step, I decided to convert the images to gray scale thinking this could have been a good approach making the network process faster as well as targeting the most important details, since the original color images showed unnecessary information.

As last step, I normalized the image data because in order to keep numerical stability in dynamic range for a set of the data, as explained in the lectures, curtaining them between 0.1 and 0.9.

I decided to generate additional data because visualizing the data distribution I noticed that the distribution itself was pretty uneven, where many classes had way below number of samples than mean, leading to an innocent towards majority classes.

Therefore, to add more data to the the data set, I used the rotation techniques, iterating trough existing images and generating new images until enough the sum of additional images overcame the mean.

The difference between the original data set and the augmented data set is the following picture where you can notice that now much more classes are at the same level.



Furthermore, in order to perform the model better I decided to shuffle and split the validation and training data. This way I provided part of the additional data also on the validation set and not just on the training set, avoiding to show good result only in the training part and nothing that special in the test section.

####2. Model architecture

The architecture is based on Le-Net solution, just with some parameters changes...

Layer	Description
Input	32x32x1 grayscale image
Convolution	1x1 stride, Valid padding, output 28x28x9
RELU	
Max pooling	2x2 stride, output14x14x9
Convolution	1x1 stride, Valid padding, output 10x10x27
RELU	
Max pooling	2x2 stride, output 5x5x27
Fully connected	Output 300
RELU	
Fully connected	Output 120
RELU	
Fully connected	Output 43
Softmax	

####3. Describe how you trained your model.

Instead using the classical Gradient Descent Optimizer I discovered the existence of AdamOptimezer which shown better performances. In order to achieve the best outcome I chose 150 as batch size, which represented the right compromise between smaller or bigger values of it. About the epochs, I ran different numbers of epochs, but I noticed that the the validation accuracy settles within 10 epochs. Therefore, I chose not to increase this value; in my opinion you should demonstrate to reach good result of your model with few iterations.

I haven't changed the hyperparamters because I didn't see improvements, on the contrary ramping sigma up I noticed an increase of the training time.

An interesting approach is about the learning rate, which I changed for every epoch, casting it to a minimum value of 0.001 and starting from a max value of 0.008. I used this solution in order to accelerate the training during initial phase smoothing it step by step, getting better results without deriving or create instability as mentioned in the lectures.

As dropout I chose a value of 0.7 instead of the common 0.5 used in the Le-Net solution, which prevent my model from overfitting in a better way.

####4. Describe the approach taken for finding a solution and getting the validation set accuracy to be at least 0.93.

In my final model I reached a validation accuracy of 0.990 and a testing accuracy of 0.912. In particularly the latter demonstrated that my model wasn't affected from overfitting.

As I mentioned before I didn't changed the Le-Net architecture in general, I tried to maintain the original scaffolding, changing just the depth of the filters and the size of every patch, with the aim to target important details from the pictures.

###Test a Model on New Images

####1.

Here are ten German traffic signs I found on the web:


gettyimages
FrankRamspott


50
gettyimages


alamy


iStock
by Getty Images

I found it difficult to classify the "Pedestrians" and "End of all limits". The elaboration of the first is completely wrong, I think that the inner shape and the complexity of the sing has lead the model to such huge mistake.

About the second one the colour of the sign can be attributed to priority road, so we can say they belong to the same family in somehow. It is not justified on the other hand the failure to recognize the shape.

A possible improvement might be introducing a new layer in the covnet in order to fill the gap concerning this aspect, with the intent to detect more important characteristics of these type of signs. An inception layer might do the trick.

Further, at first, I was little biased about the correct classification of the ciphers of the speed limit sings, nonetheless my concerns, it seems that the model works pretty good over this category.

####2. Discuss the model's predictions on these new traffic signs and compare the results to predicting on the test set.

Here are the results of the prediction:

Image	**Prediction**
Wild animals crossing	Wild animal crossing
Right of way at the next intersection	Right of way at the next intersection
Yeld	Yeld
Speed limit (50 km/h)	Speed limit (50 km/h)
Pedestrians	General caution
End of all limits	Priority road
Priority road	Priority road
Keep right	Keep right
Speed limit (30 km/h)	Speed limit (30 km/h)
Road work	Road work

The model was able to correctly guess 8 in 10 traffic signs, which gives an accuracy of 80%. I think the things might change if the prospective of the image changes and also if the complexity of the sign enhances. Anyway I'm quite satisfied about the correct classification of the speed limit sings, which as I mentioned before was my biggest concern.

####3.

The code for making predictions on my final model is located in the 61th and 62th cell of the Ipython notebook.

For the first the model is rather sure that this is a **Right of way at the next intersection** sign, probability of 1.0.

Probability	Prediction
1.0	**Wild animal crossing**
0.0	Speed limit (30 km/h)
0.0	Speed limit (80 km/h)
0.0	Speed limit (50 km/h)
0.0	Double curve
0.0	Speed limit (60 km/h)

For the second the model is rather sure that this is a **Right of way at the next intersection**, probability of 1.0.

Probability	Prediction
1.0	**Right of way at the next intersection**
0.0	Priority road
0.0	Children crossing
0.0	Beware of ice/snow
0.0	Double curve
0.0	Pedestrians

For the third the model is rather sure that this is a **Yeld,** probability of 1.0.

Probability	Prediction
1.0	**Yeld**
0.0	Keep left
0.0	Ahead only
0.0	No passing
0.0	Turn right ahead
0.0	Go straight or right

For the fourth the model is rather sure that this is a **Speed Limit (50 km/h),** probability of 1.0.

Probability	Prediction
1.0	**Speed Limit (50 km/h)**
0.0	Speed Limit (80 km/h)
0.0	Speed Limit (100 km/h)
0.0	Speed Limit (60 km/h)
0.0	Speed Limit (30 km/h)
0.0	Priority road

For the fifth the model is rather sure that this is a **Pedestrians**, probability of 1.0, but it is **completely way off base**, even if we look at the other probabilities. My model definitely doesn't work with this sign.

Probability	Prediction
1.0	**General caution**
0.0	Traffic signals
0.0	Go straight or right
0.0	Keep right
0.0	Go straight or left
0.0	Dangerous curve to the right

For the sixth the model is quite sure that this is a **Priority Road**, probability of 1.0, but it is **completely way off base**, even if we look at the other probabilities. My model definitely doesn't work with this sign.

Probability	Prediction
0.87	**Priority Road**
0.1	Roundabout mandatory
0.03	Right of way at the next intersection
0.0	Speed Limit (100 km/h)
0.0	General caution
0.0	Roundabout mandatory

For the seventh the model is quite sure that this is a **Priority Road**, probability of 0.98.

Probability	Prediction
0.98	**Priority Road**
0.02	Roundabout mandatory
0.0	Speed Limit (60 km/h)
0.0	Keep right
0.0	End of all limits
0.0	Speed Limit (80 km/h)

For the eight the model is quite that this is a **Keep right**, probability of 1.0.

Probability	Prediction
1.0	**Keep right**
0.0	Stop
0.0	Turn left ahead
0.0	Road work
0.0	Yeld
0.0	Go straight or right

For the ninth the model is quite sure that this is a **Speed Limit (30 km/h)**, probability of 1.0.

Probability	Prediction
1.0	**Speed Limit (30 km/h)**
0.0	Speed Limit (80 km/h)
0.0	Speed Limit (50 km/h)
0.0	Speed Limit (60 km/h)
0.0	Speed Limit (20 km/h)
0.0	Speed Limit (100 km/h)

For the tenth the model is quite sure that this is a **Road work**, probability of 1.0.

Probability	Prediction
0.99	**Road work**
0.01	Bicycles crossing
0.0	Bumpy road
0.0	Wild animal crossing
0.0	Traffic signals
0.0	Road narrow on the right