Traffic Sign Recognition

The purpose of this document is to provide support for the developed work regarding Udacity's Self Driving Car Nanodegree project 3, titled as "Traffic Sign Classifier".

In this document we will go through the development process to come up with a convolutional neural network capable of correctly classifying German traffic signs.

We will start by having a quick glance at what signs we are looking at, how much data we have available for training and how is the data distributed. Then, we will explore how the model architecture was built to correctly classify the traffic signs. At the end we will explore the results of the model and explore it's vulnerabilities and future improvements.

Data Set Summary and Exploration

1. <u>Provide a basic summary of the data set. In the code, the analysis should be done using python, numpy and/or pandas methods rather than hardcoding results manually.</u>

   In order to obtain a model that classifies traffic signs, we first need to have a lot of data available to use as input in the model. Thankfully, Udacity already provided the datasets as a starter, so it was not necessary to grab data manually.
   The data was already divided into three different types of datasets, all of them containing images of traffic signs and respective labels (label is the identifier of a traffic sign):
   - Training data set – this is the dataset that will be used as input while training the model. It contains 34799 images of traffic signs and respective labels.
   - Validation data set – this is the dataset that will also be used while training, but will not be used as training. It will be used to ensure that the model is not overfitting or underfitting while training. It contains 4410 images of traffic signs and respective labels.
   - Test data set – this is the dataset that will be used after training, to ensure that the model is correctly training. A high accuracy of the model while running the test data set ensures a good training, as the model as never seen these images before. A lower accuracy implies that the model did not train well and it decorated the images instead while training. This data set contains 12630 images of traffic signs and respective labels.

   All datasets contain images of the same shape. In this case, the shape of the image is:

   (32,32,3). What does this mean?

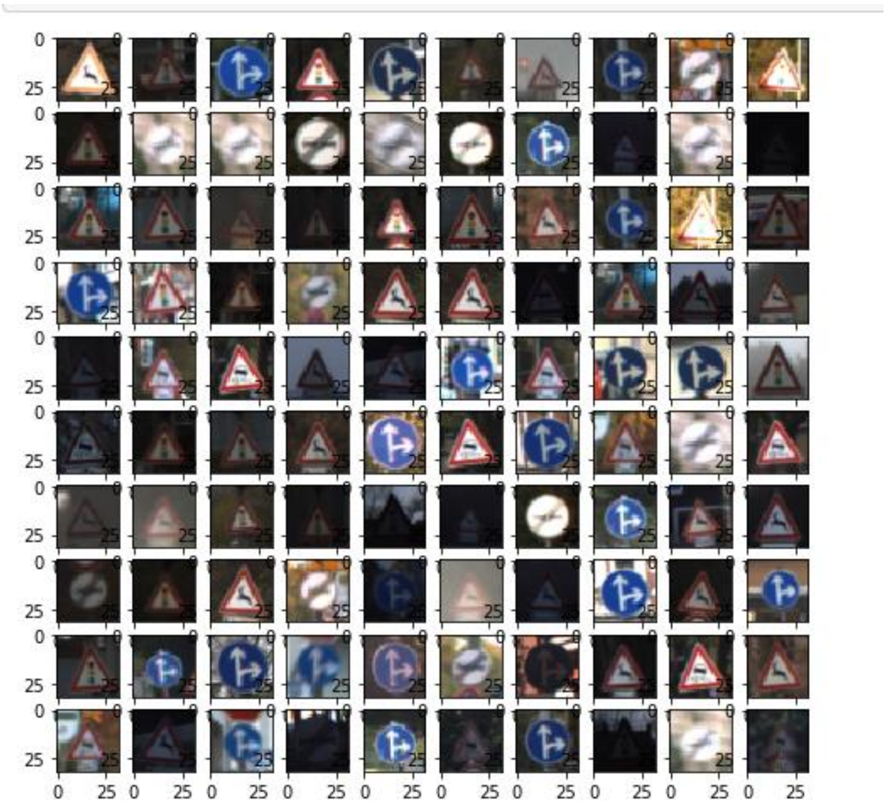
   It means that it has a width and a height of 32 pixels, and the 3 is related to the number of color channels. In this case, 3 for RGB (Red, Green and Blue).

   In this project, we are working with 43 different types of traffic signs.

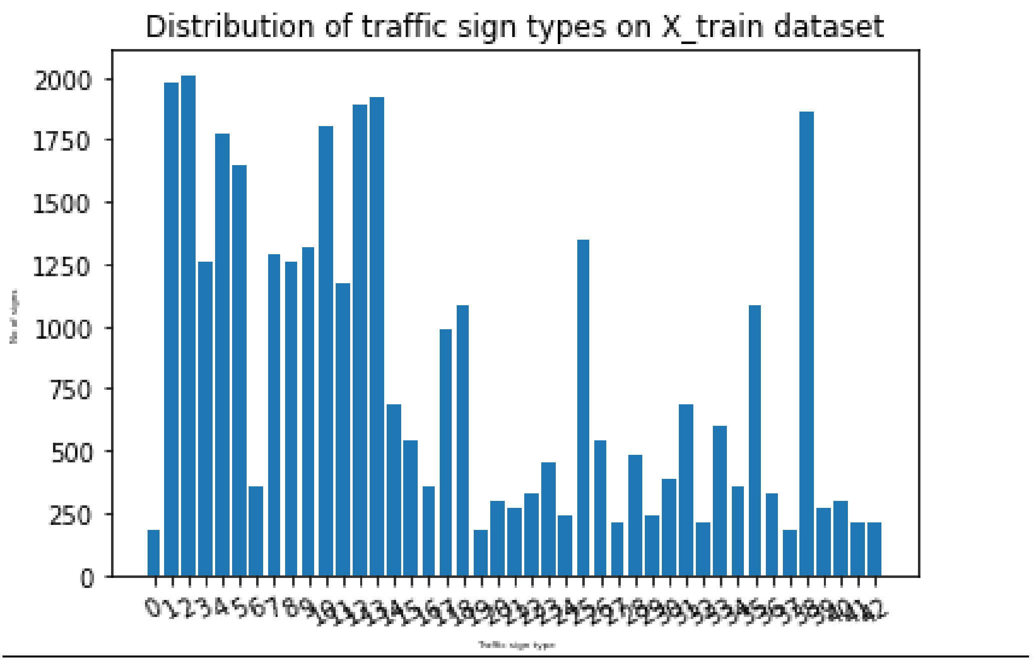
   If you are curious about the signs, please check the provided file by Udacity on my github.

2. Include an exploratory visualization of the dataset.

   In order to get familiar with the data that is going to be used to train the model, an exploratory visualization of the model was done.



   In the image above, we can see 100 random images that where taken from the training dataset. As we can observe from this sample, all the images appear to have the traffic sign in the center of the image, without much deviation.
   Lighting conditions in the other hand look different from image to image. Different types of lighting conditions should add difficulties when classifying the same traffic sign with different lighting. As it will be explained later in this report, one of the steps of preprocessing the images is to grayscale the image to avoid this issue.



In the image above we can see a distribution of the different traffic signs across the training dataset. The x-axis represents the type of traffic sign, and the y-axis represents the number of traffic signs of that class.
If we look at the provided file by Udacity, we can see that the label "1" represents a traffic sign of "Speed Limit to 30km/h" and that the label "42" represents "End of no passing by vehicles over 3.5 metric tons".
By looking at the bar chart above, we can see that we have nearly 2000 samples of the speed limit sign, but for the other sign we have less than 250 samples, almost 90% less samples.
This is important to address because even with a high accuracy of the model, we should expected a much higher confidant of correctly classifying speed limit signs when comparing to classifying "end of no passing" signs.

Design and Test a Model Architecture

In this section it is explained in more detail the process used to train the model.

The majority of the model was inspired in the previous lessons regarding the LeNet model implementation.

1. Preprocessing data

   For this project, 2 preprocessing steps were included.

   As already referred in the section above, different lighting conditions were present in the data for the training set. While this is really good to classify real world case scenarios, as there are roads with all kinds of lighting conditions during different parts of the day, it becomes harder to analyse the image in such different conditions. However, if we think about it, the color of the sign is important, but is not imperative to use it. At this stage, the shape of a sign should give us everything we need.

Having that is mind, the first step of preprocessing the data was to transform the image in a grayscale image.

This was possible by using a python computer vision library called "OpenCV", like this:

cv2.cvtColor(img, cv2.BGR2GRAY)

The second step of preprocessing training data before running the model was to improve the contrast in the image after grayscaling the image. While the grayscale step solved the issue of the lighting conditions, it added unnecessary noise to the image. It might be difficult for the model to identify where the sign ends and the road start due to the slight differences in color. A histogram equalization takes in a grayscale image and helps to identify "boundaries" in an image, improving contrast and therefore, helping to clearly separate the traffic sign from the road.

This was also possible by using OpenCV, in the following way:

cv2.equalizeHist(img)

The third and last step of preprocessing image data was to normalize the image to help the network to converge faster and obtaining better results.

2. Model architecture

The model architecture was based in the LeNet model and improved to the specific problem to be solved, until 93% or higher accuracy was found.

The following table describes the model architecture.

Layer	Description
Input	32x32x1 image
Convolution 5x5	1x1 stride, valid padding, outputs 28x28x30
Relu	Activates the convolution above
Convolution 5x5	1x1 stride, valid padding, outputs 24x24x30
MaxPooling	2x2 stride, outputs 12x12x30
Convolution 5x5	1x1 stride, valid padding, outputs 8x8x16
Relu	Activates the convolution above
MaxPooling	2x2 stride, outputs 4x4x16
FullyConnected	Flattens the inputs in a single array of 256 pixels
FullyConnected	Receives the previous 256 weights and returns 120
Relu	Activates the previous fully connected layer
FullyConnected	Receives the previous 120 inputs and returns 84
Relu	Activates the previous fully connected layer
FullyConnected	Receives the previous 84 inputs and returns 43 (one for each traffic sign type)

3. How has the model trained

To train the model, we could use some hyperparameters and choose how to proceed.

Given that this is a classification neural network, it is necessary to use a loss function that responds to this type of neural networks. A combination of cross entropy and AdamOptimizer were used.
A learning rate of 0.001 was used to train the model.
The batch size of 128 and 10 epochs were left untouched as they looked to work well for the project.

4. Results obtained

The model was trained several times until an accuracy higher than 93% was obtained. Training the LeNet model without any changes gave an accuracy of 89%. Although good, was not enough for the current project requirements.

My first attempt of modifying the original LeNet model was to add an additional convolution layer. After training, the accuracy didn't improve. Instead, it dropped.

From the lessons it was possible to verify that in recent years the max pooling layer is being discredited in regards to the dropout layer. I think it seems logic, because a max pooling layer reduces the pixels size in a half without much analysis. So I tried to used dropouts where the max pooling layers where before, but I failed to obtain a better model. I don't have a correct answer for why that happened.

The third step that I tried to improve the model accuracy and that in this case it seemed to work, was to start the first convolutional layers with more filters, 30, instead of the original LeNet of 6 filters.

In a very high level logic, it makes sense because a traffic signs should have more than just 6 "features" that make a sign identifiable.

I also rollback to the max pooling layers and added more fully connected layers at the end, and ended up reaching 95.6% accuracy.

To validate a correct training, the model was tested against the test data set.

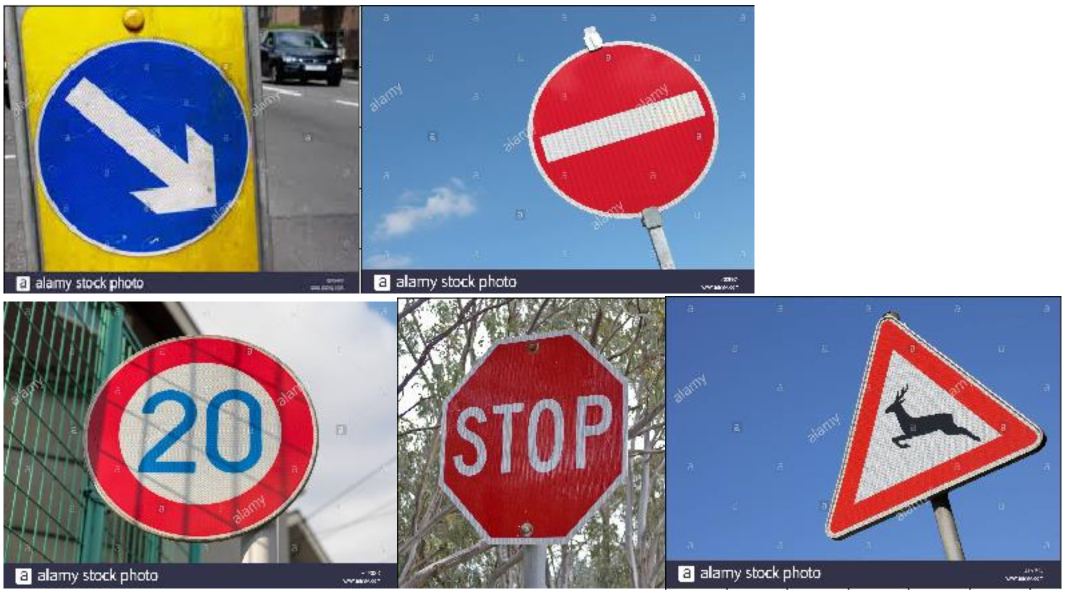
It obtained an accuracy of 93%, which proves a correct training.

Test the model on new images

As part of the project, it was required that the model was tested against new images collected from the web. In the section below we go through the process of testing these images against the model and reach some conclusions about it's credibility.

1. Presenting the images

The images used to test on the model were the following:



In our model, the signs represent:

- Stop sign – label 14
- Speed limit 20km/h – label 0
- Keep right – label 38
- No entry – label 17
- Wild animals crossing – label 31

These images were taken from the web and have all different types of sizes. Because the model is only prepared to receive inputs of shape 32x32x1 images, the presented images need to pass for additional preprocessing steps for resizing. In the jupyter notebook code, such code can be found on the function “preprocess_web_image”.

2. Results obtained

When running these images against the model, only 40% accuracy was obtained. This means that 2 images were correctly classified, and 3 were wrongly classified.

What went wrong?

These images have a direct issue that the ones on the training dataset didn’t. These images have watermarks and copyright trade information that add noise to the image and might have cause difficulties classifying. The images found on the web are also not as well centered as the images on the training dataset. Probably data augmentation by rotating the image, changing the angles and the center would dramatically help in this step.

3. Discuss top 5 softmax probability

To further analyse what went wrong in the previous step, an additional step was done to verify what was the actual classification of the image, by calculating the top 5 predictions for the image.

Stop sign

Prediction	Probability1

14-Stop	99.9%
34-Turn left ahead	0.1%
33-Turn right ahead	0.0%
13-Yield	0.0%
3-Speed limit 60 km/h	0.0%

As we can see, the stop sign was correctly classified. This image was the only image taken from the web without watermarks and copyright trade symbols. It was also fairly centered with the traffic sign, and overall the conditions for classification are good. The model correctly classified the image with almost 100% probability.

Speed limit 20km/h

Prediction	Probability
15-No vehicles	96.5%
8- Speed limit 120 km/h	3.4%
7- Speed limit 100km/h	0.1%
5- Speed limit 80km/h	0.0%
39- Keep left	0.0%

The model incorrectly classified the speed limit sign to 20km/h as a no vehicles sign.

A possible explanations is the fact that the format of the signal is the same, except for the speed indication. The fact that the sign has the speed limit in blue might have affected the classification. From the initial distribution and data analysis of the training dataset, we can also see that this specific traffic sign had fewer samples to train, about 250 samples. This was a sign with a lower spectrum of samples and that also contributes for this inaccuracy when testing against new images.

Keep right sign

Prediction	Probability
38-Keep right	99.7%
25-Road work	0.3%
20-Dangerous curve to the right	0.0%
34-Turn left ahead	0.0%
31-Wild animals crossing	0.0%

Here the model behaved as predicted, which is good.

No entry sign

Prediction	Probability
11- Right-of-way at the next intersection	99.3%
39-Keep left	0.7%
5-Speed limit 70km/h	0.0%
33- Turn right ahead	0.0%
19-Dangerous curve to the left	0.0%

There was nearly 1000 samples for the no entry sign in the training dataset, and still the sign was misclassified to one that is not particularly similar.

Trademarks and watermarks in the image probably confused the model when classifying.

Wild animals crossing sign

Prediction	Probability
14-Stop	97.9%
12-Priority road	2.1%
18-General caution	0.0%
20-Dangerous curve to the left	0.0%
40-Roundabout mandatory	0.0%

Again, the model misclassified the wild animals crossing sign as a stop sign.

Further Improvements

This project was really great to get introduce to a real world application of convolutional neural networks.

This is my second nanodegree at Udacity. I have previously completed the Computer Vision nanodegree, so Convolutional neural networks was not something completely new to me.

It was really nice though to see a true application for self driving cars.

For further improvements, there are still a lot to do for this project to be approved to be used in a real self driving car.

As we could clearly observe, 40% accuracy in images that differ a little from the original images is not enough to be production ready.

Data augmentation must be added to the data preparation. Things like uncenter the traffic sign from the image, rotating images, change angles and other changes must be added improve accuracy in production scenarios.