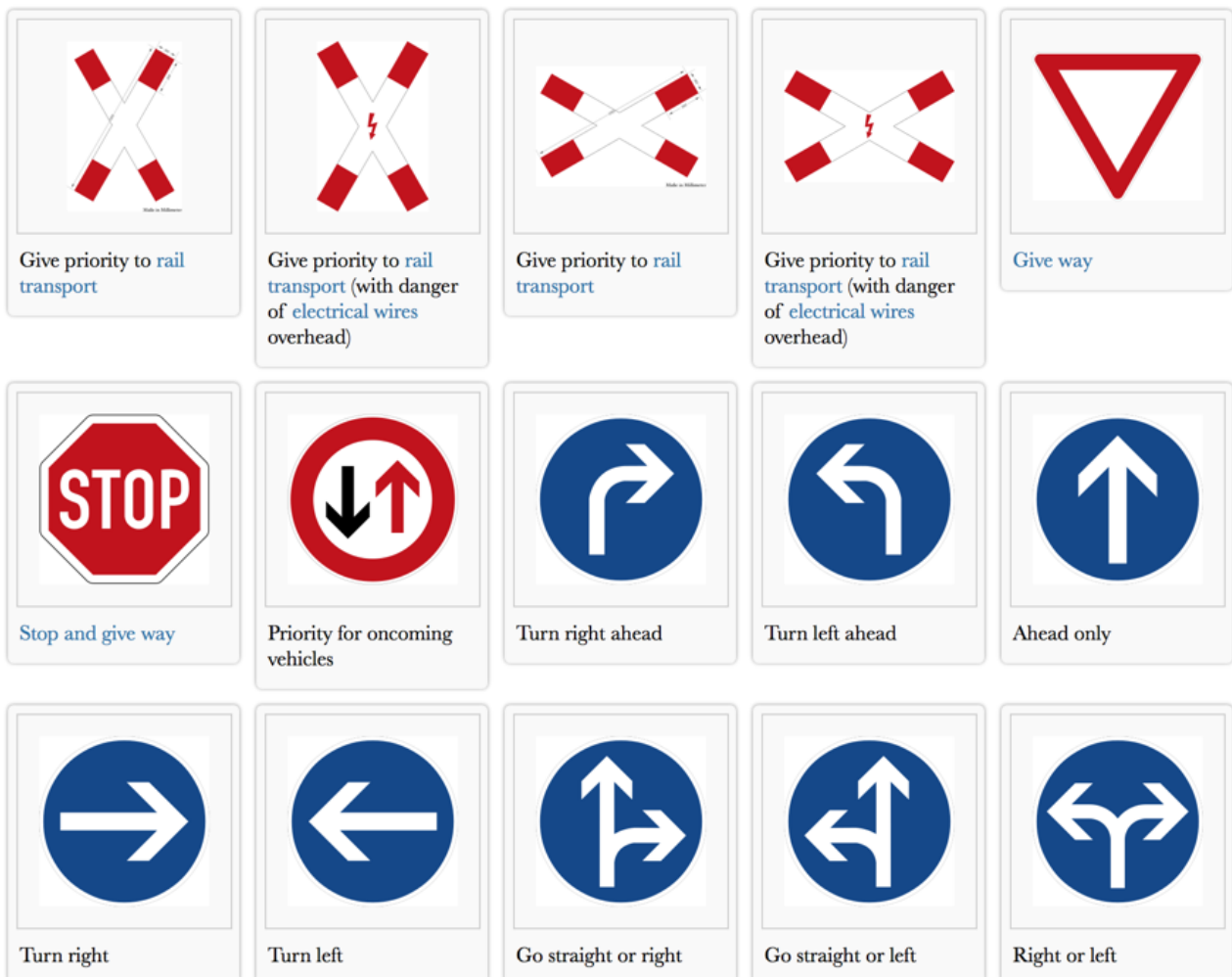# Traffic sign classifier

Self driving car is becoming a reality, soon we are going to see self driving cars on the city street. Deep Learning, especially Convolutional Neural Networks has increased the speed of the transformation. Self driving cars should take many things into consideration pedestrians, other cars, road, weather conditions and traffic signs. We are going to take a closer look that how we can implement a CNN architecture to interpret the traffic signs.

Data Set Summary & Exploration

Traffic rules may differ from country to country. For example, English uses left-hand side of the road while other European countries use right-hand side of the road. Traffic signs are not an exception, they might also change from country to country. We are going to use German road sign icons.

You can find the dataset below;

http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset

Give priority to rail transport	Give priority to rail transport (with danger of electrical wires overhead)	Give priority to rail transport	Give priority to rail transport (with danger of electrical wires overhead)	Give way
Stop and give way	Priority for oncoming vehicles	Turn right ahead	Turn left ahead	Ahead only
Turn right	Turn left	Go straight or right	Go straight or left	Right or left

You can find some examples from German Traffic sign icons, below;

Although the dataset contains a lot of icons, we are going to assume that the dataset contains only 43 icons and we are going to train our system with these icons. Once we have developed a successful model, it could be trained whole set, even with the traffic signs for other countries. The model should work fine with them as well.

You can download the dataset below.

https://d17h27t6h515a5.cloudfront.net/topher/2017/February/5898cd6f_traffic-signs-data/traffic-signs-data.zip

The data should be split into training, validation and test sets.

X_train, X_validation, y_train, y_validation = train_test_split(X, y, test_size=0.20, random_state=42)

But, you don't have to split the data into the train, validation and test sets in this case. It is already divided into three parts as Train.p, Valid.p, Test.p. You can use them as well.

We will apply LeNet architecture which expects image dimension as 32x32. The dataset already consists of the 32x32 images, so we do not need to reshape the images. You can find some other statistics of data set, below:

Number of training examples : 27839

Number of validation examples : 6960

Number of test examples: 12630
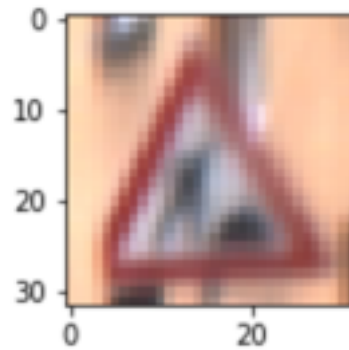
Image data shape:32x32x3 (Color Image)

The Number of Unique Class: 43

You can find one of the images from the dataset.

plt.figure(figsize=(2,2))

plt.imshow(image)

print(y_train[index])

Design and Test a Model Architecture

Data Preprocessing

First, we are going to apply normalization to the images in order to make zero mean and equal variance. You can apply normalization with the below given formula.

```
(pixel - 128)/ 128
```

Lets see an image before and after the normalization. Below, you can see image prior to normalization at left-hand side, and after normalization right-hand side.
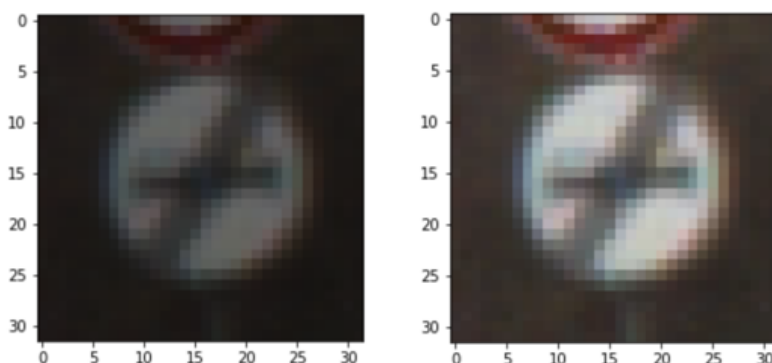
I have applied many data augmentation techniques such as rotate, flip, translate, affine, perspective transformation, convert to gray scale etc. During experimenting on data augmentation, I realized some of them did not increase accuracy so I had dropped these techniques. You can still find the dropped augmentations on Traffic_Sign_Classifier_Keras.ipynb

I have applied rotation, flipping horizontally and vertically, and translation to the images. There are two main reason why I have applied data augmentation techniques to increase date set size. First, the data set is not a huge one, it contains considerably less images. I would like to increase the size of the training dataset in order to precent overfit. Also, Traffic signs could be viewed from different angles and their position might be changed in real world applications. So, before training I preprocessed the images to prevent overfit the data set. After preprocess, the trained model would work more successfully with unseen data.

Here is an example of a traffic sign image before and after image rotation. prior to process: Left-hand side, after the process: right-hand side.
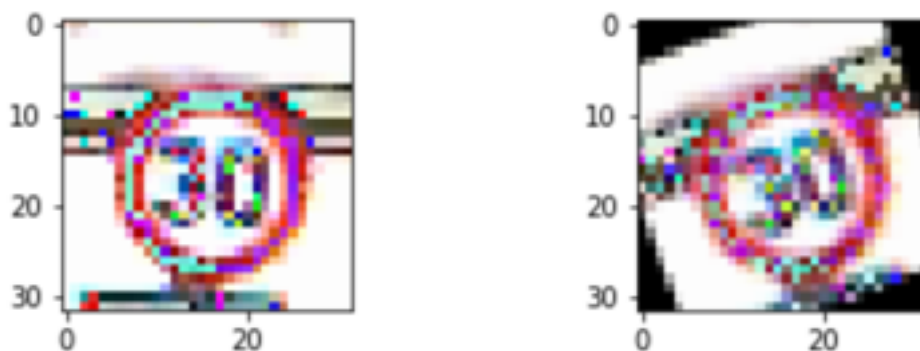
Here is an example of a traffic sign image before and after flip horizontal. prior to process: Left-hand side, after the process: right-hand side.
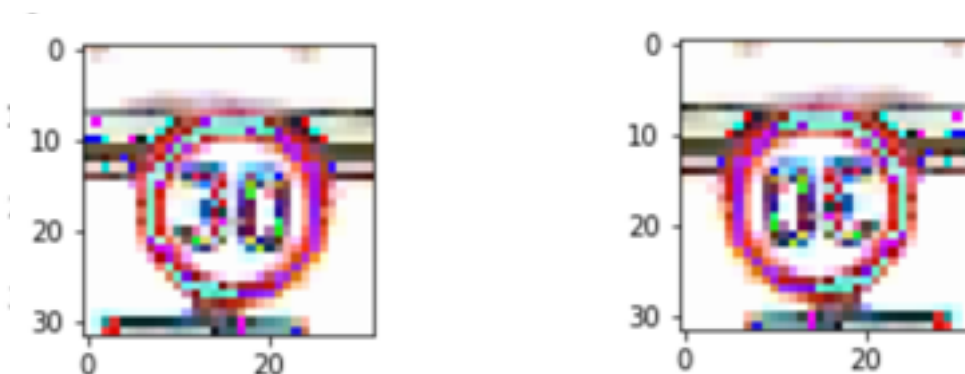
Here is an example of a traffic sign image before and after translation. prior to process: Left-hand side, after the process: right-hand side.
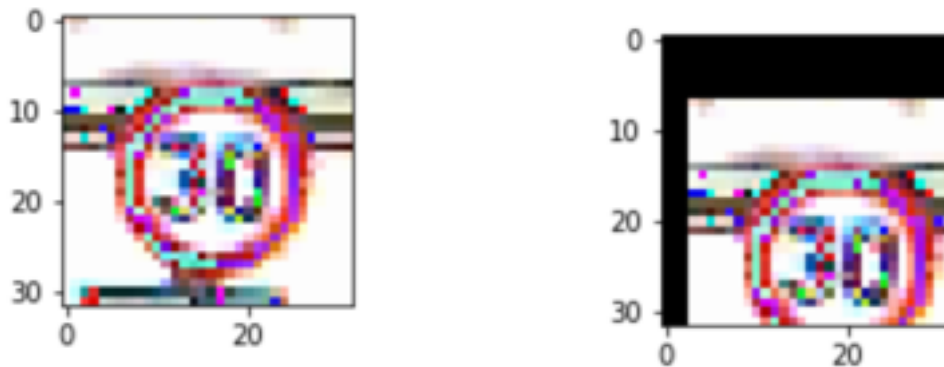
Model Architecture

I used LeNet architecture as the base of the model architecture. You can find the details for each layers of the architecture below:

Layer	Description
Input	32x32x3.
Convolution 5x5 filter	1x1 stride, valid padding, outputs 28x28x6
RELU	
Max pooling	2x2 stride, outputs 14x14x6
Convolution 5x5 filter	1x1 stride, valid padding, outputs 10x10x16
RELU	
Max pooling	2x2 stride, outputs 5x5x16
Flatten	Output = 400
Fully Connected	mean = 0, stddev = 0.1, Output = 120
RELU	
Fully Connected	mean = 0, stddev = 0.1, Output = 84
RELU	
Fully Connected	mean = 0, stddev = 0.1, Output = 43

Final output size is 43 which represents 43 unique class of german traffic signs.

Training

Training the model was a hard process. I aimed at to achieve a accuracy mode than 93%. I had managed to measure an accuracy more than 93% easily. But, on validation data set, I have struggled for days and I did not make any progress.

I am more experienced with Keras. So, first I tried to implement an architecture with keras. I have used data augmentation techniques and I increased the validation data set size enormously. I also used huge EPOCHS sizes. Than, it was really taking long time to train data. It lasted two - three hours to train a model. And, I had really disappointing results.

Then, I decided to start from scratch and use bare TensorFlow. I have used LeNet architecture as a base and I have made slight necessary changes. I have experiment on hyper-parameters until I found the best combination. You can find some hyper-parameter values below:

Number Of Epochs = 26

Batch Size = 64

Optimizer Type: Adam

Learning Rate: 0.001

In order to achieve 93% accuracy, I have chosen a well known architecture: LeNet. The German traffic signs dataset consists of 32x32x3 images, LeNet architecture is also accepting 32x32 images. So, It was easy to apply this data set to LeNet. Also, LeNet is performing well for this kind of image classification, so it was a very suitable candidate. First, I have tried to design an architecture from scratch but i could not even come closer to 93%. And, I have lost close to one week. Afterwards, I decided to use an well known architecture and make improvement on it.

My final model results were:

training set accuracy: 0.973

validation set accuracy: 0.943

test set accuracy: 0.745

Test a Model on New Images

I have downloaded some images from internet to analyze prediction accuracy for images which the model had never seen before. You can find the five images which I downloaded from internet in the below table;

11	Right-of-way at the next intersection	
12	Priority road	

27	Pedestrians	
34	Turn left ahead	
35	Ahead only	

First two images might be difficult to classify correctly because there are some other class which are very similar to them. Also, CNN first recognize the pattern and they have very common patterns; first they are in a circle shape, second the arrow signs are very common. Also, the color is very common as well.
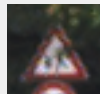
Predictions

Number	Image	Is Predicted Right?	Prediction
11	Right-of-way at the next intersection	No	[26]
12	Priority road	Yes	[12]
27	Pedestrians	No	[18]
34	Turn left ahead	No	[14]
35	Ahead only	No	[25]

Probabilities

Number	Image	First Probability	Precition
11	Right-of-way at the next intersection	0,998	26
12	Priority road	0,969	12
27	Pedestrians	0,541	18
34	Turn left ahead	0,984	14
35	Ahead only	0,424	25

First image classified wrong.

When we looked the images, we realize all of them has triangular shape. It means the model recognized the shapes and colors. It just could not recognize the shape inside the triangular.

Probability	Predictions	Images
9.98241782e-01	26	
9.64424864e-04	28	

Probability	Predictions	Images
7.63255521e-04	18	
2.13614494e-05	19	
6.71669295e-06	27	

Second image classified image.

The image has classified with a very high certainty.

Probability	Prediction
9.69958425e-01	12
3.00377645e-02	25
1.30446858e-06	4
1.29490729e-06	10
1.09004600e-06	20

Third Image classified wrong

Again, most of the image has triangular shape. The predicted images not the right one but close to the right one.

Probability	Prediction
5.41408718e-01	18
4.46163237e-01	3
1.07901664e-02	31
1.45748956e-03	26
8.78783685e-05	29

Forth Image classified wrong

Two of the predictions are circular shape as the real image. This time predictions are not very close.

Probability	Prediction
9.84079897e-01	14
8.78307410e-03	13
3.38625419e-03	25
1.46176631e-03	29
6.61018654e-04	1

Fifth image classified wrong

Although, the image classified wrong. The right one(35) in the top five prediction.

Probability	Prediction
4.24114227e-01	25
3.10869277e-01	18
2.59938747e-01	14
3.93524067e-03	26
1.10466999e-03	35