Project : Traffic Sign Classifier

23.12.2018

Submitted By:

Lalu Prasad Lenka

Overview

The project aimed at creating a pipeline using deep neural networks and convolutional neural networks to classify traffic signs.

Goals

1. Data Summary.
2. Exploratory Visualisation.
3. Preprocessing.
4. Data Augmentation.
5. Model Architecture.
6. Model Training.
7. Solution Approach.
8. Acquiring New Images.
9. Performance on New Images
10. Model Certainty - Softmax Probabilities

Pipeline(Image)

I. Data Summary

A. The model is trained on [German Traffic Sign Dataset](), which has 4 key-value pairs.

B. 'features' is a 4D array containing raw pixel data of the traffic sign images, (num examples, width, height, channels).

C. 'labels' is a 1D array containing the label/class id of the traffic sign. The file signnames.csv contains id -> name mappings for each id.

D. 'sizes' is a list containing tuples, (width, height) representing the original width and height the image.

E. 'coords' is a list containing tuples, (x1, y1, x2, y2) representing coordinates of a bounding box around the sign in the image. ***THESE COORDINATES ASSUME THE ORIGINAL IMAGE. THE PICKLED DATA CONTAINS RESIZED VERSIONS (32 by 32) OF THESE IMAGES.***

F. Number of training examples = 34799

Number of validation examples = 4410

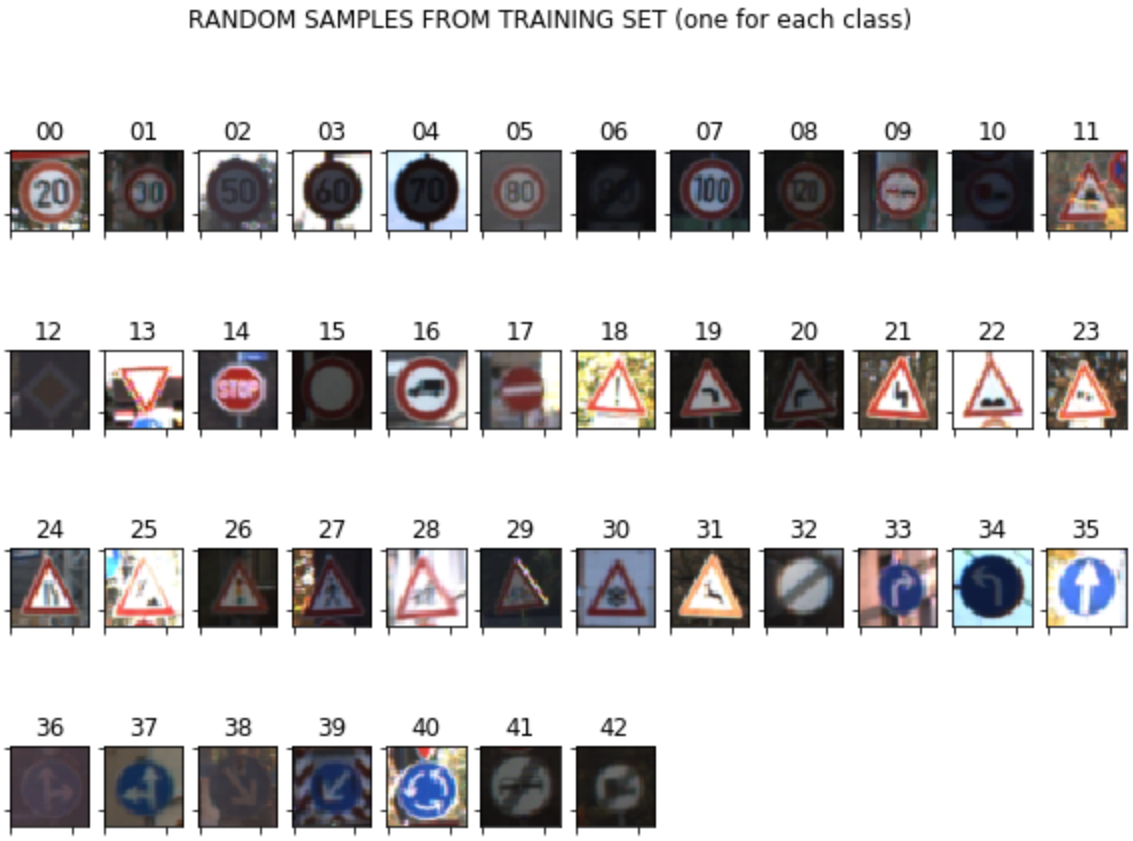
Number of testing examples = 12630

Image data shape = (32, 32, 3)

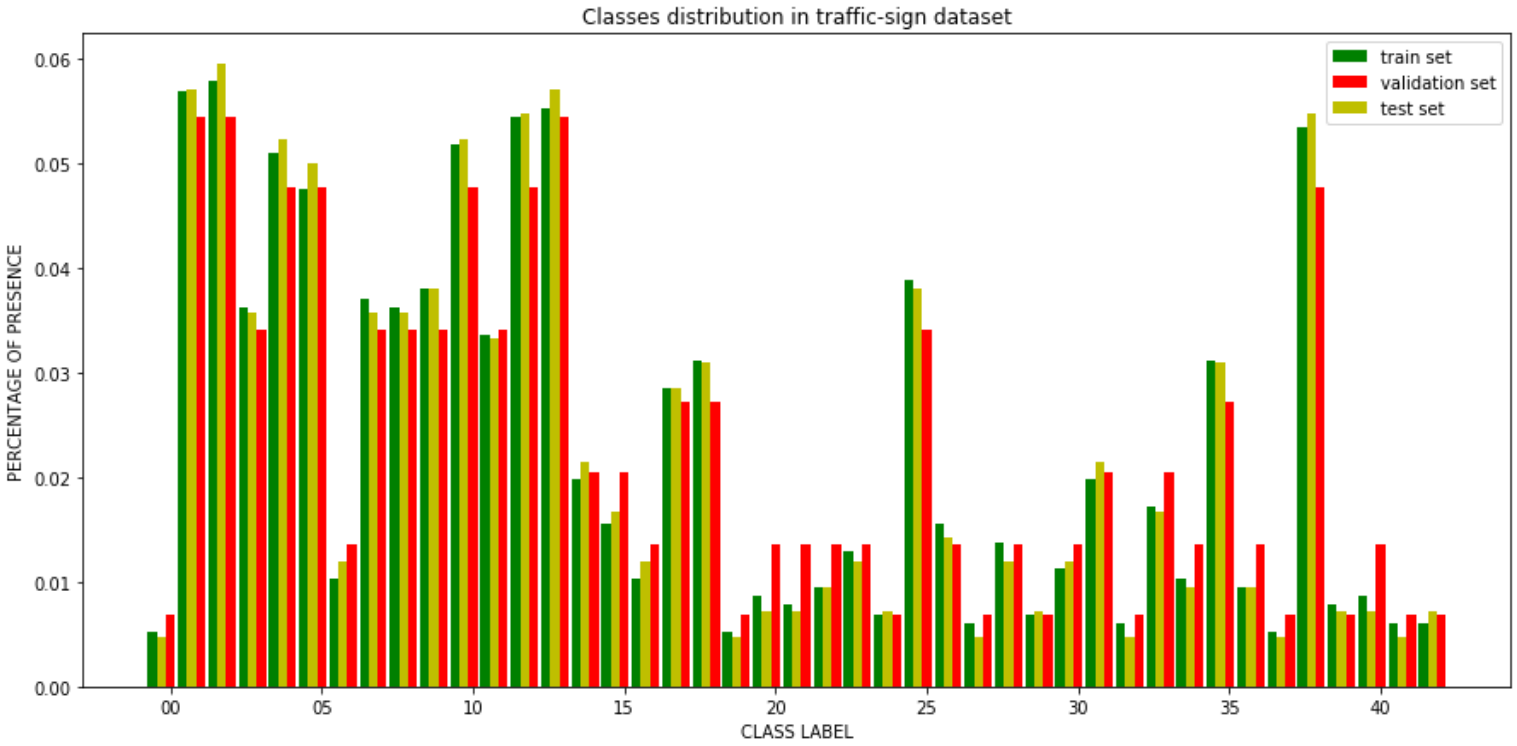
Number of classes = 43

II. Exploratory Visualisation

Visualize some images sampled from training set:



From following plot we notice that there's a strong imbalance among the classes. Indeed, some classes are relatively over-represented, while some others are much less common. However, we see that the data distribution is almost the same between training, validation and testing set, and this is good news: looks like we won't have problem related to dataset shift when we'll evaluate our model on the test data.



III. Preprocessing

Following this paper [Sermanet, LeCun] I employed three main steps of feature preprocessing:

1) Each image is converted from RGB to YUV color space, then only the Y channel is used. This choice can sound at first suprising, but the cited paper

shows how this choice leads to the best performing model. This is slightly counter-intuitive, but if we think about it arguably we are able to distinguish all the traffic signs just by looking to the grayscale image.

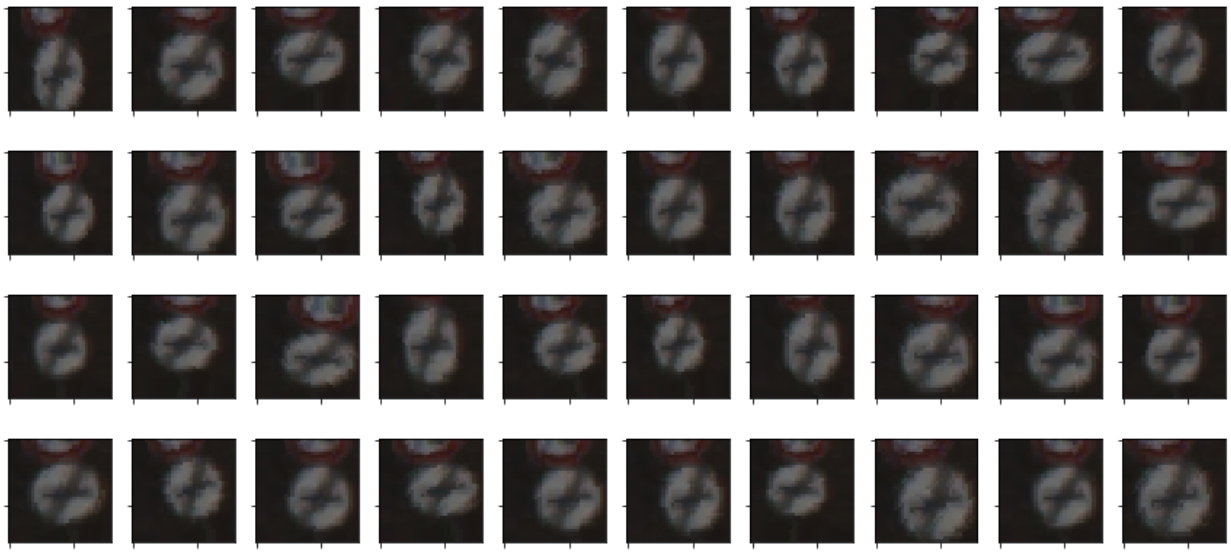
2) Contrast of each image is adjusted by means of histogram equalization. This is to mitigate the numerous situation in which the image contrast is really poor.

3) Each image is centered on zero mean and divided for its standard deviation. This feature scaling is known to have beneficial effects on the gradient descent performed by the optimizer.

IV. Data Augmentation

To get additional data, I leveraged on the ImageDataGenerator class provided in the Keras library. No need to re-invent the wheel! In this way I could perform data augmentation online, during the training. Training images are randomly rotated, zoomed and shifted but just in a narrow range, in order to create some variety in the data while not completely twisting the original feature content. The result of this process of augmentation is visible in this figure.

Random examples of data augmentation (starting from the previous image)



V. Model Architecture

The final architecture is a relatively shallow network made by 4 layers. The first two layers are convolutional, while the third and last are fully connected. Following [Sermanet, LeCun] the output of both the first and second convolutional layers are concatenated and fed to the following dense layer. In this way we provide the fully-connected layer visual patterns at both different levels of abstraction. The last fully-connected layer then maps the prediction into one of the 43 classes.

Layer	**Description**
Input	32x32x1 Grayscale Image
Convolution 3x3	1x1 stride, same padding, outputs 32x32x64
RELU	
Max Pooling	2x2 stride, outputs 16x16x64
Dropout	Keep_prob = 0.5
Convolution 3x3	1x1 stride, same padding, outputs 16x16x128
RELU	
Max Pooling	2x2 stride, outputs 8x8x64
Dropout	Keep_prob = 0.6
Fully connected	Output 64
Dropout	Keep_prob = 0.6
Fully connected	Output 43
Softmax	

VI. Model Training

For the training I used Adam optimizer, which often proves to be a good choice to avoid the patient search of the right parameters for SGD. Batchsize was set to 128 due to memory constraint. Every 5000 batches visited, an evaluation on both training and validation set is performed. In order to avoid overfitting, both data augmentation and dropout (with different drop probability in different layers) are employed extensively.

VII. Solution Approach

The network architecture is based on the paper [Sermanet, LeCun], in which the authors tackle the same problem (traffic sign classification), though using a different dataset. In section II-A of the paper, the authors explain that they found beneficial to feed the dense layers with the output of both the previous convolutional layers. Indeed, in this way the classifier is explicitly provided both the local "motifs" (learned by conv1) and the more "global" shapes and structure (learned by conv2) found in the features. I tried to replicate the same architecture, made by 2 convolutional and 2 fully connected layers.

VIII. Acquiring New Images

I collected five new images from web, some of them had bad lighting and are hazy.

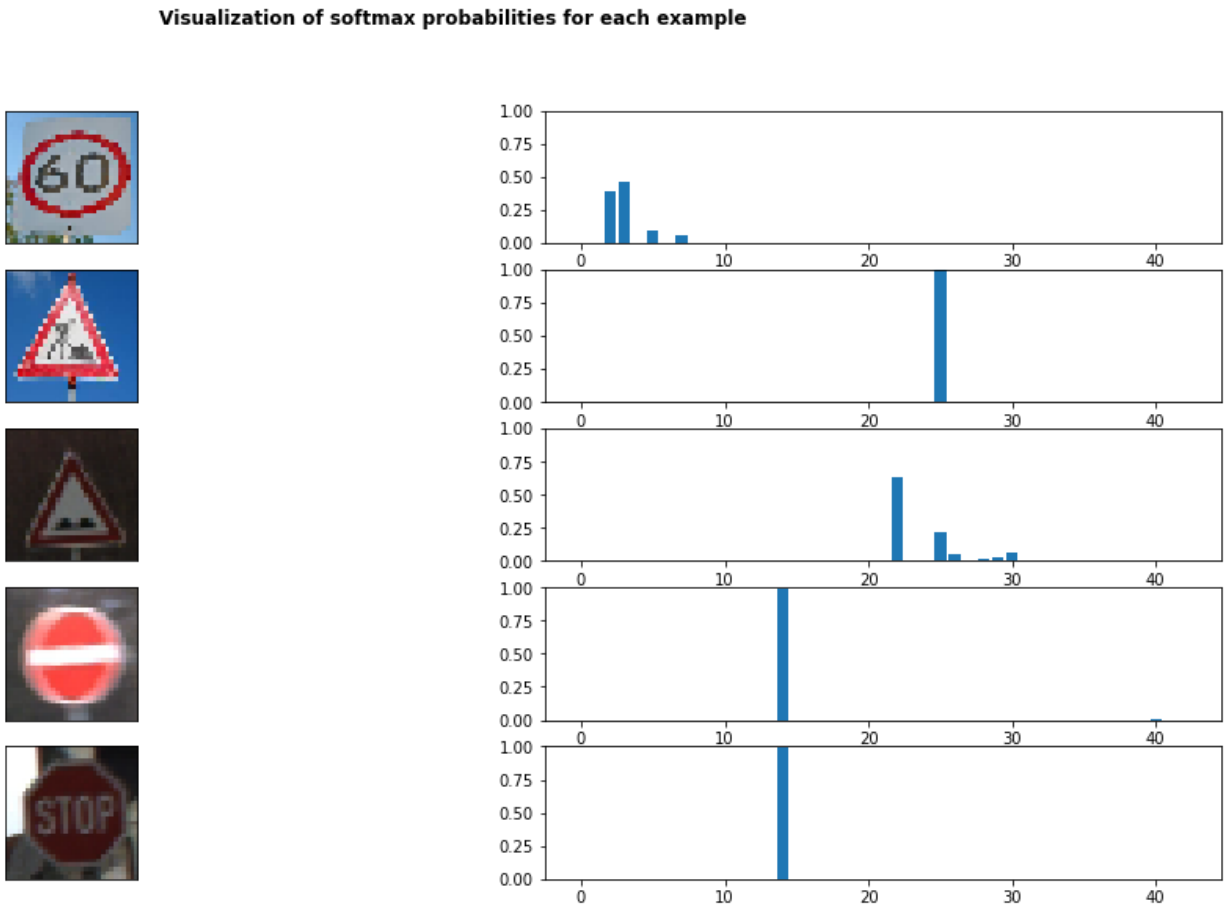
0 - Speed limit (60km/h)
1 - Road work
2 - Bumpy road
3 - No entry
4 - Stop

IX. Performance on New Images

```
Image 0 - Target = 03, Predicted = 03
Image 1 - Target = 25, Predicted = 25
Image 2 - Target = 22, Predicted = 22
Image 3 - Target = 17, Predicted = 14
Image 4 - Target = 14, Predicted = 14
> Model accuracy: 0.80
```

Evaluated on these 5 newly captured pictures, the model accuracy results 80%. While it's true that the performance drop w.r.t. the test set is high (around 15%), we must keep in mind that 5 images are too few to be of any statistical significance.

X. Model Certainty - Softmax Probabilities



As immediately emerges from the plot above, our model is quite sure of which class the second and fifth traffic signs belong, but it's slightly confused on the first and third. Indeed, in the fourth image the model is not only able to predict the right class, but shows also an high confidence, in this case example (no-entry sign) model's prediction is wrong, nonetheless the correct target appears in the top 3 predictions, which is encouraging.

In the first image of (Speed limit (60km/h)) the model got confused with (Speed limit 50km/h) and (Speed limit 80km/h) signs. In third image the confidence for right class is quite higher than other confused classes.