# Self-driving car Nanodegree (Term 1). Project 2: Traffic Sign Classifier

Ruggero Vasile

June 15, 2017

**Abstract**

Here is my report for the second project in the self-driving car nanodegree program, term 1. The project consists in building a deep learning classifier to classify street traffic signs from the German Traffic Sign dataset. The submission consists of this report pdf file and the correspondent jupyter notebook.

## 1 Introduction

In this report I collect all the relevant steps of the process towards classification of German street signs, from data preprocessing, to the built of the deep learning model and its test on new images. The commented code used can be found in the correspondent notebook.

The report consists of different sections. In the second Section I introduce the dataset, the preprocessing techniques that I applied and possible data augmentation protocols. In the third Section I describe the final architecture chosen and collect information about training, especially a comparison between the training, validation and testing accuracies. In the fourth Section I explore the prediction of my trained network on new images downloaded via internet. Finally the last section is devoted to the optional task of visualizing the features maps of the trained network. This section remains, at the moment, incomplete.

## 2 Dataset

The German Traffic Sign dataset is a set of 51839 images of traffic signs images of $32 \times 32$ pixels and three color channels, divided into 34799 training examples, 12630 testing examples and 4410 validation examples (see images samples in Fig. 1). The signs belong to 43 different classes, which are not balanced

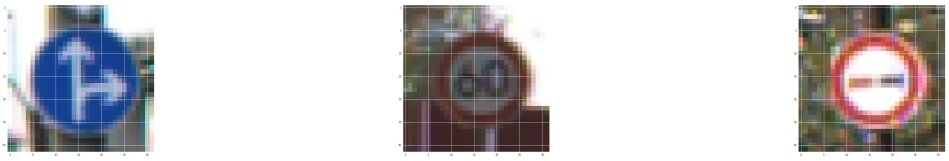

**Figure 1.** Examples of images in the German Traffic Sign Dataset.

in the original training set, as one can see in Fig.2. Many classes are heavily under-represented and this could constitute a problem when training the classifier for recognition.

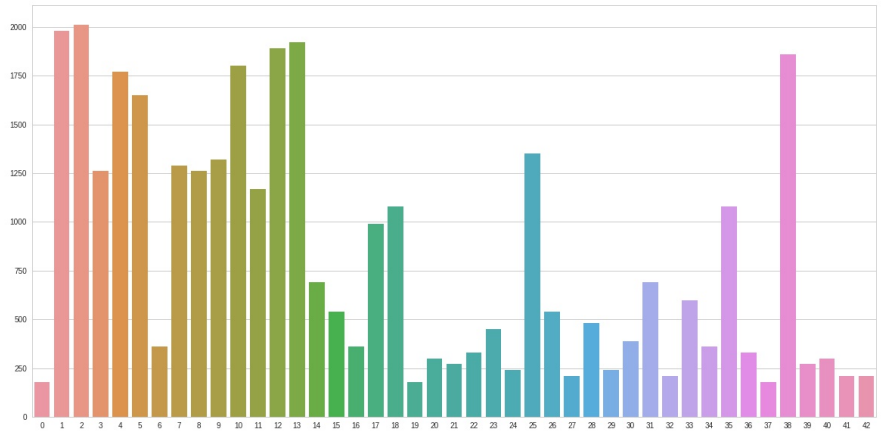


**Figure 2.** Sample count for each category class in the training set.

## 2.1 Data preprocessing

I consider two preprocessing operations for the input images applied in sequence:

- Greyscale transformation, which consists in building a greyscale channel from the three color channels of each image. This transformation on one hand has the advantage to reduce the size of the input which decreases the number of model parameters and therefore training time. On the other hand it could lead to loss of important information, since, for instance, colors are important in identifying traffic signs.
- Global contrast normalization: it consists in normalizing each image by subtracting the pixel intensity mean and dividing by the pixel intensity standard deviation.

Both these transformations are very well used in image processing and as preprocessing stages in image recognition systems. One can see examples of such transformations in Fig. 3.

## 2.2 Data Augmentation

Before proceeding with the deep learning approach we need to solve the class imbalance problem. One possible way to do this is to add new images to the training set in order that the total number of samples per class is approximately the same. The data augmentation protocol here built consists of generating new images by applying affine transformations, such as translation, rotation around the center and resizing. Examples of newly generated images are shown in Fig. 4. Data augmentation is also a technique used in deep learning to increase the size of the dataset. It is in fact an important requirement for deep learning models to have large enough dataset in order to exploit at best the high model complexities used.

For a given class of signs I generate 10000 new images by applying either a translation or a rotation or a resize operation on already present images. Extreme operations, like large shifts, rotations of large angles etc... have been proven to be detrimental regarding overfitting, therefore I limited the possible operations to small transformations:

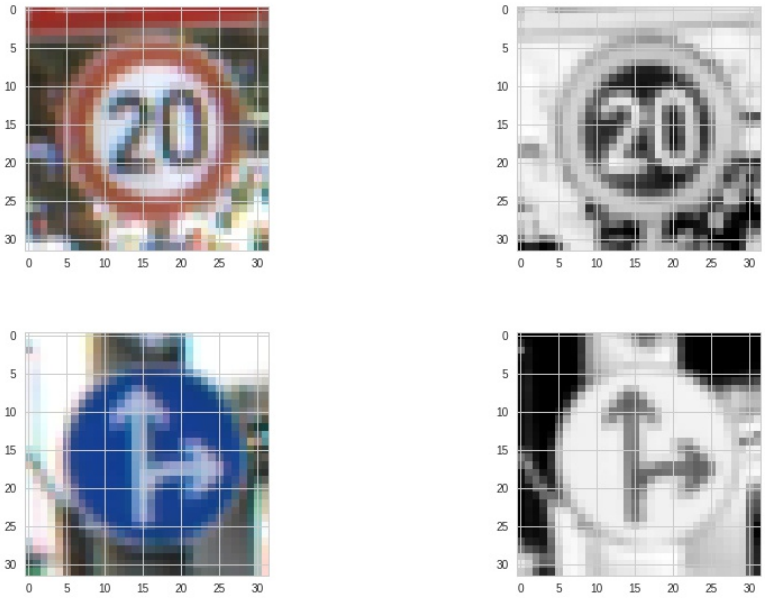

**Figure 3.** On the left original images, on the right correspondent images after greyscale transformation and global contrast normalization preprocessing.

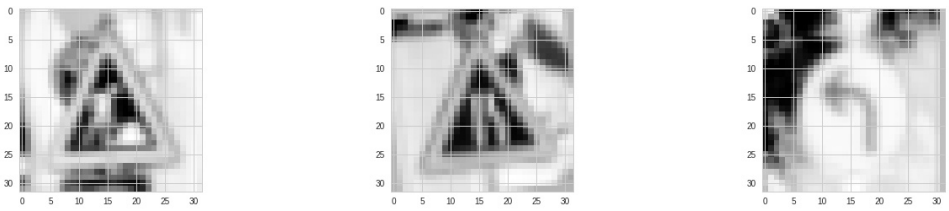

**Figure 4.** Image samples from the augmented training set.

- Translation is performed either along $x$-axis or $y$-axis. The shift is taken to be an integer in the interval $[-3, 3]$ pixels.
- Rotation around the center is performed with an angle in the range $[-10, 10]$ degrees.
- Scaling factor for both $x$ and $y$ directions range in $[0.95, 1.05]$.

Such values allow to limit overfitting and let our model train properly. After the data augmentation step the training set consists of 430000 images balanced among the 43 classes.

### 2.3 Final preprocessing considerations

I use one-hot encoding to describe the label of the images. This means that each label is defined by a 43-dimensional vector where only one component is 1 while all the others are zero. One hot encoding is encouraged for classification when the number of classes is not too high, and moreover it allows to define a model which does not only provide the predicted class for new examples, but also, through the use of the softmax activation in the last layer, the probabilities of belonging to each of the possible classes, thus allowing to estimate how good the prediction is.

## 3 Deep Learning Model

Due to performance constraints I needed to limit the depth of the deep neural network used for this project. Despite the possibility of using GPU computations, on local machine or AWS, deeper architectures would require larger training times. At the end of the project the models have been trained on my local machine using only CPU computation. Training time was about 1-2 hours long and the required minimum validation set accuracy of 93% is reached already in the first stages of the training.

### 3.1 The architecture

The final architecture chosen is shown in table 1. The process to find the right architecture passed

| Layer | Input | Output | Other Hyperparameters |
|---|---|---|---|
| Convolution + relu + LRN | 32x32x1 | 32x32x16 | kernel=5x5, stride=1, pad=SAME |
| MaxPool | 32x32x16 | 16x16x16 | size=2, pad=SAME |
| Convolution + relu + LRN | 16x16x16 | 16x16x32 | kernel=5x5, stride=1, pad=SAME |
| MaxPool | 16x16x32 | 8x8x32 | size=2, pad=SAME |
| Convolution + relu + LRN | 8x8x32 | 8x8x64 | kernel=5x5, stride=1, pad=SAME |
| MaxPool | 8x8x64 | 4x4x64 | size=2, pad=SAME |
| Fully Connected + relu | 4x4x64 | 512 | |
| dropout | | | p_dropout=0.5 (training) |
| Fully Connected + softmax | 512 | 43 | |

**Table 1.** The final architecture chosen to perform classification. LRN stands for *local response normalization*.

through the following points:

- The starting point was the LeNet architecture, which I have been using since some time, for instance, as a final project for the Machine Learning Engineer Nanodegree, where I was interested in recognising digits in the MNIST and SVHN datasets.

- The most of the work has been used trying to choose the number of Conv+relu+MaxPool blocks and the number of fully connected layers taking care of overfitting and training times. For instance adding another fully connected layer did not improve considerably the accuracy, though increasing the training times. Therefore this choice has been discarded.
- Hyperparameter have been chosen accordingly to most commonly used values in the literature. Kernel size needs to be adapted to the image size and the dimensions of objects that want to be recognised in the image. Counter-intuitively the stride of the convolution has been fixed to 1, while I would have bet that a size of 2, at least in the first layer, would have been good enough and would have even reduce the training time.
- The local response normalization operation after each convolution provides an extra help in accuracy and does not increase considerably the training time.
- Dropout has been used only in the fully connected structure and helps to reduce overfitting. It could be also employed in the convolutional structure but I found out that this is not necessary as overfitting is already taken care of, due also to the data augmentation strategy described previously.

### 3.2 Training and results

With the above chosen architecture training is quite rapid. Therefore our final model has been trained only 10 epochs, reaching already minimum required accuracy within the first epoch and minibatch size is set to 128. I chose the Adam optimizer with initial learning rate = 0.001 (suggested starting value in the literature, see for instance I. GoodFellow et. al *Deep Learning Book*). Other optimizers like Adagrad have been used with similar accuracies but larger training times. In Fig. 5 I show the training curves for the minibatch accuracy and validation set accuracies, limited to about 3 training epochs. We see that relevant values for the accuracy in the validation set are already reached within the first training epoch. Due to the success of such training I evaluate the final accuracies on the validation set (97.4%) and test set (95.3%). Both are above the required limit of 93%.

## 4 Testing on new images

In this section I test the trained network on previously unknown images taken from the web. For each of these images I predict the class type and also extract the five best softmax probabilities. Before inputing the image into the network I needed to apply similar kind of preprocessing used for the training data. This is the pipeline:

- Since the dimensions of the images are not the same as the input of the network, I first crop each image around the sign and then resize it to a $32 \times 32$ pixel image.
- On the resized image I apply greyscale transformation and global contrast normalization.

The trained model is able to classify correctly all the 5 images chosen. I also took care of analysing for each of them the softmax probabilities. The results can be found in the captions of the following 5 figures.

## 5 (Optional)Visualize the Neural Network's State with Test Images

This section is empty. Some attempt to this task can be found in the correspondent notebook.

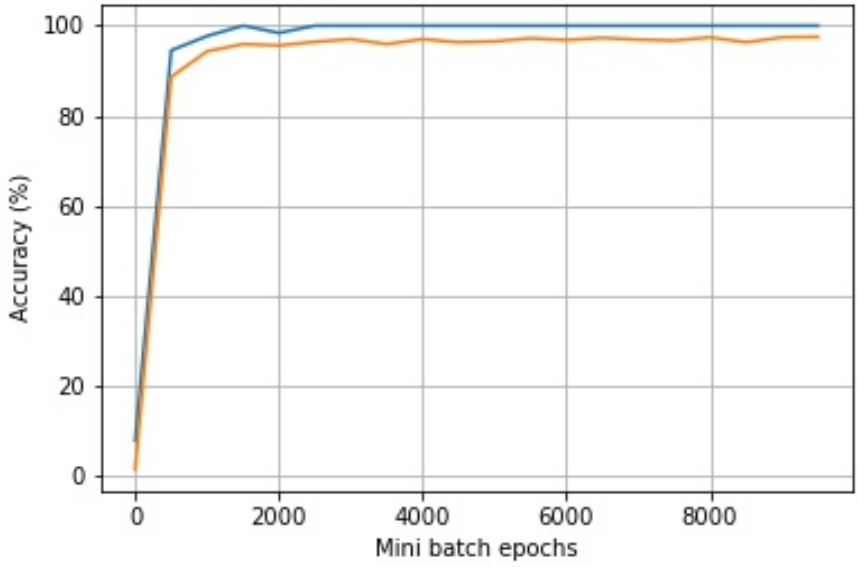


**Figure 5.** Training curves for minibatch accuracy (blue) and validation set accuracy (orange) as a function of the minibatch epoches. A minibatch epoch is defined as a single minibatch gradient descent step, and given the dimension of the training set and the batchsize there are 3360 minibatch epochs for one training epoch.

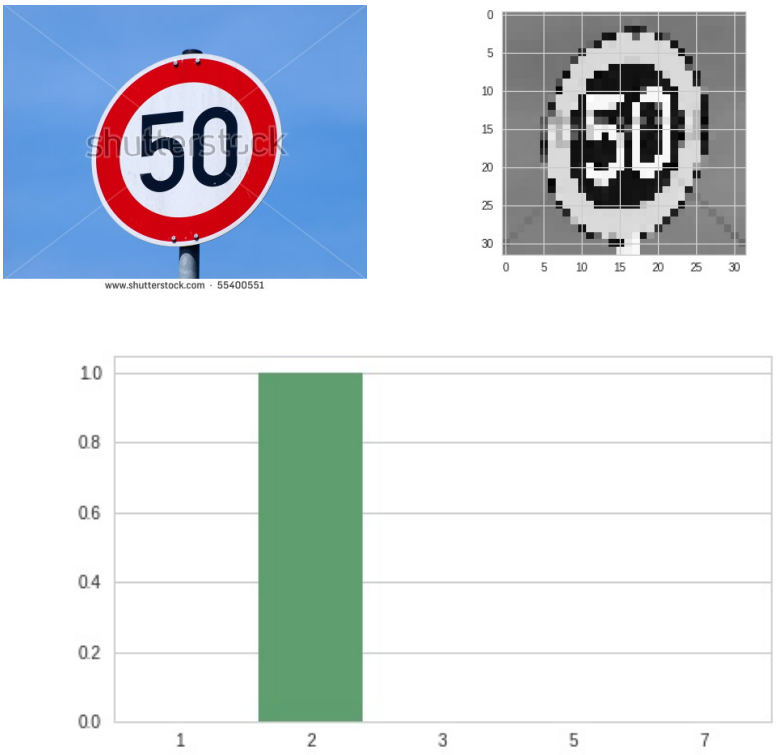


**Figure 6.** Upper left original image. Upper right the preprocessed image. Below the 5 highest softmax probabilities. The model predicts with max accuracy the correct sign (class number 2, Speed limit (50km/h) sign).

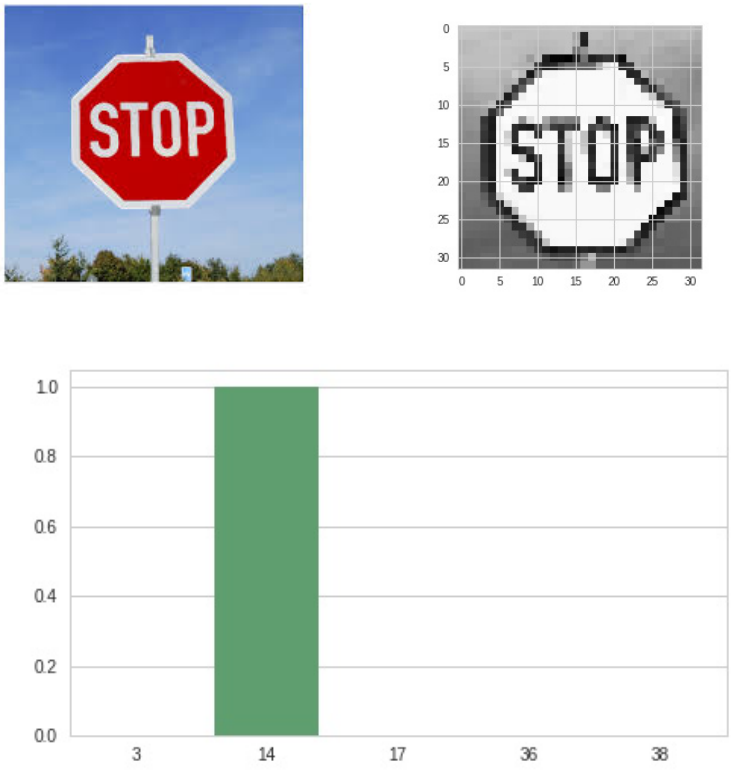


**Figure 7.** Upper left original image. Upper right the preprocessed image. Below the 5 highest softmax probabilities. The model predicts with max accuracy the correct sign (class number 14, Stop sign).

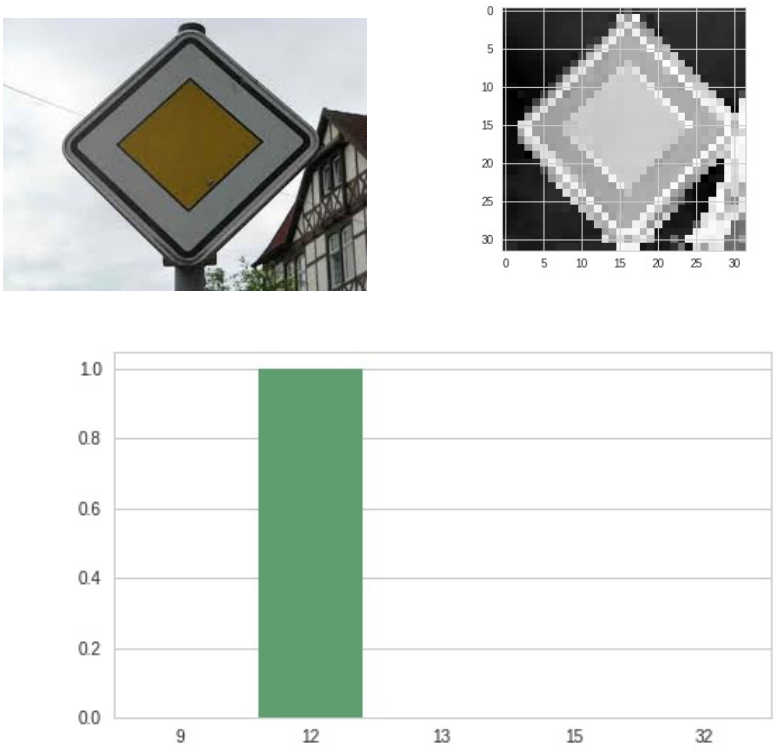


**Figure 8.** Upper left original image. Upper right the preprocessed image. Below the 5 highest softmax probabilities. The model predicts with max accuracy the correct sign (class number 12, Priority Road sign).

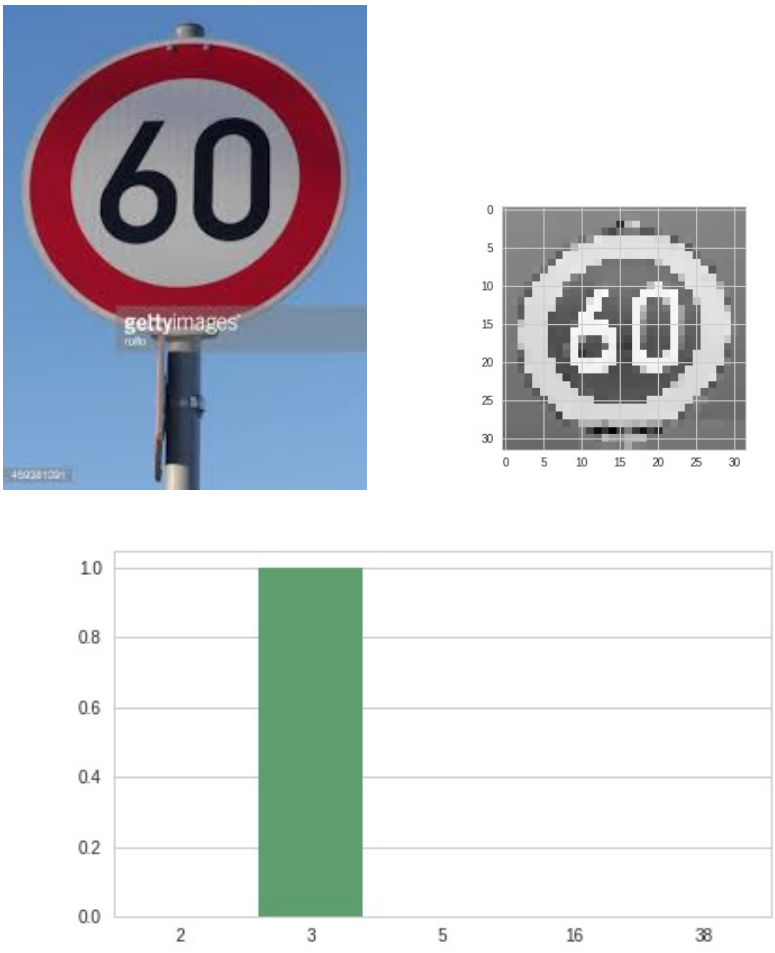


**Figure 9.** Upper left original image. Upper right the preprocessed image. Below the 5 highest softmax probabilities. The model predicts with max accuracy the correct sign (class number 3, Speed limit (60km/h) sign).

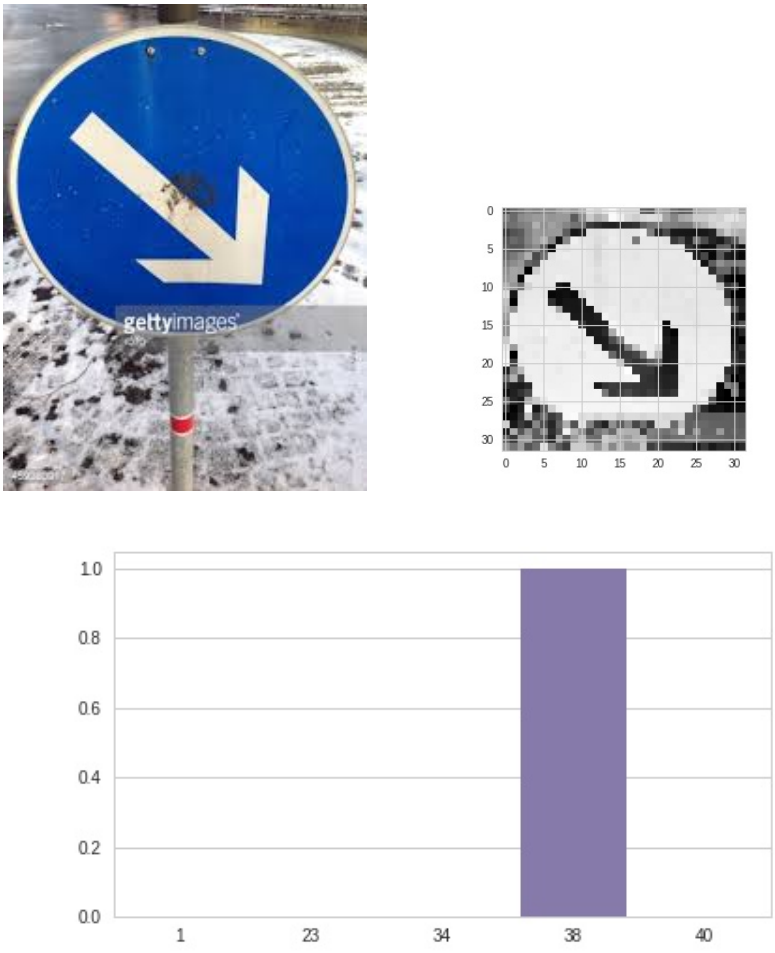


**Figure 10.** Upper left original image. Upper right the preprocessed image. Below the 5 highest softmax probabilities. The model predicts with max accuracy the correct sign (class number 38, Keep right sign).