C. D. Vijay
Udacity Car ND

Report P2: Traffic Sign Classification

Table of Contents

1. Data Set Summary & Exploration

Basic summary of the data set

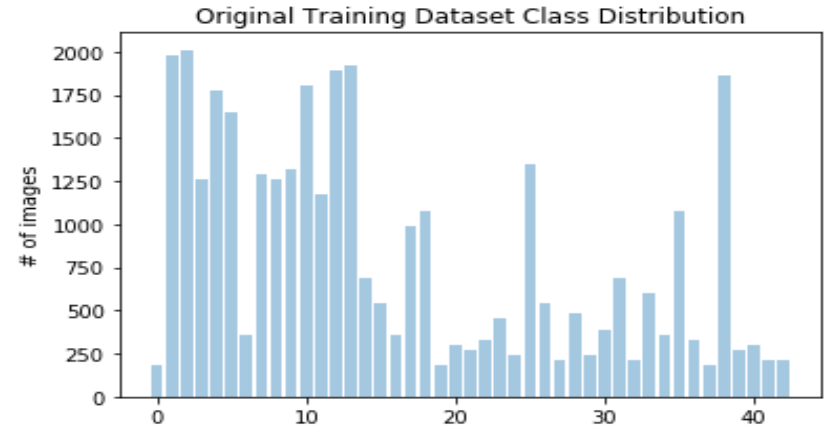
Numpy library was used to calculate summary statistics of the provided traffic signs dataset:

- *The size of training set is 34,799*
- *The size of the validation set is 4,410*
- *The size of test set is 12,630*
- *The shape of a traffic sign image is 32 x 32 x 3*
- *The number of unique classes/labels in the data set is 43*

An exploratory visualization of the dataset

The right column is a visual sampling of the dataset: Images are mostly standard traffic signs but are seen to display varied lighting conditions.

A bar chart showing data distribution in this dataset is shown below.




```
Image Class Distribution
        Mean                    =  809.28
        Standard Deviation      =  619.42
```

Figure 1

C. D. Vijay, Udacity Car ND, Term 1, P2

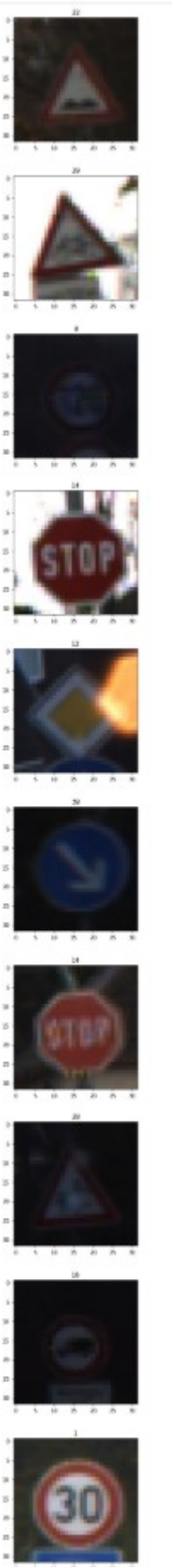


Figure 2

Clearly, the distribution is imbalanced. Proper training warrants a balanced dataset to prevent bias.

2. Model Architecture – Design and Test

Baseline results (LeNet)

`VA:Validation accuracy | TrA:Training Accuracy, TeA:Test accuracy`

LeNet architecture "out-of-the-box" yielded a VA of 89% (over 20 epochs with a batch size of 256). From this point onwards, tuning a few hyperparameters such as learning rate, # of epochs, and batch size only altered VA by an insignificant few tenths of a percentage point.

The only notable positive result came about when the batch size was changed to a serendipitous 32 to yield a VA of 95.5% [which was confirmed later by a chanced upon Reference [4]].

So, to start an iterative investigation, another convolutional and maximum pooling layers were added with a small improvement. Variations tried on it and their results are below.

# of Epochs	Batch Size	Validation Accuracy	Modifications	Time (Min.)
10	256	0.889	Original 6, "out of the box"	3.27
10	256	0.885	Original 6 + new conv + max pooling layers	4.66
10	128	0.929	Same as above	5.03
20	128	0.910	Same as above	10.77
20	256	0.914	Same as above	9.74
20	64	0.942	Same as above	8.97
50	32	**0.955**	Same as above	24.3
50	128	0.951	Same as above	22.22
50	128	0.926	Same as above + Normalizing training data	24.3
50	128	0.090	Same as above + Normalizing training, validation data	process killed
			Table continued...	

# of Epochs	Batch Size	Validation Accuracy	Modifications	Time (Min.)
50	128	0.050	Same as above + Normalizing only training + 2 dropout layers, LR=0.005 (N: X-128/128)	process killed
50	128	0.050	Same as above + Normalizing only training + 1 dropout layer, LR=0.005	process killed
50	128	0.933	Same as above + Normalizing only training + LR=0.001	26.3
Another 30 similar iterations are not reported here.				

So, short of accepting the 0.955 result with the dataset as-is, the only other thing remaining was to rectify the class (mal)distribution histogram (Figure 1), which acted as an insistent reminder for the need to balance the classes, even if approximately, to prevent learning bias during training. Also note above the drastic drop in accuracy with the addition of dropout layers (perhaps, there are other settings not fully considered here).

Data Augmentation

To do so, the usual methods of flipping images, rotation, translation and image brightness augmentation, were considered. Eventually, all of them, as detailed in the code, were included; most of this augmentation patch code is borrowed from other sources [2].

Summary parameters of the total data set

```
Original dataset length:                34799

Images augmented by flipping:           72570
Images augmented by rotation/
translation/brightness adj.:            8341
Images in dataset after augmentation:  115710 [1]
```

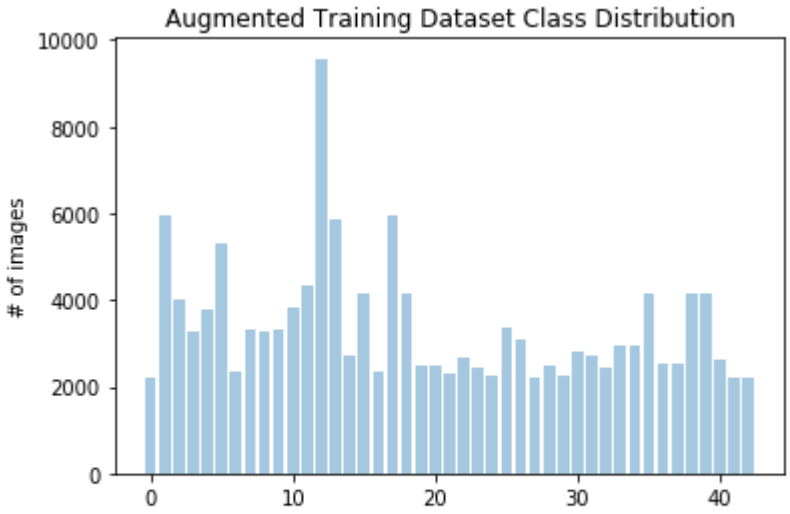
1 As an experiement, images were augmented twice for a total of approx. 200,000 images; performance, however, was not commensurate with the extra computation cost, and hence dropped.

In the right column on this page are examples of an original image followed by augmented images through these methods (Figure 3).

Figure 3

A bar chart showing the new data distribution in the total dataset with augmentation is shown next (Figure 4).

Figure 4



A few of the observed sharp variation in classes (esp. class 12) is from the addition of images from the flipping operation.¶

```
Image Class Distribution
        Mean                    =   3400.42
        Standard Deviation      =   1413.80
```

Preprocessing

Strictly, preprocessing is unnecessary as the image values are “uint8” datatype (0-255). So, only mean centering about zero was the technique chosen, and hence no perceptible change is seen between the two. See Figure 5 next.

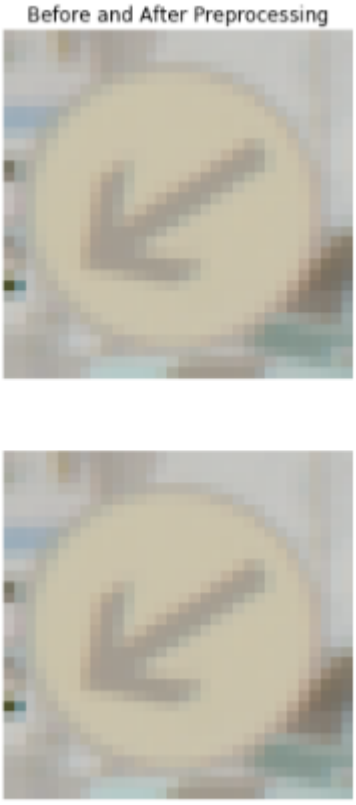


Figure 5

Final Model Architecture

The final model (called "SimpLeNet") consists of the following layers:

Layer	**Description of** SimpLeNet
Input	32x32x3 RGB image
Convolution 1 5x5	1x1 stride, VALID padding, outputs 28x28x6
RELU	Followed by L2 losses
Convolution 2 5x5	1x1 stride, VALID padding, outputs 24x24x32
RELU	Followed by L2 losses
Convolution 3 5x5	1x1 stride, VALID padding, outputs 20x20x64
RELU	Followed by L2 losses
Convolution 4 5x5	1x1 stride, VALID padding, outputs 16x16x128
RELU	Followed by L2 losses
Convolution 5 5x5	1x1 stride, VALID padding, outputs 12x12x128
RELU	Followed by L2 losses
Max pooling	Outputs 6x6x128
Fully connected 0	Flatten, outputs 4608
Dropout	Keep Prob = 0.5
Fully connected 1	Outputs 2000
RELU	
Dropout	Keep Prob = 0.5
Fully connected 2	Outputs 120
RELU	
Dropout	Keep Prob = 0.5
Fully connected 3	Outputs 43 (# of labels)

This model is an uncomplicated, modified LeNet. Most of the Dropout layers were dropped as they only decreased performance of the network no matter how few of them were placed at various stages in the model architecture. Overfitting is prevented, however, by assiduously using L2 losses, which also *drastically helped* with increasing performance.

Training Approach

To train the model, the standard albeit effective Adam optimizer was used, with a batch size fixed at 64 (Reference [2]), and learning rate changed only once (in vain). It was reset to 0.001 later.

The path to the final arrangement of the architecture and its results was a painfully slow iterative procedure (over 100 iterations on a CPU!). The image augmentation, which was supposed to remove bias and improve convergence, turned out not too helpful.

The two musketeers of this architecture turned out to be:

1. L2 Regularization (beta, β fixed at 1E-4 after trying several candidates)

2. A factor of $\sqrt{2/n}$ (n = # of inputs) for the Initial Weights vector, (code: line 7 in [1] of function SimpleNet(), see [2] for technical paper reference)

[3]. Still looking for the third!

The sum of these tweaks produced a max. epochal **validation accuracy of 97.4%.**

So, the **final model results** were (Epochs=15, Batch size: 64)

- *training set accuracy of 83.2%*
- *validation set accuracy of 97.4% (max. at epoch 11)*
- *test set accuracy of 92.7 %*

3. Testing Model on New Images

Five German traffic signs were selected from the internet (shown below).

Illustration 1

Illustration 2

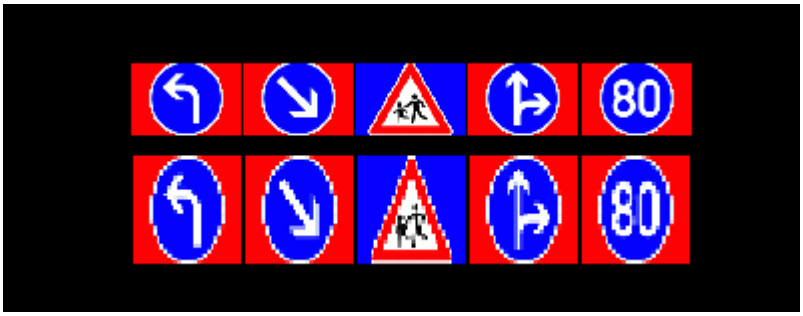
Illustration 3

Illustration 4

Illustration 5

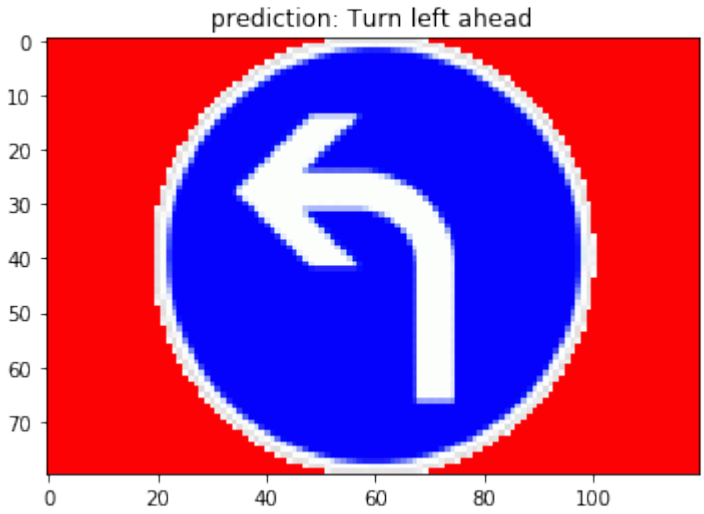
All of these images are shown at their original visually unconfusing resolution (top row below). But, the prediction will happen at half their sizes (32x32 px), which by itself appears difficult to classify because of their potential similarity to several other signs.

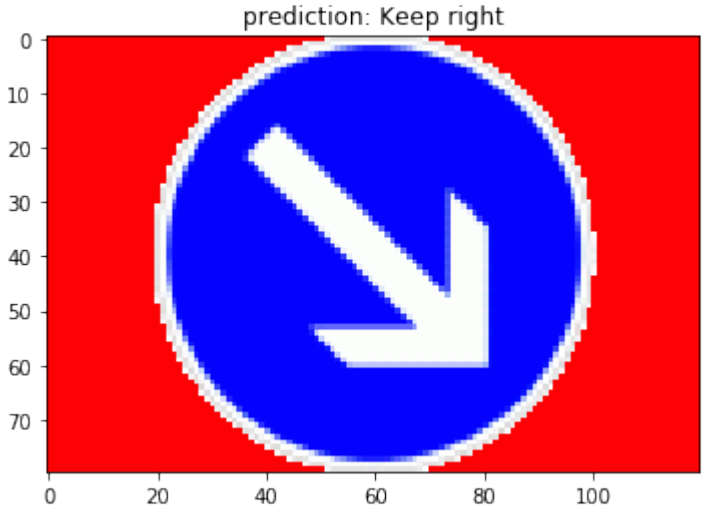
Here they are: (top row is still original just shown small, bottom row is resized to 32x32 px):

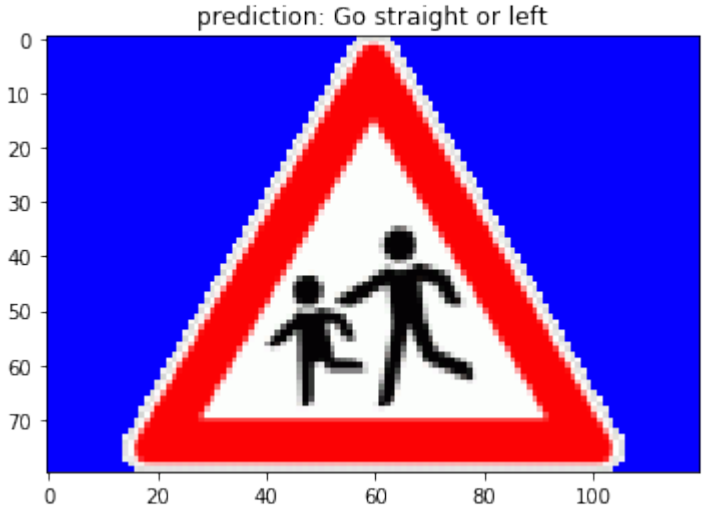


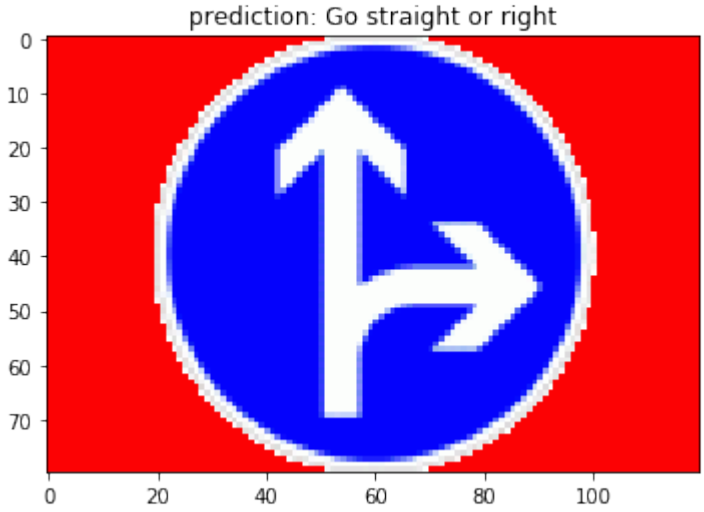
The third one (Children Crossing) especially is hard to classify even for a human. Below are the predictions made by this model.

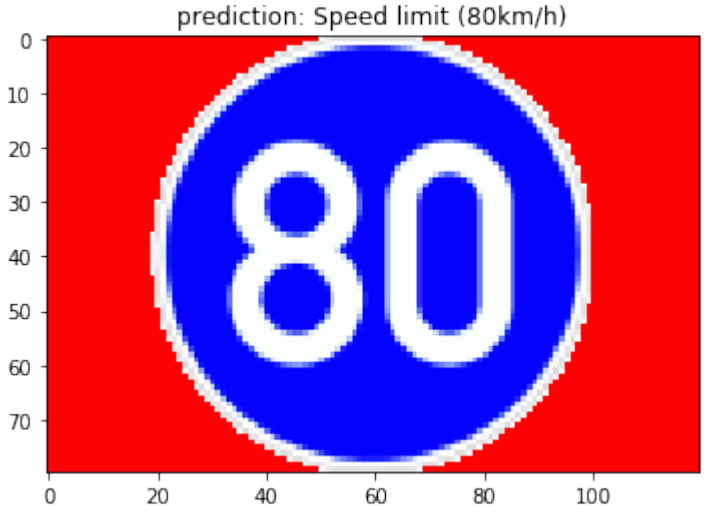
LABEL PREDICTION ON TEST IMAGES











SOFTMAX PROBABILITIES

So, as can be seen, the predictions are surprisingly accurate.

And, the softmax probabilities are (image name on left in the figure order above) `TopKV2`:

```
1-TL-p.png: [  1.00000000e+00,   9.59668789e-10,   1.63347239e-10,
               9.79630179e-12,   4.74620777e-12],

2-KR-p.png: [  9.99688506e-01,   3.06557340e-04,   4.41760085e-06,
               4.07867731e-07,   1.12017503e-07],
3-CC-p.png: [  6.71114847e-02,   6.63075075e-02,   6.10031486e-02,
               6.02040887e-02,   4.88343872e-02],
4-GSR-p.png:[  1.00000000e+00,   2.77072476e-09,   1.39458636e-11,
               1.32873762e-15,   3.77064896e-16],
5-80-p.png: [  5.13613582e-01,   1.66549638e-01,   1.21239312e-01,
               4.69864756e-02,   3.96049656e-02]], dtype=float32),

indices=array([
1-TL-p.png:       [34, 25,  1, 35, 38],
2-KR-p.png:       [38, 34, 40, 39,  5],
3-CC-p.png:       [37, 19, 20, 36, 32],
4-GSR-p.png:      [36, 38, 34, 22, 35],
5-80-p.png:       [ 5, 12, 15, 40, 31]]))
```

The top array shows probabilities while the bottom array shows the corresponding predicted labels. See the Udacity-provided signnames.csv file for legend (also part of the GitHub repo).

Here are the results of the prediction:

Image	Prediction	Probability (%)
Turn Left Ahead	Turn Left Ahead	100
Keep Right	Keep Right	99.9
Children Crossing	Go Straight or Left	6.71
Go Straight or Right	Go Straight or Right	100
80 km/h	80 km/h	51.4

The model was able to correctly guess 4 of the 5 traffic signs, which gives an accuracy of 80%. This compares favorably to the accuracy on the test set of 92.7% and confirms the earlier note that the third image was hard to classify from the start (even for a human).

The model is confident on three out of five predictions, while diffident on the third (which a human too would be), and it is relatively sure of the fifth (the speed limit).

Note that an inadvertent execution of the prediction block of the Ipython Notebook produced different predictions from what have been noted in the earlier pages. The numbers shown here (probabilities and class labels) were copied from the earlier run of the notebook.

4. Summary

So, in summary, this model, SimpLeNet, has performed (surprisingly for me) admirably. It can only become better by parameter optimization, more intelligent data augmenting, and definitely with the help of more powerful hardware and DNN models publicly available.

References

1. Jupyter Notebook (project source: p2-final4-2-Oct1.ipynb) Project code
2. https://arxiv.org/abs/1609.04836 ("On Large-Batch Training for Deep Learning")

Acknowledgments

Some parts of the data augmentation code is a mishmash of three sources: OpenCV.org, Navoshta.com and Vivek Yadav (the latter two presumably are Udacity students).