INFO8010: Traffic sign detection

Karim Khadro[1]

[1] *karim.khadro@student.uliege.be (s206176)*

I. INTRODUCTION

Vision-based technology has an essential role in smart transport systems such as driver assistance[1] and self-driving cars. A traffic sign recognition system that relies on computer vision is one of the main blocks of smart transport. The system usually consists of two stages:

1. traffic sign detection:locate the traffic sign(s) in an image captured from a camera.
2. traffic sign classification: classifies each traffic sign into a specific class.

Old detection methods[2] use color, texture, edge, and other low-level features which were hand-crafted into the system. The performance of systems like that is not good in real driving conditions due to the different shapes and sizes of traffic sing and the weather changes as well. The fast development of deep learning allowed the use of deep learning-based methods to detect traffic signs. The performance is much better than traditional methods in all aspects and conditions. Still, there are some challenges such as :

- the speed of the detection methods especially in high-speed driving and the fact that cars may not be equipped with high-end hardware.
- the size of the traffic signs is quite small in images, and small traffic sign detection is much more challenging than large traffic sign detection.

In this project, high-quality images are used, since it is inevitable to have a model with good performance. The implementation consist mainly of two parts, the first one is the detection using YOLO (you only look once) version 4[3] and, the second one is the classification using a convolutional neural network.

II. DATA

As data are the most important block to have a deep learning model that has good performance. Data from multiple sources Kaggle and BTS (Belgian Traffic Sign) were gathered in order to maximize the number of images and the number of classes as well.

A. GTSRB

The German Traffic Sign Recognition Benchmark (GTSRB)[4] dataset collected in Germany. It contains 42 classes and more than 50 000 images in total. This dataset is quite interesting to train the classifier because it only contains small dimensions traffic sign images.

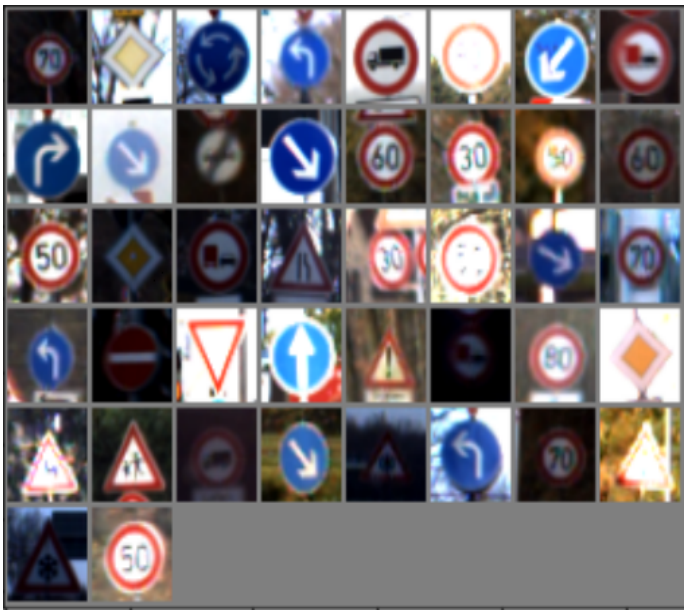


FIG. 1. GTSRB sample of the 42 classes

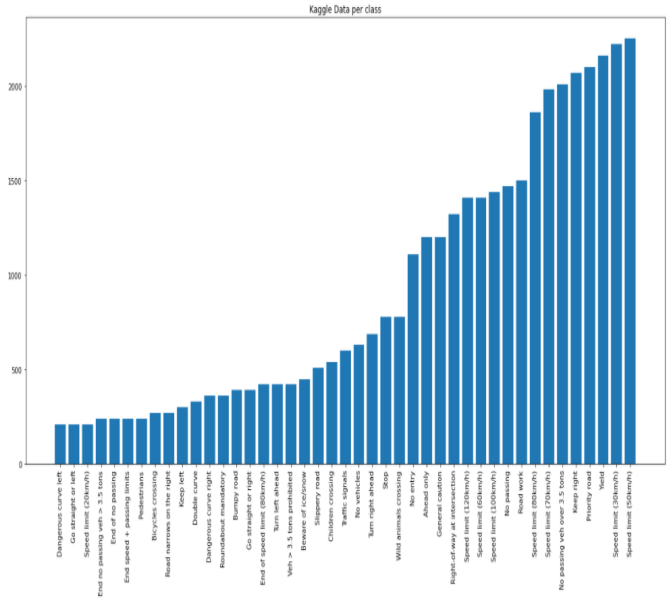


FIG. 2. GTSRB image per classe

Training images are not distributed equally on all

classes, as shown in figure 2, and 18 classes with less than 500 images. Which complicate the task of having enough images to train the classifier.

B. Belgium traffic sign

Belgium traffic sign (BTS) dataset consist of 62 classes of traffic signs, and it contains around 50 GB of full road images.which make it great for training YOLO to detect a traffic sign in an image.

Considering that this dataset contains more and different classes than GTSRB, it becomes interesting to use the data provided by BTS to train the classifier. Still, there are not enough images, as shown in figure3 to train a good classifier. Here all the classes have less than 700 image with 6 image to the smallest class and no testing images for some of the classes.

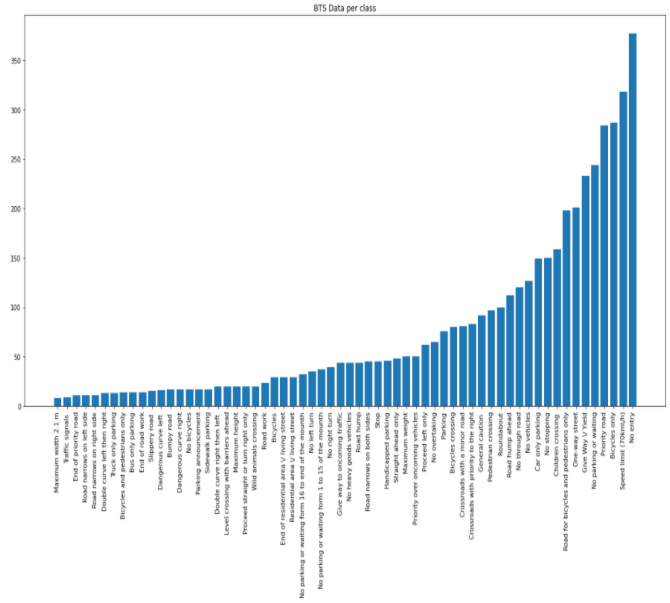


FIG. 3. BTS image per classe

C. Data pre-processing

The BTS images are provided in two formats JP2 and PPM. Which are not readable by the PILL. A pre-processing were needed to transform all images to PNG format to prevent all problems related to this aspect and having all the images in the same format.

Another pre-processing to make full road images compatible with YOLO format consist of creating a text file with the same name as the image. This file should contain a class number, starting coordinate of the traffic sign, width of a traffic sign, and height. All the coordinate value should be normalized to run from 0 to 1 in both x and y directions. The conversion done as follow :

x = x / XMAX and y = y / YMAX
where XMAX, YMAX are the maximum coordinates for the image.

```
0 0.8995515970515969 0.2957807443365696 0.1445331695331695 0.19227346278317156
```

FIG. 4. YOLO text file

D. data augmentation

Seeing that the majority of classes do not have enough images and to increase the model's performance. Data augmentation is used to help to get more images from the original one. For this project not all data augmentation techniques can be used, such as rotation, flip and mirroring because some traffic sign when flipped or rotated can have different meaning. The new augmanuted images generarted by the package Augmentor[5], which provide a variety of methods and can run on big number of images. The data augmentation methods used in this project :

random$_{c}olor$	greyscale	random$_{d}istortion$
random$_{c}ontras$	rotate (max 5°)	zoom$_{r}andom$
skew$_{c}orner$	shear	crop$_{r}andom$

TABLE I. used methods of data augmentation

The combination of all those methods allowed having 4000 images per class. From each class 600 images selected randomly to be added to the testing dataset. After all, the new data consists of 80 classes (BTS and GTSRB combined), 3400 images for training per class, and at least 600 images for testing per class. For this project, 3000 images chosen randomly used for training and 600 images chosen randomly for testing, with ratio 5:1 (20%). Image examples of all the 80 classes available on google drive[6].

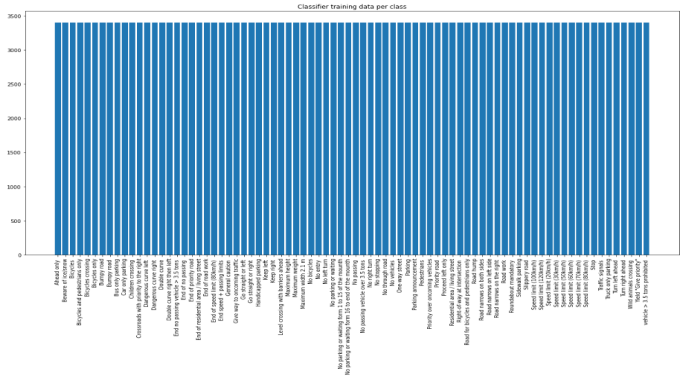


FIG. 5. classifier: 80 classes data for training after augmentation

III. IMPLEMENTATION

As described earlier, the implementation consists of two main blocks detection and classification. Instruction on how to run the code using the web interface are available on GitHub[7]. Also, the project is hosted at on Streamlit sharing platform[8].

A. Detection

YOLO is used to detect traffic sign(s) in a road image. Why using YOLO ?
First the speed, YOLO is one of the fastest real-time object detectors. Second, the high accuracy after a small time of training. YOLO gets as input a road image with or without traffic sign(s) in it. And give as output an array of traffic sign coordinate which then used as input to the classifier.

FIG. 6. YOLO input image

FIG. 7. YOLO output image

To train YOLO on detecting traffic signs, around 8800 images are used and 6 hours (3 * 2 hours) of GPU time to achieve 4000 iterations and a loss just above 0.5. After this training time, YOLO becomes capable of detecting the majority of traffic signs in an image. With some difficulties in detecting far and small traffic signs where there are bigger and nearer signs.

In order to use YOLO with the classifier, the Opencv [9] library play the intermediate between the two parts. The code of this part come from multiple sources [10] on the internet and then it is adapted to work with for this project. This is the only way that allowed the combination of the two parts.

B. Classification

The next step after the detection is the classification of the images into classes. The model consists of :

- three convolutional Neural Networks and one linear layer.
- Relu as activation function.
- batch normalize and a max pool layer after every convolutional layer.
- softmax as last layer activation function.

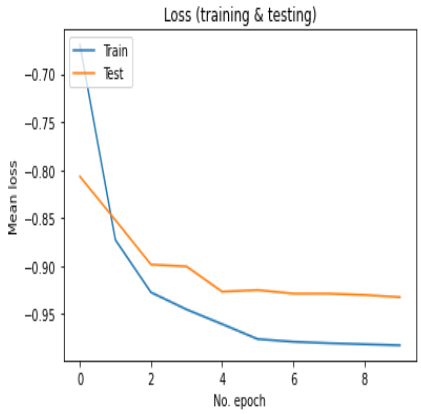
```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
CNN                                      --                        --
├─Conv2d: 1-1                            [128, 27, 30, 30]         2,052
├─BatchNorm2d: 1-2                       [128, 27, 30, 30]         54
├─Conv2d: 1-3                            [128, 54, 13, 13]         36,504
├─BatchNorm2d: 1-4                       [128, 54, 13, 13]         108
├─Conv2d: 1-5                            [128, 108, 4, 4]          145,908
├─BatchNorm2d: 1-6                       [128, 108, 4, 4]          216
├─Linear: 1-7                            [128, 80]                 34,640
==========================================================================================
Total params: 219,482
Trainable params: 219,482
Non-trainable params: 0
Total mult-adds (G): 1.32
==========================================================================================
Input size (MB): 1.57
Forward/backward pass size (MB): 72.08
Params size (MB): 0.88
Estimated Total Size (MB): 74.53
==========================================================================================
```

FIG. 8. Classifier: model details

To train the model, the Adadelta optimizer along side Negative Log-Likelihood (NLLLoss) as loss function are used. The choice of the optimizer Adadelta is based on multiple tests reilaized using the same model and same images. In those test, Adadelta out preformed the other tested optimizers (Adam and SGD). For the loss function, NLLLoss is one of loss functions for multi-class classification problem, and it gave good results. The classifier achieved 93.61% accuracy with a -0.935 loss for testing dataset.To achieve this accuracy with the classifier many designs with different numbers of layers, parameters, loss, activation, and optimizer were tested. But this architecture gives the highest accuracy with the relatively low loss after only 20 epochs. And give a speed advantage as the number of parameters is not that big.

The classifier takes only one image at time from YOLO's output, figure 7, as input and give a class number between 0 and 79 as output.

```
1
TRAIN : epoch : 1  accuracy : 68.33548736572266%  loss : -0.6694448590278625
TEST : epoch : 1  accuracy : 82.32775724160976%  loss : -0.8067376017570496
2
TRAIN : epoch : 2  accuracy : 87.8307113647461%  loss : -0.8728206753730774
TEST : epoch : 2  accuracy : 85.74024183948374%  loss : -0.8521668314933777
3
TRAIN : epoch : 3  accuracy : 93.186279296875%  loss : -0.9274091124534607
TEST : epoch : 3  accuracy : 90.13219723074995%  loss : -0.8984310626983643
4
TRAIN : epoch : 4  accuracy : 94.80963134765625%  loss : -0.9452211260795593
TEST : epoch : 4  accuracy : 90.33268592193471%  loss : -0.9004533886909485
5
TRAIN : epoch : 5  accuracy : 96.27687072753906%  loss : -0.9604504108428955
TEST : epoch : 5  accuracy : 92.91397782093854%  loss : -0.9266475439071655
6
TRAIN : epoch : 6  accuracy : 97.78569793701172%  loss : -0.9760826826095581
TEST : epoch : 6  accuracy : 92.8262640185452%  loss : -0.924994945526123
7
TRAIN : epoch : 7  accuracy : 98.0219955444336%  loss : -0.9788606762886047
TEST : epoch : 7  accuracy : 93.13117390305537%  loss : -0.928706705570221
8
TRAIN : epoch : 8  accuracy : 98.14583587646484%  loss : -0.9803518056869507
TEST : epoch : 8  accuracy : 93.13326232692187%  loss : -0.9287203550338745
9
TRAIN : epoch : 9  accuracy : 98.23759460449219%  loss : -0.981462299823761
TEST : epoch : 9  accuracy : 93.20009189065013%  loss : -0.9301160573959351
10
TRAIN : epoch : 10  accuracy : 98.3296890258789%  loss : -0.9825310111045837
TEST : epoch : 10  accuracy : 93.3776079193033%  loss : -0.9323692917823792
```



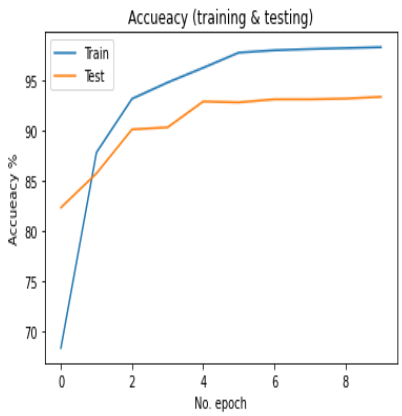
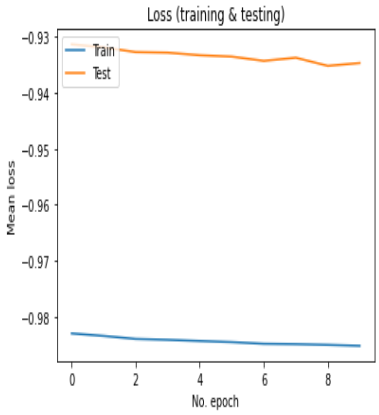


FIG. 9. Classifier: training 1 to 10 epochs

```
1
TRAIN : epoch : 1  accuracy : 98.36306762695312%  loss : -0.9830217957496643
TEST : epoch : 1  accuracy : 93.31077835557505%  loss : -0.9313204884529114
2
TRAIN : epoch : 2  accuracy : 98.39935302734375%  loss : -0.9834734201431274
TEST : epoch : 2  accuracy : 93.3253973226406%  loss : -0.9318699240684509
3
TRAIN : epoch : 3  accuracy : 98.43895721435547%  loss : -0.9840079545974731
TEST : epoch : 3  accuracy : 93.4277300920995%  loss : -0.9327054023742676
4
TRAIN : epoch : 4  accuracy : 98.45687103271484%  loss : -0.9841693639755249
TEST : epoch : 4  accuracy : 93.44861433076457%  loss : -0.932807207107544
5
TRAIN : epoch : 5  accuracy : 98.46898651123047%  loss : -0.9843818545341492
TEST : epoch : 5  accuracy : 93.46323329783013%  loss : -0.9332700371742249
6
TRAIN : epoch : 6  accuracy : 98.48567962646484%  loss : -0.9845799803733826
TEST : epoch : 6  accuracy : 93.46323329783013%  loss : -0.9335164427757263
7
TRAIN : epoch : 7  accuracy : 98.50692749023438%  loss : -0.9848778247833252
TEST : epoch : 7  accuracy : 93.54468182862394%  loss : -0.9342760443687439
8
TRAIN : epoch : 8  accuracy : 98.51231384277344%  loss : -0.9849483966827393
TEST : epoch : 8  accuracy : 93.52797443769187%  loss : -0.9337144494056702
9
TRAIN : epoch : 9  accuracy : 98.5190200805664%  loss : -0.985069990158081
TEST : epoch : 9  accuracy : 93.61986508781823%  loss : -0.9351708292961121
10
TRAIN : epoch : 10  accuracy : 98.537841796875%  loss : -0.9852437973022461
TEST : epoch : 10  accuracy : 93.56974291502203%  loss : -0.9346612691879272
```



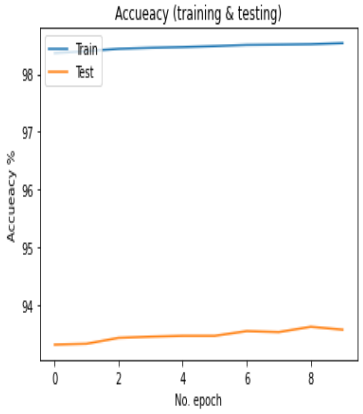


FIG. 10. Classifier: training 10 to 20 epochs

IV. RESULT

The training process of the classifier gave very good results on the accuracy side (93.6%) and the loss side (-0.935). As well did YOLO during the training with a loss value of 0.5. But still, there is no result of the two-part working together. To test the full model (detection and classification) by hand a small web interface using Streamlit[11] is created. As a first impression, everything works well. The next testing step consisted of using 2064 full road images (those images at least one traffic sign known by the classifier) to test the full model.
The final results : 220 failed (10.5%) and 1844 passed (89.5%). After some inspections, the big majority of failures become from YOLO, not detection all the traffic signs or detecting only the sign with cities name or information about the directions. Also, the speed of the model still quit high, with an average of 0.23 second for YOLO to detect traffic sign(s) in an image, and an average of 0.003 second for image to be classified.

FIG. 11. Final result

FIG. 12. Multiple detection

Even with a high classifier accuracy and advanced detection technology, failures still happen. In this project, the big majority of failures come from YOLO not detecting all signs or detecting wrong signs "direction signs". After all YOLO training is not completed yet and can be improved.
Here are some classification/detection failures.

FIG. 13. YOLO not detecting all signs

FIG. 14. YOLO detectin the city name sign and not the traffic sing

FIG. 15. The classifier fail to correctly classify signs

V. IMPROVEMENT

There are many aspects that can be improved on both detection and classification sides.

A. Detection

To improve the detection:

- Train YOLO for longer time.
- Make YOLO sees the difference between traffic sign and directions sign, by adding a class direction sings.
- Train YOLO on augmented data for detection traffic signs at night.

B. Classification

To improve the classifier:

- Add more classes.
- Use detection by category (danger,Warning and Informative) sign then use smaller classifier of each category.

[1] The 6 levels of vehicle autonomy. https://www.synopsys.com/automotive/autonomous-driving-levels.html.

[2] V. Ramesh M. Pellkofer C. Bahlmann, Y. Zhu and T. Koehler. A system for traffic sign detection, tracking, and recognition using color, shape, and motion information. 2005.

[3] AlexeyAB. Yolo v4 provided. https://github.com/AlexeyAB/darknet.

[4] Gtsrb on kaggle. https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign.

[5] Augmentor package. https://augmentor.readthedocs.io/en/master/.

[6] Google drive image example. https://drive.google.com/drive/folders/11k-WEFVmRyN_xGBKnQPXdZoxx6abUk2I?usp=sharing.

[7] Github repo. https://github.com/Karim-khadro/DL-Traffic-sign-detection.

[8] The app host on streamlit sharing. https://share.streamlit.io/karim-khadro/dl-traffic-sign-heroku/main/web.py.

[9] Opencv official website. https://github.com/Karim-khadro/DL-Traffic-sign-detection.

[10] Deep learning with opencv dnn module: A definitive guide. https://learnopencv.com/deep-learning-with-opencvs-dnn-module-a-definitive-guide/#object-detection-in-real-time-videos.

[11] Streamlit official website. https://streamlit.io/.