UNIVERSITY OF MARYLAND

ENPM 673 P6

SUBMITTED BY,

SAKET SESHADRI GUDIMETLA | CHAYAN PATODI | PRASANNA MARUDHU BALASUBRAMANIAN

INTRODUCTION

The aim of this project was to do Traffic sign recognition for a given sequence of images obtained from ETH-Zurich Vision Lab. There were two major tasks broadly speaking viz detection and classification. For the detection stage we chose the MSER algorithm and followed the suggested pipeline in Project guidelines. Once the signs were detected, we extracted the ROI, resized them, extracted features and passed them on to the classifier for identifying the image and showing the interpretation by displaying the image understood by the classifier against the traffic sign.
The pipeline will be explained in-depth in the following pages.

TRAFFIC SIGN DETECTION

For detection purpose we used the Maximally Stable Extremal Regions (MSER) algorithm which is a popular detection algorithm. For the detection we followed the instructions given in the pipeline and they are explained as follows –

1. **Denoise the image** –
   The first step was to remove noise in the image. For this purpose, we used the OpenCV function cv2.fastNlMeansDenoisingColored which implements the Non-local Means Denoising algorithm. In brief what this algorithm does is it takes a set of similar looking pixels to average out the noise since the noise is a random variable with zero mean. The algorithm first takes a pixel, then takes a small window around the image, finds similar looking windows, averages all the windows and replaces the pixel with the result.
   The images before and after denoising are shown below –

   Original image →

Image after denoising →

[P.S. WE SAVED ALL THOSE DENOISED IMAGES IN A FOLDER "denoised_input" IN ORDER TO SPEED UP PROCESSING. THE DRIVE LINK IS https://drive.google.com/open?id=1mirdmgEndhK40vgkVbThsW3dBUUMKzMi]

2. **Contrast Normalization –**

   The image after contrast normalization looks like follows –

For contrast normalization we used the following formula for each of the three B, G, R channels of the image -

iO = (iI-minI)*(((maxO-minO)/(maxI-minI))+minO)

where,

I_o	- Output pixel value
I_i	- Input pixel value
Min_i	- Minimum pixel value in the input image
Max_i	- Maximum pixel value in the input image
Min_o	- Minimum pixel value in the output image
Max_o	- Maximum pixel value in the output image

3. **Intensity normalization** –

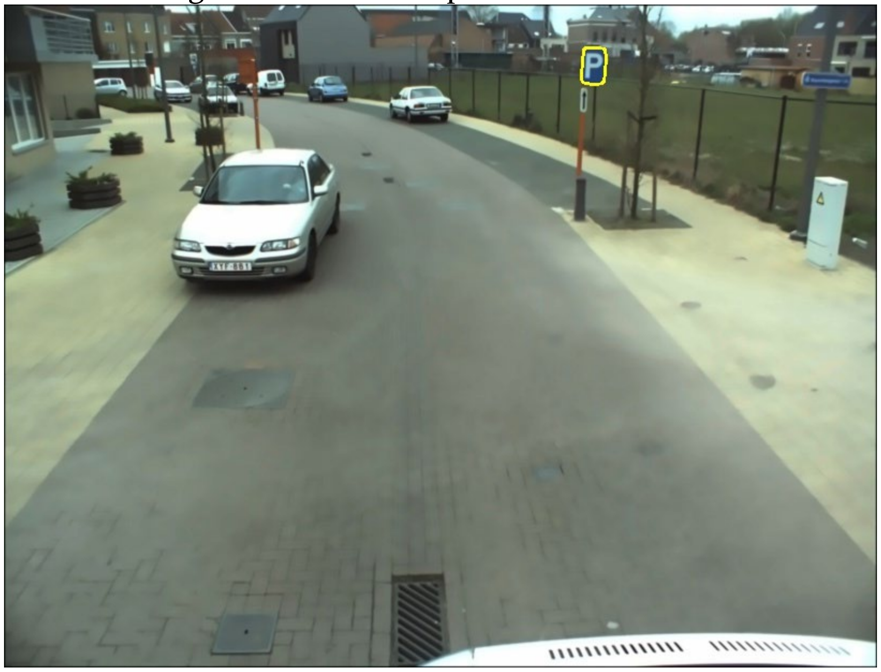
The next step was to normalize the intensities for which we used the following formulas –

Rn = max (min (R-G, R-B)/(R+G+B) - for red signs

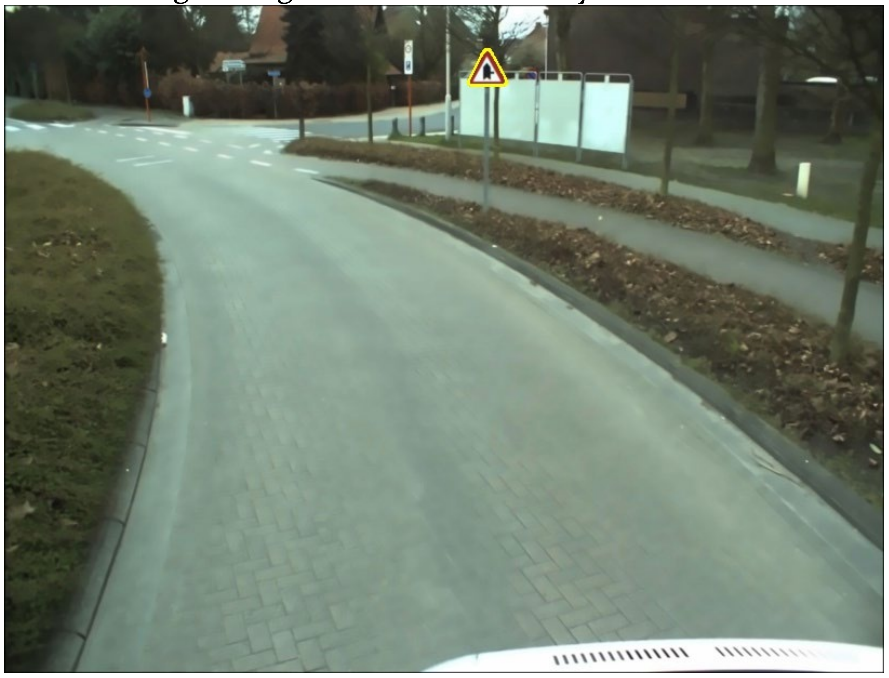
Bn = max ((B-G, B-R)/ (R+G+B),0) - for blue signs

The intensities obtained after normalization look as follows – Blue region then Red region after normalization.

4. **MSER region extraction** –
   In this step we used the mser.detectRegions() functions to find the possible regions in the image where the traffic signs might be. This step requires a bit of tweaking as by default the function detects many false positive regions. The parameters we chose were as follows –
   cv2.MSER_create (_delta = 13, _min_area = 400, _max_area = 3000)
   These values gave us decent outputs which are shown below –



The above figure shows blue parking sign being successfully detected and below shows red sign being detected successfully.

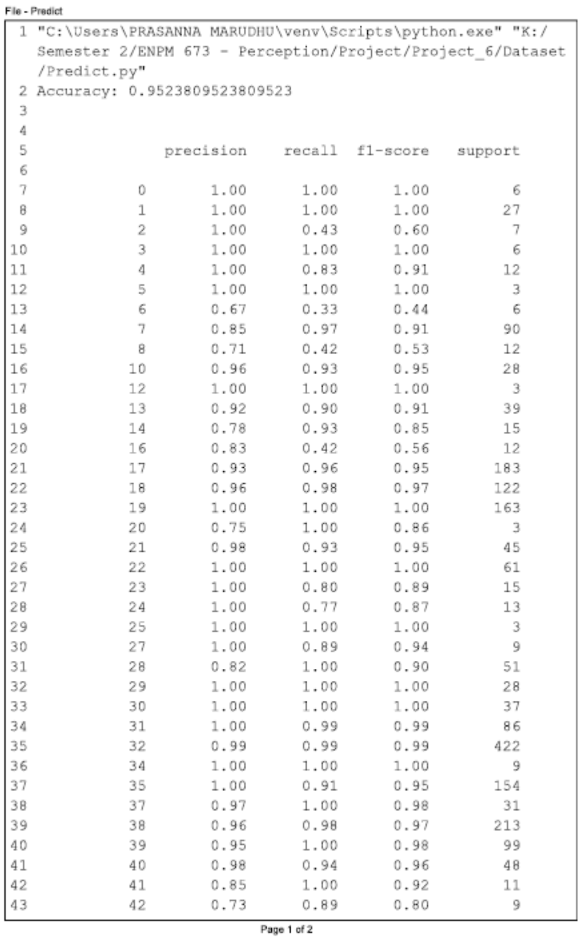


To improve the output further and to reduce the number of falsely detected regions we wrote a function sign_extract () that restricts the aspect ratio to the range of (0.6, 1.5). Also, the width and height of detected regions whose values were below 20 were ignored. We also wrote few extra lines of code to put constraints on the x and y values of the box.

TRAFFIC SIGN CLASSIFICATION

The first step was to read the images (training) provided. The next step was to extract the features for which we used HOG feature detector of OpenCV hog.compute(). We also tried HOG detector of scikit but found the OpenCV's implementation to be more robust. Then we saved the computed features and their corresponding labels in two lists Descriptors and Labels respectively. Then we created a classifier for which we used scikit's SVM function svm.SVC. It was created using the entire Training Dataset. The classifier was saved with the name ***Classifier_Data.sav***. This was implemented in the code named ***TrainClassifier.py.***

The classifier stored as ***Classifier_Data.sav*** was loaded for testing on images given in the Testing dataset(provided). We read the images, extracted the HOG features and stored the corresponding data in Descriptors as explained previously. We then passed the Descriptors to clf.predict to determine the corresponding labels. We then got an accuracy of 95% which can be seen in the accuracy report as follows –

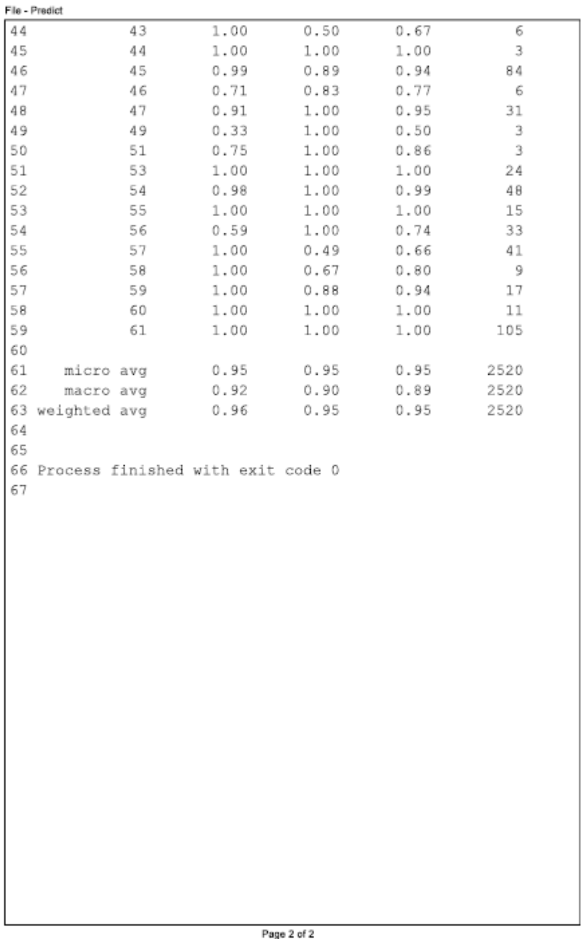

File - Predict

```
"C:\Users\PRASANNA MARUDHU\venv\Scripts\python.exe" "K:/
Semester 2/ENPM 673 - Perception/Project/Project_6/Dataset
/Predict.py"
Accuracy: 0.9523809523809523


              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00        27
           2       1.00      0.43      0.60         7
           3       1.00      1.00      1.00         6
           4       1.00      0.83      0.91        12
           5       1.00      1.00      1.00         3
           6       0.67      0.33      0.44         6
           7       0.85      0.97      0.91        90
           8       0.71      0.42      0.53        12
          10       0.96      0.93      0.95        28
          12       1.00      1.00      1.00         3
          13       0.92      0.90      0.91        39
          14       0.78      0.93      0.85        15
          16       0.83      0.42      0.56        12
          17       0.93      0.96      0.95       183
          18       0.96      0.98      0.97       122
          19       1.00      1.00      1.00       163
          20       0.75      1.00      0.86         3
          21       0.98      0.93      0.95        45
          22       1.00      1.00      1.00        61
          23       1.00      0.80      0.89        15
          24       1.00      0.77      0.87        13
          25       1.00      1.00      1.00         3
          27       1.00      0.89      0.94         9
          28       0.82      1.00      0.90        51
          29       1.00      1.00      1.00        28
          30       1.00      1.00      1.00        37
          31       1.00      0.99      0.99        86
          32       0.99      0.99      0.99       422
          34       1.00      1.00      1.00         9
          35       1.00      0.91      0.95       154
          37       0.97      1.00      0.98        31
          38       0.96      0.98      0.97       213
          39       0.95      1.00      0.98        99
          40       0.98      0.94      0.96        48
          41       0.85      1.00      0.92        11
          42       0.73      0.89      0.80         9
```

Page 1 of 2


File - Predict

```
          43       1.00      0.50      0.67         6
          44       1.00      1.00      1.00         3
          45       0.99      0.89      0.94        84
          46       0.71      0.83      0.77         6
          47       0.91      1.00      0.95        31
          49       0.33      1.00      0.50         3
          51       0.75      1.00      0.86         3
          53       1.00      1.00      1.00        24
          54       0.98      1.00      0.99        48
          55       1.00      1.00      1.00        15
          56       0.59      1.00      0.74        33
          57       1.00      0.49      0.66        41
          58       1.00      0.67      0.80         9
          59       1.00      0.88      0.94        17
          60       1.00      1.00      1.00        11
          61       1.00      1.00      1.00       105

   micro avg       0.95      0.95      0.95      2520
   macro avg       0.92      0.90      0.89      2520
weighted avg       0.96      0.95      0.95      2520


Process finished with exit code 0

```

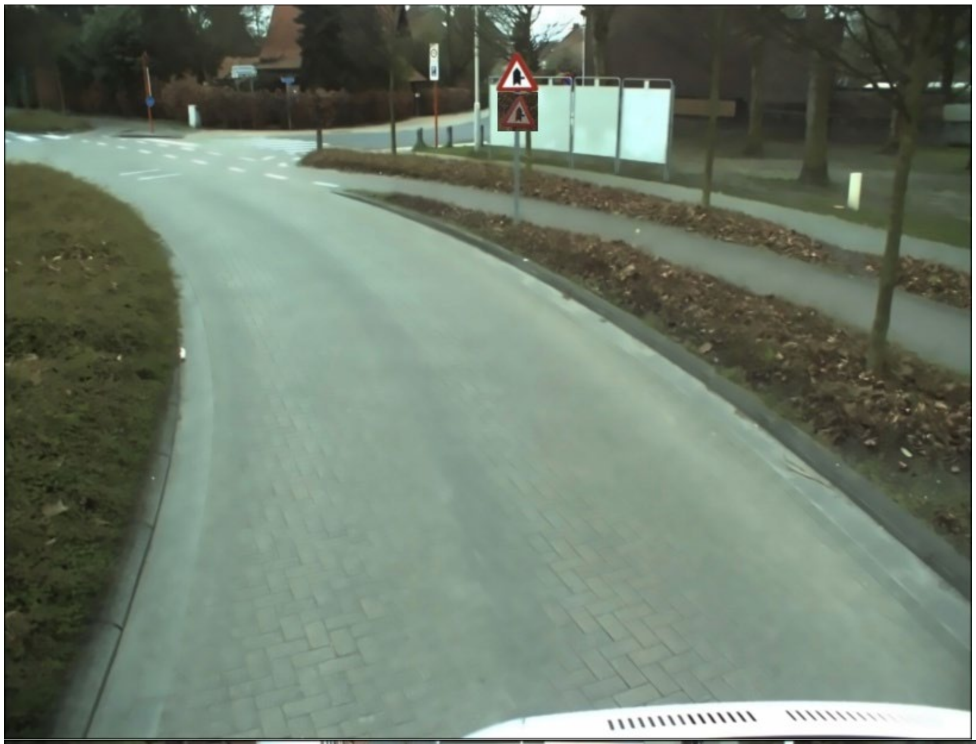
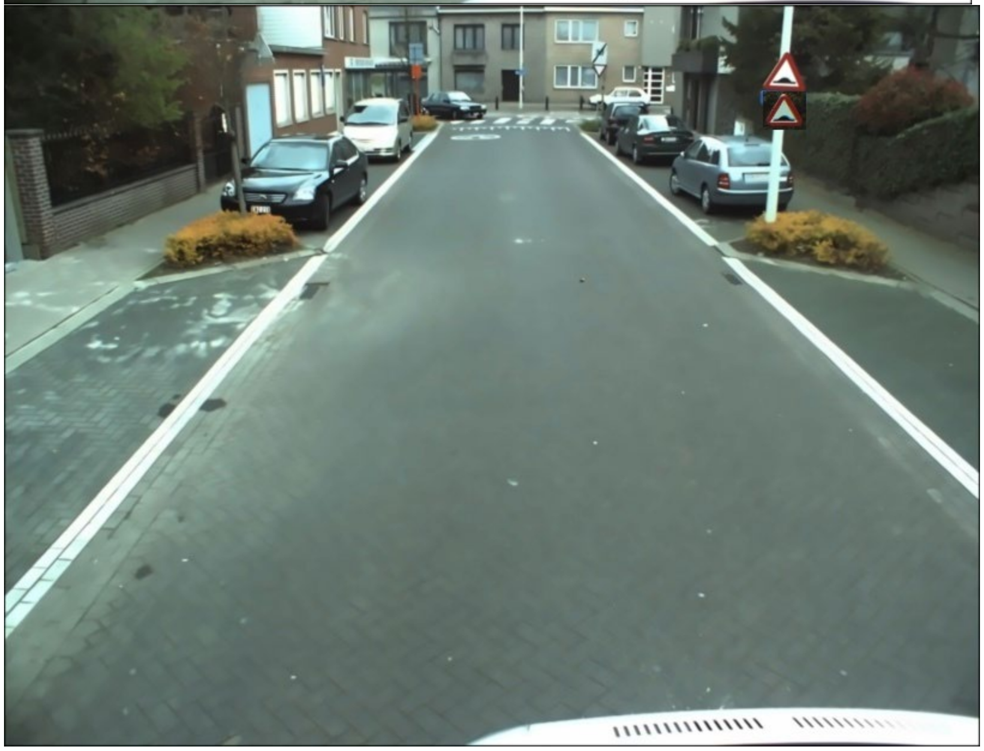
Page 2 of 2

The ROI of each detected sign was extracted, resized and then its features were detected using OpenCV HOG which was then given to the classifier fuction ***clf.predict_proba***. The output of this function is a list with all the probabilities of all the labels. We extracted the maximum value of this list and its index which was our desired label number. We then created a function to paste the corresponding images next to the signs detected. The said images are provided in the link below -

https://drive.google.com/open?id=1czFF6kJFAdGCAqmQDo7SAGTdcmtX-V8q

The outputs of this stage are as follows -

P
P

CONCLUSION

We followed most of the instructions given in the project guidelines and were able to satisfactorily detect, classify and generate the desired output for the Traffic sign recognition problem.

REFERENCES

a) OpenCV documentation
   a. https://docs.opencv.org/3.4/d3/d28/classcv_1_1MSER.html#a136c6b29fbcdd783a1003bf429fa17ed
   b. https://docs.opencv.org/3.3.1/d5/d69/tutorial_py_non_local_means.html
   c. https://pythontic.com/image-processing/pillow/contrast%20stretching

b) MSER theory slides
   http://www.micc.unifi.it/delbimbo/wp-content/uploads/2011/03/slide_corso/A34%20MSER.pdf

c) OpenCV tutorial for Digit Classification
   https://www.learnopencv.com/handwritten-digits-classification-an-opencv-c-python-tutorial/

d) Traffic Sign Dataset from ETH Zurich
   M. Mathias, R. Timofte, R. Benenson, and L. Van Gool. Tra_c sign recognition | how far are we from the solution? In The 2013 International Joint Conference on Neural Networks (IJCNN), pages 1{8, Aug 2013.