“[ENPM 673] PROJECT 6”

ENPM 673 – PERCEPTION FOR AUTONOMOUS ROBOTS

PROJECT 6 REPORT

ADITYA VAISHAMPAYAN -116077354

ISHAN PATEL – 116169103

NAKUL PATEL – 116334877

Instructor: Dr. Cornelia Fermuller

CONTENTS

Detection:

According to the pipeline that we have been given, The Traffic Sign Detection contains two parts:

1. Traffic Sign detection
2. Traffic Sign classification

In traffic sign detection we make a bounding box around the traffic sign so that the region of interest can be cropped, and hence can be given to a classifier for classification.

We have combined both the approaches of HSV and MSER to obtain a robust output of sign detection.

Pipeline:

1. We Denoise the image in respective RGB planes using the following filter: FastNIMeansDenoising() and FastNIMeansDenoisingColored()
2. Next, we apply contrast Normalization. There were many approaches tried in contrast normalization. Such as follows:
   a. Individually doing histogram equalization of each plane of an RGB image
   b. Using gamma correction, or inverse gamma correction for the individual RGB planes of the image and then merging them to together
   c. We also tried min max normalization for contrast enhancement
   d. We also explored several color spaces such HSV, YUV, YrcLab and others for contrast normalization
3. Next, we Normalize the intensity of the image as follows using the given formulae in the question:

$$\text{For Red Channel: } C' = max\left(0, \frac{min(R-B, R-G)}{R+G+B}\right)$$

$$\text{For Blue Channel: } C' = max\left(0, \frac{min(B-R)}{R+G+B}\right)$$

Fig: Blue plane obtained after doing the above step

Fig: Red plane obtained after doing the above step

4. HSV color Thresholding: HSV Color space is important for detection of blobs, considering the illumination changes. It helps to detect red and blue color easily by tweaking the value of hue channel. In this project, to detect the red color, the range of RGB limit was kep as follows:

- mask_r1 = cv2.inRange(hsv, np.array([0, 70, 100]), np.array([15, 255, 255]))
- mask_r2 = cv2.inRange(hsv, np.array([160, 70, 100]), np.array([180, 255, 255]))

   Here, the lower and upper limits for the ranges have been used, to detect the red colors easily. Similarly for blue color, folowing mask has been used:

   mask_b = cv2.inRange(hsv, np.array([100,45,50]), np.array([130,250,250]))

   Finally, the masks(for red and blue sign) were smothed with the help of Gaussian filter of 5*5 kernel.

5. Detecting MSER Features
   We used the following command to obtain the MSER features:
   Mser = cv2.MSER_create (delta = 10, min_area = 600, max_area = 1600, max_variation = 0.2)

- Delta: he parameter *delta* indicates through how many different gray levels does a region need to be stable to be considered maximally stable. For a *larger delta*, you will get *less regions*.
- **minArea**, **maxArea :** If a region is maximally stable, it can still be rejected if it has less than *minArea* pixels or more than *maxArea* pixels.
- **maxVariation:** Back to the variation from point 1 (the same function as for delta): if a region is maximally stable, it can still be rejected if the the regions variation is bigger than *maxVariation*.

  That is, even if the region is "relatively" stable (more stable than the neigbouring regions), it may not be "absolutely" stable enough. For *smaller maxVariation*, you will get *less regions*

  Thus, finally we concluded for the following parameters:

  - **delta = 10,**
  - **min_area = 600,**
  - **max_area = 1600**
  - **max_variation = 0.2**

  We have also considered HSV thresholding for taking care of the excess noise that I generated by the delta parameter. We majorly use this value of delta in order to access farther regions. Hence upon noticing closely one can find that the far of signs can also be detected easily. The Area range is set so as to remove noise regions.

  We have also combined The MSER regions with the HSV regions to get a better output

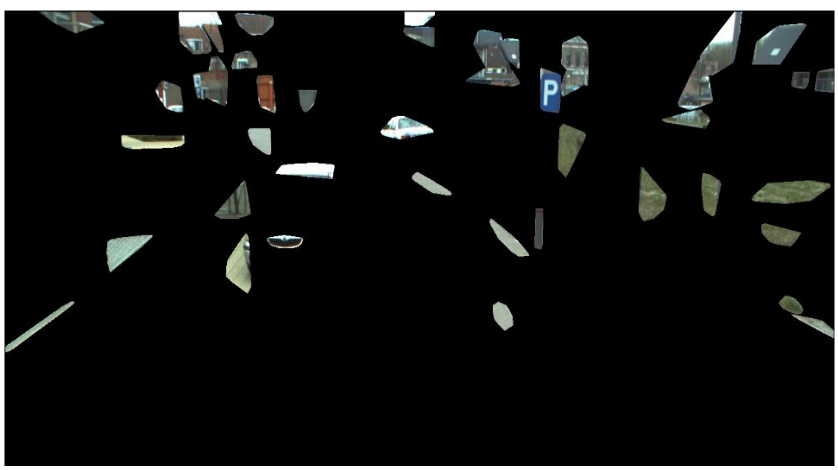


Fig: MSER regions extracted

6. The next step involves placing bounding boxes around the detected region. There was some noise present even after filtering. Hence to remove the noise, we had to use the aspect ratio. It's the ratio of width and height. We use the aspect ratio rejection beyond 0.6 and 1.2 i.e. any box with aspect ratio less than 0.6 will be deleted and any box with aspect ratio greater

than 1.2 won't be considered. The major reason behind doing this is that the traffic signs are mainly rectangular, square in shape. There are also circular and triangular shapes but they all can be fit within the bounding box of aspect ratio between 0.6 and 1.2

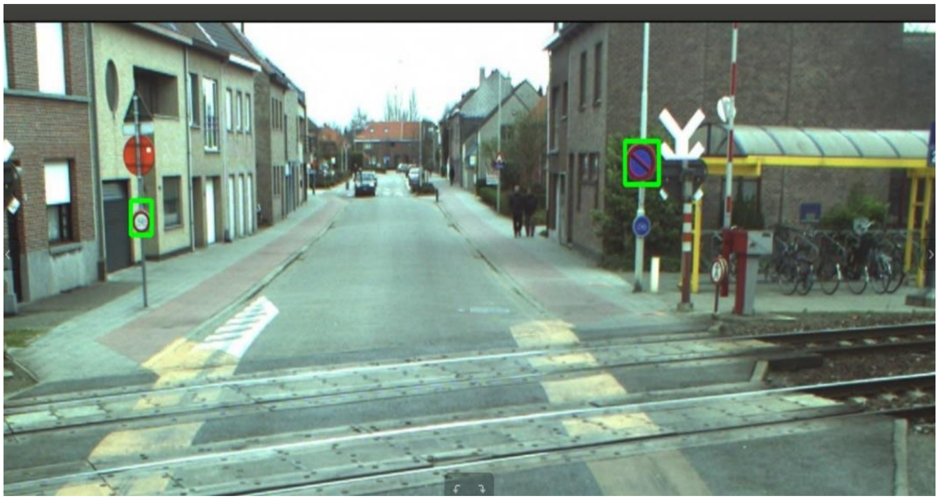


Fig: Detected bounding boxes

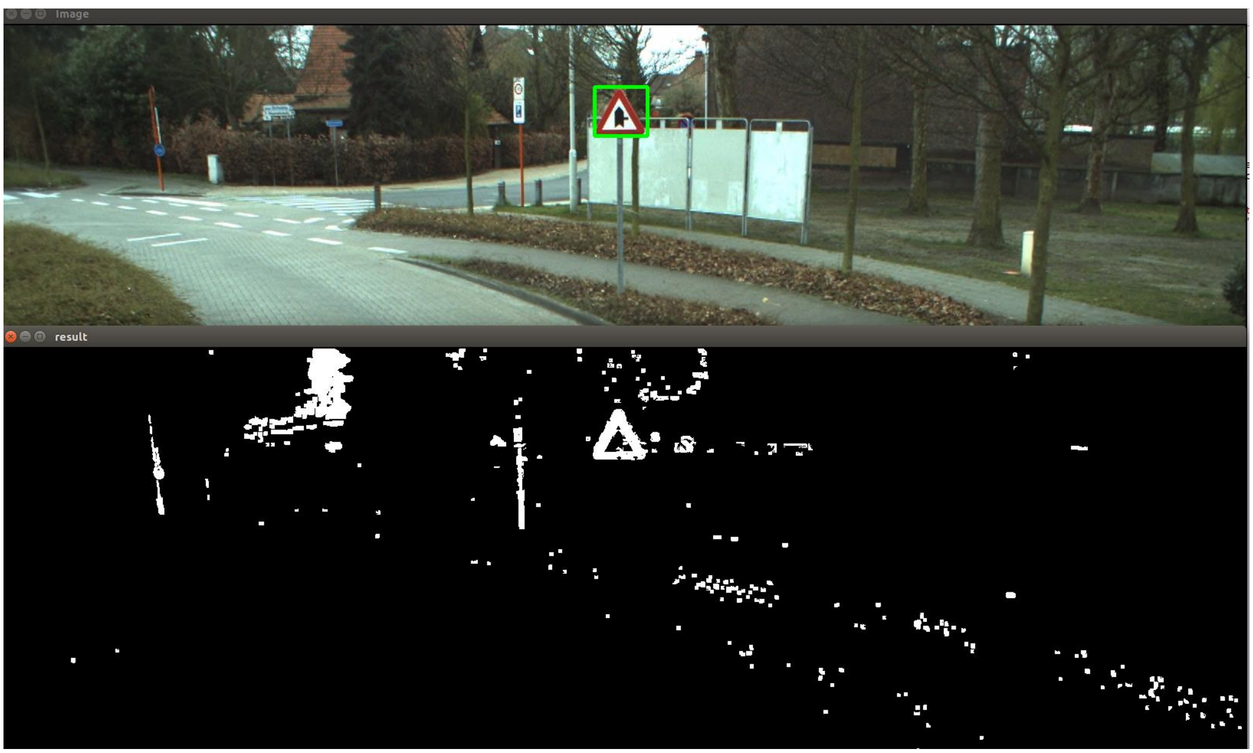
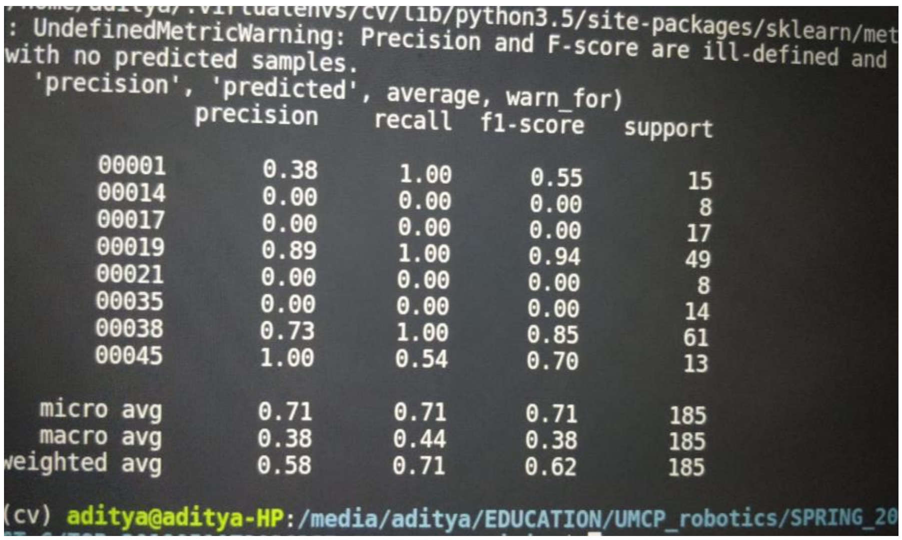


Fig: Binary image obtained from an HSV image and the bounding box

Classification:

Now we start the second part of the project that is the traffic sign classification. We have use SVM (support vector machine) for creating the classifier. Also, the kernel that is used in a **linear kernel**. Since, it gave the highest accuracy. The other kernels used were **rbf and poly.** However, those kernels gave a poor accuracy and also upon obtaining the classifier report in SVM of scikit learn, the precision of some of the classes came to zero. Whereas the precision of the Linear Kernal was pretty high and so was the accuracy.

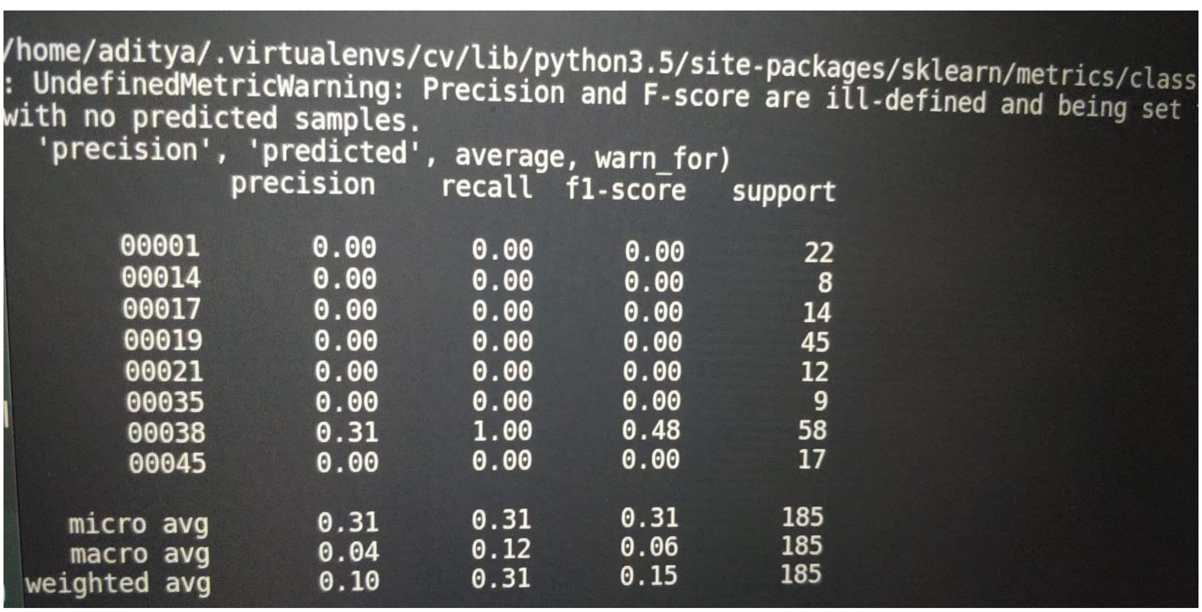


```
.virtualenvs/cv/lib/python3.5/site-packages/sklearn/met
: UndefinedMetricWarning: Precision and F-score are ill-defined and
with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

	precision	recall	f1-score	support
00001	0.38	1.00	0.55	15
00014	0.00	0.00	0.00	8
00017	0.00	0.00	0.00	17
00019	0.89	1.00	0.94	49
00021	0.00	0.00	0.00	8
00035	0.00	0.00	0.00	14
00038	0.73	1.00	0.85	61
00045	1.00	0.54	0.70	13
micro avg	0.71	0.71	0.71	185
macro avg	0.38	0.44	0.38	185
weighted avg	0.58	0.71	0.62	185

```
(cv) aditya@aditya-HP:/media/aditya/EDUCATION/UMCP_robotics/SPRING_20
```

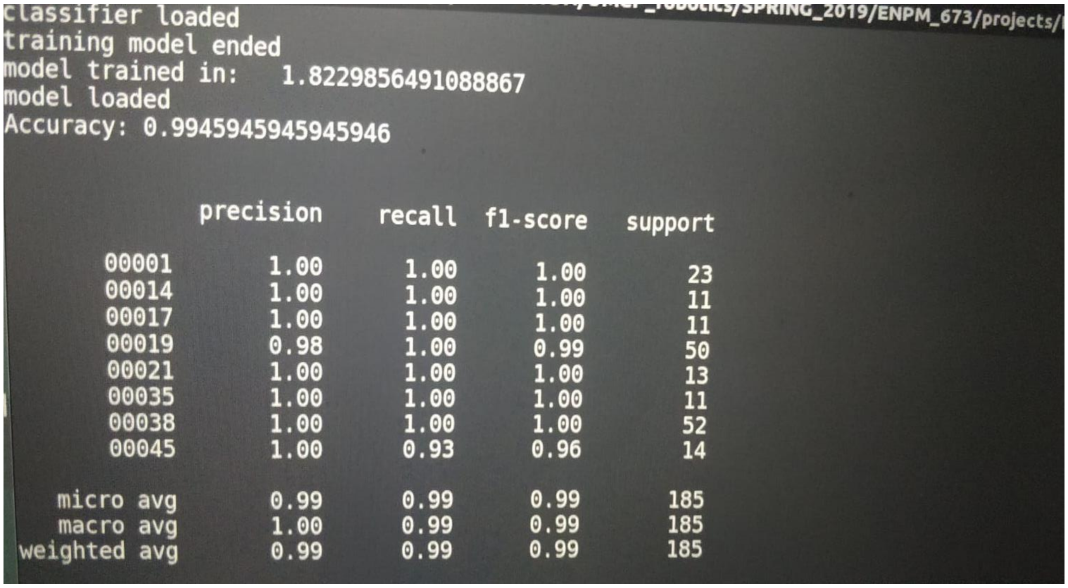
Fig: Scikit Learn Classification report for RBF kernel



```
/home/aditya/.virtualenvs/cv/lib/python3.5/site-packages/sklearn/metrics/class
: UndefinedMetricWarning: Precision and F-score are ill-defined and being set
with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

	precision	recall	f1-score	support
00001	0.00	0.00	0.00	22
00014	0.00	0.00	0.00	8
00017	0.00	0.00	0.00	14
00019	0.00	0.00	0.00	45
00021	0.00	0.00	0.00	12
00035	0.00	0.00	0.00	9
00038	0.31	1.00	0.48	58
00045	0.00	0.00	0.00	17
micro avg	0.31	0.31	0.31	185
macro avg	0.04	0.12	0.06	185
weighted avg	0.10	0.31	0.15	185

Fig: Scikit Learn Classification report for poly kernel



```
classifier loaded
training model ended
model trained in:   1.8229856491088867
model loaded
Accuracy: 0.9945945945945946
```

	precision	recall	f1-score	support
00001	1.00	1.00	1.00	23
00014	1.00	1.00	1.00	11
00017	1.00	1.00	1.00	11
00019	0.98	1.00	0.99	50
00021	1.00	1.00	1.00	13
00035	1.00	1.00	1.00	11
00038	1.00	1.00	1.00	52
00045	1.00	0.93	0.96	14
micro avg	0.99	0.99	0.99	185
macro avg	1.00	0.99	0.99	185
weighted avg	0.99	0.99	0.99	185

Fig: Scikit Learn classification report for Linear Kernel

The pipeline for Traffic Sign classification is as follows:

1. We have loaded the training images into a list and then we access them one by one to make the HOG feature vector out of them
2. Then again, we save these HOG feature vectors into another list. This acts as a training data set for our classifier.
3. We also need to create a labels list. The way we create the labels list is that each HOG feature vector has the folder name as a class.
4. Thus we finally hstack the feature vectors and the labels and thus make the dataset.
5. Now we fit the model and thus, do clf.predict to obtain a classifier
6. Now we obtain attest image, obtain the HOG feature vector and pass it to the classifier
7. This results into a classification, such as [00001] or anything else.
8. We also have used the class probability function. This gives as an output a list. And we find the maximum element value out the list.
9. If the maximum value is above the threshold then we consider that class else, we dump it.
10. This way we pass the several detected regions from the frame into the classifier. And the region with the highest probability is considered and assigned a class as well.
11. Once we know the class, we just have to place the image of the classified traffic sign next to the bounding box.

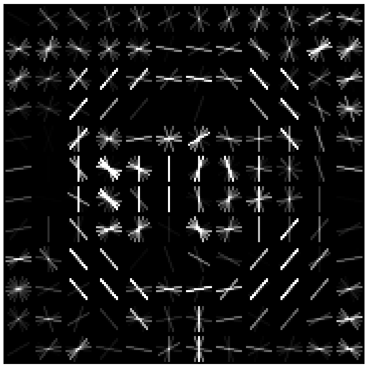


Fig: HOG for STOP sign

Fig: Classified Traffic Sign having Blue background

Fig: Classified Traffic Sign having Red Background

Refercences:

1. https://stackoverflow.com/questions/17647500/exact-meaning-of-the-parameters-given-to-initialize-mser-in-opencv-2-4-x
2. Lecture notes
3. “ A Traffic Sign Detection pipeline based on interest region extraction- Samuele Salti, Alioscia Petrelli, Federico Tombari, Nicola Fioraio and Luigi Di Stefano”