Project 6 - Traffic Sign Detection

By- Mrinali Vyas

srujan Panuganti

Akshita Pothamshetty

Instructions to run the code

Import the following files :

```
import numpy as np
import cv2
import sys
import math
from sklearn import svm
from PIL import Image
import recognize
import glob
import imutils
```

In order to run the code import the dataset provided in the folder and run the file detection.py

The google drive link for the project is as below:

https://drive.google.com/open?id=1HbnOVec68xxAA3WJnFzqdz-2oTsEYAUd

Introduction :

The aim of this project is to recognise traffic sign, there are 2 parts to the project-

Part 1 - Detection- Detecting the red and blue signs.

Part 2 - Recognition- Recognising the signs.

In the Detection stage, we extract possible regions which contain a traffic sign. In the Recognition stage, we go over each Region extracted above to identify the appropriate traffic sign.

Step- 1: Preparing the data

We are given images from a driving car, training and testing images for a set of signs. We were asked to detect 8 signs with the labels [1, 14, 17, 21, 35, 38, 45].

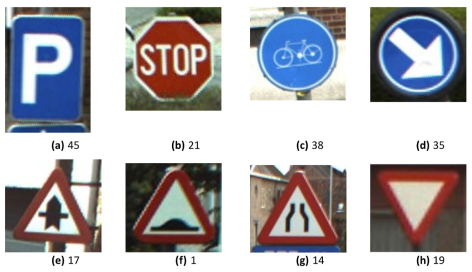

Fig: Signs to be predicted

From the dataset, we separate the data of the relevant signs into different folders and train the classifier using that.

The input video frame is first converted to the hsv colour space because converting from RGB to HSV, the color features can be extracted more easily. We then equalise the histogram of the '_ channel'.

Fig: HSV image of the input data

Step 2: Red Detection

The input image to the MSER function which is the feature detector used is first converted to hsv colour space. The MSER input image needs to be a gray image which is why we have created a custom grayscale by creating 2 masks for the red colour. This mask is bitwise-or with the input image and then we filter each channel of the masked image to remove all the noise from the data.

```
lower_red_1 = np.array([0,70,60])
upper_red_1 = np.array([10,255,255])
lower_red_2 = np.array([170,70,100])
upper_red_2 = np.array([180,255,255])
```

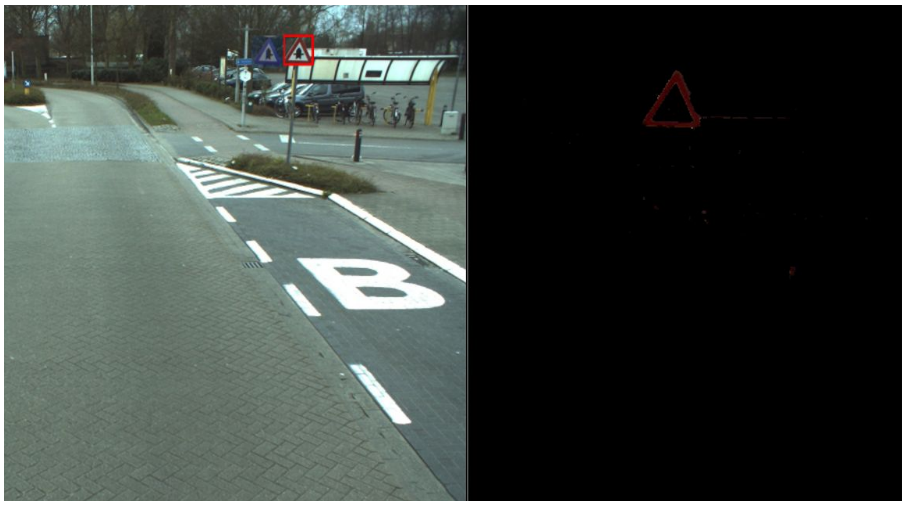

Fig: Red mask image for the red sign

The custom gray scale is defined as displayed below:

```
filtered_r = 3*filtered_r - 0.5*filtered_b - 2*filtered_g
```

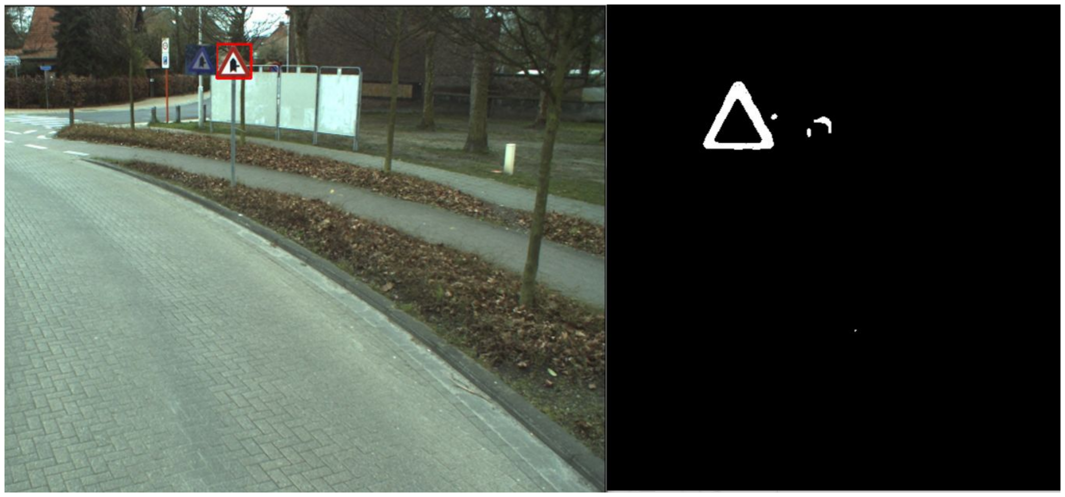

Fig: Custom Grayscale for red signs

This gray image is given to the MSER detector to find the regions of interest. The MSER parameters used for the red signs are delta = 8, minimum area = 400, maximum area = 2000.

We then store these regions by creating a bounding rectangle on these regions and storing them in a list.

On this image, we then perform morphological transformations by using the opening function in order to remove the noise from the image.

```
opening = cv2.morphologyEx(dilation, cv2.MORPH_OPEN, kernel_2)
```

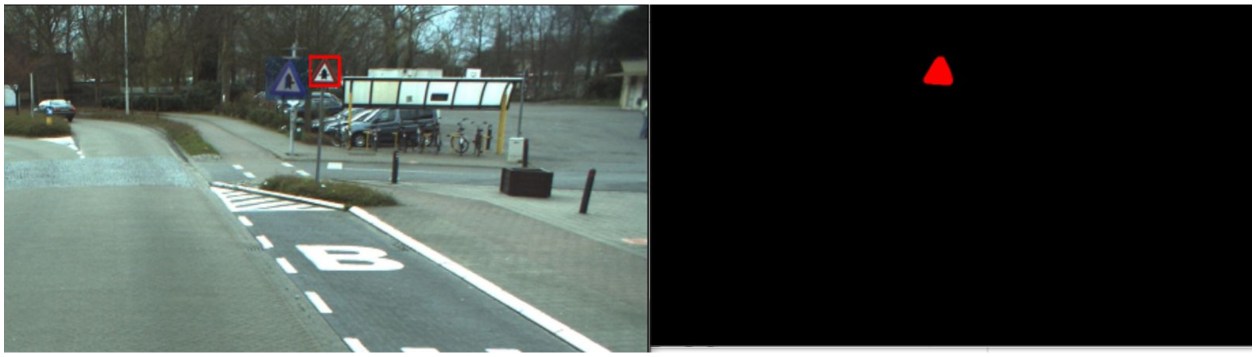

Fig: Denoised image

We then threshold the images for the red channel to 60 and then we find contours on the thresholded image.

These contours are sorted from left to right and the max contours detected per frame is fixed to 3. We then make the bounding rectangle on these contours and perform certain checks on its aspect ratio and its dimensions.

```
aspect_ratio<=0.3 or aspect_ratio>1.2
x<800 or h<20
```

We then locate the pixels of where the contour was detected and crop that region and shape it to the image of (64 x 64) and make a list of all the final detected contour.

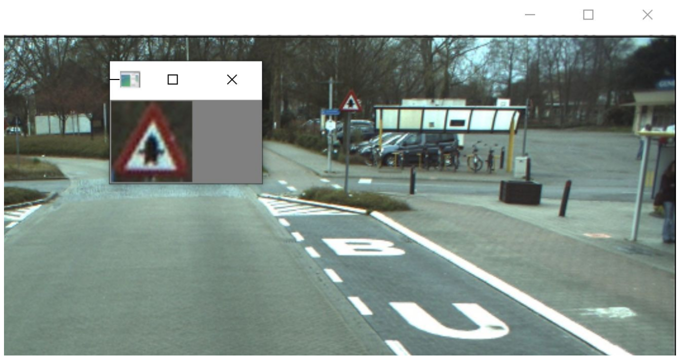

Fig: Final cropped image of the red sign

This cropped image list is then sent to the SVM model.

Step 4: Blue Detection

In this step, we perform the same steps as above with a certain variation in its parameters used.

The mask created for blue for the custom grayscale is

```
lower_blue_1 = np.array([94,127,20])
upper_blue_1 = np.array([126,255,200])
```

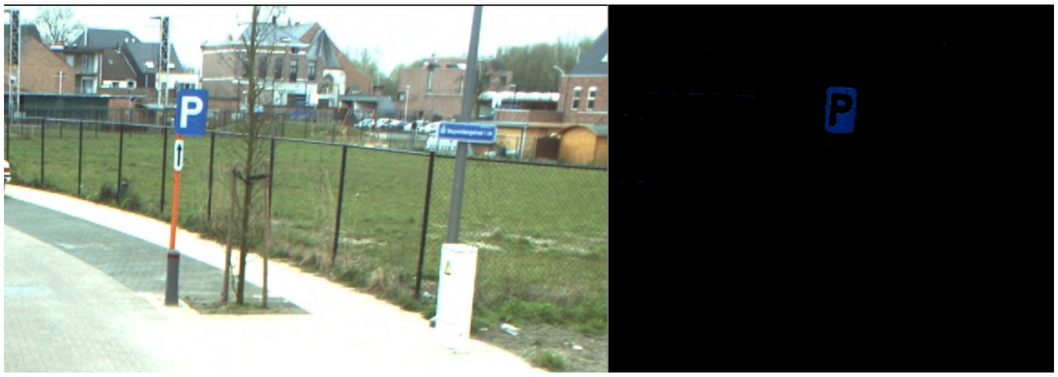

Fig: blue mask image for the blue sign

The grayscale was formed in the following manner

```
filtered_b_b = -0.5*filtered_r_b + 3*filtered_b_b - 2*filtered_g_b
```

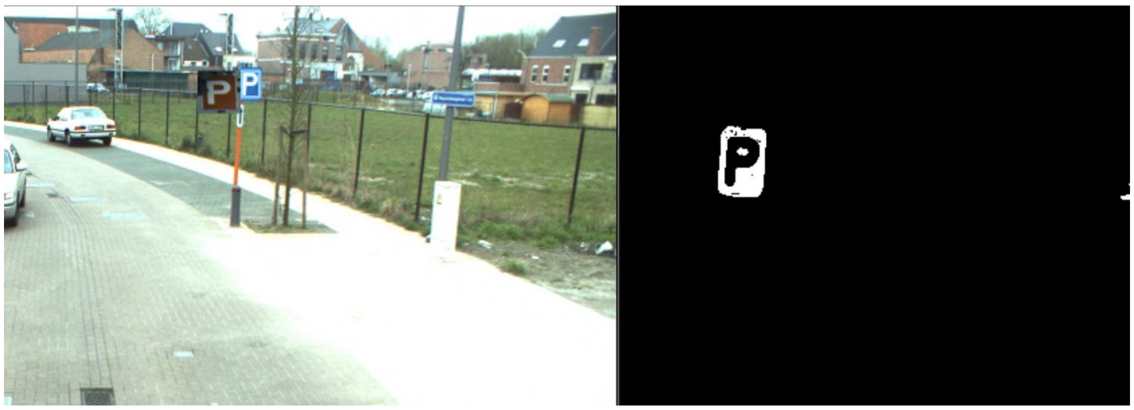

Fig: Custom Grayscale for blue sign

The MSER parameters were delta = 8, minimum area = 400, maximum area = 2000. We then store these regions from the MSER output by creating a bounding rectangle on these regions and storing them in a list. On this image, we then perform morphological transformations by using the opening function in order to remove the noise from the image.

```
opening = cv2.morphologyEx(dilation, cv2.MORPH_OPEN, kernel_2)
```

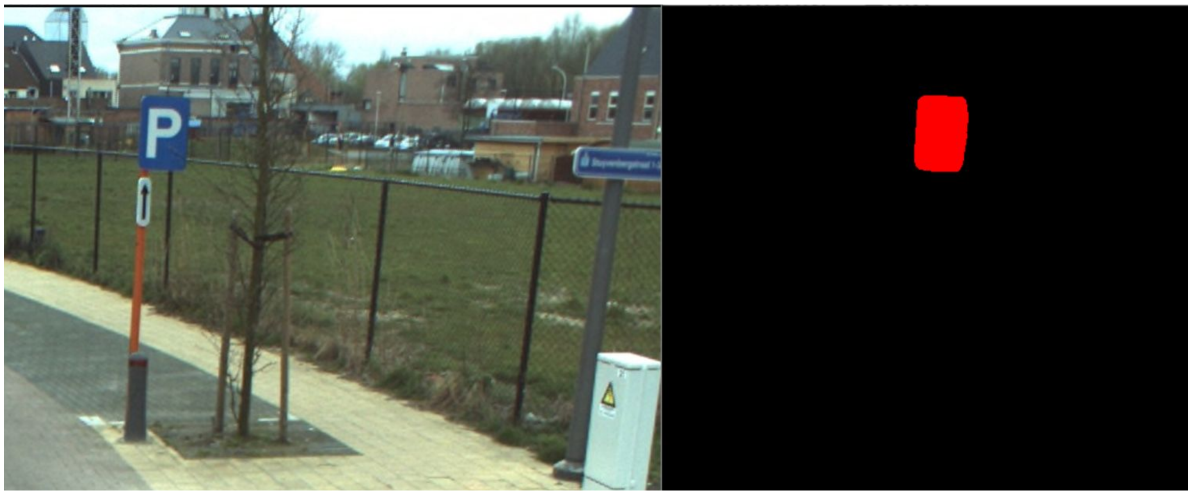

Fig: Denoised image blue

We then threshold the images for channel 2 to 60 and then we find contours on the thresholded image.

The max contours detected per frame is fixed to 3. We then make the bounding rectangle on these contours and perform certain checks on its aspect ratio and its dimensions.

```
aspect_ratio<=0.5 or aspect_ratio>1.5
x<100or h<20
```

We then locate the pixels of where the contour was detected and crop that region and shape it to the image of (64 x 64) and make a list of all the final detected contour.

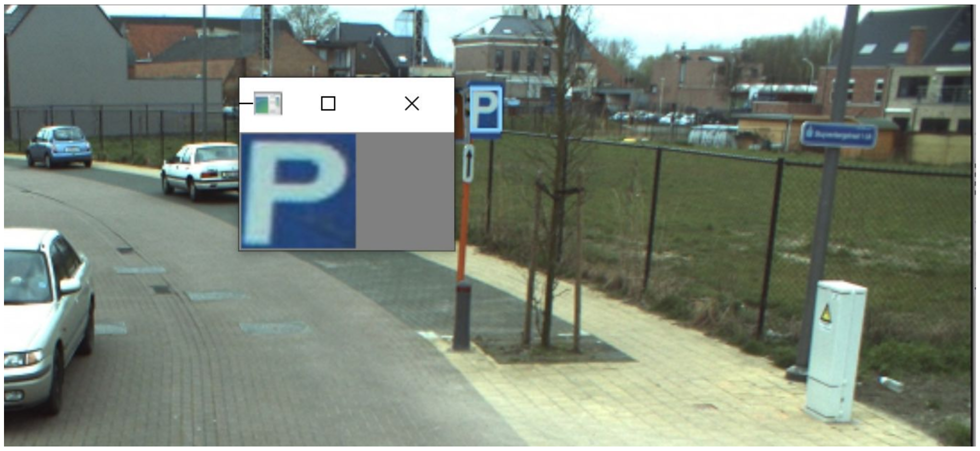

Fig: Final cropped image of the blue sign

This image list is then sent to the SVM model.

Step 5: Training of model

In order to train our model, we first extract all the HOG features of the training images which are then given to the multiclass SVM classifier to train. The parameters for the Hog detector used are :

For RED signs:

```
HOG_vector, HOG_image = hog(image, orientations=9,
pixels_per_cell=(8, 8), cells_per_block=(4, 4), transform_sqrt =
True, block_norm="L1", visualize=True, multichannel=False)
```

Fig: HOG output for the red sign

For BLUE signs:

```
HOG_vector, HOG_image = hog(img, orientations=9, pixels_per_cell=(8,
8),cells_per_block=(4, 4), transform_sqrt = True, block_norm="L1",
visualize=True, multichannel=False)
```

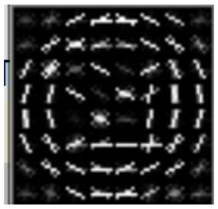

Fig: HOG output for blue sign

This hog function returns a hog vector and a hog image. The feature vector produced by this is fed directly to the classification algorithm SVM this gives us a test accuracy of *97%* .

Step 6: Testing the model.

The regions of interest extracted from steps 3 and 4 are then given to the testing function. In this the cropped regions are converted to gray image. We then extract the hog features as above then give them as input to the trained model to be predicted.

The predicted image is then pasted next to the sign.

Step 7 : Results

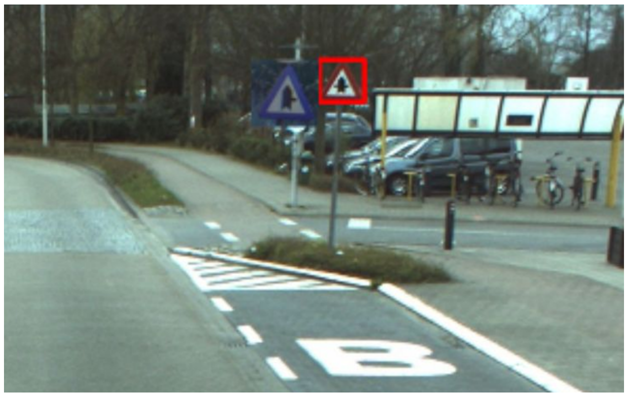

Fig: Final Red predicted image

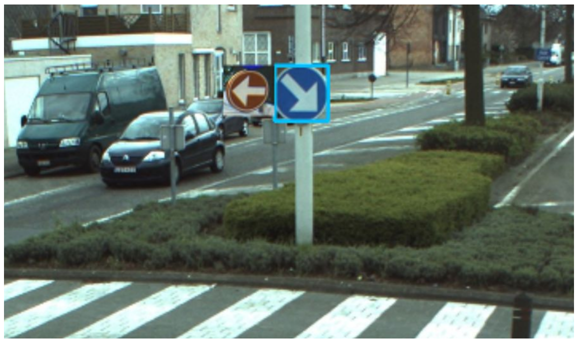

Fig: Final Blue predicted image

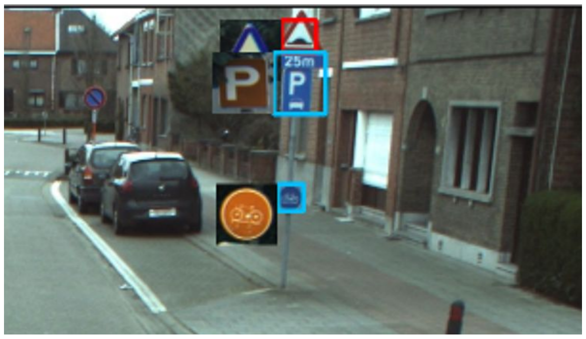

Fig: Final predicted image combined

Step 8: Problems

1. Selecting the right mask for the red and blue signs
2. Setting the parameters of MSER
3. As the algorithm gave many misdetect for the we had to add a few specific images in our training data to avoid them.

Step 9: References

1. http://scikit-learn.sourceforge.net/stable/modules/generated/sklearn.svm.SVC.html
2. https://github.com/opencv/opencv/blob/master/samples/python/mser.py
3. http://www.micc.unifi.it/delbimbo/wp-content/uploads/2011/03/slide_corso/A34%20MSER.pdf
4. https://www.pyimagesearch.com/2014/11/10/histogram-oriented-gradients-object-detection/