Speed Sign Detection

Team members:
Jufeng Yang ID: 20125011
Xingda Zhou ID: 19107471
Zhongen Qin ID: 19107579

Table of Content

1. Introduction **2**

2. Speed Sign Detection **3**

2.1 Import Relative Libraries **3**

2.2 Read pictures and Convert pictures into HSV **3**

2.3 Mask generation **5**

2.4 Find the position of RoIs **6**

3. Speed Sign Classifier

3.1 No Label descript vector generation **10**

3.2 Add Label to descript vectors **12**

3.3 RoIs Classifier **12**

4. Final Speed Sign Detector **15**

5. Conclusion **18**

6. Reference **19**

1. Introduction

The aim of this project is to detect and correctly classify speed signs in images. The process of detection requires that the images are first processed. The process is divided into 5 steps. The first step converts the RGBA image into an HSV image. The second step is to set thresholds for H, S and V respectively and to generate an image Mask. the third step is to label all remaining areas and to eliminate areas that are too small by detecting non-zero areas. The fourth step is to store the coordinates of the remaining regions (Xmin, Xmax, Ymin, Ymax). The fifth step iterates through all the images and detects the RoIs in all the images.

The process of recognition can be divided into 3 steps. The first step flattens the detected regions into a one-dimensional array and combines all N regions into N rows of data. Each row of the description vector represents one region. The second step sets the first element of the description vector as a label and refines the label element. The third step reads the new data and calculates the Euclidean distance to get the distance between the current region and the nearest description vector, returning the label of the current vector.

The final call to the two methods before draws the region of RoIs on the original image and labels the velocity values.

2. Speed Sign Detection

2.1 Import Relative Libraries

Import imutils, which contains image reading functions, image display functions, channel reading functions, etc. numpy is used for array data. scipy.ndimage is used for maximum filter and binary erosion. Import PIL to show pictures. Import OS to read pictures from memory. Import matplotlab for display images. Import skimage to convert image from RGBA to HSV and gray and resize function.

```
# Team menbers        ID
# Jufeng Yang         20125011
# Xingda Zhou         19107471
# Zhongen Qin         19107579

# Import numpy, imutils, os, PIL, skimage and so on
# import skimage color and transform
import numpy as np
import imutils
import os
import scipy.ndimage
from PIL import Image
import matplotlib.pyplot as plt
import skimage.io
from skimage.color import rgba2rgb, rgb2hsv, rgb2gray, gray2rgb
from skimage.transform import resize
```

Figure.1: Useful libraries.

2.2 Read pictures and Convert pictures into HSV

All the photos in memory are read out and stored in a four-dimensional array as shown in Figure 2. And one of the pictures is displayed as shown in Figure 3

```
# Define a method to read all images from memory
def load_images_from_folder(folder):
    # Create a list data of images
    images = []
    # Go through the folder and output the filename
    for filename in os.listdir(folder):
        # Read the pictures
        #img = plt.imread(os.path.join(folder,filename))
        img = imutils.imread(os.path.join(folder,filename), greyscale = False)
        # to append data to a image variable
        if img is not None:
            images.append(img)
    return images

# Check the method works or not
images_rgba = load_images_from_folder('./speed-sign-test-images/')
images_rgba = np.array(images_rgba)
print(images_rgba.shape)
#for i in range(0, images_rgba.shape[0]):
#    imutils.imshow(images_rgba[i,:,:,:])

# Print 1 picture
imutils.imshow(images_rgba[6,:,:,:])
```

Figure.2: Read images from memory.

As shown in Figure 3, there are a total of 28 photos read from memory, each with an RGBA data size of 960*1280*3.

Figure.3: One picture and the total array dimension of pictures.

This one image read was transformed to convert the RGBA photo into an HSV image and display the different channels separately. The results are shown in Figure 4.

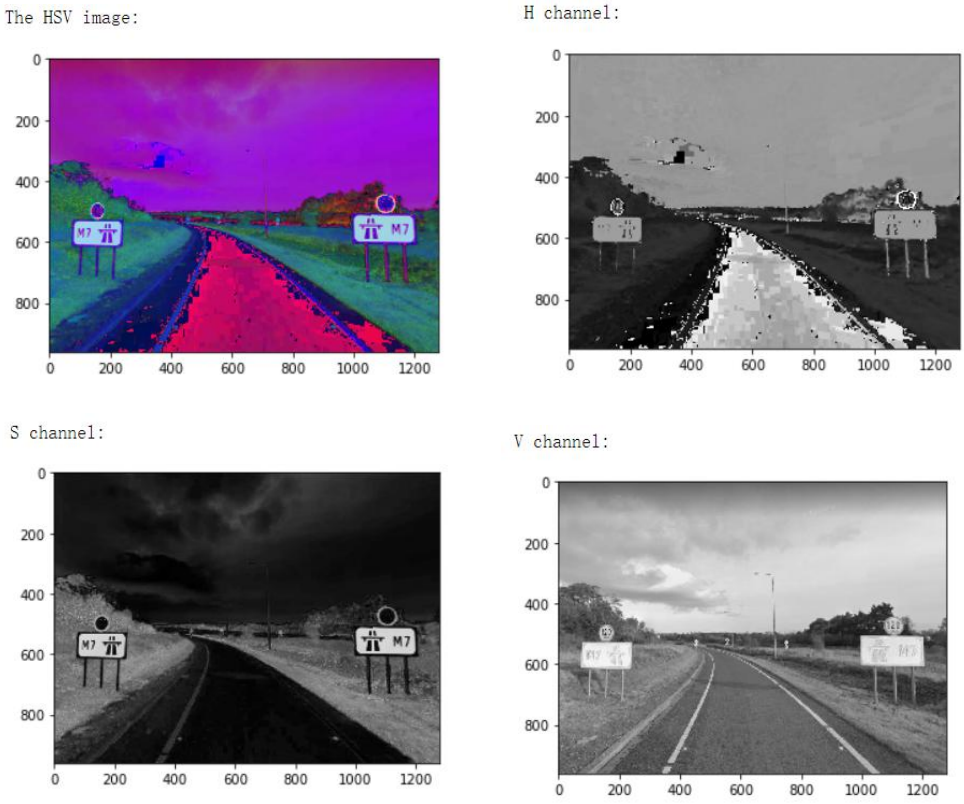


Figure.4: HSV(Upper Left), H channel(Upper Right), S channel(Down Left), V channel(Down Right)

2.3 Mask generation

Filtering is done in this section by using thresholds for the different channels of the HSV. Using the Hue channel shows the red circles of the speed markers, using Saturation on the lower red areas, everything else is filtered. Use Value to fade out most of the image content. Combine the three channels to filter out all non-red areas. The filtering operation is shown in Figure 5. The output is shown in Figure 6.

```
# Create a mask for the H channel
print("Hue Threshold ( > 0.85):")
Hue_mask_low = images_h > 0.85
imutils.imshow(Hue_mask_low)

# Create a mask for the S channel
print("\n\nSat Threshold ( > 0.3):")
Sat_mask_low = images_s > 0.3
imutils.imshow(Sat_mask_low)

# Create a mask for the V channel
print("\n\nVal Threshold ( > 0.22):")
Val_mask_low = images_v > 0.22
imutils.imshow(Val_mask_low)

# Conbine all mask together
print("\n\nCombined all threshold into a Mask:")
Mask = Hue_mask_low*Sat_mask_low*Val_mask_low
imutils.imshow(Mask)
```

Figure.5: The implementation of channels mask.

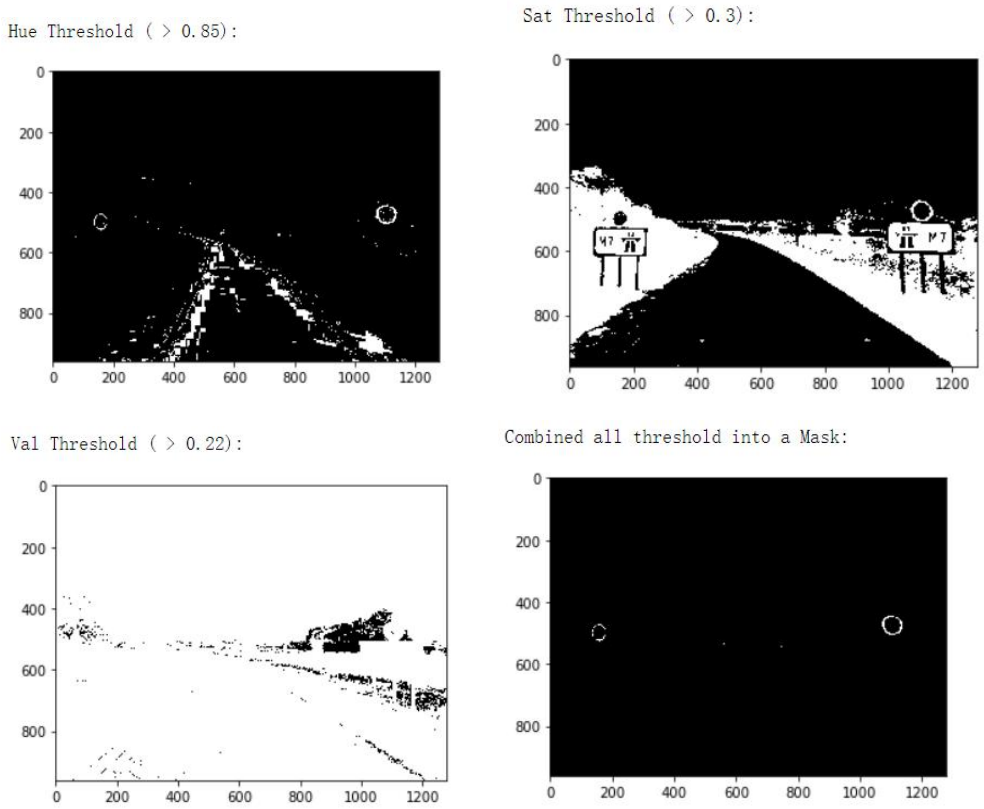


Figure.6: H mask(Upper Left), S mask(Upper Right), V mask(Down left), Combined(Down Right)

2.4 Find the position of RoIs

This is the most important part of the whole project. Find all the RoIs and return the Xmin, Xmax,Ymin, Ymax coordinates of the area and calculate the range of RoIs by Xmax - Xmin, Ymax - Ymin.

Before the Mask process, all non-zero data present in the image is enhanced with a "maximun filter". A "binary erosion" is then used to remove the very small non-zero areas of data. After this process, a picture with a small number of RoI regions is obtained. All RoIs marked. The implementation is shown in Figure 7 and the output data is shown in Figure 8.

```
# Call a filter to enhanced ROI
print("After enhance: ")
Mask_max = scipy.ndimage.maximum_filter(Mask, size = 13)
imutils.imshow(Mask_max)

# Call a binary erosion to eliminate data
print("\nAfter elimination: ")
Mask_erosion = scipy.ndimage.binary_erosion(Mask_max)
imutils.imshow(Mask_erosion)

# Label those ROIs
print("After label:")
Mask_labeled, num = scipy.ndimage.label(Mask_erosion)
imutils.imshow(Mask_labeled)
```

Figure.7: Enhance, Erosion and Label process.

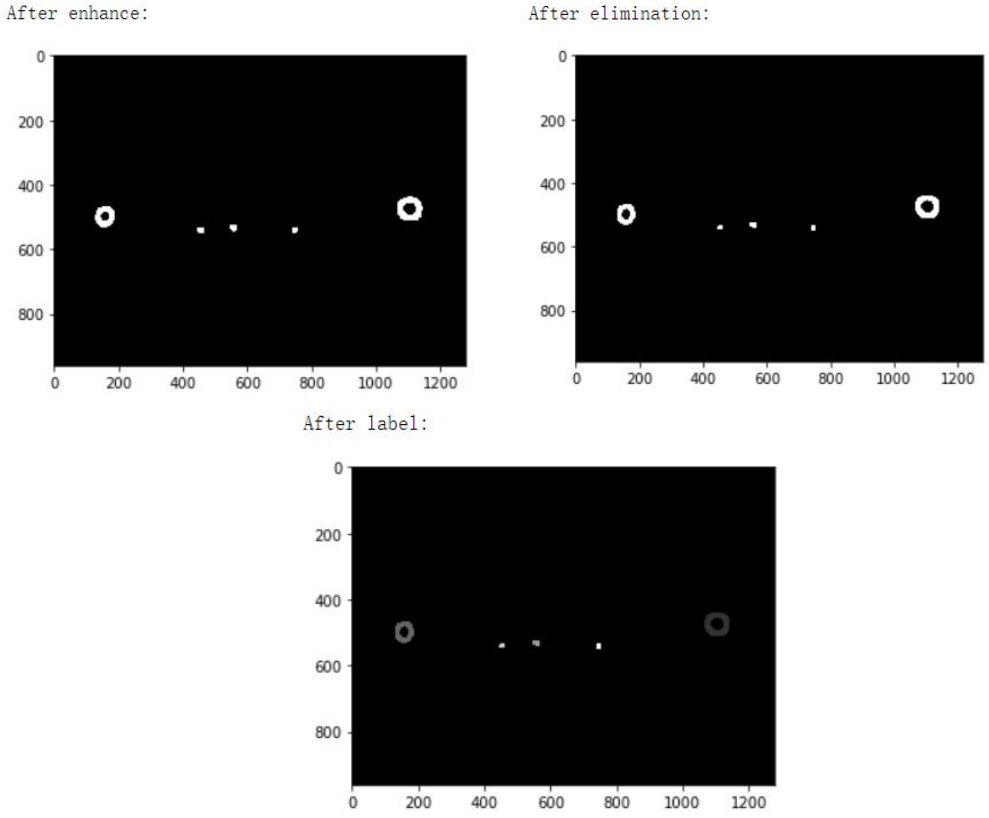


Figure.8: Enhance (Upper Left), Erosion (Upper Right), Label (Down).

Here I must explain what the label does and how it works. It is a very wise operation in the process of getting the final RoIs. label assigns different values to all the non-zero areas in the picture. In this example, as shown in Figure 8 Label (Down), the brightness of each area is not the same because the label assigns a different value to the area where the original value was 1. The higher the value, the brighter it is. Therefore, based on this property, a simple numerical condition is needed to obtain the coordinates of each region and its range in subsequent operations. The implementation is shown in Figures 9, 10.

```
# A list variable
speed_signs = []
# A for loop to go through all labels
for i in range(1, num+1):
    # Use i to filte labels reseparatly
    label_region = (Mask_labeled==i).astype(np.uint8)
    # Return all rowa and colums of nonzero
    row, colum = np.nonzero(label_region)
    # Use try to avoid the ValueError
    try:
        # Calulate the area of ROI
        label_area = ( np.amax(row) - np.amin(row) ) * (np.amax(colum) - np.amin(colum))
        # Use the condition to eliminate the small region
```

Figure.9: Use the Label to get the position of Non-zero region.

By calculating the size of the area of the non-zero region to eliminate those smaller areas that are not likely to be velocity markers. and storing the coordinates in an array. The final output speed image is resized to 64*64. and the final speed image is converted to grayscale for subsequent calculations.

```
        # Calulate the area of ROI
        label_area = ( np.amax(row) - np.amin(row) ) * (np.amax(colum) - np.amin(colum))
        # Use the condition to eliminate the small region
        if label_area < 1000:
            # Assign those small light point into 0.
            Mask_labeled[np.amin(row)-2: np.amax(row)+2, np.amin(colum)-2: np.amax(colum)+2] = 0
            Mask_erosion[np.amin(row)-2: np.amax(row)+2, np.amin(colum)-2: np.amax(colum)+2] = 0
        else:
            # Define variables to store positions of ROI
            Xmax = np.array([])
            Xmin = np.array([])
            Ymax = np.array([])
            Ymin = np.array([])

            # Return the position of Min and Max of X Y axis
            #np.append(Xmax, np.amax(row))
            #np.append(Xmin, np.amin(row))
            #np.append(Ymax, np.amax(colum))
            #np.append(Ymin, np.amin(colum))

            Xmax = np.append(Xmax, np.amax(row))
            Xmin = np.append(Xmin, np.amin(row))
            Ymax = np.append(Ymax, np.amax(colum))
            Ymin = np.append(Ymin, np.amin(colum))
            # Frame the ROI and return it
            speed_sign = images_rgb[np.amin(row): np.amax(row), np.amin(colum): np.amax(colum),:]
            #spped_sign = gray2rgb(speed_sign)

            # Resize speed sign rigion into 64*64
            speed_sign = resize(speed_sign, (64,64,3))
            # more than 1 speed signs, stored use append
            speed_signs.append(speed_sign)
            imutils.imshow(speed_sign)
    except ValueError:
        pass
```

Figure.10: Output positions of all RoIs and display the speed signs.

The final output RGB image with the speed logo is displayed as shown in Figure 11. The area with just the speed logo obtained after eliminating all the small areas is shown in Figure 12.

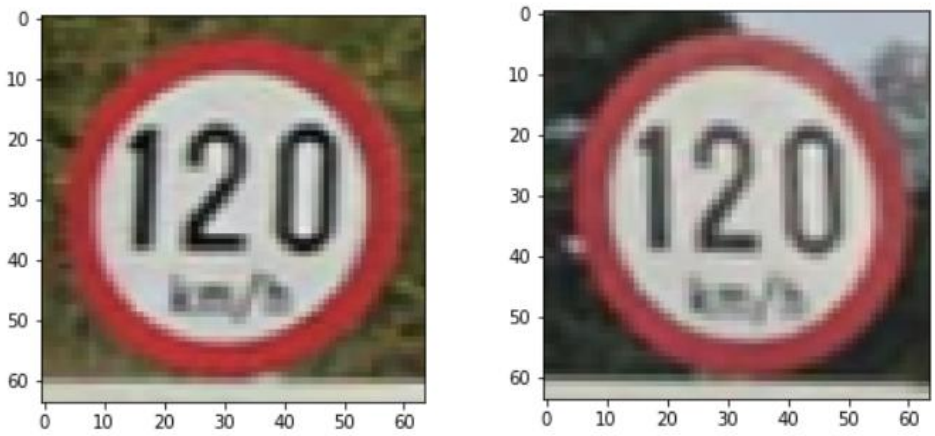


Figure.11: Speed signs in the example image.

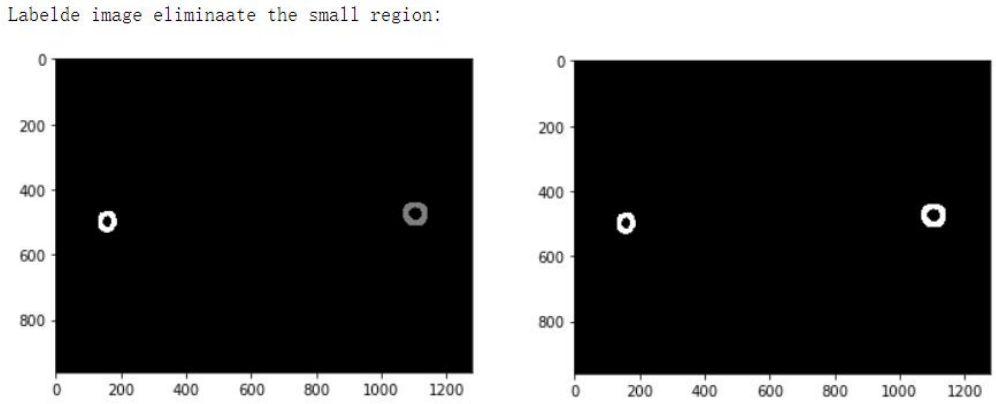


Figure.12: The pure RoIs.

After completing an instance, the process is constructed as a method, which is called to perform a test on all images and return all RoIs. The implementation is shown in Figure 13, and the total number of RoIs tested is 48, as shown in Figure 14. The final output image will be explained again later in the report.

```
speed_imgaes = load_images_from_folder('./speed-sign-test-images/')
speed_images = np.array(speed_imgaes)

all_images_data = np.array([[]])

detectionCount = 0;
#for i in range(0,2):

# Just for test the method
for i in range(0, speed_images.shape[0]):

    # Call method
    speed_signs, Xmax, Xmin, _, _ = speed_sign_detector(speed_images[i,:,:,:])
    speed_signs = np.array(speed_signs)
    print(Xmax, Xmin)
    # Go thought all ROIs
    for j in range(0,speed_signs.shape[0]):
        #speed_signs = rgb2gray(speed_signs[j,:,:])
        detectionCount = detectionCount + 1
        print(speed_signs.shape)
        signs_array_flatten = speed_signs.flatten()
        np.append([all_images_data], [signs_array_flatten])

        print(signs_array_flatten)
        print(all_images_data.shape)
        imutils.imshow(speed_signs[j,:,:])
print("The total detection RoIs: ")
print(detectionCount)
```

Figure.13: Detect RoIs in all images.

```
The total detection RoIs:
48
```

Figure.14: The total number of detection in all 28 images.

3. Speed Sign Classifier

3.1 No Label descript vector generation

Import some useful libraries: numpy use to process array. Imutils use to displat images. Method created in last session.

```
# Team member           ID
# Jufeng Yang           20125011
# Xingda Zhou           19107471
# Zhongen Qin           19107579

# import numpy and imutils
# From Speed sign detector import speed_sign_detector and load_images_from_folder
import numpy as np
import imutils
from Speed_sign_detector import speed_sign_detector
from Speed_sign_detector import load_images_from_folder
```

Figure.15: Import useful libraries.

Reads all images from memory. Based on the conclusions obtained in the previous section, construct an array of zeros with 48 rows and 4097 columns. The first of these columns is used to store the labels. The latter 4096 = 64*64 is used to store the image data. An index variable is designed so that all the RoIs detected can be stored in the array.

```
# Load images from memory
speed_imgaes = load_images_from_folder('./speed-sign-test-images/')
# Convert list into array
speed_images = np.array(speed_imgaes)

# Create a zero descript vector for assignment later
descript_vector = np.zeros((48, 4097))

# Create index variable to show how many images detected
index = 0

```

Figure.16: Create a descript vector array and index variable.

Using a for loop, iterate through each image. Call the method written in the previous section and return all the detected areas. Convert the returned region images into an array. Since each image is detected in more than one region, another loop needs to be nested to iterate through all the regions. In the subloop, flatten each region into a one-dimensional array and assign the new array of regions to the description vector using the value of index. All detected regions and the flattened array data are output, along with an indication that the current region is the current region index in the picture index. The output is shown in Figure 18 and the code implementation is shown in Figure 17.

```python
# Use for loop to go through all images
for i in range(0, speed_images.shape[0]):
    # Detect speed signs from a image, return a speed sign image
    speed_signs, _, _, _, _ = speed_sign_detector(speed_images[i,:,:,:])
    # Convert speed_sign
    speed_signs = np.array(speed_signs)

    # Use for loop to go through all labels(RoI) in a image
    for j in range(0,speed_signs.shape[0]):
        # Check the shape of speed sign
        print(speed_signs.shape)

        # Flatten all image grid data into a 1-D data
        signs_array_flatten = speed_signs[j,:,:].flatten()
        # assign all picture descript vector to a array
        descript_vector[index, 1:4097] = signs_array_flatten
        # Incease index for descript vector
        index = index+1
        print("-----------------------------------------------------")
        # Use to check the number of images and RoIs
        print("%d : %d"%(i+1,j+1))
        # Show the sign images.
        imutils.imshow(speed_signs[j,:,:])
        #print(descript_vector)

        # Print the index
        print("First %d row"%index)
        print(descript_vector[index-1,:])

# Check one row of descript vector
print(descript_vector[40,:])
```

Figure.17: Convert the RoIs into descript vectors.

The images output:

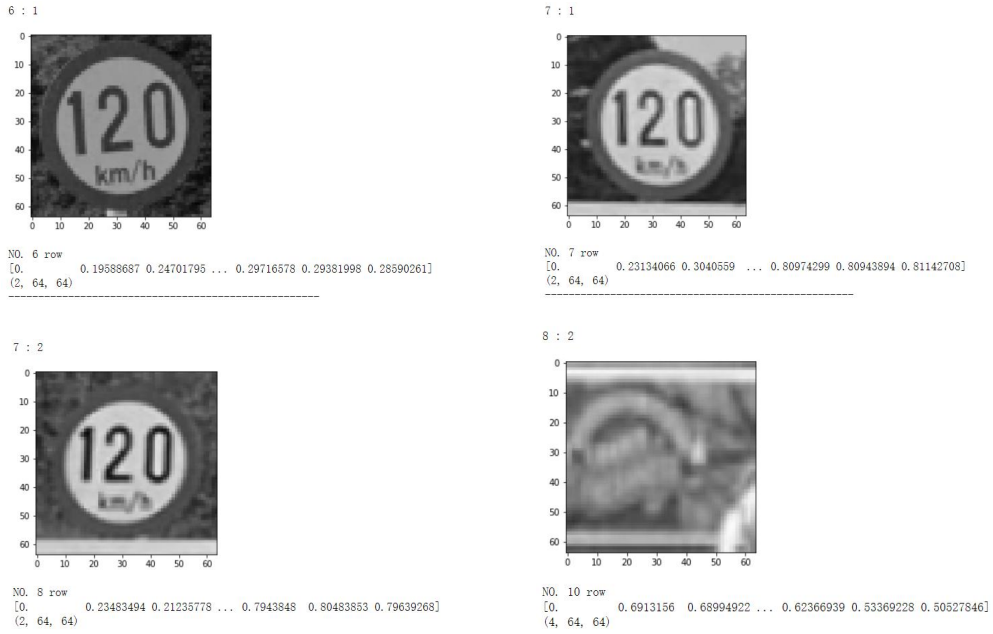


Figure.18: The output of each RoIs and corresponding descript vector.

Check any row in the descript vector array:

```
Check any row of the descript vector:
[0.          0.35989967 0.35701937 ... 0.28119157 0.26557884 0.25822264]
```

Figure.19: Check the any row of the descript vector.

3.2 Add label to descript vectors

The output in the previous block of code helps us to simply add labels for each of the areas. The labels are added to the description vector according to the order of the output in the image. The code implementation is shown in Figure 20.
In all the photos. I set the label values to 120 = 0.12, 80 = 0.08, 100 = 0.1, 40 = 0.04. Then in the final output, the label values are multiplied by 1000 to give the speed values.

```
# Confirm all labels of pictures
# 100 replace by 0.1, 120 replace by 0.12, 80 replace by 0.08, 60 replace by 0.06, 50 replace by 0.05, 40 replace by 0.04
# Non speed sign replace by -1
labels = [0.1, 0.1, 0.1, 0.1, 0.1, 0.12, 0.12, 0.12, 0.12, -1,
          -1, 0.12, -1, -1, 0.12, -1, -1, -1, 0.04, -1,
          0.04, -1, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.06, 0.06,
          0.06, 0.06, 0.06, 0.06, 0.08, -1, -1, -1, -1, -1,
          0.08, 0.08, 0.08, 0.08, 0.08, 0.08, -1, -1]
# Assign all labels to descript vectors
descript_vector[0:48, 0] =  np.array(labels)

# Check one row of descript vector
print("The NO.6 row description vector: ")
print(descript_vector[6, :])

print("\nThe shape of descript vector: ")
np.resize(descript_vector, (48, 4097))
print(descript_vector.shape)
#np.save('descript_vector', descript_vector)
```

Figrue.20: Add label to each descript.

On the first run, the final description vector generated will be saved locally. However, when it needs to be called later, it can be accessed directly from memory.
Randomly check any line of the description vector:

```
The NO.6 row description vector:
[0.12       0.23134066 0.3040559  ... 0.80974299 0.80943894 0.81142708]

The shape of descript vector:
(48, 4097)
```

Figure.21: Check any row of descript array.

3.3 RoIs Classifier

Once the description vector has been constructed, it is time to use the 1-NN algorithm model to classify the input images. At this point, the function in Part 1 will be called again to return all the RoIs of the input image, by calculating the distances between the RoIs in the input image and all the entries in the description vector. The label of the entry with the smallest distance is returned. This label is the final output. The exact implementation will be given later.

As shown in Figure 22, the method is called to read the local image and convert it to an array. Read the local description vector. A two-dimensional array is created with the number of rows as the number of entries in the description vector and the number of columns as 2. The first column stores the index number and the second column stores the distance value.
Select one image from all the images to be detected. Call the detection function, return the RoIs of the current image and convert it to an array.

```python
# Just copy the code before and create a method
def speed_sign_classifer(test_images):

    # load descript vector
    descript_vectors = np.load('descript_vector.npy')
    # Convert all image data into array
    test_images = np.array(test_images)

    # Variable to store index and distance value
    all_distance = np.zeros((descript_vectors.shape[0], 2))

    # Call the sign detector method to return the signs imagesand the position of images.
    test_signs, Xmax, Xmin, Ymax, Ymin = speed_sign_detector(test_images)
    # Convert all data to array
    test_signs = np.array(test_signs)

```

Figure.22: Test image input.

Set one variable to store the number of speed markers and four maths to store the coordinates and range of the speed markers. During the processing of the data there will be many RoIs that are not speed markers. For these data I set the label value to -1. Therefore this coordinate should be discarded when we are but the coordinate value.

Use a for loop to calculate the distance between each RoI and each row of the description vector, return the index value of the smallest distance, and use the index value to get the labels in the description vector.

```python
    # Define a variable to count number of signs
    signs_count = 0
    # Define 4 variables and use it store the position of speed signs
    Xmax_sign = np.array([])
    Xmin_sign = np.array([])
    Ymax_sign = np.array([])
    Ymin_sign = np.array([])

    # Use a for loop to go through all RoI
    for j in range(0, test_signs.shape[0]):
        # Flatten the image data in to 1-D
        test_signs_array_flatten = test_signs[j, :, :].flatten()
        # Use for loop to calculate distances with each row
        for ind in range(0, 48):
            # Substraction operation for the teat value and descript vector
            substraction = np.subtract(test_signs_array_flatten, descript_vectors[ind, 1:4097])
            # sqrt and times operation to calcualte the distance
            distance = np.sqrt((np.dot(substraction, substraction)))
            # Assign all distance value and index to a array
            all_distance[ind, :] = np.array([ind, distance])
            # Use the argmin to return the index of minmum distance
            speed_number_index = np.argmin(all_distance[:, 1])
            #print(speed_number_index)
```

Figure.23: Calculate the distance between RoIs and descript vectors.

When the current predicted label is not -1, the position information of the current RoIs is saved. The tag value of the current RoI is returned and the tag counter is added by 1. The position of the valid speed marker is finally returned.

```
        # if the returned value is not -1, which means the label is a speed sign
        if descript_vectors[speed_number_index,0] != -1:
            # sign count to record number of speed signs
            signs_count = signs_count + 1
            #print(descript_vectors[speed_number_index,0])

            # Select the position info from all ROI positions
            Xmax_sign = np.append(Xmax_sign, Xmax[j])
            Xmin_sign = np.append(Xmin_sign, Xmin[j])
            Ymax_sign = np.append(Ymax_sign, Ymax[j])
            Ymin_sign = np.append(Ymin_sign, Ymin[j])

            # Return the speed sign value from descript vector corresponding to the index
            returned_label = descript_vectors[speed_number_index,0]
            # Use the label * 1000 to generate the speed value
            returned_label = returned_label*1000
            # Output the sign info
            print("sign %d, %d km/h"%(signs_count, returned_label))
    # Print the total sign number
    print("There is(are) %d speed sign(s)"%(signs_count))
    return returned_label, Xmax_sign, Xmin_sign, Ymax_sign, Ymin_sign
```

Figure.24: Judge if the RoI is speed sign.

4. Final Speed Sign Detector

Complete the image reading method, the RoIs detection method, and the RoIs classifier method. These methods will then be called to frame the velocity representation in the original graph and indicate the velocity value in the graph, using the final parameters returned by the classifier.

Import useful libraries and method created before:

```
# Team member          ID
# Jufeng Yang          20125011
# Xingda Zhou          19107471
# Zhongen Qin          19107579

# From Speed sign detector import speed_sign_detector method and load_images_from_folder method
# From Speed_sign_detector import speed sign classfier.
# From matplotlib import patches pyplot import Rectangle and let pyplot as plt
import numpy as np
from Speed_sign_detector import speed_sign_detector
from Speed_sign_detector import load_images_from_folder
from Speed_sign_classifer import speed_sign_classifer
from matplotlib.patches import Rectangle
import matplotlib.pyplot as plt
```

Figure.25: Useful libraries in final speed sign detector.

Read all the images in memory, take one of them and input it into the speed sign detector function, which will return all the RoIs in the image. input the array containing all the RoIs into the speed sign classifier and get the speed value result and the range of valid speed signs. Use the coordinates to frame the speed markers in the graph and write the speed values. The implementation is shown in the figure below.

```
# Load all pictures from memory
speed_images = load_images_from_folder('./speed-sign-test-images/')
# Convert list data into array
speed_image = np.array(speed_images)
# Choose a image from all labels
speed_image = speed_image[7,:,:,:]
#print(speed_image.shape)

#all
detector_sign,_,_,_,_  = speed_sign_detector(speed_image)

# Call classifier method to output the speed sign value and the position and range of signs
speed_value,Xmax, Xmin, Ymax, Ymin = speed_sign_classifer(speed_image)

# Print the value of X Y axis max and min positions
#print(Xmax,Xmin,Ymax,Ymin)
#print(Xmax.shape)

# Create a figure to show the rectangle of the signs
fig, ax = plt.subplots()
# Show original image
ax.imshow(speed_image)

# Frame all signs out and print the speed value
for i in range(0, Xmax.shape[0]):
    # plot a green rectangle
    ax.add_patch( Rectangle((Ymin[i], Xmin[i]),Xmax[i]-Xmin[i], Ymax[i]-Ymin[i],fc ='none', ec ='r',lw = 1))
    # print the speed value in the image
    ax.text(Ymin[i], Xmin[i], '%d km/h'%(speed_value), style='italic',
        bbox={'facecolor': 'green', 'alpha': 0.5, 'pad':1.5})
plt.show()
```

Figure.26: The implementation of final speed sign detector.

The final output is shown in the diagram below. Here I only show the detection results for 11 images:

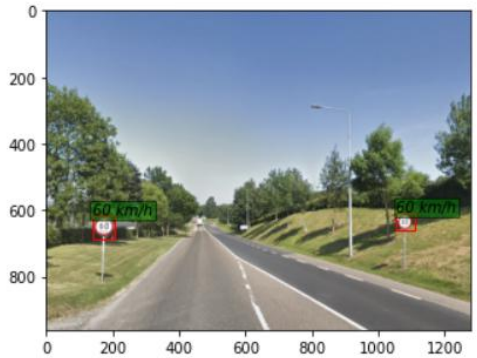
sign 1, 120 km/h
There is(are) 1 speed sign(s)

sign 1, 120 km/h
sign 2, 120 km/h
There is(are) 2 speed sign(s)

sign 1, 100 km/h
There is(are) 1 speed sign(s)

sign 1, 100 km/h
There is(are) 1 speed sign(s)

sign 1, 60 km/h
sign 2, 60 km/h
There is(are) 2 speed sign(s)



sign 1, 50 km/h
sign 2, 50 km/h
There is(are) 2 speed sign(s)

```
sign 1, 50 km/h
There is(are) 1 speed sign(s)
```


```
sign 1, 80 km/h
sign 2, 80 km/h
There is(are) 2 speed sign(s)
```


```
sign 1, 60 km/h
There is(are) 1 speed sign(s)
```


```
sign 1, 80 km/h
There is(are) 1 speed sign(s)
```


```
There is(are) 0 speed sign(s)
```

5. Conclusion

I have divided the overall code structure of this project into three parts. The first is the detection of RoIs, the second is the code for classifying RoIs, and the third is the display of the final image. During the implementation of this project, there were some crucial processing that determined the success or failure of the whole project.

In the detection function, the thresholds used in the HSV image processing for the three channels had to be very carefully adjusted, the value of the threshold being directly related to whether the speed marker could be detected or not. In addition to the enhancement of the image data after passing Mask, the enhancement of the image makes the values in the speed region more visible, while the Maximun filter does not change the values in very small areas (not visible to the naked eye). When the data is enhanced, the speed marker is not removed by the binary erosion function. This step also reduces the number of regions considerably. The regions are separated using labels and the RoIs are processed separately using the label values, leaving the RoI with the right size.

The most important part of the classification function is the generation of the description vector, as there are multiple speed signs in an image so all RoIs need to be turned into description vectors. In addition to this, it is more important to discard the -1 labels when obtaining the label values. The location of the correct speed marker is saved.

Take the key data obtained in step 2, draw it in the plot in step 3 and use matplotlib to frame the appropriate area on the original plot and indicate the speed values.

At this point, all the steps have been completed and in the final test, all the images were classified correctly. However, there is still a small problem with the code. The code still returns the correct result when there is no speed marker in an image, but at the end of the run, the code reports an error.

6. Reference

[1] C. Flanagan. 02-project2-speed-signs.4up [PDF]. Available: https://sulis.ul.ie/access/content/group/82edf4ed-f03d-4ab6-bf99-2773fa7cd801/Test%20Images/Project%20_2_%3A%20Speed%20Sign%20Recognition/02-project2-speed-signs.4up.pdf