FINAL PROJECT

COM421_ IMAGE PROCESSING

PROJECT TITLE: TRAFFIC SIGN READER

ADVISOR: Dr. Öğr. Üyesi PERİ GÜNEŞ

GROUP MEMBERS:

SÜMEYYE TAŞTAN – B1605.010004

ŞEVVAL ÖZTÜRK – B1605.010008

TABLE OF CONTENTS

1. INTRODUCTION

In our project, we identify the traffic signs on the picture and then compare these signs with the pictures in the data set we have created, and make the traffic sign say aloud. We use color values to perceive the plate through the picture. Then we determine the geometric shape on the resulting picture and cut the point where this geometric shape is located from the original picture. Thus, we remove only the part where the plate is located from the picture and subject it to a similarity test with our data. Whichever image in the image looks more like a traffic sign, we read the meaning of that image in the data set aloud. So we can make the signs speak their meaning out loud.

1.1.Literature review

The real life usage rate of traffic sign reading and detection projects is minimal. The reason for this is that traffic sign detection studies can reach a new level of accuracy that can be used in real life. Over the years, we have examined what kind of work has been done to read and understand traffic signs.

It was first published in 1997 by A. de la Escalera, L.E Moreno; M.A. Salichs; J.M. They conducted a study by Armingol people on Road traffic sign and detection and classification. In this study, it has been tried to be developed by applying reverse engineering to the design method of traffic signs. For this reason, the algorithms used in the study are based on color and shape detection. [1]

Later, in 2009, with the study conducted by the graduate student of MIDDLE EAST TECHNICAL UNIVERSITY, it was aimed to recognize the traffic signs, to be more careful of the drivers and to make a prior notification against the incoming warnings. Already existing traffic signs are available in certain colors and shapes to attract drivers' attention, the main purpose of this project is to process them. Here, all kinds of factors have been taken into consideration while performing these operations. As a result, traffic signs recognition system will be developed. The algorithm used consists of 3 stages. The point of attention here is to make use of the illumination of the surroundings and therefore first distinguishes according to the colors of the traffic signs. In the second stage, it re-creates the shape of the traffic signs from the remaining parts and defines them. [2]

Then in 2010 by Hacettepe University graduate students, a study was made in order to detect automatically mark the traffic in Turkey. In order to identify the traffic signs automatically, RGB color space and various image processing techniques were used for the separation of colors in this study. 46 pictures have been added to the database for the identification of traffic signs. Image recognition processing was tried to be performed by comparing the images that emerged after image processing techniques with the data in the database. A success rate of 94% was achieved in this study. [3]

In the following years, Tuğhan Çağlayan, Habibullah Ahmadzay and Gökhan Kofraz aimed to recognize the determined traffic signs by placing a camera in the front of the cars in their work on “Real-time traffic sign recognition based on shape and color classification”. This is a study by determining the shapes and colors of certain traffic signs. Thanks to the camera in front of the vehicle, they made it possible to understand the red color of the traffic signs, whether they are circular or triangular. Data is collected to recognize these traffic signs. After collecting this data, they used algorithms such as Support Vector Machine to detect the newly displayed image by comparing it with the data. [4]

Studies on this subject continued in the following years. In 2016, a study was carried out by Atatürk University graduate students to automatically detect traffic signs. In this study, BTIT (Belgian Traffic Sign Recognition) data set was used. In order to achieve the most successful result in the project, the use of DVM (Attributes and Support Vector Machines Classifier), KNN (Nearest Neighbor), KA (Decision Trees), TTT (Bagging Based Community) algorithms and HOG (Histogram of Directional Slopes), SIFT (Scaling Independent Attribute Transformation), SURF (Accelerated Robust Attribute) functions are used. As a result of these trials, the most successful result was obtained by using TTT algorithm and HOG and SIFT functions for feature extraction in the project. The success rate of this project is 91%. [5]

Another study (2017) on this subject is the study on "TRAFFIC SIGN RECOGNITION BASED ON ARTIFICIAL NEURAL NETWORK" conducted by NEAR EAST university graduate students. Data learning is at the top of the steps applied in the project for traffic sign detection. For data learning, 13 different traffic signs and 16 different types of noise added or masked pictures of these traffic signs were used in the project. While performing the application in this project, first the classification process was made and then the determination was made. First, some filters were used in the project to classify the images. These are Wiener filter, Median filter and Threshold. The Wiener filter is a filter that provides estimation by controlling the signal given in a certain range with its noisy values. The Median filter is used to remove the noise in the picture by converting the index values of the noisy images to an average value. The thresholding function is used in this project in order to easily detect the picture with artificial intelligence technologies in image processing operations. In the next stage of this project, the picture classification process was made. They did the picture segmentation in 2 different ways. The first picture segmentation method used in the project is to separate the pictures according to their pixels and the second is to separate the pictures according to their colors. After the completion of these processes, the picture classification part of the project is finished and after this stage, the determination process has started. ANN (Artificial neural networks) and BPANN (Back propagation artificial neural network) algorithms were used in this project for the detection of traffic signs. At the end of this project, 93 success rates were achieved. [6]

Later, a study of detecting and reading traffic signs was carried out by Fırat University master students in 2017. They used matlab while doing this project. They developed their projects by following these steps. First, they turned the picture read to gray and then analyzed the pictures with the blob algorithm based on their color and shape. The success rate in this project is lower than other projects, but it can be improved. [7]

Finally, in the study conducted by Mert ÇETİNKAYA, Tankut ACARMAN in 2020, it was aimed to detect traffic signs by using vehicle cameras. Deep learning was used in this study because of the statistical approaches. The first step of this algorithm is the determination of the points where the plate can be found. By removing the areas without plates from the detected regions, the data to be controlled has decreased. Finally, the remaining regions were scanned using CNN, and the region with the highest value was accepted as the region with the traffic sign. This study made more successful determinations compared to other studies, and it was found to be lower in terms of cost. [8]

2.SYSTEM DESIGN

2.1 Design Method and Standards

The steps of the algorithm we use to detect and make sense of traffic signs in our project are as stated in Figure 1 above.

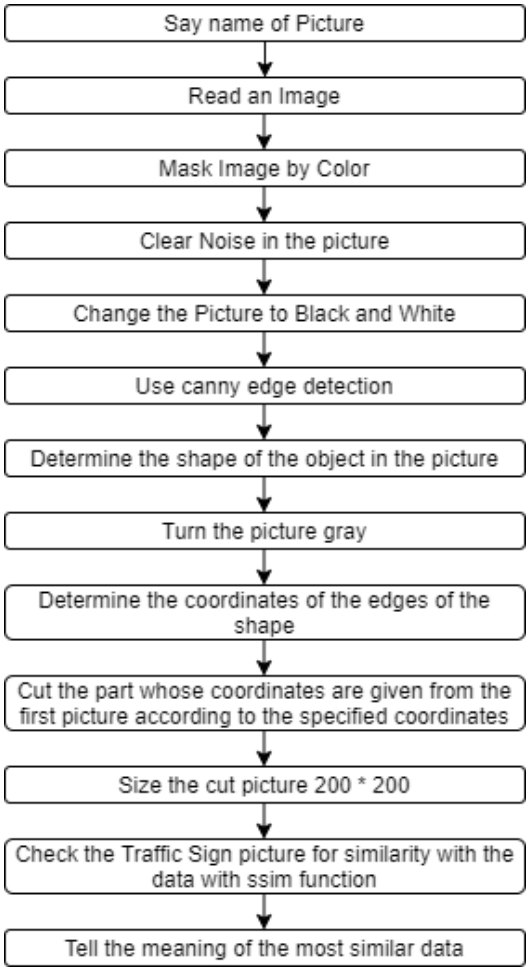

Figure 1 System Design Diagram

While the program is running, it first asks the user which picture they want to choose. Then the program reads the picture and prints on the screen. After this step, a filter that masks the red color is applied to the picture in order to detect objects that may be traffic signs. Then the noise removal function is used to clean the other objects in the picture. In the last state of the picture, all the indexes of the picture were visited one by one and the picture was completely converted to black and white colors by looking at the threshold value of the 2nd layer. The edges were made clear by using canny edge detection on the black and white picture. Then, the geometric shape of the object in the picture was determined. The image has been grayed out with rgb2gray. The edges and coordinates of the object in the picture are determined. After the filtering and masking etc. processes applied to the picture, the resulting shape and the edge coordinates obtained from it are used on the original picture that we read first, and that region is cut out from the original picture according to those coordinates. In other words, only the part that contains the traffic signs on the whole picture is cut and resized to 200 * 200 to make the cut picture the same size as the data. We compare the dimensioned picture one by one with the data we have previously created with the Structural Similarity function. We have the voice system tell the meaning of the data with the highest similarity rate and print it on the screen at the same time.

2.2 Software development tools

We used the Python language and OpenCv, speechrecognation, pyttsx3, numpy, pandas, math libraries in python to make this system. We used the Atom editor while writing the program.

3. Results

3.1 Syntax of Algorithm

First of all, when our program starts, it asks you which picture you would like to choose. The user tells the program the number of the picture. You can see it in the code below.

```
#Work done for voiceover
engine = pyttsx3.init()
hiz = engine.getProperty('rate')   # getting details of current speaking rate
print (hiz)                        #printing current voice rate
engine.setProperty('rate', 115)
engine.say("Please tell the number of the traffic sign you want to know the meaning of.")

engine.runAndWait()
# Record Audio
r = sr.Recognizer()
with sr.Microphone() as source:
    print("Say something!")
    audio = r.listen(source)

# Speech recognition using Google Speech Recognition
```

Figure 2 Code Part - 1

Then it reads the picture according to the number you said and displays it. You can see the code section below.

```
klasor="Resimler/"
numara=r.recognize_google(audio)
uzanti=".jpg"
yol=klasor+numara+uzanti
img = cv2.imread(yol)
cv2.imshow("Original Image",img)
cv2.waitKey()
```

Figure 3 Code Part - 2

Then you can see the code part of the part that we mask according to the picture color.

```
#for the mask of the red-scala color
boundaries = [ ([0, 0, 0] ,[120, 70, 250]) ] # define the list of boundaries

for (lower, upper) in boundaries:
    # create Numpy arrays from the boundaries
    lower = np.array(lower, dtype = "uint8")
    upper = np.array(upper, dtype = "uint8")
    # find the colors within the specified boundaries and apply
    mask = cv2.inRange(img, lower, upper)
    output = cv2.bitwise_and(img, img, mask = mask)  # the mask

    cv2.imshow("images", output)
    cv2.waitKey(0)
```

Figure 4 Code Part - 3

Then you can see the code part of the part where we reduced the noise of the picture below.

```
#The method we use to reduce noise
output =cv2.fastNlMeansDenoising(output,None,30.0, 7, 21)
cv2.imshow("Noise Reduced Image",output)
cv2.waitKey(0)
```

Figure 5 Code Part - 4

Then, you can see the code of the part where we convert the picture to black and white by using 70 value according to the second layer of the picture.

```
#Start of the function that converts the picture to black and white in order to de
en,boy,katman = np.shape(output) #Take an image size
yeniResim = np.ones((en,boy,katman)) #we create a white visual.
for i in range(en):
    for j in range(boy):
        if(output[i,j,2] >70): #We compare the second layer of the picture whether
            yeniResim[i,j] = 0 # If it is less than 70, paint the louse black

        else:
            yeniResim[i,j]= 1 # If it is less than 70, paint the louse white

im_floodfill_inv=yeniResim
cv2.imshow("Black and White Image",im_floodfill_inv)
cv2.waitKey(0)
```

Figure 6 Code Part - 5

Then we used the canny edge detection function to better detect the edges of the object in the picture. You can see the code for this stage below.

```
#We use canny edge method to define edges
img_canny = cv2.Canny(imgshape, 200, 400)
cv2.imshow("Image-Canny" , img_canny)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Figure 7 Code Part - 6

Then, it detects which geometric shape the object is and draws a blue line around it accordingly. We used 0.03 to determine the triangle corner points and 0.001 for circle. You can see the code for this stage below.

```
#The function we use to determine the geometric shape of the object in the picture
def get_contours(imgshape, img_contour):
  contours, hierarchy = cv2.findContours(imgshape, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
  for cnt in contours:
    area = cv2.contourArea(cnt)
    if area > 9000:
      cv2.drawContours(img_contour, cnt, -1, (255, 0, 255), 1)
      # Find length of contours
      param = cv2.arcLength(cnt, True)
      # Approximate what type of shape this is
      approx = cv2.approxPolyDP(cnt, 0.01 * param, True)
      shape, x, y, w, h = find_shape(approx)
      cv2.putText(img_contour, shape, (x+78, y+200), cv2.FONT_HERSHEY_COMPLEX, .7, (255, 0, 255), 1)
  return approx, param, img_contour, contours, cnt
#The function that prints the number of sides of the figure in the picture
def find_shape(approx):
  x, y, w, h = cv2.boundingRect(approx)
  print("Number of edge ",len(approx))
```

Figure 8 Code Part - 7

```
accuracy=0.03*cv2.arcLength(c,True)
approx=cv2.approxPolyDP(c,accuracy,True)
```

Figure 9 Code Part - 8

```
accuracy=0.001*cv2.arcLength(c,True)
approx=cv2.approxPolyDP(c,accuracy,True)
```

Figure 10 Code Part - 9

Then, we used the corner coordinates of the object, whose corners we determined, in the original painting, and we performed the masking process. You can see the code for this stage below.

```
mask = np.zeros(mediandilate.shape[:2], dtype=np.uint8)
cv2.drawContours(mask, [approx], -1, (255, 255, 255), -1, cv2.LINE_AA)

res = cv2.bitwise_and(img,img,mask = mask)
rect = cv2.boundingRect(approx) # returns (x,y,w,h) of the rect
```

Figure 11 Code Part - 10

After the masking process, we cut and resize the sheet whose coordinates are given from the original picture. You can see the code for this stage below.

```
cropped = res[rect[1]: rect[1] + rect[3], rect[0]: rect[0] + rect[2]]
dimensions = cropped.shape
height = cropped.shape[0]
width = cropped.shape[1]
channels = cropped.shape[2]

#We made it to be able to see the size, height, width and depth of the picture.
print('Image Dimension    : ',dimensions)
print('Image Height       : ',height)
print('Image Width        : ',width)
print('Number of Channels : ',channels)

#We resized our picture to 200x200.
dsize = (200, 200)
cropped = cv2.resize(cropped, dsize)
```

Figure 12 Code Part - 11

Then, we compare the cut picture one by one with the 8 data sets that we have prepared, and the similarity ratio is calculated. You can see the code for this stage below.

```
#Function to compare pictures
def mse(imageA, imageB):
    err = np.sum((imageA.astype("float") - imageB.astype("float")) ** 2)
    err /= float(imageA.shape[0] * imageA.shape[1])

    return err
#Function to find image similarity
def compare_images(imageA, imageB, title):
    a = mse(imageA, imageB)
    #find image similarity
    b = ssim(imageA, imageB)
    #It displays the original picture and the picture contained in the data on the screen.
    fig = plt.figure(title)
    ax = fig.add_subplot(1, 2, 1)
    plt.imshow(imageA, cmap = plt.cm.gray)
    plt.axis("off")
    ax = fig.add_subplot(1, 2, 2)
    plt.imshow(imageB, cmap = plt.cm.gray)
    plt.axis("off")
    plt.show()

    #The function returns the image similarity ratio.
    return b
```

Figure 13 Code Part - 12

```
        #Similarity Ratio function
        benzerlikorani = compare_images(original, data, "Original vs. Data")
        print("Similarity Ratio", benzerlikorani)

        #Function to find which images in the data are most similar
        if benzerlikorani > temp:
            temp=benzerlikorani
            tempname=(data_num)-(1) #Bringing the meaning of the picture in the Array

    print("The most similar value",temp)
    print("Meaning of the most similar data picture :",tempname)
    print("Meaning of the traffic sign ",nameoftraficsign[tempname])
```

Figure 14 Code Part - 13

After the similarity comparison, the data with the highest similarity rate is selected and its meaning is taken from the array according to its number, and it is both spoken and printed in the upper left corner from the original picture. You can see the code for this stage below.

```
#The array that holds the names of the data
nameoftraficsign=["Attention Crosswalk ","Attention Road Works ","Attention uneven road","Attention No Left Turn ",
"Attention left hand curve ","Attention speed limit 70","Attention no u-turn ","Attention steep hill downwards"]
```

Figure 15 Code Part - 14

```
        #Similarity Ratio function
        benzerlikorani = compare_images(original, data, "Original vs. Data")
        print("Similarity Ratio", benzerlikorani)

        #Function to find which images in the data are most similar
        if benzerlikorani > temp:
            temp=benzerlikorani
            tempname=(data_num)-(1) #Bringing the meaning of the picture in the Array

    print("The most similar value",temp)
    print("Meaning of the most similar data picture :",tempname)
    print("Meaning of the traffic sign ",nameoftraficsign[tempname])

    #Speech Recognition voice rate adjustment
    rate = engine.getProperty('rate')
    engine.setProperty('rate', 115)
    #Speaks the meaning of the traffic sign
    engine.say(nameoftraficsign[tempname])
    engine.runAndWait()
    #Writes the meaning of the traffic sign on the original picture
    cv2.putText(detectOrgImg, nameoftraficsign[tempname],(10, 40), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1)
    cv2.imshow("Result",detectOrgImg)
    cv2.waitKey()
```

Figure 16 Code Part - 15

All source code like this :

```
import cv2
import numpy as np
import math
import pandas as pd
import matplotlib.pyplot as plt
import pyttsx3
import os.path
import speech_recognition as sr

from scipy.stats import itemfreq
from skimage.draw import ellipse
from skimage.measure import label, regionprops, regionprops_table
from skimage.transform import rotate
from skimage.metrics import structural_similarity as ssim
engine = pyttsx3.init()

#Function to compare pictures
def mse(imageA, imageB):
    err = np.sum((imageA.astype("float") - imageB.astype("float")) ** 2)
    err /= float(imageA.shape[0] * imageA.shape[1])

    return err
#Function to find image similarity
def compare_images(imageA, imageB, title):
    a = mse(imageA, imageB)
    #find image similarity
    b = ssim(imageA, imageB)
    #It displays the original picture and the picture contained in the data on the screen.
    fig = plt.figure(title)
    ax = fig.add_subplot(1, 2, 1)
    plt.imshow(imageA, cmap = plt.cm.gray)
    plt.axis("off")
    ax = fig.add_subplot(1, 2, 2)
    plt.imshow(imageB, cmap = plt.cm.gray)
    plt.axis("off")
    plt.show()

    #The function returns the image similarity ratio.
    return b

kernel = np.ones((5,5),np.uint8) # An array is specified for the kernel.
kernel1 = np.array([[1, 1, 1, 1, 1], #same array is determined like this
                    [1, 1, 1, 1, 1],
                    [1, 1, 1, 1, 1],
                    [1, 1, 1, 1, 1],
                    [1, 1, 1, 1, 1]])

kernel1 = kernel1/sum(kernel1)

#work done for voiceover
engine = pyttsx3.init()
hiz = engine.getProperty('rate')   # getting details of current speaking rate
print (hiz)                        #printing current voice rate
engine.setProperty('rate', 115)
engine.say("Please tell the number of the traffic sign you want to know the meaning of.")

engine.runAndWait()
# Record Audio
r = sr.Recognizer()
with sr.Microphone() as source:
    print("Say something!")
    audio = r.listen(source)

# Speech recognition using Google Speech Recognition
try:

    print("You said: " + r.recognize_google(audio))
except sr.UnknownValueError:
    print("Google Speech Recognition could not understand audio")
except sr.RequestError as e:
    print("Could not request results from Google Speech Recognition service; {0}".format(e))

#The work done to find the path to the picture from the folder
klasor="Resimler/"
numara=r.recognize_google(audio)
uzanti=".jpg"
yol=klasor+numara+uzanti
img = cv2.imread(yol)
cv2.imshow("Original Image",img)
cv2.waitKey()

#for the mask of the red-scala color
boundaries = [ ([0, 0, 0] ,[120, 70, 250]) ] # define the list of boundaries
```

```
84
85  for (lower, upper) in boundaries:
86      # create Numpy arrays from the boundaries
87      lower = np.array(lower, dtype = "uint8")
88      upper = np.array(upper, dtype = "uint8")
89      # find the colors within the specified boundaries and apply
90      mask = cv2.inRange(img, lower, upper)
91      output = cv2.bitwise_and(img, img, mask = mask)  # the mask
92
93      cv2.imshow("images", output)
94      cv2.waitKey(0)
95  #The method we use to reduce noise
96  output =cv2.fastNlMeansDenoising(output,None,30.0, 7, 21)
97  cv2.imshow("Noise Reduced Image",output)
98  cv2.waitKey(0)
99
100 #Start of the function that converts the picture to black and white in order to detect the traffic sign
101 en,boy,katman = np.shape(output) #Take an image size
102 yeniResim = np.ones((en,boy,katman)) #we create a white visual.
103 for i in range(en):
104     for j in range(boy):
105         if(output[i,j,2] >70): #We compare the second layer of the picture whether it is greater or less than the threshold value of 70.
106             yeniResim[i,j] = 0 # If it is less than 70, paint the louse black
107
108         else:
109             yeniResim[i,j]= 1 # If it is less than 70, paint the louse wihte
110
111 im_floodfill_inv=yeniResim
112 cv2.imshow("Black and White Image",im_floodfill_inv)
113 cv2.waitKey(0)
114
115 #The operations we do to determine the data type of the image
116 J = im_floodfill_inv*255
117
118 print(im_floodfill_inv.dtype)
119 mediandilate = J.astype(np.uint8)
120 print(mediandilate.dtype)
121
122 imgshape=mediandilate
123 img_contour = imgshape.copy()
124 #we use canny edge method to define edges
125 img_canny = cv2.Canny(imgshape, 200, 400)
126 cv2.imshow("Image-Canny" , img_canny)
127 cv2.waitKey(0)
128 cv2.destroyAllWindows()
129
130 kernel = np.ones((5,5),np.uint8)
131 img_dilated =img_canny
132
133 #The function we use to determine the geometric shape of the object in the picture
134 def get_contours(imgshape, img_contour):
135   contours, hierarchy = cv2.findContours(imgshape, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
136
137   for cnt in contours:
138     area = cv2.contourArea(cnt)
139     if area > 9000:
140       cv2.drawContours(img_contour, cnt, -1, (255, 0, 255), 1)
141
142       # Find length of contours
143       param = cv2.arcLength(cnt, True)
144
145       # Approximate what type of shape this is
146       approx = cv2.approxPolyDP(cnt, 0.01 * param, True)
147       shape, x, y, w, h = find_shape(approx)
148       cv2.putText(img_contour, shape, (x+78, y+200), cv2.FONT_HERSHEY_COMPLEX, .7, (255, 0, 255), 1)
149
150   return approx, param, img_contour, contours, cnt
151
152 #The function that prints the number of sides of the figure in the picture
153 def find_shape(approx):
154   x, y, w, h = cv2.boundingRect(approx)
155   print("Number of edge ",len(approx))
156
157   #If the number of sides is between 3 and 6, it's a triangle.
158   if  (len(approx) >= 3 ) and ( len(approx) <= 6 ):
159     print('Triangle')
160     s = "Triangle" #determine the object is triangle
161     orig_image = mediandilate.copy()
162     #Turning the picture gray, we give the threshold value.
163     gray=cv2.cvtColor(mediandilate,cv2.COLOR_BGR2GRAY)
164     ret, thresh=cv2.threshold(gray,127,255,cv2.THRESH_BINARY_INV)
165     contours, hierarchy=cv2.findContours(thresh.copy(),cv2.RETR_LIST,cv2.CHAIN_APPROX_NONE)
166     for c in contours:
167         x,y,w,h=cv2.boundingRect(c)
```

```
        cv2.rectangle(orig_image,(x,y),(x+w,y+h),(0,0,255),2)

    cv2.waitKey(0)
    #calculate accuracy as a percent of contour perimeter
    #Since the picture is a triangle, we determine the sides by multiplying by 0.03.
    accuracy=0.03*cv2.arcLength(c,True)
    approx=cv2.approxPolyDP(c,accuracy,True)
    cv2.drawContours(mediandilate,[approx],0,(255,0,0),10)
    cv2.imshow('Detection Shape', mediandilate)
    cv2.waitKey(0)
    #We mask the picture on orijinal image
    mask = np.zeros(mediandilate.shape[:2], dtype=np.uint8)
    cv2.drawContours(mask, [approx], -1, (255, 255, 255), -1, cv2.LINE_AA)

    res = cv2.bitwise_and(img,img,mask = mask)
    rect = cv2.boundingRect(approx) # returns (x,y,w,h) of the rect
    #We cut the masked part from the original picture.
    cropped = res[rect[1]: rect[1] + rect[3], rect[0]: rect[0] + rect[2]]
    dimensions = cropped.shape
    height = cropped.shape[0]
    width = cropped.shape[1]
    channels = cropped.shape[2]

    #We made it to be able to see the size, height, width and depth of the picture.
    print('Image Dimension    : ',dimensions)
    print('Image Height       : ',height)
    print('Image Width        : ',width)
    print('Number of Channels : ',channels)

    #We resized our picture to 200x200.
    dsize = (200, 200)
    cropped = cv2.resize(cropped, dsize)

    dimensions = cropped.shape
    height = cropped.shape[0]
    width = cropped.shape[1]
    channels = cropped.shape[2]

    print('Image Dimension New    : ',dimensions)
    print('Image Height  New     : ',height)
    print('Image Width   New     : ',width)
    print('Number of Channels  New: ',channels)


    # create the white background of the same size of original image
    wbg = np.ones_like(img, np.uint8)*255
    cv2.bitwise_not(wbg,wbg, mask=mask)
    dst = wbg+res
    detectOrgImg=img.copy()

    cv2.imshow("Mask",mask)
    cv2.imshow("Cropped", cropped )
    cv2.imshow("Samed Size Black Image", res)
    cv2.imshow("Samed Size White Image", dst)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    data_num=1
    temp=0
    tempname=" "
    #The array that holds the names of the data
    nameoftraficsign=["Attention Crosswalk ","Attention Road Works ","Attention uneven road","Attention No Left Turn ","Attention left hand curve ",
    "Attention speed limit 70","Attention no u-turn ","Attention steep hill downwards"]
    #The function we use to get data images  in the folder
    for data_num in range(1,9):
        uzanti=".jpg"
        data=str(data_num)+uzanti
        data='data/'+data
        data=cv2.imread(data)
        dsize = (200, 200)
        data = cv2.resize(data, dsize) #The picture has been resized.
        original=cropped
        images = ("Original", original), ("Other", data)

        data = cv2.cvtColor(data, cv2.COLOR_BGR2GRAY)
        original = cv2.cvtColor(original, cv2.COLOR_BGR2GRAY)

        #Similarity Ratio function
        benzerlikorani = compare_images(original, data, "Original vs. Data")
        print("Similarity Ratio", benzerlikorani)
```

```
            #Function to find which images in the data are most similar
            if benzerlikorani > temp:
                temp=benzerlikorani
                tempname=(data_num)-(1) #Bringing the meaning of the picture in the Array

        print("The most similar value",temp)
        print("Meaning of the most similar data picture :",tempname)
        print("Meaning of the traffic sign ",nameoftraficsign[tempname])

        #Speech Recognition voice rate adjustment
        rate = engine.getProperty('rate')
        engine.setProperty('rate', 115)
        #Speaks the meaning of the traffic sign
        engine.say(nameoftraficsign[tempname])
        engine.runAndWait()
        #Writes the meaning of the traffic sign on the original picture
        cv2.putText(detectOrgImg, nameoftraficsign[tempname],(10, 40), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1)
        cv2.imshow("Result",detectOrgImg)
        cv2.waitKey()

    elif len(approx) == 8:
        s = "Octagon"

    else:
        s = "Circle"
        orig_image = mediandilate.copy()
        #Turning the picture gray, we give the threshold value.
        gray=cv2.cvtColor(mediandilate,cv2.COLOR_BGR2GRAY)
        ret, thresh=cv2.threshold(gray,127,255,cv2.THRESH_BINARY_INV)
        contours, hierarchy=cv2.findContours(thresh.copy(),cv2.RETR_LIST,cv2.CHAIN_APPROX_NONE)
        for c in contours:
            x,y,w,h=cv2.boundingRect(c)
            cv2.rectangle(orig_image,(x,y),(x+w,y+h),(0,0,255),2)

        cv2.waitKey(0)
        #Because it is circle, we make the edge detection rate more precise, in it we multiply it by 0.01
        accuracy=0.001*cv2.arcLength(c,True)
        approx=cv2.approxPolyDP(c,accuracy,True)
        cv2.drawContours(mediandilate,[approx],0,(255,0,0),10)
        cv2.imshow('Detection Shape', mediandilate)
        cv2.waitKey(0)

        mask = np.zeros(mediandilate.shape[:2], dtype=np.uint8)
        cv2.drawContours(mask, [approx], -1, (255, 255, 255), -1, cv2.LINE_AA)

        res = cv2.bitwise_and(img,img,mask = mask)
        rect = cv2.boundingRect(approx) # returns (x,y,w,h) of the rect
        cropped = res[rect[1]: rect[1] + rect[3], rect[0]: rect[0] + rect[2]]
        dimensions = cropped.shape
        height = cropped.shape[0]
        width = cropped.shape[1]
        channels = cropped.shape[2]

        print('Image Dimension    : ',dimensions)
        print('Image Height       : ',height)
        print('Image Width        : ',width)
        print('Number of Channels : ',channels)

        dsize = (200, 200)
        cropped = cv2.resize(cropped, dsize)

        dimensions = cropped.shape
        height = cropped.shape[0]
        width = cropped.shape[1]
        channels = cropped.shape[2]

        print('Image Dimension New    : ',dimensions)
        print('Image Height  New      : ',height)
        print('Image Width   New      : ',width)
        print('Number of Channels  New: ',channels)

        wbg = np.ones_like(img, np.uint8)*255
        cv2.bitwise_not(wbg,wbg, mask=mask)
        dst = wbg+res
        detectOrgImg=img.copy()

        cv2.imshow("Mask",mask)
        cv2.imshow("Cropped", cropped )
        cv2.imshow("Samed Size Black Image", res)
        cv2.imshow("Samed Size White Image", dst)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
```

```
332         data_num=1
333         temp=0
334         tempname=" "
335         nameoftraficsign=["Attention Crosswalk ","Attention Road Works ","Attention uneven road","Attention No Left Turn ","Attention left hand curve ",
336         "Attention speed limit 70","Attention no u-turn ","Attention steep hill downwards"]
337         for data_num in range(1,9):
338             uzanti=".jpg"
339             data=str(data_num)+uzanti
340             data='data/'+data
341             data=cv2.imread(data)
342             dsize = (200, 200)
343             data = cv2.resize(data, dsize)
344             original=cropped
345             images = ("Original", original), ("Other", data)
346 
347             data = cv2.cvtColor(data, cv2.COLOR_BGR2GRAY)
348             original = cv2.cvtColor(original, cv2.COLOR_BGR2GRAY)
349 
350             benzerlikorani = compare_images(original, data, "Original vs. Data")
351             print("benzerlik oranı", benzerlikorani)
352 
353             if benzerlikorani > temp:
354                 temp=benzerlikorani
355                 tempname=(data_num)-(1)
356 
357         print("The most similar value",temp)
358         print("Meaning of the most similar data picture :",tempname)
359         print("Meaning of the traffic sign ",nameoftraficsign[tempname])
360 
361         rate = engine.getProperty('rate')
362         engine.setProperty('rate', 115)
363         engine.say(nameoftraficsign[tempname])
364         engine.runAndWait()
365         cv2.putText(detectOrgImg, nameoftraficsign[tempname],(10, 40), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 1)
366         cv2.imshow("Result",detectOrgImg)
367         cv2.waitKey()
368 
369     get_contours(img_dilated, img_contour)
370     cv2.imshow("Resim-contour" , img_contour)
371     cv2.waitKey(0)
372     cv2.destroyAllWindows()
373 
```

3.2 Output Explanation

When the program starts, the code written in Figure 2 Code Part - 1 runs. The output of this code according to two different phrases is as in Figure 17 Screen Output-1.

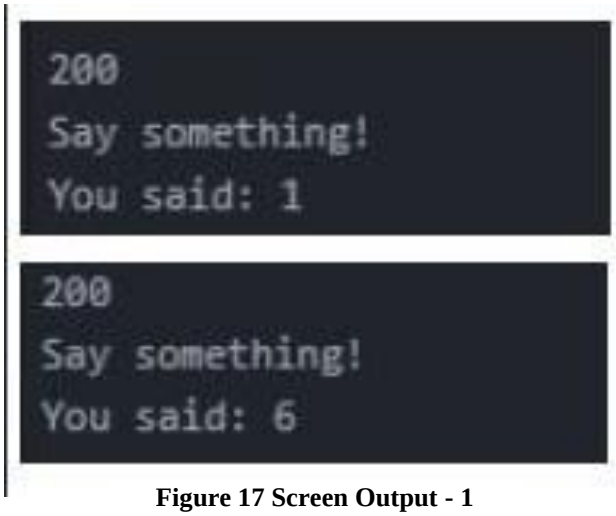

Figure 17 Screen Output - 1

Figure 18 Code Part - 2 The output of the code that gives the path of the code pictures and displays them is as in Figure 18 Screen Output - 2.

Figure 18 Screen Output - 2

The screen output of the code that we mask the pictures in Figure 4 Code Part - 3 is as in Figure 19 Screen Output - 3.

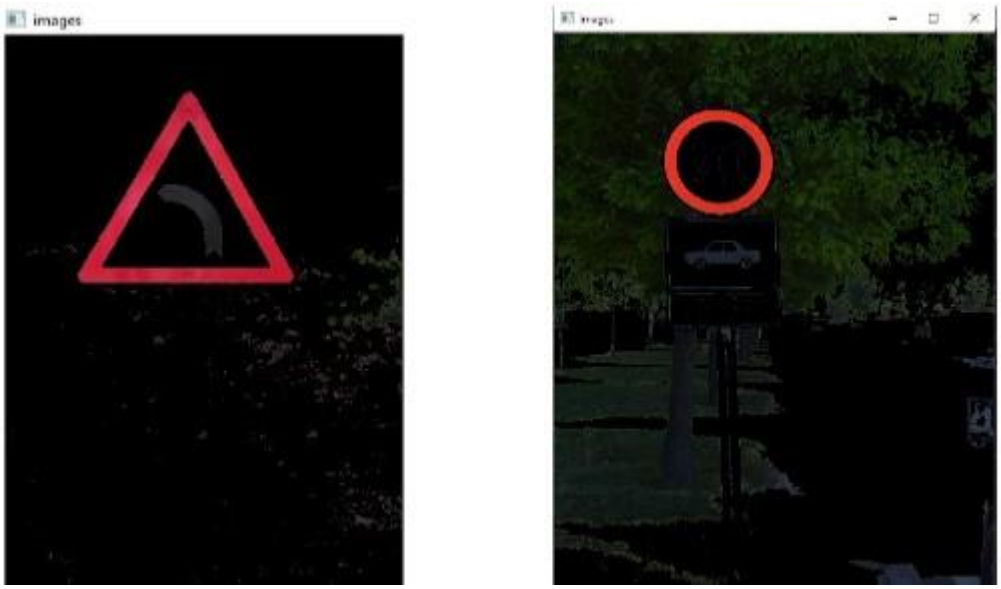

Figure 19 Screen Output - 3

We show the code we use to clean the noises in our pictures in Figure 5 Code Part - 4. The screen output of this code is as in Figure 20 Screen Output - 4.

Figure 20 Screen Output – 4

Figure 21 Code Part - 5 is the screen shot of the black and white conversion code as in Figure 21 Screen Output -5.

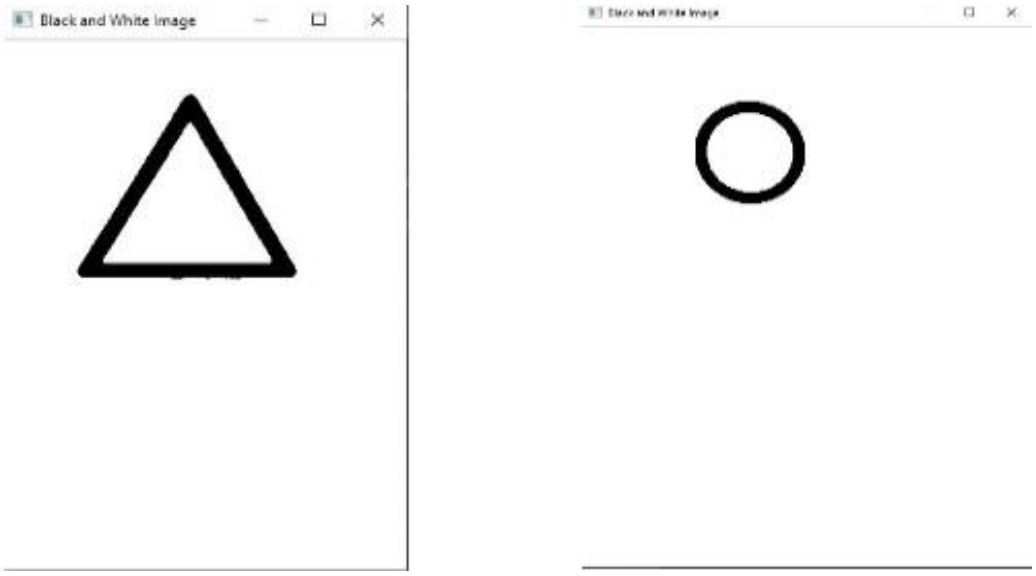

Figure 21 Screen Output - 5

The screen output of Canny Edge, which we made in Figure 7 Code Part - 6 to better perceive the edges of the picture, is as in Figure 22 Screen Output - 6.

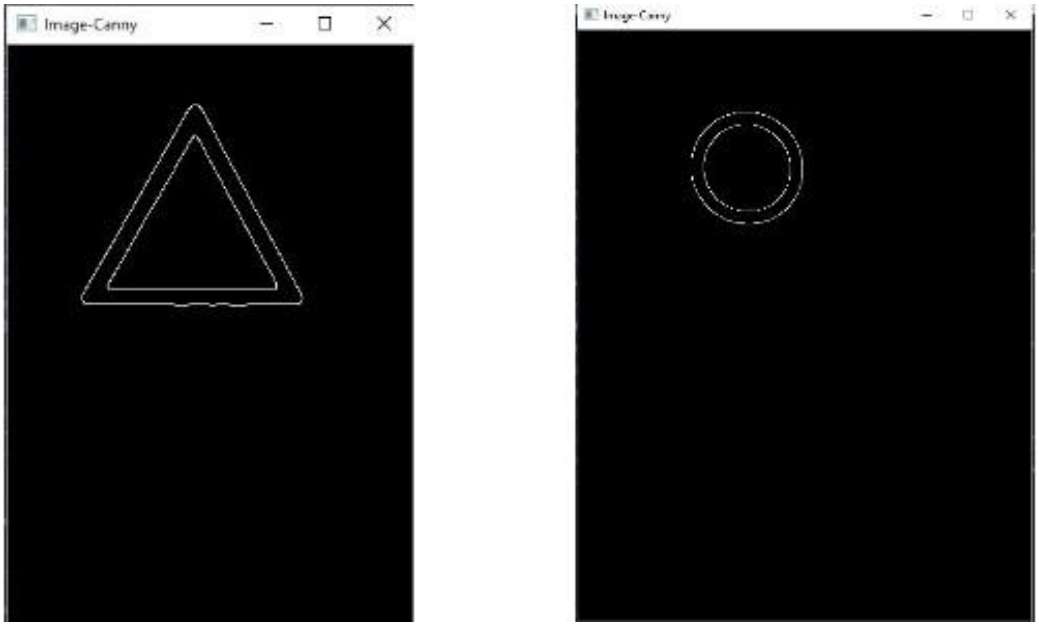

Figure 22 Screen Output -6

We can perceive geometric shapes with the code in Figure 8 Code Part -7, Figure 9 Code Part - 8 and Figure 10 Code Part - 9. Screen output is as in Figure 23 Screen Output - 7.

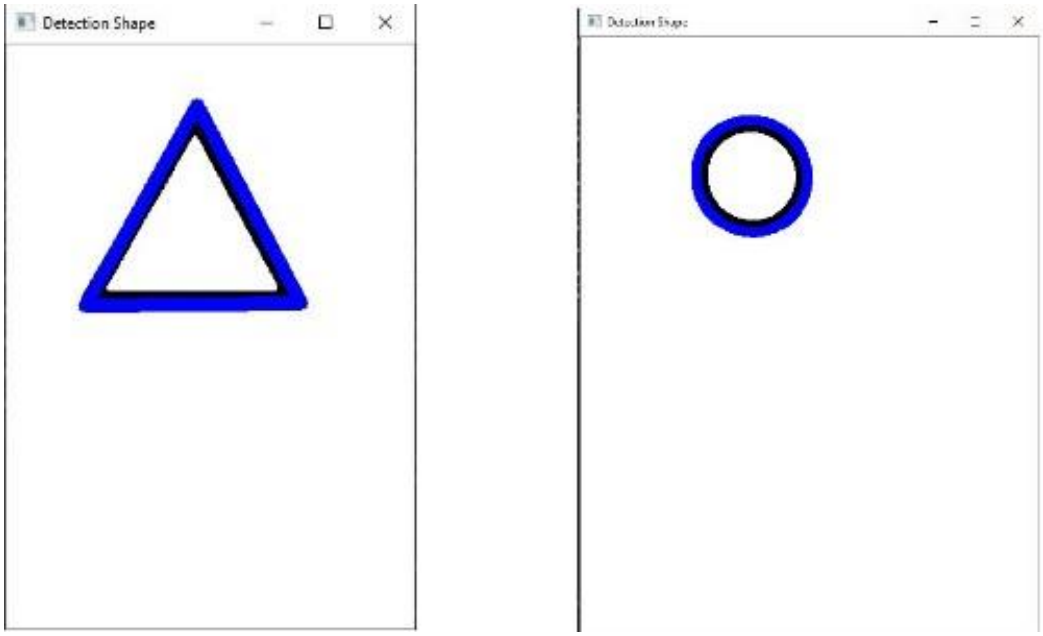

Figure 23 Screen Output - 7

The screen output of the code in Figure 11 Code Part - 10 is as in Figure 24 Screen Output - 8 and Figure 25 Screen Output - 9.

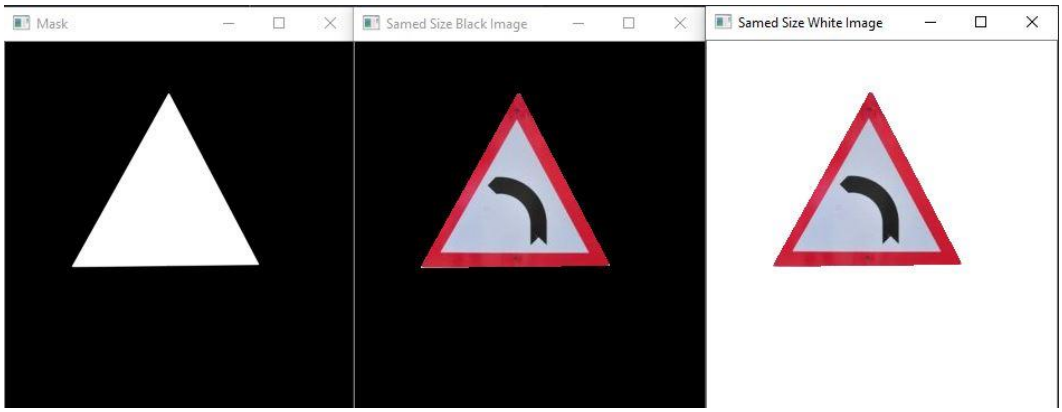

Figure 24 Screen Output – 8

Figure 25 Screen Output – 9

The screen output of the code in Figure 12 Code Part - 11 is as in Figure 26 Screen Output - 10.

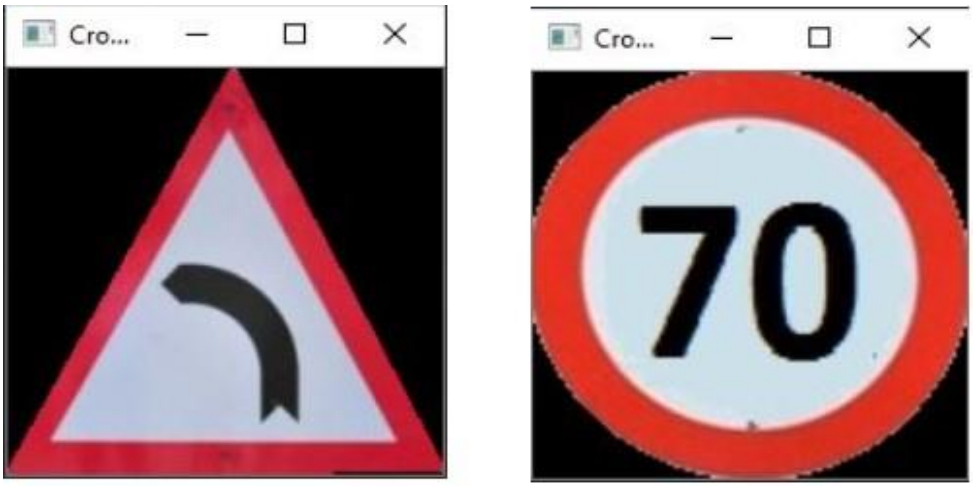

Figure 26 Screen Output – 10

Figure 13 Code Part - 12, Figure 14 Code Part - 13 similarity ratio outputs are as in Figure 27 Screen Output -11, Figure 28 Screen Output, Figure 29 Screen Output and Figure 30 Screen Output.

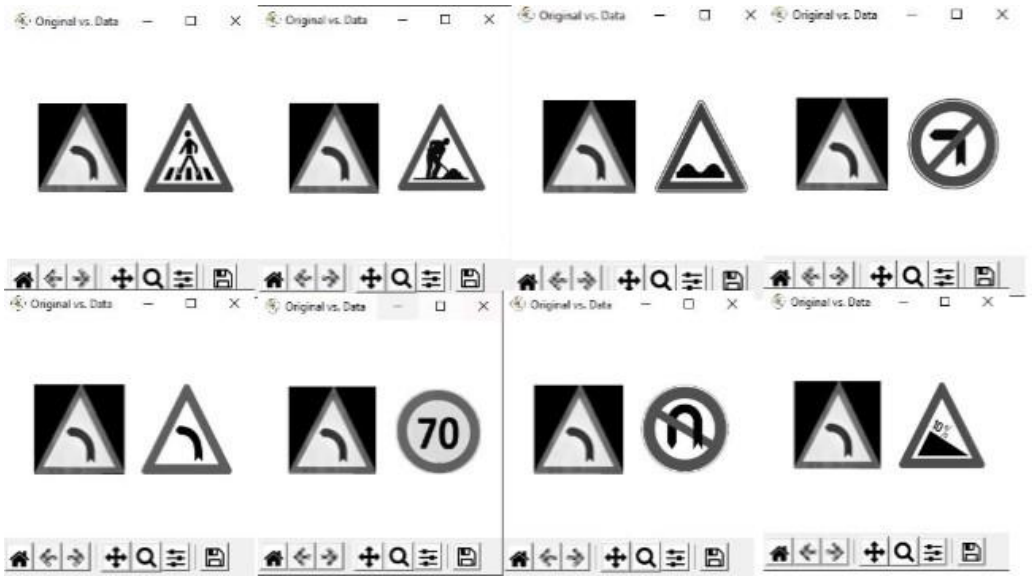

Figure 27 Screen Output - 11

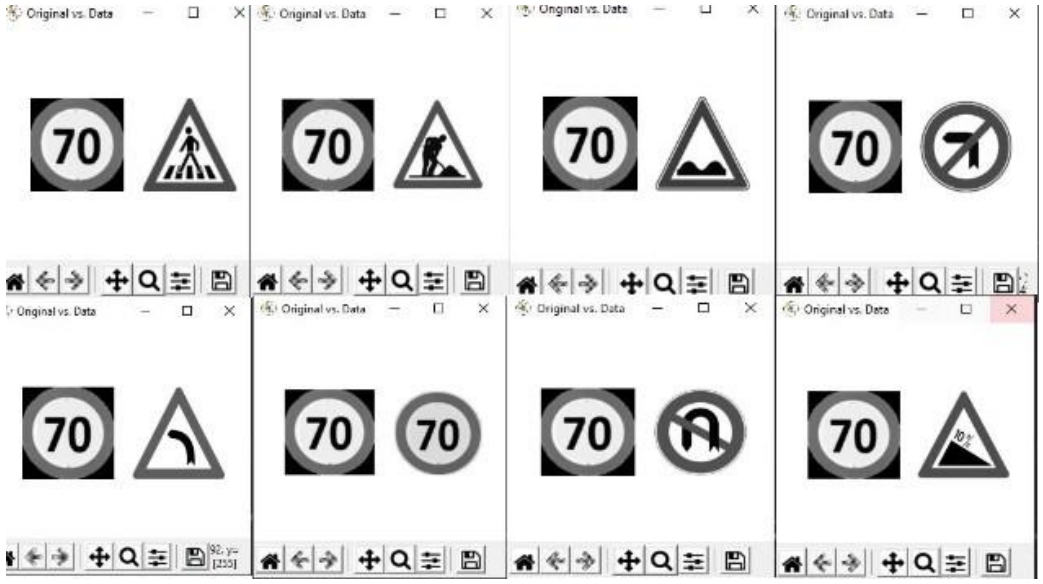

Figure 28 Screen Output – 12

```
Similarity Ratio 0.14314275895224554
Similarity Ratio 0.17001856666350473
Similarity Ratio 0.12056146481904148
Similarity Ratio 0.15795373522172132
Similarity Ratio 0.2362061702919291
Similarity Ratio 0.1533795843915343
Similarity Ratio 0.12450937239646502
Similarity Ratio 0.12532219442835407
The most similar value 0.2362061702919291
Meaning of the most similar data picture : 4
Meaning of the traffic sign  Attention left hand curve
```

Figure 29 Screen Output - 13

```
Similarity Ratio 0.14314275895224554
Similarity Ratio 0.17001856666350473
Similarity Ratio 0.12056146481904148
Similarity Ratio 0.15795373522172132
Similarity Ratio 0.2362061702919291
Similarity Ratio 0.1533795843915343
Similarity Ratio 0.12450937239646502
Similarity Ratio 0.12532219442835407
The most similar value 0.2362061702919291
Meaning of the most similar data picture : 4
Meaning of the traffic sign  Attention left hand curve
```

Figure 30 Screen Output – 14

Finally, there is an understanding of the names from the code found in Figure 15 Code Part -14. The screen output of the code in Figure 16 Code Part -15 is the same as in Figure 31 Screen Output - 15.

Figure 31 Screen Output - 15

4.Presentation Link

https://youtu.be/JINI7EgJsBo

5. References

[1] A. De La Escalera, L. E. Moreno, M. A. Salichs, and J. M. Armingol, “Road traffic sign detection and classification,” IEEE Trans. Ind. Electron., vol. 44, no. 6, pp. 848–859, 1997, doi: 10.1109/41.649946.

[2] U. S. AYDIN, “TRAFFIC SIGN RECOGNITION2.pdf,” no. May, 2009.

[3] H. Y. Yalıç and A. B. Can, “Automatic recognition of traffic signs,” ISPA 2011 - 7th Int. Symp. Image Signal Process. Anal., pp. 361–366, 2011.

[4] A. ELLAHYANI, M. EL, I. EL, and S. CHARFI, “Traffic Sign Detection and Recognition using Features Combination and Random Forests,” Int. J. Adv. Comput. Sci. Appl., vol. 7, no. 1, pp. 1–8, 2016, doi: 10.14569/ijacsa.2016.070193.

[5] P. Vargas del Valle et al., “DENGESIZ VERI SETLERINDE TRAFIK ISARETLERINI TANIMA,” Am. J. Orthod. Dentofac. Orthop., vol. ı20, no. 1, pp. 1–8, 2016, [Online]. Available: https://doi.org/10.1016/j.ajodo.2017.09.016%0Ahttp://www.ncbi.nlm.nih.gov/pubmed/?term=M+Yamaguchi+RANK+?+RANKL+?+OPG+during+orthodontic+tooth+movement%0Ahttps://doi.org/10.1016/j.ajodo.2018.10.015%0Ahttp://dx.doi.org/10.1186/s40510-016-0158-5%0Ahttp://www.crd.yo.

[6] P. Choudhary and N. R. Velaga, “TRAFFIC SIGN RECOGNITION BASED ON ARTIFICIAL NEURAL NETWORK,” pp. 1–9, 2017.

[7] P. Choudhary and N. R. Velaga, “TRAFFIC SIGN DETECTION AND RECOGNITION,” pp. 1–9, 2017.

[8] T. Acarman, “Trafik işaret levhası tespiti için derin öğrenme yöntemi,” vol. 0798, pp. 0–1.