BREADTH-FIRST

```
# BFS algorithm in Python


import collections

# BFS algorithm
def bfs(graph, root):

    visited, queue = set(), collections.deque([root])
    visited.add(root)

    while queue:

        # Dequeue a vertex from queue
        vertex = queue.popleft()
        print(str(vertex) + " ", end="")

        # If not visited, mark it as visited, and
        # enqueue it
        for neighbour in graph[vertex]:
            if neighbour not in visited:
                visited.add(neighbour)
                queue.append(neighbour)


if __name__ == '__main__':
    graph = {0: [1, 2], 1: [2], 2: [3], 3: [1, 2]}
    print("Following is Breadth First Traversal: ")
    bfs(graph, 0)
```

UNIFORM COST SEARCH

```
# Python3 implementation of above approach

# returns the minimum cost in a vector( if
# there are multiple goal states)
def  uniform_cost_search(goal, start):

    # minimum cost upto
    # goal state from starting
    global graph,cost
    answer = []

    # create a priority queue
```

```
queue = []

# set the answer vector to max value
for i in range(len(goal)):
    answer.append(10**8)

# insert the starting index
queue.append([0, start])

# map to store visited node
visited = {}

# count
count = 0

# while the queue is not empty
while (len(queue) > 0):

    # get the top element of the
    queue = sorted(queue)
    p = queue[-1]

    # pop the element
    del queue[-1]

    # get the original value
    p[0] *= -1

    # check if the element is part of
    # the goal list
    if (p[1] in goal):

        # get the position
        index = goal.index(p[1])

        # if a new goal is reached
        if (answer[index] == 10**8):
            count += 1

        # if the cost is less
        if (answer[index] > p[0]):
            answer[index] = p[0]
```

```
            # pop the element
            del queue[-1]

            queue = sorted(queue)
            if (count == len(goal)):
                return answer

        # check for the non visited nodes
        # which are adjacent to present node
        if (p[1] not in visited):
            for i in range(len(graph[p[1]])):

                # value is multiplied by -1 so that
                # least priority is at the top
                queue.append( [(p[0] + cost[(p[1], graph[p[1]][i])])* -1, graph[p[1]][i]])

        # mark as visited
        visited[p[1]] = 1

    return answer

# main function
if __name__ == '__main__':

    # create the graph
    graph,cost = [[] for i in range(8)],{}

    # add edge
    graph[0].append(1)
    graph[0].append(3)
    graph[3].append(1)
    graph[3].append(6)
    graph[3].append(4)
    graph[1].append(6)
    graph[4].append(2)
    graph[4].append(5)
    graph[2].append(1)
    graph[5].append(2)
    graph[5].append(6)
    graph[6].append(4)

    # add the cost
    cost[(0, 1)] = 2
```

```
    cost[(0, 3)] = 5
    cost[(1, 6)] = 1
    cost[(3, 1)] = 5
    cost[(3, 6)] = 6
    cost[(3, 4)] = 2
    cost[(2, 1)] = 4
    cost[(4, 2)] = 4
    cost[(4, 5)] = 3
    cost[(5, 2)] = 6
    cost[(5, 6)] = 3
    cost[(6, 4)] = 7

    # goal state
    goal = []

    # set the goal
    # there can be multiple goal states
    goal.append(6)

    # get the answer
    answer = uniform_cost_search(goal, 0)

    # print the answer
    print("Minimum cost from 0 to 6 is = ",answer[0])
```

BEST-FIRST

```
from queue import PriorityQueue

v = 14

graph = [[] for i in range(v)]


# Function For Implementing Best First Search

# Gives output path having lowest cost



def best_first_search(actual_Src, target, n):

    visited = [False] * n
```

```
    pq = PriorityQueue()
    pq.put((0, actual_Src))
    visited[actual_Src] = True

    while pq.empty() == False:
        u = pq.get()[1]
        # Displaying the path having lowest cost
        print(u, end=" ")
        if u == target:
            break

        for v, c in graph[u]:
            if visited[v] == False:
                visited[v] = True
                pq.put((c, v))
    print()

# Function for adding edges to graph


def addedge(x, y, cost):
    graph[x].append((y, cost))
    graph[y].append((x, cost))


# The nodes shown in above example(by alphabets) are
# implemented using integers addedge(x,y,cost);
addedge(0, 1, 3)
```

addedge(0, 2, 6)

addedge(0, 3, 5)

addedge(1, 4, 9)

addedge(1, 5, 8)

addedge(2, 6, 12)

addedge(2, 7, 14)

addedge(3, 8, 7)

addedge(8, 9, 5)

addedge(8, 10, 6)

addedge(9, 11, 1)

addedge(9, 12, 10)

addedge(9, 13, 2)

source = 0

target = 9

best_first_search(source, target, v)

HILL CLIMB SEARCH

```
import
random


    def randomSolution(tsp):
        cities = list(range(len(tsp)))
        solution = []


        for i in range(len(tsp)):
            randomCity = cities[random.randint(0, len(cities) -
    1)]
            solution.append(randomCity)
            cities.remove(randomCity)
```

```
    return solution


def routeLength(tsp, solution):
    routeLength = 0
    for i in range(len(solution)):
        routeLength += tsp[solution[i - 1]][solution[i]]
    return routeLength


def getNeighbours(solution):
    neighbours = []
    for i in range(len(solution)):
        for j in range(i + 1, len(solution)):
            neighbour = solution.copy()
            neighbour[i] = solution[j]
            neighbour[j] = solution[i]
            neighbours.append(neighbour)
    return neighbours


def getBestNeighbour(tsp, neighbours):
    bestRouteLength = routeLength(tsp, neighbours[0])
    bestNeighbour = neighbours[0]
    for neighbour in neighbours:
        currentRouteLength = routeLength(tsp, neighbour)
        if currentRouteLength < bestRouteLength:
            bestRouteLength = currentRouteLength
            bestNeighbour = neighbour
    return bestNeighbour, bestRouteLength


def hillClimbing(tsp):
    currentSolution = randomSolution(tsp)
    currentRouteLength = routeLength(tsp, currentSolution)
    neighbours = getNeighbours(currentSolution)
    bestNeighbour, bestNeighbourRouteLength =
getBestNeighbour(tsp, neighbours)


    while bestNeighbourRouteLength < currentRouteLength:
        currentSolution = bestNeighbour
        currentRouteLength = bestNeighbourRouteLength
        neighbours = getNeighbours(currentSolution)
        bestNeighbour, bestNeighbourRouteLength =
getBestNeighbour(tsp, neighbours)
```

```
    return currentSolution, currentRouteLength


def main():
    tsp = [
        [0, 400, 500, 300],
        [400, 0, 300, 500],
        [500, 300, 0, 400],
        [300, 500, 400, 0]
    ]


    print(hillClimbing(tsp))


if __name__ == "__main__":
    main()
```

MINIMAX

```
# A simple Python3 program to find
# maximum score that
# maximizing player can get
import math


def minimax (curDepth, nodeIndex,
             maxTurn, scores,
             targetDepth):
```

```
    # base case : targetDepth reached
    if (curDepth == targetDepth):
        return scores[nodeIndex]

    if (maxTurn):
        return max(minimax(curDepth + 1, nodeIndex * 2,
                   False, scores, targetDepth),
                   minimax(curDepth + 1, nodeIndex * 2 + 1,
                   False, scores, targetDepth))

    else:
        return min(minimax(curDepth + 1, nodeIndex * 2,
                    True, scores, targetDepth),
                   minimax(curDepth + 1, nodeIndex * 2 + 1,
                    True, scores, targetDepth))

# Driver code
scores = [3, 5, 2, 9, 12, 5, 23, 23]

treeDepth = math.log(len(scores), 2)

print("The optimal value is : ", end = "")
print(minimax(0, 0, True, scores, treeDepth))
```

ALPHA-BETA PRUNING

```
# Python3 program to demonstrate
# working of Alpha-Beta Pruning
```

```
# Initial values of Alpha and Beta
MAX, MIN = 1000, -1000

# Returns optimal value for current player
#(Initially called for root and maximizer)
def minimax(depth, nodeIndex, maximizingPlayer,
            values, alpha, beta):

    # Terminating condition. i.e
    # leaf node is reached
    if depth == 3:
        return values[nodeIndex]

    if maximizingPlayer:

        best = MIN

        # Recur for left and right children
        for i in range(0, 2):

            val = minimax(depth + 1, nodeIndex * 2 + i,
                          False, values, alpha, beta)
            best = max(best, val)
            alpha = max(alpha, best)

            # Alpha Beta Pruning
            if beta <= alpha:
```

```
            break

        return best

    else:
        best = MAX

        # Recur for left and
        # right children
        for i in range(0, 2):

            val = minimax(depth + 1, nodeIndex * 2 + i,
                          True, values, alpha, beta)
            best = min(best, val)
            beta = min(beta, best)

            # Alpha Beta Pruning
            if beta <= alpha:
                break

        return best

# Driver Code
if __name__ == "__main__":

    values = [3, 5, 6, 9, 1, 2, 0, -1]
    print("The optimal value is :", minimax(0, 0, True, values, MIN, MAX))
```

ANN for face mask

-Using pip python package manager you can install Jupyter notebook:

```
pip3 install notebook
```

And that's it, you have installed jupyter notebook

-After installing Jupyter notebook you can run the notebook server. To run the notebook, open terminal and type:

```
jupyter notebook
```

It will start the notebook server at http://localhost:8888

Make a python file train.py to write the code for training the neural network on our dataset. Follow the steps:

1. Imports:

Import all the libraries and modules required.

```
from keras.optimizers import RMSprop
from keras.preprocessing.image import ImageDataGenerator
import cv2
from keras.models import Sequential
from keras.layers import Conv2D, Input, ZeroPadding2D, BatchNormalization, Activation, MaxPooling2D, Flatten,
Dense,Dropout
from keras.models import Model, load_model
from keras.callbacks import TensorBoard, ModelCheckpoint
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
from sklearn.utils import shuffle
import imutils
import numpy as np
```

2. Build the neural network:

This convolution network consists of two pairs of Conv and MaxPool layers to extract features from the dataset. Which is then followed by a Flatten and Dropout layer to convert the data in 1D and ensure overfitting.

And then two Dense layers for classification.

```
model = Sequential([
Conv2D(100, (3,3), activation='relu', input_shape=(150, 150, 3)),
MaxPooling2D(2,2),
Conv2D(100, (3,3), activation='relu'),
MaxPooling2D(2,2),
Flatten(),
```

```
Dropout(0.5),
Dense(50, activation='relu'),
Dense(2, activation='softmax')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
```

3. Image Data Generation/Augmentation:

```
TRAINING_DIR = "./train"
train_datagen = ImageDataGenerator(rescale=1.0/255,
rotation_range=40,
width_shift_range=0.2,
height_shift_range=0.2,
shear_range=0.2,
zoom_range=0.2,
horizontal_flip=True,
fill_mode='nearest')
train_generator = train_datagen.flow_from_directory(TRAINING_DIR,
batch_size=10,
target_size=(150, 150))
VALIDATION_DIR = "./test"
validation_datagen = ImageDataGenerator(rescale=1.0/255)
validation_generator = validation_datagen.flow_from_directory(VALIDATION_DIR,
batch_size=10,
target_size=(150, 150))
```

4. Initialize a callback checkpoint to keep saving best model after each epoch while training:

```
checkpoint = ModelCheckpoint('model2-{epoch:03d}.model',monitor='val_loss',verbose=0,save_best_only=True,mode='auto')
```

5. Train the model:

```
history = model.fit_generator(train_generator,
epochs=10,
validation_data=validation_generator,
callbacks=[checkpoint])
```

Now we will test the results of face mask detector model using OpenCV.

Make a python file “test.py” and paste the below script.

```
import cv2
import numpy as np
from keras.models import load_model
model=load_model("./model-010.h5")
results={0:'without mask',1:'mask'}
GR_dict={0:(0,0,255),1:(0,255,0)}
rect_size = 4
cap = cv2.VideoCapture(0)
haarcascade = cv2.CascadeClassifier('/home/user_name/.local/lib/python3.6/site-
packages/cv2/data/haarcascade_frontalface_default.xml')
```

```
while True:
(rval, im) = cap.read()
im=cv2.flip(im,1,1)
rerect_size = cv2.resize(im, (im.shape[1] // rect_size, im.shape[0] // rect_size))
faces = haarcascade.detectMultiScale(rerect_size)
for f in faces:
(x, y, w, h) = [v * rect_size for v in f]
face_img = im[y:y+h, x:x+w]
rerect_sized=cv2.resize(face_img,(150,150))
normalized=rerect_sized/255.0
reshaped=np.reshape(normalized,(1,150,150,3))
reshaped = np.vstack([reshaped])
result=model.predict(reshaped)
label=np.argmax(result,axis=1)[0]
cv2.rectangle(im,(x,y),(x+w,y+h),GR_dict[label],2)
cv2.rectangle(im,(x,y-40),(x+w,y),GR_dict[label],-1)
cv2.putText(im, results[label], (x, y-10),cv2.FONT_HERSHEY_SIMPLEX,0.8,(255,255,255),2)
cv2.imshow('LIVE', im)
key = cv2.waitKey(10)
if key == 27:
break
cap.release()
cv2.destroyAllWindows()
```

Run the project and observe the model performance.

```
python3 test.py
```

ANN FOR TRAFFIC

https://github.com/hoanglehaithanh/Traffic-Sign-Detection/blob/master/main.py

```
import
cv2
        import numpy as np
        import matplotlib.pyplot as plt
        from math import sqrt
        from skimage.feature import blob_dog, blob_log, blob_doh
        import imutils
        import argparse
        import os
```

```
import math

from classification import training, getLabel

SIGNS = ["ERROR",
        "STOP",
        "TURN LEFT",
        "TURN RIGHT",
        "DO NOT TURN LEFT",
        "DO NOT TURN RIGHT",
        "ONE WAY",
        "SPEED LIMIT",
        "OTHER"]

# Clean all previous file
def clean_images():
    file_list = os.listdir('./')
    for file_name in file_list:
        if '.png' in file_name:
            os.remove(file_name)


### Preprocess image
```

```
def constrastLimit(image):
    img_hist_equalized = cv2.cvtColor(image, cv2.COLOR_BGR2YCrCb)
    channels = cv2.split(img_hist_equalized)
    channels[0] = cv2.equalizeHist(channels[0])
    img_hist_equalized = cv2.merge(channels)
    img_hist_equalized = cv2.cvtColor(img_hist_equalized,
cv2.COLOR_YCrCb2BGR)
    return img_hist_equalized


def LaplacianOfGaussian(image):
    LoG_image = cv2.GaussianBlur(image, (3,3), 0)          # paramter
    gray = cv2.cvtColor( LoG_image, cv2.COLOR_BGR2GRAY)
    LoG_image = cv2.Laplacian( gray, cv2.CV_8U,3,3,2)       # parameter
    LoG_image = cv2.convertScaleAbs(LoG_image)
    return LoG_image


def binarization(image):
    thresh = cv2.threshold(image,32,255,cv2.THRESH_BINARY)[1]
    #thresh =
cv2.adaptiveThreshold(image,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY,11,2)
    return thresh


def preprocess_image(image):
    image = constrastLimit(image)
    image = LaplacianOfGaussian(image)
    image = binarization(image)
```

```
    return image

# Find Signs
def removeSmallComponents(image, threshold):
    #find all your connected components (white blobs in your image)
    nb_components, output, stats, centroids =
cv2.connectedComponentsWithStats(image, connectivity=8)
    sizes = stats[1:, -1]; nb_components = nb_components - 1

    img2 = np.zeros((output.shape),dtype = np.uint8)
    #for every component in the image, you keep it only if it's above
threshold
    for i in range(0, nb_components):
        if sizes[i] >= threshold:
            img2[output == i + 1] = 255
    return img2


def findContour(image):
    #find contours in the thresholded image
    cnts = cv2.findContours(image, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE    )
    cnts = cnts[0] if imutils.is_cv2() else cnts[1]
    return cnts


def contourIsSign(perimeter, centroid, threshold):
```

```
#  perimeter, centroid, threshold
# # Compute signature of contour
result=[]
for p in perimeter:
    p = p[0]
    distance = sqrt((p[0] - centroid[0])**2 + (p[1] - centroid[1])**2)
    result.append(distance)
max_value = max(result)
signature = [float(dist) / max_value for dist in result ]
# Check signature of contour.
temp = sum((1 - s) for s in signature)
temp = temp / len(signature)
if temp < threshold: # is  the sign
    return True, max_value + 2
else:                # is not the sign
    return False, max_value + 2
```

```
#crop sign
def cropContour(image, center, max_distance):
    width = image.shape[1]
    height = image.shape[0]
    top = max([int(center[0] - max_distance), 0])
    bottom = min([int(center[0] + max_distance + 1), height-1])
    left = max([int(center[1] - max_distance), 0])
    right = min([int(center[1] + max_distance+1), width-1])
    print(left, right, top, bottom)
    return image[left:right, top:bottom]
```

```
def cropSign(image, coordinate):
    width = image.shape[1]
    height = image.shape[0]
    top = max([int(coordinate[0][1]), 0])
    bottom = min([int(coordinate[1][1]), height-1])
    left = max([int(coordinate[0][0]), 0])
    right = min([int(coordinate[1][0]), width-1])
    #print(top,left,bottom,right)
    return image[top:bottom,left:right]


def findLargestSign(image, contours, threshold, distance_theshold):
    max_distance = 0
    coordinate = None
    sign = None
    for c in contours:
        M = cv2.moments(c)
        if M["m00"] == 0:
            continue
        cX = int(M["m10"] / M["m00"])
        cY = int(M["m01"] / M["m00"])
        is_sign, distance = contourIsSign(c, [cX, cY], 1-threshold)
        if is_sign and distance > max_distance and distance >
distance_theshold:
```

```
                max_distance = distance
                coordinate = np.reshape(c, [-1,2])
                left, top = np.amin(coordinate, axis=0)
                right, bottom = np.amax(coordinate, axis = 0)
                coordinate = [(left-2,top-2),(right+3,bottom+1)]
                sign = cropSign(image,coordinate)
            return sign, coordinate



def findSigns(image, contours, threshold, distance_theshold):
    signs = []
    coordinates = []
    for c in contours:
        # compute the center of the contour
        M = cv2.moments(c)
        if M["m00"] == 0:
            continue
        cX = int(M["m10"] / M["m00"])
        cY = int(M["m01"] / M["m00"])
        is_sign, max_distance = contourIsSign(c, [cX, cY], 1-threshold)
        if is_sign and max_distance > distance_theshold:
            sign = cropContour(image, [cX, cY], max_distance)
            signs.append(sign)
            coordinate = np.reshape(c, [-1,2])
            top, left = np.amin(coordinate, axis=0)
            right, bottom = np.amax(coordinate, axis = 0)
```

```
            coordinates.append([(top-2,left-2),(right+1,bottom+1)])
    return signs, coordinates


def localization(image, min_size_components,
similitary_contour_with_circle, model, count, current_sign_type):
    original_image = image.copy()
    binary_image = preprocess_image(image)


    binary_image = removeSmallComponents(binary_image,
min_size_components)


    binary_image = cv2.bitwise_and(binary_image,binary_image,
mask=remove_other_color(image))


    #binary_image = remove_line(binary_image)


    cv2.imshow('BINARY IMAGE', binary_image)
    contours = findContour(binary_image)
    #signs, coordinates = findSigns(image, contours,
similitary_contour_with_circle, 15)
    sign, coordinate = findLargestSign(original_image, contours,
similitary_contour_with_circle, 15)


    text = ""
    sign_type = -1
```

```
        i = 0

        if sign is not None:
            sign_type = getLabel(model, sign)
            sign_type = sign_type if sign_type <= 8 else 8
            text = SIGNS[sign_type]
            cv2.imwrite(str(count)+'_'+text+'.png', sign)

        if sign_type > 0 and sign_type != current_sign_type:
            cv2.rectangle(original_image, coordinate[0],coordinate[1], (0, 255, 0),
1)
            font = cv2.FONT_HERSHEY_PLAIN
            cv2.putText(original_image,text,(coordinate[0][0], coordinate[0][1] -
15), font, 1,(0,0,255),2,cv2.LINE_4)
        return coordinate, original_image, sign_type, text

    def remove_line(img):
        gray = img.copy()
        edges = cv2.Canny(gray,50,150,apertureSize = 3)
        minLineLength = 5
        maxLineGap = 3
        lines =
cv2.HoughLinesP(edges,1,np.pi/180,15,minLineLength,maxLineGap)
        mask = np.ones(img.shape[:2], dtype="uint8") * 255
        if lines is not None:
            for line in lines:
```

```
            for x1,y1,x2,y2 in line:
                cv2.line(mask,(x1,y1),(x2,y2),(0,0,0),2)
        return cv2.bitwise_and(img, img, mask=mask)


    def remove_other_color(img):
        frame = cv2.GaussianBlur(img, (3,3), 0)
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        # define range of blue color in HSV
        lower_blue = np.array([100,128,0])
        upper_blue = np.array([215,255,255])
        # Threshold the HSV image to get only blue colors
        mask_blue = cv2.inRange(hsv, lower_blue, upper_blue)


        lower_white = np.array([0,0,128], dtype=np.uint8)
        upper_white = np.array([255,255,255], dtype=np.uint8)
        # Threshold the HSV image to get only blue colors
        mask_white = cv2.inRange(hsv, lower_white, upper_white)


        lower_black = np.array([0,0,0], dtype=np.uint8)
        upper_black = np.array([170,150,50], dtype=np.uint8)


        mask_black = cv2.inRange(hsv, lower_black, upper_black)
```

```
        mask_1 = cv2.bitwise_or(mask_blue, mask_white)
        mask = cv2.bitwise_or(mask_1, mask_black)
        # Bitwise-AND mask and original image
        #res = cv2.bitwise_and(frame,frame, mask= mask)
        return mask


    def main(args):
            #Clean previous image
        clean_images()
        #Training phase
        model = training()

        vidcap = cv2.VideoCapture(args.file_name)

        fps = vidcap.get(cv2.CAP_PROP_FPS)
        width = vidcap.get(3)  # float
        height = vidcap.get(4) # float

        # Define the codec and create VideoWriter object
        fourcc = cv2.VideoWriter_fourcc(*'XVID')
        out = cv2.VideoWriter('output.avi',fourcc, fps , (640,480))

        # initialize the termination criteria for cam shift, indicating
```

```
        # a maximum of ten iterations or movement by a least one pixel
        # along with the bounding box of the ROI
        termination = (cv2.TERM_CRITERIA_EPS |
    cv2.TERM_CRITERIA_COUNT, 10, 1)
        roiBox = None
        roiHist = None


        success = True
        similitary_contour_with_circle = 0.65   # parameter
        count = 0
        current_sign = None
        current_text = ""
        current_size = 0
        sign_count = 0
        coordinates = []
        position = []
        file = open("Output.txt", "w")
        while True:
            success,frame = vidcap.read()
            if not success:
                print("FINISHED")
                break
            width = frame.shape[1]
            height = frame.shape[0]
            #frame = cv2.resize(frame, (640,int(height/(width/640))))
            frame = cv2.resize(frame, (640,480))
```

```
            print("Frame:{}".format(count))
            #image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
            coordinate, image, sign_type, text = localization(frame,
        args.min_size_components, args.similitary_contour_with_circle, model,
        count, current_sign)
            if coordinate is not None:
                cv2.rectangle(image, coordinate[0],coordinate[1], (255, 255, 255), 1)
            print("Sign:{}".format(sign_type))
            if sign_type > 0 and (not current_sign or sign_type != current_sign):
                current_sign = sign_type
                current_text = text
                top = int(coordinate[0][1]*1.05)
                left = int(coordinate[0][0]*1.05)
                bottom = int(coordinate[1][1]*0.95)
                right = int(coordinate[1][0]*0.95)


                position = [count, sign_type if sign_type <= 8 else 8,
        coordinate[0][0], coordinate[0][1], coordinate[1][0], coordinate[1][1]]
                cv2.rectangle(image, coordinate[0],coordinate[1], (0, 255, 0), 1)
                font = cv2.FONT_HERSHEY_PLAIN
                cv2.putText(image,text,(coordinate[0][0], coordinate[0][1] -15),
        font, 1,(0,0,255),2,cv2.LINE_4)


                tl = [left, top]
                br = [right,bottom]
                print(tl, br)
```

```
                current_size = math.sqrt(math.pow((tl[0]-br[0]),2) +
math.pow((tl[1]-br[1]),2))
                # grab the ROI for the bounding box and convert it
                # to the HSV color space
                roi = frame[tl[1]:br[1], tl[0]:br[0]]
                roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
                #roi = cv2.cvtColor(roi, cv2.COLOR_BGR2LAB)

                # compute a HSV histogram for the ROI and store the
                # bounding box
                roiHist = cv2.calcHist([roi], [0], None, [16], [0, 180])
                roiHist = cv2.normalize(roiHist, roiHist, 0, 255,
cv2.NORM_MINMAX)
                roiBox = (tl[0], tl[1], br[0], br[1])

            elif current_sign:
                hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
                backProj = cv2.calcBackProject([hsv], [0], roiHist, [0, 180], 1)

                # apply cam shift to the back projection, convert the
                # points to a bounding box, and then draw them
                (r, roiBox) = cv2.CamShift(backProj, roiBox, termination)
                pts = np.into(cv2.boxPoints(r))
                s = pts.sum(axis = 1)
                tl = pts[np.argmin(s)]
                br = pts[np.argmax(s)]
```

```
                size = math.sqrt(pow((tl[0]-br[0]),2) +pow((tl[1]-br[1]),2))
                print(size)

                if  current_size < 1 or size < 1 or size / current_size > 30 or
            math.fabs((tl[0]-br[0])/(tl[1]-br[1])) > 2 or math.fabs((tl[0]-br[0])/(tl[1]-
            br[1])) < 0.5:
                    current_sign = None
                    print("Stop tracking")
                else:
                    current_size = size

                if sign_type > 0:
                    top = int(coordinate[0][1])
                    left = int(coordinate[0][0])
                    bottom = int(coordinate[1][1])
                    right = int(coordinate[1][0])

                    position = [count, sign_type if sign_type <= 8 else 8, left, top,
            right, bottom]
                    cv2.rectangle(image, coordinate[0],coordinate[1], (0, 255, 0), 1)
                    font = cv2.FONT_HERSHEY_PLAIN
                    cv2.putText(image,text,(coordinate[0][0], coordinate[0][1] -15),
            font, 1,(0,0,255),2,cv2.LINE_4)
                elif current_sign:
                    position = [count, sign_type if sign_type <= 8 else 8, tl[0], tl[1],
            br[0], br[1]]
                    cv2.rectangle(image, (tl[0], tl[1]),(br[0], br[1]), (0, 255, 0), 1)
```

```
                font = cv2.FONT_HERSHEY_PLAIN
                cv2.putText(image,current_text,(tl[0], tl[1] -15), font,
1,(0,0,255),2,cv2.LINE_4)


            if current_sign:
                sign_count += 1
                coordinates.append(position)


            cv2.imshow('Result', image)
            count = count + 1
            #Write to video
            out.write(image)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
        file.write("{}".format(sign_count))
        for pos in coordinates:
            file.write("\n{} {} {} {} {}
{}".format(pos[0],pos[1],pos[2],pos[3],pos[4], pos[5]))
        print("Finish {} frames".format(count))
        file.close()
        return



if __name__ == '__main__':
```

```
    parser = argparse.ArgumentParser(description="NLP Assignment
Command Line")

    parser.add_argument(
     '--file_name',
     default= "./MVI_1049.avi",
     help= "Video to be analyzed"
     )

    parser.add_argument(
     '--min_size_components',
     type = int,
     default= 300,
     help= "Min size component to be reserved"
     )


    parser.add_argument(
     '--similitary_contour_with_circle',
     type = float,
     default= 0.65,
     help= "Similitary to a circle"
     )

    args = parser.parse_args()
    main(args)
```