Project Development Phase

Date	14 November 2022
Team ID	PNT2022TMID18651
Project Name	Virtual Eye - Life Guard for Swimming Pools to Detect Active Drowning

```
#import necessary

packages import cv2

import os

import numpy as np

from .utils import download_file


initialize =

True net =

None

dest_dir = os.path.expanduser('~') + os.path.sep + '.cvlib' + os.path.sep +
'object_detection' +os.path.sep + 'yolo' + os.path.sep + 'yolov3'

classes = None

#colors are BGR instead of RGB

in python COLORS = [0,0,255],

[255,0,0]


def populate_class_labels():


  #we are using a pre existent classifier which is more reliable and more

  efficient than one#we could make using only a laptop

  #The classifier should be downloaded automatically when

  you run this script class_file_name = 'yolov3_classes.txt'

  class_file_abs_path = dest_dir + os.path.sep + class_file_name

  url = 'https://github.com/Nico31415/Drowning-
```

```
    Detector/raw/master/yolov3.txt' if not
    os.path.exists(class_file_abs_path):
        download_file(url=url, file_name=class_file_name,
    dest_dir=dest_dir) f = open(class_file_abs_path, 'r')
    classes = [line.strip() for line in f.readlines()]

    return classes


def get_output_layers(net):
  #the number of output layers in a neural network is the

  number of possible#things the network can detect, such as a

  person, a dog, a tie, a phone... layer_names =

  net.getLayerNames()


  output_layers = [layer_names[i[0] - 1] for i in
  net.getUnconnectedOutLayers()]


  return output_layers



def draw_bbox(img, bbox, labels, confidence, Drowning, write_conf=False):


  global

  COLORS

  global

  classes


  if classes is None:

    classes = populate_class_labels()


  for i, label in enumerate(labels):


    #if the person is drowning, the box will be drawn red

    instead of blue if label == 'person' and Drowning:

      color =

      COLORS[0]

      label =
      'DROWNING'
    else:
      color = COLORS[1]

    if write_conf:
      label += ' ' + str(format(confidence[i] * 100, '.2f')) + '%'
      #you only need to points (the opposite corners) to draw a

      rectangle. These points#are stored in the variable bbox

      cv2.rectangle(img, (bbox[i][0],bbox[i][1]), (bbox[i][2],bbox[i][3]), color, 2)


      cv2.putText(img, label, (bbox[i][0],bbox[i][1]-10),
      cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)


    return img


       def detect_common_objects(image, confidence=0.5,
                                                    nms_thresh=0.3):


    Height, Width =

    image.shape[:2] scale =

    0.00392


    global

    classes

    global

    dest_dir


    #all the weights and the neural network algorithm are already

    preconfigured#as we are using YOLO


    #this part of the script just downloads the

    YOLO files config_file_name =

    'yolov3.cfg'

    config_file_abs_path = dest_dir + os.path.sep + config_file_name


    weights_file_name = 'yolov3.weights'
```

```
    weights_file_abs_path = dest_dir + os.path.sep +
                                                 weights_file_name


url = 'https://github.com/Nico31415/Drowning-
Detector/raw/master/yolov3.cfg'


if not os.path.exists(config_file_abs_path):
  download_file(url=url, file_name=config_file_name, dest_dir=dest_dir)
```

```
url = 'https://pjreddie.com/media/files/yolov3.weights'

if not os.path.exists(weights_file_abs_path):
  download_file(url=url, file_name=weights_file_name, dest_dir=dest_dir)



global
initialize
global net

if initialize:
  classes = populate_class_labels()
  net = cv2.dnn.readNet(weights_file_abs_path,
  config_file_abs_path) initialize = False

blob = cv2.dnn.blobFromImage(image, scale, (416,416), (0,0,0), True,
crop=False)

net.setInput(blob)

outs = net.forward(get_output_layers(net))

class_ids = []
confidences
= [] boxes =
[]

for out in outs:
  for detection in out:
```

```
scores =
detection[5:]
class_id =
np.argmax(scores)
max_conf =
scores[class_id] if
max_conf >
confidence:
      center_x = int(detection[0] *
      Width)     center_y     =
      int(detection[1] * Height) w
      = int(detection[2] * Width)
      h = int(detection[3] *
      Height)x = center_x -
      w / 2
      y = center_y - h / 2
      class_ids.append(class_id)
      confidences.append(float(max_con
      f))boxes.append([x, y, w, h])


indices = cv2.dnn.NMSBoxes(boxes, confidences, confidence, nms_thresh)

bbox =
[] label
=   []
conf =
[]

for i in indices:
  i = i[0]
  box    =
  boxes[i] x
  = box[0]
  y    =
  box[1]
  w    =
  box[2] h

  = box[3]

  bbox.append([round(x), round(y), round(x+w),

  round(y+h)])

  label.append(str(classes[class_ids[i]]))

  conf.append(confidences[i])

return bbox, label, conf
```