**Python Code for – Detection, Page Routes**

**// app.py**

|  |
| --- |
| from flask import Flask, render\_template, Response, jsonify, request  import cv2  import numpy as np  # for accessing session storage  from flask import session, redirect  # cloudant imports  from cloudant.client import Cloudant  # sub-imports  # from object\_detection import Detect  # connecting client with cloudant db  client = Cloudant.iam('5e67dcf0-6dd2-49ef-ba49-548e2376d5fa-bluemix',  'T0BBzOvBQK6JyezcCq1xelsmRiuVe-AQ1PwdufX\_3XCL',  connect = True)  db = client.create\_database('veye\_users')  app=Flask(\_\_name\_\_)  class Detect:  def \_\_init\_\_(self, video\_source,  classes,  config,  weights,  frame\_title,  wait\_key,  threshold,  suppression\_threshold,  yolo\_image\_size):  self.video\_source = video\_source  self.classes = classes  self.config = config  self. weights = weights  self.frame\_title = frame\_title  self.wait\_key = wait\_key  self.threshold = threshold  self.suppression\_threshold = suppression\_threshold  self.yolo\_image\_size = yolo\_image\_size  self.detect\_count = 0  def find\_objects(self, model\_outputs, YOLO\_IMAGE\_SIZE, THRESHOLD, SUPPRESSION\_THRESHOLD):  bounding\_box\_locations = []  class\_ids = []  confidence\_values = []  for output in model\_outputs:  for prediction in output:  class\_probabilities = prediction[5:]  class\_id = np.argmax(class\_probabilities)  confidence = class\_probabilities[class\_id]  if confidence > THRESHOLD:  w, h = int(prediction[2] \* YOLO\_IMAGE\_SIZE), int(prediction[3] \* YOLO\_IMAGE\_SIZE)  # the center of the bounding box (we should transform these values)  x, y = int(prediction[0] \* YOLO\_IMAGE\_SIZE - w / 2), int(prediction[1] \* YOLO\_IMAGE\_SIZE - h / 2)  bounding\_box\_locations.append([x, y, w, h])  class\_ids.append(class\_id)  confidence\_values.append(float(confidence))  box\_indexes\_to\_keep = cv2.dnn.NMSBoxes(bounding\_box\_locations, confidence\_values, THRESHOLD, SUPPRESSION\_THRESHOLD)  return box\_indexes\_to\_keep, bounding\_box\_locations, class\_ids, confidence\_values  def mark\_detected\_objects(self, img, bounding\_box\_ids, all\_bounding\_boxes, class\_ids, confidence\_values, width\_ratio,  height\_ratio):  for index in bounding\_box\_ids:  bounding\_box = all\_bounding\_boxes[index]  x, y, w, h = int(bounding\_box[0]), int(bounding\_box[1]), int(bounding\_box[2]), int(bounding\_box[3])    # we have to transform the locations and coordinates because the image is resized  x = int(x \* width\_ratio)  y = int(y \* height\_ratio)  w = int(w \* width\_ratio)  h = int(h \* height\_ratio)  # OpenCV deals with BGR blue green red (255,0,0) then it is the blue color  # we are not going to detect every objects just PERSON and CAR  # if class\_ids[index] == 2:  # cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)  # class\_with\_confidence = 'CAR' + str(int(confidence\_values[index] \* 100)) + '%'  # cv2.putText(img, class\_with\_confidence, (x, y-10), cv2.FONT\_HERSHEY\_COMPLEX\_SMALL, 0.5, (255, 0, 0), 1)  if class\_ids[index] == 0:  self.detect\_count += 1  cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)  class\_with\_confidence = f'drowning' + str(int(confidence\_values[index] \* 100)) + '%'  cv2.putText(img, class\_with\_confidence, (x, y-10), cv2.FONT\_HERSHEY\_COMPLEX\_SMALL, 0.5, (255, 0, 0), 1)  # find\_objects  # mark\_detected\_objects  def generate\_frames(self):  capture = cv2.VideoCapture(self.video\_source)    neural\_network = cv2.dnn.readNetFromDarknet(self.config, self.weights)  neural\_network.setPreferableBackend(cv2.dnn.DNN\_BACKEND\_OPENCV)  neural\_network.setPreferableTarget(cv2.dnn.DNN\_TARGET\_CPU)  YOLO\_IMAGE\_SIZE = self.yolo\_image\_size  while True:  frame\_grabbed, frame = capture.read()  if not frame\_grabbed:  break  else:  original\_width, original\_height = frame.shape[1], frame.shape[0]  # the image into a BLOB [0-1] RGB - BGR  blob = cv2.dnn.blobFromImage(frame, 1 / 255, (YOLO\_IMAGE\_SIZE, YOLO\_IMAGE\_SIZE), True, crop=False)  neural\_network.setInput(blob)  layer\_names = neural\_network.getLayerNames()  # YOLO network has 3 output layer - note: these indexes are starting with 1  output\_names = [layer\_names[index - 1] for index in neural\_network.getUnconnectedOutLayers()]  self.detect\_count = 0  outputs = neural\_network.forward(output\_names)  predicted\_objects, bbox\_locations, class\_label\_ids, conf\_values = self.find\_objects(outputs,  self.yolo\_image\_size,  self.threshold,  self.suppression\_threshold)  self.mark\_detected\_objects(frame, predicted\_objects, bbox\_locations, class\_label\_ids, conf\_values,  original\_width / YOLO\_IMAGE\_SIZE, original\_height / YOLO\_IMAGE\_SIZE)  ret, buffer = cv2.imencode('.jpg', frame)  frame = buffer.tobytes()  yield (b'--frame\r\n'  b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')  # global declaration  source = Detect(video\_source = './media/swimming\_pool1.mp4',  classes = ['drowning'],  config = './config/yolov3\_testing.cfg',  weights = './weights/yolov3\_training\_3000.weights',  frame\_title = 'YOLO V3 Object Detection',  wait\_key = 10,  threshold = 0.5,  suppression\_threshold = 0.4,  yolo\_image\_size = 320)  @app.route('/counter', methods=['POST'])  def counter():  return jsonify('', render\_template('counter.html', dyn\_var = source.detect\_count))  @app.route('/video')  def video():  frame = source.generate\_frames()  return Response(frame,  mimetype='multipart/x-mixed-replace; boundary=frame')  @app.route('/detection', methods=["GET", "POST"])  def detection():  if (session.get("user\_token")):  return render\_template('detection.html', dyn\_var = source.detect\_count)  return render\_template("login\_redirect.html", dyn\_message = "You need to login first!")  # login & registration  @app.route('/validate\_login', methods=["GET", "POST"])  def validate\_login():  if request.method == "POST":  email = request.form.get("user\_login\_email")  password = request.form.get("user\_login\_password")    session["login\_username"] = email  session["login\_password"] = password  test\_login = {  '\_id': email,  'pword': password  }  # test\_login = {  # '\_id': 'veye\_admin',  # 'pword': 'veye\_admin'  # }  if (test\_login['\_id'] and test\_login['pword']) in db:  session["user\_token"] = db[test\_login['\_id']]['\_rev']    print(f"username: {session.get('login\_username')}; password: {session.get('login\_password')}")  return render\_template('login\_modules/login\_success.html',  dyn\_message = "You're in!")  return render\_template('/login.html', dyn\_message = "check your u/name or p/word")  @app.route('/logout')  def logout():  if session.get("login\_username"): session.pop("login\_username")  if session.get("login\_password"): session.pop("login\_password")  if session.get("user\_taken"): session.pop("user\_token")  return redirect("/")  @app.route('/about')  def about():  return render\_template("about.html")  @app.route('/register\_intro', methods=["GET", "POST"])  def register\_intro():  return render\_template('register\_user/register\_intro.html')  @app.route('/register\_name', methods=["POST", "GET"])  def register\_name():  # if request.method == "POST":  # register\_user\_name = request.form.get("user\_name")    # session["register\_user\_name"] = register\_user\_name  # print(f"name set: {session['register\_user\_name']}")  return render\_template('register\_user/register\_name.html')  @app.route('/register\_email', methods=["GET", "POST"])  def register\_email():  if request.method == "POST":  # retrieve user\_name from name page  register\_user\_name = request.form.get("user\_name")    session["register\_user\_name"] = register\_user\_name  print(f"name set: {session['register\_user\_name']}")  return render\_template('register\_user/register\_email.html')  @app.route('/register\_password', methods=["GET", "POST"])  def register\_password():  if request.method == "POST":  # retrieve user\_email from email page  register\_user\_email = request.form.get("user\_email")  session["register\_user\_email"] = register\_user\_email  print(f"email set: {session['register\_user\_email']}")  return render\_template('register\_user/register\_password.html')  @app.route('/register\_phoneNumber', methods=["GET", "POST"])  def register\_phoneNumber():  if request.method == "POST":  # retrieve user\_pass from password page  register\_user\_pword = request.form.get("user\_pass")  session["register\_user\_pword"] = register\_user\_pword  print(f"pword set: {session['register\_user\_pword']}")  return render\_template('register\_user/register\_phoneNumber.html')  @app.route('/register\_outro', methods=["GET", "POST"])  def register\_outro():  if request.method == "POST":  # retrieve user\_phone from phoneNumber page  register\_user\_phoneNumber = request.form.get("user\_phone")    session["register\_user\_phone"] = register\_user\_phoneNumber  print(f"phone number set: {session['register\_user\_phone']}")  register\_new\_document = {  '\_id': str(session.get("register\_user\_email")),  'name': str(session.get("register\_user\_name")),  'pword': str(session.get("register\_user\_pword")),  'phoneNumber': str(session.get("register\_user\_phone"))  }  new\_document = db.create\_document(register\_new\_document)  if new\_document.exists():  print(register\_new\_document)  return render\_template('register\_user/register\_outro.html',  dyn\_message = "You're in!")    return render\_template('register\_user/register\_outro.html',  dyn\_message = "Oops! Seems like there was a problem while registering you in. Contact Administrator.")  @app.route('/')  def login():  return render\_template('login.html', dyn\_message = "")  if \_\_name\_\_ == "\_\_main\_\_":  app.config["SESSION\_PERMANENT"] = False  app.config["SESSION\_TYPE"] = "filesystem"  app.secret\_key = "veye"  app.run(debug=True) |