**Mini project code –Real time object detection (yolo algorithm)**

from IPython.display import display, Javascript, Image

from google.colab.output import eval\_js

from google.colab.patches import cv2\_imshow

from base64 import b64decode, b64encode

import cv2

import numpy as np

import PIL

import io

import html

import time

import matplotlib.pyplot as plt

%matplotlib inline

from darknet import \*

network, class\_names, class\_colors = load\_network("cfg/yolov4-csp.cfg", "cfg/coco.data", "yolov4-csp.weights")

width = network\_width(network)

height = network\_height(network)

def darknet\_helper(img, width, height):

darknet\_image = make\_image(width, height, 3)

img\_rgb = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

img\_resized = cv2.resize(img\_rgb, (width, height), interpolation=cv2.INTER\_LINEAR)

img\_height, img\_width, \_ = img.shape

width\_ratio = img\_width/width

height\_ratio = img\_height/height

copy\_image\_from\_bytes(darknet\_image, img\_resized.tobytes())

detections = detect\_image(network, class\_names, darknet\_image)

free\_image(darknet\_image)

return detections, width\_ratio, height\_ratio

image = cv2.imread("data/person.jpg")

detections, width\_ratio, height\_ratio = darknet\_helper(image, width, height)

for label, confidence, bbox in detections:

left, top, right, bottom = bbox2points(bbox)

left, top, right, bottom = int(left \* width\_ratio), int(top \* height\_ratio), int(right \* width\_ratio), int(bottom \* height\_ratio)

cv2.rectangle(image, (left, top), (right, bottom), class\_colors[label], 2)

cv2.putText(image, "{} [{:.2f}]".format(label, float(confidence)),

(left, top - 5), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5,

class\_colors[label], 2)

cv2\_imshow(image)

def js\_to\_image(js\_reply):

image\_bytes = b64decode(js\_reply.split(',')[1])

jpg\_as\_np = np.frombuffer(image\_bytes, dtype=np.uint8)

img = cv2.imdecode(jpg\_as\_np, flags=1)

return img

def bbox\_to\_bytes(bbox\_array):

bbox\_PIL = PIL.Image.fromarray(bbox\_array, 'RGBA')

iobuf = io.BytesIO()

bbox\_PIL.save(iobuf, format='png')

bbox\_bytes = 'data:image/png;base64,{}'.format((str(b64encode(iobuf.getvalue()), 'utf-8')))

return bbox\_bytes

def take\_photo(filename='photo.jpg', quality=0.8):

js = Javascript('''

async function takePhoto(quality) {

const div = document.createElement('div');

const capture = document.createElement('button');

capture.textContent = 'Capture';

div.appendChild(capture);

const video = document.createElement('video');

video.style.display = 'block';

const stream = await navigator.mediaDevices.getUserMedia({video: true});

document.body.appendChild(div);

div.appendChild(video);

video.srcObject = stream;

await video.play();

// Resize the output to fit the video element.

google.colab.output.setIframeHeight(document.documentElement.scrollHeight, true);

// Wait for Capture to be clicked.

await new Promise((resolve) => capture.onclick = resolve);

const canvas = document.createElement('canvas');

canvas.width = video.videoWidth;

canvas.height = video.videoHeight;

canvas.getContext('2d').drawImage(video, 0, 0);

stream.getVideoTracks()[0].stop();

div.remove();

return canvas.toDataURL('image/jpeg', quality);

}

''')

display(js)

data = eval\_js('takePhoto({})'.format(quality))

img = js\_to\_image(data)

detections, width\_ratio, height\_ratio = darknet\_helper(img, width, height)

for label, confidence, bbox in detections:

left, top, right, bottom = bbox2points(bbox)

left, top, right, bottom = int(left \* width\_ratio), int(top \* height\_ratio), int(right \* width\_ratio), int(bottom \* height\_ratio)

cv2.rectangle(img, (left, top), (right, bottom), class\_colors[label], 2)

cv2.putText(img, "{} [{:.2f}]".format(label, float(confidence)),

(left, top - 5), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5,

class\_colors[label], 2)

cv2.imwrite(filename, img)

return filename

try:

filename = take\_photo('photo.jpg')

print('Saved to {}'.format(filename))

display(Image(filename))

except Exception as err:

print(str(err))

def video\_stream():

js = Javascript('''

var video;

var div = null;

var stream;

var captureCanvas;

var imgElement;

var labelElement;

var pendingResolve = null;

var shutdown = false;

function removeDom() {

stream.getVideoTracks()[0].stop();

video.remove();

div.remove();

video = null;

div = null;

stream = null;

imgElement = null;

captureCanvas = null;

labelElement = null;

}

function onAnimationFrame() {

if (!shutdown) {

window.requestAnimationFrame(onAnimationFrame);

}

if (pendingResolve) {

var result = "";

if (!shutdown) {

captureCanvas.getContext('2d').drawImage(video, 0, 0, 640, 480);

result = captureCanvas.toDataURL('image/jpeg', 0.8)

}

var lp = pendingResolve;

pendingResolve = null;

lp(result);

}

}

async function createDom() {

if (div !== null) {

return stream;

}

div = document.createElement('div');

div.style.border = '2px solid black';

div.style.padding = '3px';

div.style.width = '100%';

div.style.maxWidth = '600px';

document.body.appendChild(div);

const modelOut = document.createElement('div');

modelOut.innerHTML = "<span>Status:</span>";

labelElement = document.createElement('span');

labelElement.innerText = 'No data';

labelElement.style.fontWeight = 'bold';

modelOut.appendChild(labelElement);

div.appendChild(modelOut);

video = document.createElement('video');

video.style.display = 'block';

video.width = div.clientWidth - 6;

video.setAttribute('playsinline', '');

video.onclick = () => { shutdown = true; };

stream = await navigator.mediaDevices.getUserMedia(

{video: { facingMode: "environment"}});

div.appendChild(video);

imgElement = document.createElement('img');

imgElement.style.position = 'absolute';

imgElement.style.zIndex = 1;

imgElement.onclick = () => { shutdown = true; };

div.appendChild(imgElement);

const instruction = document.createElement('div');

instruction.innerHTML =

'<span style="color: red; font-weight: bold;">' +

'When finished, click here or on the video to stop this demo</span>';

div.appendChild(instruction);

instruction.onclick = () => { shutdown = true; };

video.srcObject = stream;

await video.play();

captureCanvas = document.createElement('canvas');

captureCanvas.width = 640; //video.videoWidth;

captureCanvas.height = 480; //video.videoHeight;

window.requestAnimationFrame(onAnimationFrame);

return stream;

}

async function stream\_frame(label, imgData) {

if (shutdown) {

removeDom();

shutdown = false;

return '';

}

var preCreate = Date.now();

stream = await createDom();

var preShow = Date.now();

if (label != "") {

labelElement.innerHTML = label;

}

if (imgData != "") {

var videoRect = video.getClientRects()[0];

imgElement.style.top = videoRect.top + "px";

imgElement.style.left = videoRect.left + "px";

imgElement.style.width = videoRect.width + "px";

imgElement.style.height = videoRect.height + "px";

imgElement.src = imgData;

}

var preCapture = Date.now();

var result = await new Promise(function(resolve, reject) {

pendingResolve = resolve;

});

shutdown = false;

return {'create': preShow - preCreate,

'show': preCapture - preShow,

'capture': Date.now() - preCapture,

'img': result};

}

''')

display(js)

def video\_frame(label, bbox):

data = eval\_js('stream\_frame("{}", "{}")'.format(label, bbox))

return data

video\_stream()

label\_html = 'Capturing...'

bbox = ''

count = 0

while True:

js\_reply = video\_frame(label\_html, bbox)

if not js\_reply:

break

frame = js\_to\_image(js\_reply["img"])

bbox\_array = np.zeros([480,640,4], dtype=np.uint8)

detections, width\_ratio, height\_ratio = darknet\_helper(frame, width, height)

for label, confidence, bbox in detections:

left, top, right, bottom = bbox2points(bbox)

left, top, right, bottom = int(left \* width\_ratio), int(top \* height\_ratio), int(right \* width\_ratio), int(bottom \* height\_ratio)

bbox\_array = cv2.rectangle(bbox\_array, (left, top), (right, bottom), class\_colors[label], 2)

bbox\_array = cv2.putText(bbox\_array, "{} [{:.2f}]".format(label, float(confidence)),

(left, top - 5), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5,

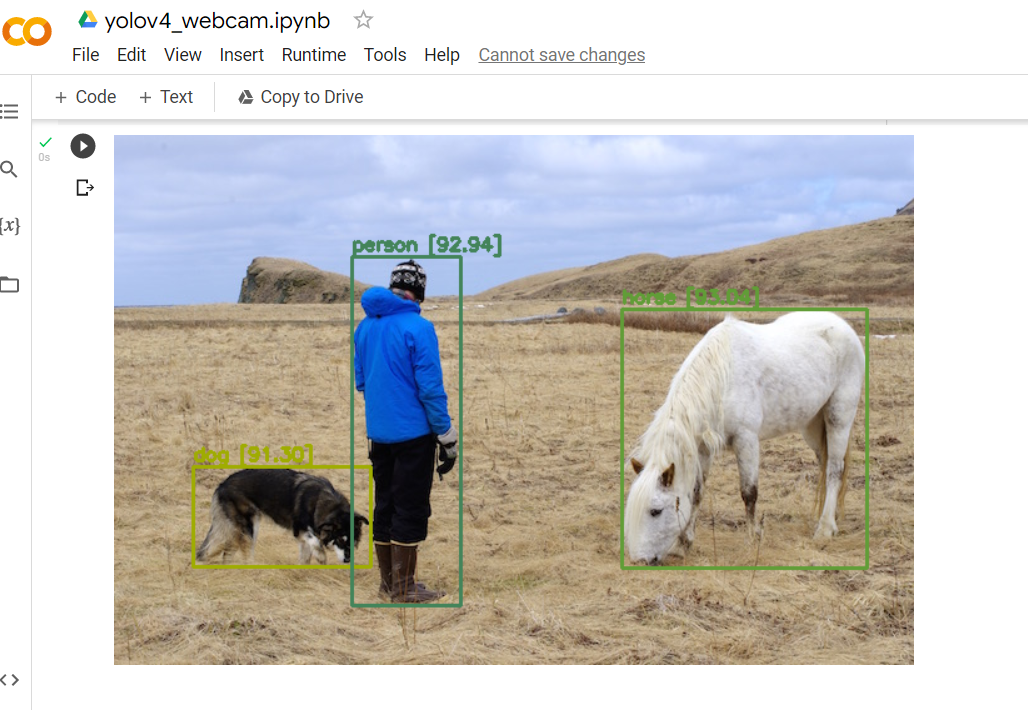
class\_colors[label], 2)

bbox\_array[:,:,3] = (bbox\_array.max(axis = 2) > 0 ).astype(int) \* 255

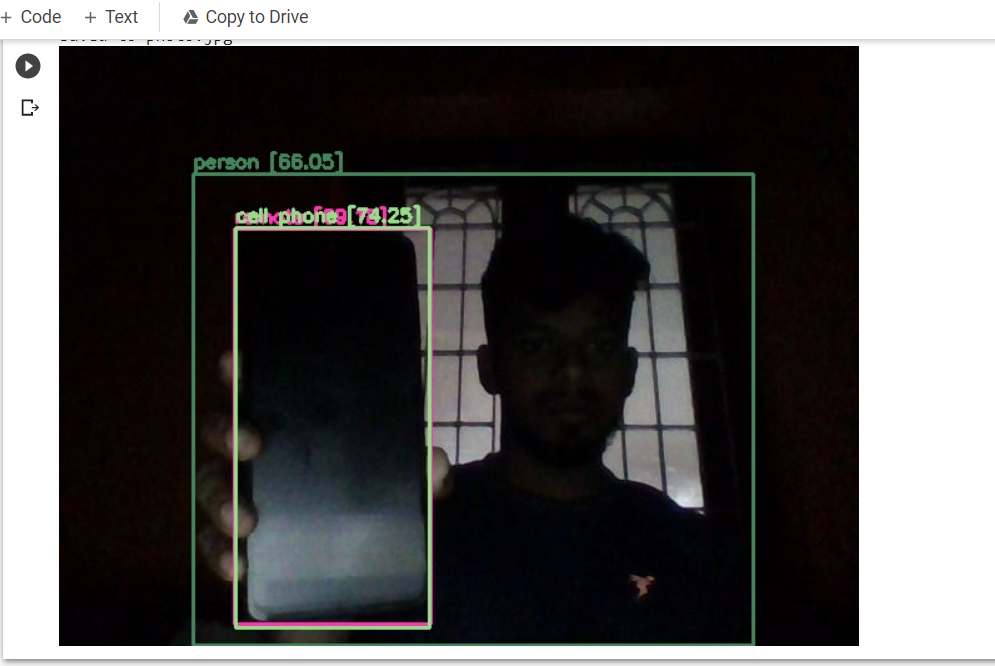
bbox\_bytes = bbox\_to\_bytes(bbox\_array)

bbox = bbox\_bytes

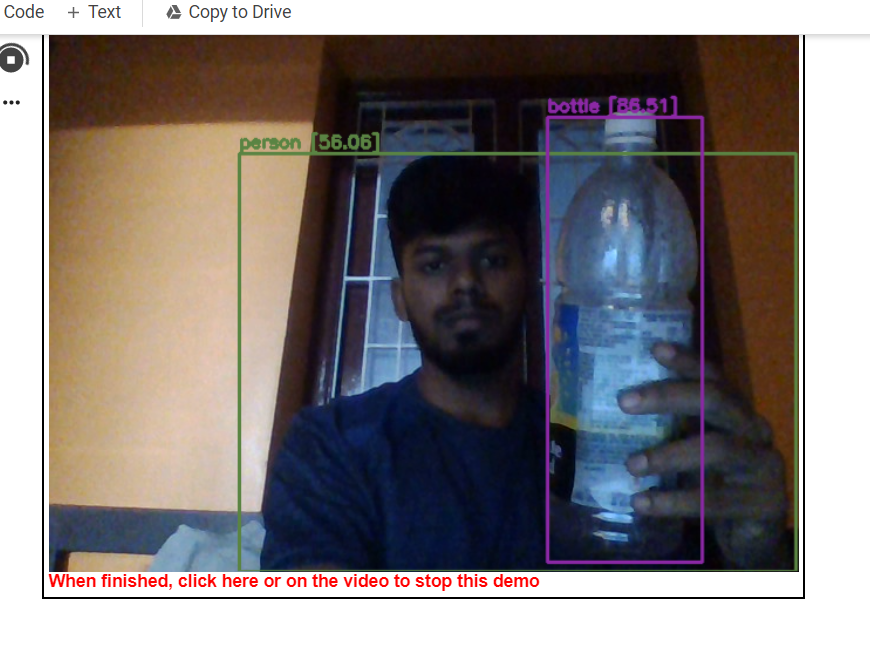
**SAMPLE OUTPUT**

****

**1. saved Image**

****

**2.captured object detection**

****