import imutils

import numpy as np

import cv2

from google.colab.patches import cv2\_imshow

from IPython.display import display, Javascript

from google.colab.output import eval\_js

from base64 import b64decode

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))

  binary = b64decode(data.split(',')[1])

  with open(filename, 'wb') as f:

    f.write(binary)

  return filename

image\_file = take\_photo()

#image = cv2.imread(image\_file, cv2.IMREAD\_UNCHANGED)

image = cv2.imread(image\_file)

# resize it to have a maximum width of 400 pixels

image = imutils.resize(image, width=400)

(h, w) = image.shape[:2]

print(w,h)

cv2\_imshow(image)

!wget -N https://raw.githubusercontent.com/opencv/opencv/master/samples/dnn/face\_detector/deploy.prototxt

!wget -N https://raw.githubusercontent.com/opencv/opencv\_3rdparty/dnn\_samples\_face\_detector\_20170830/res10\_300x300\_ssd\_iter\_140000.caffemodel

print("[INFO] loading model...")

prototxt = 'deploy.prototxt'

model = 'res10\_300x300\_ssd\_iter\_140000.caffemodel'

net = cv2.dnn.readNetFromCaffe(prototxt, model)

# resize it to have a maximum width of 400 pixels

image = imutils.resize(image, width=400)

blob = cv2.dnn.blobFromImage(cv2.resize(image, (300, 300)), 1.0, (300, 300), (104.0, 177.0, 123.0))

print("[INFO] computing object detections...")

net.setInput(blob)

detections = net.forward()

for i in range(0, detections.shape[2]):

  # extract the confidence (i.e., probability) associated with the prediction

  confidence = detections[0, 0, i, 2]

  # filter out weak detections by ensuring the `confidence` is

  # greater than the minimum confidence threshold

  if confidence > 0.5:

    # compute the (x, y)-coordinates of the bounding box for the object

    box = detections[0, 0, i, 3:7] \* np.array([w, h, w, h])

    (startX, startY, endX, endY) = box.astype("int")

    # draw the bounding box of the face along with the associated probability

    text = "{:.2f}%".format(confidence \* 100)

    y = startY - 10 if startY - 10 > 10 else startY + 10

    cv2.rectangle(image, (startX, startY), (endX, endY), (0, 0, 255), 2)

    cv2.putText(image, text, (startX, y),

      cv2.FONT\_HERSHEY\_SIMPLEX, 0.45, (0, 0, 255), 2)

cv2\_imshow(image)