Face Recognition Model By NI_AI_Suprith Shettigar

```
#importing the libraries:
import cv2 #used for powerful library for computer vision and image processing tasks. ( like :  face detection and image filtering )
import numpy as np #used for basic array operations and numerical data handling.
from IPython.display import display, Javascript, Image #used to display the output of the camera.
from google.colab.output import eval_js #used for taking the photo and storing the information,
from base64 import b64decode, b64encode #used for decoding and checking the format of the image.
import PIL #used for creating a PIL image from the binary data.
import io # This provides classes and functions for working with input and output operations.
import html # The Python Standard Library and provides functions for working with HTML-encoded text. You can use this module for 
import time # This provides functions for working with time-related operations. You can use this module to perform tasks such as measu
```

```
# function to convert the JavaScript object into an OpenCV image
def js_to_image(js_reply):
  # decode base64 image
  image_bytes = b64decode(js_reply.split(',')[1])
  # convert bytes to numpy array
  jpg_as_np = np.frombuffer(image_bytes, dtype=np.uint8)
  # decode numpy array into OpenCV BGR image
  img = cv2.imdecode(jpg_as_np, flags=1)

  return img
```

```
# function to convert OpenCV Rectangle bounding box image into base64 byte string to be overlayed on video stream
def bbox_to_bytes(bbox_array):
  #bbox_array: Numpy array (pixels) containing rectangle to overlay on video stream.
  # convert array into PIL image
  bbox_PIL = PIL.Image.fromarray(bbox_array, 'RGBA')
  iobuf = io.BytesIO()
  # format bbox into png for return
  bbox_PIL.save(iobuf, format='png')
  # format return string
  bbox_bytes = 'data:image/png;base64,{}'.format((str(b64encode(iobuf.getvalue()), 'utf-8')))

  #Returns { bytes: Base64 image byte string }
  return bbox_bytes
```

```
# initialize the Haar Cascade face detection model
face_cascade = cv2.CascadeClassifier(cv2.samples.findFile(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml'))
```

```
# JavaScript to properly create our live video stream using our webcam as input
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
```

```
# start streaming video from webcam
video_stream()
# label for video
label_html = 'Capturing...'
# initialze bounding box to empty
bbox = ''
count = 0
while True:
    js_reply = video_frame(label_html, bbox)
    if not js_reply:
        break

    # convert JS response to OpenCV Image
    img = js_to_image(js_reply["img"])

    # create transparent overlay for bounding box
    bbox_array = np.zeros([480,640,4], dtype=np.uint8)

    # grayscale image for face detection
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)

    # get face region coordinates
    faces = face_cascade.detectMultiScale(gray)
    # get face bounding box for overlay
    for (x,y,w,h) in faces:
      bbox_array = cv2.rectangle(bbox_array,(x,y),(x+w,y+h),(0,0,255),2)

    bbox_array[:,:,3] = (bbox_array.max(axis = 2) > 0 ).astype(int) * 255
    # convert overlay of bbox into bytes
    bbox_bytes = bbox_to_bytes(bbox_array)
    # update bbox so next frame gets new overlay
    bbox = bbox_bytes
```

...

Status:**Capturing**