IMAGE CAPTURE AND FACIAL SEGMENTATION

Abdullah Faisal Chaudhry

ABSTRACT

This program is a python-written code which consists of various functions, libraries and modules. The purpose of the code is to capture live images from a webcam, process the captured images via a pre-trained neural network, and perform facial segmentation on said images. The outputted images are then further cropped and resized acording to arbitrary specifications, before being inputted into a circular buffer, which is used to regulate memory usage on the host machine. Any other secondary program may then request an image for use from the circular buffer. Upon request, this program will also then dequeue the requested image and assign it to a temporary holding variable, available for the secondary program to utilize. This sequence of actions may be repeated an unlimited number of times, or as many times as required.

MODULES & LIBRARIES

- OpenCV: OpenCV is a library of programming functions mainly aimed at real-time computer vision. Used in the application of this program to capture live images from webcam of computer.
- MediaPipe: MediaPipe is a pre-trained, ultrafast face detection solution that comes with 6 landmarks and multi-face support. It is based on BlazeFace, a lightweight and well-performing face detector tailored for mobile GPU inference. The detector's real-time performance enables it to be applied to any live viewfinder experience that requires an accurate facial region of interest as an input for other task-specific models, such face region segmentation.
- OS: The OS module provides a quick and portable method to utilizing system dependant functionality. In the application of this program, the module is used to define file paths and specify directory changes.
- Time: This python module provides various basic time related functions. In the application of this program, the module is used to regulate the circular buffer enqueuement.

SEGMENT 1: DEFINING GLOBAL VARIABLES

```
8   ####################
9   iter = 0
10  Y = 0
11  H = 0
12  X = 0
13  W = 0
14  capacity = 119
15  timcap = 1
16  ####################
```

- In this segment of the code, the global variables, required for the standard operation of the rest of the code, are initialized and defined.
- iter – corresponds to the iteration of the while loop performing facial segmentation.
- Y, H, X, W – correspond to numerical values of X, Y positions of facial segmentation bounding box. H, W correspond to height and width of bounding box.
- capacity – corresponds to variable capacity of circular buffer holding processed images.
- timcap – corresponds to flag variable used in the operation of buffer enqueuement regulation.

SEGMENT 2: DEFINING CIRCULAR BUFFER

- The structure of the circular buffer functionality is defined in this segment of the code.
- The initiation function (__init__) specifies the capacity of the buffer, as well as the starting positions of the read and write pointers.
- The Enqueue and Dequeue functions specify how an image is to enter and exit the circular buffer upon request and the subsequent actions of the buffer.
- For debugging or admin purposes, the display function is able to print the active/live capacity of the buffer during operation.

```
16  ##################################################################################
17  class circularQueue:
18      def __init__ (self, capacity):
19          self.capacity = capacity
20          self.queue = [None] * capacity
21          self.tail = -1
22          self.head = 0
23          self.size = 0
24
25      def Enqueue(self, item):
26          if self.size == self.capacity:
27              print("Error : Queue is Full")
28          else:
29              self.tail = (self.tail + 1) % self.capacity
30              self.queue[self.tail] = item
31              self.size = self.size + 1
32
33      def Dequeue(self):
34          if self.size == 0:
35              print ("Error : Queue is Empty")
36              return
37          else:
38              tmp = self.queue[self.head]
39              self.head = (self.head + 1) % self.capacity
40              self.size = self.size - 1
41              return tmp
42
43      def display(self):
44          if self.size == 0:
45              print ("Queue is Empty \n")
46          else:
47              index = self.head
48
49              for i in range(self.size):
50                  print(self.queue[index])
51                  index = (index + 1) % self.capacity
52  ##################################################################################
```

SEGMENT 3: IMAGE CAPTURE INITIATION

- In this segment of the program, the facial segmentation model is initiated. The confidence parameter of the model is configured here, as well as the parameter which controls the focus distance of the face detection model.
- The while loop for the continuous operation of the program is also initiated.
- In the initial few lines under the while loop, the timcap flag is checked by an if-elif statement, which is used to regulate the frequency at which an image is enqueued into the circular buffer.

```
61  with mp_face_detection.FaceDetection(
62      model_selection=0, min_detection_confidence=0.5) as face_detection:
63    while cap.isOpened():
64      if timcap == 1:
65          start = time.time()
66      elif timcap == 0:
67          start=start
68      success, image = cap.read()
69      if not success:
70        print("Ignoring empty camera frame.")
71        # If loading a video, use 'break' instead of 'continue'.
72        continue
```

SEGMENT 4: FACIAL SEGMENTATION

- Before the image is segmented based on the facial features, the image must be converted from the native RGB color mode to BGR color mode, as the model used was trained using the BGR color mode. This is done by the OpenCV cv2.cvtColor function.
- Once the image has been processed, it is converted back to the RGB color mode to display on the host machine.
- The bounding box based the on the facial segmentation data is then drawn on the image using the mp_drawing.draw_detection function.

```
    # To improve performance, optionally mark the ima
    # pass by reference.
    image.flags.writeable = False
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    results = face_detection.process(image)
##########################################################
    # Draw the face detection annotations on the imag
    image.flags.writeable = True
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

    if results.detections:
      print("hello faisal \n")
      if noface_flag == 1:
          cv2.destroyWindow("empty image")
          noface_flag = 0
      for detection in results.detections:
        mp_drawing.draw_detection(image, detection)
```

SEGMENT 5: IMAGE CROP

- This segment of the code is responsible for obtaining the X, Y positions and the Height and Width of the segmentation bounding box.
- These values are then normalized based on the resolution of the host camera.
- It is important to note that certain boundary conditions are applied to these values to ensure that the cropping operation does not encounter logic errors in the event that a face leaves or enters the frame during operation.

```
93      # collect boundin box location data
94          location_data = detection.location_data
95          if location_data.format == location_data.RELATIVE_BOUNDING_BOX:
96              bb = location_data.relative_bounding_box
97              bb_box = [bb.xmin, bb.ymin, bb.width, bb.height]
98              #print(f"RBBox: {bb_box}")
99  ##########################################################################
100     #assign y h x w
101         Y = int(720*bb_box[1])
102         H = int(720*bb_box[3])
103         X = int(1280*bb_box[0])
104         W = int(1280*bb_box[2])
105
106         if Y < 0:
107             Y = 0
108         if Y > 720:
109             Y = 720
110
111         if H < 0:
112             H = 0
113         if H > 720:
114             H = 720
115
116         if X < 0:
117             X = 0
118         if X > 1280:
119             X = 1280
120
121         if W < 0:
122             W = 0
123         if W > 1280:
```

SEGMENT 6: CIRCULAR BUFFER ENQUE/DEQUE

- The final segment of this program pertains to the enqueuement and dequeuement of the cropped images into the circular buffer upon request.
- It should be noted that the frequency of the write action is controlled by the elapstime variable, which currently intends to control the write frequency to 2 images per second.
- Upon request by any secondary program, the buffer will read the respective image and then assign it to a temporary holding variable called outimg, from which the secondary program will be able to use the image.

```
147             if elapstime > 0.5:
148                 timcap = 1
149                 Q.Enqueue(ready_img)
150                 print("enqued")
151                 iter += 1
152                 #Q.display()
153                 if iter < 10:
154                     continue
155                 else:
156                     outimg = Q.Dequeue()
157                     #directory = r'/Users/Azzaam/Desktop/MASTER/UNI/RESE
158                     #os.chdir(directory)
159                     #print(os.listdir(directory))
160                     #filename = 'savedImage%d.jpg' %iter
161                     #cv2.imwrite(filename, outimg)
162                     continue
163             else:
164                 timcap = 0
165                 continue
166 ##########################################################################
167         else:
168             print("hello joseph\n")
169             cv2.imshow("empty image", image)
170             #time.sleep(5)
171             noface_flag = 1
172             continue
```