WINTER IN DATA SCIENCE – MID TERM REPORT AUGMENTED REALITY FACE TRACKING AND OVERLAY SYSTEM

Nitin Bamnawath

25B3936

Table of Contents

Week 0: Python Basics + Introduction to Computer Vision

In week 0 we were asked to create a basic image editor, it was with help of python libraries such as pillow, OpenCV and used matplotlib for final output.

Initially file was opened with help of pillow

`from PIL import Image` : Importing its library

`Image.open("image.jpg")` : Importing image

1. Convert RGB to BGR
   With the help of OpenCV we will convert image format from RGB to BGR
   Note: (OpenCV handles images in BGR format by default, whereas Pillow opens them in RGB, but since we are using matplotlib for image output which supports RGB by default it doesn't pose a problem)
   ```
   cv2.cvtColor(imagecv2, cv2.COLOR_RGB2BGR)
   ```

2. Resize
   ```
   cv2.resize(imagecv2,(x,y))
   ```

3. Sharpen
   ```
   kernel_sharpening = np.array([[-1,-1,-1],[-1,9,-1],[-1,-1,-1]])
   finalimage = cv2.filter2D(imagecv2, -1, kernel_sharpening)
   ```

Finally image output by matplotlib.pyplot

```
plt.imshow(finalimage)
```

Also, finally saving image as

```
plt.imsave("final_output.jpg", finalimage)
```

Week 1: Background Removal + Overlaying on Static Images

The goal for Week 1 was to implement background removal on static images without relying on external removal tools. I utilized the GrabCut Algorithm, a simpler alternative to Deep Learning Semantic Segmentation, to extracting a foreground object and overlaying it onto a new background.

Used the grabcut algorithm by OpenCV
```
cv2.grabCut(img,mask,rect,bgdModel,fgdModel,5,cv2.GC_INIT_WITH_RECT)
```

#parameter(img, mask, rect, bgmodel, fgmodel, iter, method)

By providing a bounding box (rect), we define the area we want to work with. This separates the image from its probable background. We initialized the background and foreground models (bgdModel, fgdModel) as zero arrays, which the algorithm uses internally to learn the color distribution during the 5 iterations.

Overlay Logic After generating the mask, I inverted it to create a "hollowed out" space in the background image and filled it with the foreground object. This creates a seamless cutout effect.

Then by using this we got a mask

```
img_backcut = img_back[coordinates_y : coordinates_y + y , coordinates_x :
coordinates_x + x]
img_backcut = img_backcut * ( 1 - mask_resize[:,:,np.newaxis] ) + img_resize
```
"Since our image has 3 channels (Blue, Green, Red) but our mask only has 1 channel (Grayscale), we use np.newaxis to add a third dimension to the mask. This allows us to multiply the 3D image array by the mask without shape errors."

`img_backcut * (1 - mask): This` *turns the pixels black (0) where the Pepsi logo will go. It "punches a hole" in the background.*

`+ img_resize`*: This fills that black hole with the Pepsi logo pixels.*

Week 2: Facial Landmarks Detection (no webcam)

For Week 2, the project advanced from static images to real-time video processing. I developed an Augmented Reality (AR) application that tracks a user's face and overlays the image like in Week 1 onto the nose in real-time.

Facial Landmark Detection I utilized MediaPipe Face Mesh, which maps 468 distinct 3D landmarks on the user's face.

`facial_landmarks.landmark[4]` *#nose land mark*

Nose Tip (Index 4): Used as the anchor point for the center of the overlay.

Cheeks (Indices 234 & 454): Used to calculate the width of the face.

Nose Tip (Index 4): Used as the anchor point for the center of the overlay. Cheeks (Indices 234 & 454): Used to calculate the width of the face.

Dynamic Scaling Logic Instead of a static size, the overlay resizes dynamically as the user moves closer or further from the camera.

```
# Calculate real-time face width
face_width = right_cheek_x - left_cheek_x

# Maintain Aspect Ratio for the overlay height
face_height = int(face_width * (original_overlay_height /
original_overlay_width))
```