OPENCV - COMPUTER VISION II

PROJECT 1: VIRTUAL MAKEUP

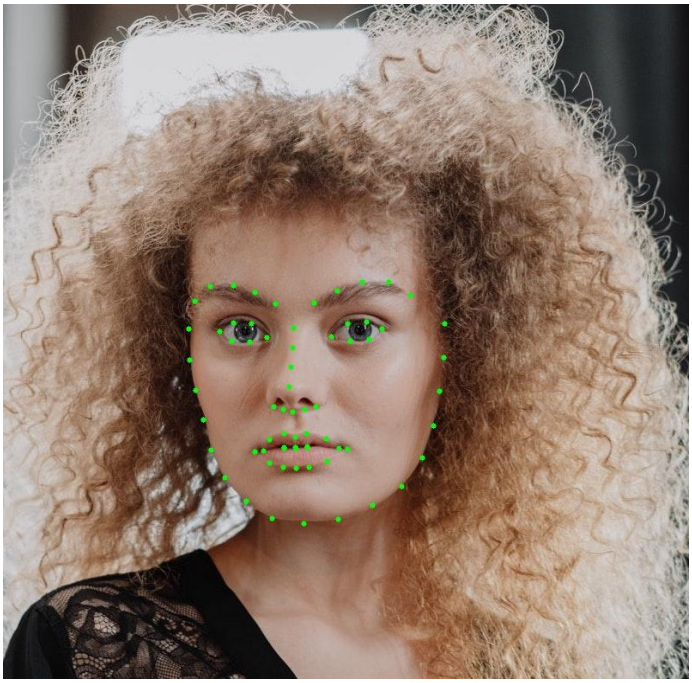
A. Feature 1: Apply Lipstick

This feature applies lipstick color to faces detected on an image. It relies on Dlib Facial Landmark Detection to find the lip region, then performs masking and linear blending to add color to the lips region of the original image. Here is a step-by-step process of implementing this feature:

1. Detecting facial landmarks with Dlib. For this step, I created _get_facial_landmarks function, which takes a single image and returns facial landmarks of detected faces.

```
    def _get_facial_landmarks(self, image, return_numpy_landmark=False):
        # Detect Faces
        face_rects = self.face_detector(image, 0)
        all_facial_landmarks = []
        face_bboxes = []
        # Generate facial landmarks for detected faces
        for face in face_rects:
            face_bbox = (int(face.left()),
                         int(face.top()),
                         int(face.right()),
                         int(face.bottom()))
            # Get dlib bounding box for each face
            rect = dlib.rectangle(face_bbox[0], face_bbox[1], face_bbox[2], face_bbox[3])
            # Find facial landmarks
            landmarks = self.landmark_detector(image, rect)
            if return_numpy_landmark:
                # Generate facial landmarks as numpy array
                temp_list = []
                for i in range(68):
                    x, y = landmarks.part(i).x, landmarks.part(i).y
                    temp_list.append([x,y])
                all_facial_landmarks.append(np.array(temp_list))
            else:
                all_facial_landmarks.append(landmarks)
            face_bboxes.append(face_bbox)
        return all_facial_landmarks, face_bboxes
```

visualize output:



2. Extract lips region. For this step, I created _extract_lips_region function, which takes a single image, facial landmarks and generates a mask for the lips region. The lips region consists of landmark points from #48 to #60.

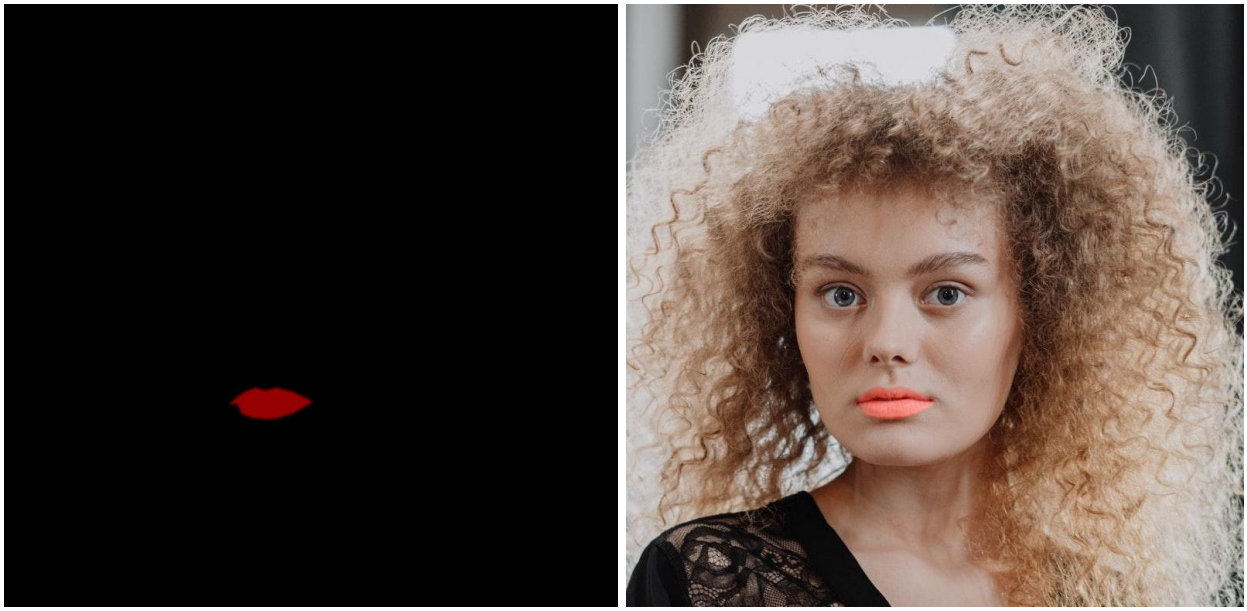
```
def _extract_lips_region(self, image, all_landmarks):
    mask = np.zeros_like(image)
    for landmarks in all_landmarks:
        mask = cv2.fillPoly(mask, [landmarks[48:61]], (255,255,255))
    return mask
```

visualize output:

3. Apply linear color blending to the image on the masked region. applyLipsColor is actually the main function that performs the previous two steps, then creates a colored mask and applies linear blending (via cv2.addWeighted) to generate the final output. Within this step, there are two small steps that are noteworthy:
    a. Apply Gaussian Blurring to the colored mask to massage its rough edges. (via cv2.GaussianBlur)
    b. Apply Linear Blending with beta fixed at 0.4 (via cv2.addWeighted)

```
def applyLipsColor(self, image, color, all_landmarks=None):
    if all_landmarks is None:
        all_landmarks, _ = self._get_facial_landmarks(image, return_numpy_landmark=True)
    lips_mask = self._extract_lips_region(image, all_landmarks)
    color_lips = np.zeros_like(image)
    color_lips[:] = color[0], color[1], color[2]
    color_lips = cv2.bitwise_and(lips_mask, color_lips)
    color_lips = cv2.GaussianBlur(color_lips, (5,5), 10)
    output_img = cv2.addWeighted(image, 1, color_lips, 0.4,0)
    return output_img
```

Visualize output:



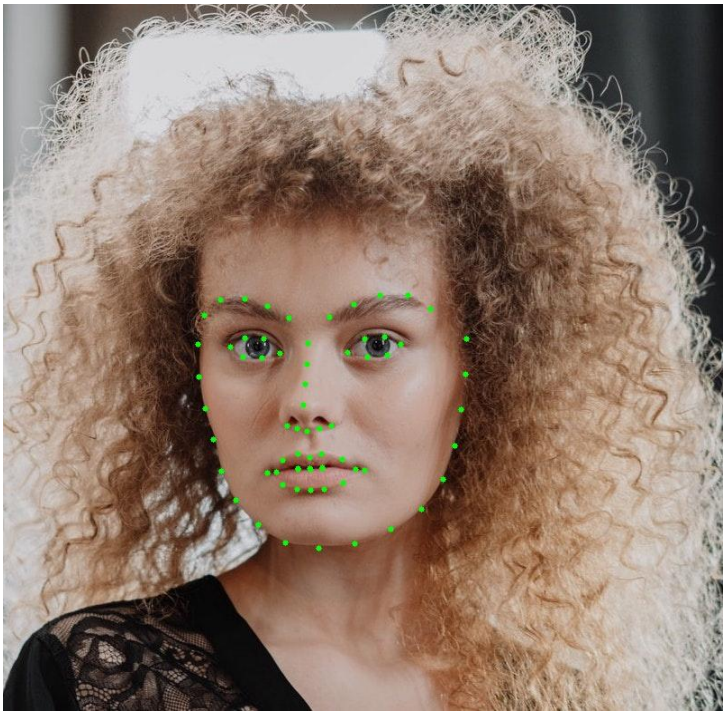
B. Feature 2: Apply Eye Color

This feature applies eye color (lens color) to faces detected on an image. It relies on Dlib Facial Landmark Detection to find the lip region, then performs masking and linear blending to add color to the eyes region of the original image. While it has the similar approach as feature 1, some extra steps are needed to extract only the pupil region in this image. Here is a step-by-step process of implementing this feature:

1. Detecting facial landmarks with Dlib. For this step, I created _get_facial_landmarks function, which takes a single image and returns facial landmarks of detected faces. This step uses the same function as in feature 1.

```python
def _get_facial_landmarks(self, image, return_numpy_landmark=False):
    # Detect Faces
    face_rects = self.face_detector(image, 0)
    all_facial_landmarks = []
    face_bboxes = []
    # Generate facial landmarks for detected faces
    for face in face_rects:
        face_bbox = (int(face.left()),
                     int(face.top()),
                     int(face.right()),
                     int(face.bottom()))
        # Get dlib bounding box for each face
        rect = dlib.rectangle(face_bbox[0], face_bbox[1], face_bbox[2], face_bbox[3])
        # Find facial landmarks
        landmarks = self.landmark_detector(image, rect)
        if return_numpy_landmark:
            # Generate facial landmarks as numpy array
            temp_list = []
            for i in range(68):
                x, y = landmarks.part(i).x, landmarks.part(i).y
                temp_list.append([x,y])
            all_facial_landmarks.append(np.array(temp_list))
        else:
            all_facial_landmarks.append(landmarks)
        face_bboxes.append(face_bbox)
    return all_facial_landmarks, face_bboxes
```

visualize output:

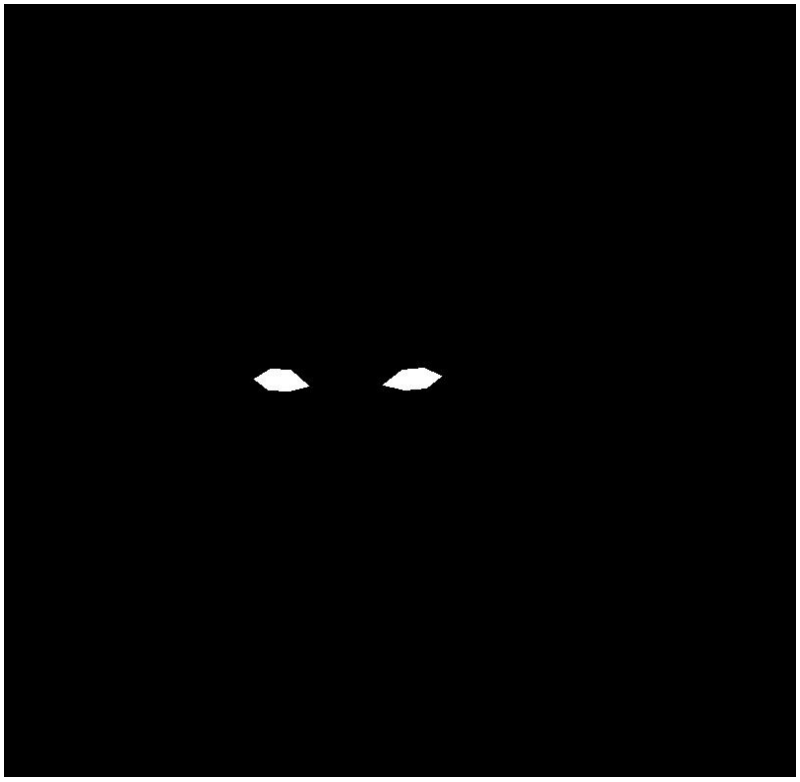


2. Extract pupil region. For this step, I created _extract_pupil_region function, which takes a single image, facial landmarks and generates a mask for the lips region. The left eye region consists of landmark points from #36 to #41, the right eye region consists of landmark points from #42 to #47.

```
def _extract_pupils_region(self, image, all_landmarks):
        # Create mask for eyes region
        eyes_mask = np.zeros_like(image)
        for landmarks in all_landmarks:
            eyes_mask = cv2.fillPoly(eyes_mask, [landmarks[36:42]], (255,255,255)) # left eye
            eyes_mask = cv2.fillPoly(eyes_mask, [landmarks[42:48]], (255,255,255)) # right eye
        # Crop eyes region of original image
        cropped_image = cv2.bitwise_and(image, eyes_mask)
        # Apply binary threshold to extract pupils region
        ret, eyes_bright_mask = cv2.threshold(cropped_image,80,255,cv2.THRESH_BINARY)
        eyes_bright_mask_binary = np.bitwise_or.reduce(eyes_bright_mask, axis=-1)
        eyes_bright_mask = cv2.cvtColor(eyes_bright_mask_binary, cv2.COLOR_GRAY2RGB)
        pupils_mask = cv2.bitwise_xor(eyes_mask, eyes_bright_mask)
        # Apply Closing & Opening to remove noises in pupil mask
        kernel = cv2.getStructuringElement(shape=cv2.MORPH_ELLIPSE, ksize=(5,5))
        pupils_mask = cv2.morphologyEx(pupils_mask, cv2.MORPH_CLOSE, kernel)
        pupils_mask = cv2.morphologyEx(pupils_mask, cv2.MORPH_OPEN, kernel)
        return pupils_mask
```

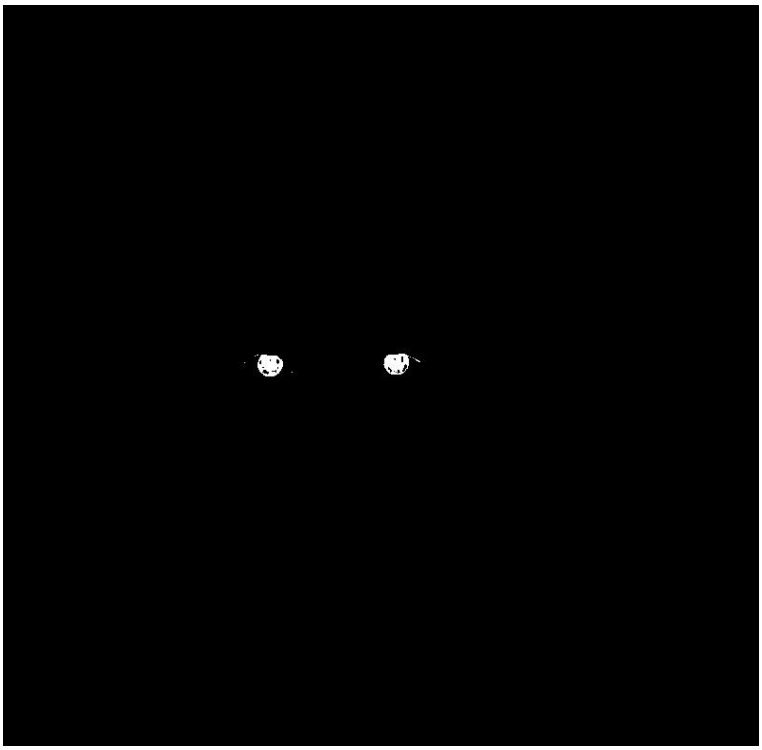
In this code, there are a few mini steps that are taken as follow:

a. Extract eyes region based on facial landmark

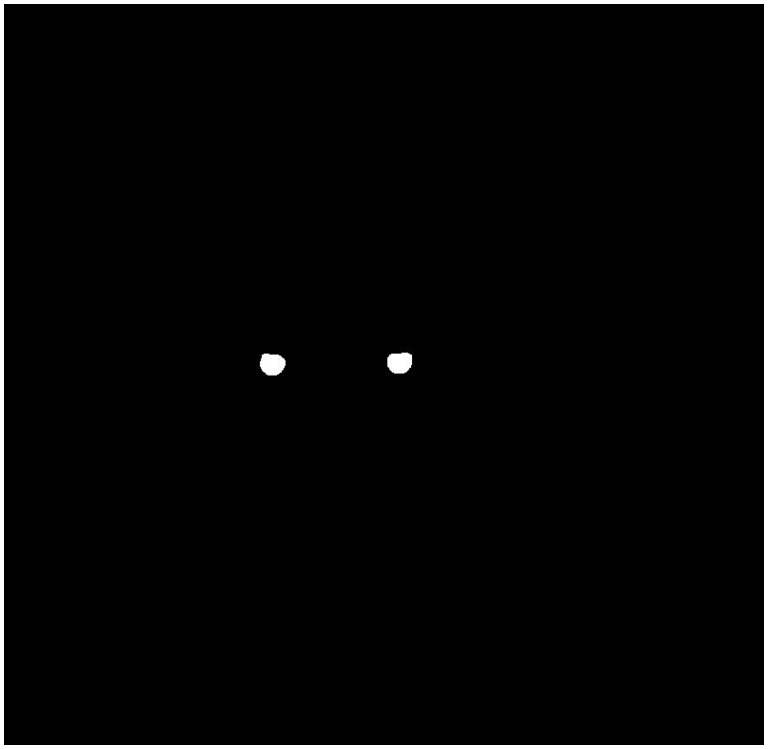


b. Crop eyes region in the original image

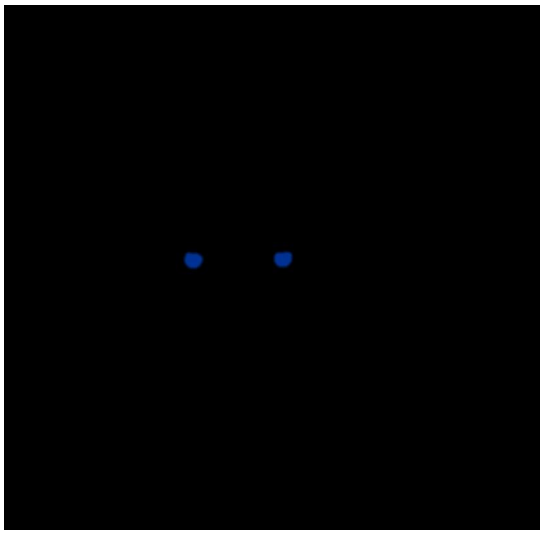
c. Extract pupil region using color thresholding. I extract the bright region in the eyes and perform XOR operation with the mask of the whole eye, this results in a new mask for only the dark region.



d. Apply closing & opening on the pupil mask to reduce the noises from applying color thresholding in the previous step.



4. Apply linear color blending to the image on the masked region. applyEyesColor is actually the main function that performs the previous two steps, then creates a colored mask and applies linear blending (via cv2.addWeighted) to generate the final output. Within this step, there are two small steps that are noteworthy:
    a. Apply Gaussian Blurring to the colored mask to massage its rough edges. (via cv2.GaussianBlur)
    b. Apply Linear Blending with beta fixed at 0.4 (via cv2.addWeighted)



NOTE: I created a simple class named FaceMakeup which provides both features. Function applyColor can be used to apply color to both lips & eye lenses. Here is the full implementation for this project:

```
class FaceMakeup:
    def __init__(self, landmark_model_path):
        self.face_detector = dlib.get_frontal_face_detector()
        try:
            self.landmark_detector = dlib.shape_predictor(landmark_model_path)
        except:
            print(f"ERROR: unable to load landmark model in this path: {landmark_model_path}")

    def applyColor(self, image, eyes_color=None, lips_color=None):
        all_landmarks, _ = self._get_facial_landmarks(image, return_numpy_landmark=True)
        output_image = image.copy()
        if lips_color:
            output_image = self.applyLipsColor(output_image, lips_color, all_landmarks)
        if eyes_color:
            output_image = self.applyEyesColor(output_image, eyes_color, all_landmarks)
        return output_image

    def applyLipsColor(self, image, color, all_landmarks=None):
        if all_landmarks is None:
            all_landmarks, _ = self._get_facial_landmarks(image, return_numpy_landmark=True)
        lips_mask = self._extract_lips_region(image, all_landmarks)
        color_lips = np.zeros_like(image)
        color_lips[:] = color[0], color[1], color[2]
        color_lips = cv2.bitwise_and(lips_mask, color_lips)
        color_lips = cv2.GaussianBlur(color_lips, (5,5), 10)
        cv2.imwrite("color_mask.jpg", cv2.cvtColor(color_lips, cv2.COLOR_RGB2BGR))
        output_img = cv2.addWeighted(image, 1, color_lips, 0.4,0)
        return output_img

    def applyEyesColor(self, image, color, all_landmarks=None):
        if all_landmarks is None:
            all_landmarks, _ = self._get_facial_landmarks(image, return_numpy_landmark=True)
        pupils_mask = self._extract_pupils_region(image, all_landmarks)
        color_pupils = np.zeros_like(image)
        color_pupils[:] = color[0], color[1], color[2]
        color_pupils = cv2.bitwise_and(pupils_mask, color_pupils)
        color_pupils = cv2.GaussianBlur(color_pupils, (5,5), 10)
        output_img = cv2.addWeighted(image, 1, color_pupils, 0.4,0)
        return output_img
```

```python
    def _extract_pupils_region(self, image, all_landmarks):
        # Create mask for eyes region
        eyes_mask = np.zeros_like(image)
        for landmarks in all_landmarks:
            eyes_mask = cv2.fillPoly(eyes_mask, [landmarks[36:42]], (255,255,255)) # left eye
            eyes_mask = cv2.fillPoly(eyes_mask, [landmarks[42:48]], (255,255,255)) # right eye
        # Crop eyes region of original image
        cropped_image = cv2.bitwise_and(image, eyes_mask)
        # Apply binary threshold to extract pupils region
        ret, eyes_bright_mask = cv2.threshold(cropped_image,80,255,cv2.THRESH_BINARY)
        eyes_bright_mask_binary = np.bitwise_or.reduce(eyes_bright_mask, axis=-1)
        eyes_bright_mask = cv2.cvtColor(eyes_bright_mask_binary, cv2.COLOR_GRAY2RGB)
        pupils_mask = cv2.bitwise_xor(eyes_mask, eyes_bright_mask)
        # Apply Closing & Opening to remove noises in pupil mask
        kernel = cv2.getStructuringElement(shape=cv2.MORPH_ELLIPSE, ksize=(5,5))
        pupils_mask = cv2.morphologyEx(pupils_mask, cv2.MORPH_CLOSE, kernel)
        pupils_mask = cv2.morphologyEx(pupils_mask, cv2.MORPH_OPEN, kernel)
        return pupils_mask

    def _extract_lips_region(self, image, all_landmarks):
        mask = np.zeros_like(image)
        for landmarks in all_landmarks:
            mask = cv2.fillPoly(mask, [landmarks[48:61]], (255,255,255))
        return mask

    def _get_facial_landmarks(self, image, return_numpy_landmark=True):
        # Detect Faces
        face_rects = self.face_detector(image, 0)
        all_facial_landmarks = []
        face_bboxes = []
        # Generate facial landmarks for detected faces
        for face in face_rects:
            face_bbox = (int(face.left()),
                         int(face.top()),
                         int(face.right()),
                         int(face.bottom()))
            # Get dlib bounding box for each face
            rect = dlib.rectangle(face_bbox[0], face_bbox[1], face_bbox[2], face_bbox[3])
            # Find facial landmarks
            landmarks = self.landmark_detector(image, rect)
            if return_numpy_landmark:
                # Generate facial landmarks as numpy array
                temp_list = []
                for i in range(68):
                    x, y = landmarks.part(i).x, landmarks.part(i).y
                    temp_list.append([x,y])
                all_facial_landmarks.append(np.array(temp_list))
            else:
                all_facial_landmarks.append(landmarks)
            face_bboxes.append(face_bbox)
        return all_facial_landmarks, face_bboxes
```