Multimedia Information Systems

Final Project

- Title: **Face Makeup Application**

- Team member list

Student ID	M11209114	M11209119
Univ. College/ School	國立臺灣科技大學 資訊管理系 碩士班	國立臺灣科技大學 資訊管理系 碩士班
Class	甲組	甲組
Name	汪諭	劉柔辰
Email	M11209114@mail.ntust.edu.tw	M11209119@mail.ntust.edu.tw

Contents

1. Introduction

Nowadays, many people often struggle with appearance anxiety, but achieving a makeup look that suits them can be a time-consuming practice that requires continuous experimentation. If there were a program that could recommend makeup styles and show the results after applying makeup, it could save a lot of trial and error.

Currently, there are many open-source programs simulating makeup, but most of them are trained on European and American faces, and the makeup styles they generate tend to be more in line with Western aesthetics. Therefore, we aim to adjust various aspects of this technology, such as parameters, and feature extraction and makeup tones, to make the system capable of providing makeup styles suitable for Asian faces.

2. Framework

Our system's framework diagram, as depicted in Figure 1, begins with inputting a facial photograph. We use methods from OpenCV to read the image, followed by employing the Histogram of Oriented Gradients (HOG) face detector from dlib. Additionally, the pre-trained shape_predictor_68_face_landmarks model is utilized to detect facial positions and landmarks.

Subsequently, various packages such as PIL, SciPy, and OpenCV are employed to draw features like eyebrows, eyeliner, blush, and lips based on the detected facial landmarks. Finally, eyelashes are added to complete the makeup look.

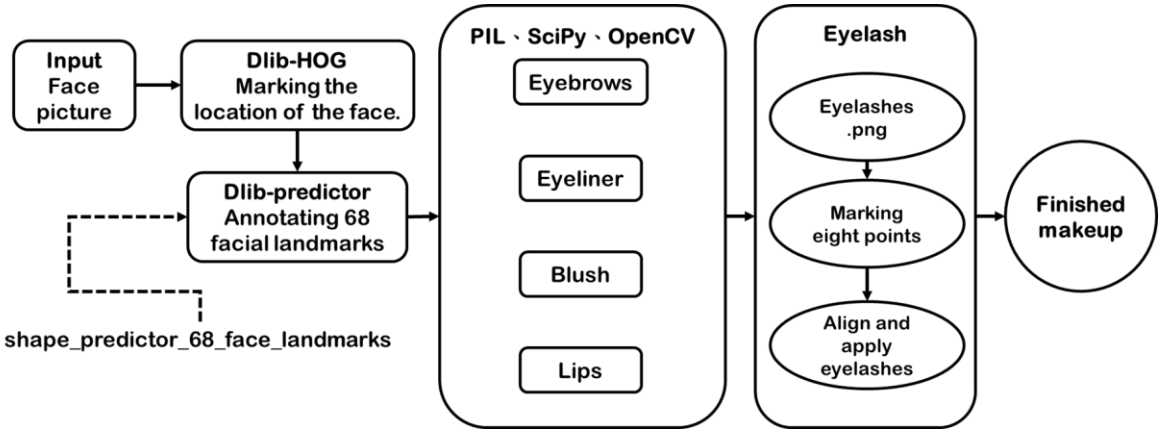


Figure 1. System Framework

3. Method

- **Histogram of Oriented Gradients (HOG)**

  To detect the position of the face using HOG, follow the steps below:

➢ **Step 1: Adjust the size of the image and convert it to grayscale.**

Converting to grayscale helps reduce computational complexity and effectively preserves shapes and information. It aids in extracting facial texture and edges.

➢ **Step 2: Calculate the gradient and direction of each pixel in the image.**

An image is composed of individual pixels, and it can be viewed as a two-dimensional function, denoted as $f(i,j)$. The gradient $G(i,j)$ of the image function $f(i,j)$ at the point (i,j) has both magnitude and direction. Let G_x and G_y represent the gradients in the horizontal x-direction and vertical y-direction, respectively.

Gradient formula for the image:

$$Gx(i,j) = I(i+1,j) - I(i,j) Gx(i,j) = I(i+1,j) - I(i,j)$$

$$Gy(i,j) = I(i+1,j) - I(i,j) Gy(i,j) = I(i+1,j) - I(i,j)$$

$$G(i,j) = Gx(i,j) + Gy(i,j)G(i,j) = Gx(i,j) + Gy(i,j)$$

$$\theta(i,j) = arctan\frac{Gx(i,j)}{Gy(i,j)}\theta(i,j) = arctanGy(i,j)Gx(i,j)$$

* $I(i,j)$represents the pixel value at the point (i,j), and $\theta(i,j)$ denotes the angle of the gradient at the pixel (i,j).

➢ **Step 3: Compute the gradient histograms of cells.**

In each cell, there is a gradient histogram, which is more effective in observing patterns in the image. In OpenCV, a cell typically consists of 8x8 (64) gradients and their corresponding gradient angles. The gradient angles range between 0 and 180 degrees, with positive and negative angles represented by the same numerical values. Based on the direction and strength of the gradients, one can compute the histogram of gradients for each direction.

➢ **Step 4: Normalize the cells.**

As gradients are highly sensitive to the brightness variations in the image, it is crucial to normalize the gradient histograms for each cell to mitigate the impact of illumination changes on features. Initially, the features of each cell and its neighboring cells (2x2) are combined to form larger blocks, increasing in size from 8x8 to 16x16. This allows for capturing more local features. Additionally, the gradient histograms for each block are normalized to alleviate the effects of illumination changes. Finally, all the gradient histogram vectors from the blocks are merged to create the ultimate HOG (Histogram of Oriented Gradients) feature vector, reflecting the local texture and shape information of the entire image.

➢ **Step 5: Draw the bounding box of the face.**

Once HOG detects the position of a face, the coordinates of the top-left and bottom-right corners are used to represent the location of the face. The result is illustrated in Figure 2。

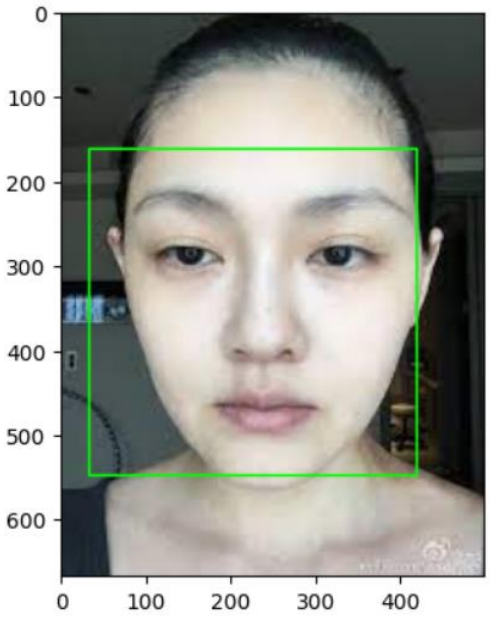
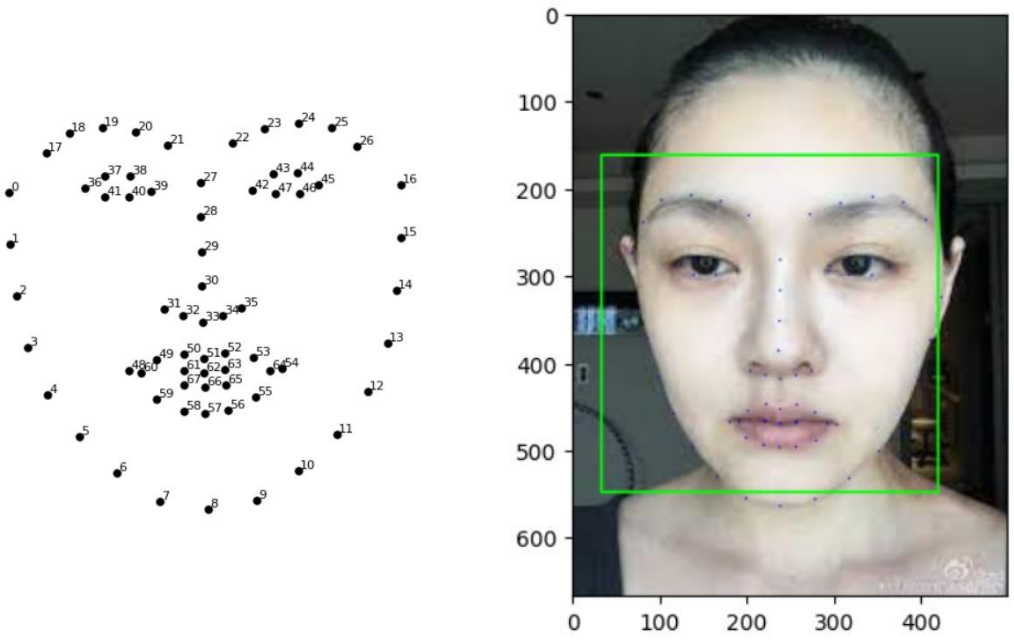


Figure 2. The position of the face

- **Face landmark estimation**

  Utilizing the pre-trained "shape_predictor_68_face_landmarks.dat" shape predictor, the system predicts the positions of specific points on the face, such as eyes, nose, and mouth. This process identifies the 68 facial landmarks on the face, as illustrated in Figure 3.



(a) (b)

Figure 3. 68 Facial Feature Points

- **Python Imaging Library (PIL)**

  ImageDraw is one of the modules in PIL, used for drawing operations on an image. It employs the RGBA color mode, representing Red (R), Green (G), Blue (B), and Alpha (A) for transparency. In this project, we utilize ImageDraw to apply makeup, specifically lipstick and eyeliner. The process involves selecting corresponding regions based on the 68 facial landmarks obtained from Face Landmark Estimation. For example, the corresponding positions for the upper lip are [48, 49, 50, 51, 52, 53, 54, 64, 63, 62, 61, 60, 48]. A specified color and transparency are then applied to draw a polygon, which is filled to complete the coloring. Similarly, the corresponding positions for the left eyeliner are [36, 37, 38, 39, 40, 41, 36]. A designated color, transparency, and thickness are applied to draw the lines. The execution results are illustrated in Figure 4.

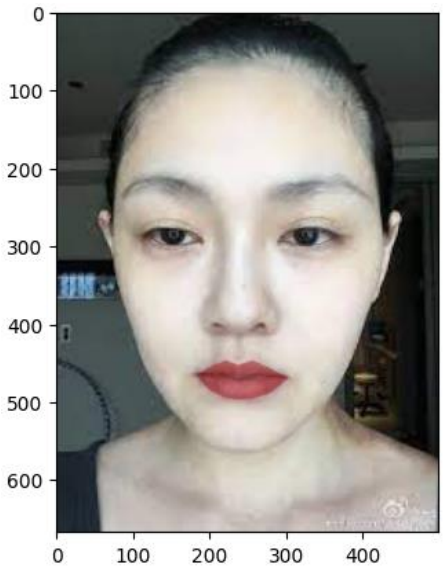


Figure 4. Drawing Results of Lipstick and Eyeliner

- **SciPy (splprep, splev)**

  Using the splprep function from SciPy, given the boundary points (xi, yi) of a polygon as standard points, a smooth estimation is performed between these standard points by creating a piecewise function. Subsequently, the splev function is employed to calculate corresponding new_yvalues based on new new_xvalues for interpolation. This process connects them to form a smooth curve or surface, as depicted in the Figure 5. We utilize SciPy to obtain the boundary points for the blush

region and eyebrow region in the " get_boundary_points" and " get_boundary_points" functions. The results are shown in the Figure 6.

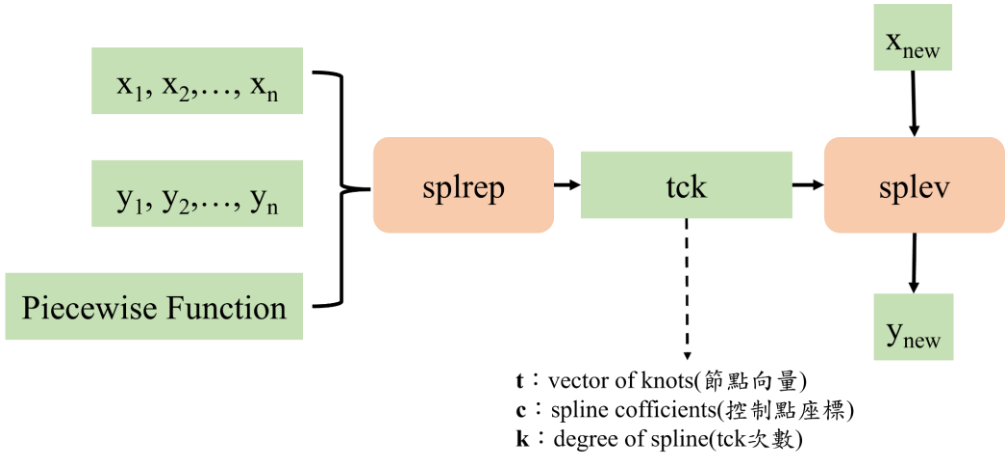


Figure 5. Scipy Flowchart

- **LAB Color Space**

  In the LAB color space, the L* component represents brightness, while the A* and B* components represent color information. Compared to the RGB color space, LAB provides a more uniform color representation to the human eye. Because brightness and color are separated in LAB, adjustments can be made independently. In this project, we use the LAB color space to apply blush. As achieving a more refined and even blush effect requires precision, we perform a conversion from the original RGB color space to LAB in the "apply_blush_color" and "apply_eyebrow_color" functions. This aims to achieve a more natural blush and eyebrow effect, as illustrated in the Figure 6.

- **cv2 (fillConvexPoly, GaussianBlur, bilateralFilter)**

  The functions fillConvexPoly, GaussianBlur, and bilateralFilter are all functions in OpenCV. fillConvexPoly is used to fill a convex polygon on an image; GaussianBlur is employed to apply Gaussian blur to an image by averaging the weight for each pixel in the image, with the weight generated by a Gaussian function; bilateralFilter is a non-linear filter that preserves edges while smoothing the image based on both color intensity and spatial location. In this project, we use fillConvexPoly in the "smoothen_blush" and "smoothen_eyebrow" functions to fill the blush and eyebrow areas. Additionally, in "smoothen_blush," we use GaussianBlur to apply a blur effect to the blush, and in "smoothen_eyebrow," we use bilateralFilter to smooth the eyebrows. The results in the Figure 6.

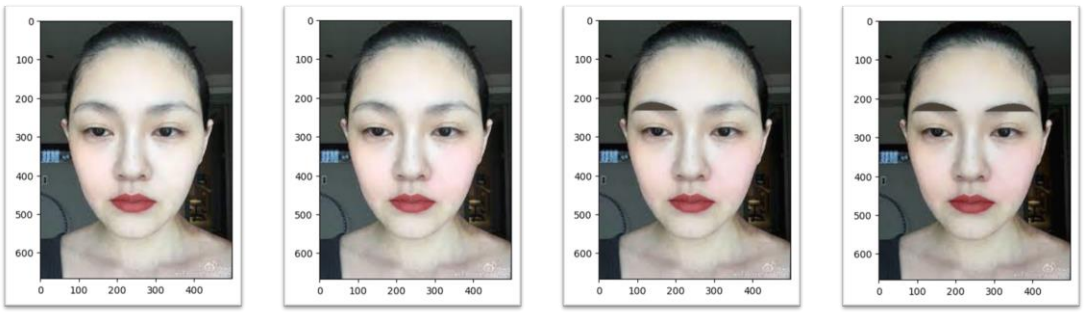


Figure 6. Drawing Results of Blush and Eyebrow

- **Method for Eyelashes (Synthesis)**

➢ **Step 1: Obtain the alpha mask for eyelashes**

1. Load the eyelash image with an alpha mask (png file), as shown in Figure 7.
2. Separate the RGBA image into individual channels.
3. Merge the channels to obtain the eyelash image in BGR format.
4. Normalize pixel values to the range [0, 1].
5. Extract the alpha channel as a mask for subsequent semi-transparent composition.

Figure 7. PNG file of eyelashes

➢ **Step 2: Manually annotate the coordinates of 8 facial landmarks on the eyelashes, such as Figure 8.**

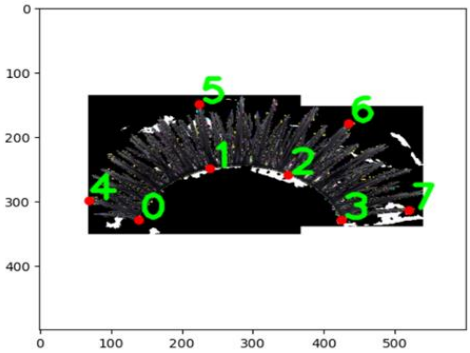


Figure 8. 8 Facial landmark points for eyelashes

- **Step 3: Build Delaunay Triangulation Based on Eyelash Landmarks (Topological Structure Between Landmarks)**
    1. Obtain the size information of the eyelashes as the bounding rectangle.
    2. Call the calculateDelaunayTriangles() function from the FBC library, using the rectangle region and the set of landmarks as input parameters to construct the Delaunay triangulation.
    3. Retrieve the list of triangles (result of Delaunay triangulation on eyelash landmark points).
- **Step 4: Select Feature Points for Left and Right Eyelashes**
    1. Based on the positions of 68 facial landmarks obtained from Face Landmark Estimation, select the corresponding ranges. For example, the corresponding positions for the left eyelashes are [36, 37, 38, 39], and for the right eyelashes, they are [42, 43, 44, 45].
    2. Choose the corresponding feature points based on specific indices.
    3. Add the anticipated feature points for subsequent eyelash synthesis.
    4. Include the anticipated feature points in the list of feature points.
- **Step 5: Apply Affine Transformations to Delaunay Triangular List**
    1. Obtain points corresponding to triangles on img1 and img2 for the first eyelash.
    2. Obtain points corresponding to triangles on img1 and img2 for the second eyelash.
    3. Apply affine transformation to the triangles on the first eyelash.
    4. Apply affine transformation to the triangles on the second eyelash.
- **Step 6: Composite Face and Eyelashes (Attach Eyelashes on the Upper Inner Eyeline)**
    1. Calculate the mask for eye eyelashes and normalize the alpha mask to the range of 0 to 1.
    2. Apply the mask to the original image to obtain the portion without applied eyelashes.
    3. Retrieve the original eyelash image corresponding to the applied eyelash portion.
    4. Add the two images to obtain the final image with applied eyelashes. The results in the Figure 9.

Figure 9. Drawing results of eyelashes

★ (FBC) faceBlendCommon.py is a utility module for facial synthesis and transformation.

1. calculateDelaunayTriangles(rect, points): Calculates the Delaunay triangulation for a set of points and returns the index vectors of the 3 points for each triangle.
2. constrainPoint(p, w, h): Constrains a point within the boundaries.
3. warpTriangle(img1, img2, t1, t2): Warps a triangular region from one image to another.
4. warpTriangle(img1, img2, t1, t2): Warps a triangular region from one image to another.

These functions are used for facial feature processing, including point extraction, affine transformation, Delaunay triangulation, etc., providing fundamental tools for facial image processing.

4. Result

Based on the mentioned method, we utilize HOG to detect the face's position and use Face Landmark Estimation to obtain 68 facial landmarks. Subsequently, we use ImageDraw to draw lipstick and eyeliner on the image in sequence. Additionally, we use cv2 to draw blush and eyebrows, resulting in the interim outcome shown in Figure 10.

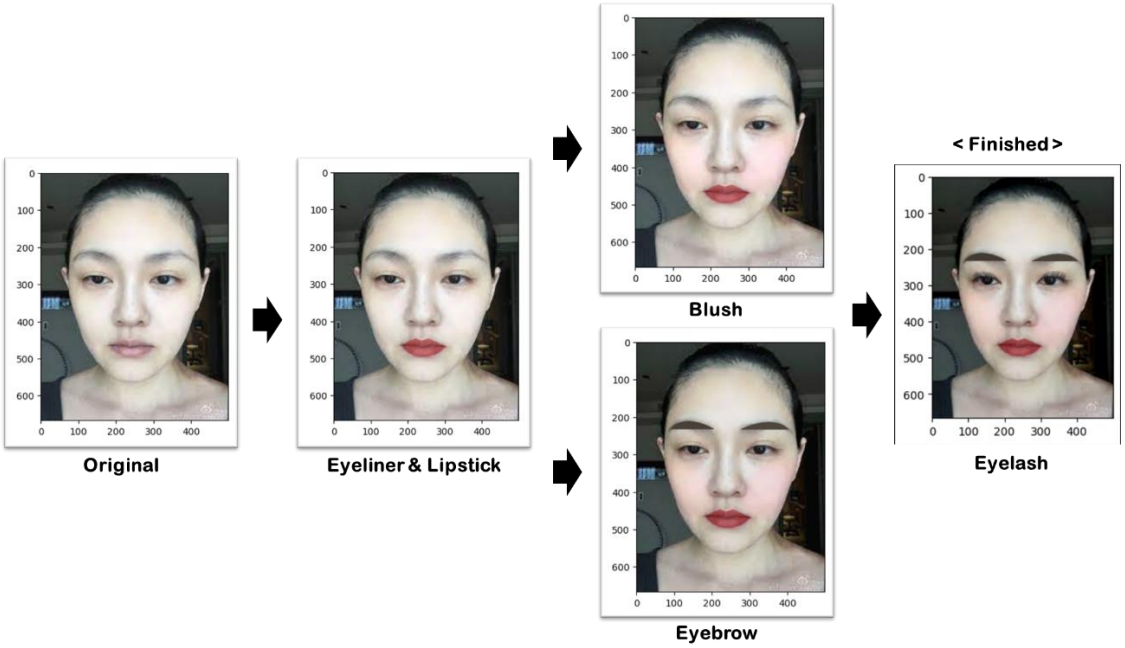


Figure 10. Finished makeup result image

5. Conclusion

We have adapted the code, originally designed with Caucasian faces as samples, to utilize Asian faces as samples. Additionally, we have adjusted the parameters within the code to create makeup styles that are better suited for Asian faces. This includes features such as eyeliner, lipstick, blush, and the addition of functionalities to draw eyebrows and eyelashes.

6. References

【Applying-Face-Makeup】

https://github.com/hiteshvaidya/Applying-Face-Makeup_

【Dlib 人臉檢測】HOG 特徵描述方法

https://www.twblogs.net/a/5eddf0633b859a4f024b5fdf

【基於 opencv 和 shape_predictor_68_face_landmarks.dat 的人臉辨識監測】

https://blog.csdn.net/monster663/article/details/118341515

【Python 小白數據: Scipy 精講 教程】

https://kknews.cc/zh-tw/code/9gvy6aj.html

【coolors】

https://coolors.co/4a4238-4d5359-508484-79c99e

【(FBC) faceBlendCommon.py】

https://github.com/spmallick/PyImageConf2018/blob/master/faceBlendCommon.py

【合成人臉與睫毛】

https://youtu.be/k3x2sncyoiE?si=sNTeoPy_pJgHRjWc