Face Beautification And Oil Painting Effects For Images - Final Report, ICG Term Project

林怡伶 (R10922165), 劉仁軒 (R10922151)

Abstract

In this project, we seek to implement two different techniques of beautifying images, face beautification and oil painting effects. The first one allows for the enhancement of the aesthetic appeal of human faces in frontal photographs while maintaining close similarity with the original. The second one presents a computer algorithm for creating an image with a hand-painted appearance from a photograph, which is capable of a wide range of visual styles. Lastly, we present the results of combining these two techniques to beautify profile images.

Key Words: Face Beautification, Image Warping, Thin-Plate Spline, KNN, SVR, Gaussian Blur, Sobel filter, Color Space

---

1. Introduction

Social media has become an integral part of people’s daily lives. It enables communication for not only one's personal life but also for business life. And a user’s profile is the starting point for it — it allows others to view curated content from that individual. Among all the elements of an engaging social media profile, a profile photo is one of the most important ones, as people tend to be more attracted to images than to texts. Besides, some research has shown that physical attractiveness has important social consequences [Little et al. 2010]. For example, attractive people usually have more dates than less attractive people and are more likely to be hired for jobs. Therefore, it could be beneficial for people to use more attractive photos in their profiles, whether to make good first impressions or to get more attention in the virtual world.

In our work, we seek to automatically enhance the facial attractiveness of human faces in frontal face photos by reproducing the method presented by Leyvand et al. [2008]. Furthermore, we adopt the method presented by Hertzmann et al. [1998] to add oil painting effects to the enhanced photos to make them more eye-catching. We will discuss our implementation of face beautification in section 2 and oil painting effects in section 3. The results are presented in section 4. Section 5 describes our thoughts on future work and the conclusions we draw.

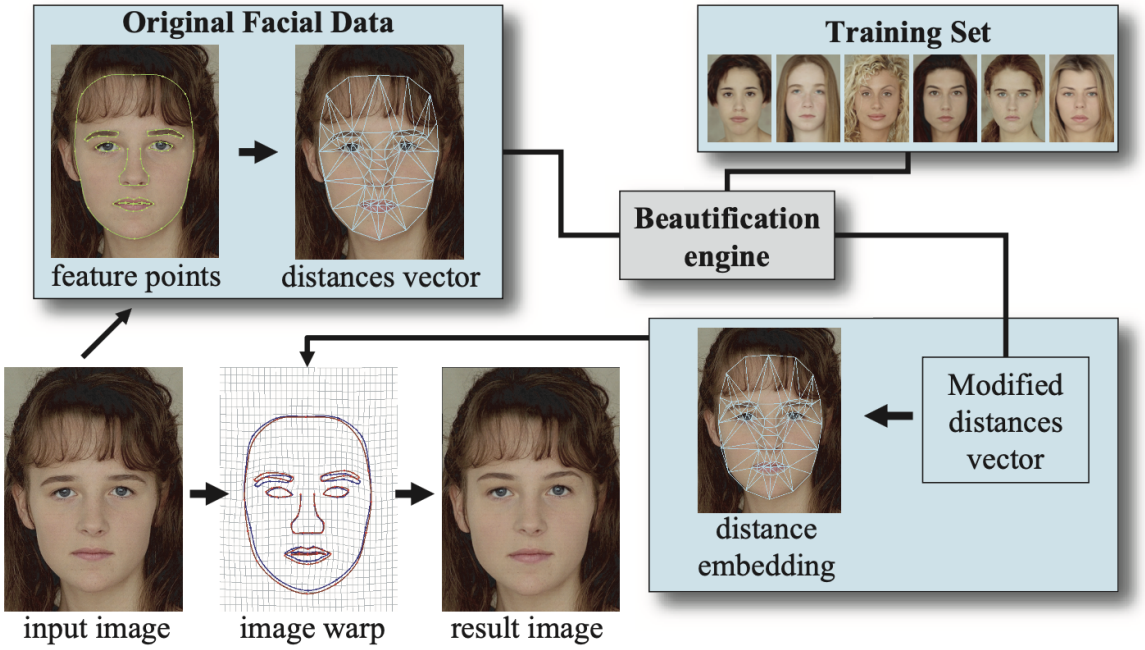

Figure 1. Facial beautification process presented by Leyvand et al. [2008]

2. Face Beautification

We follow the face beautification process shown in Figure 1. Given a frontal face picture as an input, we first detect specific facial landmarks. Then construct a Delaunay triangulation using these landmarks as vertices. The lengths of the edges in the constructed graph form a distance vector. This vector is then fed into the beautification engine, yielding a modified distance vector that has a higher predicted beauty score than the original distance vector. The corresponding distances are embedded in the plane and serve as a target to define a 2D warp field that maps the original facial landmarks to their adjusted locations.

2.1 Data Collection

We adopt a different dataset than the original paper since we cannot get access to their dataset. In the original paper, they built separate beautification engines for males and females. The training set of the female beautification engine contained 92 frontal face photos of young Caucasian females and the corresponding ratings collected by Eisenthal et al. [2006]. For the male beautification engine, a second training set of 33 portraits of young men were used, with the attractiveness of each face acquired using an identical protocol.

In our work, we use a subset of the FEI Face Database [4], which contains manually aligned frontal face pictures with a neutral or non-smiling expression of 200 individuals (including 100 males and 100 females between 19 and 40 years old with

distinct appearance, hairstyle, and adorns). We then split the 200 images into a training set of 150 images and a testing set of 50 images. The training set is used to train the SVR beauty score estimator described in the next section. And the testing set is used to demonstrate the results of the face beautification process. The attractiveness of each face is rated on a scale of 1 to 5 by 4 human raters (including us), 2 males and 2 females. The average rating of a face is referred to as its *beauty score*.

2.2 Facial Feature Extraction

The original paper used the OpenCV Haar classifier cascade and the ASM algorithm to detect facial landmarks. However, the ASM model needs to be trained with annotated landmarks. Due to time constraints, we use the landmark detector offered by dlib [6] to extract 68 facial landmarks from a given face. The resulting landmarks don't include the outline of the upper face.

2.3 SVR Beauty Score Estimator

The extracted 68 facial landmarks are used to construct a Delaunay triangulation. The triangulation consists of 178 edges. The lengths of these edges form the 178-dimensional *distance vector* corresponding to each face. The distance vectors are then normalized by the square root of the face area before the following steps.

We trained our SVR model with the normalized distance vectors and their corresponding beauty scores. The SVR defines a smooth function $f_b : R^d \rightarrow R$ that can be used to estimate the beauty score of distance vectors outside of the training set. We used the Radial Basis Function kernel and performed a grid search over the width of the kernel σ, the slack parameter C, and the tube width parameter ε, as described in the paper.

2.4 KNN Beautification

Let v_i, and b_i denote the set of distance vectors corresponding to the training set samples, and their corresponding beauty scores, respectively. Given a distance vector v, the beauty weight w_i for v_i is defined as

$$w_i = \frac{b_i}{\|\mathbf{v} - \mathbf{v}_i\|}, \tag{1}$$

where b_i gives more weight to the more beautiful examples. We then sort $\{v_i\}$ by w_i in descending order, and search for the value of K maximizing the SVR beauty score f_b of the weighted sum

$$\mathbf{v}' = \frac{\sum_{i=1}^{K} w_i \mathbf{v}_i}{\sum_{i=1}^{K} w_i}. \tag{2}$$

The weighted sum *v'* that yields the highest beauty score is then used to calculate the target landmark positions.

2.5 Distance Embedding

After yielding *v'*, to obtain the target landmark positions $q_i = (x_i, y_i)$, *where* $i = 1,2,...68$, the paper defined

$$E(q_1, \ldots, q_N) = \sum_{e_{ij}} \alpha_{ij} \left(\|q_i - q_j\|^2 - d_{ij}^2 \right)^2, \tag{7}$$

where e_{ij} is the facial mesh connectivity matrix. To reduce non-rigid distortion of facial features, Leyvand et al. set α_{ij} to 1 for edges that connect feature points from different facial features, and to 10 for edges connecting points that belong to the same feature. The target distance term d_{ij} is the entry in *v'* corresponding to the edge e_{ij}. Note that in this step, *v'* needs to be multiplied by the square root of the area of the face that we seek to beautify. The target landmark positions q_i are obtained by minimizing *E*.

Leyvand et al. used the Levenberg-Marquardt (LM) algorithm to perform this minimization. However, we when couldn't find feasible solutions using this algorithm. To solve this problem, we adopt the Trust Region Reflective algorithm and set the bound of each target landmark position to *(0, max(image_width, image_length)).*

2.6 Image Warping

The distance embedding process maps the set of feature points $\{p_i\}$ from the source image to the corresponding set of target positions $\{q_i\}$. These points are then used to find a warping field that maps all 2D points from the source image to the target one. Different from the paper, we adopt the Thin Plate Spline warping algorithm to solve for two smooth functions from which we can sample for discrete displacements along the x and y directions. The two functions are shown below.

$$f_{x'}(x, y) = a_1 + a_x x + a_y y + \sum_{i=1}^{N} w_i U(||(x_i, y_i) - (x, y)||)$$

$$f_{y'}(x, y) = a_1 + a_x x + a_y y + \sum_{i=1}^{N} w_i U(||(x_i, y_i) - (x, y)||)$$

The three first coefficients (a_1, a_x, a_y) represent the linear plane that best approximates all x' (or y') of all control points (x_i, y_i). The later terms w_i for $i \in [0,N)$ denotes the weight of its kernel surrounding each control point to the final x or y displacement. The term $U(||((x_i, y_i) - (x, y))||)$ represents the distance from a control point (x,y) to the kernel of the control point (x_i, y_i), which can be calculated as $U(r) = r^2 log(r)$.

3. Oil Painting Effects

Computer technology now allows the creation of highly realistic images of natural and imaginary scenes. But the technology for producing non-photorealistic works such as paintings and drawings is less advanced. Hertzmann et al. present a method for painting with different brush sizes to express various levels of detail in a painting [3]. In order to create such paintings, their algorithm takes an image and a list of brush sizes $R_1, R_2, \ldots R_n$ as input, it then proceeds to paint a series of "layers" using brush size R_i, from largest to smallest, on an initially empty canvas.

Algorithm 1 (Painterly rendering algorithm)

(1) function paint(sourceImage, $R_1, R_2, \ldots R_n$ *) {*

(2) canvas = new canvas with RGB(0,0,0) and same size as source image

(3) for each brush radius $R_1, R_2, \ldots R_n$ *from large to small {*

(4) referenceImage = Gaussian blur(sourceImage , $\boldsymbol{\sigma} = f_{\sigma} R_i$ *)*

(5) paintLayer(canvas, referenceImage , R_i *)*

(6) }

(7) }

3.1 Gaussian Blur

We use Gaussian Blur of standard deviation $f_{\sigma} R_i$ on the original image to make sure that the noise of the image is small enough to not make an influence in the decision on the color we paint. The larger the brush size requires the reference image to have a more significant level of blurring.

3.2 Painting Layer

Algorithm 2 (Paint layer procedure of algorithm1)

```
(1)  function paintLayer(canvas, referenceImage, R) {
(2)      S = List of strokes (position, color), initially empty
(3)      D = pointwise difference of canvas and referenceImage
(4)      for x =0 to width , stepsize = R ; y =0 to height , stepsize = R {
(5)          M = region (x-grid/2 ... x+grid/2 , y-grid/2 ... y+grid/2)
(6)          areaError = Σ_{i,j∈M} D_{i,j} / grid^2
(7)          if (areaError > T) {
(8)              (x_1, y_1) = argmax_{i,j∈M} D_{i,j}
(9)              s = makeStroke(R, x, y, referenceImage)
(10)             add s to S
(11)         }
(12)     paint strokes S on canvas in random order
(13)     }
(14) }
```

The algorithm then divides the reference image (blurred image) into grids, in which the grid size is $R_i * R_i$. Then calculate the color difference between the grid and the corresponding position on the canvas. If the difference exceeds a certain threshold, we paint the grid on the canvas.

3.2.1 Color Difference for Grid

Each pixel's color is represented as (r, g, b), and the color difference between each pixel and the corresponding position on the canvas is calculated using the Euclidean metric on R^3 as

$$|(r_1, g_1, b_1) - (r_2, g_2, b_2)| = ((r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2)^{1/2} \quad (1)$$

We then take the mean of the calculated differences of the pixels in the grid as the grid-level difference. Although some metrics are more perceptually correct, such as distance in CIE LUV [8]. Surprisingly, they gave worse results than this metric.

3.2.2 Reduce Overhead Using Z-buffer

To reduce the overhead of storing and randomizing a large list of brush strokes, the paper suggests using a z-buffer which renders each stroke with a random z value as soon as it is created. Since the computational bottleneck is not the randomization of

the list, and storing the list won't cause any burden (as far), we trade off this against multi-thread programming as we discuss later.

3.3 Make Spline Stroke

Algorithm 3 (Make stroke procedure of algorithm2)

```
( 1 )  function makeSplineStroke(R, x0, y0, referenceImage) {
( 2 )      strokeColor = color at (x0, y0) in referenceImage
( 3 )      K  = List of points in the stroke, initially contains (x0, y0)
( 4 )      (x, y) =  (x0, y0)
( 5 )      (lastDx, lastDy) = ( 0 , 0)
( 6 )      for i = 1 to maxStrokeLength{
( 7 )          if( i >  maxStrokeLength &&
( 8 )          paint strokeColor on (x,y) makes it worse){
( 9 )              return K
( 10 )         }
( 11 )         if( no gradient at point (x, y) ) {   return K  }
( 12 )         (gx, gy) = gradient direction of referenceImage at (x,y)
( 13 )         (dx, dy) = (-gy, gx)
( 14 )         if( inner product of (dx, dy) and (lastDx, lastDy) < 0){
( 15 )             (dx, dy) =  (-dx, -dy)
( 16 )         }
( 17 )         (dx, dy) = unit( fc*(dx,dy) + (1-fc)*(lastDx,lastDy) )
( 18 )         (x, y) = (x + R*dx, y+ R*dy)
( 19 )         (lastDx,lastDy) = (dx, dy)
( 20 )         add (x, y) to K
( 21 )     }
( 22 )     return K
( 23 )  }
```

This algorithm is an implementation of line 9 in algorithm 2.

3.3.1 Gradient of Image

The gradient for each pixel is from the Sobel filtered luminance of the referenced image, where the luminance is defined as

$$L(r,\ g,\ b) = 0.3 * r + 0.59 * g + 0.11 * b \qquad (2)$$

Since the referenced image has already been blurred, the computed gradient will not suffer from noise.

3.3.2 Stroke Direction

When the conditions on line (7), (8), and line (11) are satisfied, the normal vector of the gradient on that pixel is set as (dx, dy), which can be considered as the direction that makes the luminance change less. However, there are actually two sets of opposite normal vectors. We pick the candidate that has a non-negative inner product with the last direction drawn of this stroke, i.e. the angle in between is less than 90 degrees. At last, we can exaggerate or reduce the brush stroke curvature by adding a predefined filter constant f_c and calculating the next direction as below

$$D'_i = f_c D_i + (1 - f_c) D'_{i-1} \ ,\ D_i = (dx_i, dy_i)$$
$$where\ D'_i\ is\ the\ stroke\ direction\ in\ the\ i'\ th\ pick$$

3.4 Computation Speedup

Although drawing splines can improve the paintings, the computational time increases significantly when the spline lengths get bigger, and the resolution of the picture is larger. The render time of a picture with the minimum stroke length set to 5 is about ten times more than that of a picture created with point strokes. Besides, the computational time increases linearly to the number of pixels. As a result, we adopt parallel programming to speed up the computation. The loop in line (4) of algorithm 2 can be fully parallelized since there are no changes in the canvas and the reference image. There is no need to worry about race conditions and the order of execution. We have tried both the sequential and parallelized versions of rendering Figure 7. on Google Colab. It takes 21 minutes for the parallelized version and more than 3 hours for the sequential version.

	Sequential	Parallelized (Ours)
Render time of Fig 7.	> 3 hours	21 minutes

3.5 Rendering Styles and Parameter

This algorithm proposed the “style parameters” to control the rendering process. These parameters should provide an intuitive way to vary the visual quantities of the painting.
These are the style parameters we pick:
(1) Approximation threshold T: This parameter can be explained as how closely the painting must approach the source image or how much difference it can tolerate. A higher value causes rougher paintings.
(2) Brush size R_1, R_2, … R_n
(3) Curvature filter f_c : value between 0 and 1. A higher value makes it care less about the last stroke.
(4) Blur factor f_σ: value larger than 0. A higher value makes the reference image passed to algorithm 2 blur more
(5) Minimum and maximum stroke lengths

4. Results

4.1 Face Beautification

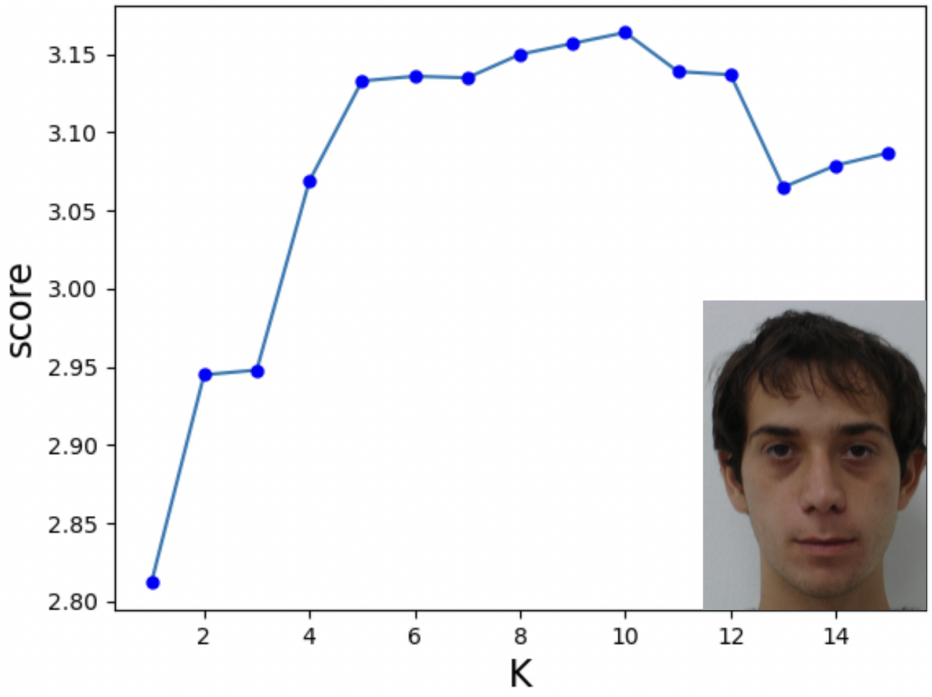

Figure 2. The beauty score is plotted as a function of K in our KNN beautification applied to one of the faces in our testing set. The optimal value of K is 10 with a SVR beauty score of 3.16. The initial beauty score of this face is 2.78.

Figure 3. From left to right: original face, KNN-beautified with K=4, KNN-beautified with optimal K=10.

Figure 4. Successful beautification of faces in the testing set. Top row: original pictures; Bottom row: the results produced by our implementation.

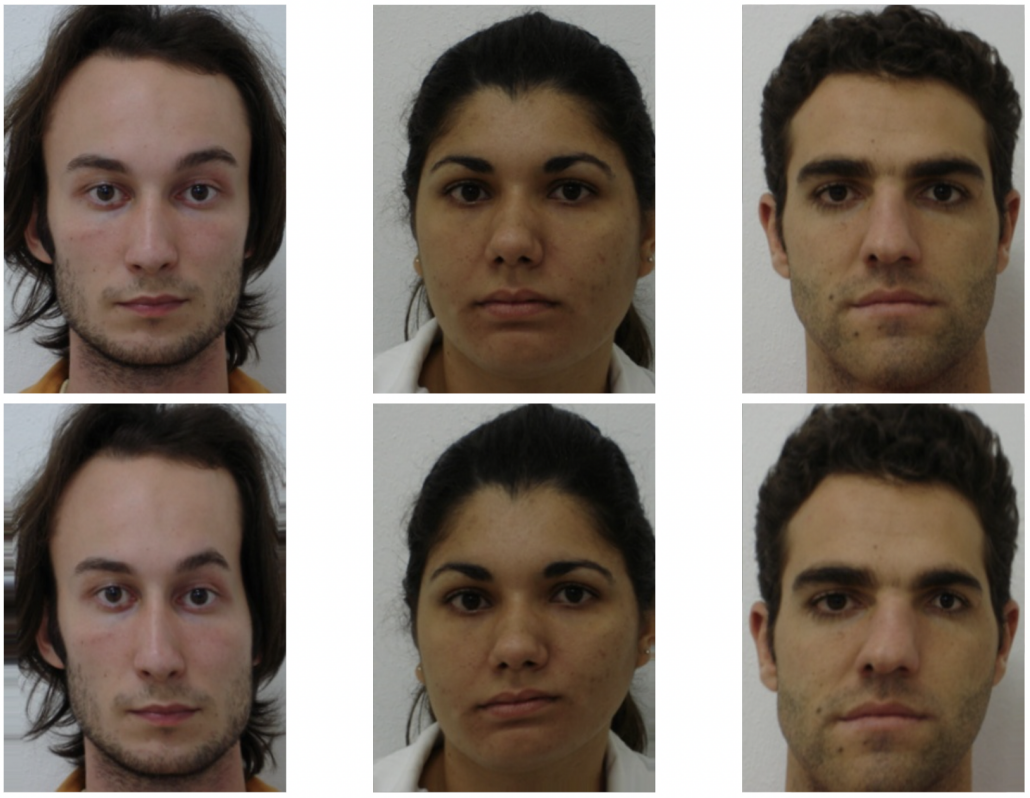

Figure 5. Distorted results of beautification of faces in the testing set. Top row: original pictures; Bottom row: the results produced by our implementation.

The plot in Figure 2. shows how the beauty score changes for different values of K. The optimal K for each face may be different. Leyvand et al. found that small values of K produce higher beauty scores than that of the average face. In our experiment, we found that the average optimal K for the faces in the testing set is 3.65, which is in line with the results from the original paper. Figure 3. shows an example of KNN-beautified faces with different values of K.

Figure 4. shows a number of input portraits (from the testing set) and their corresponding beautified versions. Our beautification increases the SVR beauty score by 28%. However, we do notice that our beautification engine mainly focuses on adjusting the outline of faces and interocular distance, while in the original paper, more sophisticated modifications can be made, such as smiles and the shape of brows. Although we were able to successfully transform most of the faces into ones that were considered more beautiful and symmetrical, there are still a few distorted results, as shown in Figure 5. The possible reasons and solutions are discussed in section 5.

4.2 Oil Painting Effects

Figure 6. Expressionist painting style. There are three layers, with brush size 8, 4, and 2 respectively (from left to right). T = 50 , f_c = 0.25, f_σ = 0.5, minLength = 10, maxLength = 16.

Figure 7. Impressionist painting style [9]. There are three layers, with brush size 8, 4, and 2 respectively (from left to right). T = 100 , f_c = 1, f_σ = 0.5, minLength = 4, maxLength = 16.

As we can see, the expressionist style has a lower approximation threshold than the impressionist one. The result can be observed in the leftmost picture of both styles. The impressionist style can tolerate more differences, which causes most of the canvas to be blank. In addition, the minimum length has a great impact on painting styles. We can create a pointillist-style painting if we restrict the minimum and maximum lengths to one.

4.3 Combined Results

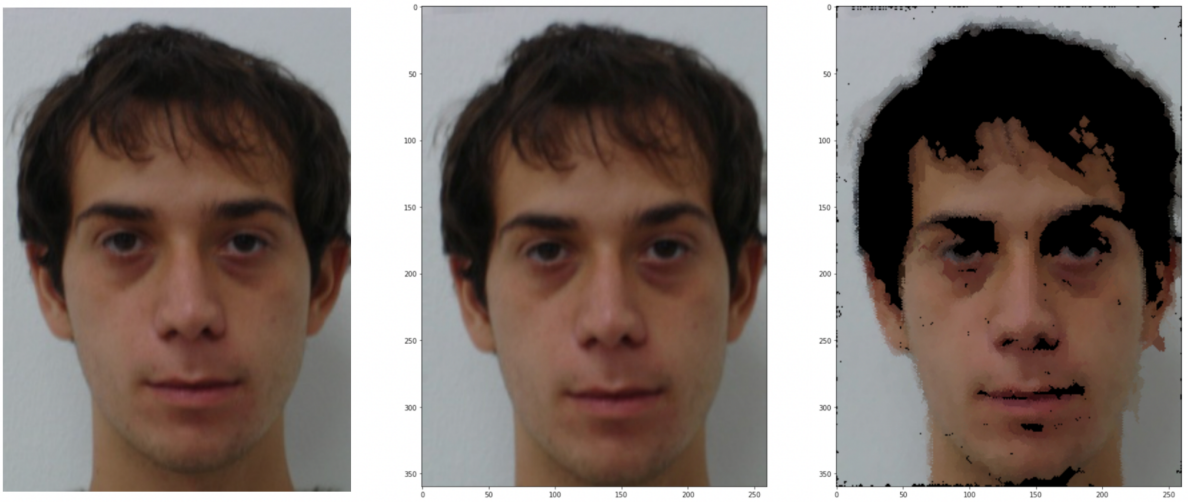

Figure 8. From left to right: original face, KNN-beautified with optimal K, beautified face with oil painting effects added.

Figure 8. shows an example of combining both our implementations on a face in the testing set. We can see that the face in the middle picture is more symmetrical than the original one. After adding the oil painting effects to the beautified picture, it becomes more eye-catching.

5. Conclusions and Future Work

As discussed in section 4.1, our results of face beautification weren't entirely successful. The possible reasons for this are: (1) we train the SVR model jointly for males and females from different ethnicities, however, the features of attractive faces depend highly on gender and ethnicity. (2) we only have 4 human raters, while the beauty scores in the original paper were rated by 28 people. (3) most of the faces in the dataset we use are considered much less attractive than the faces in the training set of the original paper. Therefore, we think that it is necessary to train the SVR model separately for different genders and ethnicities. Also, with a small number of human raters, the derived beauty scores may not be able to reflect consensus opinions on facial beauty. Accordingly, more human raters or facial attractiveness predictor trained on a large dataset is needed. Above all, we found that the attractiveness of the faces in the training set has a significant impact on the resulting beautified images since our resulting beautified faces lie within the corresponding face space. To yield better results, the dataset used needs to be collected or chosen carefully.

As for oil painting effects, although the results are quite satisfying, there are a few key points that we haven't achieved to simulate human paintings as we wish.
The first point is the texture of the painting. Some algorithms specified with a certain painting style have textures in each stroke they paint [10]. This includes other factors, such as the material of the canvas, the paint used, the lighting, and the reflection of the painting. The second point is the transparency of the painting. Painters don't often paint the exact color of a position. There are a lot more factors that they take into consideration, such as whether the color is too dark that it affects the color we wish to paint on it. This is why the background color is often yellow or bright and the dark colors are left until last.

6. Individual Contribution

(1) 林怡伶 (R10922165): Face Beautification
(2) 劉仁軒 (R10922151): Oil Painting Effects, help with debugging face beautification

References

[1] Little AC, Jones BC, DeBruine LM. Facial attractiveness: evolutionary based research. Philos Trans R Soc Lond B Biol Sci. 2011;366(1571):1638-1659. doi:10.1098/rstb.2010.0404

[2] Tommer Leyvand, Daniel Cohen-Or, Gideon Dror, and Dani Lischinski. 2008. Data-driven enhancement of facial attractiveness. ACM Trans. Graph. 27, 3 (August 2008), 1–9. https://doi.org/10.1145/1360612.1360637

[3] Aaron Hertzmann. Painterly Rendering with Curved Brush Strokes of Multiple Sizes. 1998

[4] FEI Face Database https://fei.edu.br/~cet/facedatabase.html

[5] EISENTHAL, Y., DROR, G., AND RUPPIN, E. 2006. Facial attractiveness: Beauty and the machine. Neural Computation 18, 1, 119–142.

[6] dlib landmark detector
http://dlib.net/files/shape_predictor_68_face_landmarks.dat.bz2

[7] Thin Plate Spline https://profs.etsmtl.ca/hlombaert/thinplates/

[8] JAMES FOLEY, ANDRIES VAN DAM, STEPHEN FEINER, JOHN HUGHES. Computer Graphics: Principles and Practice, Addison- Wesley, 1995.

[9] PETER LITWINOWICZ. Processing Images and Video for An Impressionist Effect. SIGGRAPH 97 Conference Proceedings, pp. 407-414. August 1997.

[10] Gao, Junjie, Dajin Li and Wenran Gao. “Oil Painting Style Rendering Based on Kuwahara Filter.” IEEE Access 7 (2019): 104168-104178.