CS 543 - Final Project Report

Artistically Stylizing the Face using Portrait Paintings

Yukang Shen (yukangs2), Yujin Zhang (yujinz2)

Github: https://github.com/yujinz/art-face

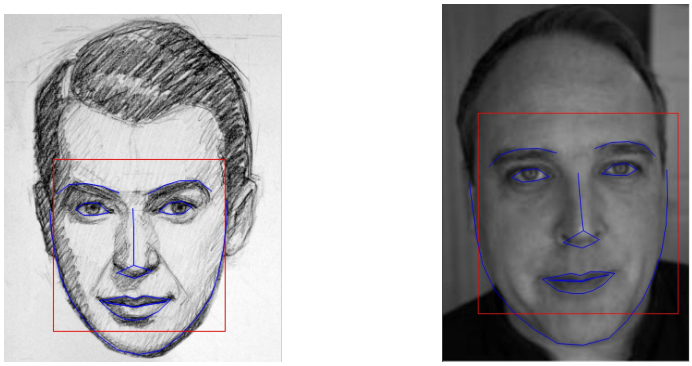
I. Introduction

With the rapid development of neural network, a lot of applications nowadays use this technology on image processing and are able to achieve some amazing results. One of the most notable outcome is Google's flagship smartphone pixel 2, whose camera especially portrait mode outperform all flagships from other companies including iPhone X and Samsung Galaxy S8 in 2017 with shockingly one camera [8], whereas portrait photos usually need two cameras to obtain the depth information and blur the background. Pixel 2 achieve this by using uses a convolutional neural network (CNN) with skip connections written in TensorFlow Mobile and successfully perform foreground-background segmentation in software level. It still has one of the best camera in a smartphone among all competing smartphones with dual or even triple cameras. However, although neural network can produce impressive results in image processing, it has its own limitations. For example, in image stylizing task, for the styles that contain rich texture information, neural-based style transfer method tends to distort local visual features and makes the overall appearance of the synthesized output becomes notably different from the original style exemplar. [3] Also training a neural network requires huge amount of data and computing power. In order to avoid those issues, we want to implement a stylizing method that does not rely on neural networks.

Inspired by the amazing work "Example-Based Synthesis of Stylized Facial Animations" by Fiser et al, [3] we want to downscale the problem from video input to static image, as well as reducing computational complexity by simplifying the implementation while maintaining a relatively high quality of the output. We focused on stylizing the photo of human faces using portrait paintings and we use an approach without deep learning that is described below in detail.

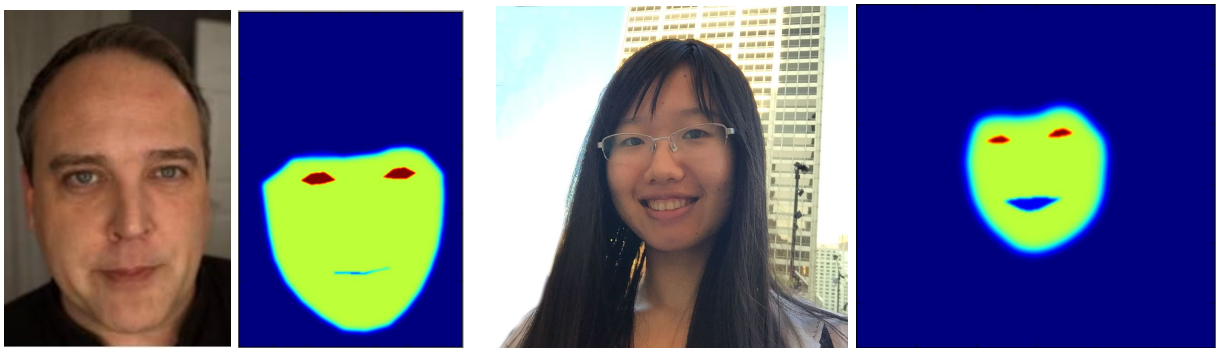
II. Our Approach

Step 1: Detect facial keypoints of both the art style image and the target image.



Step 2: Deform the art style image to match the facial structure the target image using the Moving Least Square algorithm [5] with the key points from Step 1 as control points

Step 3: Segment the face, eye, and teeth area of the target face according to the key points from Step 1. Generate a weight mask with different weight on different areas.



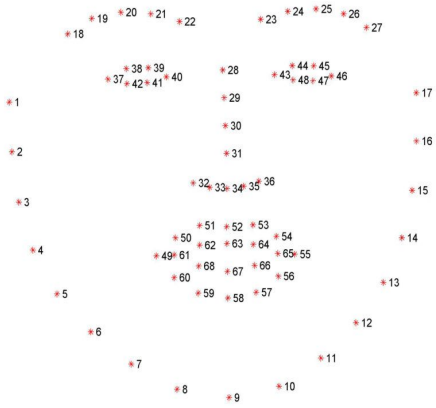
(Showing another target image with teeth for better illustration)

Step 4: Blend the deformed art style image onto the target image according to the weight mask from Step 3.

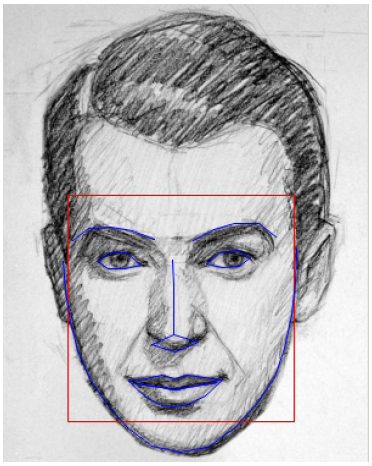
Below are the detailed explanation of each step:

1. Facial Landmark detection:

   The program first generates a bounding box of the face, and then label the facial keypoints with the help of the bounding box, both using pre-trained models from Dlib. The facial keypoints are represented by a ordered 68x2 array according to the landmark model [4].



Facial key point model [4]



Detected bounding box and connected keypoints

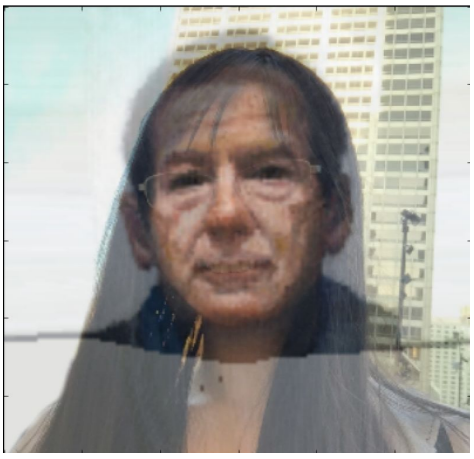
   As explained in the progress report, we decided to use Dlib for these two task because it has better documentation for Python interface than CLandmark on facial keypoints detection, and is as powerful as OpenCV on detecting the bounding box. We referred to the official snippet from Dlib [1] as well as a tutorial for video input [2] as the start files. We fixed bugs in the tutorial and modified it to suite our own use case.

2. Image Deformation:

   In order to align the art style face and target face, we need to perform image deformation. We decided to use the Moving Least Squares (MLS) algorithm [5] rather than simple transformations like homography to generate better results. Comparing to the old Least Squares Deformation, this method does not restrict on affine transformation and is able to apply different transformation for each point in image contrast from apply the same transformation to each point. We tried different implementation of this method from various repositories [6], comparing their result on the same set of images and modify the method according to our need. We finally chose the Python one as it's easy to integrate to our project and it is mostly bug-free as long as the input image is large enough.

3. Facial feature segmentation

   Experiments show that we need special processing when blending the eye, teeth, and background area. We used the `fillConvexPoly` function from OpenCV to generate the mask of an area given a list of keypoints on its boundary. We also apply Gaussian filters with different sizes to blur the boundaries of different areas to avoid hard transitions and make the blended image looks more natural.



Blending with uniform weight

Blending with weight mask

Blending with uniform weight

Blending with weight mask

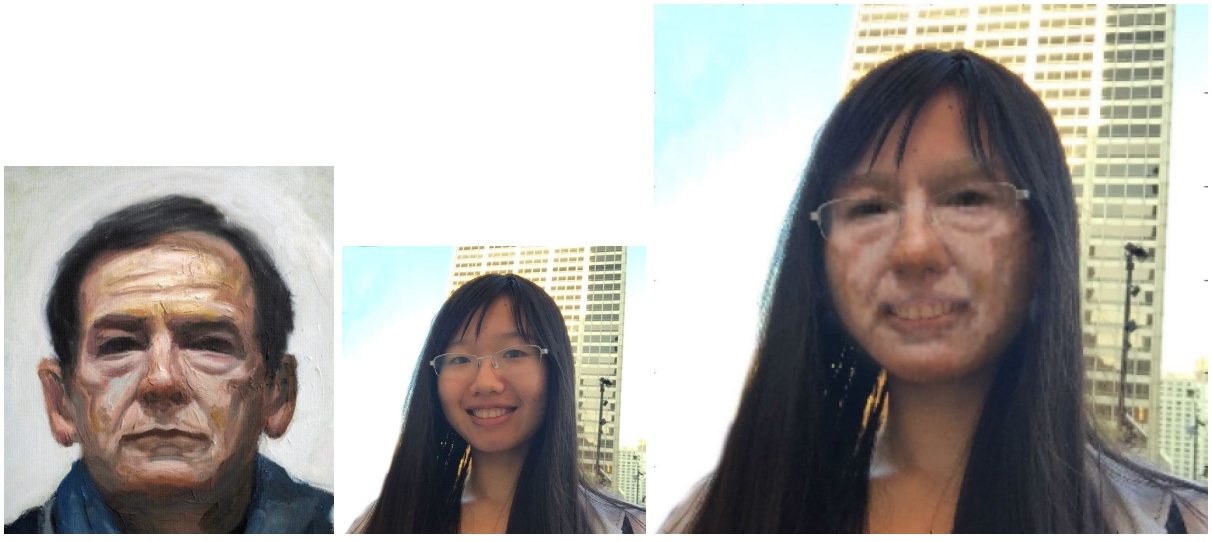
III. Results

Our program could take any two pictures with one front-facing face in each, and stylize the target picture according to the style of the painting picture. Here are some sample results with the order of original art image, original target photo and synthesized image:

A.

B.

C.



D.

E.

F.

G.

H.

IV. Discussion

While our stylization approach produced fun and desirable results overall, we learned lessons during the process and there are still some improvements that could be made upon our current progress:

1. Robustness of facial keypoint detection and Moving Least Square algorithm

   We once stuck on the inaccurate facial keypoints and buggy MLS transformation in the middle of the project. Eventually, we found that increasing the resolution of the input image can avoid the problem. When the input image is smaller than 200*160, it is likely that the art style image can be transformed incorrectly.

2. Performing face segmentation for better stylizing area

   In this project, we only stylized the face up to eyebrows and leaving the forehead unstylized, because all current facial landmark models do not detect keypoints on hairline, therefore we were not able to get a mask of the whole face. Ideally, we could achieve this by using a face segmentation model. However, some segmentation models

like Watershed Algorithm could not give us desired result, while other models [9] requires Keras or Caffe. As we decided not to involve deep learning at the beginning of this project, and both of the team members do not have experience on using them, we decided to leave the segmentation as future work and stylize the face up to eyebrows and blur the boundaries. Another limitation of our implication is that we cannot make change to the background, since decent foreground-background segmentation also requires neural network model.

V. Conclusion

Our implementation successfully produced some compelling results as stated in the project proposal, even through there are still some limitations like unstylized forehead. Those possible improvements could be leaved as future works for people who interested in integrating deep learning technics onto this project. We hope future research can take advantage of both approaches of stylizing image and create faster and more precise image synthesis application.

VI. Individual contribution

Yujin: Facial keypoint detection, weight mask and blending.
Yukang: Moving Least Square, facial feature segmentation.

VII. References

[1] http://dlib.net/face_landmark_detection.py.html
[2] https://www.pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/
[3] Jakub Fiser, Ondrej Jamriska, David Simons, Eli Shechtman, Jingwan Lu, Paul Asente, Michal Lukac and Daniel Sykora. “Example-Based Synthesis of Stylized Facial Animations”. *ACM Transactions on Graphics 36(4):155, 2017 (SIGGRAPH 2017).* July 2017.
[4] http://dlib.net/files/shape_predictor_68_face_landmarks.dat.bz2
[5] Scott Schaefer, Travis McPhail, Joe Warren. “Image deformation using moving least squares”. SIGGRAPH '06 ACM SIGGRAPH 2006 Papers. 533-540. July 2006. http://faculty.cs.tamu.edu/schaefer/research/mls.pdf
[6] Various Moving Least Square implications:
C++:

- https://github.com/jonghewk/ImageDeformation_MovingLeastSquare
- https://github.com/cxcxcxcx/imgwarp-opencv#imgwarp-opencv
- https://github.com/royshil/CurveDeformationMLS

MATLAB:

- https://github.com/zhangzhensong/movingleastsquare
- https://www.mathworks.com/matlabcentral/fileexchange/12249-moving-least-squares

Java:

- https://github.com/fiji/VIB/blob/master/src/main/java/Moving_Least_Squares.java

Python:

- https://github.com/Jarvis73/Moving-Least-Squares

[7] https://3dthis.com/facemorph.htm
[8]
https://www.dxomark.com/google-pixel-2-reviewed-sets-new-record-smartphone-camera-quality
[9] https://github.com/YuvalNirkin/face_segmentation