CS445 Final Project Report
Portrait Face Swap with Face Detection & Style Transfer

Brayden Bingham

Brian Reinbold

Ola Pietka

1 Motivation

Our final project is a unique and innovative topic that blends art with technology. We aim to go beyond a typical two-person face swap and extend its capabilities by swapping a face with one from a famous painting while still maintaining the painting's style. This idea was inspired by the popularity of image filters in social media applications like Snapchat, Instagram, and TikTok. We think incorporating more fun and unique filters where you can turn yourself into the Mona Lisa, for example, could add significant value to these platforms. This tool could also work well in image processing software such as Photoshop. Although creating and optimizing a full-fledged smartphone application would have taken well beyond the 60-hour mark, we have still developed a complete portrait face swapping pipeline that could serve as an informative proof of concept. While its impact may mainly serve as entertainment, working on this project required us to incorporate numerous computer vision fundamentals to develop it, and these skills that we developed will translate to a wide range of fields as computer vision increasingly impacts our lives. At the very least, we enjoyed working on this project, and we hope you enjoy the results too[1]

2 Approach

Below are the key steps in our program:

1. **Face Swap**: Swap the face into the portrait and blend. Face swap is made of the following components:
   (a) **Keypoint Detection**: Detect facial keypoints in face image and portrait.
   (b) Two methods for aligning face over portrait:
      - **Face Warp**: Warp face directly to portrait.
      - **Face Morph**: Averages the facial characteristics between the image and portrait.
2. **Face Detection**: Creating an algorithm to detect faces in image and portrait to crop faces.
3. **Style transfer**: Preserving the original color and texture of the painting onto the morphed face.
4. **Reconstruction**: Add cropped face back to portrait.

We divided these steps amongst our group into several milestones, so that we could each contribute to the project independently. Then together we integrated and tested the components to develop a fully functioning portrait face swap application. Now we describe our approach in more detail.

[1] Our project's code, data, and additional results can be found here.

Face Swap Once two images are supplied, we use the python library DLIB to detect 68 facial keypoints, which encompasses the boundaries of the face, eyes, mouth, and nose. This step is critical as many downstream tasks depend on it. Next, there are two methods to align and warp the face onto the portrait: face warp and face morph. Face warping transforms the face image directly onto the portrait by (1) computing the Delaunay triangulation for each face using their respective facial keypoints, (2) calculating affine transforms for each triangle correspondence, and (3) warping each triangle from the face to the portrait. It is almost like copying-and-pasting the face image onto the portrait, but there is a slight warping effect since the facial characteristics of the face image have to fit onto facial characteristics of the portrait.

Face morphing, in contrast, first warps the faces in the image and portrait towards an intermediate face. Morphing averages the characteristics of both faces, creating a nice effect. To accomplish the face morph: (1) align and scale face image relative to portrait (2) average the keypoints from both faces to create the average face, (3) warp both faces towards average face using same process as described in face warp, and (4) perform a cross-dissolve between the warped face and warped portrait. We accomplished image alignment and scaling automatically by calculating a homography between the keypoints in the face and the portrait. Using this homography, we apply a perspective shift, which will align and scale the face to the portrait. Alignment greatly improves the quality of the face morph.

Face swapping is accomplished by first finding the convex hull of the facial keypoints, which is the smallest convex set that encloses the face. We can create a mask out of this facial boundary to extract the face from the warped/morphed face. Then we can position the warped/morphed face over the portrait by calculating the mean of the portrait's convex hull, which will center the warped/morphed face over the portrait. Next we perform color correction by scaling each pixel by the ratio of pixel intensities between the portrait and the warped/morphed face across each color channel. Color correction generally is useful when applying the face warp method, but typically unnecessary in the face morph method since applying a cross-dissolve beforehand serves as an initial step in transferring the painting style to the face. Finally, we Poisson blend the warped/morphed face and portrait together.

Face Detection The next step is to detect a face in both images and portraits. We use a Haars cascade classifier from OpenCV2, which returns a bounding box of the detected face. Then we crop the face and pass it to the next steps in the process. Also, localization of the faces will be useful when reconstructing the results to perform the face swap. We also tried to train our own face detector. In total we used over 2,500 images, but the classifier was not accurately working. This led us to abandon our own classifier generation and using the default OpenCV2 classifier due to the lack of quality training images available to us.

Style Transfer Finally the process culminates in style transfer, which applies the original color and texture of the portrait onto the morphed face. At first, we thought of doing a patch based stylizing, but we felt like the results were not sufficient (and also it was quite similar to one of our MPs in the course). Instead, we put a lot of effort into finding a new approach that would better extract the portrait's characteristics. We decided to use a neural-based approach involving extracting intermediate features from VGG19 (Simonyan and Zisserman, 2015) and generate images by minimizing a loss function. Finding the right loss function is crucial to our success, as it affects the quality and realism of the generated images. We were able to find the most suitable loss function for the style transfer task through extensive research, analysis, and multiple trial-and-error iterations. Our method stands out because it utilizes a novel combination of different approaches, which, as far as we know, has not been attempted before. The style transfer process consists of two phases:

1. In phase one we introduced the Laplacian pyramid (Burt and Adelson, 1983) to minimize the content loss (Gatys et al., 2015; Selim et al., 2016) and style loss (Gatys et al., 2015), on each level. The output image from one pyramid level is bilinearly upsampled and passed as input to the next computation. This phase ensures the style of painting and content characteristics are preserved.

2. In phase two we focused on smoothing the details with additional computation of histogram loss (Risser et al., 2017) and total variation loss (Johnson et al., 2016) on the output from the previous phase. This minimizes instabilities from phase 1 and improves the overall quality.

Reconstruction After generating a new image it is time to convert it back to its original shape. For this to work we used a part of each previous stage, such us: detected face bounding box, mask of swapped region, original style image and newly generated image.

3 Results

Figure 1: Final portrait face swap results with style transfer. Original portrait and face image are in the bottom-right

Overall our entire portrait face swap pipeline produces excellent results (Figure 1) for numerous examples. When results are not desirable, we have provided an interface so that the user could select a different method or parameters at each step of the process to get their face swap just right.

Face Swap Generally, the face warp method in face swap produces the most consistent results (Figure 2). Misalignment can occur in the face swap using face morph, since the face morph process first transforms both faces to an intermediate face while the face warp method directly warps the face onto the portrait. Oftentimes misalignment can be corrected by providing some parameters to shift the center. However, if the face is significantly smaller than the portrait's face, face morph will produce a small, morphed face that will look peculiar on the portrait. Despite these tradeoffs, face morph can provide a fun effect to the face swap, and the cross-dissolve also serves as an initial step in applying the painting style to the face, which helps improve the overall quality of the blend. Obviously, our method could fail if the faces cannot be detected, but the model from DLIB is robust and works surprisingly well on portraits too. Granted if a portrait is too abstract, that could present a challenge. Still, it worked on a Pablo Picasso, so we are very pleased with the generality of our tool.

Face Detection Our face detection method performed very well at detecting faces based on the default classifier XML file. It performed much better than our own attempt to create a classifier XML file with 2,500 images. Overall we were able to detect most faces, but did see some cases where faces got cut-off in the cropped image and did not return the whole face. To mitigate this issue, a user can modify the parameters height scalar and/or width scalar to increase the size of the final cropped image ensuring the whole face is returned for style transfer. If a face could not be detected than we decided as a team to not continue processing that image due to the fact that it would be too abstract for a plausible face swap that looked synthetic.

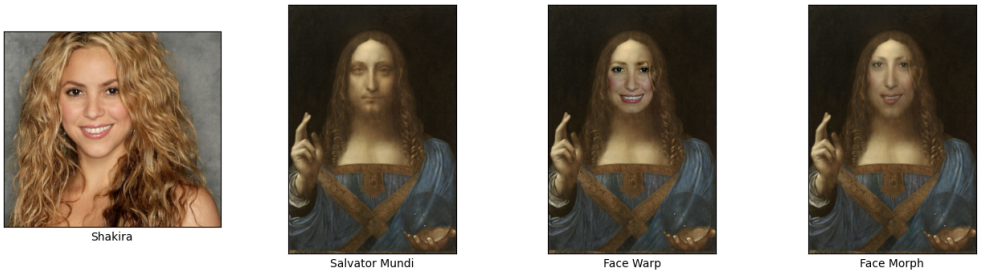

Figure 2: Face swap of Shakira with Salvator Mundi by Lenonardo da Vinci, 1500. Face warp does a better job of preserving the facial characteristics in the original face image. Face morph averages the facial characteristic between the face image and the portrait.

Style Transfer Style transfer does an excellent job of maintaining the portrait's style, producing a more realistic face swap. We observed that dividing the process in two separate phases increased the overall quality of generated image, and enhanced the effectiveness of style transfer (Figure 3). Finding the right loss function was a real challenge and it would not be possible without a lot of research and time spent on reading scientific papers. After a lot of trials and errors, a combination of different approaches was utilized making this (as far as we know) a new method for style transfer that had not been tried before. It takes approximately twenty minutes to run a full process on a GPU, which would not be ideal for a consumer application like a Snapchat filter, but would be perfect in professional photo editing software where achieving the perfect results is worth the additional computational cost. One potential drawback is if the resolution in the face image is much lower than in the portrait, style transfer can magnify this distortion.

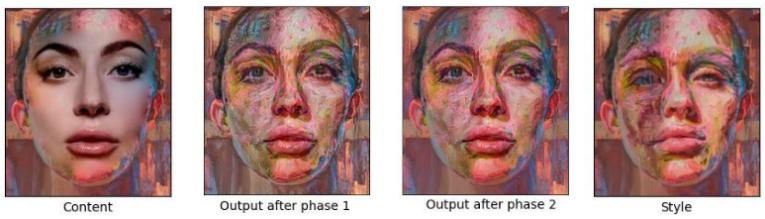

Figure 3: Stages of style transfer process.

3.1 Results Testing

We implemented three different testing validation methods to show more information about the quality of our results beyond visual inspection alone. Please refer to our project notebook to see the results of our tools.

1. **Histogram Comparing Method**: This method creates a histogram for both images and scores them from 0 to 1 indicating similarity.
2. **Fourier Transform Method**: This method creates frequency charts for both image to show the user the underlying frequency's of the result. It also calculates an MSE error between the two frequency images.
3. **Edge Detection by Local Region Scoring Method**: This method finds vertical and horizontal edges for both images to easily identify any edge artifacts that could have been created during the process. We followed a tutorial from Kharkar.

4 Details on Implementation

The project was written entirely in python using Google Colab with exception to the software Cascade Trainer GUI that was used for the face detection classifier attempt. We primarily used tools from OpenCV and DLIB in our implementation. We adapted some code chunks from tutorial by Rosebrock to adapt DLIB to our own problem. We also followed the tutorial steps for creating our own custom haar classifier by Gote. The logic behind the morphing process was inspired by work of Mallick and Efros, but we developed our own implementation. In face swap, the idea to extract the face using the convex hull of facial keypoints came from one line of code in the Github repo by Huikai. Also, implementing color correction by a relative scale factor was inspired by this one line of code in the same repo, however, we developed our own, simpler implementation.

Style transfer used a pre-trained VGG19 network (Simonyan and Zisserman, 2015) on ImageNnet dataset to get features from intermediate layers. The loss function was minimized with the Adam optimizer (Kingma and Ba, 2017) on GPU V100. The final loss function we minimized was inspired and based on these amazing papers: Gatys et al. (2015); Risser et al. (2017); Selim et al. (2016); Johnson et al. (2016). All portraits used in our project come from the open source art gallery WikiArt.org. For the test examples we used photos of Shakira (Images), Barack Obama (Souza), Lady Gaga (Vinoodh), Leonadro DiCaprio (Will), and Jennifer Lawrence (Harvey)

5 Why this Project is Challenging and Innovative

We believe our project warrants full credit. Our project presented a significant challenge due to the incorporation of a wide range of complex ideas such as image processing, image transformations, homographies, keypoint and face detection, image morphing, blending, and more. Additionally, we recognized that debugging code alone would not suffice; we leveraged our expertise in computational photography to devise creative solutions. For example, when we encountered less-than-stellar image morphing results, we drew on our knowledge of photo stitching to incorporate homographies and image alignment, which greatly improved our output. Our project stood out from other face swap programs because it incorporated not only a face warp but also a face morph, which produced a unique and compelling effect. Moreover, we pushed beyond our coursework to incorporate a state-of-the-art neural-based model for style transfer by utilizing a combination of several approaches, making this a new method that has never been tried before, which further elevated the quality of our project. Finally, we planned a project for four group members, but after we lost a group member in the final week of the project, we were still able to implement his section and produce a complete program. Overall, we successfully achieved our project goals, and we are pleased with the results.

References

P. Burt and E. Adelson. 1983. The laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, 31(4):532–540.

Alexei Efros. Cs194-26/cs294-26: Image manipulation and computational photography.

Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. 2015. A neural algorithm of artistic style.

Vipul Silip Gote. Guide to make custom haar cascade xml file for object detection with opencv.

Anthony Harvey. Jennifer lawrence photography.

Wu Huikai. Faceswap - swap face between two photos for python 3 with opencv and dlib.

Getty Images. Shakira photography.

Justin Johnson, Alexandre Alahi, and Li Fei-Fei. 2016. Perceptual losses for real-time style transfer and super-resolution.

Ritvik Kharkar. Edge detection in python.

Diederik P. Kingma and Jimmy Ba. 2017. Adam: A method for stochastic optimization.

Satya Mallick. Face morph using opencv — c++ / python.

Eric Risser, Pierre Wilmot, and Connelly Barnes. 2017. Stable and controllable neural texture synthesis and style transfer using histogram losses.

Adrian Rosebrock. Facial landmarks with dlib, opencv, and python.

Ahmed Selim, Mohamed Elgharib, and Linda Doyle. 2016. Painting style transfer for head portraits using convolutional neural networks. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, pages 129:1–129:18.

Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition.

Pete Souza. Barack obama photography.

Inez Vinoodh. Lady gaga photography.

WikiArt.org. Wikiart -visual art encyclopedia.

Vistoria Will. Leonardo dicaprio photography.