CIS581: Computer Vision & Computational Photography
Face Swapping Final Project Report

Deniz Beser, Mia Chiquier, John Wallison

1. Introduction

This final project aimed to build an algorithm for face swapping. Given two videos, the face swapping algorithm automatically swap faces between the two videos, by detecting faces, and replacing the target faces in the target video with source faces in the source video. The challenging aspect of this problem is keeping the video natural, such as maintaining the emotions of the target face after the transformation. Our implementation of face swapping employed various techniques we learned and used in the class, including CNNs, image morphing, and gradient domain blending.

2. Methodology

Our method utilizes the Python face recognition and OpenCV libraries to implement the face swapping algorithm. At the core of our methodology is the assumption that emotions are primarily dependent of facial gestures determined by the organization and movements of dominant facial components; by primarily analyzing the positioning and movement of eye, lips, chin, nose, and eyebrows, we can learn the emotion of a face. Consider any emoji; only a few of these facial components often suffice to express an emotion such as a smiley face.

In order to learn the emotion of a face, apply it to a source face, and swap it into the location of the target face, we use the following procedure. We first extract a source face (the face to be inserted in the target video). Then, for each frame in the target video, we analyze the facial gestures of the target image. We form a mapping between the source face to the target face in that frame by applying image warping. Then, we smoothly embed the morphed source face into the target video by applying gradient domain blending.

To extract emotions in the target frame, as well as to analyze the characteristic of the source image, we utilize the Python face recognition library, which allows us to get the locations of faces in a video, as well as a list of facial markers. Then, we use our code from Project 2 (Image Warping) to morph the source image, with its gestures, in the to target image in each frame. Finally, we use the OpenCV library to do gradient domain blending (equivalent to our code from Project 3 for seamless cloning) to produce a smooth embedding of the source face into the target video. Overall, applying this procedure allows us to recreate the target video with the source face. The process is applied in the same fashion to produce multiple face swaps when needed.

We also experimented with adding autoencoders as the first step of our pipeline, using neural networks to learn a latent representation of the face, and using this to reconstruct a source face on each frame. However, we found that the training required to get satisfying performance from the autoencoders was not possible with the limited training data. This is discussed in detail below.

3. Results

Our pipeline, despite its simplicity, performs well on the test videos. We can capture emotions quite accurately, and even complicated patterns such as blinking.

An example output can be found in our presentation (here). We had the best performance on the various easy videos, but also reasonably good performance swapping onto some of the medium videos. Other outputs have been uploaded to a Google Drive folder in their full resolution.

4. Future work

Although our simple effectively pipeline performs face swapping, there are numerous other ways that one can approach the problem and refine the results. Incorporating other methods of image transformation (instead of morphing) such as a secondary level of neural architecture or homographic transformations is one way to potentially improve the model.

We had experimented with using autoencoders as a deep learning alternative to reconstructing the swapped face. These autoencoders are neural networks which learn to reconstruct a full input face by first transforming it into a smaller, but higher dimensional encoding (known as the latent face), and then decoding this latent face into the original image. These two components of the network, the encoder and decoder, can be separated. In the context of face swapping, one can train two autoencoders, for both the source and replacement faces. Once trained, a target face can be encoded, then decoded with the decoder of the source face. This effectively harvests the emotion of the replacement face, and reconstructs a source face with that identical facial expression.

We found that this step requires vast amounts of training, and data, to achieve reasonable results. The short inputs videos didn’t provide enough variable training data for the networks to truly learn an encoding which sufficiently captured the face. As such, we decided not to add this step to our pipeline. Future work could be done to incorporate this “deepfake” technology into our face swapping pipeline.

To enhance the smoothness of the face from frame to frame, one could also experiment with using optical flow to track features in successive iterations. In our current pipeline, we get new features from the replacement video each frame. Although this simple method works very well, it does lead to choppy behavior near the edges of the face, where there is greater variability from frame to frame in the features obtained from Face Recognition than, for example, features on the nose and chin.

5. References

- Implementation (github): https://github.com/jandwally/nn-faceswap
- Face recognition API: https://github.com/ageitgey/face_recognition
- OpenCV: https://www.learnopencv.com/seamless-cloning-using-opencv-python-cpp/
- Outputs: https://drive.google.com/drive/folders/1YlOUKHAZaSfPDbe65d8O7F4aHfojCw2t