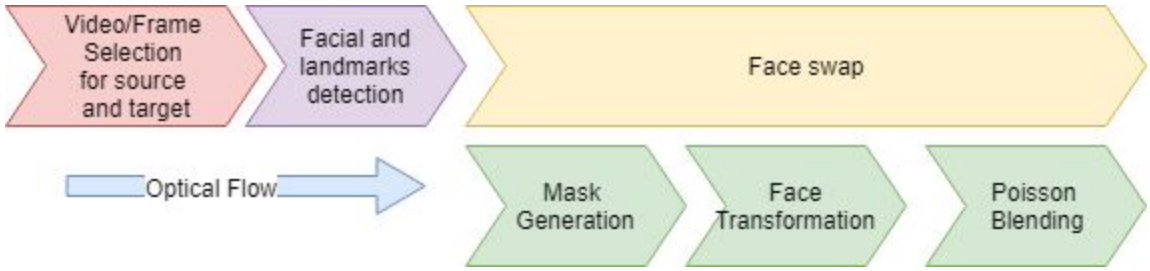
Face Swapping in Videos

Aishwarya Singh, Garvit Khandelwal, Venkata Sai Nikhil Thodupunuri
(Group 18)

1. Introduction

The aim of the work outlined here is to automatically detect faces in two given videos and swap them without the source face emoting the target face.
This was achieved by detecting faces and facial landmarks in the two videos. These facial landmarks were used to control the warp along the convex hull of the face. This was done for each frame of the target frame and motion changes had to be accounted for using optical flow techniques. The warp was blended as well to make the swap look seamless. This was done across all frames of the target image.

2. Methodology



Source and Target Selection

Face swapping was implemented on all the difficulty levels of data provided. The target videos used for face swapping were Frank Underwood's video from the Easy dataset, the Luciano Rosso videos from the Medium dataset, and the Joker as well as Leonardo DiCaprio videos from the Hard dataset.

For consistency, the source face was always kept as Elliot's face from the Mr. Robot video. The frame was selected manually such that it captured the best features of the source.

Face and Landmark Detection

Two different methods were explored for face and landmark detection. The primary method was *OpenFace*, a toolkit capable of face and facial landmark detection. Although *Openface* is capable of detecting multiple faces, we kept it restricted to just one face. Next, it extracts 68 landmarks on the face along the eyebrows, eyes, nose, mouth and chin. In videos where the eyes were big enough, *Openface* got us an additional 20 features within the eye (which is part

of its gaze tracking feature). *Openface* performed extremely well even in poor lighting and exposure conditions, as in the case of the Joker videos.

It functions by detecting faces and then extracting 68 landmarks using dlib's face landmark detectors. *Openface* also uses affine transformations to normalize faces and align them. An example of *Openface*'s landmark detection for our source frame is shown in Figure 1 below.

In cases of failure, as experienced in certain fast motion frames in the Luciano Rosso videos, *dlib*'s face landmark detector, get_frontal_face_detector was used by itself without *Openface*'s extra refinements.

Optical Flow

If features were missing in a successive frame, those landmarks were interpolated using optical flow estimation using the *Kanade-Lucas-Tomasi* method. We attempted to use *Spatial Pyramid Network* (spynet) for optical flow as well, but it was taking too long to process the video.

Face Swapping

Mask Generation

Once the the facial landmarks has been extracted out using *OpenFace* and *dlib*, we create a convex hull out of the facial landmarks using *cv2.convexHull* and generate face masks using *cv2.fillConvexPoly*. This results out in a binary mask for both the source and the target face. We also tried to use a *Position Map Regression Network* to obtain a dense 3D mask of the face across all frames. While this was giving good results, it was taking too long (~a minute for each frame), and hence we used the simple mask as described above.

Transformation

We then take source image and target masked images and apply Delaunay Triangulation using *scipy.spatial.Delaunay*. Triangles are constructed using the facial landmarks obtained in previous steps and then we do a affine transformation of the target face to source face triangle by triangle using *cv2.getAffineTransform* and *cv2.warpAffine*. This result in the source image swapped on to the target image while emoting the target image.

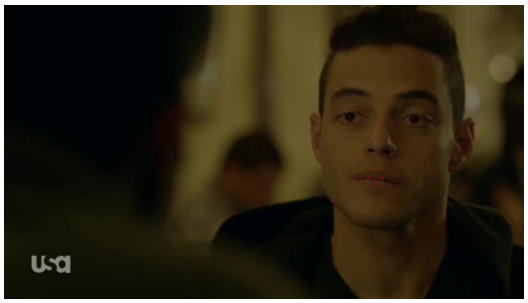
Refinement Using Blending

We experimented with two methods for blending the warped faces. We used a simple Poisson blending using *cv2.seamlessClone* between the frames with much better results than the *Gaussian-Poisson GAN (GP-GAN). GP-GAN* blends two high-resolution images by utilizing a joint optimization constrained by the gradient and color information. However, the results obtained from this weren't promising. Therefore we went with *cv2.seamlessClone* which provides gradient domain blending. So, once the target face is ready we seamlessly clone the target face mask with source.

To get better results we do not swap the mouth features of the target's face with the source's face as results with replacing mouth features were not visually appealing. The resultant face after performing all these steps has the source's face features with target face emotions intact.

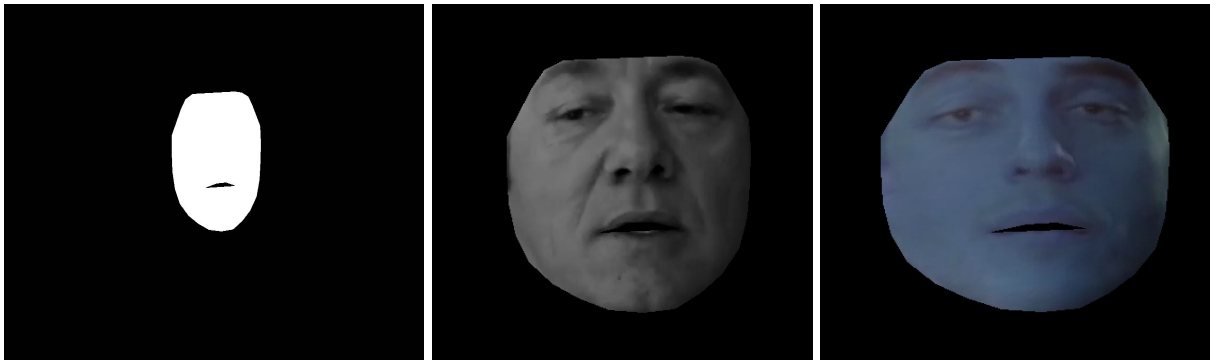
3. Results

Source Image with facial landmarks



Target Image with facial landmarks

Face swap without emotions



Mask Generated

Output

4. Libraries and Third party code used:

Libraries	*Third Party Code*
Dlib, OpenCV, Numpy, Scipy, PIL	*OpenFace, SpyNet, PR Net, GP-GAN (experimented)*

5. Future Work

1. Use of 3D mask as we got better results but were unable to do so because of resource constraints, therefore a possible work down the line can be using a dense 3D face mask obtained using PR net on a GPU.
2. Swap two faces in real time just like Snapchat filter.

References

1. *Amos, Brandon, and Bartosz Ludwiczuk and Satyanarayanan, Mahadev, OpenFace: A general-purpose face recognition library with mobile applications, 2016*
2. *https://docs.opencv.org/3.0-beta/index.html#*
3. *https://pypi.org/project/dlib/*
4. *https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.spatial.Delaunay.html*
5. *Wu, Huikai and Zheng, Shuai and Zhang, Junge and Huang, Kaiqi, GP-GAN: Towards Realistic High-Resolution Image Blending, aXiv preprint arXiv:1703.07195, 2017*
6. *Feng, Wu, Shoa, Wang, Lu, "Joint 3D Face Reconstruction and Dense Alignment with Position Map Regression Network", 15th European Conference on Computer Vision, 2018*
7. *Ranjan, Anurag and Black, Michael J, "Optical Flow Estimation Using a Spatial Pyramid Network", IEEE Conference on Computer Vision and Pattern Recognition, 2017*