Face Swap

Abstract

In this project, we used OpenCV framework to implement the detection and face swapping between two videos. By following the pipeline, we first used the dlib library to detect landmarks of human faces on both images. Then triangulation and affine transform is used to warp the face from source image to target image. Utilizing OpenCV built-in function seamlessClone, we implemented face swapping based on the warped triangles. Then, we used the OpenCV classes, VideoCapture and VideoWriter, to process the video frame-by-frame and apply our face swapping algorithm.

1. Facial Landmarks Detection

The first step of face swapping is to detect facial landmarks as corresponding points between two images. Here we implemented landmark detection with OpenCV dlib library. Since we used Google Colab, we only need to download the dataset and import dlib at the beginning. The output of the predictor will be the locations of 68 points representing the eyes, nose, mouth, etc. Each feature corresponds to several points in the predictor, thus we can find eyes in both images by simply specifying the index of the landmarks.

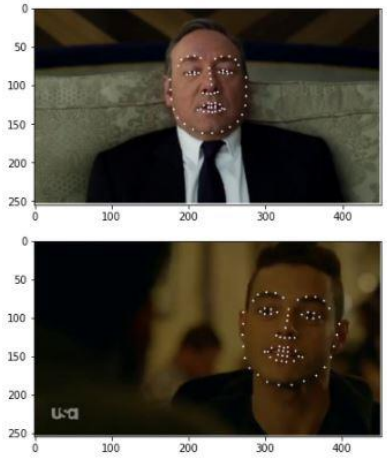

Fig. 1 landmarks on both images

2. Delaunay Triangulation

Since the faces of two images are of different shapes and sizes, it is impossible to copy and paste the face directly. Affine transformation is needed. Similar to project 5, we did Delaunay triangulation using OpenCV built-in method getTriangleList() first and warped the triangles instead of warping the whole face area. To avoid different meshing for two images, triangulation was only done for the source image and the corresponding vertices are connected as triangles in the target image.

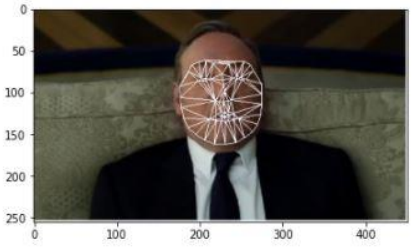

Fig.2 triangulation on source image

3. Affine Transform and Warp Triangles

In order for the source face to match the destination face, we have to warp the triangles of the source to match the target using affine transformation. To accomplish this, we used the OpenCV built-in functions cv2.getAffineTransform() and cv2.warpAffine. Then, to link all the triangles together and reconstruct the new face in the final image, we use the OpenCV functions cv2.bitwise_and(), cv2.bitwisePoly(), cv2.bitwise_not, and finally cv2.add().

4. Face Swap and Seamless Clone

The next step for face swapping is to cut the face mask from the source face and paste it to the target face. By applying masks on both images, we cut the face from the source image and get the background target image without a face. Unlike project 5 where we merged two images by cross dissolve, we used OpenCV built-in method seamlessClone() to implement the last step of combining two images seamlessly.

a) warped source image's face

b) target image without face

c) adding two images together

d) final result after seamlessClone

5. Video Processing

The final step for this project is applying our face swapping algorithm to the videos. First, we use the OpenCV built-in function to turn the source video into a VideoCapture object. This allows us to process the video frame-by-frame, which essentially turns the video into a series of images. Now that the video can be processed as images, our face swapping algorithm can be applied to each image with cv2.imwrite. After we process the images, we then save it as a video by creating an OpenCV VideoWriter object, which is the resulting video with deep fake.

6. References

1) dlib library: http://dlib.net/

2) OpenCV video processing: https://docs.opencv.org/master/dd/d43/tutorial_py_video_display.html

3) getAffineTransform: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_geometric_transformations/py_geometric_transformations.html
4) seamlessClone: https://docs.opencv.org/master/df/da0/group__photo__clone.html
5) Subdiv2D Class: https://docs.opencv.org/master/df/dbf/classcv_1_1Subdiv2D.html