# CS 512 COMPUTER VISION

Fall 2017

## Face Swapping

Akhil Suryadevara

Phattaraphol Suesorsit

## Introduction

- ➔ Face Swapping refers to a person’s face being swapped with the face of another person or an animal or with some other object.
- ➔ This started with photoshop in early 2000’s and then buzzfeed started publishing about face swaps with some striking examples in 2012.
- ➔ In 2016, Snapchat introduced face swap filters to it’s over 150 million users which led to a significant trend.
- ➔ Sometimes, failures of these apps results in “memes”.
- ➔ Some of the documentation is taken from Matthew Earl’s GitHub

## What Makes Implementing Face Swapping Difficult?

## This is easier!!

## This is what we want!

## How is that possible?

- ➔ Two Swedish Computer Vision Researchers Kazemi and Sullivan, created the One MilliSecond Face Alignment with Ensemble of regression Trees.
- ➔ They Developed a quick method to find the facial Landmark.
- ➔ This method is present in Dlib Library

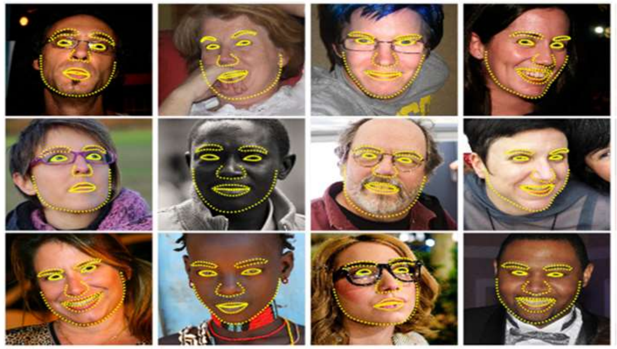
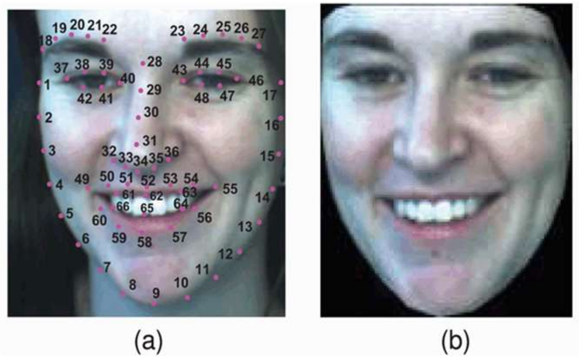


Figure 6. Final results on the HELEN database.

## Face Swapping

- ➔ **Problem**: Different face images will have their own different characteristics.
- ➔ **Steps to solve**
  - ◆ Extract the characteristics of faces
  - ◆ Adjust the translation, rotation, and scaling
  - ◆ Color Balancing
  - ◆ Create the mask to indicate the area to blend two images in
  - ◆ Modifying the result image

## Facial Landmarks



Install Dlib and Download the pre-trained model from [http://dlib.net/files/shape_predictor_68_face_landmarks.dat.bz2](http://dlib.net/files/shape_predictor_68_face_landmarks.dat.bz2)

Found by using detector() and predictor() functions

## Steps to solve

- ➔ Extract the characteristics of faces
  - ◆ Using dlib library to detect facial landmarks of both images
    - into two numpy arrays size 68x2 represents points of characteristics.
- ➔ Adjust the translation, rotation, and scaling
  - ◆ Translation and scaling: using ordinary procrustes analysis
    - Mean & SD
  - ◆ Rotation: singular-value decomposition
  - ◆ Construct the transformation matrix M to use in the affine transformation.

*By Kieff - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=11416486*

## Steps to solve

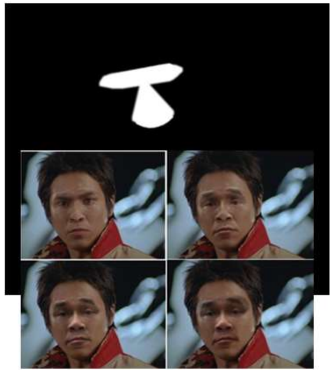
- ➔ Color Balancing
  - ◆ RGB scaling on specified area using cv2.GaussianBlur
    - “too small” kernel size: cv2.GaussianBlur could not make it to be the plain intensity
    - “too large” kernel size covers the excessive area and that affects the color balancing process.

## Steps to solve

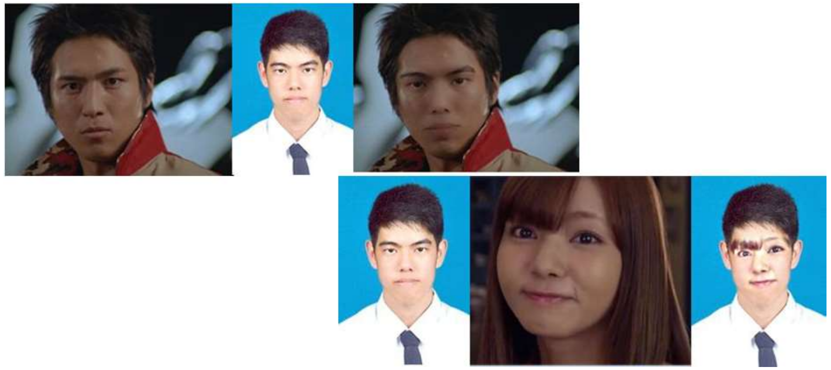
- ➔ Create the mask to indicate the area to blend two images in
  - ◆ making them a plain white area using cv2.fillConvexPoly
  - ◆ compare which to be blended by using function numpy.max
    - White on img2
    - Black on img1
    - Grayscale value along the edges

*output = img1 * (1-mask) + im2 * mask*

- ➔ Modifying the result image
  - ◆ Using trackbar
    - Adjust kernel size for color scaling



## Other (GOOD & BAD) results



## Live Video Swapping

- ➔ In Live video processing, The Live Video is face swapped with a filter(or other other person's face)
- ➔ To find the facial features we either use Dlib or Haar Cascade Classifiers
- ➔ We get the faster frame rate at the output with Haar Cascade Classifiers
- ➔ Performance wise they both are almost same
- ➔ We have Dlib On and Off option in our code

## What is HAAR Cascade?

- ➔ It's an edge detection method that inputs Haar Features into a series of classifiers to identify objects in the image
- ➔ They're trained to find one type of object
- ➔ But, we can use several of them in parallel e.g detecting eyes and face together

## They're Almost Same!

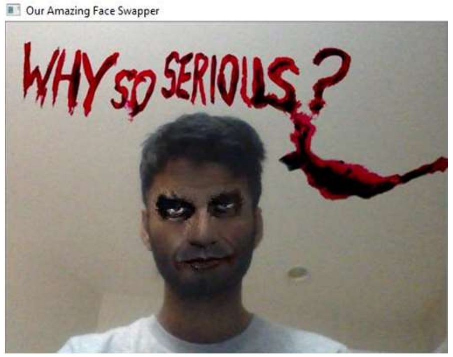
- ➔ In Picture Swapping, we swapped faces between two images

- ➔ In Live video Swapping, we are continuously capturing the live image and swapping them with selected filter if the face is detected in the live image.

## Let's see some examples!

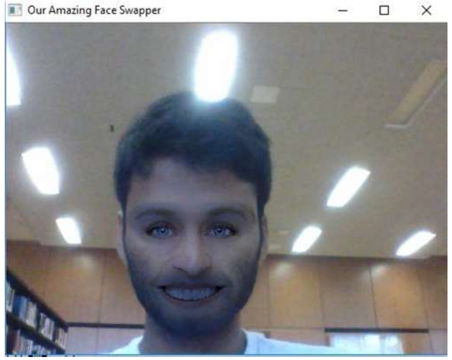
Input:

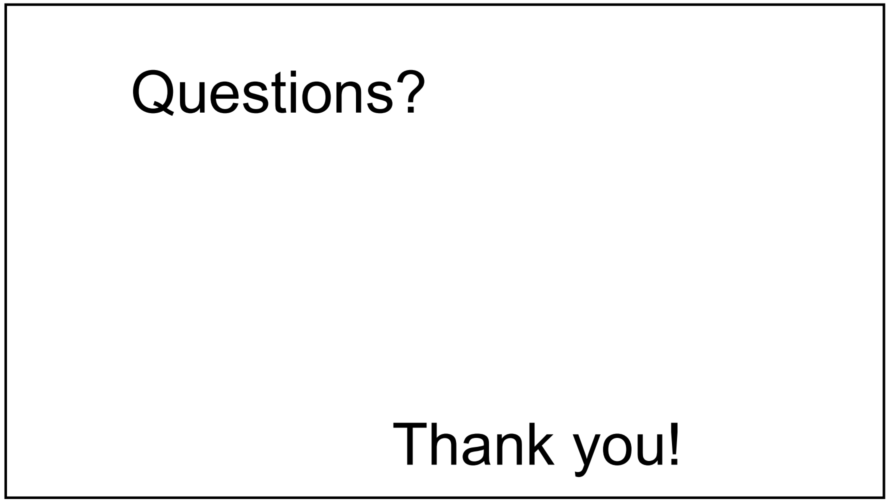
## Output



## Example 2 (Input)

## Output





Questions?

Thank you!