Snapchat like AR face filters

1 Introduction

The objective of this session is to work on an Augmented Reality application using several machine learning algorithms. You can get more information about lectures and associated practical sessions on my website: `http://chateaut.fr`. Essential needed knowledges include a beginning level of computer science (Linux, Python programming with `Sklearn`, `Numpy`, `Matplotlib`, `Opencv` and `Pytorch` (for sessions on Deep Learning)) and applied mathematics.

The application of this practical is to design an algorithm that superimpose a mask on a face (a snapchat like filter). The main parts of the method are illustrated on fig 1:

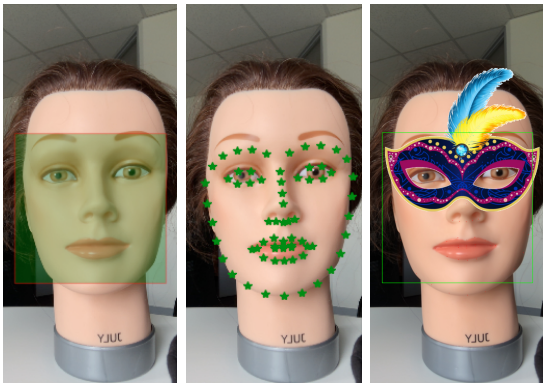

Figure 1: Illustration of the method. From left to right: detected face, 2D keypoints, result of AR

1. Detect faces into an image (using a HOG/SVM approach). [1]
2. Estimate face image keypoints using a regressor based method. [2]
3. Compute the position of the center of each eye.
4. Compute corresponding mask image by warping the original image
5. superimpose the warped mask with the input image and display

2 Face Detection

The first step consists in applying a face detector to the original image. The selected face detector used here is based on Dalal and Triggs work [1]. It is composed by three components : 1) a sliding windows strategy, 2) an histogram of gradient feature extractor

and 3) a linear support vector machine (SVM) classifier. We will use the python wrapper of `dlib` library[1] :

```
detector = dlib.get_frontal_face_detector()
```

This function defines the object `detector`.

```
dets = detector(img, 1)
```

This function returns the detections achieved on the image `img`. `dets` is a list of regions of interest detected. The i^{th} detection provides it position by:

```
dets[i].left(), dets[i].right(), dets[i].width(), dets[i].height()
```

You should work from the script `face_detection1.py`

1. Modify the script to initialize the detector and apply it to one image.
2. Test the algorithm on several images: which are the limitations?
3. Modify the script to display images with several faces (you may write a loop on `dets` using the syntax: `for k, d in enumerate(dets):` where k is a count and d is a detection). Test your script on the multiple faces images samples of directory `images`.
4. Test the detector for the images in the directory `images/orientation` and conclude on its limitations.

3 Keypoint Detection

This step estimates the position of face keypoints in the image. The method used is based on the work of [2]. this paper proposes a cascaded regressor to estimate keypoints in a coarse to fine strategy from an initial mean shape.

In `dlib` library, the keypoint regressor is initialized by:

```
predictor = dlib.shape_predictor(predictor_path)
```

with `predictor_path`: the `path+filename` of the `dat` file of the keypoints model learnt (`shape_predictor_68_face_landmarks.dat` for example) Getting keypoints from an image, the detection is achieved by:

```
keypoints = predictor(image, d)
```

This algorithm returns a list of keypoints (68 in the classic model) from which x and y coordinates are given by: `keypoint.part(0).x` and `keypoint.part(0).y` for the first point.

1. You have to save the file `face_detection1.py` to `landmark_detection1.py` and modify it to extract and display keypoints for an image containing only one face. You can use the function `plt.plot(x,y,'+r')` to display one point.
2. Test your script for several poses (directory `images/orientations`) and conclude on the limitations.

[1] Dlib is a computer vision and machine learning powerful library: `http://dlib.net`

4 Warping and creating the AR image

The center position for each eye can be computed from the keypoints (eg: mean position between left and right point for each eye). The figure 2 shows the position of each detected keypoint.

1. Compute the center position of each eye
2. The mask to be superimposed is the one of the file `mask.png`. The simplest way to warp this mask on the initial image is to apply an Affine Warping: ($x_2 = a_x.x_1 + b_x$ and $y_2 = a_2.y_1 + b_2$ with (x_1, y_1), the coordinates of a point into the mask image and (x_2, y_2) the coordinates of a point into the initial image. Here, the four parameters are estimated using both center of eye matching. The Warping will be applied using the opencv function : `dst = cv2.warpAffine(immask,M,(cols,rows))`. Matrix M is a 3×3 transformation matrix. The result of the warping should be an image of the same résolution of the input image with the mask positionned at the right place. Save or Display this image to check your code.

5 Superimpose

It's quite easy to superimpose the mask using the alpha channel as a binary mask on the warping mask image in order to overlay only the mask image pixels with alpha channel egal to one in the original image.

6 additional question

1. Using an homography transformation instead of a simple translation and scale one should be better. You have to manually link at least two more positions between the keypoints and the mask image. IT is then possible to compute the corresponding homography and to warp the mask image using this new transformation.
2. Modify the code to apply the mask on the video sequence provided by the webcam (using a resolution of 640x480).
3. To increase the processing time, it is possible to:
   - reduce the image size (ie: VGA using the opencv command:
     `ret = cap.set(cv2.CAP_PROP_FRAME_WIDTH,640)`
     `ret = cap.set(cv2.CAP_PROP_FRAME_HEIGHT,480)`
   - since the face detection is most time expensive part of the method, it is possible to swap, after the first image, to a tracking mode by initializing the keypoint detection directly from the previous image.

References

[1] Navneet Dalal and Bill Triggs. Histograms of Oriented Gradients for Human Detection. In Cordelia Schmid, Stefano Soatto, and Carlo Tomasi, editors, *International*

Conference on Computer Vision & Pattern Recognition (CVPR '05), volume 1, pages 886–893, San Diego, United States, June 2005. IEEE Computer Society.

[2] Vahid Kazemi and Josephine Sullivan. One millisecond face alignment with an ensemble of regression trees. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1867–1874, 2014.

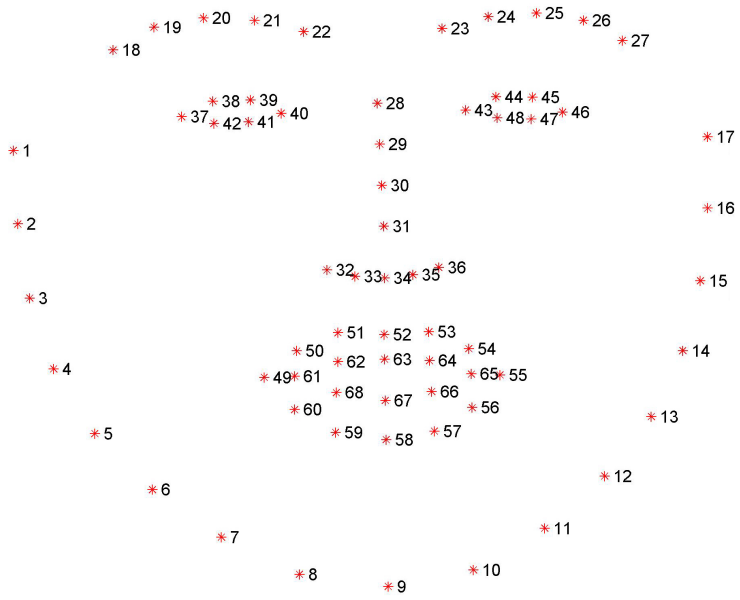

Figure 2: The widely used 68 points model