Face Detection and its Applications

5330 Final Project Report

Sida Zhang
Khoury College of Computer Sciences
Northeastern University
zhang.sid@northeastern.edu

Hongyu Wan
Khoury College of Computer Sciences
Northeastern University
wan.hongy@northeastern.edu

Xichen Liu
Khoury College of Computer Sciences
Northeastern University
liu.xic@northeastern.edu

Xiang Wang
Khoury College of Computer Sciences
Northeastern University
wang.xiang2@northeastern.edu

***Abstract*—Face recognition is an identification technology based on human facial feature information. It includes a series of related technologies that use cameras or cameras to collect images or video streams containing human faces, automatically detect and track human faces in the images, and then perform face recognition on the detected faces. In this project, we will be designing a computer vision app based on face detection techniques and exploring its possible applications.**

I. Introduction

Our final project is designed to be a computer vision-based facial detection app emphasizing feature extraction and analysis. With live camera inputs, we are able to detect faces and pull the features for functions including filters and characteristic transformation. It is a facial recognition app based on python. In this app, we mainly use live footage as data input, and users are able to use various filters to enrich their videos. During the live broadcast, it analyzes the screen, extracts possible facial features, identifies the user's face and its expressions, and extracts the user's facial data. With this data, our app will be able to match these features to our pre-designed filters and present users with rich visual effects.

II. Related work

A. Paper 1

Facial Feature Detection Library for Teaching Algorithm Basics in Python is a good source for our design. This paper is mainly a description of an approach how to teaching face detection algorithms with a face detection tool built in Python. After reading this, the audience will understand the related background and be ready to design and build a facial feature detection app with image processing skills. With the help of this paper, we were able to discover and learn the dlib package, and start off our design.

B. Paper 2

Apart from our design, there are still a lot of amazing applications that could be achieved by the techniques we used. *A Methodology for Automated Facial Expression Recognition Using Facial Landmarks* describes a valid and fascinating direction to explore.

As far as we know, the recognition of facial expressions accounts for 55% of verbal information and is considered an important part of psychology.

In traditional psychology, trained human observers are required to recognize changes in facial muscles and use facial movement coding systems to map muscle movements to emotions. While the system helps ensure objectivity and descriptiveness, its main disadvantage is the inability to effectively train human observers. With the advent of faster computers and the use of pixel/megapixel image elements, machine learning and computer vision have enabled effective and vivid descriptions of human faces. This paper describes how one can apply this knowledge to help develop and implement a related software engineering method.

C. Paper 3

Image equalization based on singular value decomposition

This paper proposes an image equalization technique based on singular value decomposition (SVD).

The method is called Singular Value Equalization (SVE) and is compared to the standard Grayscale Histogram Equalization (GHE) method. The visualization and quantitative results show that the proposed SVE method significantly outperforms the GHE method.

It provides important guidance and theoretical support for our program in identifying and manipulating facial features.

III. Methods

The manifestation of our project is an app developed in Python. It has four parts of functionalities, as the following first four subsections suggest. It includes a basic function for face detection, which is also the basis for almost all other

functions; a face modification function, which is an extended function from the detected faces in the first function; a face swap function, which will switch the faces shown in the live stream; a face matching face, where the user will be able to capture a frame and be matched with a most similar celebrity in our database based on CNN. The last part of the development of this app is the GUI. It supports a live video stream, which enables the users to manipulate and observe the above functions with a live camera.

A. Face Detection

Our face detection relies on the package dlib. The underly technique is to use the pre-trained HOG to compute a linear SVM face detector. The full name of HOG is a histogram of oriented gradients, which is the histogram of gradient directions. In 2005, it was first proposed by Dr. Mubarak Shah in the paper Histograms of Oriented Gradients for Human Detection at CVPR 2005. As a feature descriptor, the HOG feature is mainly used for object detection in computer vision and image processing.

The HOG feature is a feature formed by calculating and counting the gradient direction histogram of the local area of the image. HOG feature extraction divides the image into windows, blocks, cells, and bins. The window size is an integer multiple of the block, and the block slides with a fixed step size in the window; the block is an integer multiple of the unit, and the block can just fill the integer multiple units, and the gradient calculation is also performed in the unit; is the range of the horizontal axis in the histogram. Thanks to the Histogram of Oriented Gradients (HOG), the descriptor will not be influenced by the rotation of the face or the change of viewing angles.

After the detector is built, we still need the face to be highlighted so that we could actually use the data to do something. At this point, we will introduce shape predictor to our app. A predictor is a tool that takes in an image region containing some object and outputs a set of point locations that define the pose of the object. In other words, it shrinks the data into critical parts, displaying the eyes, outer shape, noses, and mouths in a few points to help us localize the face and save it for later uses. In our case, it is a pack of 68 feature points, displayed in the later experiments and result section.

B. Face Modifications

Given the feature points from the last part, we will be able to do some modifications to the faces in the scene at this point. The approach is to localize the target area by calculating the landmarks. For example, if we were to put on a pair of sunglasses, we would like to know where the eyes are. After using the landmark to identify the eye area, we could put the loaded sunglasses image to this image, and resized it as appropriate. The rotate angle is verified by changing steepness in the horizontal direction in the new projected points, and the image is rotated based on adding this angle times the resized height to the original pixel.

C. Face Swap

In this part, we first need to identify the target areas to be switched. They are cropped with the same measure we used in the face modification when we define the area for modification. Then, we would define an affine transformation where we solve the Procrustes problem by subtracting centroids, scaling by the standard deviation, and then using the SVD to calculate the rotation. After the affine transformation is computed, we use the wrap function to apply the affine transformation back to the mask, which will later be applied to the frame. After the mask is ready, we blur the color difference in the skin by applying a Gaussian blur under the wrapped affine transformation to try to make the skin look more natural. After all these, we would just simply substrate the modified mask from the original frame.

To be honest, we were to connect the points and use the natural angle calculated from the points to calculate the rotation, just like what we did in the face modification part. As it turns out, it didn't work very well because there are always issues with face sizes and edge problems, where the edges of the swapped face would show inside or outside the outline of the original faces. After that, we decided to use SVD for rotation calculation and only substitute the "T section", which includes the eyes, the nose, and the mouse. This avoids the issues and becomes our current version.

D. Face Matching

To implement the face matching to the most similar celebrity, we found the runtime of calculating the distance between the image of the user and the images in the database is larger than we expected. So we use the network structure to project the features into an embedding space. Embedding is a technique that reduces the number of dimensions of a datapoint.

We use the neural network model to project the image points into embedding space. In the neural network, after bunches of layers in a convolutional stack, we will have the points of an image in embedding space. The distance between images in embedding space will have the same proportions as their actual distances. Therefore, by calculating the distance between the image of the user and the faces in the database in the embedding space, the program can come up with the similarity between the faces. Thus, the matching functionality can display the most similar face from the database.

E. GUI

Python language has many GUI libraries that are easy to implement and effective. Because GUI is not the main part of our project, we choose to use the components in the Tkinter library with simple syntax but powerful functions to implement GUI. Another reason to choose Tkinter is that it has good compatibility with video stream display and image processing.

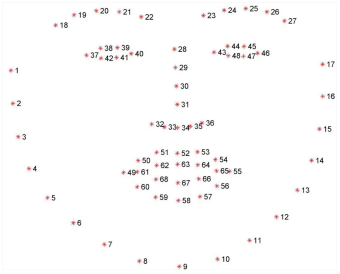
IV. Experiments and results

In this section, we will demonstrate the functionalities of the app and illustrate them through a list of screenshots.

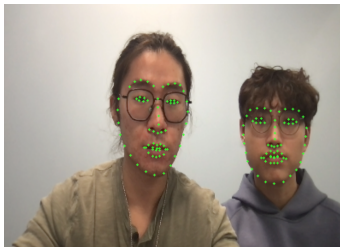
There will be five subsections in general, including face detection, face modifications, face swap, face matching, and GUI implementation in Python.

A. Face Detection

As the last section presented, our face detection is accomplished by detecting 68 feature points



The 68 points can be categorized into 4 parts. The first part is the exterior line that drew the outline of the face. The second part is the left and the right eyes. The third part is the nose and the fourth part is the mouth. The four parts are able to depict the general shape and characteristics of a person's facial appearance. However, there still exists some room for improvement:



This is a model for the 68 landmarks in their original form. By the comparison of the original model and our result, it is clear that these landmarks do not consider interfering factors including glasses, and if wearing glasses, it is easy to get the twisted resulting model from the algorithm.

This could become a part for us to work on after the project.

B. Face Modifications

The second part would be using the landmarks detected in the last part to implement the face modifications. We have two modifications implemented. One is to wear sunglasses and the second is to put on a clown mouth.

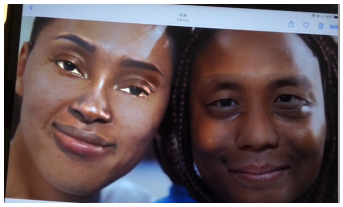
After the 68 landmarks are detected, we could treat the face just like the checkerboard in Project 4. It became a surface where we will be able to put on whatever we want. The objects will be localized by the landmark just like the projected points.

C. Face Swap

Face Swap is continued functionality based on the landmarks detected in the first section.

Below is the original image. Credit to shutterstock.com.

Below is a result for the face swap function in our app.



Another example:

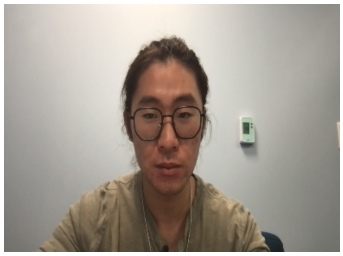
As the parts in the face swapped are only the eyes, noses, and the mouth, which is what we called "the T section", and the skin is adjusted with the Gaussian Blur to smooth the differences, we would see that this swap captured by a snapshot of our app would suggest the designed task is accomplished.

D. Matching

The last functionality of this app is facial matching. In the video stream, the user is able to capture the current frame and run it through our data to find the most similar celebrity. The result will be demonstrated by popping up the image of this particular celebrity.

Below is an example.

The original frame:



The matched image:

We do generally observe similarities in face shape and appearance between the original frame and the matched result in the behavior.

However, what cannot be ignored is there is no guarantee that the input frame and the output result will display gender unity. It actually makes sense because even though different gender usually displays certain different appearance characteristics, there is no evidence that certain gender will definitely own some particular appearance characteristics. We decided to leave it alone, as it is also an interesting add-on for this functionality.

E. GUI

We have a GUI implemented by Python. It is tested and recorded in the presentation, as the buttons and functions of a GUI will be better illustrated in a video.

F. Optimization

For the matching functionality, we ran into some running time issues. Our first design was to create an embedding space and perform matching after matching is triggered, but we soon realized that it takes a huge amount of time.

In order to optimize the user experiences, we have split the matching functionality into two parts to enhance the overall performance: creating embedding space to the databases, and enhancing performance on matching. The databases contain the 50 element vectors and the labels of images.

The second improvement was to improve the running time on matching. A couple of implementations on performance were done for this optimization. The first experiment was to unify datatypes' conversion between float and tensor when calculating the distance metric. Dataloader form PyTorch was first set to 4 threads and it dramatically affects the running time on our GUI, Tkinter; thus, we have set the thread to default. The last optimization was saving the 50 element vector of the target image to a list and avoiding re-calculating during distance metric calculation. The above three optimizations on the GUI significantly improved the overall running time from 120 seconds to 0.5650 seconds (about 20,168%)

V. Discussion and summary

Just like facial motion capture, or facial expression capture, in the 3-D movie industry, instead of using mechanical devices and equipment to record human facial expressions and movements, and convert them into a series of parameter data, we used a pack of 68 feature points to capture human faces in real-time. For some commonly used face libraries, the position of the face frames and the feature points of the face are often provided. Although all the 68 feature points are provided, it is not necessary to use all the feature points for facial alignment, for example, glasses and clown mask filters. Thus, understanding the detection order of the feature points can help us quickly find the specific feature points we needed.

References

[1] Dlib.net. 2022. Classes — dlib documentation. [online] Available at: http://dlib.net/python/index.html#dlib.shape_predictor [Accessed 3 May 2022].

[2] Dlib.net. 2022. dlib C++ Library. [online] Available at: http://dlib.net [Accessed 3 May 2022].

[3] Suhair, Y. and Kumar, R., 2022. Installing dlib with python 3.8 windows 10 error. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/59350831/installing-dlib-with-python-3-8-windows-10-error [Accessed 3 May 2022].

[4] GeeksforGeeks. 2022. Setting the position of TKinter labels - GeeksforGeeks. [online] Available at: https://www.geeksforgeeks.org/setting-the-position-of-tkinter-labels/ [Accessed 3 May 2022].

[5] GitHub. 2022. GitHub Selfie_Filters_OpenCV: This deep learning application can detect Facial Keypoints (15 unique points). They mark important areas of the face - the eyes, corners of the mouth, the nose, etc.. [online] Available at: https://github.com/acl21/Selfie_Filters_OpenCV [Accessed 3 May 2022].

[6] Josey, J. and Acharya, S., 2022. A Methodology for Automated Facial Expression Recognition Using Facial Landmarks. [online] Peer.asee.org. Available at: https://peer.asee.org/a-methodology-for-automated-facial-expression-recognition-using-facial-landmarks [Accessed 3 May 2022].

[7] Josey, J. and Acharya, S., 2022. A Methodology for Automated Facial Expression Recognition Using Facial Landmarks. [online] Peer.asee.org. Available at: https://peer.asee.org/a-methodology-for-automated-facial-expression-recognition-using-facial-landmarks [Accessed 3 May 2022].

[8] Kaggle.com. 2022. Famous people faces. [online] Available at: https://www.kaggle.com/datasets/caldodepollo/famous-people-faces [Accessed 3 May 2022].

[9] Matthewearl.github.io. 2022. Switching Eds: Face swapping with Python, dlib, and OpenCV - Matt's Ramblings. [online] Available at: https://matthewearl.github.io/2015/07/28/switching-eds-with-python/ [Accessed 3 May 2022].

[10] Medium. 2022. Glasses Detection - OpenCV & DLIB. [online] Available at: https://medium.com/mlearning-ai/glasses-detection-opencv-dlib-bf4cd50856da [Accessed 3 May 2022].

[11] Medium. 2022. Tutorial: Selfie Filters Using Deep Learning And OpenCV (Facial Landmarks Detection). [online] Available at: https://towardsdatascience.com/facial-keypoints-detection-deep-learning-737547f73515 [Accessed 3 May 2022].
[12] Python, R., 2022. Python GUI Programming With Tkinter – Real Python. [online] Realpython.com. Available at: https://realpython.com/python-gui-tkinter/ [Accessed 3 May 2022].
[13] Tutorialspoint.com. 2022. Python - Tkinter pack() Method. [online] Available at: https://www.tutorialspoint.com/python/tk_pack.htm [Accessed 3 May 2022].
[14] Ucar, M. and Hsieh, S., 2022. Board 137: MAKER: Facial Feature Detection Library for Teaching Algorithm Basics in Python. [online] Peer.asee.org. Available at: https://peer.asee.org/board-137-maker-facial-feature-detection-library-for-teaching-algorithm-basics-in-python [Accessed 3 May 2022].
[15] Oakley, B., 2022. Python tkinter: What are the correct values for the anchor option in the message widget?. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/41769497/python-tkinter-what-are-the-correct-values-for-the-anchor-option-in-the-message [Accessed 3 May 2022].