CSS 487 Final Project

REAL-TIME FACE DETECTION AND OVERLAY EFFECTS

December 11, 2020

Carl Howing
Max Nguyen

Introduction

With the rise of photo and video sharing social networking services, face detection has become one of the most popular and fastest growing fields in computer vision. Following the quarantine restriction in response to the COVID-19 pandemic, the emergence of teleconference applications, namely Zoom, has motivated us to dive into the concepts of face detection and augmented-reality filters for our final project. The following pictures show what we were trying to accomplish.

Snapchat Headband Filter

Project Objectives

Overview

Implementing computer vision concepts and OpenCV, our program accesses the user's webcam to implement face detections, then allows users to interact with the GUI to add a filter in real-time.

Project Description

Face Detection

Using the Haar-based detector provided by OpenCV, our program aims to detect the user's face in real-time by locating the frontal face region of the user.

First, converting the color camera image captured from the user's webcam to grayscale is mandatory for the face detection to work. Then prior to the detection process, the camera image is shrinked for faster processing. After that, histogram equalization is implemented to improve the contrast and brightness of the image for better detection in low-light environments.

After the image has been processed, face detection is executed with a green rectangle to mark the face of the user on the webcam frame.

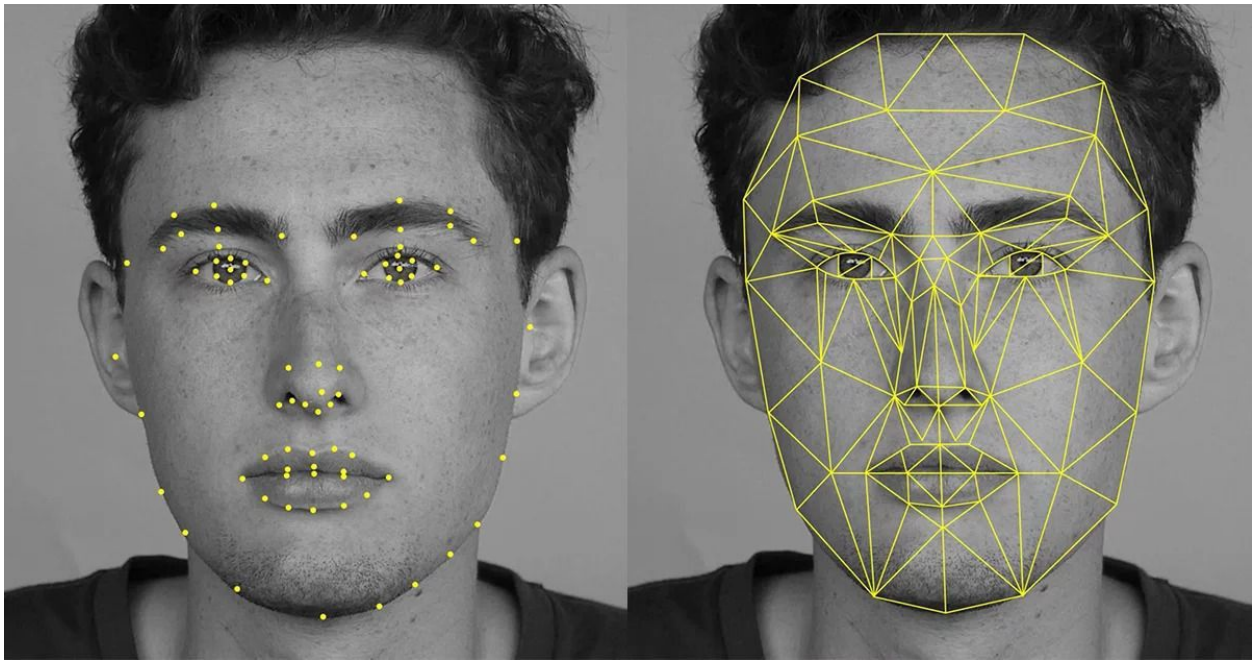
Eye Detection

The program's eye detection method is relatively similar to the face detection process. Using pre-trained XML detection provided by OpenCV "haarcascade_lefteye_2splits.xml" and "haarcascade_righteye_2splits.xml," red circles are used to mark the eyes detected. The detectors are able to detect both open and closed eyes, but not as effective for users wearing glasses.

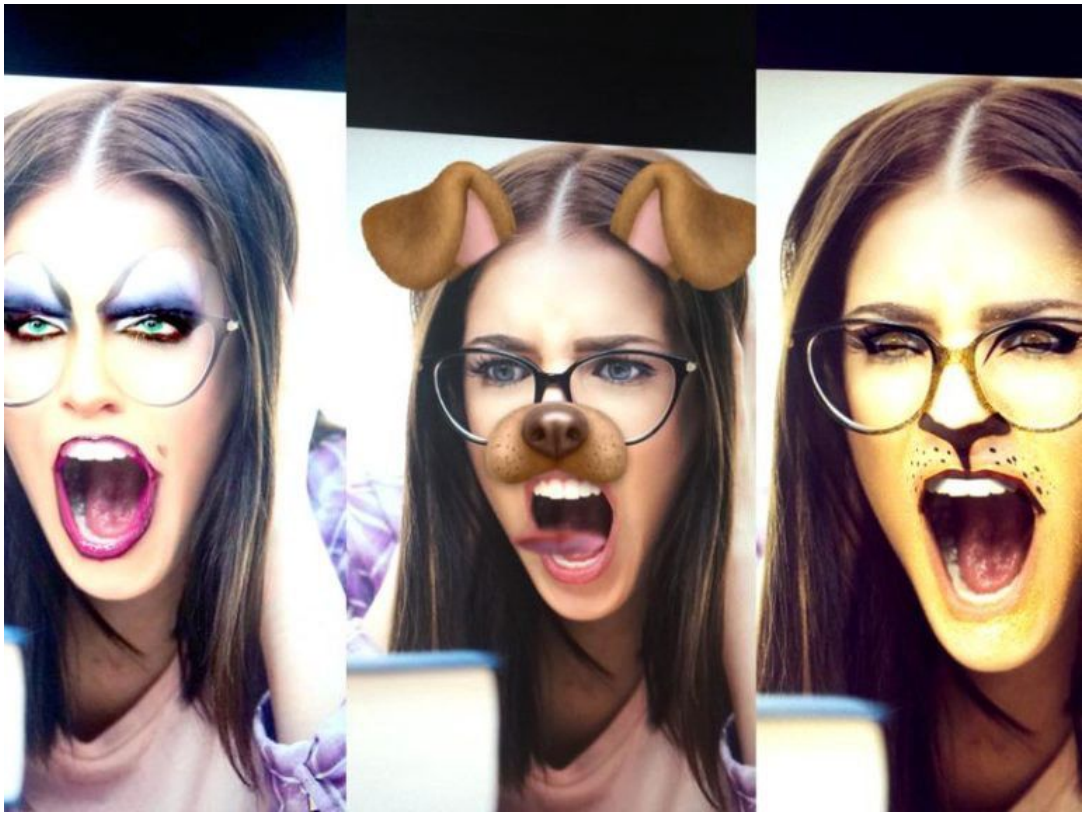
Snapchat-like Filters

The idea of snapchat filters is to detect the face and overlay an image on top of the location. Often these overlays are fun and exciting to the user. Figure below shows what the Snapchat application produces.

Snapchat Facial Landmark Mask



Snapchat Face Filter



Background Overlay

One of the features we also tried to implement was creating a virtual background. This was inspired by both Zoom and Snapchat. Both applications allow users to select the specific background which is overlaid to create an effect similar to a “green screen.”

GUI Elements

Designed to be user-friendly and interactive, our program’s GUI provides the user the ability to add or remove every overlay effect. Using OpenCV provided functions cv::putText() and cv::getTextSize(), buttons are provided on the webcam frame window in real-time for the user to interact with.

Project Accomplishments

Face Detection

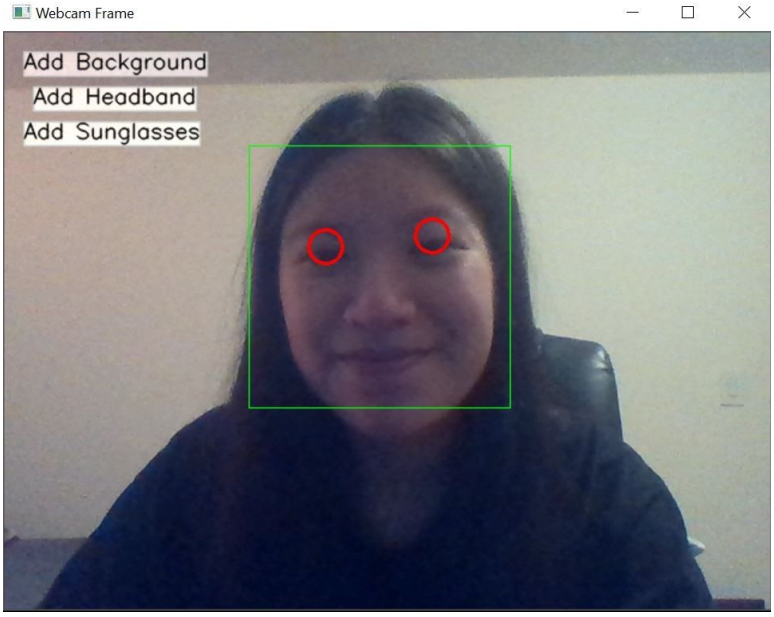
Our program is able to implement face detection relatively fast and effective. However, it currently does not work well if the user leans their head sideways.

Eye Detection

The program's eye detection does not work as efficiently as the face detection, possibly due the low reliability of the provided cascade classifiers.

We attempted to crop the face detected image into the top left corner for left eye detection and top right corner for right eye detection; however, the eye classifiers did not work with those cropped sections

We have also tried different eye classifiers, such as "haarcascade_eye_tree_eyeglasses.xml," but decided that it was not more reliable than before.

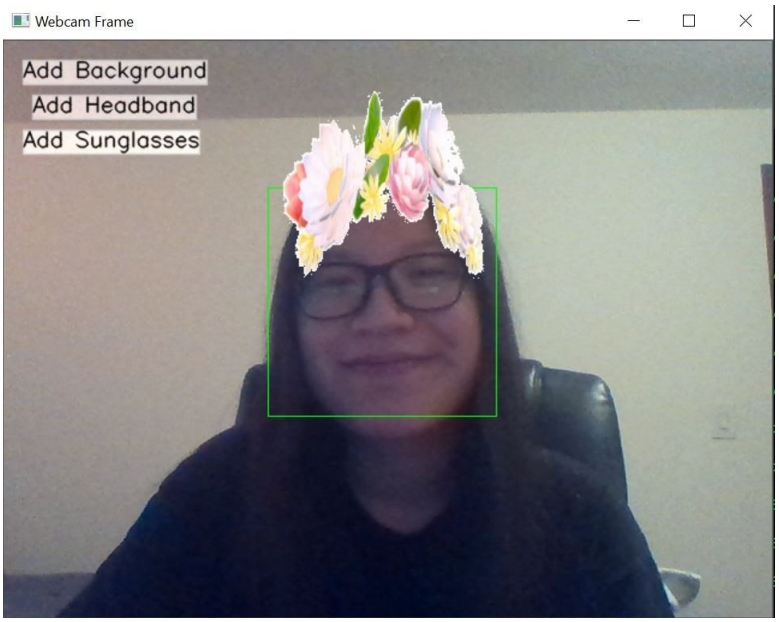


Snapchat-like Filters

Approaching this problem was a little hard for us, we had to do a lot of research to better understand what is required to make such features. Our team was able to achieve this using the haarcascade frontal face detector from openCV. Once the face is detected the detector provides

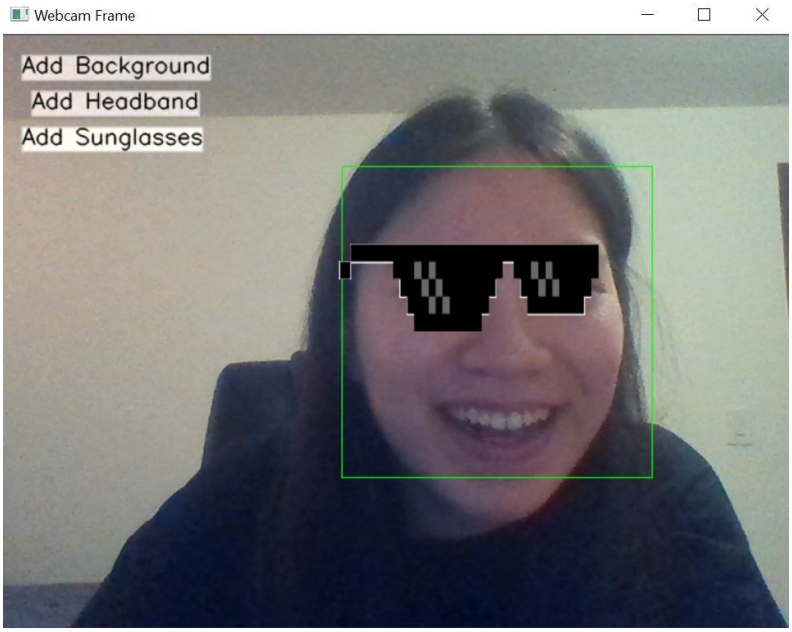
specific location information in the image. The information is stored in a “Rectangle” object which contains four variables, X, Y, height, and width. This information was used to scale the overlay image to ensure it fits the users face appropriately.

One of the issues we ran into was that the background was white and therefore overlayed the background onto the resulting image. This was not what we expected and we had to adjust our code to remove the white background. This was done using thresholding, we checked each pixel location to determine the values of Blue, Green, and Red (BGR). If the pixel was below a threshold value then we determined that it was part of the segmented image. This produced very good results. This feature only handles overlaying filters above the head.



The next step was to try and introduce eye detection, as stated above this proved to be very unreliable. Our original hypothesis was that because we scaled the image down for the face detection, the search space for the eyes was too small. We tried to combat this by scaling the searchspace back up and the results improved but was not very reliable using the real time camera feed. Our idea was that if we could locate the eyes using the haarcascade eye detector, then we can do a similar technique to the above the head filter. Upon further testing, this was deemed to be wrong.

We then tried to approach this problem using a different detector, the detector we looked at was the facial landmark detector which detects key points of the face. We initially ran into issues installing the contrib directory of openCV which contains this detector. Therefore, we were unable to use the facial landmarks to place filters over the eyes.



Lastly we were going to try to put overlays over the mouth. The figure below shows what we were trying to achieve but were unsuccessful because the detector was unreliable or unable to install in time to implement the feature. In order to overlay mouth fitlers we needed the face landmark detection to be working.

Background Overlay

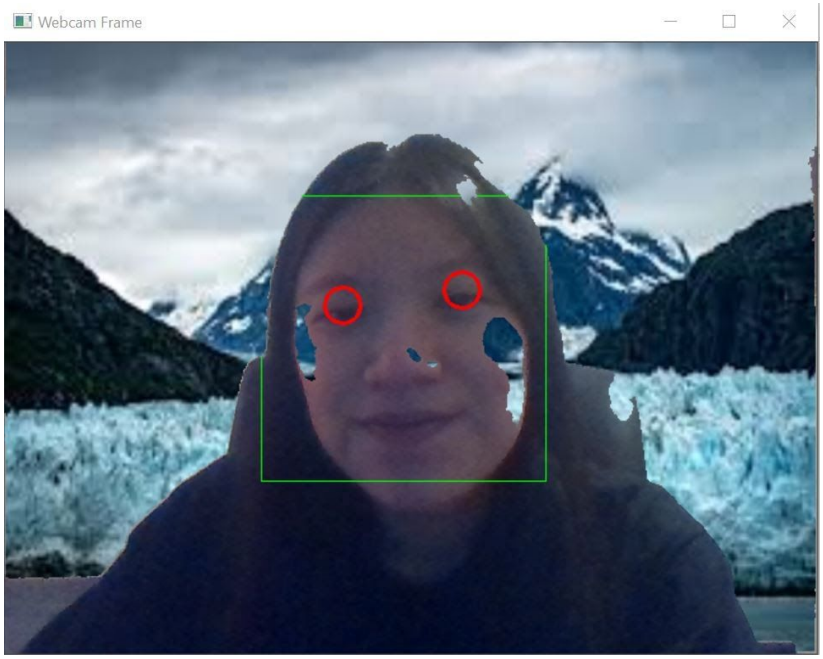
Our initial technique was to use the histogram of colors technique. This technique required us to loop through the image and calculate the most common color, then based on these statistics we thresholded the image. The results were not satisfactory to our liking as this did not work well in real world situations because different lighting and obstructing objects in the background behind the user.

The team decided to try doing some research on image segmentation techniques. The results we found looking specifically at how zoom and snapchat are able to segment the user was using machine learning models to segment the user. We decided to try a couple different techniques: one was trying to threshold the image using the start image as a grayscale. This resulted in a binary image that was used to determine which pixels were to be replaced with the background. This technique actually showed pretty good results, however it was not very reliable with changing lighting conditions and background. If the user had a complex background, for instance pictures on the wall, the edges of the pictures were shown on the resulting image.

The next technique was to use the same grayscale image, but use a histogram to determine what the threshold should be. Creating the histogram allowed us to dynamically threshold the

grayscale image, but the results were not much better. After further research, one technique that looked promising was the background subtraction method.

Background subtraction technique is used often in video surveillance, this technique works well in this scenario because the background remains somewhat constant. Therefore, any moving objects are shown. This technique uses the base background image as a baseline, and each frame is compared to the baseline. The difference between the baseline and each frame creates a mask and therefore new object's/motion can be detected (“How to Use Background Subtraction Methods”).This technique looked promising because the background in our case would remain somewhat static. However, this required that the background image be captured without the user in the image. We tested this approach by taking a background image then inserting a new object into the image, which should result in the mask showing only the new image. This proved to be not the case, because of noise the image had to be blurred then tested again, this resulted in some of the background objects being displayed in the mask. This proved to be an unreliable method to segment the user from the background. However we did implement these techniques to test the results, we even tried changing the threshold to determine what is different between the background static image and the newly changed image.



Performance of running in real-time

Due to multiple functionalities, the program’s performance in real-time is relatively low and laggy.

Project Challenges

Face recognition

Face recognition requires installing the opencv_contrib extra module; however, the installation process caused a big setback to our project due to its complications. We were not able to implement face recognition until it was close to the deadline. We also had to use an older version of OpenCV (4.1.1 instead of 4.5.0) in order for the extra module to work, so we had a hard time having to figure out the different formats between the two updates.

Training a model also took us a lot of time, and image databases took up a significant amount of space on our computers. Therefore, unfortunately, we did not have enough time to figure out how to load a trained model properly.

If we were able to have it installed, we would have included the following in our program:

- Use the CelebA dataset of celebrity images
- Create a CSV file containing the location of the images from the dataset
- Have the program read the CSV file and collect the data for training
- Use EigenFaceRecognizer algorithm
    - Initialize the model:
        - Ptr<EigenFaceRecognizer> model = EigenFaceRecognizer::create()
    - Train the model with the dataset and save the model
        - model->train(images, labels)
        - model->save("pretrained-model.yml")
    - Comment the two lines above as the model has been trained, and only load the trained model
    - Predict which celebrity looks most closely to the user's face:
        - int predictedLabel = model->predict(faceImg)

Snapchat Filters

We had some trouble finding Snapchat filters to use as most of them are not free to use.

Remote Learning

Conducting our project remotely had been challenging amidst a global pandemic that results in virtual communication and limited access to resources.

Lessons Learned

- A vast amount of functionality provided by the OpenCV package
- Machine learning can take up a great amount of time and storage space to train a model
    - Requires data sets to train and test
- Understanding OpenCV documentation is crucial

The followings are things we will do differently next time:

- Brainstorm more ideas as back-up in case a feature of the program fail
- Do more research during brainstorming week at the beginning
    - Researching more beforehand will result in better understanding of what needs to be done.
    - Narrows the scope of the project
- Do more testing in multiple lighting conditions and angles
- Image segmentation is a complex task for complex images.

References

Baggio, Daniel Lelis, et al. “Mastering OpenCV with Practical Computer Vision Projects.” 2012. www.cs.ccu.edu.tw/~damon/photo/,OpenCV/,Mastering_OpenCV.pdf.

“Drawing Functions.” OpenCV, docs.opencv.org/master/d6/d6e/group__imgproc__draw.html.

“Face Recognition with OpenCV.” OpenCV, docs.opencv.org/3.4/da/d60/tutorial_face_main.html.

Le, James. “Snapchat's Filters: How Computer Vision Recognizes Your Face.” *Medium*, Cracking The Data Science Interview, 22 July 2018, medium.com/cracking-the-data-science-interview/snapchats-filters-how-computer-vision-recognizes-your-face-9907d6904b91.

“How to Use Background Subtraction Methods.” *OpenCV*, docs.opencv.org/master/d1/dc5/tutorial_background_subtraction.html.

“Transparent Flower Crown Png Tumblr - Flower Crown Snapchat Filter, Png Download(979x592) - PngFind.” *PngFind.com*, www.pngfind.com/mpng/TixhbmJ_transparent-flower-crown-png-tumblr-flower-crown-snapchat/.