Computer Vision Project 4

Members of your group and group name:
Lauren Hunter
Guillermo Terrazas
Eduardo Zapata
Chris Metcalf
Max Timkovich

Final Report

Topic : Facial and gesture recognition, as well as eye tracking capabilities.

Overview: enumerate the final objectives of your project and key results that you achieved.

Objectives:

1. Recognize a person from a Webcam

Achieved Tasks :

- Our first task was to locate the person's face frame by frame, which we were able to do using openCV Haar cascade
- After we located the face in each frame, we create an image of the face
- Once we have that image, we use it to compare against our database of faces using an open source project called pyfaces: https://code.google.com/p/pyfaces/ (we describe how we use that project in background)
    - each frame produces an image, which is then used to compute a match in the database and returns the image filename
    - once we have 10 frames, we determine who the person is by finding the filename that was returned the most
- It will also add a person to the database if they aren't already in it

Challenges :

- If multiple people are in the view of the webcam, it will pick up all faces and use all the faces to identify a person, which will give skewed results. We couldn't really filter out the extra faces since we couldn't really define where the face would be, so we just make sure there isn't any one or images of faces (pictures of people) in the background.
- Another problem we ran into was all the images in the database had to be the same dimensions, which could be a problem since the distance of the person from the webcam could vary. We figured most people would be sitting in front of their laptops at about an arms distance, so we found the dimension that would best fit that distance.
- Lastly, we found that if a person is not in the database, it will still identify that person with someone. So we "fixed" that by prompting the user to be added to the database if he/she is not in it already.

2. Eye Tracking

Achieved Tasks :

- Our first task was to identify the limbus, the area where the color meets the white part of the eye. We used Haar cascades to single out a region of interest where the cascades detected eyes to be in the video. We then used Hough circle detection to find the pupil and iris within this region. The Hough circle function returned the x and y values of the centers of the circles it finds which translates to the center of the pupil in our case.
- Once we had the center of the pupil for each frame, we decided the best way to go about tracking eye movement was to analyze coordinate values to determine what direction the eye is moving. We are able to detect any change in direction by comparing the x and y values in the previous frame to the values of the current frame. We added an error threshold, so that minor changes in the pupil's x and y due to camera or head movement would not affect our tracking of the eyes.

Challenges :

- The Hough circles function picks up any circular shape in the frame so we had to reduce the noise by cropping each frame in order to improve the accuracy. We did this by using the opencv eyes Haar cascade function. This cropped the image around the eye area. We then ran the Hough circle function in this cropped image and received much better results.
- Even by using the eyes Haar cascade, we still had somewhat unreliable results, Hough circles always picked up at least one eye (an almost equal number of times for each eye), but rarely both. What we decided to do then was to only track the movement of one eye under the assumption that if one eye moves in a certain direction then the other does as well. This way we could disregard Hough's inconsistencies in finding only one eye.

3. Identify Facial gestures

Achieved Tasks :

- Building off of our facial recognition portion of the project, our first task was to build a database of eigenfaces that corresponded to different emotions. Our primary goal was to detect if someone had a neutral or happy expression. We were unable to consistently detect the difference.

Challenges :

- At first we attempted to use optical flow for detecting emotion. However we ran into the issue of accurately detecting the specific feature points to track. In order to track the movement of the feature in question, for instance a mouth, we need to have several points that outlined the mouth so that we could follow the movement. This proved to be a very challenging task as none of the methods we tried were accurate enough to

track at that small scale. Right now, we are able to track movements in the face but it would take more time to determine what the flow produces in terms of an emotion. To run this:
    - cd into emotion_recognition
    - python op_flow.py : this uses a webcam as the source of video, can be changed to use a video.
- Next we tried following the guidelines set out in the Stanford paper linked here. We used our previous code to recognize the face and were able to complete the pre-processing that the paper mentioned, going so far as to apply the mask to a normalized image and applying a Gabor filter. However we hit a roadblock once we tried applying PCA to our samples which halted us from making any further progress.
- Our last attempt we made was to create an "average" expression made from a set of faces. We used Pyfaces again to go through the dataset and create eigenfaces from each picture and an average. We crop the pictures in half in order to allow the algorithm to focus on only the mouth area and not other unnecessary features. We then used the matching algorithm to match a person from the webcam to the new set of average happy or average neutral. However we were unable to get the the matching to differentiate between different gestures. The matching algorithm was too strict and once it had matched to one picture, it would always match to that same picture again, regardless of change in gesture. It turned out that it cached results, which allowed to rely on past matches it wasn't able to match the face.

Background: Give a brief description of the state-of-the-art in the area of your project. If you referred to or used directly previous work (academic papers, open source code, data sets, etc), cite it fully and describe its relevance to your project.

1. Facial Recognition
   Facial Recognition today: this application is already in use today in the "real world", Facebook uses it to suggest people who to tag in photos! Some future applications are face scan for phones, facial recognition for advertising, face for currency? There are a lot of uses for facial recognition and the research in that area is pretty hot right now.

   We used a couple of sources for this part of the project:
   - This source helped with initially finding the face in a webcam https://realpython.com/blog/python/face-detection-in-python-using-a-webcam/
   - This source helped with understanding how eigenfaces works http://docs.opencv.org/modules/contrib/doc/facerec/facerec_tutorial.html#eigenfaces
   - This source is used to find the revelance of facial recognition today http://blog.ted.com/2013/10/17/the-future-of-facial-recognition-7-fascinating-facts/

- This is where we got our harrcascade to find a face https://code.google.com/p/ios-face-detection/source/browse/OpenCV-2.2.0/haarcascades/haarcascade_frontalface_alt2.xml?r=d35a62f475aa2813e4f3c80e50c33b7112389746
- This is the open source code we used to do most of the heavy lifting https://code.google.com/p/pyfaces/

pyfaces takes an image and a database, it creates a set of eigenfaces from the database.

Eigenfaces: based of the concept Principal Component Analysis (PCA) - a high-dimensional dataset is often described by correlated variables and therefore only a few meaningful dimensions account for most of the information. The PCA method finds the directions with the greatest variance in the data, called principal components.

The way this project creates eigenfaces is by taking the average of all the images in the database

-Then subtracts avg val from each orig val to get adjusted faces
-Then computes the Covariance Matrix by the dot product of the adjusted faces and the transpose of the adjusted faces
-Then from the matrix finds the eigenvalues and eigenvectors
-Then Order the eigenvectors descending by their eigenvalue
-Then finds the eigenfaces by performing the (dot product of the transpose of the adjusted faces and the eigenvectors $)^T$, the rows are the eigenfaces.
- Then creates these images

Then we take the images from the database and the given image and Project into the PCA subspace, then find the nearest neighbor between the projected training images and the projected image given.

2. Eye Tracking
   - The automotive, medical and defense industries have applied eye tracking technology to make safer products. The automotive industry evaluates the cognitive state of the driver in simulators and on the road in order to design safer cars. The medical field uses eye tracking to study surgeons during live procedures

in order to analyze best practices. The defense department has used eye tracking software to improve the way soldiers interact with computer-based systems.
- The fields of advertising and web design have benefited significantly from studying the eye behavior of consumers, giving them insight as to what practices work best based on what things consumers looked at more/less.
- In the medical field, people with ALS use eye trackers to communicate via virtual keys, to control computers, software and environmental devices.

3. Identify Facial gestures

   Facial gestures recognition have a large use by advertising/marketing companies. They often use emotion recognition software to determine the consumers reaction to a certain product.
   - This source was used as a guideline for implementing the facial expression recognition. We followed the steps listed in here in order to attempt to detect emotion from static images. http://cs229.stanford.edu/proj2009/AgrawalCosgriffMudur.pdf
   - Again this is the open source project used to do the heavy lifting. https://code.google.com/p/pyfaces/
   - This face database was used to create an average face for each expression. http://www.anefian.com/research/face_reco.htm

Methods: Describe the software and other components you created to achieve your objectives. What languages, libraries, hardware, and so forth did you employ? How did you structure your system? What algorithms did you apply or create?

1. Facial Recognition
   - We used the webcam in our laptops.
   - Pyfaces - Open source project used for creating Eigenfaces.
   - Python and OpenCV for the general face recognition and image manipulation
   - PCA - Main algorithm used to create Eigenfaces. At first we attempted to implement it ourselves, but then we ended up using the project above.
2. Eye Tracking
   - We used cell phone cameras to record videos of people moving their eyes.
   - We designed our own algorithm that compares coordinate values in order to determine any changes in direction.
3. Identify Facial gestures
   - Pyfaces - Open source project used for creating Eigenfaces.
   - Python and OpenCV for the general face recognition and image manipulation
   - PCA - Main algorithm used to create Eigenfaces. At first we attempted to implement it ourselves, but then we ended up using the project above.
   - we used Dense Optical Flow to track movements in the face, we were able to find the flow of the face, but ran out of time and were not able to identify which flows indicated what.

Results: Demonstrate concretely the objectives you completed with quantitative results. For example, if you planned to track faces with 90% accuracy, demonstrate your success rate by

measuring it and listing your actual success rate. Include screenshots, graphs, and videos (also checked in with your report) to prove what you achieved.

1. Facial Recognition
    - Our goal was to recognize faces in 75% of test cases.
    - We created unit tests to test pyfaces by passing in an image and a database. we used 4 different people and got 4/4 correct.
2. Eye Tracking
    - Our goal was to correctly identify the eye and its movement in 75% of the test cases.
    - We created a ground truth information for a test video of eyes moving that we found on YouTube. We sampled 20 frame from the video, and manually located where the center of the right pupil was in each frame.
    - We then ran our code, and printed out where the program determined the center of the right pupil to be at the sampled frames. The results are listed below.
    - As evident from the final chart, the absolute value difference between most of the x and y values for the location of the pupil is negligible (less than 5 pixels of difference). Only on frames 225 and 495 was there a large amount of error.
    - Thus, if we assume that 5 pixels is an acceptable error, we have unacceptable error in 10% of the frames. A larger sample size is necessary to narrow the true error rate, along with running tests measured over multiple videos and experiments. However, this gives us an indication that we achieved our stated goal.

Ground Truth

Frame Number	Eye X	Eye Y
0	234	126
45	230	126
90	231	126
135	230	126
180	248	126
225	264	126
270	265	126
315	266	126
360	251	126
405	247	109
450	248	126
495	248	140
540	248	126
585	248	126
630	248	126

675	248	126
720	248	126
765	248	126
810	248	126
855	248	126

Our Results

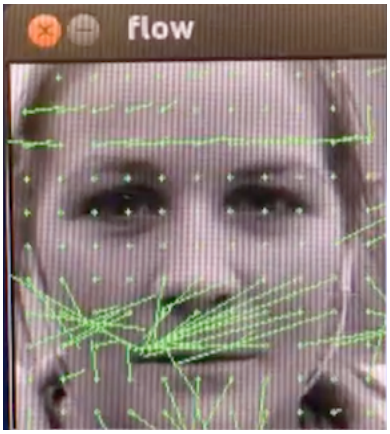
Frame Number	Eye X	Eye Y
45	229	124
90	230	127
135	230	124
180	244	124
225	264	171
270	264	124
315	261	124
360	249	124
405	248	110
450	246	123
495	263	138
540	248	126
585	248	126
630	248	124
675	248	125
720	247	124
765	244	124
855	246	124

Absolute Value of Difference between Ground Truth and Results

Frame Number	Difference X	Difference Y
0	5	2
45	1	2
90	1	1
135	0	2
180	4	2
225	0	45
270	1	2

315	5	2
360	2	2
405	1	1
450	2	3
495	15	2
540	0	0
585	0	0
630	0	2
675	0	1
720	1	2
765	4	2
810	2	2
855	2	2

3. Identify Facial gestures
    - Our goal was to identify gestures in 75% of the test cases.
    - We attempted three different ways to identify facial gestures. The first two attempts tried to classify emotion from static images. After those failed due to the complexity of using static images to classify emotions we moved on to using optical flow to find the movements in the face.
    - Attempt 1: We tried to use the steps listed in the paper here to implement emotion recognition. From the static image we applied some preprocessing, which included scaling the image, masking the relevant features, and changing it to grayscale. From there we applied a Gabor filter to extract the relevant features. We hit a roadblock when we tried to implement our own version of PCA. The complexity of the whole algorithm threw us off and we ended up trying a different route.
    - Attempt 2: We attempted to create a "average" expression based off of our previous work with Pyfaces. After curating a set of images with "happy" or "neutral" expressions, we applied the Pyfaces eigenfaces algorithm to them to create a set of images which we hoped to be able to match to another static image. What we did not realize is that these static images will not match to other expressive features very well, especially after combining all the features from multiple pictures.
    - Attempt 3: We used dense optical flow to track movements from the webcam, then focused just on the movements around the face. There is still a lot of noise but if we had more time could filter it out by averaging the flow vectors of a couple frames as well as blurring the image so it wouldn't pick up extra noise. We were also unable to establish neutral vs smiling because of the time limit. If we had more time, we would first define



two classes, neutral and smiling and then classify what types of flow those would have.