Final Project
COMP 3800-11
James Salvatore, Evan Kowaleski, Tom Haley, Kyle Kable

Overview

Our project is a system designed to notify a homeowner when a person approaches their front door. In order to implement such a system, our project will utilize real-time face detection and machine learning in order to both determine the presence of a person as well as his or her identity.

Selection of AI Methods & Tools

For this project, we utilized a guide for model selection and OpenCV setup [5]. Our goal was to improve data acquisition and incorporate the use of our own emailing algorithm upon facial recognition. Although the author of the guide includes information on Raspberry Pi usage with OpenCV, we had to take an entirely different approach due to a different Raspbian release on our PI and due to the fact that we were using a Camera Serial Interface (CSI) to capture a video stream instead of a USB Camera. These issues are detailed in the *AI Methods & Tools Discussion* section of the report.

A Raspberry Pi was used to host this project. We chose to use a Raspberry Pi because it offered a cost effective way to house the entire project. The source code is entirely written in python, which aided in our learning since it's what we have been using throughout the entirety of the class.

In order to detect the presence of a face in real-time, our project utilizes OpenCV. Because our project will be deployed on a Raspberry Pi we decided that it would be best to use a library that is widely supported and can run on many devices. OpenCV proved to be a good fit because it can run on any device that can run C code. In addition, OpenCV provides several pre-trained classifiers that can extract a variety of features—such as the eyes, face and mouth—to determine the presence of a face in each frame captured by OpenCV. Each of these classifiers are known as Haar Cascade classifiers, which are a combination of several Haar-like features that can be extracted from an image. A Haar-like feature is extracted by computing the difference in sums of adjacent rectangular regions at specific locations of an image. This difference in then used to categorize subsections of the image. After extracting these features from an image, these features are passed through a cascade of classifiers, in which an image is discarded if it fails any of the classifiers [1]. This prevents wasting computational time on images that will likely not meet the requirements for correct classification. The pre trained classifier we use is the Haar Cascade frontal-face classifier. To match a person's identity with a positive sample from the classifier, our project uses the LBPH (Local Binary Pattern Histograms) face recognizer provided by OpenCV.

Problem Specification

Camera System

The problem intended to be solved by our system is the recognition and identification of a person in realtime when he or she approaches your home. In order to perform this recognition, our system needs a camera and a means for processing each frame captured by the camera. The Raspberry Pi and PiCam module offers an embedded solution to address this problem, in which the camera can record each frame in it's designated space, while the Raspberry Pi can process each recorded frame. The program running on the Raspberry Pi is then responsible with deciding how to deal with each captured frame.

Data Acquisition

To acquire the data needed to train the model, the system needs the images that represent each class (person). Although these images can be supplied from external sources, the system provides a separate program called 'capture_data' for capturing faces of each person to be recognized. The program works by asking the user to provide an id and name of the person used to create an image class that will correspond to each of the captured images during the runtime of the program. The person to be classified stands in front of the PiCam, where each frame deemed to contain a face from the result of the Haar Cascade frontal-face classifier is stored in that class dataset. The number of faces captured for that person is specified by the user at runtime. The id and name for that class is stored in a JSON file that mimics the structure of a contacts list, such that each key is the id of that person, and the value is his or her name.

Model Training

Training the model for the system relies on the image datasets for each class. After the user has collected a desired number of images for each class, the model can be trained by running the 'train' program. The model is trained using the LBPH face recognizer provided by OpenCV. LBPH recognition works by creating several binary patterns for each pixel in the image. More specifically, each pixel of the image is compared to it's 8 closest neighbors in order to create a binary pattern for each of the neighboring pixels. Any neighboring pixel with a greater intensity than the center pixel (the pixel being inspected), results in placing a 1 in the binary pattern for that corresponding neighbor, otherwise a 0 is used. The result of the binary pattern is then converted to a decimal value. This process is repeated in order to create a new, textured image that better represents the characteristics of the original image. This new image is then divided into several 8x8 grids, in which a histogram is created for each grid. The histograms are then concatenated to create the final histogram that represents the image [2]. This histogram that can be used to determine the likelihood of a match (confidence) to a class trained by the model. The model is trained against each of the image classes created by the user, in which the resulting model is stored in a YML file.

Real Time Recognition

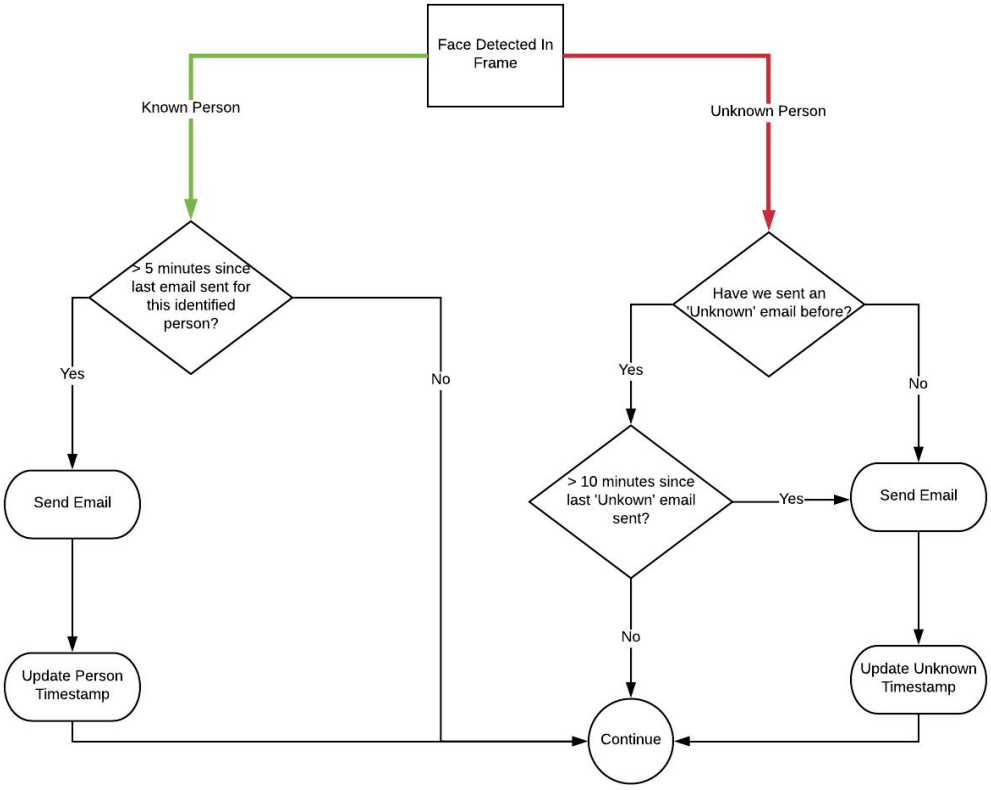
After the model has been trained, the user can run the main program called 'recognize' that performs the real time recognition for each frame that contains a face while the system is running. Each face that is extracted from the frame is passed into the model, where the model makes a prediction and returns it's confidence value for the person that is potentially identified. The confidence value is the distance to the closest item in the dataset that matches the face in the frame, where 0 corresponds to a

perfect match, with increasing values denoting less of an accurate match [3]. This value is subtracted from 100 to determine the actual 'probability' percentage that the face recognized corresponds to the class (person) being identified. Any percentage less than 1 is results in the face being deemed as 'Unknown.' In order to retrieve the name of the class that was identified, the contacts JSON file is used to look-up the name of that class (person) that corresponds to that classification's id.

Emailing Algorithm

The result of the model's prediction is used to send an email to the homeowner notifying him or her that a person is outside of their home. Figure 1 denotes the algorithm used to determine whether or not email should be sent when a face is detected in a captured frame. The algorithm makes its decision to send an email based upon the last time an email was sent for either the identified person, or anyone who was deemed 'Unknown.' The reason for this algorithm is to reduce the volume of emails sent to the homeowner, as failure to track the timing information could result in several emails being sent every couple of seconds. The time threshold for an identified person is 5 minutes, and 10 minutes for any 'Unknown' person. The email sent to the homeowner will that frame as an image attachment, where the title will contain the name of the person if the face was identifiable, otherwise 'Unknown.'

Figure 1. Emailing Algorithm



AI Methods & Tools Discussion

To deploy our project we decided to use a Raspberry Pi 3 Model B. By using this we benefit from its low-cost, portability, wifi capability, and its natively-run python environment. By using a Raspberry Pi 3 Model B we were able to quickly work in our desired environment because we were able to purchase one without the funding from the department. Some of the limitations, however, include the lack of memory and processing power. By opting for a smaller computer model we are sacrificing half the memory and processing power but given the scope of the project we weighed our options and found this to be the best method of deployment.

We utilized OpenCV in order to detect the presence of a face in real-time. OpenCV provides several pre-trained classifiers that can extract a variety of features—such as the eyes, face and mouth—to determine the presence of a face in each frame captured by the Pi. The pre trained classifier we use is the Haar Cascade frontal-face classifier. OpenCV also provide a set of recognizers. To match a person's identity with a positive sample from the classifier, our project uses the LBPH (Local Binary Pattern Histograms) face recognizer. Upon completing the project, we noticed some limitations to the classifier that was used. The confidence level dropped to about 30% when more than one person entered the frame. The lighting was also a major constraint of this project, poor lighting or shadows would also result in a confidence level at about 20-30%, and sometimes not even recognize that a person was in front of the camera at all. However, we were able to achieve a confidence value between 70-80% when a single person was in front of the camera. This confidence value would decrease as the person stepped farther away from the camera.

An issue that was encountered was we were not able to open a video stream using the opencv(cv2) library. This is because the VideoCapture method from the opencv library only supports USB camera devices and does not support camera serial interfaces(CSI). A first attempt at solving this problem was loading the bcm2835-v4l2 camera drivers as kernel module using modprobe. By doing this we were able to open the video stream using the built in VideoCapture method, however, whenever we tried to capture a frame the stream would close after one or two frames and would have to be re-opened to be able to capture images again. To solve this we switched the video stream capturing to be handled by the picamera library and do the image processing with opencv.

Method & Tools Questionnaire

The tool that was used to implement this project was OpenCV, which stands for open source computer vision. OpenCV is essentially a library of functions to perform real time computer vision. OpenCV offers many different advantages over using other tools like Matlab. First, Matlab is built on Java, which is then built upon C. This means that at runtime your computer has to interpret all of the matlab code, convert it into Java and then finally execute the code, while OpenCV is written in C/C++. This essentially cuts out the middle step of converting the code, resulting in fast runtimes while using OpenCV. Another advantage that OpenCV has over Matlab is resource allocation.[4]

Summary

Upon conclusion of this project, we found it difficult to achieve a confidence level over 80%. When trying to raise the confidence level of the project, we expand the dataset to over 1,000 pictures for each person, however this resulted in the live feed of the camera to freeze and basically stop functioning all together. So in order to achieve a confidence level that was roughly over 50% on average we used approximately 200 pictures per person. In reality, viewing the live feed of the camera is not necessary, but the classifier would still have to read through the model file of histograms so performance would still be negatively impacted when using a large number of images per person.

Sources:

[1] Face Detection using Haar Cascades
https://docs.opencv.org/3.4.1/d7/d8b/tutorial_py_face_detection.html
[2] Face Recognition: Understanding LBPH Algorithm.
https://towardsdatascience.com/face-recognition-how-lbph-works-90ec258c3d6b
[3] OpenCV face recognition prediction and confidence values.
http://answers.opencv.org/question/175195/opencv-face-recognition-prediction-and-confidence-values/
[4] OpenCV performance information
https://www.learnopencv.com/opencv-c-vs-python-vs-matlab-for-computer-vision/
[5] Real-time Face Recognition: An End-To-End Project
https://www.hackster.io/mjrobot/real-time-face-recognition-an-end-to-end-project-a10826