Final Report on Camera Attendance-Taking System

Daniel Carr

Introduction

Taking attendance certainly has its values: it tracks student engagement, progress, and how much they've learned. However, taking attendance takes a long time. If taking attendance takes two minutes per class, three classes per day, five days a week, forty weeks a year for the 117 teachers at TJ, then a total of 2340 hours a year are spent on taking attendance. While attendance is important, this time could be used for other more productive purposes, like class time, review, questions, or quizes. One solution is to create an automated system to take attendance. A camera would be connected to the teacher's computer and take video of students walking in. The student's faces would be analysed and machine learning would be applied to determine which students are present and which aren't.

Previous Work

Last year, Sagar Saxena took on this challenge and laid down a strong foundation for the project. He created a strong system which used several packages in order to accomplish this task, including Matlab, Keras, SQLite3, and a CMU-built face recognition tool. Throughout the year, I worked to fix bugs and create more warning and checks throughout the program.

The first step in taking attendance is receiving the input. A recorded video or live shots can be inputted into the system. Once it has the video, the software can move onto detecting and tracking faces. In order to detect faces, accuracy was prioritized over speed. CMU's Tiny Face Detection algorithm is a robust detection algorithm which is capable of detecting multiple low resolution faces. Tracking is maintained with the Hungarian algorithm. This takes two shapes

and divides the overlapping area over the total area, meaning that close shapes are tracked and separate ones return zero.

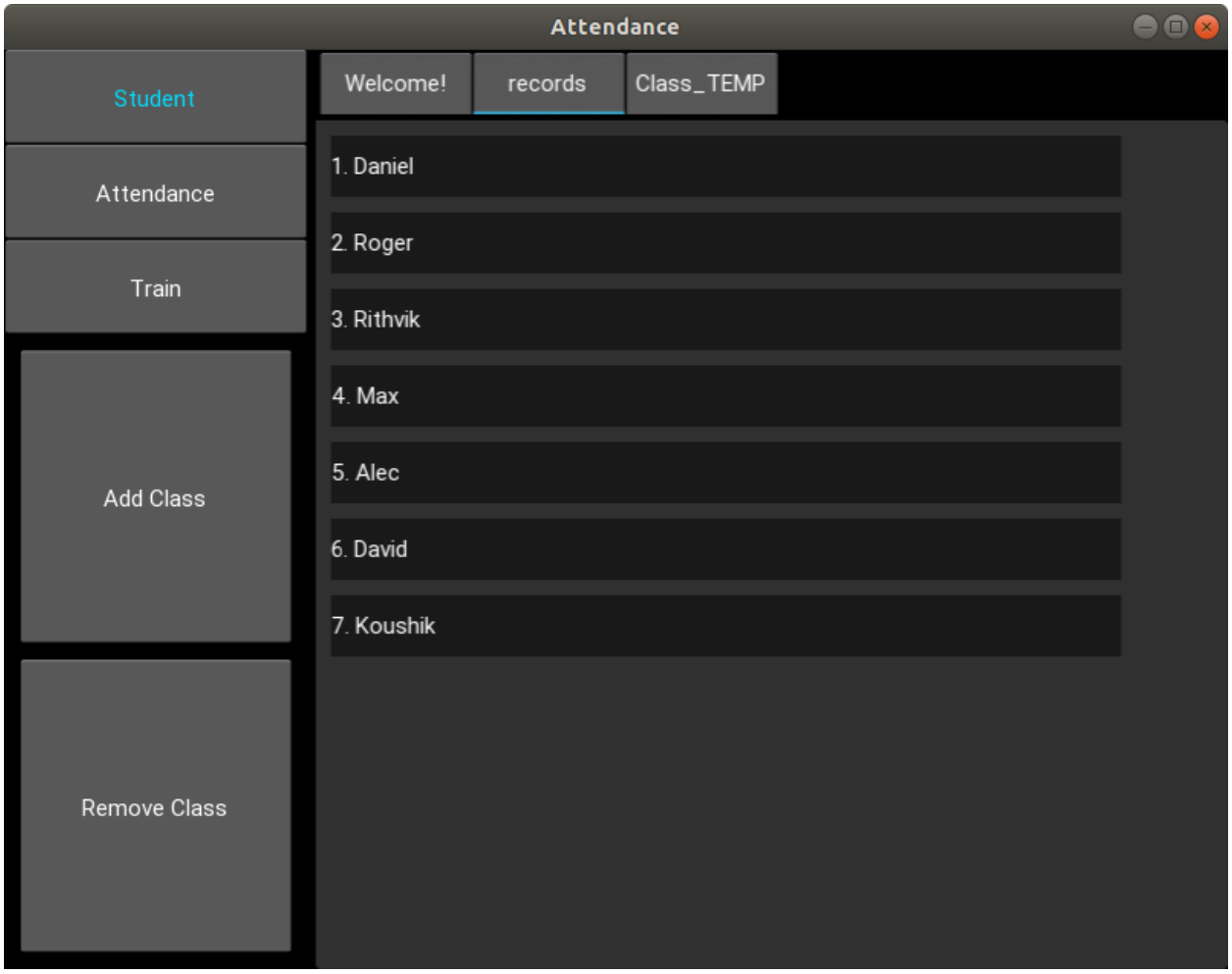
Faces are then cropped and stored in a database along with other information, such as room location, time, student ID, and their generated tracking ID in the previous step. In order to create data, users would manually tag a student to their face to associate the images with student ID. Then, using the data, the user can train a neural network in order to create associations between images and students. This system uses a Convolutional Neural Network (CNN), a network commonly used with visual analysis. It uses a model similar to AlexNet, an award-winning object-classification network with high accuracy. The network used here has been scaled down to use eight layers and a smaller input as AlexNet includes over 1000 layers and much larger input images.

After the network is trained, users can use it to classify students faces and take attendance. One advantage to using a neural network is that it can adapt over the year and match the students' slowly changing faces. The system ensures that these changes will be accounted for throughout the year for each student.

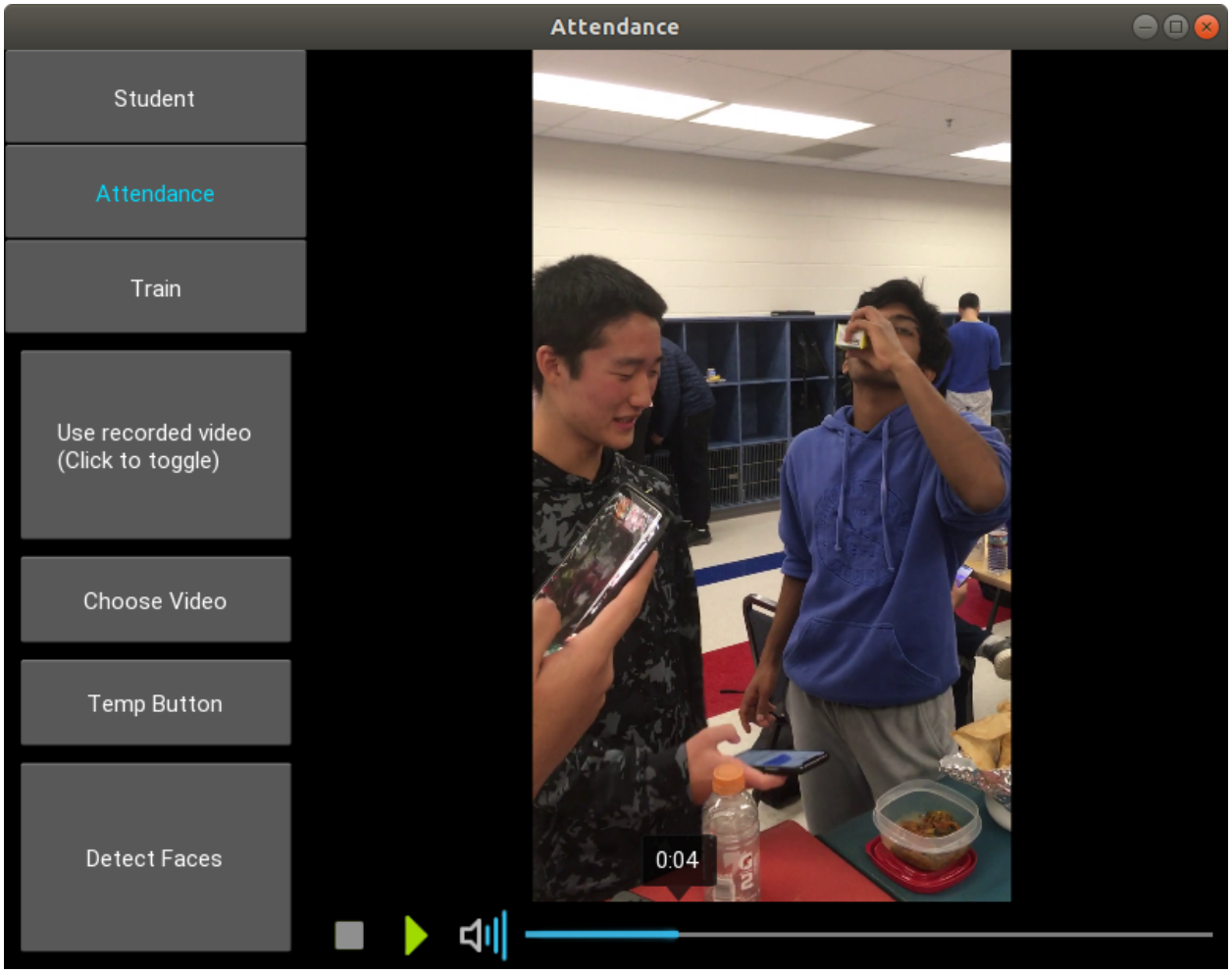
User Interface

I focused primarily on developing the UI so that the system would be more accessible. I used Kivy to create the UI because it was Python compatible, fairly accessible, and had a strong community behind it. Upon using it for a while, I found that there were some really specific features it was missing, like the ability to release cameras, but didn't run into too many problems. Kivy uses a system with Python objects and a .kv file in order to specify objects attributes and positions. I created and nested countless layouts and widgets.

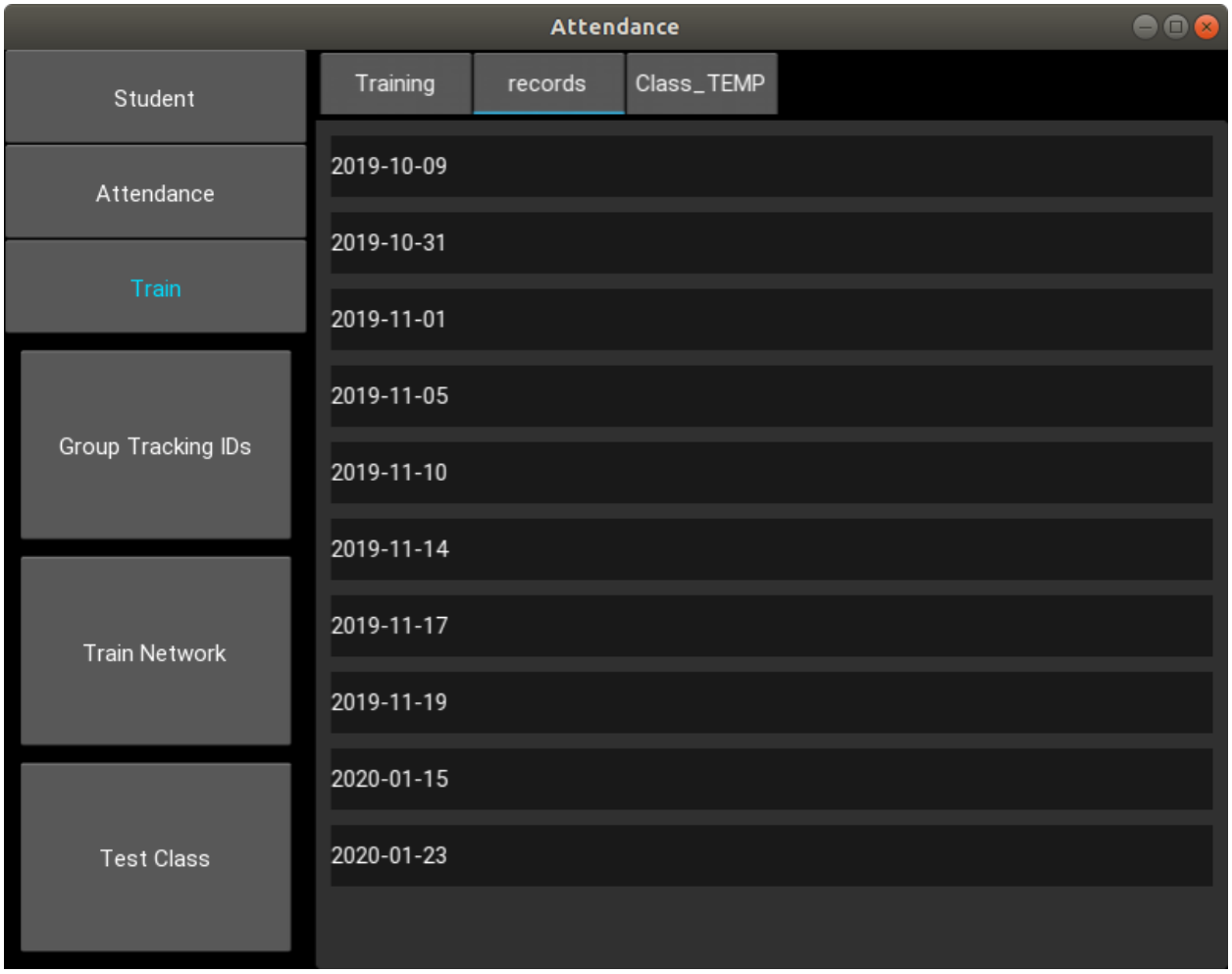
In order to hold all of the features that the system originally had and to make the program more organized, I divided the UI into three primary tabs: student, attendance, and training. Below, the tabs can be changed on the left and the buttons on the bottom-left are changed between tabs. The main area is on the center-right and includes the primary content of each tab.



The student tab is where users can manage classes and find students. Each tab at the top represents a class, for example, “First Period” or “Second Period,” and has their own database. The names “records” and “Class_TEMP” here were just the existing class names during development. Under each tab, a scrollable list of students will appear. Currently, the students are selectable, but doing so doesn’t prompt anything yet; this could easily be changed in the future.



The attendance tab is used primarily to detect students from video. Users can upload a video, which will prompt a file selector and change the video on the right. Alternatively, the users can toggle to live camera, which will use either the built-in or an external camera. Once the user wants to analyze a video, the user can click the “Detect Faces” button. This opens a dialogue which allows the user to input parameters, automatically filled with suggestions, and then runs the detection algorithm in a separate thread. The video screen will change to reflect output from the algorithm in order to keep the user updated on what’s happening. After the process is finished, the data is saved in the database for the corresponding class.



The training tab is used for training the neural network. Each tab on the right corresponds to the classes and contains a list of dates below for when data was collected on the day. Clicking a date will open a pop-up with information about the data collected on that day. Each tracked group of images will show up and a button below will allow the user to set a face to a student, or if it is in artifact, to the “Unidentified” setting. After a user creates the training data, they can click the “Train Network” button to input a few parameters, including the day to train, to further develop the network. Once a network is accurate, the user can use the “Test Class” button to check a day’s data and the system will automatically guess who’s who with a confidence number to the side. In testing, I saw accuracy rates around 80 to 90 percent.

Future Work

Through my work on this project, I made it my goal to increase the project's accessibility and ease-of-use. While considerable work was finished on the front-end, there is still much left to do behind the scenes, including speed-ups. Currently, detecting faces can take from twenty seconds up to an hour per second of video, depending on hardware. This was partially solved by not processing every frame, and instead, doing every two or three. However, skipping too many frames can make the tracking in the system inaccurate and can potentially miss students entering in some edge cases. Another solution might be to scale down the Tiny Face Detector or to switch to a separate algorithm.

Additionally, the Python-Matlab Integration slowed down the system a lot. The Tiny Face Detector uses Matlab and in it, each image is being passed individually. This means that each time numerous constants need to be reinstantiated which takes unnecessary processing power. If the connection could be strengthened or switched, the system could stand to benefit considerably.

Lastly, privacy is a major concern in regards to a project like this. Student and researcher consent was obtained throughout this project, but future researchers should maintain this ethical standard as well. In the system, student images and information should be encrypted to protect privacy. Images could also be removed if not needed anymore or after long periods of time. Implementing the encryption of data would also prepare the project for server-side scaling and possible cloud computing. In addition to encrypting data, users should also obtain user consent throughout the project and implementation of the system, and ensure that students can easily opt-out and not have images of them saved in the database.

Future Considerations

There are still many things left to do, such as the speed-ups mentioned previously. UI-wise, there are also many areas that could be further built on. Rebasing the code so that it is more organized would be useful to future developers, though I haven't looked in-depth into doing so. Additionally, outputting the attendance results to the "Students" tab if the system believes that the confidence is above a certain threshold would be useful, or outputting a json. Showing images as feedback and changing running from dialogue options to more interactive options, such as selecting multiple dates in the "Train" tab are some examples of ways to improve the system. Additionally, there are several bugs still left to be fixed. A program-breaking one is in the face detection algorithm where the program errors if there are no faces on the frame. Other spots in the UI also have ideas that couldn't be implemented in the given time frame, such as the class creation through a text file.

Useful Resources

Kivy: https://kivy.org/#home

Getting Matlab (TJ Students): https://user.tjhsst.edu/get_matlab

Installing Matlab: https://www.mathworks.com/help/matlab/matlab-engine-for-python.html

Sagar's Paper (more back-end details):

https://user.tjhsst.edu/2019ssaxena/documents/attendance_paper.pdf

Sagar's Links: https://user.tjhsst.edu/2019ssaxena/documents/attendance_resources.pdf

Github: https://github.com/randomerz/attendance