Facial Recognition Attendance Tracker Documentation

Release 1.0.0

Samrat Sahoo

Jun 18, 2020

Contents

CHAPTER 1

Table of Contents

1.1 Setup

Setting up the attendance tracker is very easy!

- Step 1: Clone the repository from GitHub using `git clone https://github.com/SamratSahoo/Facial-Recognition-Attendance-Tracker.git`
- Step 2: Make sure you have Python 3.6 or Higher installed
- Step 3: Run `pip install -r reqs.txt`

To Run the application, you can use `python Interface.py`

1.2 Directories

1.2.1 Folder List

Cascades

This is the Cascades Directory! Within the Cascades Directory, you will notice another folder known as data. There are files within here that are named such as `haarcascade_eye.xml`. These files are known as Haar Cascades! Haar Cascades are most notably used for facial detection through finding key features within the face. You will see arbitrary values within the cascades such as `-1.2550230026245117e+00` which are computer generated values based on the features of the faces They were previously used in original iterations of this facial recognition project, however, because of the integration of face detection with the face_recognition library, these are no longer used. These files while not used are kept as archives of previous works and to explain the facial recognition process.

Naming Conventions

Thera are several cascades within the Cascades folder. These cascades all detect different features of the face. For example the `haarcascade_eye.xml` detects eyes in faces. The `haarcascade_frontalface_alt.xml` detects the face. The `haarcascade_frontalface_alt2.xml` also detects the face however it is just another version of the original haar cascade that was created.

Docs

The `docs` folder has one sole purpose: to house the documentation of this project. The HTML files that make up the documentation can be found in the Docs folder. Within the `docs` folder there are 2 subdirectories along with 2 batch files.

Build Subdirectory

The `build` subdirectory houses all of the automatically generated HTML files for this project. This subdirectory is automatically updated when the `make html` command is run.

Source Subdirectory

The `source` subdirectory houses all of the RST files for this project. The RST files are very similar to markdown because they allow for an ease of creating web-based documentation through the ReadTheDocs system without long hours of web development.

Batch Files

The batch files serve as the method to convert the RST files to HTML files. When the `make html` command is run, these batch files scan the RST files and make the respective HTML files based on RST files.

Encodings

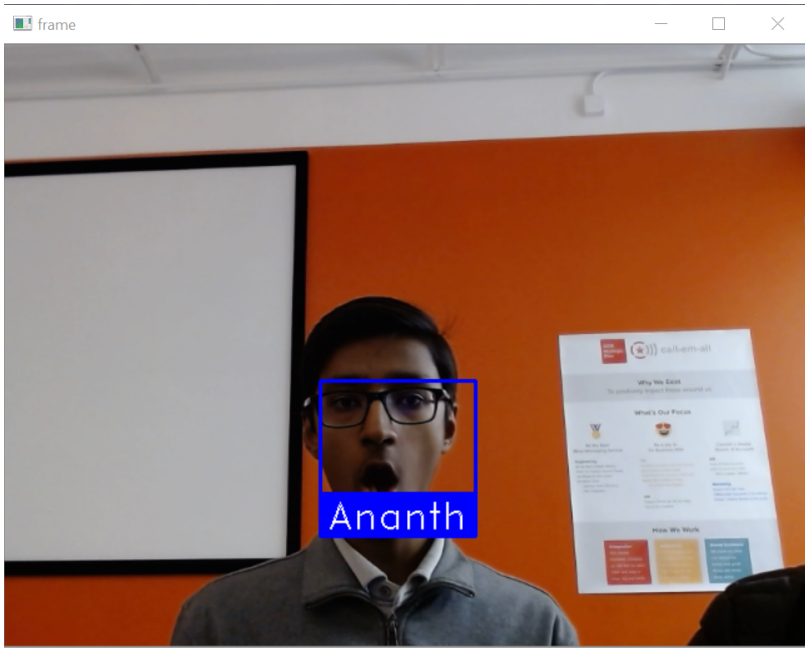
In software development, taking into account the runtime is easily one of the most important aspects of developing a powerful program. In this project, the Encodings directory serves as one of the key features to reduce runtime. The `Encodings` directory houses several files such as `SamratEncoding.npy`. This is a numpy file in what is known as an embedding.

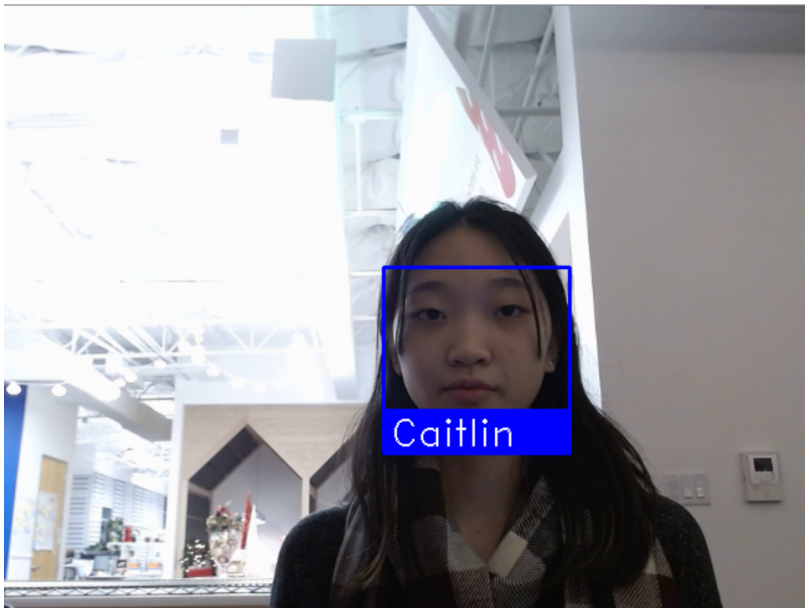
Purpose in Facial Recognition Process

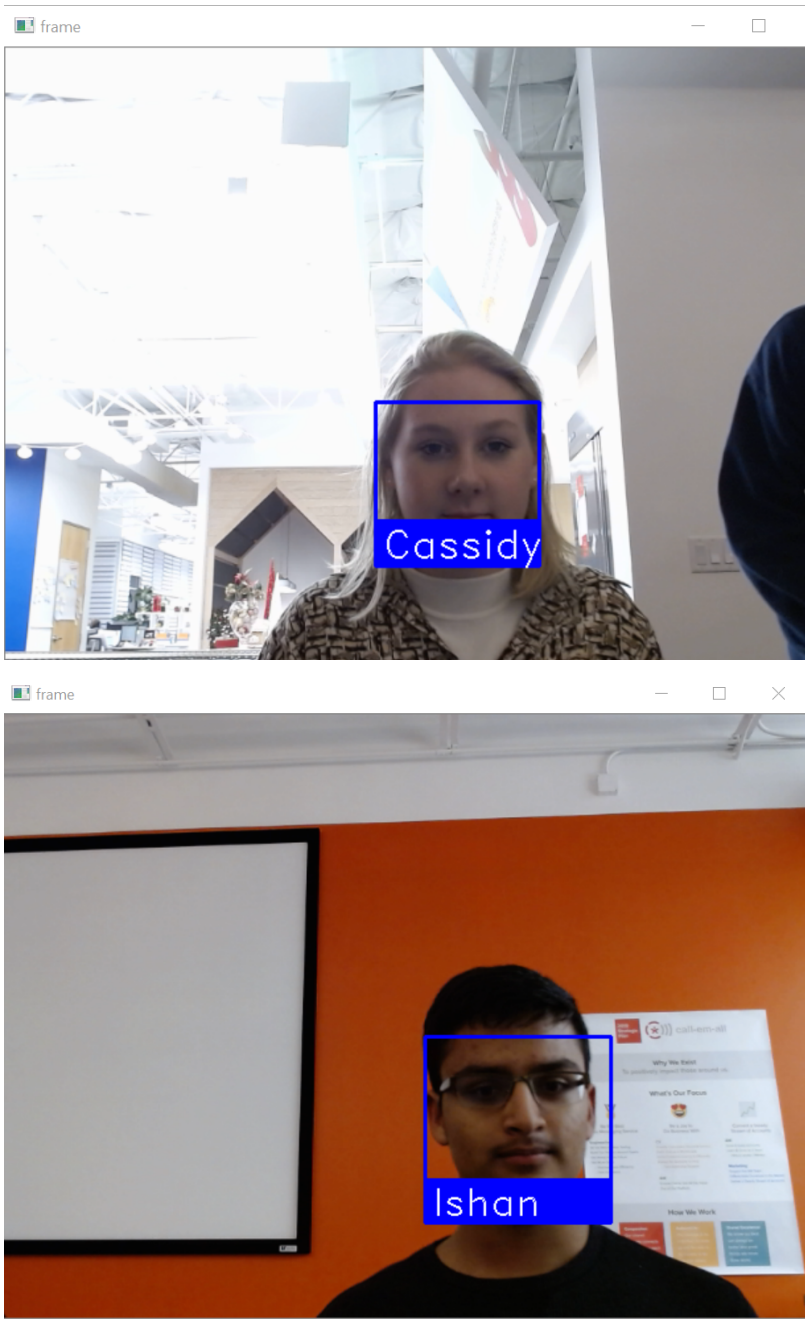
In the image pre-processing state, it is necessary to convert raw images to usable data for the computer. In order to do that, we convert the images into numpy arrays using the Histogram of Oriented Gradients algorithm. These files hold numpy arrays for the information of the person so that the computer would not have to recalculate these arrays every time the program is run. This drastically reduced runtime from 10 minutes to to 10 seconds. This was a major improvement from the original program design that had to recalculate data over and over again.

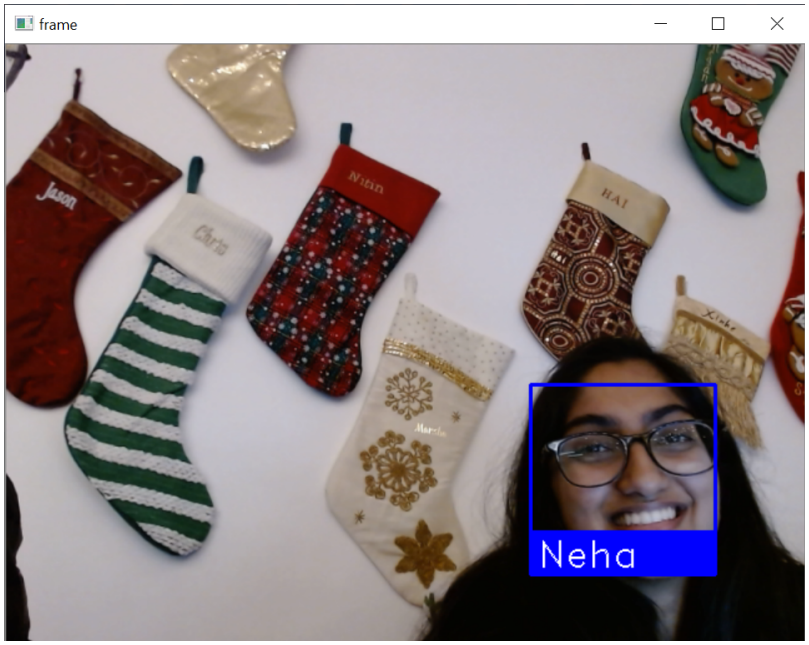
Example Executions

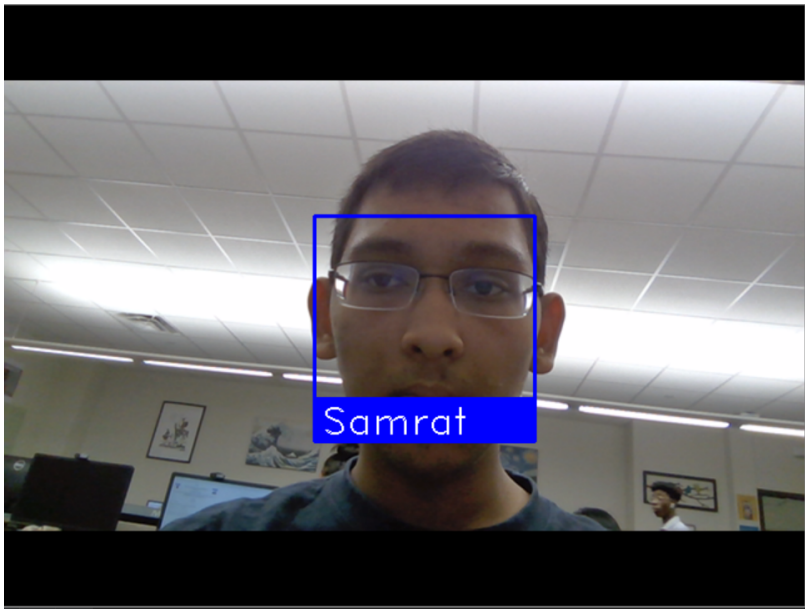
Here you can find some example executions of the facial recognition process.

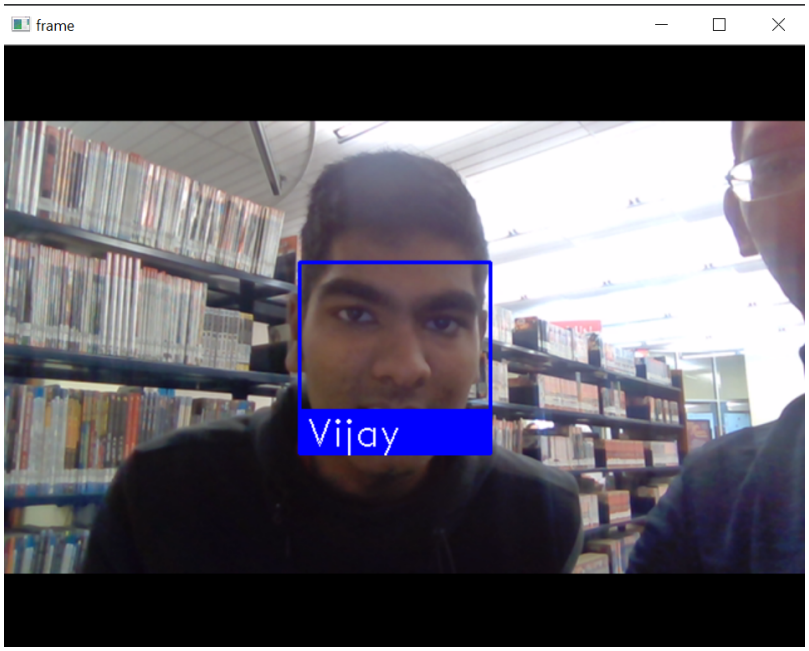





frame
Cassidy
frame
Ishan


frame
Neha


Samrat



List Information

The list information folder has 3 different text files: `Encoding Names`, `Face Names Known`, `Full Student Names`, These text files are loaded into lists within the python files. These text files have also allowed for the feature of dynamic addition of faces to emerge

- `Encoding Names.txt`: The `Encoding Names` file holds the names of the numpy files found in the `Encodings` subdirectory. This text file also adds names of new encodings.
- `Face Names Known.txt`: The `Face Names Known` file allows us to attribute the faces in the database to encodings. This consists of only the first names.
- `Full Student Names.txt`: The `Full Student Names` allows us to store full names so that they can be outputted onto a excel spreadsheet, google sheet, or text file.

Model

The Model directory holds the machine learning models that are used for liveness detection. Within the directory there are 2 files: the H5 model file and the Json model file.

- `model.h5`: This file holds the actual distribution of data for the model. It is then run through Keras using the `LivenessModel.py` to process this data into usable information
- `model.json`: This is a file that also holds model information with the necessary requirements to process the model. This file is actually readable to humans.

People Images Folder

The People images folder is the local database of photos for each person. Within the People Images folder you will see subdirectories of each of the names found in `List Information/Face Names Known` and each subdirectory holds the respective photos of that person. Within each person subdirectory you will see 2 photos, `0.jpg`, `1.jpg`

JPG File Names Explained

The JPG files are named `0.jpg, 1.jpg` respectively because within `EncodingModel.py` we have a method called `encodeDirectory()` which requires at least 2 images to process and outputs an embeeding to `Encodings/`. The `0.jpg, 1.jpg` are the two files that it processes.

Static

The static folder was made because with the flask application framework, it is necessary to have a static directory to hold cascading stylesheets files, images and javascript files. Within the `static` folder, you will see a `css` folder `js` folder and an `img` folder.

- `css`: Holds Cascading Stylesheets Files
- `js`: Holds JavaScript Files
- `img`: Holds Images

Templates

The `templates` folder is another necessary folder for the flask web application framework. It houses all of the HTML files that created the interface of the project. The starting tab is defined by `index.html`.

1.3 Files

1.3.1 File List

Application.py

The Application.py file was originally used to run the core application by checking if there were enough images in each of the respective folders in `People Images` but now has been abandoned.

Imports

```
import os
from init import *
from Sheets import *
```

- `os`: Necessary to access file systems
- `init`: Necessary to access the arrays
- `Sheets`: Necessary to access the formatPage() method that is later used

Functions

The `getFolderSize()` function makes sure there are at least 2 different people in the `People Images/` directory before proceeding with the application. If there are less than 2 people it will bypass that and make sure the application runs anyways

```
def getFolderSize():
 folderSize = len(next(os.walk("People Images/" + str(faceNamesKnown[x])))) - 1
 if folderSize < 2:
     folderSize = 2
 return folderSize
```

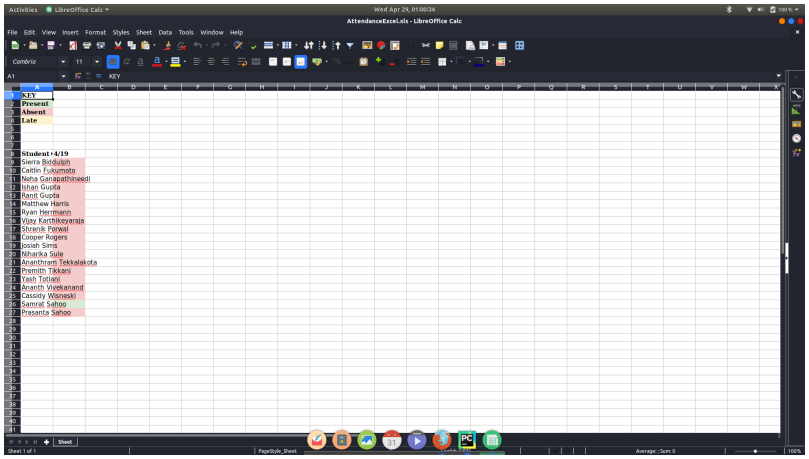
Main Method

The main method will run `TransferLearning.py` if all of the folders have the correct amount of images.

```
if __name__ == '__main__':
 for x in range(0, len(faceNamesKnown)):
     formatPage()
     if getFolderSize() == 2:
         import TransferLearning
```

AttendanceExcel.xls

This is a file that visually presents the data on a certain day through a Microsoft Excel Sheet.



AttendanceSheet.txt

This is a text file that lexicographically presents the attendance on a certain date.

Camera.py

The `Camera.py` file serves to be the connection between the HTML dashboard and the OpenCV camera. The Camera.py file controls all the camera functions.

Imports

```
import os
import sys
import cv2
import face_recognition
from TransferLearning import loadDictionary, loadLists, toList, getLivenessValue,␣
↪runInParallel, dynamicAdd, \
    getFolderSize, checkIfHere
from init import *
import numpy as np
from Excel import *
from LivenessDetection import getModel, getModelPred
import socket
from Sheets import *
from timeit import default_timer as timer
```

- `os`: Necessary to access file systems
- `sys`: Necessary to access the operating system
- `cv2`: Necessary to access computer vision tools
- `face_recognition`: Necessary to access face recognition tools
- `TransferLearning`: Necessary to access helper methods in original program
- `init`: Necessary to access the arrays
- `numpy`: Necessary to access Linear Algebra functions
- `Excel`: Necessary to access Microsoft Excel methods

- `LivenessDetection`: Necessary to access the Liveness Detection models
- `socket`: Necessary to check internet connection
- `Sheets`: Necessary to access Google Sheets methods
- `timeit`: Necessary to take times

Feature Control Variables

These variables serve to control different features related to the camera

```
global dynamicState
global pauseState
global onlineMode
dynamicState = False
pauseState = True
onlineMode = False
```

- `dynamicState`: Controls whether you want to add a new person or not
- `pauseState`: Controls whether the camera will be paused or not
- `onlineState`: Controls whether to use google sheets or excel

Static Functions

The `addPerson()` toggles the `dynamicState` variable from True to False and vice versa.

```
def addPerson():
 global dynamicState
 dynamicState = True
```

The `internetCheck()` function will use the socket class and try to create a connection with Google.com. If it fails, it will throw an Exception and return False.

```
def internetCheck():
 try:
     socket.create_connection(("www.google.com", 80))
     return True
 except OSError:
     pass
 return False
```

Objects

The VideoCamera Object initializes with several starting variable amounts including initial arrays, liveness models, timestamps, encodings, and internet connections.

```
def __init__(self, source):
 try:
     # Call on OpenCV Video Capture
     self.video = cv2.VideoCapture(source)

     # Some global variables
```

(continues on next page)

(continued from previous page)

```
    global processThisFrame, faceLocations, faceEncodings, faceNames, encodingList,␣
↪encodingNames
    global faceNamesKnown, fullStudentNames, inputFrames, model, start, internetCheck

    # Initialize variables
    faceLocations = []
    faceEncodings = []
    faceNames = []
    inputFrames = []
    processThisFrame = True

    # Load List information
    fullStudentNames = loadLists("List Information/Full Student Names")  # List with␣
↪full Student Names
    faceNamesKnown = loadLists("List Information/Face Names Known")  # List With␣
↪Face Names
    encodingNames = loadLists("List Information/Encoding Names")  # List With␣
↪encoding names
    loadDictionary("List Information/Face Names Known", faceEncodingsKnown)  #␣
↪Dictionary with Encodings
    encodingList = toList(faceEncodingsKnown)

    # Load encodings
    for x in range(0, int(len(encodingList))):
        encodingList[x] = np.load("Encodings/" + str(encodingNames[x]))

    # Load Liveness Model
    model = getModelPred()

    # Start Late timer
    start = timer()

    # Internet Check
    internetCheck = internetCheck()

except Exception as e:
    print(e)
```

When it is destroyed, it deletes the camera.

```
def __del__(self):
    # Delete Video Capture
    self.video.release()
```

Object Functions

The `addFace()` function will apply the `dynamicAdd()` from `TransferLearning.py` if and only if a face is found and it is Unknown. It will then reload all of the arrays and encodings, At the end, it will turn the `dynamicState` variable to False.

```
def addFace(self):

    # Some global variables
    global dynamicState, encodingNames, fullStudentNames, faceNamesKnown,␣
↪encodingList, frame
```

(continues on next page)

(continued from previous page)

```
    # Only run Dynamic Addition if a face is found and is unknown
    if 'Unknown' in faceNames and len(faceLocations) > 1:
        # Run dynamic core addition
        dynamicAdd(frame)

        # Relaod Lists
        fullStudentNames = loadLists("List Information/Full Student Names")  # List␣
↪with full Student Names
        faceNamesKnown = loadLists("List Information/Face Names Known")  # List With␣
↪Face Names
        encodingNames = loadLists("List Information/Encoding Names")  # List With␣
↪encoding names
        loadDictionary("List Information/Face Names Known", faceEncodingsKnown)  #␣
↪Dictionary with Encodings

        # Run Encoding Model as necessary
        if getFolderSize("Encodings/") != len(encodingNames):
            import EncodingModel

        # Reload Enecodings
        encodingList = toList(faceEncodingsKnown)
        for x in range(0, int(len(encodingList))):
            encodingList[x] = np.load("Encodings/" + str(encodingNames[x]))

        # Turn off dynamic addition once done
        dynamicState = False
```

The `getRawFrame()` function will return solely the frame the OpenCV camera sees.

```
def getRawFrame(self):
    # Returns the raw frame
    _, frameToReturn = self.video.read()
    return frameToReturn
```

The `goOnline()` function will control the onlineMode feature control variable to control whether to use online or offline mode.

```
def goOnline(self):
    global onlineMode
    onlineMode = not onlineMode
```

The `getFrame()` function is the core function and has been split into different parts for the purpose of readability and easier to understand documentation.

Here we are declaring some global variables that are used universally throughout `Camera.py`

```
def getFrame(self):
    try:
        # Some global variables
        global processThisFrame, faceLocations, faceNames, encodingList,␣
↪faceNamesKnown, fullStudentNames
        global model, inputFrames, frame, dynamicState, start, internetCheck,␣
↪onlineMode
```

Next we are reading the frame and converting it into the correct dimensions and formats for our needs. This also includes calculating the elapsed time.

```
# Read OpenCV video
success, frame = self.video.read()
# Resize as necessary
smallFrame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)
# Change Colors as necessary
rgbSmallFrame = smallFrame[:, :, ::-1]
# End time for Late feature
end = timer()
# Calculate time spent
elapsedTime = end - start
```

We are then using the `processThisFrame` variable to process every other frame so that the User Experience is better. We also calcualte the locations and encodings of the faces in the current frame being analyzed. We declared an empty list that will store all face names in the frame.

```
# Only process every other frame of video to save time
if processThisFrame:
    # Find all the faces and face encodings in the current frame of video
    faceLocations = face_recognition.face_locations(rgbSmallFrame)
    faceEncodings = face_recognition.face_encodings(rgbSmallFrame, faceLocations)

    # Empty Face names for every iteration
    faceNames = []
```

We then calculate the blur amount using a Laplacian function and if the blur is low enough we will perform face recognition to the frame. The face recognition is done through calculating a Frobenius Norm to find the variance between saved encodings and encodings within the frame. The lowest variance is the face that is recognized. If the variance is an outlier, then it will assume the face is not in the database and give it an unknown tag.

```
# Calculate Blur; if its too blurry it won't do facial recognition
blurAmount = cv2.Laplacian(frame, cv2.CV_64F).var()
    if blurAmount > 40:
        for faceEncoding in faceEncodings:
            # See if the face is a match for the known face(s)
            matchesFound = face_recognition.compare_faces(encodingList, faceEncoding)
            name = "Unknown"

            # Or instead, use the known face with the smallest distance to the new␣
↪face
            faceDistances = face_recognition.face_distance(encodingList, faceEncoding)
            matchIndex = np.argmin(faceDistances)
            if matchesFound[matchIndex]:
                name = faceNamesKnown[matchIndex]
            # Add name to the faceNames array
            faceNames.append(name)
    # Process every other frame
    processThisFrame = not processThisFrame
```

This will calculate the coordinates to draw the faces. It also calculates liveness values and blur amounts once again.

```
# Display the results
for (top, right, bottom, left), name in zip(faceLocations, faceNames):
    # Scale back up face locations since the frame we detected in was scaled to 1/4␣
↪size
    top *= 4
    right *= 4
    bottom *= 4
```

(continues on next page)

(continued from previous page)

```
    left *= 4

    # Draw a box around the face
    cv2.rectangle(frame, (left, top), (right, bottom), (255, 0, 0), 2)

    # Draw a label with a name below the face
    cv2.rectangle(frame, (left, bottom - 35), (right, bottom), (255, 0, 0), cv2.
↪FILLED)
    font = cv2.FONT_HERSHEY_DUPLEX
    # Recalculate blur
    blurAmount = cv2.Laplacian(frame, cv2.CV_64F).var()
    # Calculate liveness amount
    livenessVal = getLivenessValue(frame, inputFrames, model)
```

This part will actually draw the box with a name if and only if the image is alive.

```
# if liveness is over 95% then continue recognition
if livenessVal > 0.95:
    # Blur must be over 40 in order to accurately recognize a face
    if blurAmount > 40:
        cv2.putText(frame, name, (left + 6, bottom - 6), font, 1.0, (255, 255, 255),␣
↪1)
```

This part will check if the User is online or offline and their respective mode. If they are online and in online mode, it will record attendance on Google Sheets. This part will also check if they are late or not.

```
# Online/Offline Mode
        if internetCheck and onlineMode:
            for x in range(0, len(fullStudentNames)):
                if name in fullStudentNames[x]:
                    # Check if they are late
                    if elapsedTime > 300:
                        updateLatePerson()
                    else:
                        updatePresentPerson()
```

If they are offline it will put it on the Microsoft Excel sheet.

```
else:
    for x in range(0, len(fullStudentNames)):
        if name in fullStudentNames[x]:
            # Check if they are late
            if elapsedTime > 300:
                updateLatePersonExcel(fullStudentNames[x])
            else:
                updatePresentPersonExcel(fullStudentNames[x])
```

This will record it on the text file

```
for x in range(0, len(faceNamesKnown)):
    checkIfHere(name, faceNamesKnown[x])
```

If it is a spoof, it will warn the user,

```
else:
    # Do not mark anyone if its a spoof
```

(continues on next page)

(continued from previous page)

```
    cv2.putText(frame, "WARNING: SPOOF DETECTED", (100, 75), font, 1.0, (0, 0, 255),␣
↪2)
```

This will encode the frame into a .jpeg file so that it can be displayed on the Flask Dashboard.

```
# Encode frame so it can be displayed on a webpage
ret, jpeg = cv2.imencode('.jpg', frame)
return jpeg.tobytes()
```

This will catch any potential errors that may occur.

```
except Exception as e:
    # Enceptions to get file + line numbers errors are on
    exceptionType, exceptionObject, exceptionThrowback = sys.exc_info()
    fileName = os.path.split(exceptionThrowback.tb_frame.f_code.co_filename)[1]
    print(exceptionType, fileName, exceptionThrowback.tb_lineno)
    print(e)
```

DynamicAddition.py

The `DynamicAddition.py` file serves as a file that houses helper methods to assist with the dynamic addition process.

Imports

```
import cv2
from EncodingModel import *
import numpy as np
```

- `cv2`: Necessary to access computer vision tools
- `EncodingModel`: Necessary to access modules to encode images into usable data
- `numpy`: Necessary to access Linear Algebra functions

Methods

The `pauseCamera()` method is used to pause the camera. This is no longer used but was once used in `TransferLearning.py`

```
def pauseCamera():
    cv2.waitKey(-100)
```

The `dynaicAdd()` method is used to add faces. This method was also once used in `TransferLearning.py` but since has been abandoned. This gets the full name of the person.

```
def dynamicAdd(image):
    firstName = input("What is your first name: ")
    lastName = input("What is your last name: ")
    fullName = firstName + " " + lastName
```

Then the text files that store names of people are edited.

```
with open("List Information/Full Student Names", "a") as f:
    f.write(fullName)
    f.write("\n")
    f.close()

with open("List Information/Face Names Known", "a") as f:
    f.write(firstName)
    f.write("\n")
    f.close()
with open("List Information/Encoding Names", "a") as f:
    f.write(firstName + "Encoding.npy")
    f.write("\n")
    f.close()
```

Separate directories are made for each person to store their images and their images are saved

```
os.makedirs("People Images/" + firstName)
cv2.imwrite(os.path.join("People Images/" + firstName, '0.jpg'), image)
cv2.imwrite(os.path.join("People Images/" + firstName, '1.jpg'), image)
```

Their images are encoded and the encoding is saved as a numpy file.

```
encoding = encodeDirectory(firstName)
np.save('Encodings/' + str(firstName).replace(" ", "") + 'Encoding.npy', encoding)
```

EncodingModel.py

The `EncodingModel.py` File serves as the core for the encoding process. This allows images to be converted into usable data for the computer to use.

Imports

```
import face_recognition
import numpy as np
import os
from init import faceNamesKnown, faceEncodingsKnown, encodingNames
```

Methods

The `encodeFace()` method takes in an image path and return an encoding after having analyzed the image.

```
def encodeFace(imageDirectory):
    # Load Images
    image = face_recognition.load_image_file(imageDirectory)
    # Encode Images
    encoding = face_recognition.face_encodings(image, None, 5)[0]
    return encoding
```

The `encodeDirectory()` method takes in a directory of images and returns an average encoding after having analyzed the multiple images. It takes advantage of the `encodeFace()` method to encode several images. It then adds up the encodings and takes the average of all of the encodings.

```
# Method encodes a directory of images and returns the average encoding of the images
def encodeDirectory(directoryName):
    # Create list for all encodings
    allEncodings = []
    # Go through directory of files
    for filename in os.listdir("People Images/" + directoryName):
        # Get amount of files in directory
        fileAmount = len(next(os.walk("People Images/" + directoryName)))
        if filename.endswith(".jpg"):
            # iterate through files in directory
            for fileNum in range(0, fileAmount - 1):
                # Add encodings to list
                allEncodings.append(encodeFace("People Images/" + directoryName + "/"␣
↪+ str(fileNum) + ".jpg"))
    # List Length
    listLength = len(allEncodings)
    # Return average of encoded arrays array
    return sum(allEncodings) / listLength
```

Main Method

The main method will encode every directory in the `People Images` folder and save the files for each respective person in the `Encodings` folder.

```
for x in range(0, len(faceNamesKnown)):
    faceEncodingsKnown[x] = encodeDirectory(faceNamesKnown[x])
    np.save('Encodings/' + encodingNames[x], faceEncodingsKnown[x])
```

Excel.py

The `Excel.py` file controls the outputs to a Microsoft Excel sheet. There are several helper methods in `Excel.py` that make outputting possible.

Imports

```
from datetime import datetime

from openpyxl import Workbook
from openpyxl.styles import PatternFill, Font
from init import *
```

- `datetime`: Necessary to get the date
- `openpyxl`: Necessary to manipulate the Excel file
- `init`: Necessary to access the arrays

Methods

The `loadLists()` method will allow for us to load the list information from `Full Student Names.txt` into the arrays in `init.py`

```
def loadLists(textFile):
    with open(textFile) as file:
        list = file.readlines()
        file.close()
        list = [x[:-1] for x in list]
    return list
```

The `absentCell()` method marks a given cell red.

```
def absentCell(sheet, cell):
    # Add Red Color Cell Format
    redFill = PatternFill(start_color='F4CCCC',
                          end_color='F4CCCC',
                          fill_type='solid')
    sheet[cell].fill = redFill
```

The `presentCell()` method marks a given cell green.

```
def presentCell(sheet, cell):
    # Add Green Color Cell Format
    greenFill = PatternFill(start_color='D9EAD3',
                            end_color='D9EAD3',
                            fill_type='solid')
    sheet[cell].fill = greenFill
```

The `lateCell()` method marks a given cell yellow.

```
def lateCell(sheet, cell):
# Add Yellow Color Cell Format
yellowFill = PatternFill(start_color='FFF2CC',
                         end_color='FFF2CC',
                         fill_type='solid')
sheet[cell].fill = yellowFill
```

The `resetCell()` method marks a given cell white.

```
def resetCell(sheet, cell):
    # Add White Color Cell Format
    whiteFill = PatternFill(start_color='FFFFFF',
                            end_color='FFFFFF',
                            fill_type='solid')
    sheet[cell].fill = whiteFill
```

The `addKeyExcel()` method adds the Sheet key to the upper left hand corner of the sheet.

```
def addKeyExcel(sheet):
    # Reset Top Cells
    for n in range(1, 5):
        cellLocation = 'A' + str(n)
        resetCell(sheet, cellLocation)

    # Add Key Colors and Labels
    presentCell(sheet, 'A2')
    absentCell(sheet, 'A3')
    lateCell(sheet, 'A4')
    sheet['A1'] = 'KEY'
    sheet['A1'].font = Font(bold=True)
```

(continues on next page)

(continued from previous page)

```
    sheet['A2'] = 'Present'
    sheet['A2'].font = Font(bold=True)
    sheet['A3'] = 'Absent'
    sheet['A3'].font = Font(bold=True)
    sheet['A4'] = 'Late'
    sheet['A4'].font = Font(bold=True)
```

The `addStudentNamesExcel()` method adds the Student names in the first column of the Excel sheet.

```
def addStudentNamesExcel(sheet):
    # Format and write Student Name subtitle
    sheet['A8'] = 'Student Names'
    sheet['A8'].font = Font(bold=True)
    # Write student names from init list
    for n in range(0, len(fullStudentNames)):
        cellLocation = 'A' + str(9 + n)
        sheet[cellLocation] = fullStudentNames[n]
```

The `getRowNumber()` method gets the row number to mark. This is used to mark a certain student.

```
def getRowNum(personToFind):
startCellNum = 9
for x in range(0, len(fullStudentNames)):
    # Find how many to go down from row 9 by comparing names + arrays
    if fullStudentNames[x].strip() == personToFind.strip():
        # Update row to go to
        startCellNum += x
return startCellNum
```

The `getColumnLetter()` method gets the column letter to mark. This is used to mark on a certain date.

```
def getColumnLetter(sheet):
# Start column is B
cellStartNum = ord('B')
# Get date because column will correspond
date = datetime.today().strftime('X%m/X%d')
date = date.replace('X0', 'X').replace('X', '')
columnFound = False
# Compare current date to column date
while not columnFound:
    currentCell = str(chr(cellStartNum)) + '8'
    # If found, return cell column Letter
    if sheet[currentCell].value == date:
        return cellStartNum
    else:
        cellStartNum += 1
```

The `addDateExcel()` method adds the current date. In coordination with the application, it marks the date the application is launched.

```
def addDateExcel(sheet):
# Get and format date
date = datetime.today().strftime('X%m/X%d')
date = date.replace('X0', 'X').replace('X', '')
# character number for "B"
cellStartNum = ord('B')
```

(continues on next page)

(continued from previous page)

```
# Flag boolean to exit loop
emptyDateCell = False

while not emptyDateCell:
    # Get Current cell location
    currentCell = str(chr(cellStartNum)) + '8'
    # If the date is already there, then you do not need to add another column
    if sheet[currentCell].value == date:
        break
    else:
        # # If cell is not empty, move over one cell horizontally
        if sheet[currentCell].value != None:
            cellStartNum += 1
        else:
            # If cell is empty, write the date
            sheet[currentCell] = date
            sheet[currentCell].font = Font(bold=True)
            emptyDateCell = True
```

The `formatPageExcel()` method formats the page as needed if it has already not been formatted.

```
def formatPageExcel(sheet):
    # Adds key, student names, and current date
    if sheet['A1'] != 'KEY':
        addKeyExcel(sheet)
    addStudentNamesExcel(sheet)
    addDateExcel(sheet)
```

The `updatePresentPersonExcel()` method updates an excel sheet passed on the person's name.

```
def updatePresentPersonExcel(personToFind):
    # Change numerical values to cell value
    cellToPresent = chr(getColumnLetter(ws)) + str(getRowNum(personToFind))
    # Mark present
    presentCell(ws, cellToPresent)
```

The `updateAbsentPersonExcel()` method updates an excel sheet passed on the person's name.

```
def updateAbsentPersonExcel(personToFind):
    # Change numerical values to cell value
    cellToAbsent = chr(getColumnLetter(ws)) + str(getRowNum(personToFind))
    # Mark Absent
    absentCell(ws, cellToAbsent)
```

The `updateLatePersonExcel()` method updates an excel sheet passed on the person's name.

```
def updateLatePersonExcel(personToFind):
    # Change numerical values to cell value
    cellToAbsent = chr(getColumnLetter(ws)) + str(getRowNum(personToFind))
    # Mark Late
    lateCell(ws, cellToAbsent)
```

The `markAbsentUnmarkedExcel()` method will mark all people who were not present as absent.

```
def markAbsentUnmarkedExcel():
    rowStart = 9
```

(continues on next page)

(continued from previous page)

```
    for x in range(0, len(fullStudentNames)):
        cellToCheck = str(chr(getColumnLetter(ws))) + str(rowStart)
        if str(ws[cellToCheck].fill.start_color.index) not in '00D9EAD3':
            absentCell(ws, cellToCheck)
            rowStart += 1
        else:
            rowStart += 1
    wb.save("AttendanceExcel.xls")
```

Main Method

The main method here will first load all of the lists, then create a Workbook and worksheet for the Excel Spreadsheet. Finally, it will format the spreadsheet as needed.

```
try:
    fullStudentNames = loadLists("List Information/Full Student Names")
    wb = Workbook()
    ws = wb.active
    formatPageExcel(ws)
except Exception as e:
    print(e)
```

init.py

The `init.py` file serves as a file where every file in the project can access globally declared arrays. This allows for shared variables within files. All variables are empty because they are inputted respective values in other files.

Dictionaries

The `faceEncodingsKnown` dictionary is used to hold an encoding name as its key and the respective encoding as its value.

```
faceEncodingsKnown = {

}
```

Lists

The `faceNamesKnown` list holds the first names only from the `Face Names Known.txt` file.

```
faceNamesKnown = [

]
```

The `fullStudentNames` list holds the first and last names from the `Full Student Names.txt` file.

```
fullStudentNames = [

]
```

The `encodingNames` list holds the encoding namesfrom the `Encoding Names.txt` file.

```
encodingNames = [

]
```

Interface.py

The `Interface.py` file controls the complete backend for this project. This includes all of the interface button bindings.

Imports

```
import sys
from flask import render_template, Flask, Response
from webui import WebUI
from Camera import VideoCamera
import os
from shutil import copyfile
from DynamicAddition import dynamicAdd
from Excel import markAbsentUnmarkedExcel
```

- `sys`: Necessary to access the operating system
- `flask`: Necessary to access Python Backend to Web Application Front End
- `webui`: Necessary to turn the flask web app to a desktop interface
- `Camera`: Necessary to access Camera Object and functions
- `os`: Necessary to access file systems
- `shutil`: Necessary to be able to copy files
- `DynamicAddition`: Necessary to access DynamicAddition methods
- `Excel`: Necessary to access Microsoft Excel methods

Variables

The `app` variable declares that the HTML, CSS, and JS is to be used for a flask web application. The `ui` variable converts the app to a desktop app.

```
app = Flask(__name__)
ui = WebUI(app, debug=True)
```

These global variables are used to control states of different features or store certain values. .. code-block:: python

    global cameraState, addState, frames, framesRaw, onlineState cameraState = False addState = False onlineState = False framesRaw = [] frames = []

Page Access Methods

These methods are all used to access different pages or tabs within the Interface.

```
@app.route('/')
@app.route('/index')
def indexPage():
    return render_template('index.html')


@app.route('/configure')
def configurePage():
    return render_template('configurations.html')


@app.route('/attendance')
def attendancePage():
    return render_template('attendance.html')


@app.route('/settings')
def settingsPage():
    return render_template('settings.html')


@app.route('/contact')
def contactPage():
    return render_template('contact.html')


@app.route('/help')
def helpPage():
    return render_template('help.html')
```

Methods

The `downloadText()` method and `downloadExcel()` method, are both there to make a copy of the text file or Excel file into the user's downloads directory.

```
@app.route('/download-text')
def downloadText():
    try:
        finalPath = os.path.join(os.path.expanduser("~"), "Downloads/AttendanceSheet.
↪txt")
        copyfile('AttendanceSheet.txt', finalPath)
    except Exception as e:
        print(e)
    return render_template('index.html')


@app.route('/download-excel')
def downloadExcel():
    try:
        finalPath = os.path.join(os.path.expanduser("~"), "Downloads/AttendanceExcel.
↪xls")
        copyfile('AttendanceExcel.xls', finalPath)
    except Exception as e:
        print(e)

    return render_template('index.html')
```

The `startCamera()` and `stopCamera()` methods are used to toggle the camera on and off based on the button pressed. If the Start Camera button is pressed, `startCamera()` is called and `cameraState` will be `True` but if the Stop Camera button is pressed, `stopCamera()` is called and `cameraState` will be `False` and the camera will turn off.

```
@app.route('/start-camera')
def startCamera():
    global cameraState
    cameraState = True
    return render_template('index.html')


@app.route('/stop-camera')
def stopCamera():
    global cameraState
    cameraState = False
    markAbsentUnmarkedExcel()
    return render_template('index.html')
```

The `gen()` method is the core method for `Interface.py`. It first calls the values global variables.

```
def gen(camera):
    global addState, cameraState, frames, framesRaw, onlineState
```

If the application is not set to dynamically add a face, it will get a raw frame and converted frames using the object methods in `Camera.py`. It will append raw frames to the `framesRaw` array and output the converted frames onto the Interface.

```
while cameraState or addState:
    if not addState:
        global frames, framesRaw
        frame = camera.getFrame()
        frames.append(frame)
        framesRaw.append(camera.getRawFrame())
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')
```

If it is in the state to dynamically add a face, it will get the last frame that was displayed before the dynamic add button was pressed and freeze it on that frame. It will then process that frame through the dynamic add method. After it finishes, it will return back to camera mode and exit dynamic add mode.

```
if addState:
    frameToSave = len(frames) - 1
    yield (b'--frame\r\n'
           b'Content-Type: image/jpeg\r\n\r\n' + frames[frameToSave] + b'\r\n\r\n')
    try:
        dynamicAdd((framesRaw[frameToSave]))
        camera.additionProcess()
        cameraState = True
        addState = False
    except Exception as e:
        exceptionType, exceptionObject, exceptionThrowback = sys.exc_info()
        fileName = os.path.split(exceptionThrowback.tb_frame.f_code.co_filename)[1]
        print(exceptionType, fileName, exceptionThrowback.tb_lineno)
        print(e)
    break
```

If the online mode button is pressed, the application will switch to the Google Sheets output

```
if onlineState:
    camera.goOnline()
    onlineState = False
```

Finally every time the camera mode and dynamic add mode is exited, it will mark everyone who was not present as absent.

```
markAbsentUnmarkedExcel()
```

The `addFace()` method and `onlineMode()` method are both used to toggle booleans that control the modes the application is in.

```
@app.route('/add-face')
def addFace():
    global addState
    addState = True
    return render_template('index.html')


@app.route('/online-mode')
def onlineMode():
    global onlineState
    onlineState = True
    return render_template('index.html')
```

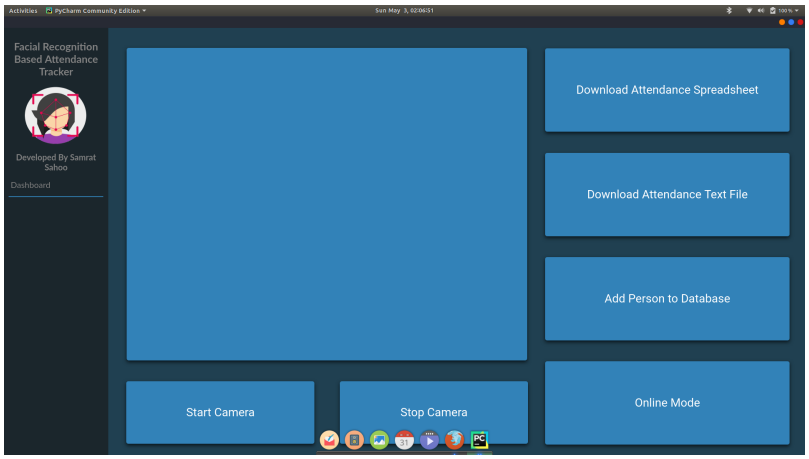
The `video_feed()` method simply places the video feed into the web based dashboard.

```
@app.route('/video_feed')
def video_feed():
    return Response(gen(VideoCamera(source=-1)),
                    mimetype='multipart/x-mixed-replace; boundary=frame')
```

Main Method

The main method in `Interface.py` launches the Dashboard through using the `run()` method on the ui object.

```
if __name__ == '__main__':
    try:
        ui.run()
    except Exception as e:
        print(e)
```



LivenessDetection.py

The `LivenessDetection.py` file controls the LivenessDetection model processing which differentiates real faces from flat images.

Imports

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.layers import Conv3D, MaxPooling3D
```

- `Keras`: Necessary for deep learning functions to process the model

Methods

The `getModel()` method is used to process the data within a model so that is more usable. This is done through a Sequential model with several layers to correctly process the data.

```
def getModel():
    model = Sequential()
    model.add(Conv3D(32, kernel_size=(3, 3, 3),
                     activation='relu',
                     input_shape=(24, 100, 100, 1)))
    model.add(Conv3D(64, (3, 3, 3), activation='relu'))
    model.add(MaxPooling3D(pool_size=(2, 2, 2)))
    model.add(Conv3D(64, (3, 3, 3), activation='relu'))
    model.add(MaxPooling3D(pool_size=(2, 2, 2)))
    model.add(Conv3D(64, (3, 3, 3), activation='relu'))
    model.add(MaxPooling3D(pool_size=(2, 2, 2)))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(2, activation='softmax'))

    return model
```

The `getModelPred()` method is used to simply initialize and load the model with its respective weights.

```
def getModelPred():
    model = getModel()
    model.load_weights("Model/model.h5")
    return model
```

Sheets.py

The `Sheets.py` file controls the outputs to a Google sheet. There are several helper methods in `Sheets.py` that make outputting possible.

Imports

```
import gspread
import pygsheets
from oauth2client.service_account import ServiceAccountCredentials
from gspread_formatting import *
import datetime
from init import fullStudentNames
from datetime import datetime
```

- `gspread`: Necessary to access the google sheet
- `pygsheets`: Necessary to manipulate the Google Sheet
- `oauth2client`: Necessary to connect to Google's servers
- `gspread_formatting`: Necessary to format the Google Sheet
- `datetime`: Necessary to get the date
- `init`: Necessary to access the arrays

Methods

The `loadLists()` method will allow for us to load the list information from `Full Student Names.txt` into the arrays in `init.py`

```
def loadLists(textFile):
    with open(textFile) as file:
        list = file.readlines()
        file.close()
        list = [x[:-1] for x in list]
    return list
```

The `absentCell()` method marks a given cell red.

```
def absentCell(cell):
    # Add Red Color Cell Format
    format = CellFormat(backgroundColor=Color(.96, .80, .80))
    # Update a Cell as Absent
    format_cell_range(sheet, cell, format)
```

The `presentCell()` method marks a given cell green.

```
def presentCell(cell):
    # Add Green Color Cell Format
    format = CellFormat(backgroundColor=Color(.85, .93, .82))
    # Update a Cell as Present
    format_cell_range(sheet, cell, format)
```

The `lateCell()` method marks a given cell yellow.

```
def lateCell(cell):
    # Add Yellow Color Cell Format
    format = CellFormat(backgroundColor=Color(1.00, .95, .80))
    # Update a Cell as Late
    format_cell_range(sheet, cell, format)
```

The `resetCell()` method marks a given cell white.

```
def resetCell(cell):
    # Add White Color Cell Format
    format = CellFormat(backgroundColor=Color(1, 1, 1))
    # Reset a Cell
    format_cell_range(sheet, cell, format)
    sheet.update_acell(cell, '')
```

The `addKey()` method adds the Sheet key to the upper left hand corner of the sheet.

```
def addKey():
    # Reset Top Cells
    for n in range(1, 5):
        cellLocation = 'A' + str(n)
        resetCell(cellLocation)

    # Add Key Colors and Labels
    presentCell('A2')
    absentCell('A3')
    lateCell('A4')
    format = CellFormat(textFormat=TextFormat(bold=True))
    format_cell_range(sheet, 'A1', format)
    sheet.update_acell('A1', 'KEY')
    sheet.update_acell('A2', 'Present')
    sheet.update_acell('A3', 'Absent')
    sheet.update_acell('A4', 'Late')
```

The `addStudentNames()` method adds the Student names in the first column of the sheet.

```
def addStudentNames():
    # Format and write Student Name subtitle
    format = CellFormat(textFormat=TextFormat(bold=True))
    format_cell_range(sheet, 'A8', format)
    sheet.update_acell('A8', 'Student Names')
    # Write student names from init list
    for n in range(0, len(fullStudentNames)):
        cellLocation = 'A' + str(9 + n)
        sheet.update_acell(cellLocation, fullStudentNames[n])
```

The `addDate()` method adds the current date. In coordination with the application, it marks the date the application is launched.

```
def addDate():
    # Get and format date
    date = datetime.today().strftime('X%m/X%d')
    date = date.replace('X0', 'X').replace('X', '')
    # character number for "B"
    cellStartNum = ord('B')
    # Flag boolean to exit loop
    emptyDateCell = False
    # Format Date Subtitles
    format = CellFormat(textFormat=TextFormat(bold=True), horizontalAlignment='RIGHT')

    while not emptyDateCell:
        # Get Current cell location
        currentCell = str(chr(cellStartNum)) + '8'
        # If the date is already there, then you do not need to add another column
        if sheet.acell(currentCell).value == date:
            break
        else:
            # # If cell is not empty, move over one cell horizontally
            if sheet.acell(currentCell).value != '':
                cellStartNum = cellStartNum + 1
            else:
                # If cell is empty, write the date
                format_cell_range(sheet, currentCell, format)
                sheet.update_acell(currentCell, date)
                emptyDateCell = True
```

The `formatPage()` method formats the page as needed if it has already not been formatted.

```
def formatPage():
    # Adds key, student names, and current date
    if sheet.acell('A1').value != 'KEY':
        addKey()
    addStudentNames()
    addDate()
```

The `getRowNumber()` method gets the row number to mark. This is used to mark a certain student.

```
def getRowNum(personToFind):
startCellNum = 9
for x in range(0, len(fullStudentNames)):
    # Find how many to go down from row 9 by comparing names + arrays
    if fullStudentNames[x].strip() == personToFind.strip():
        # Update row to go to
        startCellNum += x
return startCellNum
```

The `getColumnLetter()` method gets the column letter to mark. This is used to mark on a certain date.

```
def getColumnLetter(sheet):
# Start column is B
cellStartNum = ord('B')
# Get date because column will correspond
date = datetime.today().strftime('X%m/X%d')
date = date.replace('X0', 'X').replace('X', '')
columnFound = False
# Compare current date to column date
```

(continues on next page)

(continued from previous page)

```
while not columnFound:
    currentCell = str(chr(cellStartNum)) + '8'
    # If found, return cell column Letter
    if sheet[currentCell].value == date:
        return cellStartNum
    else:
        cellStartNum += 1
```

The `updatePresentPerson()` method updates a Google sheet passed on the person's name.

```
def updatePresentPerson(personToFind):
    # Change numerical values to cell value
    cellToPresent = chr(getColumnLetter(ws)) + str(getRowNum(personToFind))
    # Mark present
    presentCell(cellToPresent)
```

The `updateAbsentPerson()` method updates an Google sheet passed on the person's name.

```
def updateAbsentPerson(personToFind):
    # Change numerical values to cell value
    cellToAbsent = chr(getColumnLetter(ws)) + str(getRowNum(personToFind))
    # Mark Absent
    absentCell(cellToAbsent)
```

The `updateLatePerson()` method updates a Google Sheet passed on the person's name.

```
def updateLatePerson(personToFind):
    # Change numerical values to cell value
    cellToAbsent = chr(getColumnLetter(ws)) + str(getRowNum(personToFind))
    # Mark Late
    lateCell(cellToAbsent)
```

The `markOnce()` method is used to make sure a cell is not overwritten.

```
def markOnce(name):
    # Change numerical values to cell value
    cellToCheck = str(chr(getColumnLetter())) + str(getRowNum(name))
    # Return False if cell is not white or red
    return worksheet.cell(cellToCheck).color != (None, None, None, None) or worksheet.
↪cell(cellToCheck).color != (
        .96, .80, .80, 1.00)
```

The `markAbsentUnmarked()` method will mark all people who were not present as absent.

```
def markAbsentUnmarked():
    rowStart = 9
    for x in range(0, len(fullStudentNames)):
        cellToCheck = str(chr(getColumnLetter())) + str(rowStart)
        if worksheet.cell(cellToCheck).color == (None, None, None, None):
            absentCell(cellToCheck)
            rowStart += 1
        else:
            rowStart += 1
```

Main Method

The main method will authorize all of the necessary credentials and then find the Google Sheet within the Google Drive of the respective account. It will lastly autoformat the page.

```
try:
    fullStudentNames = loadLists("List Information/Full Student Names")
    # Gets scope of sheet
    scope = ["https://spreadsheets.google.com/feeds", "https://www.googleapis.com/
↪auth/spreadsheets",
             "https://www.googleapis.com/auth/drive.file", "https://www.googleapis.
↪com/auth/drive"]
    # Gets sheet credentials and authorizes it
    creds = ServiceAccountCredentials.from_json_keyfile_name("creds.json", scope)
    client = gspread.authorize(creds)
    # Opens sheet based on sheet name
    sheet = client.open("19/20 Attendance").sheet1

    # Authorize Pygsheets library
    gc = pygsheets.authorize()
    worksheet = gc.open('19/20 Attendance').sheet1
    formatPage()
except Exception as e:
    print(e)
```

TransferLearning.py

The TransferLearning.py file has been abandoned. However, many methods within `TransferLearning.py` are used in other modules. `TransferLearning.py` used to be the original core file until the switch to `Interface.py`.

Imports

```
import sys
from init import *
from Sheets import *
from DynamicAddition import *
import cv2
import face_recognition
import numpy as np
import os
from multiprocessing import Process
from LivenessDetection import getModel
```

- `os`: Necessary to access file systems
- `sys`: Necessary to access the operating system
- `cv2`: Necessary to access computer vision tools
- `face_recognition`: Necessary to access face recognition tools
- `init`: Necessary to access the arrays
- `numpy`: Necessary to access Linear Algebra functions
- `LivenessDetection`: Necessary to access the Liveness Detection models

- `Sheets`: Necessary to access Google Sheets methods
- `DynamicAddition`: Necessary to access DynamicAddition method
- `multiprocessing`: Necessary to run multiple methods at once

Variables

`TransferLearning.py` takes advantage of global variables in order to modularize the complete file. Below are all the global variables in `TransferLearning.py`.

```
global fullStudentNames, faceNamesKnown, encodingNames, model, video, encodingList,␣
↪faceLocations, faceEncodingsKnown
global faceEncodings, faceNames, inputFrames, processThisFrame, x, file, smallFrame,␣
↪rgbFrame, livenessVal, name
```

Methods

The `checkIfHere()` method makes sure that each name found in the frame only appears once.

```
def checkIfHere(name, nameToCheck):
    if name is nameToCheck:
        with open("AttendanceSheet.txt", 'r') as f:
            if nameToCheck in f.read():
                pass
            else:
                with open("AttendanceSheet.txt", 'a') as f2:
                    f2.write(name + "\n")
                    f2.close()
```

The `getFolderSize()` method returns the folder size of a given folder.

```
# Method to get amount of files in a certain folder
def getFolderSize(folderName):
    fileList = os.listdir(folderName)
    numberFiles = len(fileList)
    return numberFiles
```

The `adjustBrightness()` method takes advantage of HSV values in order to adjust the brightness when the frame is too dark.

```
# Method to adjust to a certain brightness
def adjustBrightness(img):
    # Converts frame from RGB to HSV
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    # Splits HSV type into 3 different arrays
    h, s, v = cv2.split(hsv)
    # Calculates image's average brightness
    averageBrightness = np.sum(v) / np.size(v)
    # Set minimum brightness
    brightnessThreshold = 125
    # Calculate how much to increase the brightness
    brightnessIncrease = brightnessThreshold - int(averageBrightness)
    # See if average brightness exceeds the threshold
    if averageBrightness < brightnessThreshold:
```

(continues on next page)

(continued from previous page)

```
        # Increases brightness
        lim = 255 - brightnessIncrease
        v[v > lim] = 255
        v[v <= lim] += brightnessIncrease
    # Merge the HSV values back together
    finalHSV = cv2.merge((h, s, v))
    # Redetermine image value & Return Image
    img = cv2.cvtColor(finalHSV, cv2.COLOR_HSV2BGR)
    return img
```

The `toList()` method, `loadLists()`, and `loadDictionary()` methods are used in order to manipulate the text files in `List Information/` and load all the arrays with the correct information.

```
def toList(dictionary):
    listToReturn = list(dictionary.values())
    index = int(len(listToReturn))
    listToReturn = listToReturn[:index]
    return listToReturn


def loadLists(textFile):
    with open(textFile) as file:
        list = file.readlines()
        file.close()
        list = [x[:-1] for x in list]
    return list


def loadDictionary(file, dictionary):
    with open(file, "rt") as f:
        for line in f.readlines():
            dictionary[line.strip()] = None
```

The `runInParallel()` method allows us to run function in parallel. It is most notably used for dynamic addition.

```
def runInParallel(*fns):
    proc = []
    for fn in fns:
        p = Process(target=fn)
        p.start()
        proc.append(p)
    for p in proc:
        p.join()
```

The `getLivenessValue()` method manipulates the matrices of the last 24 frames and is able to return a liveness value from 0 to 1. The higher the value the more live the frame is.

```
def getLivenessValue(frame, inputFrames, model):
    livenessFrame = cv2.resize(frame, (100, 100))
    livenessFrame = cv2.cvtColor(livenessFrame, cv2.COLOR_BGR2GRAY)
    inputFrames.append(livenessFrame)
    input = np.array([inputFrames[-24:]])
    input = input / 255
    if input.size == 240000:
        input = input.reshape(1, 24, 100, 100, 1)
        pred = model.predict(input)
```

(continues on next page)

(continued from previous page)

```
        return pred[0][0]
    return 0.96
```

Omitted Method Documentation

Due to similarities in `TransferLearning.py` and `Camera.py`, documentation for the `preProcess()`, `optimizeWebcam()`, `recognizeFaces()`, `dynamicallyAdd()`, `writeOnStream()`, and `writeToFile()` methods have been omitted in this page and have instead have their documentations on the `Camera.py` documentation. This allows for the brevity of documentation.

Main Method

The main method here combines several of the methods in order to to put together the complete application. When `q` is pressed, the application will end.

```
if __name__ == '__main__':
    preProcess()
    while True:
        try:
            # Open Webcam + Optimize Webcam
            ret, frame = video.read()
            optimizeWebcam(frame)
            recognizeFaces()
            dynamicallyAdd(frame)
            writeOnStream(frame)
            writeToFile()
            cv2.imshow('Frame', frame)

            # If q is pressed, exit loop
            if cv2.waitKey(20) & 0xFF == ord('q'):
                break

        except Exception as e:
            exceptionType, exceptionObject, exceptionThrowback = sys.exc_info()
            fileName = os.path.split(exceptionThrowback.tb_frame.f_code.co_
↪filename)[1]
            print(exceptionType, fileName, exceptionThrowback.tb_lineno)
            print(e)

    # ============================================== Post Program␣
↪============================================================
    # Upon exiting while loop, close web cam
    video.release()
    cv2.destroyAllWindows()

markAbsentUnmarked()
```

1.4 Contact

1.4.1 Email

If you have any questions, you can email me at samratsahoo2013@gmail.com

1.4.2 File an Issue

If you have an issue, you can file it on Github.

1.4.3 Other Social Media

- Twitter
- LinkedIn

1.4.4 Research Portfolio

You can find all of the research conducted over the past year at my Research Portfolio.

CHAPTER 2

About the ISM Program

This project was made for the Independent Study Mentorship Program. The Independent Study Mentorship Program is a rigorous research-based program offered at Frisco ISD schools for passionate high-achieving individuals.

Throughout the course of the program, students carry-out a year-long research study where they analyze articles and interview local professionals. Through the culmination of the research attained, students create an original work, presented at research showcase, and a final product that will be showcased at final presentation night. Through the merits of this program, individuals are able to follow their passions while increasing their prominence in the professional world and growing as a person.

The ISM program is a program in which students carry-on skills that not only last for the year, but for life. ISM gives an early and in-depth introduction to the students' respective career fields and professional world. Coming out of this program, students are better equipped to handle the reality of the professional world while having become significantly more knowledgeable of their respective passions. The ISM program serves as the foundation for the success of individuals in the professional world.

CHAPTER 3

Project Preface

This attendance system uses image data sets to create an average encoding of the person. This encoding is then compared with frame encodings from the camera stream. The respective encoding with the least variance is the chosen face that is recorded.

CHAPTER 4

Acknowledgements

I would just like to give a special thank you to Adam Geitgey for the creation of his face_recognition class.

I would also like to give a special thanks to the following individuals for their contributions to my research throughout this project.

- Trey Blankenship [Raytheon]
- Won Hwa Kim [UT Arlington]
- Tim Cogan [ams AG]
- Vijay Nidumolu [Samsung Electronics America]
- Sehul Viras [Dallas Baptist University & IntelliCentric]

One last thank you for the Radicubs Robotics Team for helping me test this attendance tracker.