IDC409: Data Science Project

Submitted by:

Avleen Singh MS20033
Kumar Ayush MS20013
Chiranjib Mahanta MS20007
Shivam Kumar MS20185

Video Link of working code:
https://drive.google.com/file/d/1-maLUvR8qAxrI3vHAM2wKWHFwoaS4iRQ/view?usp=sharing

Drive Link to Folder with data and notebook:
https://drive.google.com/drive/folders/1TxIv765jVQ2CF5BpocHX_3ukum1eKHok?usp=sharing

Our assigned project is on Image Processing. We focus on face-recognition here.

Following code describes how a face recognition system can be built in Python. In Image classification, we have 'image detection' and 'image recognition'. In the following code, both aspects are tried to be covered.

Face recognition algorithms incorporate a variety of steps. Pre-processing and data organization becomes an important aspect for training the face recognition model. We here incorporate OpenCV and face recognition libraries to perform the same.

Further, we try to associate this with a classroom attendance system.

There are multiple algorithms for face recognition. Mainly, all the algorithms find key points in a face, their coordinates, trains on them, and then one can test it out. These coordinates are named as facial features and there are as many as 66 of them.

Following four major steps, we try building the model.

Detection: A picture is scanned for the face of a person, following which a boundary around it is drawn and coordinates of the same are stored

Alignment: Since we have trained the model with a training set of images, it becomes very important to align the test images to appropriate measures such that comparison can then be made.

Extraction: Here, the pic's features are scrutinized and subsequently used for training and recognition.

Recognition: It is the comparison step, based on which output (in this case attendance) is given/ not given.

LIBRARIES

dlib

This is a library in C++ environment, providing essential tools for tasks like face recognition, image processing etc. Out of all essentials features, our intended tools are listed:

1. Routines for reading and writing common image formats.
2. Automatic color space conversion between various pixel types
3. Common image operations such as edge finding and morphological operations
4. Implementations of the SURF, HOG, and FHOG feature extraction algorithms.
5. Tools for detecting objects in images including frontal face detection and object pose estimation.
6. High quality face recognition

face-recognition

It is a library in python that recognizes faces. For more info: face-recognition · PyPI

OpenCV

It is a library of programming functions mainly for real-time vision.

Let's now describe how the code is working:

IMPORTING THE LIBRARIES

```
import cv2
import face_recognition
import os
import numpy as np
from datetime import datetime
import pickle
✓ 0.0s
```

As described above, we first import the libraries. Here os is module providing us functions that allow us to work with the operating system

Datetime is for attendance.
pickle allows us to create portable serialized representations of Python objects.

SETTING PATH

```
path = "data"
```
✓ 0.0s

It describes the folder with training images.

CREATING LISTS

```
images = []
classNames = []
mylist = os.listdir(path)
for cl in mylist:
    curImg =cv2.imread(f'{path}/{cl}')
    images.append(curImg)
    classNames.append(os.path.splitext(cl)[0])
```
✓ 0.4s

`images = []` and `classNames = []` are empty lists created to store image data and corresponding class names.

`mylist = os.listdir(path)` retrieves a list of files in the directory specified by the variable `path`. The code then enters a loop (`for cl in mylist:`) where `cl` iterates over the items in `mylist`.

`curImg = cv2.imread(f'{path}/{cl}')` reads an image file. It uses the `cv2` module, which is commonly associated with OpenCV, a popular computer vision library in Python. `f'{path}/{cl}'` constructs the full path of the current file by concatenating `path` and `cl`.

`images.append(curImg)` appends the loaded image (`curImg`) to the `images` list.

`classNames.append(os.path.splitext(cl)[0])` extracts the base filename (without the extension) using `os.path.splitext(cl)[0]` and appends it to the `classNames` list. This is useful for associating a name or label with each image.

ENCODING ALL TRAIN IMAGES

```
def findEncodings(images):
    encodeList = []
    for img in images:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        encoded_face = face_recognition.face_encodings(img)[0]
        encodeList.append(encoded_face)
    return encodeList
encoded_face_train = findEncodings(images)
```

✓ 26.3s

`def findEncodings(images):`: This line defines a function named `findEncodings` that takes a single argument, `images`.

`encodeList = []`: This initializes an empty list called `encodeList` that will be used to store the encoded representations of faces.

`for img in images:`: This starts a loop where `img` iterates over the list of images passed as an argument.

`img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)`: Here, each image (`img`) is converted from BGR color space to RGB color space using OpenCV (`cv2`). This is necessary because the face_recognition library expects images in RGB format.

`encoded_face = face_recognition.face_encodings(img)[0]`: This line computes the face encoding of the image. It uses the `face_recognition` library to extract facial features and generate a numerical representation of the face. `[0]` is used to get the first (and typically only) face detected in the image.

`encodeList.append(encoded_face)`: The encoded face is then appended to the `encodeList`.

`return encodeList`: Once all images have been processed, the list of encoded faces is returned.

`encoded_face_train = findEncodings(images)`: Finally, the `findEncodings` function is called

with the `images` list as an argument. The resulting list of encoded faces is assigned to the variable `encoded_face_train`.

ATTENDANCE

```
def markAttendance(name):
    with open('Attendance.csv','r+') as f:
        myDataList = f.readlines()
        nameList = []
        for line in myDataList:
            entry = line.split(',')
            nameList.append(entry[0])
        if name not in nameList:
            now = datetime.now()
            time = now.strftime('%I:%M:%S:%p')
            date = now.strftime('%d-%B-%Y')
            f.writelines(f'n{name}, {time}, {date}')
```

✓ 0.0s

`with open('Attendance.csv','r+') as f:`: This line opens a file named `Attendance.csv` in read and write mode (`'r+'`). The file is opened within a `with` statement, which ensures that the file is properly closed after the code block is executed.

`myDataList = f.readlines()`: This reads all the lines of the file and stores them as a list of strings in `myDataList`.

`nameList = []`: This initializes an empty list called `nameList`.

`for line in myDataList:`: This starts a loop that iterates over each line in `myDataList`.

`entry = line.split(',')`: For each line, it splits the line into a list of elements based on commas (`,`). This is assuming that the CSV file has comma-separated values.

`nameList.append(entry[0])`: It appends the first element of the split line (which is assumed to be the name) to the `nameList`.

`if name not in nameList:`: This checks if the provided `name` is not already in the `nameList`.

Inside the `if` block:
- `now = datetime.now()`: This gets the current date and time.
- `time = now.strftime('%I:%M:%S:%p')`: This formats the current time in a specific format (hours:minutes:seconds:AM/PM).
- `date = now.strftime('%d-%B-%Y')`: This formats the current date in a specific format (day-month-year).

`f.writelines(f'n{name}, {time}, {date}')`: If the provided `name` is not already in the

`nameList`, this line writes a new entry to the file. It includes the `name`, `time`, and `date` separated by commas. The `f` before the string indicates that it's using f-string formatting to include variables in the string.

TESTING AND WEBCAM

```
# Initialize webcam
cap = cv2.VideoCapture(0)
marked_names = []
while True:
    success, img = cap.read()
    imgS = cv2.resize(img, (0, 0), None, 0.25, 0.25)
    imgS = cv2.cvtColor(imgS, cv2.COLOR_BGR2RGB)
    faces_in_frame = face_recognition.face_locations(imgS)
    encoded_faces = face_recognition.face_encodings(imgS, faces_in_frame)
```

`cap = cv2.VideoCapture(0)`: This line initializes a video capture object (`cap`) using the OpenCV library (`cv2`). It is set to capture video from the default webcam (index 0).

`marked_names = []`: This initializes an empty list called `marked_names`.

`while True:`: This starts an infinite loop, which means the following code will keep running until the loop is explicitly broken.

`success, img = cap.read()`: Within each iteration of the loop, this line reads a frame from the webcam using the `cap.read()` method. It returns two values: `success` (a boolean indicating whether the frame was successfully read) and `img` (the frame itself).

`imgS = cv2.resize(img, (0, 0), None, 0.25, 0.25)`: This line resizes the captured frame (`img`) to a smaller size. This can help improve processing speed. It reduces the width and height to one-quarter of the original size.

`imgS = cv2.cvtColor(imgS, cv2.COLOR_BGR2RGB)`: The colors in the resized image (`imgS`) are converted from BGR (Blue, Green, Red) to RGB format. This is important because the `face_recognition` library expects images in RGB format.

`faces_in_frame = face_recognition.face_locations(imgS)`: This line uses the `face_recognition` library to detect face locations in the frame (`imgS`). It returns a list of tuples, where each tuple contains the coordinates (top, right, bottom, left) of a detected face.

`encoded_faces = face_recognition.face_encodings(imgS, faces_in_frame)`: This line uses the `face_recognition` library again to generate face encodings for the detected faces in the frame. It takes the resized image (`imgS`) and the face locations (`faces_in_frame`) as inputs. It returns a list of numerical encodings representing the detected faces.

```
        if matches[matchIndex]:
            name = classNames[matchIndex].upper().lower()
            if name not in marked_names:
                marked_names.append(name)
                y1, x2, y2, x1 = faceloc
                y1, x2, y2, x1 = y1*4, x2*4, y2*4, x1*4
                cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2)
                cv2.rectangle(img, (x1, y2-35), (x2, y2), (0, 255, 0), cv2.FILLED)
                cv2.putText(img, name, (x1+6, y2-5), cv2.FONT_HERSHEY_COMPLEX, 1, (255, 255, 255), 2)
                markAttendance(name)
            else:
                cv2.putText(img, "Already Marked", (x1+6,y2+30), cv2.FONT_HERSHEY_COMPLEX, 0.5, (255, 255, 255), 1)
        else:
            cv2.putText(img, "Unknown Face", (x1+6, y2+30), cv2.FONT_HERSHEY_COMPLEX, 0.5, (255, 255, 255), 1)
    cv2.imshow('webcam', img)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

`for encode_face, faceloc in zip (encoded_faces, faces_in_frame):` - It starts a loop that iterates over pairs of detected face encodings (`encode_face`) and their corresponding face locations (`faceloc`). The `zip` function is used to combine elements from `encoded_faces` and `faces_in_frame` into pairs.

`matches = face_recognition.compare_faces(encoded_face_train, encode_face)`: For each detected face, this line compares it (`encode_face`) with a set of known face encodings (`encoded_face_train`). It returns a list of boolean values (`matches`), where each element indicates whether there is a match between the detected face and any of the known faces.

`faceDist = face_recognition.face_distance(encoded_face_train, encode_face)`: It calculates the distances between the detected face (`encode_face`) and all the known faces in `encoded_face_train`. The result is an array of numerical distances (`faceDist`).

`matchIndex = np.argmin(faceDist)`: Here, `np.argmin()` is used to find the index of the minimum value in the `faceDist` array. This index corresponds to the known face in `encoded_face_train` that is the closest match to the detected face.

`print(matchIndex)`: This line prints the index of the closest matching face for the current detected face.

```
for encode_face, faceloc in zip(encoded_faces, faces_in_frame):
    matches = face_recognition.compare_faces(encoded_face_train, encode_face)
    faceDist = face_recognition.face_distance(encoded_face_train, encode_face)
    matchIndex = np.argmin(faceDist)
    print(matchIndex)
```

if matches[matchIndex]: This checks if there is a match for the detected face. matches[matchIndex] is a boolean value indicating whether there is a match between the detected face and any of the known faces.

name = classNames[matchIndex].upper().lower(): If there is a match, this line retrieves the name associated with the known face using the **matchIndex**. It converts the name to lowercase for uniformity.

if name not in marked_names: This checks if the name has not already been marked (meaning the person has not already been recorded as present).

marked_names.append(name): If the person has not been marked already, their name is added to the list of marked names to prevent duplicate entries.

The lines following extract the coordinates of the detected face (**faceloc**) and scale them by a factor of 4. This is necessary because

earlier in the code, the captured frame was resized to improve processing speed. A green rectangle is drawn over the detected face.

Subsequently, name is displayed. If marked onced, an “Alredy Marked” sign appears. If system fails to recognize or picture is not present in training set, the “Unknown Face” is displayed.

if cv2.waitKey(1) & 0xFF == ord('q'): This checks for a key press. If the 'q' key is pressed, it breaks out of the loop, effectively stopping the program.

cap.release()

cv2.destroyAllWindows() : These lines release the webcam capture and close any open windows associated with OpenCV, cleaning up resources after program finishes.

LIMITATIONS

Building facial recognition systems presents challenges in real-world images taken without constraints. Key challenges include illumination changes, sensitivity to pose, variations in facial expressions, low resolution, and aging.

As can be seen, not all predictions are accurate. Sometimes, it may inaccurately predict known faces as unknown.

REFERENCE

Machine Learning is Fun! Part 4: Modern Face Recognition with Deep Learning by Adam Geitgey

https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78