# CSC420 Final Project

Carlos E. Cadalso Rivero

April 2023

## 1 Project 1

### 1.1 Q(a)

The trailers are inside the zip file, in the vid/cut/ directory.

The three selected trailers were from the movies "The Godfather", "Shawshank Redemption", and "Schindler's List".

The cut versions of the trailers used in the code are the files "godfather_cut.mp4", "schindler_cut.mp4", and "shawshank_cut.mp4". The rest of the mp4 files in the folder were used in testing, but the program cannot identify the faces of the actors, as it does not have pictures of them in its data set.

These three trailers were selected because they represent 3 edge cases that I believed would come up before starting.

1. Does the program work with relatively low-resolution videos?
2. How well does the program perform in videos with fast shows and low illumination?
3. Does the program work for videos in black and white?

### 1.2 Q(b)

The photos of 26 actors are in the dataset folder, each photo is within a different folder based on who the actor is. The file "actor_names.txt" identifies each of these actors.

The included actors are 20 actors from IMDB's "Top Actors" list, and 2 actors from each movie trailer.

### 1.3 Q(c)

Shots are separated in the output by a black screen with "Scene Change" written in the center. Shot change is detected through a color histogram comparison. The program compares the color histograms of two adjacent shots and if that difference is under a certain threshold, it adds 10 frames announcing there was a cut.

### 1.4 Q(d)

I would evaluate how well I am detecting shots based on 3 factors:

1. How many shot transitions were missed?
2. How many false positive shots changes were added?
3. How close to the actual shot transition (in time) is it?

I believe these 3 factors are a good metric because they account for both a lack and an excess of marked shot transitions, and reward accuracy for the timing of each transition.

Manually selected shot boundaries:
"The Godfather"
3 shot cuts within the first second
Shot cuts at seconds 1 and 7
A slow transition at second 8
Shot cuts at seconds 10, 12, 16, 20, 24, and 28
"Shawshank Redemption"
A shot cut at second 3
A slow transition at second 5
Shot cuts at second 11, 13, 15, 16, 17, 18, 20, 22, 26, and 32
"Schindler's List"
Shot cuts at seconds 3, 5, 13, 15, 20, 27, and 36

**Evaluation of "The Godfather"** The performance for this trailer is not good because a lot of cuts have inaccurate timings and the program detects a lot of shot transitions where there are none. However, the program does mark the shot transitions that are in the original correctly, even if some are a bit off-timing.

**Evaluation of "Shawshank Redemption"** The performance for this trailer is good, though not perfect. The program was accurate with the timing of all the cuts it detected. However, it failed to detect a gradient transition into a panning shot and detected a false positive around second 30.

**Evaluation of "Schindler's List"** The performance for this trailer is fine, but not great. Although the program managed to detect every cut that I found with perfect precision, it also detected multiple false positives, mostly due to sudden lighting changes in the environment.

## 1.5 Q(e)

Faces are detected in the video through 2 methods. A green square represents a face detected by a method I implemented myself, a white square is a more accurate face detection that also detects the eyes of the face, and a red square is face detection using a 3rd party library that I used to identify faces. I am aware that I could have removed the green and white squares to make the program run much faster, but as the assignment takes into account what we implemented, I intentionally left them.

The method I implemented uses the cv2 library and a Cascade Classifier class to search for faces using Haar Cascade face detection algorithm in each frame of the video. It then attempts to confirm the accuracy of its initial detection by detecting the eyes within the face. Unfortunately, the cv2 library used only works for faces facing forward, so the program struggles with sideways faces.

I used Haar Cascade face detection because it is a fairly lightweight and not-time consuming algorithm, depite not being the most accurate.

The face_detection library used for face recognition finds the faces and stores them in boxes (coordinates where the face is), then it compares the contents of each box with a previously made (in this case by me) set of data encodings that store the facial embeddings of each actor provided. Afterward, the program takes the actor with the most matches and assumes it is the actor whose face was detected. Normally, a threshold would be set to separate actors that are not present from actors that are close but wrong. However, while I did test it, I removed this feature due to the limited amount of images for each actor.

I chose to use face_detection after failing to implement face recognition on my own. While researching for ways of implementing it, I encountered this library and found that it has what I needed for the project. One flaw I see with it is that my computer struggles to run the code at any rate that might be considered fast.

### 1.6 Q(f)

After the program detects a face it notes its position and checks whether any previously detected face in the shot is within a certain range (in both size and position) of the newly detected faces. If they are, the program will readjust the size of the square without having to re-identify the face. Do note that the program will fail to keep track of highly variable faces, e.g. when someone rotates their entire head from left to right.

### 1.7 Q(g)

The program writes the name of every face it manages to identify within the square using the method described in the answer to question E. If it is unable to find any matches between the photos from the data set and the detected face, it will write "Unidentified Actor" instead of a name.

### 1.8 Q(h)

A visualization of the results is in the vid/out/ directory. It was made by compiling each of the frames used into a video. As a result, it lacks audio.

## 2 Code:

```
import numpy as np
import pickle
import cv2
import face_recognition
import os
from bing_image_downloader import downloader

THRESHOLD = 9
FACE_POSITION_THRESHOLD = 0.9
FACE_SIZE_THRESHOLD = 0.2
SCD_THRESHOLD = 1000000
frame_counter = 0
ACTOR_DATA_PATH = "./dataset/"
ORIGINAL_ACTOR_IMAGE_PATH = "./actor_data/"


def get_image_for_dataset(propt):
    # This function downloads 50 images from the internet to use in the dataset
    downloader.download(propt, limit=50, output_dir='./downloader_dataset/',
                        adult_filter_off=False)


def get_actor_names():
    # Gets the names of the actors based on the directory
    name_dict = dict()
    name_file = open('./dataset/actor_names.txt', 'r')
    full_file_string = name_file.read()
```

```
    file_line_list = full_file_string.split('\n')
    for line in file_line_list:
        index, actor_name = line.split(':')
        name_dict[index] = actor_name
    name_file.close()
    return name_dict


def encode_faces():
    # This is a function that builds the dataset from the previously images downloaded
    imagePaths = list()
    # For each file in the folder
    for actor in os.listdir(ACTOR_DATA_PATH):
        # If the file is a dir (to ensure only the folders get selected)
        if '.' not in actor:
            # For each image in the folder selected
            for image in os.listdir(ACTOR_DATA_PATH + actor):
                if image.endswith(".png") or image.endswith(".jpg"):
                    # Get the actor images into the imagePaths list
                    imagePaths.append(ACTOR_DATA_PATH + actor + "/" + image)

    # initialize the list of known encodings and known names
    knownEncodings = list()
    knownNames = list()
    actor_name_dict = get_actor_names()

    # For each image path
    for (i, imagePath) in enumerate(imagePaths):
        # Get the name of the actor
        index_to_find = imagePath.split("/")[-2]
        name_in_dict = actor_name_dict[index_to_find]

        # Read the image and identify the faces
        image = cv2.imread(imagePath)
        rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        boxes = face_recognition.face_locations(rgb, model="hog")
        encodings = face_recognition.face_encodings(rgb, boxes)

        # Store the names and encodings
        for encoding in encodings:
            knownEncodings.append(encoding)
            knownNames.append(name_in_dict)

    # Write the results into a file with the encodings
    data = {"encodings": knownEncodings, "names": knownNames}
    encoding_output = open("encodings.txt", "wb")
    encoding_output.write(pickle.dumps(data))
    encoding_output.close()
```

```
def export_video(frames, output_video_name):
    # Exports the frames as a video
    print("Processing video")
    fourcc = cv2.VideoWriter_fourcc(*'MJPG')  # video codec
    height, width, layers = frames[0].shape  # Set the video dimensions
    # Set where the new frames will go to
    video_output = cv2.VideoWriter(output_video_name, fourcc, 20.0, (width, height))

    # Write each frame into the video
    for frame in frames:
        video_output.write(frame)

    video_output.release()  # releasing the video generated
    cv2.destroyAllWindows()


def get_input_video(video_name):
    # Import the video as a list of frames
    vid = cv2.VideoCapture(video_name)

    return vid


def track_and_identify_faces(frame, prev_frame, is_different_scene, face_cascade, eye_cascade,
                             face_trackers, actor_names, current_face_ID):
    # Convert each frame to grayscale
    grayscale_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    confirmed_faces = dict()

    # Detect faces in the frame using the previosly defined CascadeClassifier
    # Note: Faces are detected in grayscale, but the ractagle is drawn over
    # the coloured frame
    faces = face_cascade.detectMultiScale(grayscale_frame, scaleFactor=1.2, minNeighbors=5,
                                          minSize=(30, 30))

    for (nx, ny, nw, nh) in faces:
        is_tracked = False  # A flag that checks whether the face was already seen in the
                            # previous frame
        if prev_frame is not None:  # and not is_different_scene
            for face_id in face_trackers.keys():
                x = face_trackers[face_id][0]
                y = face_trackers[face_id][1]
                w = face_trackers[face_id][2]
                h = face_trackers[face_id][3]

                # If the face is close enough in location and size to the previous face then
                # it is likely the same
```

```
                # face
                if (abs(x - nx) < frame.shape[0] * FACE_POSITION_THRESHOLD) and \
                        (abs(y - ny) < frame.shape[1] * FACE_POSITION_THRESHOLD) and \
                        (w * (1 - FACE_SIZE_THRESHOLD) < nw < w * (1 + FACE_SIZE_THRESHOLD)) and \
                        (h * (1 - FACE_SIZE_THRESHOLD) < nh < h * (1 + FACE_SIZE_THRESHOLD)) and \
                        not is_different_scene:
                    # If the box is within the frame
                    if (x < frame.shape[0] and y < frame.shape[1]) and (
                            x + w < frame.shape[0] and y + h < frame.shape[1]):
                        # Detect eyes
                        face_gray = grayscale_frame[ny:ny + nh, nx:nx + nw]
                        cv2.rectangle(frame, (nx, ny), (nx + nw, ny + nh), (0, 255, 0), 2)
                        eyes = eye_cascade.detectMultiScale(face_gray)
                        if len(eyes) > 0:
                            # Draw rectangle around the face
                            cv2.rectangle(frame, (nx, ny), (nx + nw, ny + nh), (255, 255, 255), 2)
                            confirmed_faces[face_id] = (nx, ny, nw, nh)
                    is_tracked = True  # The face has been tracked
        # In case this is a new face
        if not is_tracked:
            face_trackers[current_face_ID] = (nx, ny, nw, nh)
            current_face_ID = current_face_ID + 1
    return current_face_ID, confirmed_faces


def process_video(input_video_name):
    # Import the data encoding
    data_encodings = pickle.loads(open("./encodings.txt", "rb").read())

    frame_num = 0
    frames_since_transition = 20

    # Get the video
    vid = get_input_video(input_video_name)

    # Create a CascadeClassifier, an object for face detection using Haar
    # feature-based cascade classifiers as proposed by Paul Viola and Michael Jones
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
                                         "haarcascade_frontalface_default.xml")
    eye_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_eye.xml")

    # A dictionary of faces with ids as keys
    face_trackers = dict()

    # A dictionary with the names of actors using the same keys as face_trackers
    actor_names = dict()

    # A list of frames that will later be compiled into a video
```

```
list_of_frames = list()

# Get the first frame of the video
# This first time is outside the loop to emulate a "do{} while()" loop
frame_exists, frame = vid.read()
frame_num = frame_num + 1

# This will store the previous frame
prev_frame = frame.copy()

# This is a temporary id for identifying faces
current_face_ID = 0

# Making a histogram for scene change algorithm
hist_size = 256
hist_range = [0, 256, 0, 256, 0, 256]
prev_hist = cv2.calcHist([prev_frame], [0, 1, 2], None, [hist_size,
                                                         hist_size, hist_size], hist_range)

# Get the image for marking scene changes
scene_change_frame = cv2.imread('./vid/scene_change.png')

# This crops the scene_change_frame to be the same size as the frame
width = frame.shape[1]
height = frame.shape[0]
dim = (width, height)
scene_change_frame = cv2.resize(scene_change_frame, dim)

max_face_id = 0
recognized_id = 0

# Loop through each frame in the video
while frame_exists:
    # A flag to indicate when the scene changes
    is_different_scene = False

    # This section detects whether this frame is a different shot from the last ==============
    # Calculate histogram for current frame
    curr_hist = cv2.calcHist([frame], [0, 1, 2], None, [hist_size, hist_size,
                                                        hist_size], hist_range)

    # Calculate histogram difference between current and previous frames
    diff = cv2.compareHist(prev_hist, curr_hist, cv2.HISTCMP_CHISQR)

    # If histogram difference is above threshold, scene change detected
    if diff > SCD_THRESHOLD and frames_since_transition >= 30:
        frames_since_transition = 0
        attach_frame = scene_change_frame.copy()
```

```
    cv2.putText(attach_frame, str(frame_num), (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1,
                (255, 255, 255), 2)
    for i in range(10):
        list_of_frames.append(attach_frame)
    is_different_scene = False
else:
    is_different_scene = False
    frames_since_transition = frames_since_transition + 1

# Update previous histogram
prev_hist = curr_hist

# Note: The following uses the face_recognition module. This module also includes face
# detection. However, I implemented face detection using cv2 for the sake of relying
# less on already existing modules while doing the assignment. I utilize this module
# for face recognition because I did not find any other reliable way of doing it.
# Change the format of the frame into RGB to face_recognition to work with it
frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

previous_boxes = dict()
named_boxes = dict()

# Detect faces (again)
boxes = face_recognition.face_locations(frame_rgb, model="hog")

for (by1, bx2, by2, bx1) in boxes:  # Boxes
    for key in previous_boxes.keys():  # Previously seen boxes
        (sby1, sbx2, sby2, sbx1) = previous_boxes[key]
        # If the face is close enough in location and size to the previous
        # face then it is likely the same
        # face, and it is identified, then skip the identification process
        # and simply draw the rectangle
        if (abs(sbx1 - bx1) < frame.shape[0] * FACE_POSITION_THRESHOLD) and \
                (abs(sby1 - by1) < frame.shape[1] * FACE_POSITION_THRESHOLD) and \
                (abs(sbx2 - bx2) < frame.shape[0] * FACE_POSITION_THRESHOLD) and \
                (abs(sby2 - by2) < frame.shape[1] * FACE_POSITION_THRESHOLD) and \
                not is_different_scene and \
                (sbx1 < frame.shape[0] and sby1 < frame.shape[1]) and \
                (sbx2 < frame.shape[0] and sby2 < frame.shape[1]) and \
                named_boxes[key] != "Unidentified Actor":
            boxes.remove((by1, bx2, by2, bx1))
            cv2.rectangle(frame, (bx1, by1), (bx2, by2), (0, 0, 255), 2)
            if by1 - 15 > 15:
                text_y = by1 - 15
            else:
                text_y = by1 + 15
            cv2.putText(frame, actor_name, (bx1, text_y), cv2.FONT_HERSHEY_SIMPLEX,
                        0.75, (0, 255, 0), 2)
```

```
            break

# A list of names to be used later
recognized_faces_names = list()

# Default name in case no face is found identifiable
resulting_name = "Unidentified Actor"

# Compute the facial embeddings for each face to differentiate them
# (and later associate them with the actor name)
encodings = face_recognition.face_encodings(frame_rgb, boxes)

# For each facial embeddings
for encoding in encodings:
    # Get the faces that match
    matches = face_recognition.compare_faces(data_encodings["encodings"], encoding)
    # If there is a match
    if True in matches:
        matched_indexes = []
        # Go through each true match
        for i, result in enumerate(matches):
            if result:
                # Attach the associated index to matched indexes to mark it
                # as a possible recognition
                matched_indexes.append(i)
        # This dictionary and the subsequent loop is meant to count how many
        # votes the match has
        counts = dict()
        for i in matched_indexes:
            resulting_name = data_encodings["names"][i]
            counts[resulting_name] = counts.get(resulting_name, 0) + 1
        # Then take the match with the highest amount of votes as the correct result.
        # Do note that if there are no matches, the resulting_name will be
        # "Unidentified Actor"
        resulting_name = max(counts, key=counts.get)
recognized_faces_names.append(resulting_name)

for ((y1, x2, y2, x1), actor_name) in zip(boxes, recognized_faces_names):
    cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255), 2)
    previous_boxes[max_face_id] = (y1, x2, y2, x1)
    named_boxes[max_face_id] = resulting_name
    max_face_id += 1
    if y1 - 15 > 15:
        text_y = y1 - 15
    else:
        text_y = y1 + 15
    cv2.putText(frame, actor_name, (x1, text_y), cv2.FONT_HERSHEY_SIMPLEX,
                0.75, (0, 255, 0), 2)
```

```
        # The following is my own implementation of face detection ================================
        current_face_ID, faces_in_frame = track_and_identify_faces(frame, prev_frame,
                                                                   is_different_scene,
                                                                   face_cascade, eye_cascade,
                                                                   face_trackers, actor_names,
                                                                   current_face_ID)

        # Once the frame has been processed, it is added to a list of frames
        # to later be made into a video  ===========================================================
        # Add the frame to the list
        list_of_frames.append(frame)

        # Get the following frame for the next loop iteration
        prev_frame = frame.copy()
        frame_exists, frame = vid.read()
        frame_num = frame_num + 1

    # Export the frames into a new video
    export_video(list_of_frames, "./vid/out/output_" + VIDEO_TO_RUN + "_cut.mp4")


VIDEO_TO_RUN = "godfather"

process_video("./vid/cut/" + VIDEO_TO_RUN + "_cut.mp4")
```

References:

Ageitgey. (2022, June 10). *Ageitgey/face_recognition: The world's simplest facial recognition API for python and the command line*. GitHub. Retrieved April 17, 2023, from https://github.com/ageitgey/face_recognition

Diepen, G. (2018, July 12). *Detecting and tracking a face with python and opencv*. GuidoDiepen. Retrieved April 17, 2023, from https://www.guidodiepen.nl/2017/02/detecting-and-tracking-a-face-with-python-and-opencv/

Diepen, G. (2018, July 12). *Tracking multiple faces*. GuidoDiepen. Retrieved April 17, 2023, from https://www.guidodiepen.nl/2017/02/tracking-multiple-faces/

Nipun, M. S., Sulaiman, R. B., & Kareem, A. (2022). Efficiency Comparison of AI classification algorithms for Image Detection and Recognition in Real-time. *ArXiv*. https://doi.org/ https://doi.org/10.48550/arXiv.2206.05842

OpenCV Dev Team. (2014). *Face Recognition with OpenCV*. Face Recognition with OpenCV - OpenCV 2.4.13.7 documentation. Retrieved April 17, 2023, from https://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html

Python Software Foundation. (2022, February 12). *Bing-image-downloader*. PyPI. Retrieved April 17, 2023, from https://pypi.org/project/bing-image-downloader/

Rosebrock, A. (2023, March 15). *Face recognition with opencv, python, and Deep Learning*. PyImageSearch. Retrieved April 17, 2023, from https://pyimagesearch.com/2018/06/18/face-recognition-with-opencv-python-and-deep-learning/