AIMS Research - DOC

Section 1)COMPILED/OVERALL RESEARCH AND EXPLORATION :

Github link:

In the pursuit of identifying and evaluating open-source libraries and models for face detection and tracking, I focused on several prominent options: MediaPipe BlazeFace, HAAR Cascades, MTCNN, YOLO, DeepFace library, SORT, DeepSORT, and BoT-SORT. Each of these libraries/models offers unique features and capabilities. Here is an overview of my research and evaluation based on accuracy, speed, and ease of integration.

#### **1. MediaPipe BlazeFace**

**Description**: MediaPipe BlazeFace is a lightweight and highly optimized face detection model developed by Google. It is designed for real-time applications and can be used on both mobile and desktop platforms.

**Performance**:

* **Accuracy**: BlazeFace achieves high accuracy with a focus on real-time performance. It is particularly good at detecting faces in various poses and lighting conditions.
* **Speed**: The model is optimized for speed, achieving real-time performance on mobile devices. This makes it ideal for applications requiring low latency.
* **Ease of Integration**: MediaPipe provides a well-documented API that is easy to integrate into applications. It supports multiple platforms, including Android, iOS, and web.

**Evaluation**: BlazeFace is an excellent choice for real-time applications requiring high accuracy and speed. Its ease of integration further enhances its appeal.

#### **2. HAAR Cascades**

**Description**: HAAR Cascades is a traditional computer vision algorithm for object detection, including face detection, introduced by Paul Viola and Michael Jones.

**Performance**:

* **Accuracy**: HAAR Cascades can detect faces with reasonable accuracy but is less robust compared to modern deep learning models. It struggles with variations in lighting and face angles.
* **Speed**: The algorithm is relatively fast and can run in real-time on modern CPUs.
* **Ease of Integration**: HAAR Cascades are available in OpenCV, which provides extensive documentation and support. Integration is straightforward, especially for those familiar with OpenCV.

**Evaluation**: HAAR Cascades is suitable for simple face detection tasks where computational resources are limited. However, it lacks the robustness and accuracy of newer models.

#### **3. MTCNN (Multi-task Cascaded Convolutional Networks)**

**Description**: MTCNN is a deep learning-based face detection and alignment model that performs exceptionally well in detecting faces and facial landmarks.

**Performance**:

* **Accuracy**: MTCNN offers high accuracy in detecting faces and facial landmarks, even under challenging conditions.
* **Speed**: The model is relatively fast but not as optimized for real-time performance as BlazeFace.
* **Ease of Integration**: MTCNN is available in several deep learning frameworks, including TensorFlow and PyTorch. Integration requires some understanding of deep learning frameworks.

**Evaluation**: MTCNN is ideal for applications requiring high accuracy in face detection and landmark localization. It is slightly slower than BlazeFace but still performs well.

#### **4. YOLO (You Only Look Once)**

**Description**: YOLO is a state-of-the-art, real-time object detection system that can detect faces and other objects in images and videos.

**Performance**:

* **Accuracy**: YOLO achieves high accuracy and can detect faces along with other objects. It is robust to various conditions.
* **Speed**: YOLO is designed for real-time performance and is one of the fastest object detection models available.
* **Ease of Integration**: YOLO models are available in several versions (YOLOv3, YOLOv4, YOLOv5, YOLOv8) with extensive documentation and community support. Integration is relatively straightforward.

**Evaluation**: YOLO is a versatile model suitable for applications requiring real-time detection of faces and other objects. Its high accuracy and speed make it a top choice.

#### 

#### 

#### 

#### 

#### 

#### 

#### **5. DeepFace Library**

**Description**: DeepFace is an open-source deep learning library for facial recognition, verification, and analysis.

**Performance**:

* **Accuracy**: DeepFace achieves high accuracy in facial recognition and analysis tasks.
* **Speed**: The library is optimized for performance but may require powerful hardware for real-time processing.
* **Ease of Integration**: DeepFace provides an easy-to-use API with extensive documentation, making integration into various applications straightforward.

**Evaluation**: DeepFace is ideal for facial recognition and analysis tasks. Its high accuracy and ease of integration make it a valuable tool for face-related applications.

#### **6. SORT (Simple Online and Realtime Tracking)**

**Description**: SORT is a simple and efficient algorithm for real-time object tracking, including face tracking.

**Performance**:

* **Accuracy**: SORT provides reasonable accuracy for tracking objects but may struggle with occlusions and fast movements.
* **Speed**: The algorithm is designed for real-time performance and is computationally efficient.
* **Ease of Integration**: SORT is easy to implement and integrate, with code available in several programming languages.

**Evaluation**: SORT is a good choice for simple tracking tasks where speed is a priority. It may not be suitable for complex tracking scenarios.

#### **7. DeepSORT (Deep Simple Online and Realtime Tracking)**

**Description**: DeepSORT is an extension of SORT that incorporates appearance information to improve tracking performance.

**Performance**:

* **Accuracy**: DeepSORT offers higher accuracy than SORT by using deep learning-based appearance descriptors, making it more robust to occlusions and re-identification tasks.
* **Speed**: While slightly slower than SORT, DeepSORT is still capable of real-time performance.
* **Ease of Integration**: DeepSORT is more complex to integrate due to the need for appearance descriptors, but it is well-documented and supported.

**Evaluation**: DeepSORT is suitable for applications requiring more accurate and robust tracking. It strikes a good balance between accuracy and speed.

#### **8. BoT-SORT (Tracking by Overlapping Tracklets)**

**Description**: BoT-SORT is an advanced object tracking algorithm that improves upon DeepSORT by generating and merging short-term tracklets.

**Performance**:

* **Accuracy**: BoT-SORT achieves high accuracy and robustness, particularly in handling occlusions and reappearances.
* **Speed**: The algorithm is more computationally intensive than SORT and DeepSORT but can still achieve real-time performance with sufficient hardware.
* **Ease of Integration**: BoT-SORT is more complex to integrate due to its advanced features, but it provides significant benefits in tracking performance.

**Evaluation**: BoT-SORT is ideal for applications requiring highly accurate and robust tracking, especially in challenging scenarios with frequent occlusions.

After researching evaluating all of these and some more architectures/methods/models for facial detection and tracking I thought it would be most appropriate to proceed with the implementation of HAAR cascades, MediaPipe and YOLO (as seen later)

Section 2)BASIC FACE TRACKING IMPLEMENTATION:

### **Single Face Detection and Tracking: Implementation Using an Ensemble of MediaPipe and HAAR Cascades**

#### **Capabilities**

In this task, I implemented a face detection system using an ensemble of MediaPipe BlazeFace and HAAR Cascades. This solution detects both single and multiple faces in real-time from live feeds and video files.

#### **Description of the Code**

The system starts by initializing two face detection models: MediaPipe BlazeFace and HAAR Cascades. MediaPipe BlazeFace is optimized for real-time performance and high accuracy, while HAAR Cascades is a classical computer vision technique available through OpenCV.

**Processing Pipeline**:

1. **Frame Capture**:
   * The system captures frames from a video source, which can be either a live feed from a webcam or a video file.
2. **Face Detection with BlazeFace**:
   * Each frame is converted from BGR to RGB format to be compatible with MediaPipe.
   * BlazeFace processes the frame to detect faces, returning bounding boxes and confidence scores for each detected face.
3. **Face Detection with HAAR Cascades**:
   * The frame is also converted to grayscale for HAAR Cascade detection.
   * HAAR Cascades detect faces and return bounding boxes. Since HAAR Cascades do not provide confidence scores, a default score of 1.0 is assigned to each detection.
4. **Combining Detections**:
   * Detections from both BlazeFace and HAAR Cascades are combined into a single list of bounding boxes and scores.
5. **Non-Maximum Suppression (NMS)**:
   * To handle overlapping detections from the two models, NMS is applied. This technique ensures that only the best bounding box (with the highest confidence score) is retained when multiple boxes overlap significantly.
6. **Drawing Bounding Boxes**:
   * The final set of bounding boxes, filtered through NMS, is drawn on the frame. Each detected face is enclosed in a rectangle.
7. **Display**:
   * The processed frame, now with annotated face detections, is displayed to the user.
   * The loop continues to process each frame until the user interrupts by pressing the 'q' key.

**Performance**:

* The system works efficiently for both live feed and video input, detecting and tracking single or multiple faces across frames.

#### **Challenges Faced**

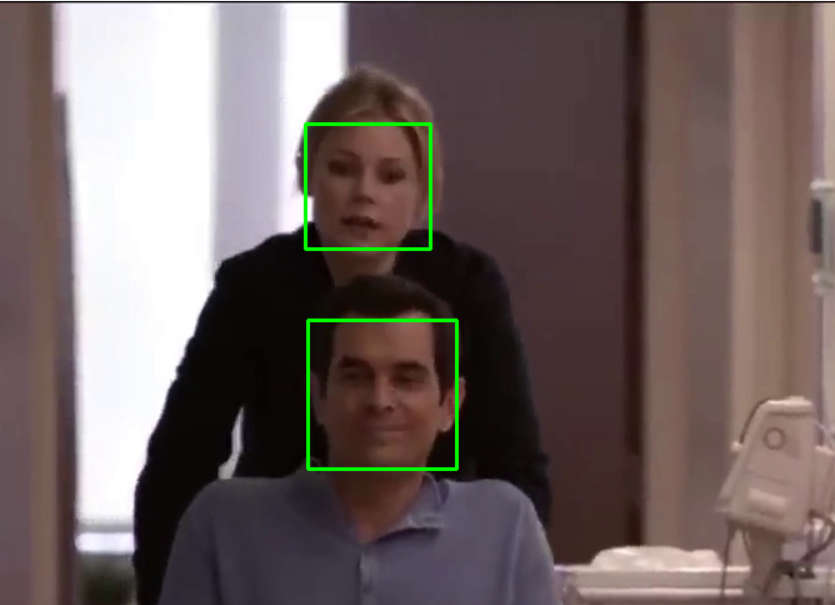
**Choosing NMS**:

* **Reason**: NMS was chosen to handle overlapping detections from BlazeFace and HAAR Cascades. This ensures that only the most confident detections are displayed, improving accuracy and reducing redundancy.
* **Yolo Integration**: Adding YOLOv3 to the ensemble did not significantly improve detection capabilities. The minor differences observed were not substantial enough to justify the increased complexity and potential confusion in the system.

#### **Solutions Implemented**

**Simplification**:

* **Decision**: I decided to move forward without including YOLOv3 in the ensemble because:
  + The addition of YOLO did not substantially enhance detection capabilities.
  + The ensemble became more complex and harder to maintain.
* **Outcome**: By simplifying the system to use just MediaPipe BlazeFace and HAAR Cascades, I maintained a balance of high accuracy and ease of use. This approach proved efficient for detecting faces in real-time, whether from a live feed or a video file, and was capable of handling both single and multiple faces effectively.

OUTPUT SAMPLE : 

**Usage Instructions:**

**Please run face\_Detection\_ensemble.py from provided github repository for inference**

Section 3)INTERMEDIATE FACE TRACKING :

### **Prototype Development: Real-Time Face Detection and Tracking Using YOLO and SORT**

#### **Capabilities**

In this prototype development task, I implemented a face detection system using YOLO (You Only Look Once) pretrained models integrated with SORT (Simple Online and Realtime Tracking). This approach detects faces in real-time, assigns unique IDs to each face, and works seamlessly with both live feeds and video files. The system is more stable compared to the previously used ensemble method.

#### **Description of the Code**

The provided script follows a structured approach to real-time face detection and tracking using YOLO and SORT:

1. **Initialization**:
   * **YOLO Model**: The YOLO model is loaded using the configuration (face.cfg) and weights (face.weights). The model is set up to detect faces with high accuracy.
   * **SORT Tracker**: The SORT tracker is initialized to handle the tracking of detected faces, assigning unique IDs to each detected face.
2. **Video Capture**:
   * The video capture is initialized, which can be either a live feed or a pre-recorded video file.
3. **YOLO Face Detection**:
   * Each frame is processed to detect faces using YOLO. The frame is converted to a blob, which is then passed through the YOLO network to obtain detections.
   * Detected faces are stored along with their confidence scores.
4. **Non-Maximum Suppression (NMS)**:
   * NMS is applied to filter out overlapping bounding boxes, retaining only the most confident detections.
5. **SORT Tracking**:
   * The SORT tracker is updated with the current frame's detections, assigning and maintaining unique IDs for each detected face.
   * The tracked faces are drawn on the frame with their respective IDs.
6. **Display**:
   * The processed frame, now with bounding boxes and IDs, is displayed to the user. The loop continues until the user interrupts by pressing the 'q' key.

#### **Challenges Faced**

**Choosing the Right Version of YOLO**:

* **Difficulty**: Figuring out the most suitable version of YOLO was challenging. Initially, using YOLOv3 trained for faces provided good outputs but was relatively slow.
* **Bounding Box Fluctuations**: Sometimes, the bounding boxes fluctuated, causing different IDs to be assigned to the same face.

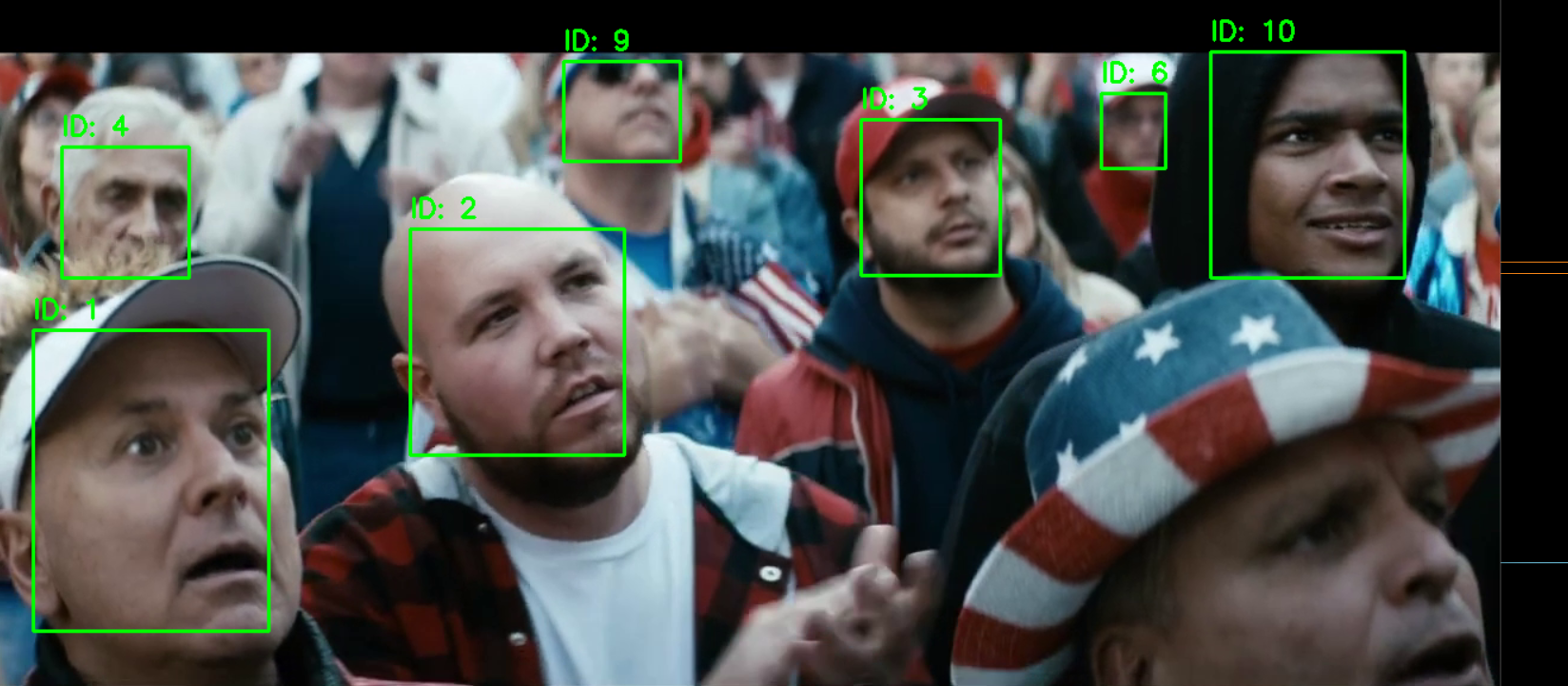
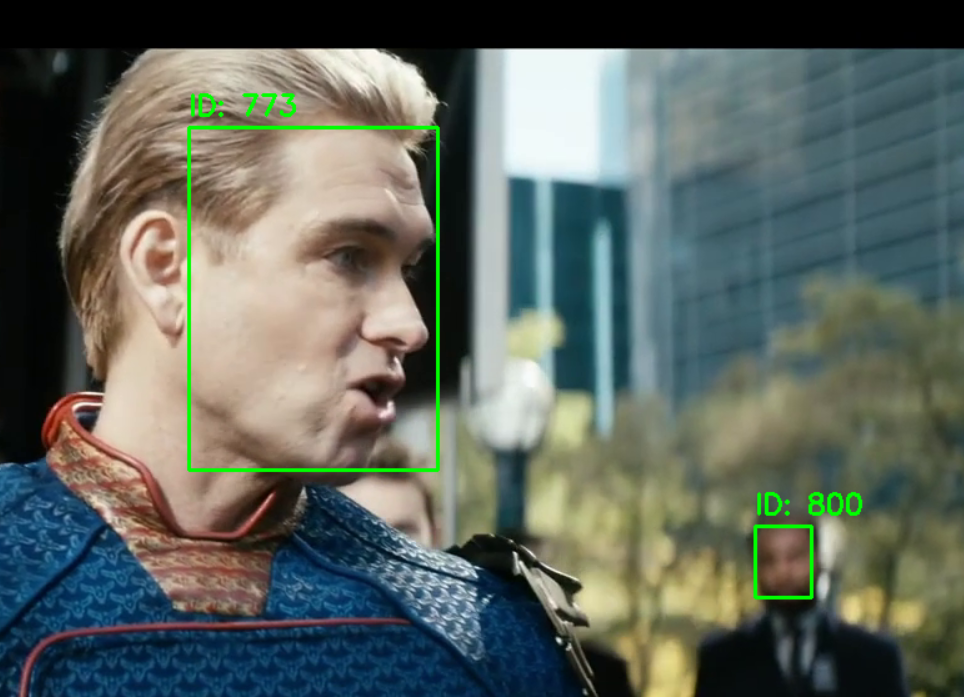
#### **Solutions Implemented**

**Model Selection**:

* **Experimentation**: I experimented with different versions of YOLO to find a balance between speed and computational efficiency.
* **Learning**: Leveraging the Ultralytics official documentation helped in understanding the nuances of various YOLO versions and their uses.

**Simplification**:

* **Decision**: I decided to move forward without including the ensemble of models since adding YOLO did not significantly enhance detection capabilities and increased the system's complexity.
* **Outcome**: Using only YOLO with SORT resulted in a simpler, more stable, and efficient system for real-time face detection and tracking.

OUTPUT IMAGES: 

**Usage Instructions:  
Please run the yolo\_with\_SORT.py file from the github repository provided. A small note would be that since the video inference for this one was quite slow , the video input has been sped up (this has been fixed later)**

Section 4)ADVANCED FACE TRACKING :

### **Multi-face Detection and Tracking: Implementation Using DeepSORT and BoT-SORT with YOLOv8n-Face**

#### **Capabilities**

In this task, I modified the basic algorithm to detect and track multiple faces simultaneously in the video stream. After evaluating various architectures and models, I opted for DeepSORT and BoT-SORT integrated with the YOLOv8n-face pretrained model. These models provide the most stable detections to date. The final submission includes two different models:

1. DeepSORT with YOLOv8n-face.
2. BoT-SORT with YOLOv8n-face.

These models not only track faces accurately but also plot the center of the bounding boxes, creating track lines that follow the face detections over time. This setup works for both live feeds and video inputs, and it assigns unique IDs to detected faces, ensuring consistent tracking. It is more stable compared to the previously used ensemble method.

#### **Challenges Faced**

This phase presented numerous challenges as I experimented with various architectures to improve accuracy:

1. **ResNet50 with DeepSORT**:
   * **Issue**: The combination yielded poor results, failing to detect and track faces reliably.
2. **YOLO's Instance Segmentation**:
   * **Issue**: Although instance segmentation was promising, I was unable to limit detections to faces only.
3. **YOLOv8N Architecture**:
   * **Issue**: The standard YOLOv8N detected multiple objects, not just faces. Attempts to fine-tune the model using the WIDER\_FACE dataset and isolate the human\_faces class were unsuccessful.
4. **Version Compatibility**:
   * **Issue**: The DeepSORT repository used had deprecated items, causing compatibility issues.
5. **BoT-SORT Implementation**:
   * **Issue**: Unlike DeepSORT, manual implementation of BoT-SORT was fraught with errors that I could not resolve within the given time frame.

#### **Solutions Implemented**

1. **Resolving Deprecated Items**:
   * **Solution**: I addressed deprecated item errors in the DeepSORT repository by looking up solutions on Stack Overflow and updating the code accordingly.
2. **Testing YOLO Versions**:
   * **Solution**: I tested numerous versions of YOLO to identify the most effective one for face detection. I found that the YOLO-FACES repository offered a pretrained YOLOv8 model specifically for detecting human faces.
3. **Ultralytics Library for BoT-SORT**:
   * **Solution**: Due to difficulties with manual implementation, I used the Ultralytics library, which integrates BoT-SORT automatically. This integration provided not only facial bounding boxes but also a few landmarks, enhancing the tracking capability.

### **Description of the Code**

The provided script uses DeepSORT with the YOLOv8n-face model to achieve multi-face detection and tracking:

1. **Initialization**:
   * **YOLO Model**: The YOLOv8n-face model is loaded for accurate face detection.
   * **DeepSORT Tracker**: DeepSORT is initialized with a cosine distance metric and a feature encoder.
2. **Video Capture**:
   * The video capture is initialized to read from a video file.
3. **YOLO Face Detection**:
   * Each frame is processed to detect faces using the YOLOv8n-face model. The frame is converted to a blob, which is then passed through the YOLO network to obtain detections.
   * Detected faces are stored along with their confidence scores.
4. **Non-Maximum Suppression (NMS)**:
   * NMS is applied to filter out overlapping bounding boxes, retaining only the most confident detections.
5. **DeepSORT Tracking**:
   * The DeepSORT tracker is updated with the current frame's detections, assigning and maintaining unique IDs for each detected face.
   * The tracked faces are drawn on the frame with their respective IDs.
6. **Plotting Movement**:
   * The center of each bounding box is calculated and tracked over time.
   * Lines connecting these centers are drawn to show the movement of each face.
7. **Display**:
   * The processed frame, with bounding boxes, IDs, and movement plots, is displayed to the user.
   * The loop continues until the user interrupts by pressing the 'q' key.

### **Summary**

**Final Submission**:

* **Models**:
  + **DeepSORT with YOLOv8n-face**: Provides stable multi-face detection and tracking.
  + **BoT-SORT with YOLOv8n-face**: Enhances stability and includes facial landmarks.

**Capabilities**:

* Works for both live feed and video inputs.
* Assigns unique IDs to detected faces, ensuring consistent tracking.
* Tracks the center of bounding boxes to create visual track lines.

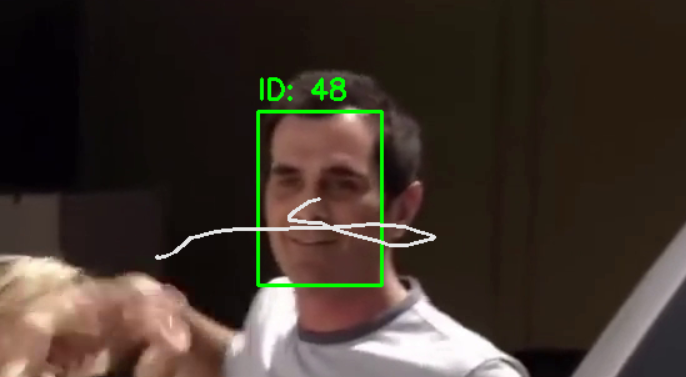
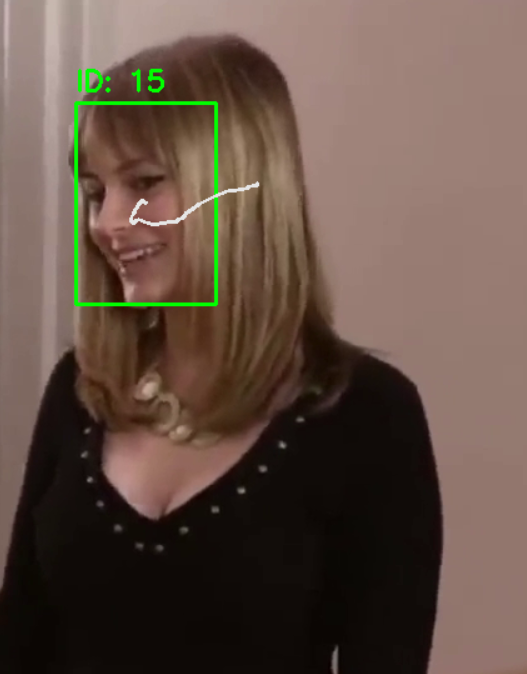
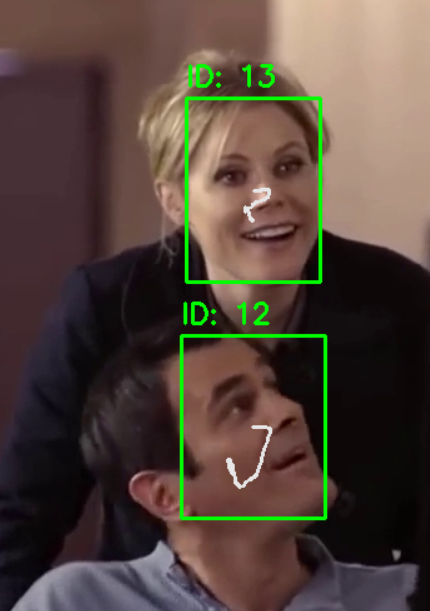
**Challenges**:

* Numerous attempts with other architectures were unsuccessful, particularly due to poor detection accuracy and version compatibility issues.
* The main difficulties included finding a face-specific model and handling deprecated items in DeepSORT.

**Solutions**:

* Systematically resolved deprecated item errors.
* Tested various YOLO models to find the most suitable one.
* Leveraged the YOLO-FACES repository for a pretrained YOLOv8 model tailored for face detection.
* Used the Ultralytics library for seamless BoT-SORT integration, resulting in a robust and efficient multi-face detection and tracking system.

OUTPUT IMAGES:



**Usage Instructions :**

**My final submission is the deepsort\_tracking\_plot.py file , please run that from the repository provided**