Analysis-1:Number of Dribbles Performed by the Player Using computer vision

Analysis Methods

Basketball dribble video involves several key methods aimed at extracting meaningful insights from the footage

1)Object Detection:

- **Method**: Utilized the YOLO (You Only Look Once) algorithm.
- **Description**: YOLO is a popular algorithm for object detection tasks, known for its speed and accuracy. It divides the image into a grid and predicts bounding boxes and class probabilities for each grid cell simultaneously. YOLOv5, specifically, is a variant of the YOLO algorithm known for its efficiency and ease of use.
- **Purpose**: The object detection method was employed to identify and locate the basketball within each frame of the video footage. This information serves as the basis for further analysis, such as dribble counting and player tracking.

2) Dribble Counting:

- **Method**: Custom algorithm based on analyzing the vertical movement of the basketball.
- **Description**: This method involves tracking the vertical position of the basketball in consecutive frames and applying thresholding to detect significant upward or downward movements indicative of dribbles. A dribble is counted when the vertical movement exceeds a predefined threshold. Additionally, measures are taken to prevent double-counting of dribbles.
- **Purpose**: Dribble counting provides quantitative insights into the player's dribbling performance, including dribble frequency and

control. This information can be valuable for assessing player skill and technique.

Algorithms Used

1. YOLO (You Only Look Once) Object Detection Algorithm:

- **Purpose:** Detecting and localizing objects, specifically basketballs, within video frames.
- **Description:** YOLO is a state-of-the-art object detection algorithm that divides the image into a grid and predicts bounding boxes and class probabilities directly from the entire image.
- **Usage:** The YOLOv5 model from the Ultralytics library is utilized for detecting basketballs in each frame of the video.
- Implementation: The model is applied to the video frames, and bounding boxes around detected basketballs are obtained, providing their coordinates for further analysis.

2. Custom Dribble Counting Algorithm:

- **Purpose:** Counting dribbles based on the vertical movement of the basketball.
- **Description:** The algorithm tracks the vertical position of the basketball in consecutive frames and determines dribbles based on predefined thresholds for upward and downward movements.
- **Implementation:**
    - It calculates the change in vertical position between consecutive frames.
    - It detects upward and downward movements exceeding predefined thresholds.

- It counts dribbles accordingly and avoids double-counting by implementing logic to increase the dribble count only after a certain number of frames since the last dribble.

3. Frame Processing Algorithm:

- **Purpose:** Processing each frame of the video for analysis.
- **Description:** This algorithm iterates through each frame of the video, applies object detection for basketball detection, and performs dribble counting.
- **Implementation:**
  - It reads each frame from the video file.
  - It applies the YOLO object detection model to detect basketballs in the frame.
  - It analyzes the detected basketballs' positions to count dribbles using the custom dribble counting algorithm.

4. Thresholding Algorithm:

- **Purpose:** Establishing thresholds for detecting dribbles based on vertical movement.
- **Description:** This algorithm defines thresholds for upward and downward movements of the basketball to identify dribbles accurately.
- **Implementation:**
  - It sets thresholds for the minimum vertical displacement required to count a dribble.
  - It ensures that only movements exceeding these thresholds are considered as dribbles, filtering out noise and minor fluctuations.

Challenges faced during implementation.

1) Model Selection and Configuration:

- **Challenge:** Selecting the appropriate object detection model and configuring it for accurate basketball detection posed a challenge.

Different models have varying complexities, inference speeds, and accuracies, making the selection process non-trivial.

- **Solution**: Extensive experimentation and evaluation of multiple object detection models were conducted to identify the most suitable one for the task. Fine-tuning of model parameters, such as confidence thresholds, was performed to optimize detection accuracy while minimizing false positives.

2) Dribble Detection Threshold Tuning:

- **Challenge:** Determining the appropriate threshold for detecting dribbles based on the vertical movement of the basketball was challenging. Setting the threshold too low could result in false positives, while setting it too high could lead to missed dribble detections.
- **Solution:** A thorough analysis of the video footage and dribble patterns was conducted to establish an empirical threshold that accurately captured dribble events. Iterative testing and adjustment of the threshold value were performed to achieve optimal results.

3) Real-time Processing and Performance Optimization:

- **Challenge:** Ensuring real-time or near real-time processing of video frames while performing object detection and dribble counting presented a significant challenge. Processing large video files with high-resolution frames in real-time requires efficient algorithms and optimized code.
- **Solution:** Code optimization techniques, such as parallelization, asynchronous processing, and leveraging hardware acceleration (e.g., GPU), were employed to enhance processing speed and efficiency. Additionally, frame skipping or downscaling techniques were considered to reduce computational overhead while maintaining analysis accuracy.

4) Handling Complex Dribble Scenarios:

- **Challenge:** Dealing with complex dribble scenarios, such as occlusions, erratic ball movements, and variations in dribbling styles, presented challenges for accurate detection and counting of dribbles.
- **Solution:** Robust algorithms and heuristics were implemented to handle occlusions and irregular ball movements. Techniques such as motion smoothing, trajectory prediction, and adaptive thresholding were employed to improve detection accuracy and resilience to challenging scenarios.

Code documentation

1. `import cv2`: This line imports the OpenCV library, which is a popular open-source computer vision and image processing library in Python.
2. `from ultralytics import YOLO`: This line imports the YOLO object detection model from the Ultralytics library. YOLO is a deep learning-based object detection algorithm that can detect objects in images and videos.
3. `class DribbleCounter:`: This line defines a Python class named `DribbleCounter`, which encapsulates the functionality for detecting and counting dribbles in a basketball video.
4. `def __init__(self, video_path):`: This line defines the constructor method of the `DribbleCounter` class, which is called when an instance of the class is created. It initializes the object with the provided `video_path`, which specifies the path to the input video file.
5. `self.model = YOLO("C:/Users/dilee/Downloads/yolov8s.pt")`: This line initializes an instance of the YOLO object detection model with the specified pre-trained weights file ("yolov8s.pt").
6. `self.cap = cv2.VideoCapture(video_path)`: This line creates a VideoCapture object using the OpenCV library, which is used to read frames from the input video file specified by `video_path`.

7. `self.prev_y_center = None`: This line initializes a variable to store the vertical position of the basketball in the previous frame. It is set to `None` initially.
8. `self.prev_delta_y = None`: This line initializes a variable to store the change in vertical position of the basketball between consecutive frames. It is set to `None` initially.
9. `self.dribble_count = 0`: This line initializes a variable to store the number of dribbles detected in the video. It is initially set to 0.
10. `self.dribble_threshold = 197`: This line sets a threshold value for the vertical change in position of the basketball to be considered as a dribble.
11. `self.increase_count_frames = 200`: This line specifies the number of frames that need to pass after detecting a dribble before incrementing the dribble count.
12. `self.frames_since_last_increase = 0`: This line initializes a variable to keep track of the number of frames that have passed since the last dribble was detected.
13. `self.upward_movement = False` and `self.downward_movement = False`: These lines initialize boolean variables to track whether the basketball is currently moving upward or downward in the frame.
14. `def run(self):`: This line defines a method named `run` within the `DribbleCounter` class, which is responsible for processing each frame of the input video and detecting dribbles.
15. `while self.cap.isOpened():`: This line starts a loop that iterates over each frame of the input video until the end of the video is reached or the user interrupts the process.
16. `success, frame = self.cap.read()`: This line reads the next frame from the input video and stores it in the variable `frame`.
17. `results_list = self.model(frame, verbose=False, conf=0.65)`: This line passes the current frame to the YOLO object detection model to detect objects, such as basketballs, in the frame. The detected objects are stored in `results_list`.

18. `for results in results_list:`: This line iterates over each detection result in the `results_list`.
19. `for bbox in results.boxes.xyxy:`: This line iterates over each bounding box detected by the YOLO model.
20. `x1, y1, x2, y2 = bbox[:4]`: This line extracts the coordinates of the top-left and bottom-right corners of the bounding box.
21. `y_center = (y1 + y2) / 2`: This line calculates the vertical center of the bounding box, which represents the vertical position of the basketball in the frame.
22. `self.update_dribble_count(y_center)`: This line calls the `update_dribble_count` method to update the dribble count based on the detected position of the basketball.
23. `annotated_frame = results.plot()`: This line generates an annotated version of the frame with the detected objects and their labels.
24. `cv2.putText(annotated_frame, f"Dribbles: {self.dribble_count}", (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)`: This line adds text to the annotated frame indicating the current dribble count.
25. `cv2.imshow("YOLOv8 Inference", annotated_frame)`: This line displays the annotated frame with the dribble count overlaid on it.
26. `if cv2.waitKey(20) & 0xFF == ord("q"):`: This line checks for the 'q' keypress event, which is used to quit the application.
27. `self.cap.release()`: This line releases the VideoCapture object, releasing the input video file.
28. `cv2.destroyAllWindows()`: This line closes all OpenCV windows.
29. `def update_dribble_count(self, y_center):`: This line defines a method named `update_dribble_count` within the `DribbleCounter` class, which is responsible for updating the dribble count based on the vertical position of the basketball in the frame.
30. `if self.prev_y_center is not None:`: This line checks if the previous vertical position of the basketball is available.

31. `delta_y = y_center - self.prev_y_center`: This line calculates the change in vertical position of the basketball between consecutive frames.
32. `if delta_y > self.dribble_threshold and not self.upward_movement:`: This line checks if the change in vertical position exceeds the dribble threshold and if the basketball is currently moving upward.
33. `self.upward_movement = True`: This line sets the `upward_movement` flag to indicate that the basketball is moving upward.
34. `if delta_y < -self.dribble_threshold and self.upward_movement:`: This line checks if the change in vertical position is negative (indicating downward movement) and if the basketball was previously moving upward.
35. `self.dribble_count += 1`: This line increments the dribble count when a dribble action is detected.
36. `if self.frames_since_last_increase >= self.increase_count_frames:`: This line checks if the cooldown period has elapsed since the last dribble.
37. `self.frames_since_last_increase += 1`: This line increments the counter for the number of frames since the last dribble.
38. `if __name__ == "__main__":`: This line checks if the script is being run directly (as opposed to being imported as a module).
39. `video_path = "C:/Users/dilee/Downloads/WHATSAAP ASSIGNMENT.mp4"`: This line specifies the path to the input video file.
40. `dribble_counter = DribbleCounter(video_path)`: This line creates an instance of the `DribbleCounter` class with the specified video path.
41. `dribble_counter.run()`: This line calls the `run` method to start processing the video and detecting dribbles.

Analysis-2:Speed of the Dribble Using computer vision Analysis Methods

1. **Object Detection and Tracking:**
    - The method employs object detection and tracking techniques to locate the basketball within each frame of the input video.
    - Preprocessing steps such as converting frames to grayscale and applying Gaussian blur are used to enhance the detectability of the basketball.
    - Basic thresholding is applied to detect bright objects, assuming that the basketball appears brighter than its surroundings.
    - The method identifies the largest contour in the thresholded image as the basketball and computes its position as the center of the enclosing circle.
2. **Distance Calculation:**
    - For each pair of consecutive frames, the method calculates the distance moved by the basketball.
    - It computes the Euclidean distance between the centers of the basketball in the current and previous frames.
3. **Speed Estimation:**
    - After processing all frames, the method estimates the dribble speed of the basketball.
    - The average dribble speed is calculated by dividing the total distance moved by the basketball by the product of the assumed frame rate and the total number of frames processed.

Algorithms Used

1. **Gaussian Blur:**
    - Applied to the grayscale version of each frame to reduce noise and improve the accuracy of object detection.
2. **Thresholding:**

- A basic thresholding technique is used to segment the image and detect bright objects. It assumes that the basketball appears brighter than its surroundings.

3. **Contour Detection:**
   - The contours of the thresholded image are detected using the `cv2.findContours()` function, which identifies connected components in the binary image.
4. **Minimum Enclosing Circle:**
   - The minimum enclosing circle of the largest contour is computed using `cv2.minEnclosingCircle()`. This circle provides an approximation of the basketball's position and size.
5. **Euclidean Distance:**
   - The distance between the centers of the basketball in consecutive frames is calculated using the Euclidean distance formula. This distance represents the movement of the basketball between frames.
6. **Average Speed Calculation:**
   - The average dribble speed is calculated by dividing the total distance moved by the basketball by the product of the assumed frame rate and the total number of frames processed.

Challenges Faced During the Implementation

1. **Object Detection Accuracy:**
   - The accuracy of object detection can be affected by factors such as varying lighting conditions, shadows, reflections, and occlusions. Ensuring reliable detection of the basketball in different situations can be challenging.
2. **Noise and Distortion:**

- Noise in the video frames and distortion caused by motion blur or camera movement can impact the accuracy of object tracking. Preprocessing techniques like Gaussian blur may help reduce noise, but handling motion blur can be more challenging.

3. **Parameter Tuning:**
   - The effectiveness of thresholding and contour detection techniques heavily depends on parameter tuning. Finding the right threshold values and contour parameters to accurately detect the basketball while minimizing false positives and negatives can be time-consuming.
4. **Performance Optimization:**
   - Processing video frames in real-time or at high speeds may require optimization to ensure efficient use of computational resources. Implementing algorithms that can handle large video files efficiently while maintaining real-time performance can be challenging, especially on resource-constrained devices.
5. **Robustness to Scene Variability:**
   - The code assumes that the basketball is the brightest object in the scene. However, in real-world scenarios, the appearance of the basketball may vary depending on factors like lighting, background clutter, and the presence of other objects. Ensuring the robustness of the algorithm to such variability is essential.
6. **Handling Occlusions and Partial Visibility:**
   - Occlusions and partial visibility of the basketball, such as when it goes behind a player's body or another object, can pose challenges for accurate tracking. Strategies for handling occlusions and predicting object motion in such situations may be necessary to improve tracking performance.
7. **User Interaction and Error Handling:**
   - Providing a user-friendly interface for controlling the video playback and handling errors gracefully, such as when the video

file cannot be read or processed, is essential for a smooth user experience.

Code Documentation

1. Importing Libraries:
    - `import cv2`: Imports the OpenCV library for image and video processing.
    - `import numpy as np`: Imports the NumPy library for numerical computations.
2. Distance Calculation Function:
    - `def distance(point1, point2)`: Defines a function to calculate the Euclidean distance between two points in a two-dimensional space.
3. Video Capture Initialization:
    - `cap = cv2.VideoCapture("C:/Users/SUNEETHA/Desktop/Internshala_assignment/stealth_health/main/WHATSAAP ASSIGNMENT.mp4")`: Initializes the video capture object with the specified video file path.
4. Variable Initialization:
    - `prev_ball_position = None`: Initializes the variable to store the previous position of the basketball.
    - `total_distance = 0`: Initializes the variable to store the total distance moved by the basketball.
    - `frame_count = 0`: Initializes the variable to count the number of frames processed.
5. Video Processing Loop:
    - `while(cap.isOpened()):`: Initiates a loop to process each frame of the video until the end.

- `ret, frame = cap.read()`: Reads the next frame from the video capture object.
- `if not ret: break`: Breaks out of the loop if there are no more frames to read.

6. Preprocessing:
   - `gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)`: Converts the frame to grayscale for easier processing.
   - `blurred = cv2.GaussianBlur(gray, (5, 5), 0)`: Applies Gaussian blur to the grayscale frame to reduce noise and smooth edges.
7. Object Detection:
   - `_, thresh = cv2.threshold(blurred, 220, 255, cv2.THRESH_BINARY)`: Applies thresholding to create a binary image, highlighting bright objects.
   - `contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)`: Finds contours in the binary image to identify objects.
8. Basketball Position Calculation:
   - `c = max(contours, key=cv2.contourArea)`: Selects the contour with the largest area, assuming it corresponds to the basketball.
   - `((x, y), radius) = cv2.minEnclosingCircle(c)`: Finds the minimum enclosing circle around the contour to locate the basketball position.
9. Distance Calculation and Accumulation:
   - `if prev_ball_position:`: Checks if the previous basketball position is available.

- `total_distance += distance(prev_ball_position, ball_position)`: Calculates the distance moved by the basketball and adds it to the total distance.

10. Displaying Frames:
    - `cv2.imshow('Frame', frame)`: Displays the current frame with the detected basketball.
11. User Interaction:
    - `if cv2.waitKey(25) & 0xFF == ord('q'):`: Waits for the user to press the 'q' key to exit the video processing loop.
12. Frame Count Increment:
    - `frame_count += 1`: Increments the frame count to track the number of frames processed.
13. Dribble Speed Calculation:
    - `dribble_speed = total_distance / (fps * frame_count)`: Calculates the average dribble speed in pixels per frame based on the total distance moved and the total number of frames processed.
14. Output:
    - `print("Average dribble speed:", dribble_speed, "pixels/frame")`: Prints the average dribble speed to the console.
15. Resource Release:
    - `cap.release()`: Releases the video capture object.
    - `cv2.destroyAllWindows()`: Closes all OpenCV windows.

Analysis-3: Distance covered During dribble

Analysis Methods

1. **Background Subtraction:**

- Background subtraction is utilized to segment moving objects, presumably the basketball, from the static background. This method helps isolate the basketball's motion in the video frames.

2. **Contour Detection:**
   - Following background subtraction, contours are detected within the foreground mask using the `cv2.findContours()` function. Contours represent the boundary of segmented objects, enabling the identification of the basketball's location.
3. **Dribble Distance Calculation:**
   - The code calculates the distance covered by the basketball during dribbles by computing the Euclidean distance between consecutive centroid positions of the detected basketball. This distance is accumulated over multiple frames to determine the total dribble distance covered.
4. **Visualization:**
   - To aid in result interpretation and validation, the code visualizes the detected basketball's position on each frame by overlaying a circle marker. This visualization provides a qualitative assessment of the accuracy of the object detection and tracking process.
5. **Total Dribble Distance Computation:**
   - The code accumulates the dribble distances calculated for each frame to determine the total distance covered by the basketball during all dribbles throughout the entire video. This total distance provides a quantitative measure of the basketball player's movement dynamics during dribbling activities.
6. **Unit Conversion:**
   - Assuming knowledge of the physical dimensions of the basketball court, the code converts the total dribble distance from pixels to meters. This conversion enables the interpretation of the results in real-world terms, facilitating a better understanding of the basketball player's performance.

7. **Resource Management:**
    - Upon completion of the analysis, the code releases the video capture object and closes any OpenCV windows opened during execution. This ensures efficient resource utilization and prevents potential memory leaks.

Methods Used

1. **Background Subtraction:**
    - Background subtraction is employed to segment moving objects, such as the basketball, from the static background in each frame of the video. This technique helps isolate the regions of interest where motion occurs.
2. **Contour Detection:**
    - Following background subtraction, the code utilizes contour detection to identify the boundaries of the segmented foreground objects. Contours represent the outline of objects in an image and are useful for object localization and analysis.
3. **Object Centroid Calculation:**
    - After detecting contours, the code computes the centroid of the largest contour, which presumably corresponds to the basketball. The centroid is calculated using the moments of the contour, providing a reference point for the basketball's position in the frame.
4. **Euclidean Distance Calculation:**
    - The code calculates the Euclidean distance between consecutive centroid positions of the detected basketball across frames. This distance metric quantifies the movement of the basketball during dribbling activities and is used to measure the distance covered during each dribble.
5. **Unit Conversion:**
    - Assuming knowledge of the physical dimensions of the basketball court, the code converts the dribble distance from

pixels to meters. This conversion enables the interpretation of the results in real-world units, facilitating a better understanding of the basketball player's movement dynamics.

6. **Background Model Update (BackgroundSubtractorMOG2):**
    - The code utilizes the Gaussian Mixture-based Background/Foreground Segmentation algorithm (MOG2) to model the background of the video. This algorithm adapts the background model over time to accommodate changes in lighting conditions and scene dynamics.

Challenges Faced During Implementation

1. **Background Variability:**
    - Variations in the background scene, such as changes in lighting conditions, moving objects (e.g., spectators), or occlusions, can complicate background subtraction and result in false positives or missed detections of the basketball.
2. **Noise and Artifacts:**
    - Noise in the video feed, including sensor noise, compression artifacts, or sudden camera movements, can introduce unwanted elements into the foreground mask, leading to inaccurate contour detection and tracking.
3. **Adaptive Thresholding:**
    - Determining optimal parameters for background subtraction methods, such as the history length and variance threshold in the MOG2 algorithm, may require experimentation and tuning to adapt to different video sequences and environmental conditions.
4. **Contour Identification:**
    - Identifying the correct contour corresponding to the basketball, especially in crowded scenes or when multiple objects are present, can be challenging. This task may require additional criteria, such as size, shape, or motion characteristics, to accurately discriminate the basketball contour from other objects.

5. **Dribble Detection:**
    - Distinguishing between dribbling movements and other types of ball motion (e.g., passing, shooting) can be challenging, particularly when the basketball interacts with the player's hands or other body parts. Robust algorithms are needed to reliably detect dribbles while minimizing false positives.
6. **Frame Rate and Motion Blur:**
    - Low frame rates or motion blur in the video can affect the accuracy of motion estimation and distance calculation. Higher frame rates and motion deblurring techniques may be necessary to improve the precision of dribble distance measurements.
7. **Calibration and Scaling:**
    - Accurately converting pixel distances to real-world units (e.g., meters) requires knowledge of the spatial scale of the scene, such as the dimensions of the basketball court. Calibration errors or inaccurate scaling factors can introduce uncertainties into the dribble distance analysis.

Code Documentation

1. Import Libraries:
    - `import cv2`: Imports the OpenCV library, which is used for computer vision tasks.
    - `import numpy as np`: Imports the NumPy library, which is used for numerical computations.
2. Calculate Distance Function:
    - `calculate_distance(point1, point2)`: Defines a function to calculate the Euclidean distance between two points in a 2D space.
3. Video Capture:
    - `video_path = '.../WHATSAAP ASSIGNMENT.mp4'`: Specifies the path to the basketball video file.

- `cap = cv2.VideoCapture(video_path)`: Opens a video capture object to read frames from the specified video file.

4. Variable Initialization:
   - `prev_ball_pos`, `dribble_distance`, `total_dribble_distance`: Initialize variables for tracking the previous ball position, dribble distance in each frame, and total dribble distance, respectively.
   - `fgbg = cv2.createBackgroundSubtractorMOG2(...)`: Creates a background subtractor object using the MOG2 algorithm with specified parameters for foreground/background segmentation.
5. Main Loop Through Video Frames:
   - `while(cap.isOpened()):`: Starts a loop to iterate through each frame of the video until the end.
   - `ret, frame = cap.read()`: Reads the next frame from the video capture object.
   - `if not ret: break`: Checks if the frame was successfully read; if not, breaks out of the loop.
   - `fgmask = fgbg.apply(frame)`: Applies background subtraction to the current frame to obtain a foreground mask.
   - `contours, _ = cv2.findContours(...)`: Finds contours in the foreground mask using the RETR_EXTERNAL mode.
   - `max_contour = max(contours, key=cv2.contourArea)`: Finds the contour with the largest area, assumed to be the basketball.
   - `moments = cv2.moments(max_contour)`: Computes moments of the largest contour to find its centroid (ball position).

- `ball_position = (int(moments["m10"] / moments["m00"]), int(moments["m01"] / moments["m00"]))`: Calculates the centroid coordinates.
- `if prev_ball_pos is not None: ...`: Computes the distance covered by the basketball in each frame using the `calculate_distance` function and updates `dribble_distance`.
- `prev_ball_pos = ball_position`: Updates the previous ball position for the next iteration.
- `cv2.circle(frame, ball_position, 5, (0, 0, 255), -1)`: Draws a circle at the ball position on the frame for visualization.
- `cv2.imshow('Frame', frame)`: Displays the current frame with the detected basketball position.
- `if cv2.waitKey(25) & 0xFF == ord('q'): break`: Checks for the 'q' key press to exit the loop.

6. Dribble Distance Calculation:
   - `total_dribble_distance += dribble_distance`: Accumulates the dribble distance for each frame to compute the total dribble distance.
7. Release Resources and Display Results:
   - `cap.release()`: Releases the video capture object.
   - `cv2.destroyAllWindows()`: Closes all OpenCV windows.
   - Prints the total dribble distance in pixels and meters using the calculated pixel-to-meter ratio.