

Soccer Events

Title	Soccer Events								
Area/Tech	VisionAI								
Version	1								
Start Date	6 May 2024								
End Date									
Document Contributors	<div>@Telma Garção @Luis André @Joana Sousa</div> <div>@Bruno Fernandes @Joao Ferreira @Pedro Malheiro</div>								
Document Status	DRAFT								
JIRA	<div><table><tr><th>Type</th><th>Key</th><th>Summary</th><th>Assignee</th></tr><tr><td><input checked="" type="checkbox"/></td><td>IPM-300</td><td>Soccer Events</td><td><div><div></div>Telma Garção</div></td></tr></table><div>1 item Synced just now </div></div>	Type	Key	Summary	Assignee	<input checked="" type="checkbox"/>	IPM-300	Soccer Events	<div><div></div>Telma Garção</div>
Type	Key	Summary	Assignee						
<input checked="" type="checkbox"/>	IPM-300	Soccer Events	<div><div></div>Telma Garção</div>						
Project Dependencies	<div> Action-Audio experience for low-sight audiences</div>								
Project Sharepoint	<div> https://nostechnology.sharepoint.com/:f:/r/sites/TEAM_DESENPRODUTO_NOSinovacao-IdeationandInnovation/Documentos%20Partilhados/Ideation%20and%20Innovation/07%20-%20POCs/Action-Audio%20IPL?csf=1&web=1&e=vDbJF9 Connect your OneDrive account</div>								
Overleaf									
Git Repository	<div> https://github.com/nosportugal/nosi-fusion-innovation-soccer-events Connect your Github account</div>								
GCP Bucket	-								
IPR	-								
Demo	-								

Challenge

Goals

Goal	Description
Goal 1	Description 1

Goal 2	Description 2
Goal 3	Description 3

Literature Review

Football Analysis: [GitHub - FootballAnalysis/footballanalysis](#)

Tarefas principais

- Localização dos objectos (jogadores, bola)
 - Problema de detecção YOLO
- Tracking dos objectos
 - Perceber onde estão no espaço 3d
 - Traduzir esta localização
- Transformação de perspectiva: imagem 3D (vídeo original) -> 2D Bird's EyeView
 - Uso de algebra com matrizes

1. Object Detection with YOLO

YOLO is a deep learning-based object detection algorithm that simultaneously identifies objects in an image (like players, referees, and the ball) and locates them by drawing bounding boxes around these objects. In this project, the YOLO model is used to detect:

- **Players**
- **Goalkeepers**
- **Referees**
- **The Ball**

YOLO is a single-pass detection method, meaning it runs through the image just once to detect objects.

2. Optical Flow

Optical Flow is a computer vision technique used to estimate the motion of objects or features between two consecutive frames in a video sequence. It calculates how pixels move from one frame to the next. This is crucial in scenarios where objects (or keypoints) need to be tracked when detection fails or when the video frame has missing information (such as occlusions or low-quality frames).

In this project, Optical Flow is particularly useful for tracking **keypoints** across frames when fewer than 4 keypoints are detected in a frame (which is insufficient for calculating the homography matrix). It helps bridge the gap by estimating where the keypoints are likely to have moved.

How Optical Flow Works

Optical flow calculates the movement of individual pixels or groups of pixels by comparing changes in intensity (brightness) between consecutive frames. It assumes that the intensity of a pixel remains the same between two frames, and any difference is due to movement.

Mathematically, for a pixel at position (x, y) in frame $I(x, y, t)$, where t denotes time (the current frame), the basic optical flow constraint equation is:

$$I(x,y,t)=I(x+dx,y+dy,t+dt) \quad I(x, y, t) = I(x + dx, y + dy, t + dt) \quad I(x,y,t)=I(x+dx,y+dy,t+dt)$$

Where:

- dx, dy is the displacement of the pixel in the next frame.
- dt is the time difference between the two frames.

To compute this, **Lucas-Kanade** optical flow method is typically used, which estimates motion by minimizing the difference between pixel intensities. In this project, Lucas-Kanade is implemented with:

- A window size (search area) of (15, 15)
- Maximum pyramid levels of 2
- Termination criteria: either a small movement or a maximum of 10 iterations.

In the project, Optical Flow is used to **track keypoints** that have already been detected in previous frames but might not be visible in the current frame. If fewer than 4 keypoints are detected by the HRNet model, the system uses Optical Flow to estimate the position of the missing keypoints.

Here's a step-by-step process of how optical flow is applied:

1. **Previous Keypoints:** The system stores the positions of keypoints detected in the previous frame.
2. **Motion Estimation:** Optical flow is used to estimate how these keypoints have moved in the current frame by analyzing the pixel movements between the previous and current frame.
3. **Filtering:** Not all keypoints are valid after motion estimation. Keypoints that move too much or are in regions where the color has changed drastically (indicating possible occlusion) are filtered out.

This technique ensures smooth transitions in tracking keypoints across frames, even when the keypoint detection model misses some of them due to occlusions or other issues.

3. HRNet (High-Resolution Network)

HRNet is a deep learning model specifically designed for **keypoint detection** and **pose estimation**. In this context, keypoints refer to important landmarks on the football pitch, such as intersections of the field lines or corners. These keypoints are crucial for calculating the homography matrix, which maps image coordinates from the broadcast footage to real-world coordinates on the football pitch.

Unlike other models that downsample the input image to extract features (resulting in low-resolution feature maps), HRNet maintains high-resolution representations throughout the network. This is particularly beneficial for tasks that require precise localization of keypoints, such as detecting field markings in football footage.

Architecture of HRNet

- **Stem Network:** The initial part of HRNet reduces the resolution of the input image slightly while increasing the number of feature channels. This helps capture more abstract features while maintaining a fairly high resolution.
- **Stage 2, 3, 4:** HRNet processes the image through multiple stages (or branches). Each stage handles a different scale of the image and extracts feature maps at various resolutions. These different resolutions are combined through **fusion layers** to produce accurate and detailed feature maps for keypoint detection.
- **Multi-Scale Representation:** HRNet fuses information from different resolutions, ensuring that both fine details (high resolution) and broader contextual information (low resolution) are captured in the output feature maps.

In this project, HRNet is trained to detect **57 keypoints** on the football pitch. These keypoints correspond to important areas like:

- The center circle
- Penalty areas
- The corners
- Line intersections

The keypoints detected in each frame are then used for:

1. **Homography Transformation:** The keypoints serve as reference points to compute the homography matrix, which allows the system to map 2D image coordinates to real-world coordinates on the football pitch.
2. **Calibration:** If keypoints are detected inaccurately, the system can adjust them using the calibration process (explained later) to ensure the keypoints are aligned with visible field markings.

Keypoint Calibration

A key step after detecting the keypoints is **calibration**. This is done to ensure that the keypoints are aligned with the most visible areas on the field. The system checks the brightness around each detected keypoint (in the HSV color space) and, if needed, adjusts the keypoint's position to the brightest spot nearby. This ensures that keypoints are accurately placed on the most visible parts of the field (e.g., lines or corners).

4. Homography Transformation

Homography is a mathematical transformation that maps points from one plane (in this case, the 2D video image) to another plane (the real-world football pitch). Homography is critical in this project because it allows us to take coordinates from the video and convert them into real-world pitch coordinates, enabling accurate tracking of player positions on the field.

To compute a homography matrix, the system needs at least 4 corresponding points between the image and the real world. These points are provided by the keypoints detected by HRNet.

Once these corresponding points are found, **OpenCV's** `findHomography` function is used to compute the homography matrix. This matrix allows the system to transform any point (such as a player's position) from the 2D image space into the real-world pitch space. This transformation is applied to all detected objects (players, referees, the ball) to track their positions on the football pitch.

5. BoTSORT for Object Tracking

Object tracking refers to the task of consistently identifying and following the same object across multiple frames in a video. In this project, once the YOLO model detects objects like players, the ball, and referees in a single frame, the system needs to track these objects over time, ensuring that the same player or ball is correctly identified in every frame.

BoTSORT (Byte-Track-SORT) is an enhanced version of the widely-used **SORT (Simple Online and Realtime Tracking)** algorithm. SORT is a basic tracking algorithm that uses Kalman filtering and the Hungarian algorithm to associate detections with tracked objects. BoTSORT improves upon SORT by adding appearance-based re-identification, allowing for more robust tracking, especially in crowded scenes (such as a football match with many players on the field).

BoTSORT integrates:

1. **Kalman Filter**: Used for predicting the next position of an object based on its current motion (velocity, acceleration, etc.). This helps the tracker make educated guesses about where an object will move in the next frame.
2. **Hungarian Algorithm**: Used for data association. It matches the detected objects in the current frame with the predicted positions of objects from previous frames. This ensures that the correct object IDs are maintained across frames.
3. **Appearance Re-identification**: BoTSORT introduces appearance-based re-identification, where the visual features of objects are used to track them more accurately. This is particularly useful when players or the ball get temporarily occluded or when objects are in close proximity to each other.

In this project, BoTSORT is responsible for **tracking players, referees, and the ball** across multiple video frames. It takes the bounding boxes output by the YOLO object detection model and:

1. Predicts the next position of each object using the Kalman filter.
2. Matches detected objects from the current frame with tracked objects using the Hungarian algorithm.
3. Uses appearance-based features (extracted using a pre-trained model) to re-identify objects when there are ambiguities, such as when multiple players are close together or occlude each other.

By using BoTSORT, the system can robustly track the same player or ball across the entire video sequence, ensuring that the object IDs remain consistent.

Ball Action: Pass and Drive

The **Ball Action Spotting** ([GitHub - IRomul/ball-action-spotting](https://github.com/IRomul/ball-action-spotting): SoccerNet@CVPR | 1st place solution for Ball Action Spotting Challenge e 2023) repository focuses on the challenge of detecting significant actions in football matches, such as passes, drives, and other ball-

related events. The solution, which won first place in the SoccerNet Ball Action Spotting Challenge 2023, leverages a combination of 2D and 3D convolutions to extract both spatial and temporal features from video frames.

Here's a detailed breakdown of the process:

1. Problem Definition

The task is to detect and classify ball actions (e.g., passes, shots) in dense football footage. The challenge involves recognizing these actions in real-time, which requires a precise understanding of the spatial and temporal dimensions of the game. The model needs to learn the patterns of player movement, ball movement, and how they relate to actions like shots, passes, or fouls.

2. Model Architecture

The solution uses a hybrid architecture combining **2D convolutions** for spatial feature extraction and **3D convolutions** for temporal feature extraction:

- **2D Convolutions:** Extract spatial features from each frame. This involves learning from the pixel information within the frame to understand the spatial configuration, such as where players and the ball are positioned.
- **3D Convolutions:** Capture temporal dynamics by processing sequences of frames. This is crucial for understanding actions like a pass or shot, which are defined by how the ball moves over time, as well as the context provided by player movement.

This hybrid approach ensures that the model can understand both the appearance of individual frames (2D) and the changes in those frames over time (3D).

3. Training and Transfer Learning

- **Transfer Learning:** Pre-trained models are utilized to speed up training and improve accuracy. The model is initially trained on a large dataset, learning general features, before being fine-tuned on football-specific data to specialize in detecting actions.
- **Long-Sequence Training:** Instead of processing short clips, the model is trained on long video sequences to better capture the context of actions. This helps in detecting long-range dependencies in the video, which is crucial for spotting actions that unfold over multiple seconds.

4. Efficient Video Processing

The video frames are preprocessed into grayscale to reduce computational overhead, as color information is less important in action detection compared to spatial and temporal patterns. Grayscale video frames also reduce memory consumption, allowing the model to process longer sequences more efficiently.

5. Loss Functions and Optimization

The model uses a loss function that balances between **temporal localization** and **action classification**. The goal is to not only detect when an action happens but also classify it accurately. The optimization process ensures that the model is sensitive to both the exact timing of actions and their specific types.

6. Post-Processing and Action Spotting

Once the model outputs action predictions, a post-processing step is used to refine the detections. This involves applying non-maximum suppression to eliminate duplicate predictions and ensure that each action is only detected once. Additionally, temporal smoothing is used to avoid false positives by considering the context of surrounding frames.

Unifying Global and Local Scene Entities Modelling for Precise Action Spotting: Goal and Goal Try

1. Introduction and Motivation

Action spotting in sports videos, such as soccer matches, is a challenging task that involves accurately identifying moments like passes, shots, or fouls. Traditional methods, which rely heavily on global scene features, often fail to detect actions involving smaller, more subtle elements, like the ball or player gestures. The repository from **UGLF** (Unifying Global and Local Features) ([GitHub - Fsoft-AIC/UGLF: \[IJCNN 2024\] Unifying Global and Local Scene Entities Modelling for Precise Action Spotting](#)) enhances this by combining global scene understanding with local, object-specific detection. This dual approach improves precision, especially for small but important entities like the ball and players.

Previous models suffered from issues such as:

- **Occlusion** of important entities.
- **Dynamic camera shifts** that obscure the flow of action.
- **Class imbalance** with underrepresented action types (e.g., fouls, red cards).

The **UGLF** repository complements these challenges with an end-to-end framework that integrates **global scene features** and **local entity detection** to improve both spatial and temporal action recognition.

2. Key Challenges Addressed by the Hybrid Model

- **Background Clutter**: Sports footage contains extraneous information like crowds or advertisements. Global feature extractors often pick up on this noise.
- **Dynamic Camera Angles**: Fast camera movements in sports introduce abrupt changes, making it difficult to track entities across time.
- **Smaller Action Objects**: Actions involving small items like a football or red/yellow cards are easily missed.
- **Class Imbalance**: Actions like goals or fouls are rare, leading to poor performance in these categories.

3. Model Overview

The proposed hybrid model in UGLF addresses these challenges by combining:

1. **Global Environment Understanding**: Extracts global scene features like field, player distribution, and overall spatial layout.
2. **Local Object Detection**: Focuses on smaller, high-relevance objects such as the ball, players, or referee signals using **object detection** techniques.
3. **Long-Term Temporal Understanding**: Temporal features allow the model to learn long sequences of actions like passes or shots by analyzing multiple frames over time.

4. Unifying Global and Local Features (UGLF) Module

The UGLF architecture integrates **global** and **local** features:

a. Global Features

- **Global Convolutional Layers**: Extract large-scale spatial features from the full frame using **2D Convolutional Neural Networks (CNNs)**. This provides a holistic view of the playing field and helps identify larger action contexts.
- **Temporal Feature Shift (TFS)**: Mimics the behavior of 3D convolutions by shifting global features across time, reducing the need for complex 3D convolutions. The TFS module achieves temporal consistency by processing sequences of frames in sliding windows.

b. Local Features

- **Local Object Detection**: Focuses on the ball, players, and referee gestures using localized detection methods like **RoIAlign** and **Vision-Language Pre-training (GLIP)**. GLIP links detected entities with predefined labels, making the model aware of localized objects important to the action.

c. Feature Fusion

- Both global and local features are fused using **attention mechanisms** to highlight critical regions in the frame. The self-attention mechanism emphasizes the most relevant spatial-temporal regions during key moments in the match.

5. Long-Term Temporal Reasoning (LTR) Module

This module captures the evolution of player and ball movements across time:

- **Gated Recurrent Unit (GRU):** The GRU processes sequences of frames and aggregates the local-global feature fusion to predict actions over time. The GRU captures long-term dependencies, helping detect complex actions like dribbles or multi-pass combinations.
- **Temporal Action Proposals:** The GRU outputs potential action moments, which are then classified by a fully connected layer into specific ball actions. Redundant predictions are filtered using **Non-Maximum Suppression (NMS)**, ensuring clean action boundaries.

6. Training Methodology

The model is trained on extensive sports datasets like **SoccerNet-v2** with annotations for key actions. The training pipeline includes:

- **Loss Function:** A combination of **Focal Loss** for handling imbalanced classes and **Softmax** for classification. Focal Loss adjusts for underrepresented actions like fouls or cards.
- **Augmentation:** Video augmentations (e.g., flipping, cropping) expand the diversity of the training data.

The training leverages **transfer learning**, where the backbone is initialized with pre-trained weights from large datasets (e.g., ImageNet) before fine-tuning on sports-specific data.

7. Performance Metrics

Performance is evaluated using the **Tight Mean Average Precision (T-mAP)**, which measures both the accuracy of the action detection and how well it aligns temporally with ground truth labels. The **UGLF model** consistently outperforms other methods, particularly on smaller-scale actions like card detections or ball movements.

Architecture

1. High-Level Architecture Overview

At a high level, the architecture of this project consists of several key modules:

1. **Input Module:** Reads the video frames and prepares them for processing.
2. **Detection Module:** Detects objects and keypoints in each frame using neural networks (YOLO for object detection, HRNet for keypoint detection).
3. **Tracking Module:** Tracks the detected objects across multiple frames, ensuring the same objects are consistently identified over time using BoTSORT.
4. **Transformation Module:** Converts the detected objects' image coordinates into real-world coordinates on the football pitch using a homography matrix.
5. **Post-Processing and Calibration:** Refines the detection results, performs calibration on keypoints, and filters outliers.
6. **Output Module:** Outputs the transformed coordinates and any relevant metadata for further analysis or visualization.

2. Input Module

Video Frame Preprocessing

The system starts by ingesting video frames. These frames are typically in BGR format (as required by OpenCV), which means that the pixel values are stored as Blue, Green, and Red channels.

- **Resizing and Normalization:** Before feeding frames into the neural networks (YOLO and HRNet), they are resized and normalized. This ensures that the frames fit the input size expected by the models and that pixel values are scaled appropriately.
- **Conversion to Tensor:** Since the neural networks (implemented in PyTorch) require the input data to be in the form of tensors, the frames are converted into tensors, with pixel values normalized between [0,1].

The input module is responsible for:

1. **Reading frames** from the video.
2. **Resizing** and **normalizing** the frames to fit the input requirements of the YOLO and HRNet models.
3. **Converting frames** into tensors for deep learning inference.

3. Detection Module

Object Detection with YOLO

The YOLO model is a key part of the detection module. It performs real-time object detection, meaning it identifies objects within each frame and provides:

- **Bounding Boxes:** A rectangular area around each detected object.
- **Class Labels:** A label that indicates the type of object (e.g., player, ball, referee).
- **Confidence Scores:** The confidence level of the model in the detection, ranging from 0 to 1.

In the context of football, YOLO is used to detect:

- Players
- Goalkeepers
- Referees
- The Ball

The YOLO model used in this project is either an ONNX or a PyTorch model, depending on the hardware being used. The detection module will either load the appropriate YOLO model weights and run the object detection task on each frame. The model outputs an array of bounding boxes, class labels, and confidence scores, which are passed to the tracking module for further processing.

Keypoint Detection with HRNet

HRNet (High-Resolution Network) is used to detect keypoints on the football pitch. These keypoints are specific locations on the field, such as:

- Corners of the pitch.
- Intersections of lines.
- Center circle and penalty areas.

HRNet Architecture Overview

HRNet is unique because it maintains high-resolution representations throughout the network. Most convolutional neural networks downsample the image significantly, losing spatial precision. In contrast, HRNet:

1. **Maintains high-resolution feature maps** throughout its layers, which helps in tasks like keypoint detection that require high spatial precision.
2. **Fuses multi-scale information** by combining high-resolution and low-resolution feature maps at different stages in the network. This allows the network to capture both fine details and broader contextual information.

The HRNet architecture can be broken down into several components:

1. **Stem Network:** The first layers downsample the input slightly and extract basic feature maps.
2. **Multi-Scale Stages (Stage 2, 3, 4):** These stages process the image at multiple scales (resolutions) and pass information between different branches of the network. Each branch processes a different resolution of the input image, and the information is fused together at each stage.
3. **Final Output:** The final output is a set of keypoint heatmaps, where each heatmap corresponds to a specific keypoint on the field. The peaks in these heatmaps represent the detected keypoint locations.
4. The input video frame is passed through HRNet.
5. HRNet outputs heatmaps for each keypoint, where the brightest point on the heatmap represents the most likely location of that keypoint.

6. The detected keypoints are then used to calculate the homography matrix, which will map image coordinates to pitch coordinates.

4. Tracking Module (BoTSORT)

BoTSORT uses a combination of:

1. **Bounding Box Information:** The positions of detected objects from YOLO are fed into BoTSORT.
2. **Kalman Filter:** A Kalman filter is used to predict the next position of each object based on its current motion (velocity, acceleration, etc.). This helps smooth out the tracking and deal with minor detection errors.
3. **Hungarian Algorithm:** The Hungarian algorithm is used to match the detected objects in the current frame with the previously tracked objects. It ensures that the correct object IDs are maintained over time.
4. **Appearance-Based Re-Identification:** BoTSORT introduces appearance re-identification to further strengthen the tracking. It extracts visual features (e.g., color and texture) from the detected objects and uses this information to maintain consistent object identities, even when objects are occluded or appear very close to each other.
5. For each frame, YOLO provides the bounding boxes for detected objects.
6. BoTSORT takes these bounding boxes and matches them to previously tracked objects using Kalman filtering and the Hungarian algorithm.
7. If multiple players or objects overlap or are occluded, BoTSORT uses visual re-identification to maintain accurate tracking of the objects.
8. The result is a continuous and smooth tracking of players, referees, and the ball across all frames.

This allows the system to generate consistent object IDs for players, ensuring that, for example, "Player 1" remains "Player 1" throughout the entire video.

5. Transformation Module

Once the objects have been detected and tracked, the next step is to transform their image coordinates (i.e., their positions in the video frame) into real-world coordinates on the football pitch. This is done using a **homography transformation**.

Homography Transformation

To compute the homography matrix, we need at least 4 corresponding points between the video frame and the football pitch. These points are provided by the keypoints detected by HRNet. Once the homography matrix is calculated, it can be used to:

- Map the bounding box coordinates of the detected players, referees, and the ball into real-world coordinates on the football pitch.
- Ensure that the detected objects are accurately positioned on the pitch, regardless of camera angle or perspective.

Coordinate Transformation Process

1. **Keypoints Detected by HRNet:** Keypoints on the pitch are detected in the video frame.
2. **Correspondence with Real-World Points:** These keypoints are matched with known real-world coordinates (e.g., the corners and intersections of the field).
3. **Homography Matrix Calculation:** Using OpenCV's `findHomography` function, the homography matrix is calculated based on the correspondence between the keypoints and the real-world coordinates.
4. **Coordinate Transformation:** The homography matrix is applied to transform the image coordinates of detected objects (e.g., players, ball) into real-world pitch coordinates.

6. Post-Processing and Calibration

Once the keypoints and objects have been detected and transformed, the system performs some final post-processing steps to ensure accuracy.

Keypoint Calibration

The system filters out keypoints and objects that are detected as outliers. For instance:

- If a keypoint is detected in an area where there shouldn't be one (e.g., outside the pitch boundaries), it is discarded.

- If a player or ball's transformed coordinates fall outside the pitch boundaries after the homography transformation, they are also filtered out.

7. Output Module

Finally, the system outputs the transformed coordinates of players, the ball, and referees, along with additional metadata such as:

- **Frame Number**
- **Time Stamp** (based on the video's FPS)
- **Detected Keypoints**: Used for plotting the pitch and understanding the field layout.

Frame-by-Frame Processing Pipeline

Let's summarize how all of these components come together in the frame-by-frame processing pipeline:

1. **Frame Input**: For each frame in the video, the system first reads the image and prepares it for processing.
2. **Keypoint Detection**: HRNet detects keypoints on the pitch, which are used to calculate the homography matrix.
3. **Object Detection**: YOLO detects objects (players, referees, ball) and provides bounding boxes and class labels for each object.
4. **Optical Flow (if needed)**: If fewer than 4 keypoints are detected, optical flow is used to estimate the missing keypoints from previous frames.
5. **Homography Calculation**: Using the detected keypoints, the system calculates a homography matrix that transforms image coordinates into real-world pitch coordinates.
6. **Object Tracking**: BoTSORT tracks objects across frames, ensuring that the same player or ball is correctly identified in each frame.
7. **Coordinate Transformation**: The bounding box coordinates of detected objects are transformed into pitch coordinates using the homography matrix.
8. **Result Storage**: The system stores the transformed coordinates, along with the time and keypoint data, for each frame.

Ball Action: Pass and Drive

1. Model Architecture: Hybrid 2D-3D Convolutions

The core of the model relies on a hybrid approach combining **2D convolutional neural networks (CNNs)** for spatial feature extraction and **3D CNNs** for temporal feature extraction.

2D Convolutions (Spatial Feature Extraction)

- **Purpose**: The 2D convolutional layers are responsible for analyzing individual frames to extract spatial information about the soccer pitch, player positions, ball location, and other visual elements.
- **Functionality**: These layers capture information from each frame independently, focusing on identifying static or momentary features like player positions, ball placement, or static field elements (e.g., goalposts, sidelines). The model processes each frame of the video and builds a representation of the spatial context.
- **Pre-trained Models**: Often, the spatial model could be initialized using pre-trained models like **ResNet** or **EfficientNet** to leverage previously learned image features (e.g., corners, edges) and then fine-tuned for soccer-related data.

3D Convolutions (Temporal Feature Extraction)

- **Purpose**: The 3D convolutional layers capture the temporal dynamics, i.e., how the visual elements (players, ball) change over time across consecutive video frames. These layers are crucial for understanding motion and detecting when a ball action happens (like a pass or a shot).
- **Functionality**: 3D convolutions extend across both the spatial dimensions (width and height of the frame) and the temporal dimension (frames over time), thus allowing the model to detect patterns like player movements, ball trajectories, and action sequences (passes, shots). The 3D CNNs capture continuous player and ball movements that are integral to spotting actions like dribbles or passes.

2. Input Data Handling

The input to the model is a **sequence of grayscale video frames**. Grayscale frames reduce computational load and memory usage, focusing only on the intensity of pixels (which is sufficient for detecting spatial movements in football actions). This preprocessing step is crucial because the model needs to handle large sequences efficiently without compromising performance.

3. Long-Sequence Training

The model is trained on long video sequences, not just individual frames or short clips. This ensures that the model captures the full context of an action. For example, a pass might develop over several seconds, and the movement of players and the ball during that period is essential for correctly identifying the action. Training on long sequences enables the model to understand not just isolated moments but entire sequences that culminate in key ball actions.

4. Model Components and Layers

Spatial Branch (2D CNN):

- **Feature extraction backbone:** The 2D CNN extracts visual features from each frame independently.
- **Stacked Frames:** A certain number of consecutive grayscale frames are processed in parallel through this branch.

The result is a **feature map** for each frame, which captures key visual features like player positions and ball locations.

Temporal Branch (3D CNN):

- The extracted 2D feature maps are then passed through 3D convolutional layers, which process sequences of frames.
- **Motion Representation:** By convolving over multiple consecutive frames, the 3D CNN captures player and ball movement patterns. For instance, a pass is recognized based on how the ball moves across the field over time, and how players move relative to each other.

5. Output and Post-processing

The output of the model consists of temporal segments, with each segment predicted as one of the key football actions (e.g., pass, shot, cross). This raw output is refined through several post-processing steps:

- **Non-Maximum Suppression (NMS):** Applied to avoid duplicate detections for the same action.
- **Temporal Smoothing:** The model may produce jittery action detections in dense action sequences, so smoothing is used to maintain temporal consistency and reduce false positives by considering the surrounding frames' context.

6. Loss Function and Optimization

Loss Function:

The loss function combines **temporal localization loss** and **action classification loss**:

- **Temporal Localization:** This aspect of the loss penalizes incorrect predictions about when an action occurs. The model needs to correctly pinpoint the start and end times of actions like passes, shots, or fouls.
- **Classification Loss:** This penalizes incorrect predictions about the type of action. For example, predicting a "pass" when the action was a "shot" results in a penalty.

Optimization:

The optimization process involves using standard optimizers like **Adam** or **SGD**, with learning rate decay. The model balances between precise action spotting and fast detection (since the task requires real-time or near real-time action detection).

7. Transfer Learning

The model leverages **transfer learning** by starting with a pre-trained 2D convolutional model (such as **ResNet** or **EfficientNet**) to capture general spatial features before fine-tuning it on the football dataset. This helps reduce training time and improves performance by allowing

the model to benefit from previously learned image representations.

8. Post-Processing Techniques

After the raw predictions are made, post-processing ensures higher accuracy and better temporal precision:

- **Non-Maximum Suppression (NMS):** Helps eliminate overlapping or redundant predictions for the same action. This is crucial in dense action environments, where multiple actions could be detected within a short time frame.
- **Temporal Smoothing:** Applied to refine action spotting, ensuring the predictions align smoothly over time, avoiding jittery detections or false positives.

9. Evaluation and Performance Metrics

The model's performance is evaluated using:

- **Precision:** Measures how many of the detected actions are correct.
- **Recall:** Measures how many true actions were detected by the model.
- **Temporal Precision:** Evaluates how accurately the model detects when an action starts and ends.

Unifying Global and Local Scene Entities Modelling for Precise Action Spotting: Goal and Goal Try

The architecture of the **Unifying Global and Local Features (UGLF)** model is designed to tackle the problem of action spotting in sports by integrating **global scene understanding** and **local entity detection**, allowing precise detection of both large-scale environmental features and smaller, localized objects like the ball or players.

1. Global Feature Extraction

The model begins by capturing **global environmental features** using a **2D Convolutional Neural Network (CNN)**. This component extracts high-level spatial representations of the scene, such as field layout, player distributions, and game context, while maintaining efficiency.

- **Backbone Architecture:** The backbone uses a pre-trained CNN, like **RegNet-Y** (or an equivalent architecture), for its feature extraction. RegNet-Y was selected due to its computational efficiency and strong feature extraction capabilities in spatial tasks.
- **Temporal Feature Shifting (TFS):** To capture temporal continuity without resorting to heavy 3D CNNs, the architecture incorporates a **time-shift module (Gated Shift Network - GSM)**. This method moves feature maps forward and backward in time, emulating 3D convolutions with fewer computational costs. By shifting these feature maps across frames, the model captures temporal evolution in a computationally light manner.

2. Local Feature Extraction

While global features provide context, **local entities** (like players and the ball) are crucial for precise action recognition. This is where the **local feature extraction module** comes into play.

- **Vision-Language Model (GLIP):** The model leverages a **Vision-Language Pre-training (GLIP)** approach. GLIP extracts entities from the scene, such as players, the ball, and cards, using a pre-trained model that grounds these detections in predefined vocabularies (e.g., "ball," "player," "goal").
- **RoIAlign:** Regions of Interest (RoIs) are identified and processed using **RoIAlign** to extract fine-grained features of localized objects. This method ensures the model focuses on smaller, high-relevance areas like a player's actions or the ball's location, which are key to understanding actions such as passes, dribbles, or shots.

3. Global-Local Feature Fusion

The **feature fusion component** combines the global scene features with the local entity representations. This fusion is achieved through a **self-attention mechanism**, which dynamically weights the importance of various regions in the scene. This process ensures the model

focuses on both the general context and specific objects critical to the action at hand.

- **Attention Mechanism:** The self-attention layer balances the contributions of the global and local features. For instance, during a goal-scoring event, the global context might highlight the player's position, while the local context zooms in on the ball's trajectory.

4. Long-Term Temporal Reasoning

Once global and local features are combined, they are passed through a **Long-Term Temporal Reasoning (LTR) module** to model the temporal dependencies in the action sequence.

- **Gated Recurrent Unit (GRU):** The LTR module employs a single-layer **GRU**, which processes the frame-wise fused features (global-local) over time, allowing the model to detect when actions occur in a sequence of frames. GRUs are lightweight recurrent networks ideal for processing temporal data in sequential tasks.
- **Temporal Aggregation:** The GRU's output, which contains action predictions for each frame, is further processed to aggregate temporal information and detect precise moments when actions occur.

5. Action Proposal and Classification

The final component of the model is the **action proposal** and **classification** layer. This is where the model predicts both the **timing** and **type of action** occurring in the video.

- **Fully Connected Layer:** A fully connected layer classifies each temporal snippet into one of the predefined action categories (e.g., "pass," "shot," "no-action").
- **Non-Maximum Suppression (NMS):** To avoid redundant detections, **NMS** is applied. This step ensures that overlapping action proposals are merged, and only the most relevant proposals are retained.

6. Post-processing

- **Temporal Smoothing:** The predictions are refined using temporal smoothing to prevent jittery action spotting. This ensures that detected actions are temporally consistent and that false positives are minimized.

Flow Diagram

 Unknown macro: 'flowchart'

Implementation

This project focuses on detecting and tracking objects (players, referees, the ball) in football broadcast videos. It extracts both image coordinates and real-world coordinates (on a football pitch) for these objects using a combination of neural networks, computer vision techniques, and tracking algorithms. The primary objectives include:

- Detecting objects (e.g., players, ball) using a pre-trained YOLO model.
- Detecting pitch keypoints using HRNet.
- Tracking objects across frames with the BoTSORT algorithm.
- Converting image coordinates into real-world pitch coordinates using homography transformations.

Initial Setup and Device Configuration

The script first configures the environment to support various computation devices such as CPUs, CUDA-enabled GPUs, or Apple's MPS for macOS devices. This ensures that the system runs efficiently on the available hardware. Device detection happens at runtime, allowing the models to either run on a high-performance GPU or fall back to a CPU if necessary.

Model Initialization

The main models initialized are:

- **YOLO Object Detection Model:** Used to detect players, referees, and the ball. It provides bounding boxes around detected objects along with their class labels and confidence scores. YOLO is efficient in detecting objects in real-time, which is why it's used here for processing multiple frames in a video.
- **HRNet for Keypoint Detection:** This model detects specific keypoints on the football pitch, such as intersections of lines and other critical features on the field. These keypoints are necessary for calibrating the field and performing the homography transformation, which maps 2D image coordinates to 3D pitch coordinates.

Concepts:

Object Detection with YOLO

The YOLO (You Only Look Once) model is employed to detect objects in each frame of the video. The model provides bounding boxes for each detected object along with class labels (e.g., player, ball, referee) and confidence scores. The bounding box gives the position of the detected object in the 2D image space.

Once objects are detected, their information (bounding box coordinates, class label, and confidence score) is used for tracking and transformation into real-world coordinates.

Keypoint Detection with HRNet

HRNet (High-Resolution Network) is responsible for detecting keypoints on the football pitch, such as intersections and other specific areas. Keypoints are essential because they help align the image with the actual football field through **homography transformation**.

In each frame, the keypoints are detected by passing the image through the HRNet model. These keypoints are used to compute the homography matrix, which maps the 2D coordinates from the image into the 3D real-world space of the football field.

Homography Transformation

Homography is a mathematical transformation that allows you to map points from one plane (the broadcast video image) to another plane (the actual football pitch). In this context, the keypoints detected on the football field serve as control points for creating this mapping.

Every few frames (based on a configurable interval), the system recalculates the homography matrix using the detected keypoints. If fewer than four keypoints are detected in a frame, optical flow (a method to track motion between frames) is used to estimate the keypoints based on previous frames. This allows the system to maintain accurate field calibration even when not enough keypoints are detected.

Optical Flow for Keypoint Tracking

In cases where the keypoint detection model fails to detect sufficient keypoints, the system uses **optical flow** to track the movement of keypoints from one frame to the next. Optical flow estimates the movement of pixels between consecutive frames, helping to predict the position of undetected keypoints by comparing them with previously detected ones.

This is particularly useful when there are occlusions (e.g., a player blocks part of the field), or when the keypoint detection model does not perform well on a particular frame.

Tracking with BoTSORT

Once objects (e.g., players, ball) are detected in a frame, their positions need to be tracked over time across subsequent frames. For this, the system uses **BoTSORT** (a tracking algorithm). BoTSORT ensures that objects detected in one frame are correctly associated with the same objects in future frames, even if they move.

BoTSORT works by using both the bounding boxes from YOLO and the appearance of the objects to maintain their identity across multiple frames. This way, the system can consistently track the same player or ball throughout the video, ensuring that the coordinates are associated with the correct object.

Coordinate Transformation and Filtering

Once objects are detected and tracked, the bounding box coordinates of the objects (e.g., players or the ball) need to be transformed from image space (the pixel coordinates of the video frame) into real-world pitch coordinates. This transformation is performed using the homography matrix calculated from the detected keypoints.

The transformed coordinates are checked to ensure that they fall within the boundaries of the football field. If any transformed coordinate lies outside the expected pitch dimensions, it is discarded as an outlier.

Keypoint Calibration

To ensure that the detected keypoints are accurate, a calibration process is performed. This process checks the brightness around each detected keypoint using the HSV (hue, saturation, value) color space. If the detected keypoint's brightness is below a predefined threshold, a local search is conducted to find the brightest spot near the keypoint. This helps refine the accuracy of the detected keypoints and ensures the homography transformation is as accurate as possible.

For each video frame, the system follows these steps:

1. **Keypoint Detection:** Detect keypoints on the pitch. If the frame does not contain enough keypoints, use optical flow to estimate them.
2. **Object Detection:** Detect players, the ball, and referees using the YOLO model.
3. **Object Tracking:** Use BoTSORT to track the detected objects across multiple frames.
4. **Coordinate Transformation:** Convert the detected objects' 2D image coordinates into real-world pitch coordinates using the homography matrix.
5. **Result Storage:** For each frame, store the detected object coordinates, time, and keypoints for further analysis or visualization.

Ball Action: Pass and Drive

1. Video Preprocessing

Before feeding the video into the neural network, the frames undergo preprocessing steps to ensure efficiency:

- **Grayscale Conversion:** Video frames are converted to grayscale to reduce the complexity of the data and speed up processing. Since color information is not crucial for action detection, removing color data reduces computational requirements without losing important spatial and temporal details.
- **Resizing:** Frames are resized to fit the input requirements of the neural networks, allowing the model to process them efficiently.
- **Stacking Frames:** To capture temporal context, multiple consecutive frames are stacked together. This helps the model understand the motion and changes in the scene over time.

2. Model Architecture Implementation

The architecture, as discussed previously, uses a combination of **2D and 3D convolutions**.

2D CNN (Spatial Features)

- The **2D convolutional network** processes each frame independently to extract spatial features such as the position of players and the ball.
- **Pre-trained Backbones:** The 2D CNN often employs pre-trained models like **ResNet**, which have been fine-tuned on large datasets like **ImageNet**. The network learns key features such as edges, shapes, and regions of interest within each frame, representing spatial information about the soccer pitch and player locations.

3D CNN (Temporal Features)

- The **3D convolutional network** operates on the stacked sequence of frames to detect the temporal evolution of movements in the game, such as player runs, ball passes, and shots.

- The 3D CNN learns from sequences of consecutive frames and models the temporal relationships between objects, capturing how movements evolve across time. It's essential for identifying actions that are defined by motion, such as dribbling, passing, or kicking.

The model is implemented using **PyTorch**, where:

- **2D CNN layers** process spatial features, and the **3D CNN layers** handle temporal features.
- **Layer-wise Design:** The architecture includes convolutional, ReLU activation, and pooling layers, followed by fully connected layers to classify the action at each temporal location.

3. Action Spotting

Once the neural network extracts features from the video frames, the next step is to predict the type and timing of the action. The implementation focuses on recognizing **when** and **what** action occurs:

- **Temporal Detection:** The 3D CNN predicts when a certain action (pass, shot, etc.) occurs. This involves predicting the start and end time of each action.
- **Action Classification:** The network assigns a label to the detected action, such as "pass," "shot," "cross," etc.

This is achieved through the following:

- **Sliding Window Approach:** The model processes the video in small chunks (windows) to capture local actions, which helps in spotting actions continuously across a long video.
- **Output Predictions:** For each time window, the model outputs both the action type and its temporal position.

4. Training

The training process involves several key components:

- **Loss Functions:**
 - **Temporal Localization Loss:** Ensures that the model detects the start and end times of the actions correctly.
 - **Action Classification Loss:** Ensures that the correct action type is predicted.
 - These losses are typically combined into a single optimization objective that the model seeks to minimize during training.
- **Data Augmentation:** Various augmentation techniques, like random cropping, flipping, and rotation, are applied to the video frames to improve the model's robustness to different viewing angles and camera setups.
- **Long Sequences:** The model is trained on **long sequences of video frames** to improve its ability to capture longer-term temporal dependencies, which is crucial for detecting actions that evolve over time (like a long pass or a dribble).

5. Inference

During inference:

- **Sliding Window:** The model processes the video in chunks using a sliding window approach. It evaluates each segment for potential actions and makes predictions.
- **Post-Processing:** After the model makes its predictions, post-processing is applied:
 - **Non-Maximum Suppression (NMS):** This technique is used to eliminate duplicate detections of the same action in consecutive frames.
 - **Temporal Smoothing:** To avoid jittery predictions across time, smoothing algorithms are applied to the predicted action segments. This helps in reducing false positives and providing more accurate start and end times for actions.

6. Transfer Learning

The implementation uses **transfer learning** from pre-trained models, which allows the model to leverage knowledge from large-scale datasets (like ImageNet for spatial features) before fine-tuning on football-specific data. This significantly reduces the training time and improves the accuracy of action spotting.

7. Efficiency and Speed Optimization

The model is designed to run in real-time, or near real-time, which is crucial for live-action detection during a football match. Several techniques are applied for efficiency:

- **Grayscale Video Frames:** As discussed, reducing the input to grayscale frames decreases the amount of data that needs to be processed without losing essential information.
- **Batch Processing:** Video frames are processed in batches, improving computational efficiency by leveraging modern GPU capabilities.
- **Model Pruning:** Techniques like pruning may be used to reduce the model's size by eliminating unnecessary neurons or layers that do not contribute significantly to the final prediction.

8. Evaluation and Metrics

The model's performance is evaluated based on:

- **Precision and Recall:** Evaluating how accurately the model detects the actions (precision) and how many of the true actions it finds (recall).
- **Mean Average Precision (mAP):** Commonly used in action detection tasks to measure the accuracy of action classification and temporal localization.
- **Speed (Frames per Second - FPS):** Given the real-time requirement, the speed of the model is also a critical factor. The architecture is designed to optimize the trade-off between speed and accuracy.

Unifying Global and Local Scene Entities Modelling for Precise Action Spotting: Goal and Goal Try

1. Data Preprocessing

- **Video Frames:** Video data is loaded frame-by-frame. The frames are resized to fit the model's input requirements and converted into grayscale or color images, depending on the implementation.
- **Normalization:** Frames are normalized to ensure that pixel values are on a common scale, which aids in faster convergence during training.

2. Global Features Extraction (2D CNN)

- **Model Initialization:** A **2D CNN backbone** (e.g., ResNet or RegNet-Y) is initialized, either from scratch or by loading pre-trained weights (commonly from ImageNet).
- **Feature Extraction:** Each frame is passed through the CNN, which extracts spatial features from the entire frame. These features encode general information about the game, including the field layout, player positions, and overall context.
- **Temporal Feature Shifting:** The model uses a **temporal shift module** like **GSM (Gated Shift Network)**, which shifts feature maps across frames. This allows the model to emulate 3D convolution behavior (which captures temporal dependencies) without the computational cost. By shifting features in the forward and backward time directions, the model captures motion dynamics at a lower computational cost.

3. Local Features Extraction (Object Detection)

- **Object Detection Initialization:** A **Vision-Language Pretrained Model (GLIP)** is used to detect and localize specific entities like the ball, players, or other relevant objects.
- **RoIAlign:** For each detected object, **RoIAlign** is applied to ensure accurate localization. This method takes into account the exact object location and resizes it into a fixed feature map for further processing.
- **Feature Representation:** The detected entities are then encoded into local feature maps. These feature maps capture small but significant objects, such as the ball's position or a player's motion, which are crucial for identifying certain actions like passes or shots.

4. Feature Fusion with Attention Mechanism

- **Attention Mechanism:** Global and local features are combined using a **self-attention mechanism**. This mechanism calculates the importance of each region of the frame, allowing the model to focus on the most action-relevant areas.
- **Attention Calculation:** Self-attention works by computing weighted sums of the feature maps from both the global (scene-wide) and local (object-specific) features. This helps the model to dynamically emphasize important regions in the video frames, ensuring that small but significant actions (like a pass or card) are not overlooked.

5. Temporal Feature Aggregation (GRU)

- **GRU Initialization:** A **Gated Recurrent Unit (GRU)** is initialized to handle temporal dependencies. The GRU processes the fused feature maps over a sequence of frames.
- **Temporal Sequence Processing:** The GRU takes in a sequence of feature maps (both global and local) and processes them in temporal order. This allows the model to learn long-term dependencies in the video, such as continuous ball movement or player interactions over time.
- **Frame-wise Predictions:** For each frame (or small window of frames), the GRU outputs a prediction for whether an action occurs and, if so, what type of action it is (e.g., pass, shot).

6. Action Proposal and Classification

- **Fully Connected Layer:** The GRU outputs are passed through a **fully connected layer**, which classifies each frame into one of several possible action classes (or a 'no-action' class).
- **Softmax Activation:** A **softmax** activation function is applied to convert the raw scores from the fully connected layer into probabilities for each action class.

7. Non-Maximum Suppression (NMS) and Post-Processing

- **Action Proposals:** The model generates action proposals for each detected event in the video sequence.
- **Non-Maximum Suppression (NMS):** To avoid multiple detections of the same action (e.g., a pass being detected several times in a sequence), **NMS** is applied. NMS eliminates overlapping detections and retains only the most confident prediction for each event.
- **Temporal Smoothing:** **Smoothing** is applied to the temporal predictions to ensure that the detected actions are temporally consistent across frames. This minimizes false positives and prevents jittery predictions that might occur due to noise or rapid changes in the video.

8. Training

- **Dataset:** The model is trained on large, annotated sports datasets (e.g., SoccerNet-v2), where key moments (actions) are manually labeled. The training data includes video frames, with each frame associated with an action label (or 'no-action').
- **Focal Loss:** To address the class imbalance (since some actions, like goals or fouls, are rare), **Focal Loss** is employed. This loss function focuses more on rare events by penalizing incorrect classifications of minority classes more heavily.
- **Optimizer:** Standard optimizers such as **AdamW** are used, with a learning rate schedule that includes **cosine annealing** to improve convergence.
- **Data Augmentation:** Video data is augmented with random cropping, flipping, and brightness adjustments to increase the robustness of the model.

9. Inference

- **Sliding Window Approach:** During inference, video frames are processed in sliding windows. This ensures that the model can continuously detect actions without missing any.
- **Real-Time Processing:** The model is optimized for real-time performance, capable of processing 30-60 frames per second depending on the hardware. This allows the model to spot actions in live sports scenarios with minimal latency.

Next Steps

Step	Description
Step	Description
Step	Description

Institutions and Contributors

Institution	Contributor
Institution A	Person name 1
	Person name 2
Institution B	Person name 1
	Person name 2
Institution c	Person name 1
	Person name 2

Model Cards

Keypoints Detection Model Card

Model Name: Keypoints Detection Model (HRNet-based)

Overview

This model is designed to detect **keypoints** on a football pitch using the **High-Resolution Network (HRNet)** architecture. The keypoints correspond to critical field markers such as line intersections, goalposts, and penalty box corners. These markers are vital for accurate pitch mapping, homography transformations.

The model generates **57 heatmaps**, each representing a specific field marker, where the brightest pixel in each heatmap corresponds to the predicted location of the keypoint.

Model Details

- **Model Architecture:** HRNet (High-Resolution Network)
- **Training Framework:** PyTorch
- **Input Shape:** (N, 3, H, W), where H and W represent the height and width of the video frame.
- **Output:** Heatmaps of 57 keypoints (each heatmap represents a specific field marker, with the highest intensity representing the predicted keypoint).
- **Preprocessing:** The input images are resized and normalized before being fed into the model.

- **Model Weights:** `keypoints_main.pth` (trained weights based on the HRNet backbone).

HRNet Architecture

HRNet is a unique network architecture that maintains high-resolution representations throughout the network's depth, making it suitable for spatially precise tasks such as keypoint detection. The key characteristics of the HRNet used in this project include:

1. **Stem Network:** The initial layers extract features from the input image while preserving high spatial resolution.
2. **Multiple Resolution Stages:** The model maintains multiple branches processing images at different resolutions. Each stage fuses the information from high- and low-resolution feature maps.
3. **Final Layer:** The final output consists of 57 keypoint heatmaps, where each keypoint's location is the pixel with the highest activation on the heatmap.

Model Performance and Use

- **Targeted Keypoints:** The model detects specific locations such as the center circle, goalposts, penalty areas, corner points, and line intersections.
- **Primary Task:** The model is used to assist in field calibration and homography transformation, converting 2D broadcast video frames into real-world coordinates.
- **Precision:** High-resolution processing makes HRNet highly precise for detecting fine details in the field, such as intersections or pitch lines.

Training Data and Methodology

The model was trained using data from the **keypoint-detection** repository, which involves labeled datasets of football fields. The annotations for these datasets include:

- **Field markers** such as lines and corners.
- **Manual labeling** of keypoints corresponding to real-world positions on the pitch.

Training Process

- **Dataset:** The training dataset consists of broadcast video frames with manually annotated keypoints. The dataset provides labeled points corresponding to important pitch markers.
- **Augmentation:** The input images are augmented through resizing, color normalization, and flipping to increase the dataset size and improve model generalization.
- **Loss Function:** A combination of heatmap-based loss functions is used to ensure accurate keypoint prediction. The loss function encourages the model to output heatmaps where the correct keypoints are highlighted with high intensity.
- **Optimizer:** Standard optimization methods such as Adam or SGD with learning rate decay are applied to minimize the loss during training.
- **Pre-trained Weights:** The HRNet is typically initialized with pre-trained weights from large-scale image classification tasks (e.g., ImageNet) to leverage general image features before fine-tuning on the football-specific task.

Inference Details

- **Input:** Video frames are passed through the model, which outputs heatmaps for each keypoint. The pixel with the highest intensity on each heatmap is considered the predicted location of the keypoint.
- **Post-Processing:** The heatmaps are converted into pixel coordinates corresponding to the keypoints. The model's output is then fed into the homography transformation pipeline to map these points to real-world pitch coordinates.

Performance Metrics

The model was evaluated on several metrics:

- **Keypoint Localization Error:** Measures the pixel-level accuracy of the predicted keypoints compared to the ground truth.

- **Robustness:** The model performs well in most broadcast scenarios but may struggle in frames where keypoints are occluded or where field markings are unclear due to low resolution or lighting conditions.

Limitations

- **Occlusions:** The model may miss keypoints in frames where the field markers are occluded by players or other objects.
- **Camera Angles:** Extreme camera angles may affect the accuracy of the homography transformation and keypoint detection.
- **Lighting Conditions:** Poor lighting or low video quality can degrade the performance of the model.

Fine-tuning and Improvements

To further improve the model's performance, it can be fine-tuned using datasets specific to particular broadcasting environments, stadiums, or lighting conditions. Higher-resolution training data or domain-specific augmentations can also enhance the model's ability to detect keypoints in challenging scenarios.

Citation and References

The model and training process are based on the [Keypoint Detection repository](#), which provided the architecture and methodology for detecting pitch-specific keypoints.

Players and Ball Detection Model Card

Model Name: Players and Ball Detection Model (YOLO-based)

Overview

This model is designed to detect **players**, **goalkeepers**, **referees**, and the **ball** in broadcast football footage using a **YOLO-based architecture**. The model is part of the **Eagle Project** and specifically trained to locate these key entities in a football match. The model outputs bounding boxes for each detected object, along with confidence scores and class labels.

This model enables real-time tracking of players and the ball, which is essential for creating accurate game analyses, player statistics, and further data-driven insights into player movements and ball possession.

Model Details

- **Model Architecture:** YOLO (You Only Look Once)
- **Training Framework:** PyTorch (ONNX for inference)
- **Input Shape:** (N, 3, H, W), where H and W are the height and width of the video frame.
- **Output:** Bounding boxes, class labels, and confidence scores for each detected object.
- **Classes Detected:** Players, Goalkeepers, Referees, Ball
- **Model Weights:** `detector_medium.onnx` (ONNX model format for CPU-based inference).

YOLO Architecture

The model follows a YOLO-based architecture, which is known for its efficiency in object detection tasks. YOLO detects multiple objects in a single pass through the network, making it ideal for real-time applications like tracking players and the ball during live football broadcasts. YOLO models divide the image into a grid, and within each grid cell, they predict bounding boxes, object confidence scores, and class probabilities.

The **detector_medium** model is a smaller version of the YOLO network, optimized for speed while maintaining reasonable accuracy for detecting players and balls in a football field.

Model Performance and Use

- **Targeted Classes:** The model is trained to detect:
 - Players (field players)
 - Goalkeepers
 - Referees
 - The ball
- **Primary Task:** The model assists in real-time tracking of players, ball positions, and referees during a match for statistical analysis and strategy planning.
- **Precision:** While the model sacrifices some accuracy for speed, it maintains reasonable precision for real-time applications.

Training Data and Methodology

The model was trained using labeled datasets that included a diverse set of football match scenarios, covering different stadiums, lighting conditions, and camera angles. The dataset includes annotated bounding boxes for players, goalkeepers, referees, and the ball.

Training Process

- **Dataset:** The training data consists of football broadcast footage with labeled bounding boxes for players, goalkeepers, referees, and the ball.
- **Data Augmentation:** The input images are augmented through various transformations such as scaling, cropping, flipping, and adjusting brightness to improve the generalization of the model.
- **Loss Function:** A combination of localization loss, confidence loss, and classification loss is used to optimize the model. The localization loss ensures accurate bounding boxes, confidence loss adjusts for prediction certainty, and classification loss minimizes misclassification of objects.
- **Optimizer:** Adam or SGD with learning rate scheduling is applied to train the model, with a focus on balancing between detection accuracy and computational speed.
- **Pre-trained Weights:** The model uses pre-trained weights from general object detection tasks (e.g., COCO dataset) before fine-tuning it for football-specific entities.

Inference Details

- **Input:** The input is a single video frame in BGR format.
- **Output:** The model outputs bounding boxes with confidence scores and class labels (players, goalkeepers, referees, or ball).
- **Post-Processing:** The bounding boxes are post-processed to remove duplicate detections (Non-Maximum Suppression) and assign the final object class based on the highest confidence score.

Performance Metrics

The model was evaluated on several key metrics to measure its performance in real-time football scenarios:

- **Detection Accuracy:** Measures the precision and recall for detecting objects (players, ball, referees) within the video frames.
- **Bounding Box Accuracy:** Assesses the overlap between the predicted bounding boxes and the ground truth (measured by IoU - Intersection over Union).
- **Speed (FPS):** The model is optimized to run in real-time (30-60 frames per second) on modern CPUs, ensuring efficient tracking in live broadcasts.

Limitations

- **Occlusion:** The model may miss detecting players or the ball when they are occluded by other players or objects.
- **Small Objects:** Detecting the ball in long-range camera shots (where the ball appears small) can be challenging for the model.
- **Complex Backgrounds:** In certain situations, the model may struggle with false positives due to complex backgrounds or crowd scenes.

Fine-tuning and Improvements

To improve performance, the model can be further fine-tuned on:

1. **High-Resolution Data:** Using higher-resolution images may improve the detection of small objects like the ball.
2. **Domain-Specific Data:** Fine-tuning the model on a dataset containing challenging camera angles, specific stadium environments, and different lighting conditions can enhance robustness.
3. **Model Pruning and Optimization:** Further pruning and quantization techniques can be applied to reduce model size and improve inference speed without significantly compromising detection accuracy.

Citation and References

This model is part of the **Eagle Project**, and it uses the methodology and architecture inspired by the YOLO family of object detection algorithms.

Ball Action Spotting Model Card

Model Name: SoccerNet Ball Action Spotting Model

Overview

This model is developed for detecting and classifying **ball-related actions** in football videos. It focuses on identifying key moments such as **passes, shots, and drives** in game footage. The model utilizes a hybrid architecture combining **2D CNNs** for spatial feature extraction and **3D CNNs** for capturing the temporal dynamics of the video. This combination allows the model to effectively process video frames in sequence and detect significant ball actions over time. The model was developed for the **SoccerNet Ball Action Spotting Challenge 2023**, where it achieved state-of-the-art results.

Model Details

- **Model Architecture:** Hybrid 2D-3D Convolutional Neural Network
- **Framework:** PyTorch
- **Input Shape:** Stacked grayscale video frames of shape (N, 3, H, W), where H and W represent the height and width of each frame.
- **Output:** Temporal segments with ball-related action labels (e.g., pass, shot, drive) and corresponding time frames.
- **Model Weights:** Pre-trained weights are based on a combination of general-purpose video models fine-tuned for the specific task of football action spotting.

Model Architecture

2D CNN (Spatial Feature Extraction)

- The 2D CNN processes individual frames to extract spatial features such as player positions and ball location.
- Pre-trained backbones like **ResNet** are used for the spatial feature extraction, and the model is fine-tuned on football-specific video data.

3D CNN (Temporal Feature Extraction)

- The 3D CNN layers process sequences of frames, analyzing temporal changes across them. This enables the model to detect actions by recognizing movements in the video over time.
- This part of the model is essential for understanding how the ball moves and how players interact over time, allowing for action recognition like passes, shots, or crosses.

Training and Transfer Learning

- The model is trained using transfer learning, leveraging **pre-trained models** for both 2D and 3D CNNs. This allows the model to use general-purpose video feature extraction before fine-tuning on the football-specific dataset.
- The training dataset consists of labeled football videos with annotated action spots (e.g., passes, shots), providing the model with ground truth temporal and spatial data for action spotting.
- **Long-sequence training** is employed, allowing the model to learn not just short-term events but also the broader context of actions within the game.

Inference and Post-processing

- **Sliding Window Approach:** The model processes video frames in small sliding windows, detecting actions as they unfold in real-time.
- **Post-Processing:** Includes **Non-Maximum Suppression (NMS)** to remove duplicate action detections and **temporal smoothing** to ensure that action predictions are temporally consistent and not jittery.

Performance

The model achieves top performance in detecting ball actions in dense football footage. Evaluation metrics include:

- **Precision and Recall:** For action detection, focusing on minimizing false positives and maximizing the detection of true actions.
- **Mean Average Precision (mAP):** Used to evaluate the overall performance in classifying and localizing ball actions.
- **Inference Speed:** Optimized for real-time performance, capable of processing up to 60 frames per second depending on hardware specifications.

Limitations

- **Occlusions:** The model may miss actions where the ball or players are heavily occluded by other players or objects.
- **Small Ball Detection:** Detecting the ball in distant or long-range shots can be challenging due to the small size of the ball in the video frame.
- **Complex Scenarios:** Some complex player formations or dense scenes may lead to false positives in action detection.

Fine-tuning and Improvements

To improve performance, the model can be fine-tuned further using datasets specific to particular leagues, camera angles, or stadium conditions. Additionally, the following improvements can be made:

1. **Higher-resolution training data:** Enhancing the model's ability to detect smaller objects such as the ball in wide shots.
2. **Domain-specific augmentations:** Further data augmentation specific to football, such as weather conditions or different lighting, could improve robustness.

Citation and References

This model is part of the SoccerNet Ball Action Spotting Challenge 2023.

UGLF Model Card

Model Name: Unifying Global and Local Features (UGLF) Model

Overview

The UGLF model is designed for **action spotting in sports videos**, particularly in football, by leveraging both **global scene features** and **local object detection**. The model uses a combination of **2D CNNs** for spatial understanding and **Vision-Language Pre-training (GLIP)** to detect specific entities like the ball, players, and referees. The architecture is optimized for detecting complex and fast-paced actions like passes, shots, and fouls, improving the overall accuracy of sports video analytics.

Model Details

- **Architecture:** Combination of 2D CNNs for global feature extraction and Vision-Language models for local object detection.
- **Framework:** PyTorch
- **Input Shape:** Stacked video frames (N, 3, H, W), where N represents the sequence of frames, and H and W are the height and width of the frame.
- **Output:** Temporal segments of actions with class labels (e.g., pass, shot, no-action) and precise start and end times.
- **Pretrained Models:** The backbone utilizes pre-trained models like RegNet-Y and GLIP for local object detection, fine-tuned for the task of sports action spotting.

Model Architecture

Global Features Extraction (2D CNN)

- A **2D CNN** processes each frame to extract global features that capture the overall scene context, such as player positions and field layout.
- **Temporal Shift Module:** The global features are passed through a **Gated Shift Network (GSM)**, which shifts feature maps across time to capture temporal dependencies without the need for heavy 3D convolutions.

Local Features Extraction (GLIP)

- The **Vision-Language Pre-trained (GLIP)** model is used to detect and localize entities like the ball, players, and other small objects within each frame.
- **RoIAlign:** For each detected entity, **RoIAlign** extracts localized feature maps, which are then combined with global scene features.

Global-Local Feature Fusion

- The global and local features are fused using a **self-attention mechanism**, ensuring that the model can focus on the most relevant regions of each frame.
- This attention mechanism balances between large-scale global information and small, action-relevant details such as the ball or player movements.

Temporal Feature Aggregation (GRU)

- The fused global-local features are passed through a **Gated Recurrent Unit (GRU)** to model temporal dependencies and detect actions over sequences of frames.
- The GRU outputs frame-wise predictions of actions, determining both the type of action and the time it occurs in the video.

Action Proposal and Post-processing

- **Fully Connected Layer:** The GRU's output is classified into action labels via a fully connected layer, followed by a softmax activation.
- **Non-Maximum Suppression (NMS):** Post-processing includes NMS to remove redundant action proposals and retain only the highest-confidence predictions.
- **Temporal Smoothing:** A smoothing technique is applied to ensure that action predictions are temporally consistent and avoid jittery detection.

Training and Loss Function

- **Focal Loss:** Used to address class imbalance, especially for rare actions like fouls and goals, by weighting the loss more heavily for misclassifications of minority classes.
- **Optimizer:** The model is optimized using **AdamW**, with a learning rate schedule involving **cosine annealing**.
- **Dataset:** The model is trained on large-scale sports datasets like SoccerNet-v2, which includes labeled video data with annotated actions.

Inference

- **Sliding Window Approach:** During inference, video frames are processed in sliding windows to ensure continuous action detection.
- **Real-time Capability:** The model is optimized to run in real-time, processing up to 60 frames per second.

Performance Metrics

- **Precision and Recall:** Used to evaluate the accuracy of action detection, especially for smaller objects like the ball or rare actions like fouls.
- **Mean Average Precision (mAP):** The main evaluation metric, measuring both the accuracy of action classification and the temporal precision of action spotting.
- **Inference Speed:** The model is optimized for real-time performance to handle live or near-live sports event processing.

Limitations

- **Occlusions:** The model may struggle with detecting actions where key entities (e.g., the ball) are occluded by players or other objects.
- **Class Imbalance:** Despite the use of Focal Loss, actions that occur infrequently (like fouls or cards) may still be underrepresented in predictions.
- **Complex Backgrounds:** In some scenarios, such as crowded stadiums or complex scenes, the model may produce false positives.

Fine-tuning and Improvements

- **Higher Resolution Data:** Using higher resolution training data can help improve detection of smaller objects, especially the ball in long-range shots.
- **Custom Augmentations:** Domain-specific augmentations tailored to sports video data (e.g., lighting conditions or crowd noise) can further enhance the model's robustness.
- **Future Directions:** Future improvements could involve integrating **Graph Neural Networks (GNNs)** to model interactions between players and objects more effectively.

Citation and References







If using this model, please cite the UGLF repository.


Project Data

Use Cases/ Ideas

Case/Idea	Description
Case/Idea 1	Description 1
Case/Idea 2	Description 2

References

Case/Idea	Description
Ref 1	 Hawk Eye · Automatic Birds Eye View Registration of Sports Videos
Ref 2	 GitHub - SoccerNet/SoccerNet-v3
Ref 3	 GitHub - eddwebster/football_analytics: 🏈🌐 A collection of football analytics projects, data, and analysis by Edd Webster (@eddwebster), including a curated list of publicly available resources published by the football analytics community.
Ref 4	 footballanalysis/Bird's eye view at main · FootballAnalysis/footballanalysis
Ref 5	 GitHub - TrackingLaboratory/tracklab: A modular end-to-end tracking framework for research and development
Ref 6	 GitHub - SoccerNet/sn-gamestate: [CVPRW'24] SoccerNet Game State Reconstruction: End-to-End Athlete Tracking and Identification on a Minimap (CVPR24 - CVSports workshop)

Ref 7	https://arxiv.org/pdf/2404.11335v1
Ref 8	 GitHub - nreHieW/Eagle: Convert broadcast data to tracking data using Computer Vision