UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

u20503289

u20424575

u18103007

u21488275

COS 791 : Object Tracking

Hockey ball tracking for the Visually Impaired.

Introduction:

Objective: The goal of this project is to design an algorithm to track a field hockey ball in video footage, enhancing its visibility by enlarging and altering its colour. The algorithm must address challenges such as lighting variations, changes in ball colour, and occlusions.

Data Augmentation:

The initial data set of images was sourced from two roboflow datasets, one data set was a sample of about 2539 images from ice hockey [3] and the other data set was 2900 images from field hockey [4]. These images were then combined and randomly picked and renamed so that the data has no underlying pattern. This data set is quite small and as a result the training dataset was augmented to prepare the model for conditions that it might encounter, such as different lighting, colours, camera angles, and slight blurring or noise. Below is a breakdown of how augmentation was performed for each element in the pipeline:

Augmentation Pipeline

1. **Brightness and Contrast Adjustments**: Random changes in brightness and contrast (up to 20%) simulate different lighting conditions, making the model more resilient to varied environments.
2. **Hue, Saturation, and Value (HSV) Modifications**: Adjustments in hue, saturation, and value mimic changes in colour and lighting by shifting these properties within specified limits, increasing the model's adaptability to diverse colour settings.
3. **Perspective Transformations**: Slight perspective shifts replicate the effect of viewing the ball from different angles. This is useful for detecting the ball from varying viewpoints.
4. **Rotation, Scaling, and Shifting**: Random rotations (up to 45 degrees), scaling, and translations create spatial diversity in the dataset. This aids the model in recognizing the ball regardless of its orientation and positioning in the frame.
5. **Colour Variations (RGB Shifts and Channel Shuffle)**: By randomly shifting the RGB channels and shuffling them, this step introduces variations in ball colour, helping the model to generalise better across different ball colours.
6. **Motion and Gaussian Blur**: The addition of motion and Gaussian blur simulates motion effects. This helps the model to recognize the ball even when it appears blurry in action shots.
7. **ISO Noise**: Adding camera noise, especially for low-light conditions, prepares the model for scenarios where the ball might be less visible or where image quality is compromised.

These augmentation steps are applied selectively, based on a defined probability. After augmentation the training set increased to 15159 images with their associated labels. The final data partition is 15159 images in the training set, 414 images in the testing set, and 679 images in the validation set.

Bounding Box Adjustment

Each augmentation operation is applied with bounding box adjustments, so that the bounding box coordinates remain accurate after transformations. The bounding boxes for the augmented images are dropped if the ball/bounding box is less than 25% visible in the new image.

Implementation

The training images and their associated bounding box coordinates are loaded, augmented, and then the augmented images along with the associated labels are added to the training set only.

Ball Detection:

Object detection is the task of identifying and locating objects within an image by classifying relevant regions and marking their positions, typically with bounding boxes. It is the combination of classification (determining object presence) and localization (estimating object location) to detect object(s) accurately.

Algorithm Design

There is a script called ball_detection in the src folder it contains the code for training and testing the model so that it performs predictions on the provided videos. The ball detection in this project is achieved using the YOLOv11 model. The script has two main functions namely, train_yolo and test_yolo.

Training Function (train_yolo): This function uses a pretrained model (such as yolo11n.pt) or any YOLOv11 model specified by the user. Key arguments include:

- epochs: Defines the number of times the model should process the entire dataset.

- patience: Prevents overfitting by specifying the number of epochs to continue training without improvement in metrics before stopping.

- imgsz (image size): Ensures that all images are resized to a uniform size before being processed through the model's convolutional layers.

- Momentum: Helps speed up training by using past updates to guide the current step, making learning smoother and faster.

- Learning Rate: Controls how big each training step is; too high can overshoot, too low can make training slow.

When this function is done it generates the final and best model in modelsAndLogs folder along with arguments used and the metrics per epoch.

Testing Function (test_yolo): This function tests the YOLO model on images or videos to detect objects (like the hockey ball). Key arguments include:

- model: The name of the trained model to be used for testing.

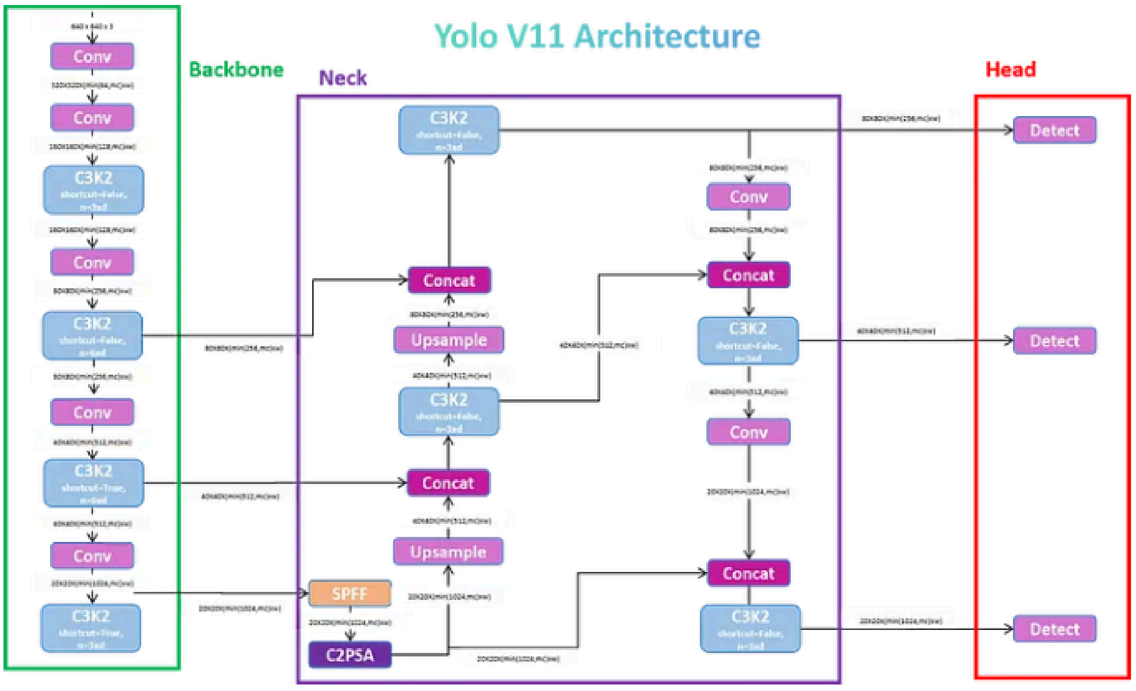
- conf: Confidence threshold, which controls how sure the model needs to be to confirm a detection.

- iou: Intersection over Union threshold, which determines how close detected objects must be to be considered the same.

- max_det: Sets the maximum number of detections per frame.

When this function is done it generates images with the ball detections or videos with the ball detections in the data/predictions/pictures or data/predictions/videos paths respectively.

YOLO Model Architecture:



YOLOv11 Model Architecture

Backbone (Left, Green Box)

- The backbone consists of a series of convolutional layers (Conv) and specialized blocks (C3K2) for feature extraction.
- Layers include:
  - Convolutional blocks for initial feature extraction.
  - C3K2 blocks, which handle more refined feature extraction at multiple levels of the backbone.

    - These blocks process the input image and generate feature maps by gradually reducing spatial dimensions while increasing depth.

Neck (Center, Purple Box)

- The neck aggregates and refines the features extracted by the backbone.
- Key components:
    - C3K2 blocks and Concat operations, which merge information from different levels to capture multi-scale context.
    - Upsample layers, which increase the resolution of feature maps to allow detailed object detection.
    - SPFF (Spatial Pyramid Pooling Fast), used to pool multi-scale features for robust detection, especially for small objects.
    - C2PSA (Cross Stage Partial with Spatial Attention), an attention mechanism that helps the model focus on spatially relevant regions.

Head (Right, Red Box)

- The head is responsible for generating final detection outputs.
- Consists of multiple Detect layers, which make predictions at different scales (small, medium, and large objects) using the feature maps generated by the neck.
- These detection layers output bounding boxes and confidence scores for each object across multiple scales, enabling the model to detect objects of various sizes.

Hyper Parameter Optimization

Hyperparameter optimization was achieved using the Optuna Library to get the optimal performance out of the YOLO Model. The goal is to achieve a higher mean Average Precision at 50% Intersection over Union(mAP50). The Optuna library allowed for the use of the “suggest” method, which allows for the specifying of ranges and type of values to explore for each hyperparameter. Optuna will automatically sample values from the defined search space for each trial and provide the best combination of hyperparameters for the best trial(model), in our case, a model that has a higher mAP50. The parameters' search space was as follows:

- epochs: 50 ~ 300
- patience: 10 ~ 20
- image size: 1920 ~ 2048 (step size is 160)
- batch: 0.8 ~ 0.95
- momentum: 0.85 ~ 0.95
- learning rate: 0.001 ~ 0.1 (logarithmic scale was also used for exploration over several orders of magnitude)

This code is on a separate branch in the repo, under “edwin/Tuning+Training”

Results:

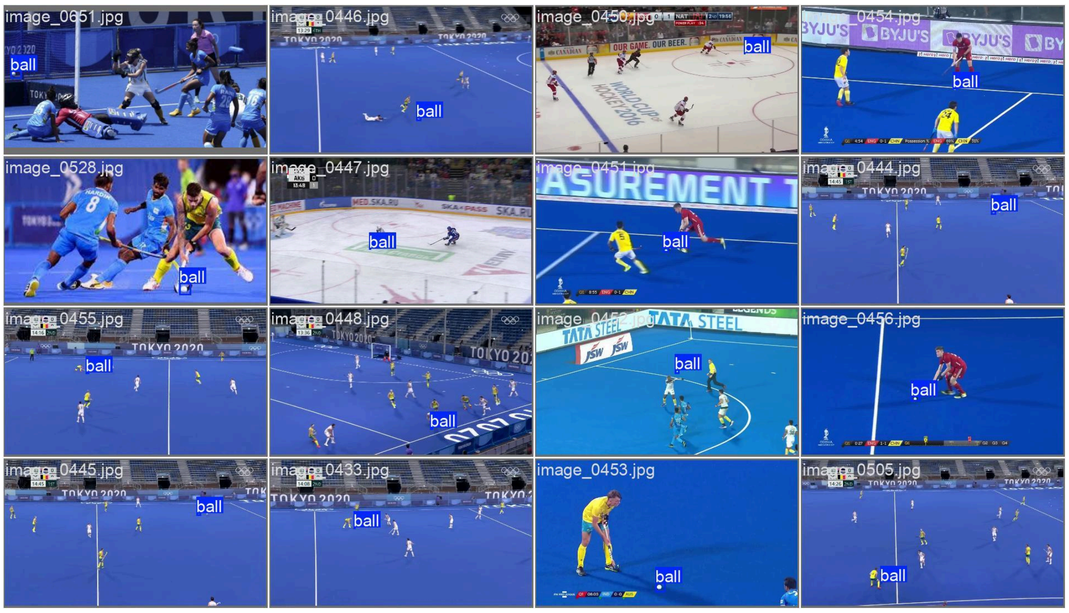
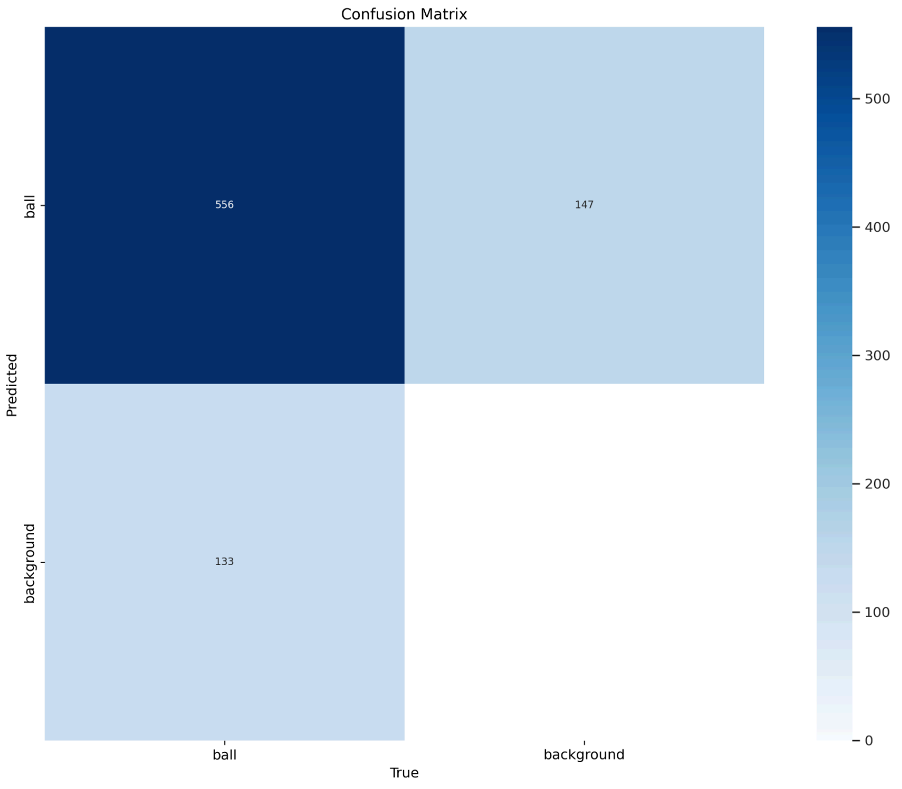
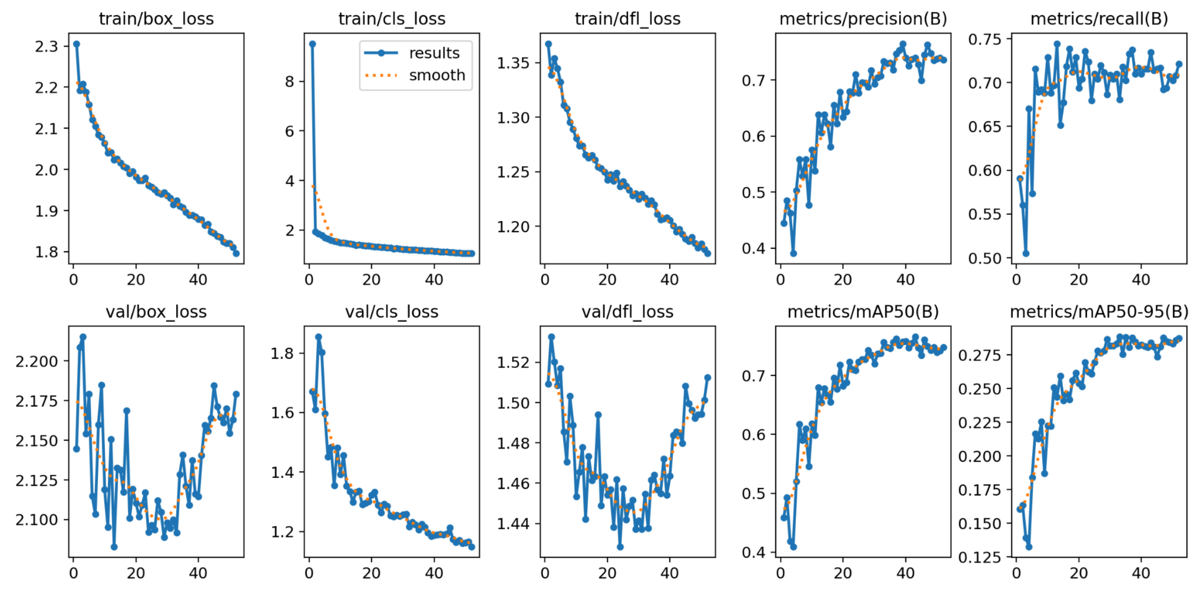


image_0651.jpg
ball
image_0446.jpg
ball
image_0450.jpg
ball
0454.jpg
ball
image_0528.jpg
ball
image_0447.jpg
ball
image_0451.jpg
ball
image_0444.jpg
ball
image_0455.jpg
ball
image_0448.jpg
ball
image_0452.jpg
ball
image_0456.jpg
ball
image_0445.jpg
ball
image_0433.jpg
ball
image_0453.jpg
ball
image_0505.jpg
ball



Confusion Matrix
Predicted
True
ball
background
556
147
133
500
400
300
200
100
0



These results were gathered from the best model that was trained through Optuna's Hyperparameter Optimization.
The first image, showing a sample of the predictions for the positions of the ball in each image of the batch.
The second image is a Confusion Matrix for the Predictions vs. Actual for the ball and the background.
The third image shows all the metrics of the model's epochs in the training phase.

From the results of the model, it demonstrates that there is a clear trend of improvement across Precision, Recall, and mAP metrics throughout the epochs. The loss values also decrease as the training progresses, indicating that it is getting more accurate with the detections. The best model was able to achieve an mAP50 of 76%.

Ball Tracking and Modification:

Algorithm Design:

There is a script called ball_tracking in the src folder it contains the code for tracking the ball, enlarging it and changing its colour. The script has a class which uses the Kalman filter for ball tracking and there is a function called test_yolo_with_tracking which initialises the class and performs enlargement and colour change based on the tracking.

BallTracker Class

The BallTracker class is designed to track a moving ball across video frames. It uses a Kalman filter to handle instances where the ball may be temporarily lost or occluded, predicting its position until it reappears.

1. __init__ Method

- Initialises the Kalman filter with a 4-state vector: [x, y, dx, dy], where (x, y) is the position and (dx, dy) is the velocity.
- Sets up matrices:
    - Measurement Matrix: Maps the predicted state to the observed state.
    - Transition Matrix: Updates the state with each frame.
    - Process Noise Covariance: Helps the filter adjust for small errors in prediction.
- Other attributes:
    - max_disappeared: Limits the number of frames the ball can be "lost" before predictions stop.
    - positions: Stores recent ball positions.
    - disappeared: Counts frames with no detected ball.
    - initialised: Flags whether the filter is ready.

2. initialize_filter Method

- Initialises the Kalman filter with the ball's starting position.
- Sets the initial state vector to [x, y, 0, 0], where 0, 0 represents initial velocities.

3. update Method

- Updates the ball's position each frame.
    - If a position is detected: Updates the Kalman filter with the detected position, resets disappeared count, and appends the position to positions.
    - If no position is detected: Increments disappeared and makes a prediction if disappeared exceeds max_disappeared.
- Return Value: Predicts the next position if the ball disappears in subsequent frames.

test_yolo_with_tracking Function

The test_yolo_with_tracking function detects, tracks, and enhances the ball in each video frame.

Key Steps:

1. YOLO Model Initialization: Loads a trained YOLO model to detect the ball in each frame.
2. Video Setup: Reads the video frames and sets up an output file to save processed frames.
3. Frame Processing:
    - Ball Detection: Runs YOLO on each frame to detect the ball, returning bounding box coordinates if found.
    - Ball Enhancement:

        - Adjusts the ball’s hue (colour) for visibility.
        - Enlarges the ball region and overlays it on the frame.
    - Tracking: Updates the BallTracker with the ball’s position:
        - If detected, the position is added to the tracker.
        - If undetected, the Kalman filter predicts the position.
    - Visualising the Track: Draws a small circle at the predicted or detected position for reference.
4. Output: Writes each processed frame to the output video file.

Detection using YOLO:

The YOLO model that was developed from the ball detection script’s training function detects the ball in each video frame based on specified confidence and intersection-over-union (IoU) thresholds, allowing for real-time localization of the ball's position. For each detected instance, the YOLO model provides bounding box coordinates, which are processed to determine the ball’s centre.

Kalman Filter:

The Kalman filter [5] in the BallTracker class predicts the ball’s position by combining a linear motion model with Bayesian estimation. It tracks the ball's state, including its position (x,y)(x, y)(x,y) and velocity (dx,dy)(dx, dy)(dx,dy), enabling continuous tracking even when detections are temporarily missed.

During the prediction step, the filter uses a constant velocity model with a transition matrix that updates the ball’s position by adding its current velocity, projecting the next state as follows:

When a new measurement (e.g., from YOLO) is available, the Kalman filter corrects its prediction using a measurement matrix, which represents that only the position (not the velocity) is directly observed in each measurement. The filter then combines the predicted state with the new measurement, weighted by the Kalman gain. The Kalman gain reflects the uncertainty in both the prediction and the measurement, yielding an updated estimate of the ball’s position and velocity for improved tracking accuracy.

If no measurement is available (i.e., the ball is not detected), the filter relies on its predicted position, assuming the ball will continue in the same direction and speed. This assumption maintains tracking continuity even during brief detection lapses. However, if detections are absent for an extended period, the filter's uncertainty grows, yet it continues to provide an estimated location for the ball.

Enlarging and changing the colour of the ball:

The ball enlargement process involves using the YOLO’s detected bounding box in each frame, calculating its centre coordinates, and scaling the box dimensions to enlarge the ball region by a specified factor. First, the bounding box coordinates are extracted, and the centre of the ball is calculated. The bounding box dimensions are then multiplied by a scaling factor to determine the enlarged size. The detected ball region is extracted, a hue shift applied to the region in HSV colour space. The hue shift allows for adjusting the ball's colour while preserving the original intensity and saturation values, which makes the ball stand out in the frame. Additionally, the saturation and value channels are boosted within the ball's area to ensure that the hue shift is visually effective, even on darker or less saturated colours.. This adjusted region is resized to the new enlarged dimensions and then overlaid onto the original frame, centred around the ball’s detected position. The result is an enlarged, colour-adjusted representation of the ball, enhancing its visibility in the video. Please click this link to find the final video with ball detection, tracking, enlargement and colour changes.

Running the code:

To run the project code please go to github and follow the setup instructions there in the readme. Please click this link to locate the github repository of the project.

To watch the video demonstration of the algorithm being applied to the provided videos please click this link.

References:

1. Ultralytics. (2024, November 2). *YOLO11 🚀 NEW*. Ultralytics YOLO Docs.

    https://docs.ultralytics.com/models/yolo11/#performance-metrics

2. Rao, S. N. (2024, October 22). YOLOv11 Architecture Explained: Next-Level Object Detection with Enhanced

    Speed and Accuracy. *Medium*.

    https://medium.com/@nikhil-rao-20/yolov11-explained-next-level-object-detection-with-enhanced-speed-and-accuracy-2dbe2d376f71

3. *Детекция шайбы Object Detection Dataset (v3, 2024-09-02 4:05pm) by fastdeploy*. (n.d.). Roboflow.

https://universe.roboflow.com/fastdeploy/-w2x0g/dataset/3

4. *shp_hockey_ball Object Detection Dataset (v6, 2023-07-16 9:50pm) by SuhasPatel*. (n.d.). Roboflow.

https://universe.roboflow.com/suhaspatel/shp_hockey_ball-twdio/dataset/6

5. *Getting up to speed with Kalman filters · VectorNav*. (n.d.).

https://www.vectornav.com/resources/inertial-navigation-primer/math-fundamentals/math-kalman