Application of Deep Learning Model in Sports: Detection, Classification, and Visualization of the Tracking of Ball and Players*

Ron Huang

May 7, 2024

Abstract

This paper summarizes the progress of the project on the application of the deep learning model in sports and provides a detailed explanation for each step of the tasks and the corresponding codes. Currently, most applications of machine learning models in sports, especially in soccer, which is the main sport in this project, involve the detection of players and ball tracking. However, it is much harder to track and classify each player. The major contribution of this project is to explore a new algorithm for classifying and tracking players based on the distance between bounding boxes produced by real-time detection models, You Only Look Once (YOLO). The new algorithm improves the performance of tracking and classifying players but still makes mistakes when there are multiple missing boxes. For future work, the project will focus on implementing CNN in assisting with classification tasks of both players and teams, key point detections, and projection of all movement onto a tactical board by tomography. Updates will be posted on the GitHub Repository

*Huang: Quantitative Sciences, Emory University. E-mail: ron.huang@emory.edu

1 Setup

The project is built on Google Colab with most of the required deep learning frameworks, including `keras` and `tensorflow`, but depending on different CPU or GPU backbones, there might be some small discrepancies. Here is a full list of all packages I imported and installed:

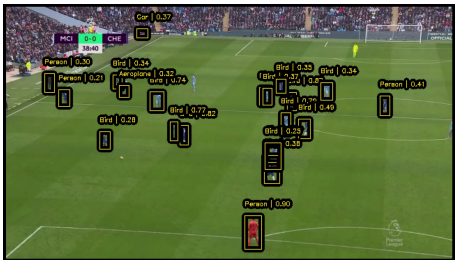
```
pip install keras_cv
pip install ultralytics

import os

os.environ["KERAS_BACKEND"] = "jax"  # @param ["tensorflow", "jax",
↪  "torch"]

from tensorflow import data as tf_data
import tensorflow_datasets as tfds
import keras
import keras_cv
from keras_cv import bounding_box
import os
from keras_cv import visualization
import tqdm
import os
from IPython.display import HTML
from base64 import b64encode
from ultralytics import YOLO
import numpy as np
import matplotlib.pyplot as plt
import cv2
from PIL import Image
from google.colab.patches import cv2_imshow
import pickle
```

Since the project is built on Google Colab, we link Google Drive to the Colab

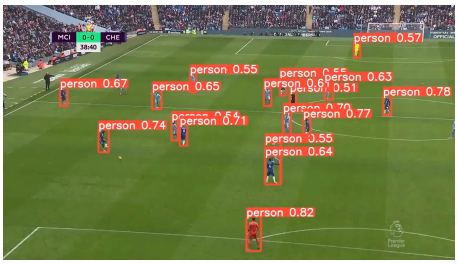
workspace so that we can efficiently save and extract files/data. To prepare for a smooth replication of the project, there are a few crucial steps of data preprocessing before getting into any prediction/visualization. First, we need to have the same folder structure/relative paths if you want to run the code with one click.

```
/content/drive/MyDrive/
|---frames
|    frame0.jpg
|---output
|    output_frame0.jpg
|---yolo_v8_results
|    output_frame0.jpg
|---yolo_spec_results
|    output_frame0.jpg
|---yolo_spec_team_results
|    output_frame0.jpg
|---player_output
|    player_output_frame0.jpg
|---ball_output
|    ball_output_frame0.jpg
|---ball_frame
|    ball_frame0.pkl
|---player_frame
|    player_frame0.pkl
|---ref_frame
|    ref_frame0.pkl
```

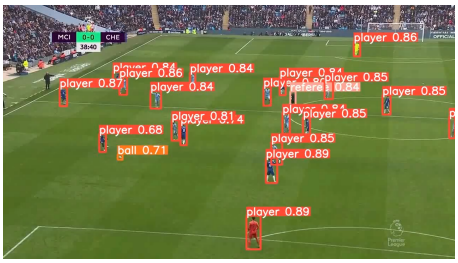
Second, a sample video needs to be uploaded to `MyDrive` and converted into frames following the codes in the notebook. It would be suggested that the length of the video should not exceed 30 seconds for a quick taste of this project. After all these, we will be well prepared to dive into the major implementation of the application of DL in Sports. For the rest of the paper, I will be presenting my results and some instructions for running the codes.



(a) `yolo_v8_m_pascalvoc`.



(b) `yolov8n`.



(c) `yolov8n_joint_detect`.

Figure 1: Object Detection

2 Object detection

For the object detection, we first experimented with two embedded models in the package: `yolo_v8_m_pascalvoc` and `yolov8n`. The first model is a pre-trained model embedded in the `keras_cv` package, and the second model is a pre-trained model embedded in the `ultralytics` package. The first model can identify most of the players, but the prediction of the bounding boxes is not very stable and sometimes detects other objects outside the soccer field. The second model can detect most of the players more consistently. However, both of the models cannot detect the ball, and it is very time consuming for us to build up a training dataset and train the model ourselves by annotating each frame. Fortunately, we found a model, `yolov8_joint_detect`, that is trained to detect the player and ball simultaneously. Currently, it is in fact not very crucial that the pre-trained has to provide the correct labeling. As long as the model can draw decent bounding boxes for each object of interest in each frame, we can customize the labeling and classification on our own. See the Jupyter Notebook for more details on how to loop through each frame, visualize the annotated frame, and compile them back together into a video. The corresponding video for each model is also embedded in the notebook.

3 Team Classification

We now turn our attention to team classification. As we have seen in Figure 1, the pre-trained model only detects objects and differentiates players from the ball. It does not classify between teams. This is relatively easy to deal with because of different jerseys' colors. We can simply pick the pixel at the very center of the bounding boxes and compare the color channel vector to the pre-specified color of the jersey. Let's see how to accomplish this by first exploring the structure of the output from YOLOv8.

```
[ultralytics.engine.results.Results object with attributes:

 boxes: ultralytics.engine.results.Boxes object
 keypoints: None
 masks: None
 names: {0: 'person', 1: 'bicycle', 2: 'car', 3: 'motorcycle', 4:
 ↪ 'airplane', 5: 'bus', 6: 'train', 7: 'truck', 8: 'boat', 9: 'traffic
 ↪ light', ...}
 obb: None
 orig_img: array([[[ 68,  50,  81],
         [ 53,  35,  66],
         [ 39,  21,  52],
         ..., dtype=uint8)
 orig_shape: (720, 1280)
 path: '/content/drive/MyDrive/frames/frame0.jpg'
 probs: None
 save_dir: 'runs/detect/predict'
 speed: {'preprocess': 2.2423267364501953, 'inference':
 ↪ 192.89803504943848, 'postprocess': 2.427816390991211}]
```

This is a standard output by `results = model.predict(image)`. To extract the bounding boxes, we use `results[0].boxes`, which will present another output structure that contains various information about the bounding boxes, including the class, confidence interval, dimension of the boxes, and the coordinates of the boxes

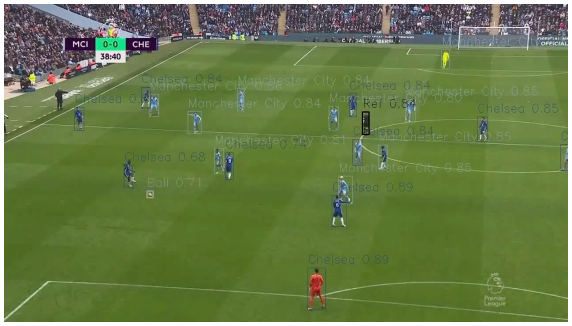


Figure 2: Team Labels

in `xyxy`, `xywh` or `xxyy`. We will use `xyxy` format, which records the coordinates of the top left and bottom right corners of the bounding boxes, for the rest of the project. `results[0].boxes` will have a length equal to the number of bounding boxes labeled in each frame. Therefore, to extract the information of each box, we use `results[0].boxes[i]` for i in the range of the length of the object, and use `results[0].boxes[i].xyxy` to extract the coordinates of the box. This is a one-dimension tensor object, so we can simply save each coordinate by `x1,y1,x2,y2 = results[0].boxes[i].xyxy[0]`. Depending on the version of your package, sometimes you may need `results[0].boxes[i].cpu()` to properly extract the data. Next, we select the pixel at the center of the image cropped based on the shape of the bounding box and the corresponding RBG values in the color channel.

```
center_x = (x1 + x2) // 2
center_y = (y1 + y2) // 2

# save the color channels for future comparison
rgb = []
for e in [0,1,2]:
    channel_data = image[center_y, center_x, e]
    rgb.append(channel_data)

col_diff = []
# calculate the color difference
for f in [ch_g, ch_p, man_g, man_p]:
  col_diff.append(np.linalg.norm(np.array(rgb) - np.array(f)))
```

`ch_g`, `ch_p`, `man_g`, and `man_p` are the pre-specified RBG values for the color of the jersey for the player and goalkeeper in each team. In other words, because the goalkeeper and other players in the same team have different colors of jersey, we need to assign the team label for goalkeepers and others separately. The algorithm is simple here: we classify based on the shortest Euclidean norm of the difference between the color vector of the center pixel and each pre-specified vector. The output video is embedded in the `First stage progress in classifying the team` section in the Jupyter Notebook.

4 Player Bounding box Classification

Now, we introduce the player classification/tracking algorithm and the code for visualizing the trajectory of each player's movement. This first attempt is also a straightforward one: for the first iteration, we predict the first frame, record all the bounding boxes labeled as a player, and save the coordinates to the corresponding list created for each player; for the rest of the iteration, we determine which bounding box belongs to which player by take the difference between each bounding box in the ith iteration and each box in the i-1th iteration. Each bounding box in the ith frame will be appended to the list which has the -1 element having the smallest difference with the bounding box in the ith frame.

```
TL = [x1, y1]
BR = [x2, y2]

frame_diff = []
for f in range(22):
  # skip if fth local()[f'player_{f}'] is empty
  if not locals()[f'player_{f}']:
    frame_diff.append(1000)
  else:
  # calculate the distance between the current box and the previous
  ↪  box
    frame_diff.append(
```

```
    np.linalg.norm(np.array(locals(
    [f'player_{f}'][-1][0]) - np.array([TL])))
# here we arbitrarily determine that if the norm of distance less
 ↪  than 10, two bounding boxes belong to the same player
if min(frame_diff) < 10:
  for n in range(22):
    if min(frame_diff) == frame_diff[n]:
      locals()[f'player_{n}'].append([TL, BR])

if min(frame_diff) >= 10:
  # if there is any empty list in locals()[f'player_{f}'], then append
   ↪  the current box to that list
  for m in range(22):
    if not locals()[f'player_{m}']:
      locals()[f'player_{m}'].append([TL, BR])
    else:
      for n in range(22):
        if min(frame_diff) == frame_diff[n]:
          locals()[f'player_{n}'].append([TL, BR])

```

However, this simple method results in unstable predictions if two players are too close to each other. Also, when one player is not detected in one frame and its sequence in the box output is ahead of one other player's, which does not belong to the player who does not have a bounding box but is the closest box to that player, this algorithm will mess up all the subsequent prediction by tracking the wrong player for the rest of the frames. To improve will introduce another algorithm that could potentially reduce the probability of making such mistakes.

5 New Algorithm for tracking player

Instead of determining the assignment of the bounding boxes to each player according to the sequence of their position in the output, we store all the differences

between each bounding box extracted from two consecutive frames in a list till the end of each iteration. We find the pair of bounding boxes that have the shortest distance to each other to be determined first, assuming that they are more likely to be correctly assigned than the rest of the pairs that have larger differences. After the first classification, we eliminate the assigned bounding box from the comparison with the rest of the boxes.

```
locals()[f'frame_diff_{seq}'].append(
np.linalg.norm(np.array(locals()[f'player_{f}'][-1][0]) -
↪  np.array([TL])))

min_diff = []
for f in range(22):
  # if temp_player is not empty
  if len(locals()[f'temp_player_{f}']) == 0:
    min_diff.append(1300)
  else:
    min_diff.append(min(locals()[f'frame_diff_{f}']))

decision_list = []
for _ in range(len(min_diff)):
  for h in range(22):
    if (min(min_diff) == min_diff[h]) & (min(min_diff)<1290):
      for z in range(22):
        if min_diff[h] == locals()[f'frame_diff_{h}'][z]:
          if z in player_unselect:
            locals()[f'player_{z}'].append(
            locals()[f'temp_player_{h}'][0])
            for y in range(22):
              if len(locals()[f'frame_diff_{y}']) != 0:
                locals()[f'frame_diff_{y}'][z] = 1290
            min_diff = []
            for f in range(22):
              # if temp_player is not empty
```

```
            if len(locals()[f'temp_player_{f}']) == 0:
              min_diff.append(1300)
            else:
              min_diff.append(min(locals()[f'frame_diff_{f}']))
        decision_list.append(h)
        for r in decision_list:
          min_diff[r] = 1290
        player_select.append(z)
        player_unselect.remove(z)
```

This effectively prevents double selection and wrong selection due to missing bounding boxes in the previous frame. In other words, each frame can only be selected once, and the matching process will only include the unmatched boxes. Imagine when two players are very close to each other, which causes the model to omit one of the bounding boxes in the ith frame: for the player who is not annotated by the bounding box, the closest bounding box for him would be the one of another player. This results in misclassification for all of the rest boxes for this player. In the new algorithm, that bounding box will not be misclassified because it will be assigned to the right player with the shortest distance first. See the recompiled video in the `Third stage progress in classifying players` section in Jupyter Notebook.

6 Conclusion and Future work

The new algorithm largely improves the performance of the tracking, but it is still not perfect in cases where there are multiple missing bounding boxes (and on average there are around 3 missing ones per frame). You can compare the video in `Second stage progress in Tracking Visualization` section and another one in `Third stage progress in classifying players` section in the Jupyter Notebook to see the difference between the first simple algorithm and the updated one.

There are several things left before we can perfectly wrap up this project: the Key Point detection task, Homography projection, improvement on team classification with the CNN, and improvement on player tracking by combining the previous

Figure 3: Sample Training Data for CNN in Classification test

algorithm with CNN. Due to time constraints and limited human resources, we will not be able to start those parts of the project as of May 7, 2024. However, any update will be posted on the GitHub Repository

For the team classification problem, we will be able to improve it by training a CNN to predict the team label using the training data created by extracting the images cropped by the bounding boxes. This could potentially further stabilize the prediction. This training data will also be useful for the classification between players, though it needs to be a combination of two methods. As mentioned, we can mostly correctly classify and track players by calculating the distance between bounding boxes with an updated decision-making algorithm. However, when the overlapped players are all undetected, the classification beginning the next iteration will be all wrong. One solution is that we can depend on CNN when differencing is insufficient to make a correct decision. For example, when two players get overlapped and both bounding boxes disappear for several frames, and further suppose these are the only two players overlapped who are undetected at the moment, we apply CNN by training a model using the cropped images inside the bounding boxes to assist the classification task. This is easy to automate, since when bounding boxes disappear, we append "None" to the list. Therefore, we apply CNN for players who have "None" in the current. Given that usually 3-4 may be missing on average in a frame, this will simply be a CNN with an MNL output layer.