Player Re-identification in Sports Footage.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

Task 01 - Cross Camera Player Mapping

Available Contents

1. Pre-Trained Model named as 'best.pt'
2. Footages from 2 Cameras - 'Broadcast.mp4' and 'Tacticam.mp4'

Importing Basic Libraries

```
[20] import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     from matplotlib_venn import venn2
```

Installing Torch and Ultralytics Modules

```
[2] ! pip install torch
```

Show hidden output

```
! pip install ultralytics
```

Show hidden output

Loading Pre-Trained Model

```
[4] from ultralytics import YOLO

    load_model = YOLO('best.pt')
```

```
Creating new Ultralytics Settings v0.0.6 file
View Ultralytics Settings with 'yolo settings' or at '/root/.config/Ultralytics/settings.json'
Update Settings with 'yolo settings key=value', i.e. 'yolo settings runs_dir=path/to/dir'. For help see https://docs.ultralytics.com/quickstart/#ultralytics-settings.
```

Tracking the Players using pre-trained model for Broadcast sample Footage

```
[5] results_01 = load_model.track('/content/broadcast.mp4',)
```

Show hidden output

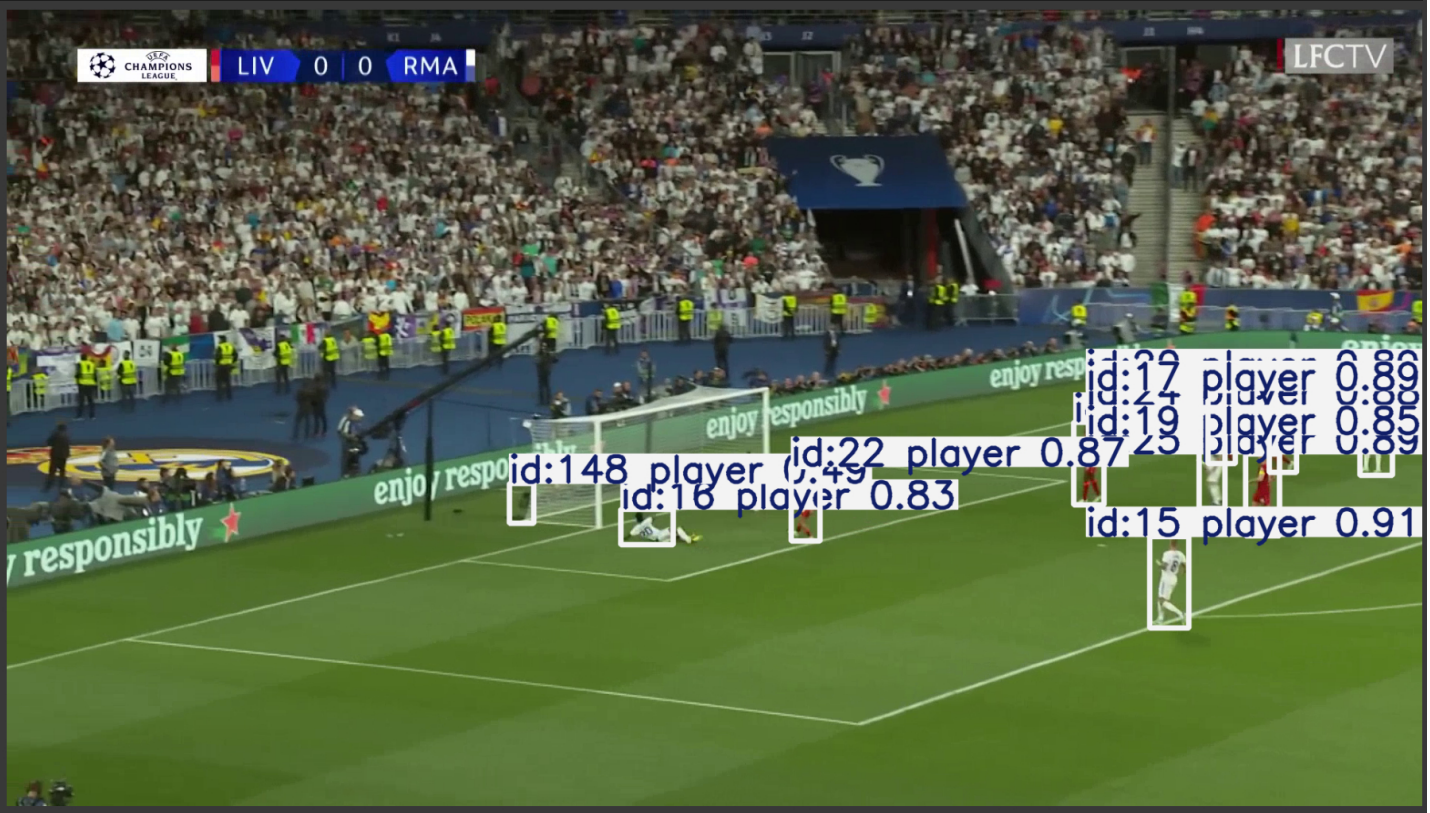
```
[40] # let's see the outputs of results_01
     results_01[100].boxes
```

```
ultralytics.engine.results.Boxes object with attributes:

cls: tensor([2., 2., 2., 2., 2., 2., 2., 2., 2., 2.])
conf: tensor([0.9083, 0.8854, 0.8491, 0.8691, 0.8833, 0.8935, 0.9092, 0.9144, 0.8928, 0.4879, 0.8255])
data: tensor([[1.5490e+03, 7.1434e+02, 1.6028e+03, 8.3872e+02, 1.5000e+01, 9.0833e-01, 2.0000e+00],
        [1.6282e+03, 5.1679e+02, 1.6659e+03, 6.1522e+02, 1.7000e+01, 8.8535e-01, 2.0000e+00],
        [1.6169e+03, 5.7669e+02, 1.6513e+03, 6.7695e+02, 1.9000e+01, 8.4907e-01, 2.0000e+00],
        [1.0635e+03, 6.1922e+02, 1.1023e+03, 7.2098e+02, 2.2000e+01, 8.6913e-01, 2.0000e+00],
        [1.7117e+03, 5.3634e+02, 1.7482e+03, 6.2772e+02, 2.4000e+01, 8.8329e-01, 2.0000e+00],
        [1.6791e+03, 6.0259e+02, 1.7251e+03, 7.1331e+02, 2.5000e+01, 8.9352e-01, 2.0000e+00],
        [1.8347e+03, 5.2707e+02, 1.8781e+03, 6.3190e+02, 2.1000e+01, 9.0923e-01, 2.0000e+00],
        [1.4467e+03, 5.5959e+02, 1.4877e+03, 6.7197e+02, 1.8000e+01, 9.1438e-01, 2.0000e+00],
        [1.5839e+03, 5.0044e+02, 1.6263e+03, 5.8166e+02, 2.0000e+01, 8.9281e-01, 2.0000e+00],
        [6.8117e+02, 6.4261e+02, 7.1412e+02, 6.9604e+02, 1.4800e+02, 4.8790e-01, 2.0000e+00],
        [8.3279e+02, 6.7771e+02, 9.0380e+02, 7.2550e+02, 1.6000e+01, 8.2549e-01, 2.0000e+00]])
id: tensor([ 15.,  17.,  19.,  22.,  24.,  25.,  21.,  18.,  20., 148.,  16.])
is_track: True
orig_shape: (1080, 1920)
shape: torch.Size([11, 7])
xywh: tensor([[1575.8838,  776.5335,   53.7401,  124.3771],
        [1647.0769,  566.0057,   37.6844,   98.4230],
        [1634.0858,  626.8204,   34.4116,  100.2598],
        [1082.8922,  670.1006,   38.8394,  101.7550],
        [1729.9365,  582.0326,   36.4890,   91.3755],
        [1702.1151,  657.9490,   45.9324,  110.7166],
        [1856.3928,  579.4862,   43.3376,  104.8293],
        [1467.2012,  615.7772,   41.0491,  112.3762],
        [1605.1067,  541.0529,   42.4707,   81.2199],
        [ 697.6461,  669.3243,   32.9577,   53.4291],
        [ 868.2930,  701.6035,   71.0123,   47.7927]])
xywhn: tensor([[0.8208, 0.7190, 0.0280, 0.1152],
        [0.8579, 0.5241, 0.0196, 0.0911],
        [0.8511, 0.5804, 0.0179, 0.0928],
        [0.5640, 0.6205, 0.0202, 0.0942],
        [0.9010, 0.5389, 0.0190, 0.0846],
        [0.8865, 0.6092, 0.0239, 0.1025],
        [0.9669, 0.5366, 0.0226, 0.0971],
        [0.7642, 0.5702, 0.0214, 0.1041],
        [0.8360, 0.5010, 0.0221, 0.0752],
        [0.3634, 0.6197, 0.0172, 0.0495],
        [0.4522, 0.6496, 0.0370, 0.0443]])
xyxy: tensor([[1549.0138,  714.3450, 1602.7539,  838.7220],
        [1628.2346,  516.7943, 1665.9191,  615.2172],
        [1616.8800,  576.6906, 1651.2916,  676.9503],
        [1063.4725,  619.2231, 1102.3119,  720.9781],
```

```
            [1711.6920,  536.3448, 1748.1810,  627.7203],
            [1679.1489,  602.5908, 1725.0813,  713.3073],
            [1834.7240,  527.0715, 1878.0616,  631.9009],
            [1446.6766,  559.5891, 1487.7257,  671.9653],
            [1583.8713,  500.4430, 1626.3420,  581.6629],
            [ 681.1673,  642.6098,  714.1250,  696.0389],
            [ 832.7868,  677.7072,  903.7991,  725.4998]])
    xyxyn: tensor([[0.8068, 0.6614, 0.8348, 0.7766],
            [0.8480, 0.4785, 0.8677, 0.5696],
            [0.8421, 0.5340, 0.8600, 0.6268],
            [0.5539, 0.5734, 0.5741, 0.6676],
            [0.8915, 0.4966, 0.9105, 0.5812],
            [0.8746, 0.5580, 0.8985, 0.6605],
```

```
[34] results_01[100].show()
```



Creating DataFrame for the Classes and Player ID

```
[28] # creating 2 lists as player_id and classes
     player_id  = []
     classes = []

     # using a for loop to iterate over the results_01.boxes
     for i in range(len(results_01)):
       if results_01[i].boxes.id==None:
          continue
       else:
         player = results_01[i].boxes.id
         player = player.tolist()
         classs = results_01[i].boxes.cls
         classs = classs.tolist()
         player_id.extend(player)
         classes.extend(classs)

     # verification of player_id and respective classes
     if len(player_id)==len(classes):
       broadcast = pd.DataFrame()
       broadcast['player_id'] = player_id
       broadcast['classes'] = classes
       print(broadcast.head(5))

     else:
       print("Error : Check the looping code")
```

```
    player_id  classes
0         1.0      2.0
1         2.0      2.0
2         3.0      2.0
3         1.0      2.0
4        15.0      2.0
```

Player_ID in Broadcast Video Sample

```
[29] # Identifying the unique player_ids from the dataframe
     broadcast_player_id = broadcast[broadcast['classes']==2]['player_id'].unique()
```

Tracking the Players using pre-trained model for Tacticam sample Footage

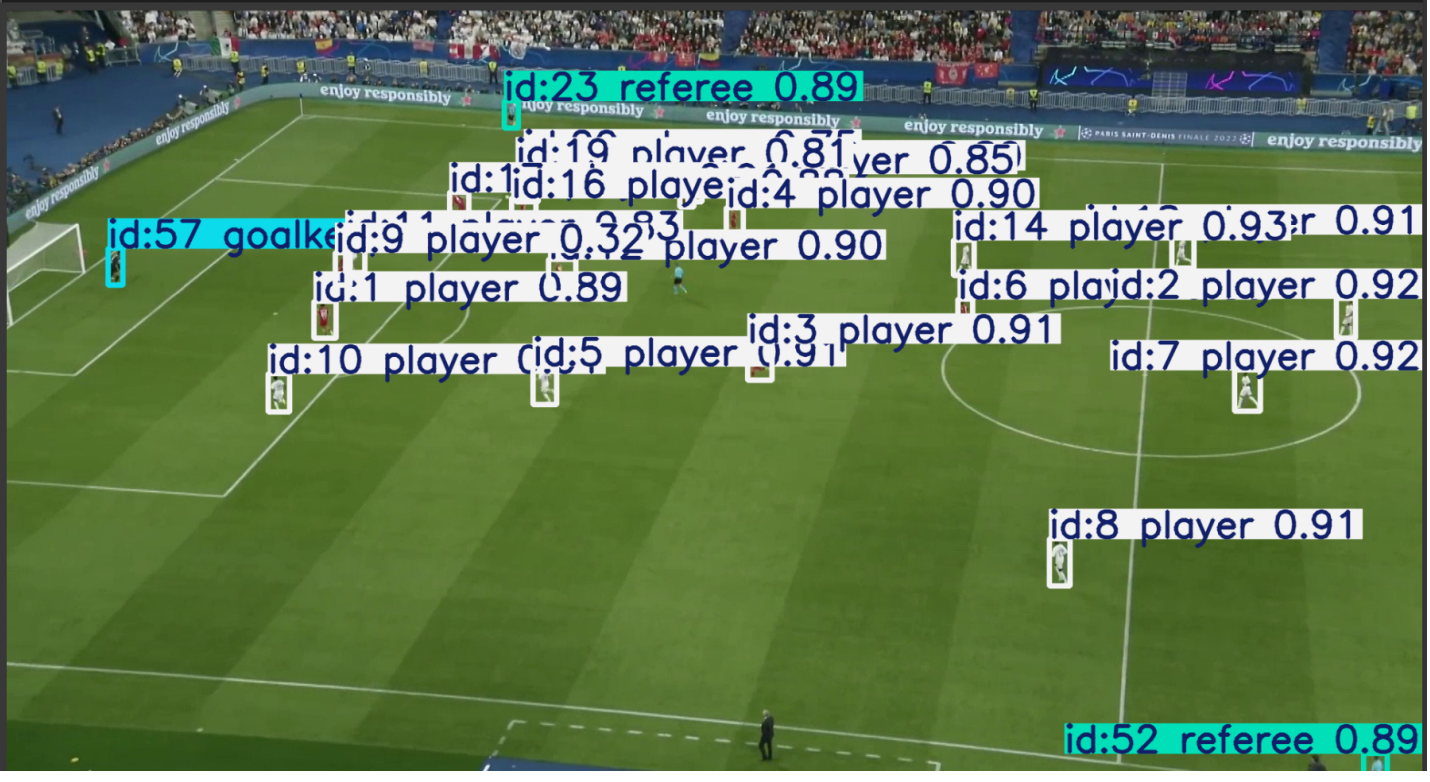
```
[10] results_02 = load_model.track('/content/tacticam.mp4')
```

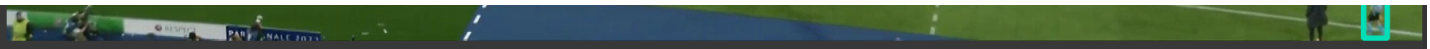
Show hidden output

```
[41] # Let's see the output of results_02
     results_02[100].boxes
```

```
        [0.3195, 0.2457, 0.0139, 0.0357],
        [0.3674, 0.2183, 0.0157, 0.0433],
        [0.4802, 0.2258, 0.0103, 0.0410],
        [0.6767, 0.3851, 0.0105, 0.0454],
        [0.6757, 0.3109, 0.0143, 0.0449],
        [0.9662, 0.9646, 0.0166, 0.0640],
        [0.3726, 0.2088, 0.0143, 0.0435],
        [0.3564, 0.1303, 0.0097, 0.0338],
        [0.4858, 0.2200, 0.0100, 0.0397],
        [0.2247, 0.3884, 0.0156, 0.0461],
        [0.8305, 0.3025, 0.0155, 0.0421],
        [0.0767, 0.3220, 0.0105, 0.0464]])
xyxy: tensor([[1802.4354,  390.4871, 1828.1942,  444.4698],
        [1004.3550,  451.1995, 1037.0745,  501.8501],
        [ 975.1485,  267.3763,  998.7192,  311.2076],
        [ 713.9367,  482.5835,  745.2928,  534.5438],
        [1664.0876,  487.7961, 1699.3809,  543.1016],
        [1413.5503,  716.4907, 1441.2893,  780.6464],
        [ 445.0092,  328.7304,  465.2719,  372.0719],
        [ 354.9276,  492.4080,  383.2590,  544.7711],
        [ 459.2202,  311.9779,  485.8131,  355.2376],
        [ 734.9860,  337.8866,  771.3984,  383.0435],
        [ 684.2465,  254.6015,  717.4453,  297.9273],
        [ 600.0575,  246.1340,  626.7221,  284.6746],
        [ 690.3668,  212.3456,  720.5862,  259.1288],
        [ 912.0261,  221.7384,  931.8187,  266.0652],
        [1289.1836,  391.4364, 1309.4069,  440.4361],
        [1283.7084,  311.5000, 1311.1519,  359.9598],
        [1839.1405, 1007.2321, 1871.0947, 1076.3705],
        [ 701.6271,  201.9924,  729.1513,  249.0048],
        [ 674.9760,  122.4880,  693.5432,  159.0402],
        [ 923.0416,  216.1253,  942.2476,  259.0158],
        [ 416.4179,  394.6216,  446.2827,  444.3657],
        [1579.7311,  303.9303, 1609.4269,  349.3869],
        [ 137.1783,  322.7448,  157.2872,  372.8539]])
xyxyn: tensor([[0.9388, 0.3616, 0.9522, 0.4115],
        [0.5231, 0.4178, 0.5401, 0.4647],
        [0.5079, 0.2476, 0.5202, 0.2882],
        [0.3718, 0.4468, 0.3882, 0.4949],
        [0.8667, 0.4517, 0.8851, 0.5029],
        [0.7362, 0.6634, 0.7507, 0.7228],
        [0.2318, 0.3044, 0.2423, 0.3445],
        [0.1849, 0.4559, 0.1996, 0.5044],
        [0.2392, 0.2889, 0.2530, 0.3289],
        [0.3828, 0.3129, 0.4018, 0.3547],
        [0.3564, 0.2357, 0.3737, 0.2759],
        [0.3125, 0.2279, 0.3264, 0.2636],
        [0.3596, 0.1966, 0.3753, 0.2399],
        [0.4750, 0.2053, 0.4853, 0.2464],
        [0.6714, 0.3624, 0.6820, 0.4078],
        [0.6686, 0.2884, 0.6829, 0.3333],
        [0.9579, 0.9326, 0.9745, 0.9966],
        [0.3654, 0.1870, 0.3798, 0.2306],
        [0.3516, 0.1134, 0.3612, 0.1473],
        [0.4808, 0.2001, 0.4908, 0.2398],
        [0.2169, 0.3654, 0.2324, 0.4114],
        [0.8228, 0.2814, 0.8382, 0.3235],
        [0.0714, 0.2988, 0.0819, 0.3452]])
```

```
results_02[100].show()
```





Creating a dataframe for the Tacticam player_id and Classes

```
[30] # Defining two list for player id and classes respectively
     player_id_2  = []
     classes_2 = []

     # Looping through the results_02.boxes
     for i in range(len(results_02)):
       if results_02[i].boxes.id==None:
          continue
       else:
         player = results_02[i].boxes.id
         player = player.tolist()
         classs = results_02[i].boxes.cls
         classs = classs.tolist()
         player_id_2.extend(player)
         classes_2.extend(classs)

     # Verifying the player_id with respective classes and creating DataFrame
     if len(player_id_2)==len(classes_2):
         tacticam = pd.DataFrame()
         tacticam['player_id'] = player_id_2
         tacticam['classes'] = classes_2
         print(tacticam.head(5))

     else:
       print("Error : check the looping code")
```

```
   player_id  classes
0        1.0      2.0
1        2.0      2.0
2        3.0      2.0
3        4.0      2.0
4        5.0      2.0
```

Player Id in Tacticam Footage

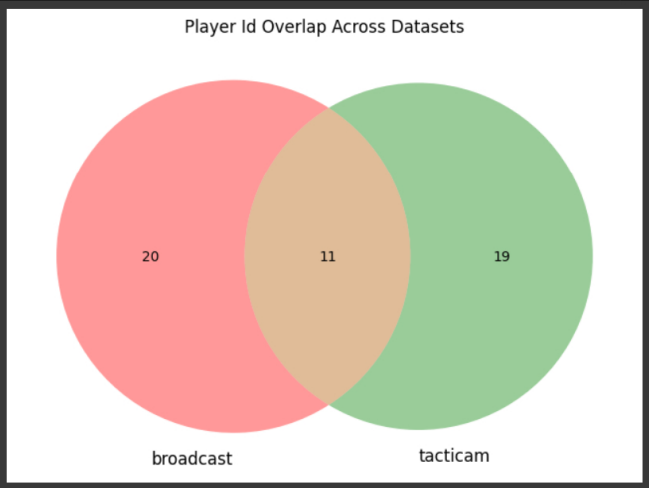
```
[31] # Identifying the unique player ids in Tacticam footage and extracting to variable
     tacticam_player_id = tacticam[tacticam['classes']==2]['player_id'].unique()
```

Let's Visualize the Common Players in both Footage using Venn Diagram

```
[32] # Defining DataFrame consisting of only players (Excluding - Refree, goalkeeper and ball)
     broadcast_df = broadcast[broadcast['classes']==2]
     tacticam_df = tacticam[tacticam['classes']==2]

     # unique player_ids in Dataframes
     set1 = set(broadcast_df['player_id'])
     set2 = set(tacticam_df['player_id'])

     # Visualization of Venn Diagram
     plt.figure(figsize=(8,6))
     venn2([set1,set2],set_labels=['broadcast','tacticam'])
     plt.title("Player Id Overlap Across Datasets")
     plt.show()
```



Player Id in Both Footage

```
[16] player_id_mapped = []
     for i in tacticam_player_id:
       if i in broadcast_player_id:
        player_id_mapped.append(int(i))

     print(f"Player Id Mapped in both videos : {player_id_mapped}")
```

```
Player Id Mapped in both videos : [1, 2, 3, 15, 16, 17, 18, 19, 20, 21, 23]
```


[ ] Start coding or generate with AI.
Colab paid products - Cancel contracts here
Variables
Terminal
12:10
T4 (Python 3)