Pitch geometry recruitment task description

1 Introduction

Since my goal is to develop a model that estimates both visibility and position of each of the football pitch keypoints and processing speed of the model should be considered, YOLO architecture seems to be suitable for this task. I except that it would outperform the classic fully convolutional neural network (FCNN). Although it is feasible to solve this problem using a classical FCNN with transfer learning, the estimation of keypoint positions would require additional post-processing, scanning, or a more complex model involving classification of keypoints and regression of coordinates.

The bounding boxes featured in the YOLO architecture return both the class and location in the image, and the entire model is much faster. This solution can be easily scaled to make predictions on videos, such as live football matches.

I used YOLOv7 – last version that is scallable to 1280x1280 image input. YOLOv8 supports size 640px only. I continued training for weights downloaded from official github with changed classes, so my model is based on transfer learning.

I used YOLOv7, the latest version that is scalable to a 1280x1280 image input. YOLOv8, on the other hand, only supports a size of 640px. I continued training using weights downloaded from the official GitHub repository, but with modified classes. Therefore, my model is based on transfer learning.

2 Exploratory data analysis

EDA is conducted in the Jupyter Notebook titled '1_EDA.ipynb – please refer to it for more details. Here are some brief findings about the task dataset:

- No missing values
- *vis* column values are correct
- A few instances of mislabeled keypoints, like 29 confused with 39 below and other confusions – but these occurances are very rare. Due to the large size of the dataset (almost 4000 images), it was not possible to thoroughly check every instance. However, a sample of 100 images was randomly checked, but some mistakes in the dataset may still remain unnoticed.
- 45 images had unlabelled keypoints, but were marked as *train.* They were all discarded.

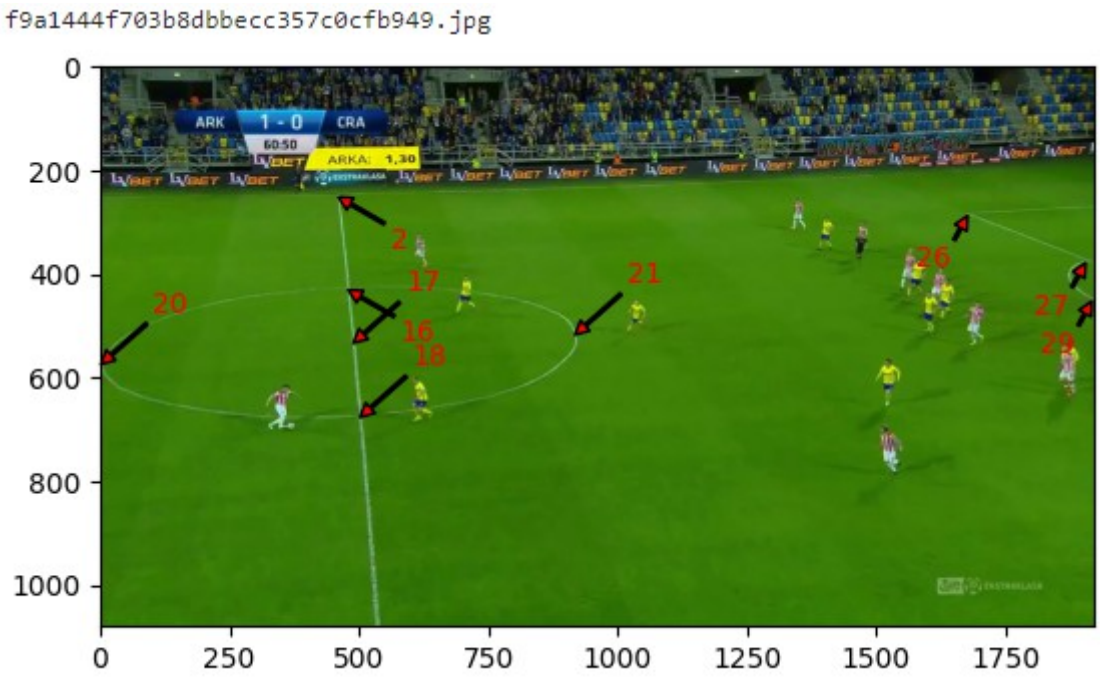

Fig. 1 - Example of confused label (29, should be 39)

- Classes are heavily imbalanced – keypoints related to bottom corners of the pitch (3 and 23) are extremely rare – 23 and 34 images respectively. Similarly, keypoint 10 – the intersection of bottom side of the pitch and half line is rare – appearing in only 150 images, compared to some of the most common classes like 2, 16, 17 (1910, 1978, 1989 images respectively.)

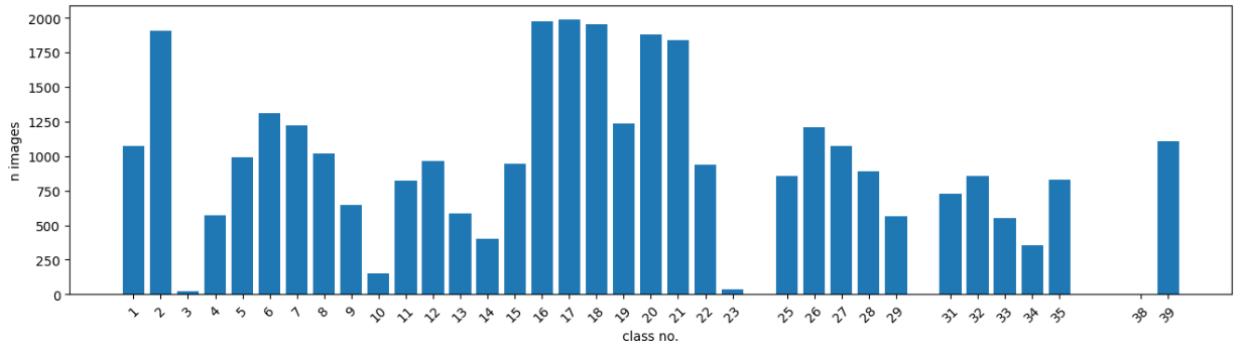

Fig. 2 - Class distribution across the dataset

- Classes 24, 36, 37, 38 (the last has only one image) are missing, but this is correct according to task manual.
- Class 30 is missing , which is incorrect according to the task manual. Images containing this particular keypoint should be labelled manually. Its rarity will be similar as the class on the other side of the pitch – 10, so I except a population of 150 – 200 images (more, because neighboring class 23 has higher population than class 3 – corresponding on the other side of the pitch). In order to identify potential images that may include class 30, an assumption was made that any of the neighboring classes (23, 29, 33, 34) should also be visible.

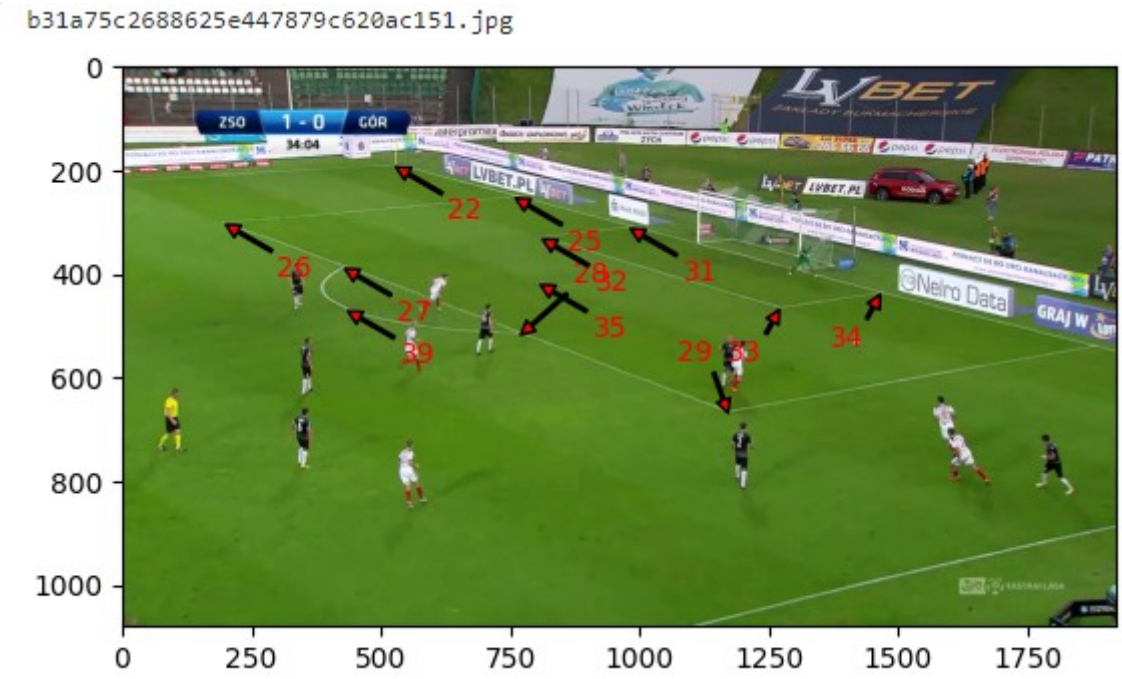

Fig. 3 - Example of missing keypoint 30

- Since manually labeling a large number of images can be time-consuming, I have decided to estimate the position of class 30 using the following approach:

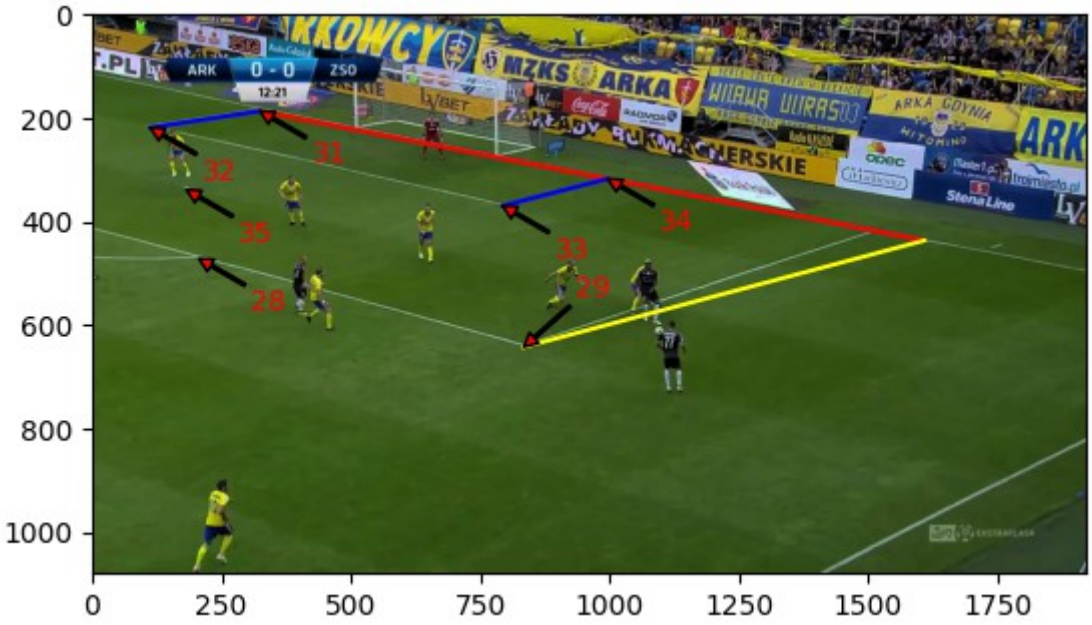

Fig. 4 - How to estimate position of keypoint 30?

1. Keypoint 30 will certainly lay on line that includes points 31 and 34. At first I hoped that parallelity to line 33-34 would be enough (in reality this is obviously false) but it was not, as can be seen above.
2. It can be noticed that given the line form Ax + By + C = 0, line 31-32 has quite low and negative slope (*m* = -A/B), line 33-34has a slightly higher, negative slope,

3. Therefore, the line formed by points 29-30 (shown in yellow in the picture) would have a slightly higher slope than that of 33-34, but still lower than that of 31-32 in relation to the 31-32 line.
4. The solution involves empirically determining the correct ratio and randomizing a point on the 31-34 line, within additional boundaries such as ensuring that the x-coordinate is higher than x34 and less than 1920.
5. It is important to note that while this idea is not always accurate. The majority of points fit well, but some points may deviate slightly. At the end almost all of them are included within the YOLO boundary box.
6. A few images did not include the required points, and thus, I labeled them manually. Ideally, all images should be labeled manually, but this process is time-consuming. Alternatively, an improved estimation algorithm could be developed.

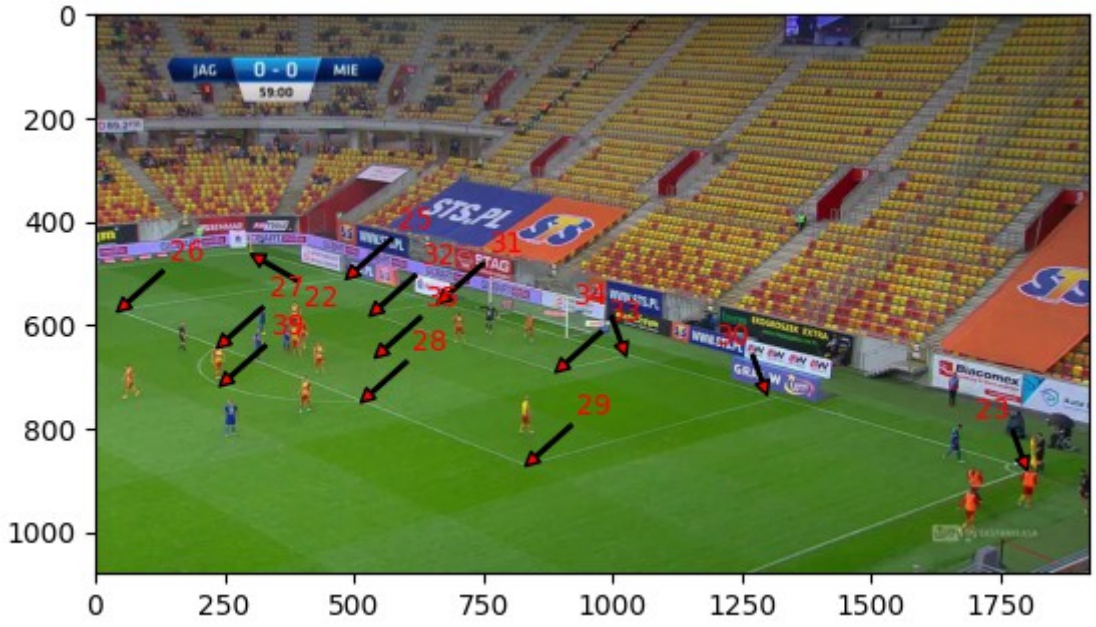

Fig. 5 - Calculated approximate position of keypoint 30

3 Data preparation

All steps from previous section led to csv file with corrected labels, that was futher passed to Jupyter Notebook titled '2_Data_Preparation.ipynb' – please refer to it for more details. Here is a brief summary of data preparation pipeline:

- Despite heavy class imbalance, data augmentation is not applied for first training. I wanted to know the results without augmentation.
- Random train – validation split resulting in 0.75/0.25 split was made.
- Class balance between train and validation set was compared. It was pretty much the same.

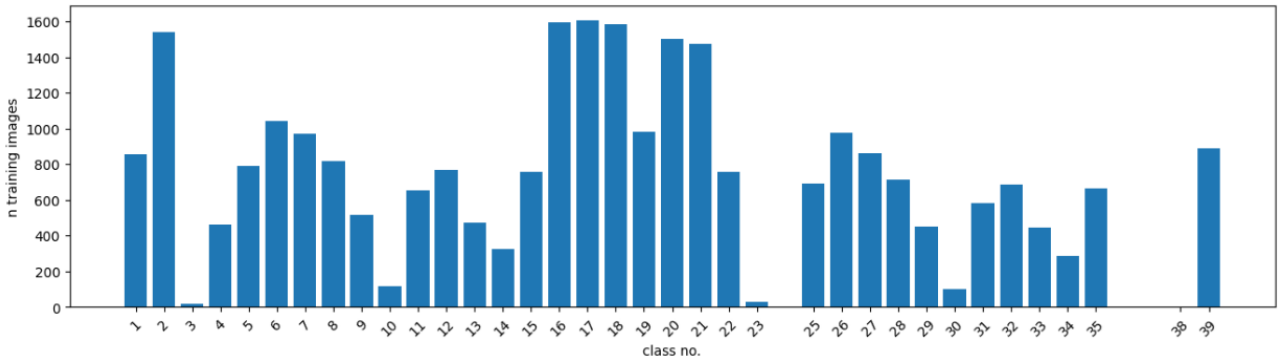

Fig. 6 - Class distribution across the training dataset

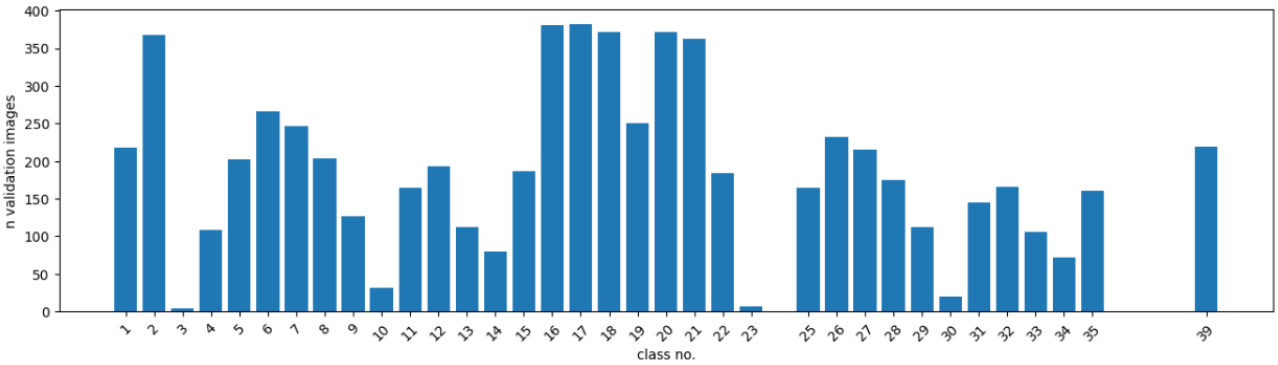

Fig. 7 - Class distribution across the validation dataset

- Files with labels are generated in accordance with YOLO format: *<object-class-id> <x> <y> <width> <height>*. Width and height define the square boundary box with side length 100 px (arbitrarily chosen). Note that x, y, with, height have to be normalized.

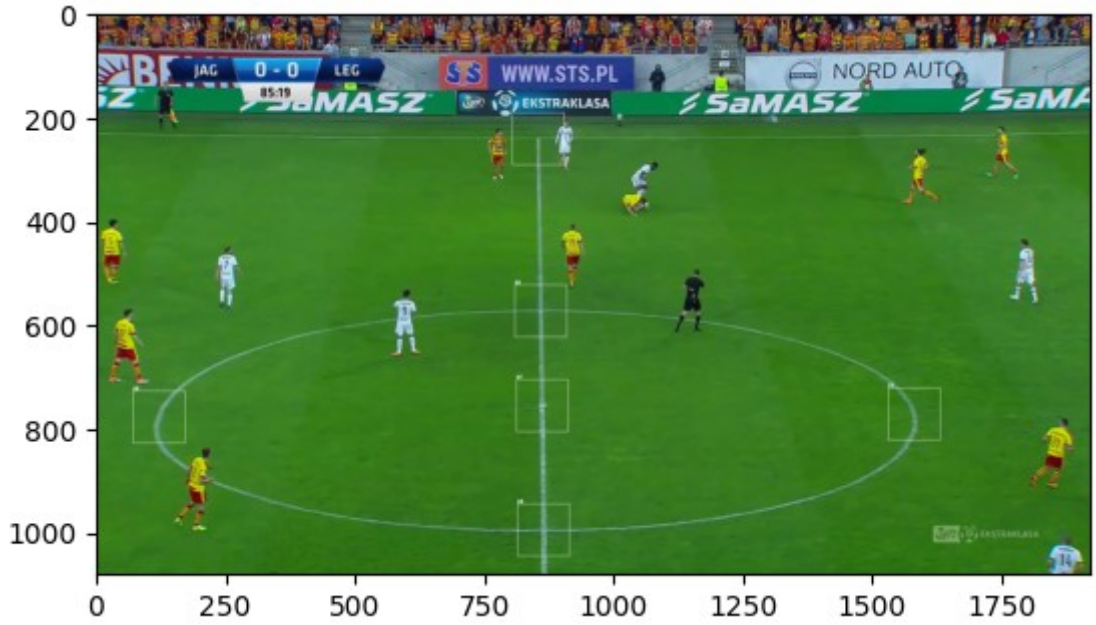

Fig. 8 - Exemplary bounding boxes

- Files and labels were divided in corresponding *train* and *val* folders in accordance with YOLOv7 documentation.

4 Training

Model was trained in accordance with YOLOv7 documentation using the CLI, but first:

- The yolov7.yaml config file that contains definition of FCNN architecture was edited. The value of "nc" (number of classes) was set to 40 to avoid confusion and allow for the inclusion of class number 39. YOLO restricts class names to be within the range of 0-34 for "nc" values of 35. It adds some parameters to the model, but does not change anything in case of results. New config file was named yolov7_pitch.yaml
- coco.yaml config file was renamed to pitch_data.yaml. Lines which download COCO dataset and define test directory were removed, „nc" was set to 40 and classes were renamed.
- yolov7.pt weights were downloaded and placed in main folder.
- Training with batch size of 8 images, 100 epochs and default hyperparameters for transfer learning was executed with following command:

python train.py --workers 1 --device 0 --batch-size 8 --epochs 100 --img 640 640 --data data/pitch_data.yaml --hyp data/hyp.scratch.custom.yaml --cfg cfg/training/yolov7_pitch.yaml --name yolov7_pitch_0 --weights yolov7.pt

Loss functions, metrices charts and training summary is presented below:

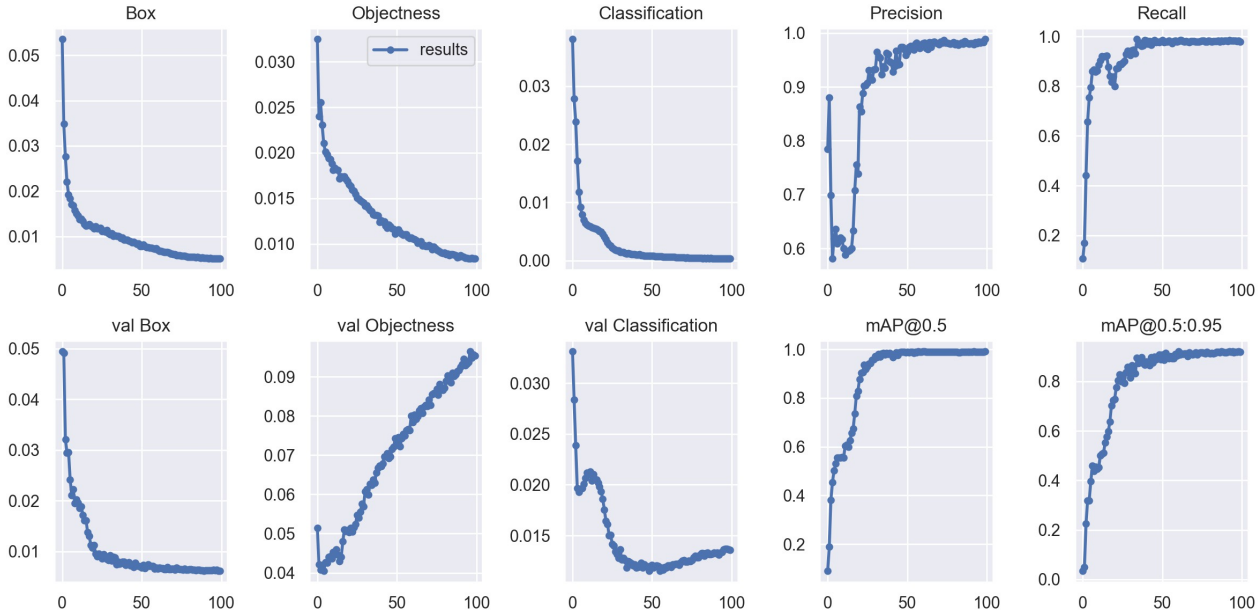

Fig. 9 - Losses and metrics charts generated by YOLO

We can see that the training could be ended near 50 epoch, since then classification loss on validation set started increasing, which may indicate overfitting. On the other hand precision and mean average precision (mAP) improved in further training, as well as localization loss – which will strongly affect precision of predicted coordinates.

```
     99/99     7.44G  0.005086  0.008384 0.0003575   0.01383        12       640: 100%|
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95:
                 all         752        6608       0.989       0.979       0.991       0.919
                   1         752         218           1       0.968       0.991       0.974
                   2         752         368       0.997       0.995       0.994       0.989
                   3         752           4       0.976           1       0.995       0.866
                   4         752         109       0.982       0.985       0.993       0.967
                   5         752         202       0.995       0.957       0.983       0.937
                   6         752         266       0.992       0.977       0.995       0.979
                   7         752         246       0.988       0.992        0.99       0.964
                   8         752         203       0.995       0.986       0.993       0.945
                   9         752         127           1       0.991       0.995       0.975
                  10         752          31           1       0.977       0.995       0.971
                  11         752         164       0.993        0.97       0.984        0.95
                  12         752         193       0.995       0.969       0.994       0.955
                  13         752         112           1       0.992       0.996       0.975
                  14         752          80        0.99           1       0.996       0.971
                  15         752         187       0.983       0.935       0.973       0.942
                  16         752         380       0.997       0.997       0.996       0.993
                  17         752         382       0.995       0.997       0.998       0.986
                  18         752         372       0.996       0.997       0.996       0.991
                  19         752         251       0.987       0.937       0.977       0.617
                  20         752         372       0.995       0.994       0.996        0.89
                  21         752         363       0.997       0.992       0.997       0.884
                  22         752         184       0.994       0.981       0.993       0.966
                  23         752           7       0.908           1       0.995       0.851
                  25         752         164       0.976       0.985        0.99        0.95
                  26         752         232       0.996       0.975       0.995       0.965
                  27         752         215       0.995       0.991       0.996       0.964
                  28         752         175       0.994       0.985       0.996       0.922
                  29         752         112       0.987           1       0.996       0.973
                  30         752          20           1        0.95       0.946       0.572
                  31         752         145       0.988       0.993       0.996       0.941
                  32         752         166       0.993           1       0.996        0.95
                  33         752         106        0.99       0.991       0.994       0.939
                  34         752          72       0.986       0.984       0.986       0.927
                  35         752         161       0.962       0.935       0.989        0.94
                 39'         752         219        0.98       0.902       0.987       0.584
100 epochs completed in 11.485 hours.
```

Fig. 10 - Training summary

We can see that in general model performs very well, although mAP for higher IoU values is much lower for some classes:

- 3 and 23, what is probably caused by their rarity
- 30, what is probably caused by insufficient precision of boundary box placement. It is indicated by good performance of corresponding keypoint 10 - on the other side of the pitch. Manually labelingf this point will certainly help.
- 19 and 39, what is probably caused by generic difficulties of the model to catch proper fragment of the curve. We can observe similar but weaker effect for classes 20 and 21, but these areas have often more friendly camera angle . In my opinion it is easier to precisely label proper pixel when the camera field is straight, than when it is angled – see pictures below for clarification. This could be fixed by bigger boundary box for these points – big enough to catch line 7-8 for point 19 and line 27-28 for point 39.

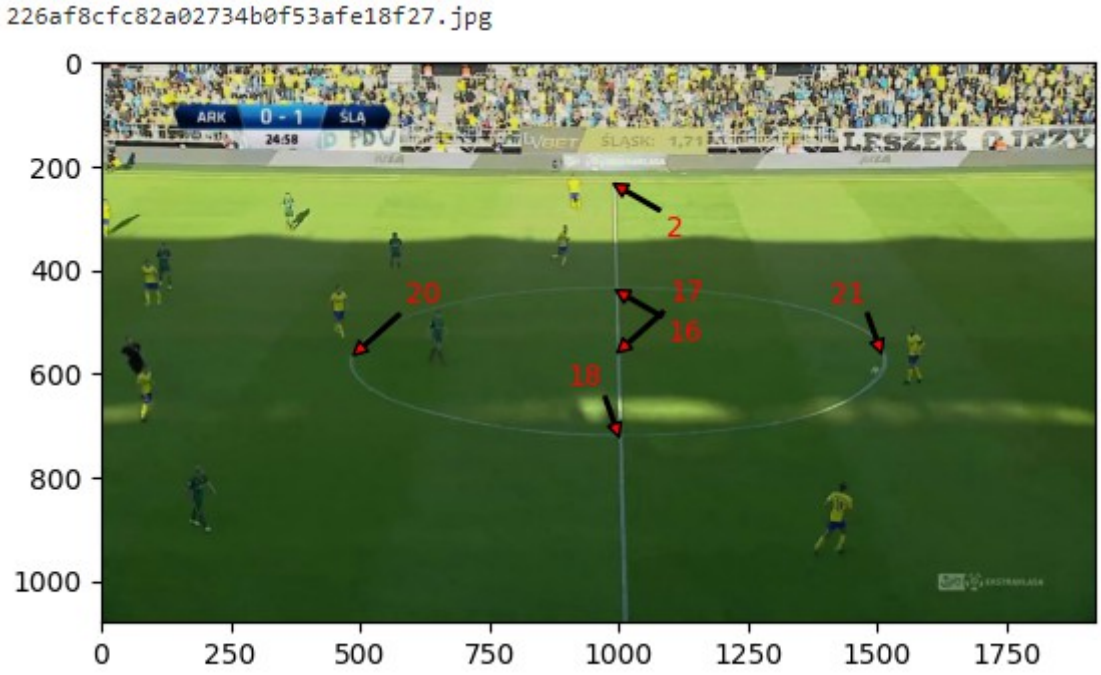

Fig. 11 – Example of camera angle for points 20 and 21

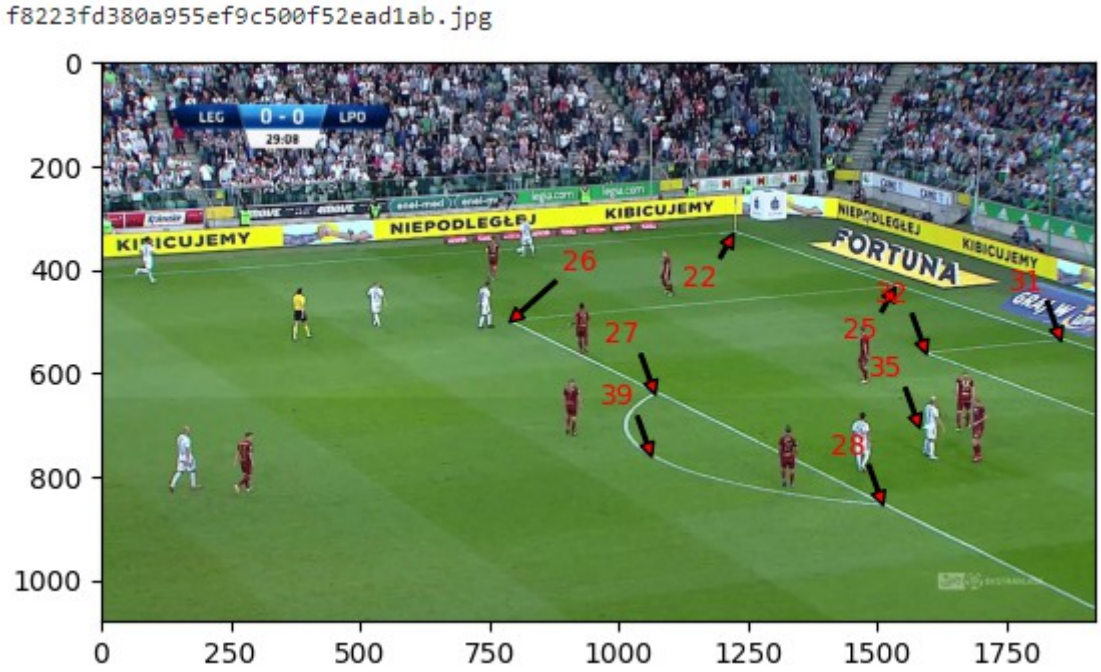

Fig. 12 - Example of camera angle for point 39

- At first I suspected that the model confuses point 19 and 39 corresponingly with 20 and 21 – see picture below, but this idea does not find confirmation in confusion matrix below. Generally based on confusion matrix I can claim that the model is really robust. Confusion matrix in better resolution is located in folder runs/train/yolov7_pitch_02.

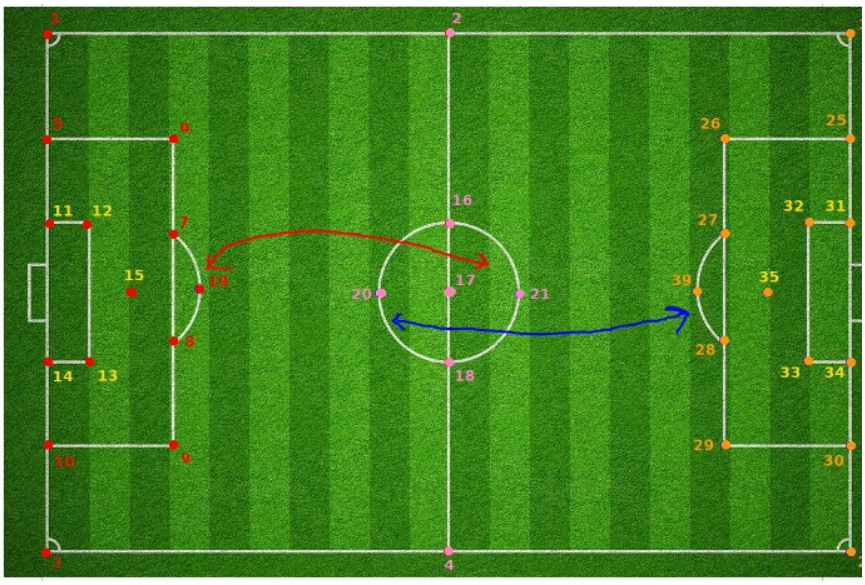

Fig. 13 - Potential confusion

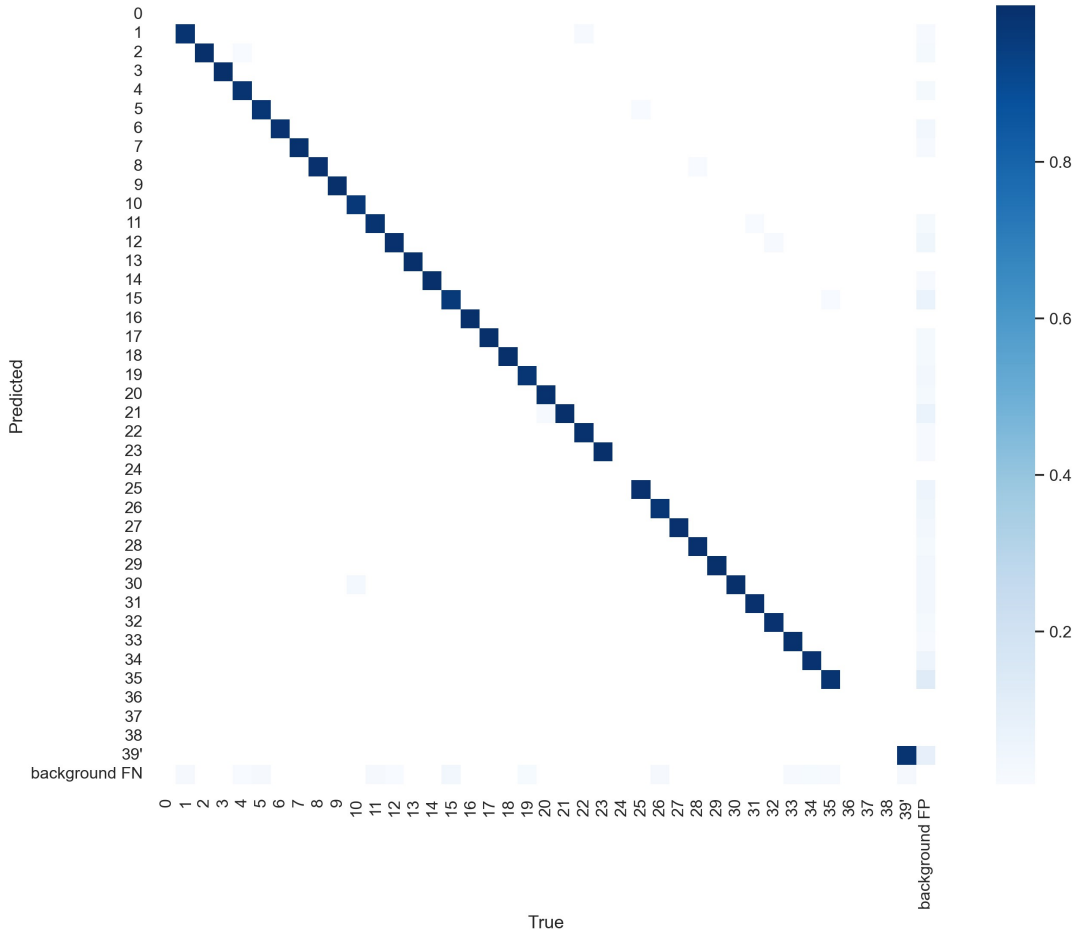

Fig. 14 Confusion matrix

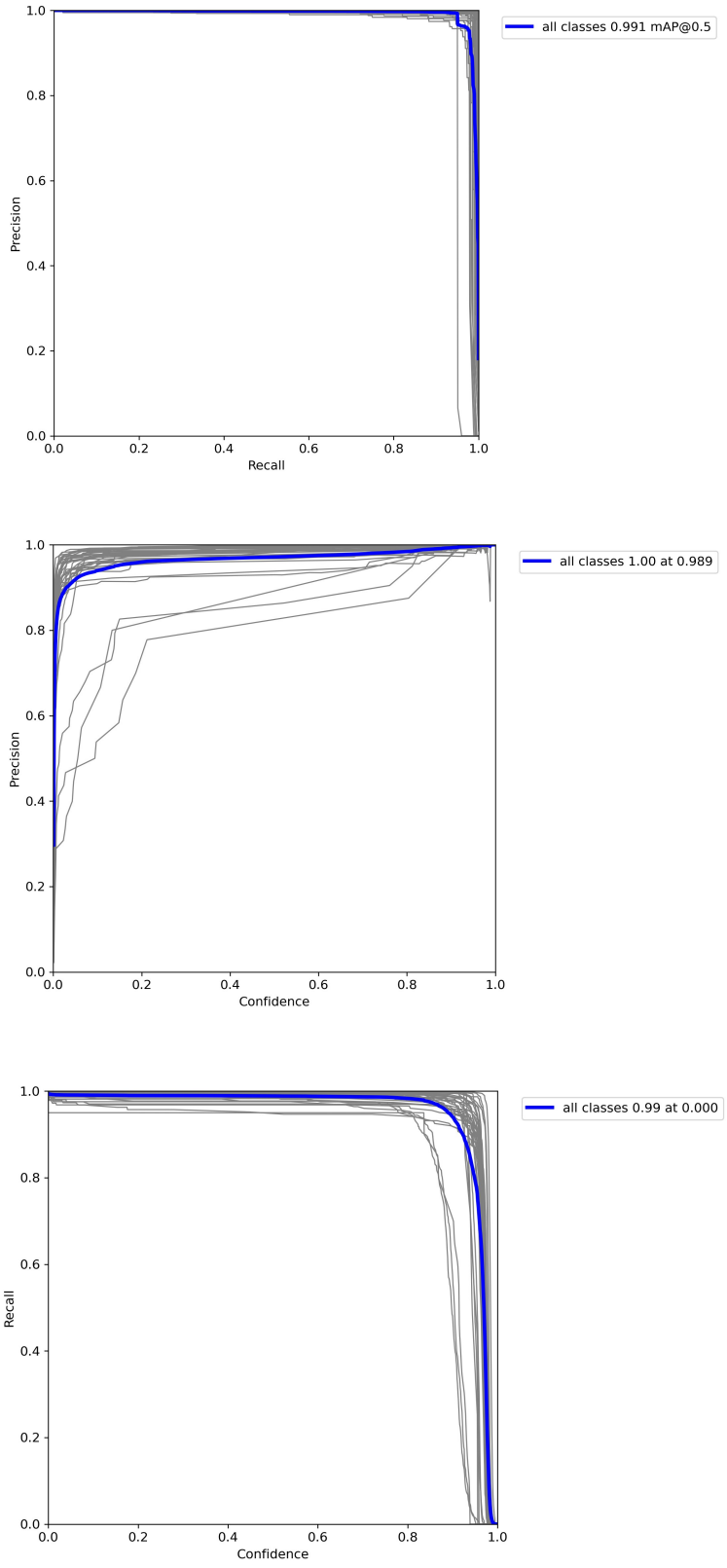

Fig. 15 - PR, P, R curves

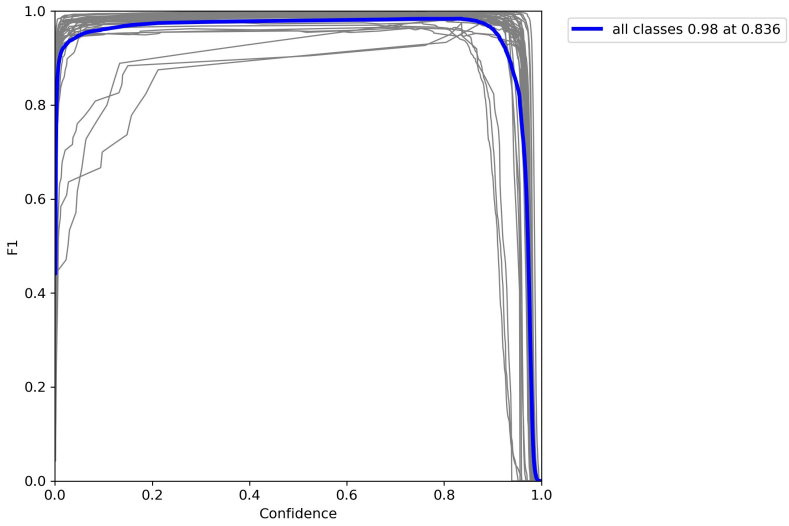

Fig. 16 - F1 curve

5 Model evaluation

According to charts and confusion matrix presented in previous section we can suspect, that models peforms very well, but it should be checked on real inference. Actually I should do it on test data that was not used to evaluate model during training, but then the training set might be not big enough. I did not use validation set to stop the training or tune hyperparameters anyway.

Model evaluation and metrics selection is done in Jupyter Notebook titled '3_Training_and_Evaluation.ipynb' – please refer to it for more details. Here is a brief summary of evaluation pipeline:

- Inference was run on validation set with confidence 0.5 command:
  python detect.py --weights best_0.pt --conf 0.5 --img-size 640 --source inference/images --save-txt
- Label txt files are read and compared with original data
- The model never returns the same class twice for one image – it is quite obvious, since training data never contained 2 classes at once, but it was good to confirm it.
- 'vis_pred' was calculated based on predicted coordinates (x_pred and y_pred).
- I compared labels from validation set with labels generated by model for inference and checked if classes are the same.
- But it does not indicate reliably if inference is made correctly. For example, the model could correctly predict that both points 10 and 11 are visible, but the bounding boxes were confused with each other. To avoid this, for each prediction made by model I checked if predicted x and y are withing range of 50 px near true value – then I marked it as 'corr_loc'= 1 from correct location. Then it is True Positive (TP). If both 'vis' and 'vis_pred'are 0, then it is True Negative (TN).
- If location test was failed ('corr_loc'= 0). I checked if prediction fits another point – again with 50 px margin (so the model confused prediction with other point) and all coordinates (pred and true) are higher than 0 – I called it class 'CONF' – very rare in practice, only 12 points.

- If location test was failed ('corr_loc'= 0) and predicted point has 'vis' = 0 in true data and 'vis_pred' = 2, then it is a False Positive (FP). If predicted point has 'vis' = 2 in true data and 'vis_pred' = 0, then it is a False Negative (FN).
- So the results for estimation of visibility can be interpreted as simple binary classification task. I chose metrics widely used for classification purpose – recall, precision, f1 and MCC.
- Additionally metrics for CONF class were introduced – CONF_R = CONF / (TP + FN + FP) – confused classes to all predictions and CONF_AM = CONF / (FN + FP) – share of confused classes among all mistakes.
- Estimation of the position for each class is a regression task, so I chose metrics widely used for such task – mean absolute error (MAE) and mean squared error (MSE).
- It can be approached in several ways:

  1. Soft – the model should not be penalized during regression for confusing classes, FN and FP. Therefore MAE and MSE calculation is made only for TP. - I would stick to soft MAE
  2. Medium – the model will be penalized for confusing classes. Therefore calculcation is made for TP and CONF
  3. Hard – the model is penalized for every mistake – TP, CONF, FP and FN are considered.

- For more details regard calculation, please refer to the Notebook - '3_Training_and_Evaluation.ipynb'

ACC	P	R	F1	MCC	CONF_R	CONF_AM
0.9906	0.9720	0.9904	0.9812	0.9750	0.176 %	4,78%

Tab. 1 - Average classification metrics for all classes

We can see, that in general the model performance is really good. Perhaps we can increase the confidence to higher value and reach higher precision (and lower recall), if higher precision is improtant if this case. I would say that FP and FN are similarily sever and balance between precision and recall should be reached.

	x	y	vis	kid	dataset	image_path	x_pred	y_pred	vis_pred	corr_loc	pred_kid
517	324	116	2	11	train	images/train/2abdbbdf85ec6b89bc387d79e361e4.jpg	253.0	125.0	2.0	0	8
8482	890	435	2	20	train	images/train/45e8fc9c955252393d432dcc3d3521.jpg	1435.0	356.0	2.0	0	0
8483	1451	364	2	21	train	images/train/45e8fc9c955252393d432dcc3d3521.jpg	868.0	421.0	2.0	0	0
8969	1315	705	2	39	train	images/train/17b60566817ca0274a139012630660.jpg	1261.0	691.0	2.0	0	5
10290	1689	375	2	34	train	images/train/d360acbbfa929fce1ca969f7bd3a5d.jpg	1748.0	388.0	2.0	0	2
14084	67	73	2	6	train	images/train/4d2a18fb109e930f4b34ef68500040.jpg	565.0	112.0	2.0	0	1
14085	568	114	2	7	train	images/train/4d2a18fb109e930f4b34ef68500040.jpg	367.0	208.0	2.0	0	0
14086	367	208	2	8	train	images/train/4d2a18fb109e930f4b34ef68500040.jpg	45.0	362.0	2.0	0	0
19008	708	531	2	16	train	images/train/ea74ceabee4807adafa7548c5380b6.jpg	716.0	370.0	2.0	0	0
19010	716	370	2	18	train	images/train/ea74ceabee4807adafa7548c5380b6.jpg	708.0	530.0	2.0	0	0
25033	1074	602	2	35	train	images/train/3713373b2f0a180f1bbb2758775dcf.jpg	1012.0	587.0	2.0	0	28
29368	339	980	2	2	train	images/train/7bbd1145d44589c5244893f962f46b.jpg	425.0	225.0	2.0	0	0

Tab. 2 - All found confusions

The models confuses classes with each other extremely rarely. It occured only 12 times – for 8 images. Some of these confusions did not fit to any points (pred_kid = 0) in table above.

	accuracy	precision	recall	f1	MCC	CONF_rate	CONF_per_F
1	0.986928	0.972727	0.981651	0.977169	0.968032	0.000000	0.000000
2	0.990838	0.986486	0.994550	0.990502	0.981683	0.002688	0.142857
3	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000	NaN
4	0.994771	0.972973	0.990826	0.981818	0.978820	0.000000	0.000000
5	0.983007	0.974874	0.960396	0.967581	0.956116	0.000000	0.000000
6	0.984293	0.966790	0.988679	0.977612	0.965661	0.003650	0.083333
7	0.988220	0.972000	0.991837	0.981818	0.973216	0.003968	0.111111
8	0.998691	1.000000	0.995050	0.997519	0.996635	0.004950	1.000000
9	0.992157	0.961832	0.992126	0.976744	0.972198	0.000000	0.000000
10	0.998693	1.000000	0.967742	0.983607	0.983069	0.000000	0.000000
11	0.989529	0.969697	0.981595	0.975610	0.968973	0.005952	0.125000
12	0.989542	0.974359	0.984456	0.979381	0.972400	0.000000	0.000000
13	0.996078	0.973913	1.000000	0.986784	0.984601	0.000000	0.000000
14	0.990850	0.929412	0.987500	0.957576	0.953003	0.000000	0.000000
15	0.979085	0.952381	0.962567	0.957447	0.943606	0.000000	0.000000
16	0.990838	0.984375	0.997361	0.990826	0.981760	0.002597	0.142857
17	0.993464	0.992167	0.994764	0.993464	0.986931	0.000000	0.000000
18	0.990838	0.984043	0.997305	0.990629	0.981751	0.002653	0.142857
19	0.989542	0.984064	0.984064	0.984064	0.976282	0.000000	0.000000
20	0.986911	0.978780	0.994609	0.986631	0.973932	0.002639	0.100000
21	0.988220	0.980926	0.994475	0.987654	0.976475	0.002710	0.111111
22	0.988235	0.972973	0.978261	0.975610	0.967864	0.000000	0.000000
23	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000	NaN
25	0.989542	0.953488	1.000000	0.976190	0.969947	0.000000	0.000000
26	0.984314	0.966102	0.982759	0.974359	0.963134	0.000000	0.000000
27	0.986928	0.959641	0.995349	0.977169	0.968333	0.000000	0.000000
28	0.992157	0.972067	0.994286	0.983051	0.978055	0.000000	0.000000
29	0.989542	0.933333	1.000000	0.965517	0.960156	0.000000	0.000000
30	0.997386	0.909091	1.000000	0.952381	0.952182	0.000000	0.000000
31	0.993464	0.966667	1.000000	0.983051	0.979220	0.000000	0.000000
32	0.992157	0.965116	1.000000	0.982249	0.977471	0.000000	0.000000
33	0.996078	0.972477	1.000000	0.986047	0.983895	0.000000	0.000000
34	0.988220	0.887500	1.000000	0.940397	0.935935	0.012500	0.111111
35	0.980366	0.934132	0.975000	0.954128	0.941995	0.005848	0.066667
39	0.990838	0.968889	1.000000	0.984199	0.977991	0.004444	0.142857

Tab. 3 - Metrics per class for classification

Based on metrics per class table above and charts below we can say, that general performance of the model is very solid, although precision falls for some classes – for point 30 it is lower probably becasue of labelling precision. Point 34 is the only one with precision lower than 0.9. Perhaps higher confidence value need to be applied, since recall is much higher than precision. Based on curves from training. Confidence of 0.8 should be best.

Note – rare classes had very low number of samples in validation set – only 4 for class 3 and 7 for class 23. We cannot trust there results too much.

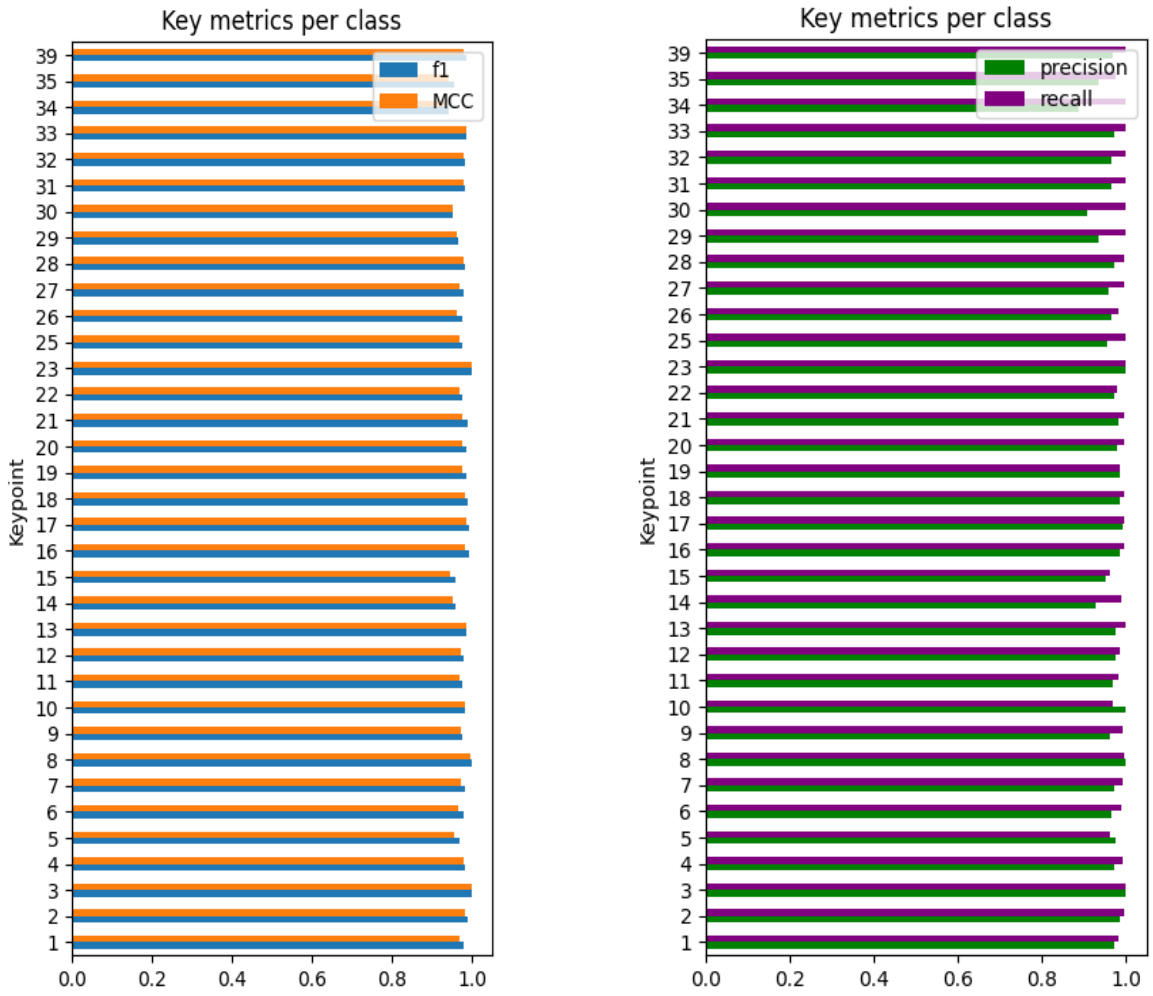

Fig - 17 Key classification metrics per class

MAE_soft	MAE_med	MAE_hard	MSE_soft	MSE_med	MSE_hard
2.87	3.41	43.70	3.44	4.06	53.03

Tab. 4 – Metrics for regression of coordinates

We can see that in general error related to regression of coordinates is quite low. I think that only soft version of these metrics should be used, so we evaluate only correctly assigned classes. In metrics per class table below we can see that MAE and MSE (soft) is worst for class 30, what is caused by insufficient precision of labelling. Worse than average performance on classes 19 and 39 is probably caused by the effect desrcibed in previous section. Worse performance of keypoint 4 was noticed during classification.

kid	MAE_soft	MAE_med	MAE_hard	MSE_soft	MSE_med	MSE_hard
1	2.473837	2.473837	28.771273	2.808411	2.808411	33.348214
2	1.814034	3.885258	18.232993	2.093151	4.385246	21.319035
3	3.398810	3.398810	3.398810	4.000000	4.000000	4.000000
4	8.115583	8.115583	60.080389	8.703704	8.703704	80.339286
5	2.511922	2.511922	35.395643	2.896907	2.896907	43.811594
6	2.178952	4.070000	43.073899	2.473282	4.505703	51.578182
7	2.042925	2.943955	21.774204	2.349794	3.549180	25.233202
8	2.285952	4.041622	4.689123	2.731343	5.074257	5.807882
9	2.644583	2.644583	34.145379	3.000000	3.000000	35.636364
10	3.753702	3.753702	20.736691	4.300000	4.300000	26.677419
11	1.879780	2.312627	30.309322	2.256250	2.739130	36.426036
12	2.190495	2.190495	33.968800	2.484211	2.484211	40.606061
13	2.110944	2.110944	19.405517	2.392857	2.392857	20.260870
14	2.839004	2.839004	50.246492	3.151899	3.151899	55.848837
15	2.426948	2.426948	82.319301	2.844444	2.844444	103.428571
16	1.467117	1.888572	22.900526	1.711640	2.153034	27.235751
17	1.614596	1.614596	18.554235	1.894737	1.894737	21.914286
18	1.494790	1.922566	20.630698	1.721622	2.169811	24.933862
19	8.588852	8.588852	28.963816	11.380567	11.380567	35.298039
20	3.284682	4.764172	26.057270	3.981030	5.656757	32.060526
21	3.163900	4.777795	48.147448	3.925000	5.686981	59.908108
22	2.452324	2.452324	58.819786	2.772222	2.772222	70.576720
23	6.150619	6.150619	6.150619	7.571429	7.571429	7.571429
25	2.898325	2.898325	55.884456	3.292683	3.292683	66.668605
26	1.896426	1.896426	62.348396	2.214912	2.214912	72.512500
27	2.228977	2.228977	65.563440	2.556075	2.556075	78.839286
28	3.113186	3.113186	57.677729	3.683908	3.683908	72.605556
29	1.979700	1.979700	116.765564	2.294643	2.294643	151.850000
30	10.463668	10.463668	123.261023	12.500000	12.500000	154.000000
31	2.383402	2.383402	65.362890	2.793103	2.793103	76.920000
32	2.424655	2.424655	54.852502	2.801205	2.801205	66.622093
33	3.192568	3.192568	50.712598	3.650943	3.650943	62.220183
34	3.070761	3.867212	209.140555	3.521127	4.472222	252.802469
35	2.435754	2.826537	114.689350	2.903846	3.375796	144.744186
39	8.877664	9.091854	61.239660	11.623853	11.881279	73.234513

Tab. 5 - Metrics for regression of coordinates per class

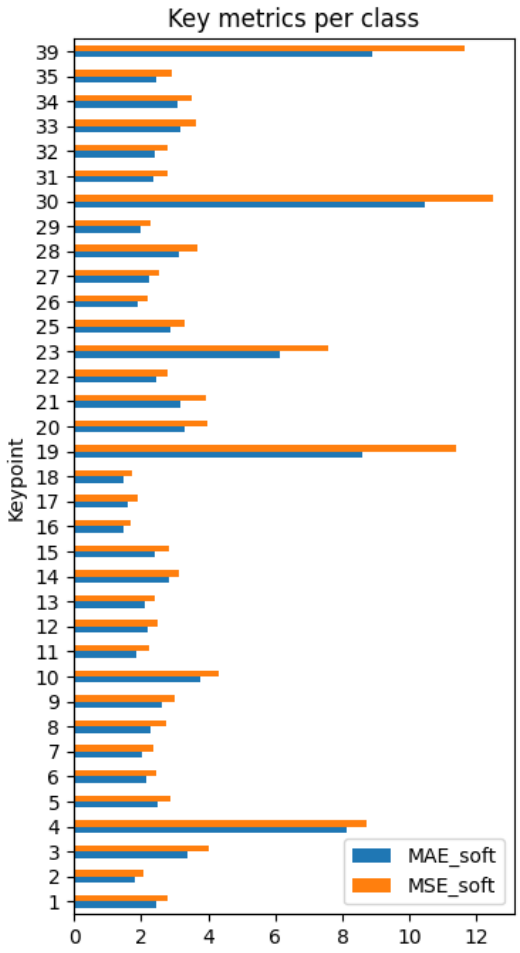

Fig. 17 - Key metrics for regression per class

6 *Inference on test data*

Inference on test data is done in Jupyter Notebook titled '0_Inference_on_test_data.ipynb'. File with filled coordinates is named df_keypoints_test_filled.csv

7 *Further ideas*

- Cross-validation would be extremely useful – e.g. 4 fold. I did not have capacity to to this.
- Train existing model further and check if val Objectness and val Classification losses eventually fall. Please refer to 'Additional experiments.pdf' file – there I saw premises, that all models might be underfitted. Of course if losses do not fall, we are actually overfitting.
- Train models with data augmentation and bigger boundary boxes futher (see Additional experiments.pdf) and check if results improve.
- If not – experiment with other kind of data augmentation – image rotation.

- **Manually label class 30**
- Do fine-tuning on best improving model (use metrics introduced in section 5 to determine best model) – this would require train-validation-test split and perhaps more data – properly augmented or gathered.
- Gather more data – especially more images with rare classes.
- Ensemble models that perform best for specific points on pitch.
- Introduce some kind of postprocessing – we can 'manually' validate potential confusions – for example keypoint 3 will always have y lower than keypoint 10.
- Apply a filter at start – we know that pitch is always green, eventually a little bit yellow. We can zero out whole blue and red channels and switch to grayscale image, where white is white, green is gray and everyhing else is black. Then we will have 3x less parameters – we can train longer or switch to more complex architecture.
- Switch to more complex model – 1280 px images – see YOLO documentation.
- Incorporate metrics defined in Section 5 in training pipeline – by YOLO side