1. Dataset: <https://drive.google.com/file/d/1zp3nXIYfq-YP_Pb631zPI2J150StWm4s/view?usp=sharing>

I hope you enjoy this quite interesting task that we have solved in one of our recent projects.

There are video frames captured by a camera over a table football. We have to detect a ball at every frame where it is present. As far as it is supposed to work at high FPS, you should do it not by a traditional object detection network, but by a lightweight segmentation CNN that will segment pixels of two classes: ball and background. During training, you should use albumentations library for all reasonable transformations.

At the output, you will simply get a mask ideally with a single blob (practically there can be a few blobs) that will indicate a ball position. At the end, you should just get coordinates of the bounding box (Bbox) of that blob.

At evaluation you should calculate precision and recall metrics for detection on the Test dataset using IoU criterion:

* if a detected BBox with a GT BBox has IoU >0.5 count TP+=1
* if a detected BBox with a GT BBox has IoU <0.5 count FP+=1
* if there are detected BBoxes but no GT BBox (frames without a ball) then count FP+=1
* if there are no detected BBoxes but there is GT BBox, then count FN+=1
* if there are no detected BBoxes and no GT BBox, then count TN+=1

At inference you should take a folder with frames and at the output save a video (for better perception make it with FPS=5 to get an effect of slowmo) with visualization of detected bounding boxes on those frames. Bounding box should be well visible, but with relatively thin edges that do not fully cover a ball.

**Deliverables for this task:**

* code that has traditional 3 functions (train, test/evaluate, inference on a folder with images)
* trained model
* document with metrics values
* recorded video with inference on the test dataset.

2) **BONUS**

You should train your own autoencoder that will denoise handwritten digits in the MNIST dataset. As an input, you should feed images corrupted by some well visible noise. At the output, you should get the denoised image. You can evaluate your previously trained classification network on noisy data and on denoised images.

**Deliverables for this task:**

* code that has functions for training and inference on a single image
* trained model
* optionally: metrics with classification CNN evaluation on the test set.