Current Solution

Step 1: Taking pictures

For a new dataset I decided to take pictures of vending machines and detect what is a snack(food) and what is a drink. Fortunately, taking pictures of the inside of a vending machine went without problems that occured with store drinks. It's mainly because everything inside the machines is restricted to it own place.

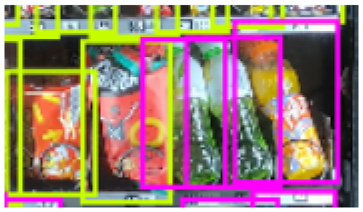
Example of a part of an image:

Step 2: Annotating

I just simply follow the same algorithm that I used in the unsuccessful solution. Just in this case everything is much simpler and I can everywhere use the box tool instead of complex.

Also it's important to note that in my dataset I have a vending machine (actually just a bar) with only drinks in it, so I made sure that this image is not the only image in validation or test dataset (so each class is represented)

Example of a part of an annotated image:



Step 3: Training a Faster RCNN

For this step I firstly watched the video from roboflow on how to train a model from detectron2 zoo. And just simply followed the steps.

However there are a few changes from the original jupiter notebook:

- I choosed appropriate model for the task
- Lowered number of iteration down to 1500 (still quite a big number)

Step 4: Training a YOLOv8

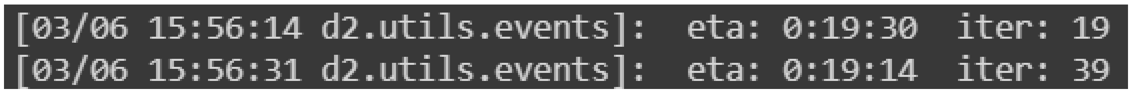
Here I just simply followed the instructions in the Jupiter notebook for YOLOv8 from roboflow.

The only problem is that the smallest YOLOV8 model (nano) just simply makes no predictions. There are no boxes with probabilities of a certain class.

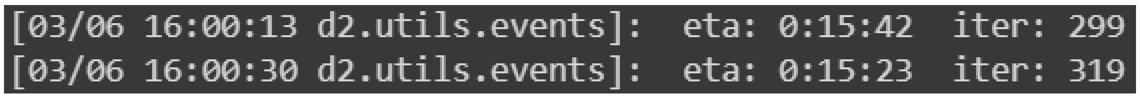
I have no idea why it's happening so I just simply follow the initial variant(from roboflow ipynb) - small size YOLOv8s

Step 5: Analysis

Training the Fatser RCNN model took ~20 minutes what is much more slower than the YOLOv8. However I have to say that RCNN accuracy reaches 0.9 value already at 300th iteration, so there is no actual need to wait that much.

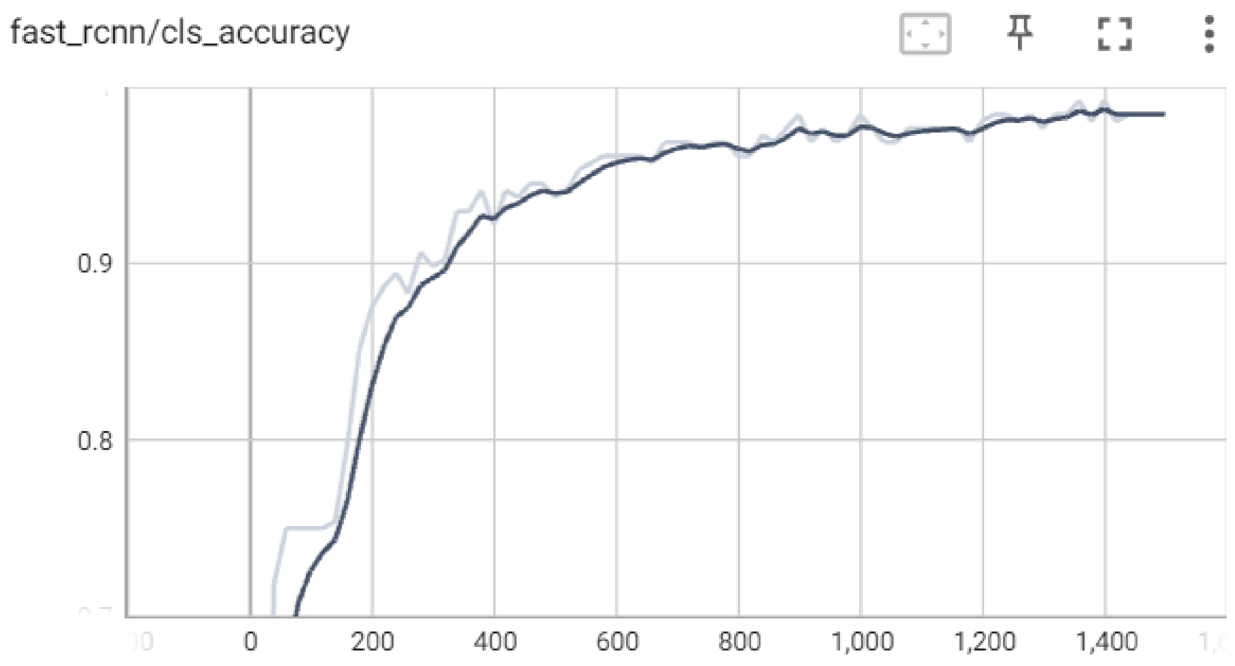


```
[03/06 15:56:14 d2.utils.events]:  eta: 0:19:30  iter: 19
[03/06 15:56:31 d2.utils.events]:  eta: 0:19:14  iter: 39
```

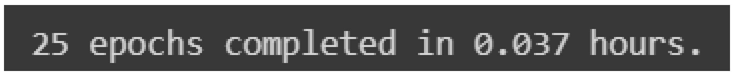


```
[03/06 16:00:13 d2.utils.events]:  eta: 0:15:42  iter: 299
[03/06 16:00:30 d2.utils.events]:  eta: 0:15:23  iter: 319
```

Judging by eta values (which is time estimated before the finish) it would took around 5 minutes to train the model to this level of accuracy.



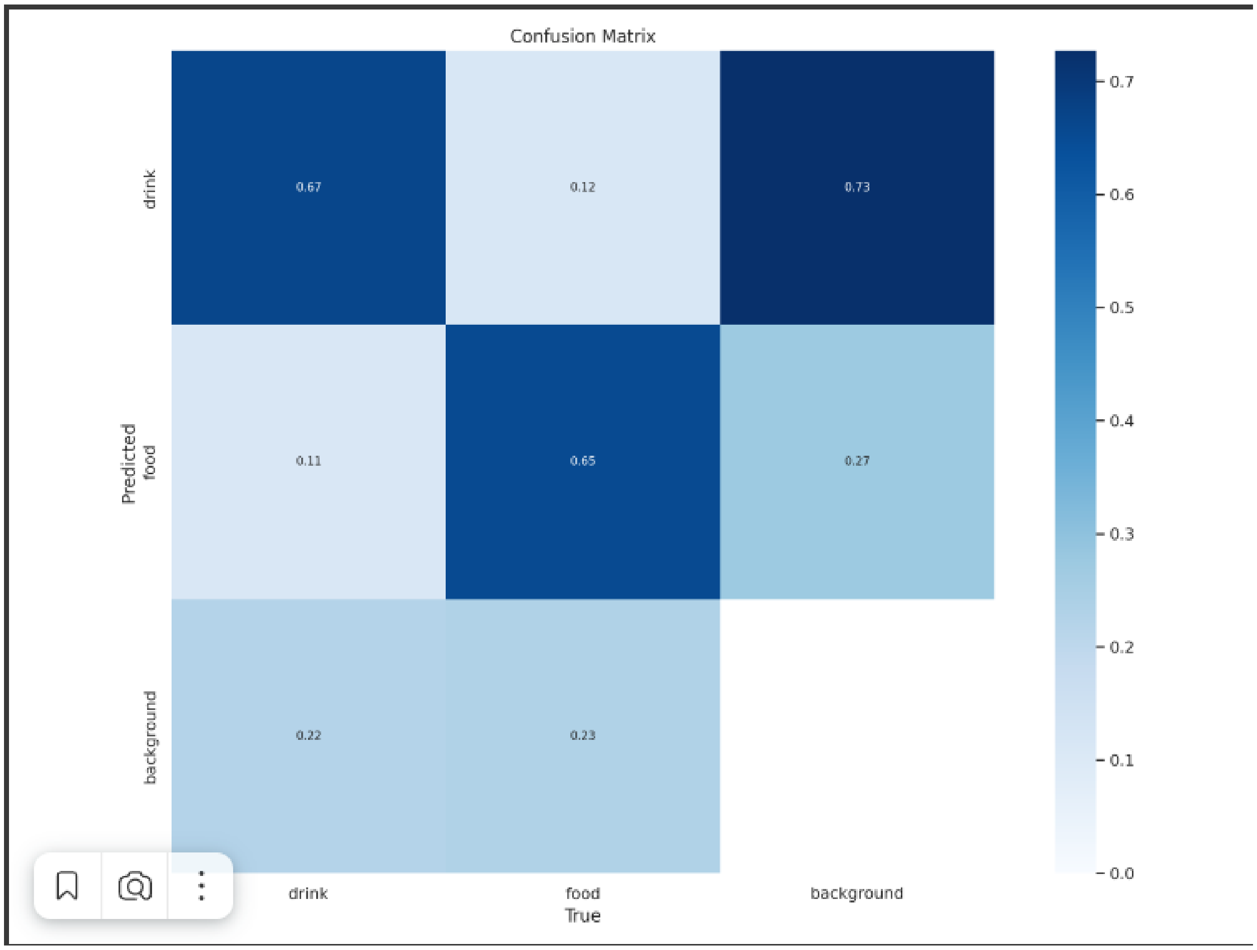
However It’s still slower than YOLOv8s. Which took only ~2 minutes to train on 25 epochs (0.037 hours = 2.22 minutes)

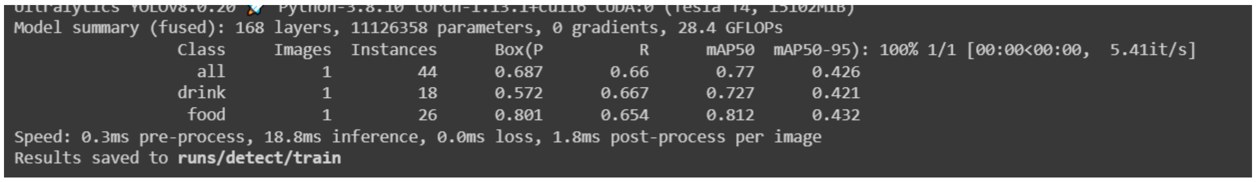


```
25 epochs completed in 0.037 hours.
```

This difference in training time and the size of models correspondingly affects the accuracy of this models.

Here is the confusion matrix and metrics for YOLOv8

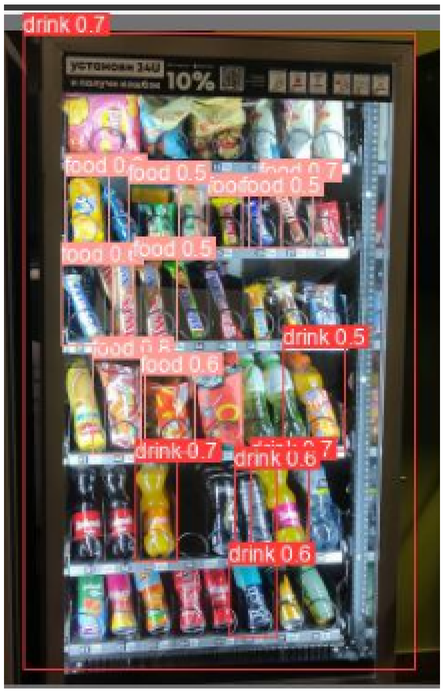




```
Ultralytics YOLOv8.0.20 🚀 Python-3.8.10 torch-1.13.1+cu116 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 11126358 parameters, 0 gradients, 28.4 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  5.41it/s]
                   all          1         44      0.687       0.66       0.77      0.426
                 drink          1         18      0.572      0.667      0.727      0.421
                  food          1         26      0.801      0.654      0.812      0.432
Speed: 0.3ms pre-process, 18.8ms inference, 0.0ms loss, 1.8ms post-process per image
Results saved to runs/detect/train
```

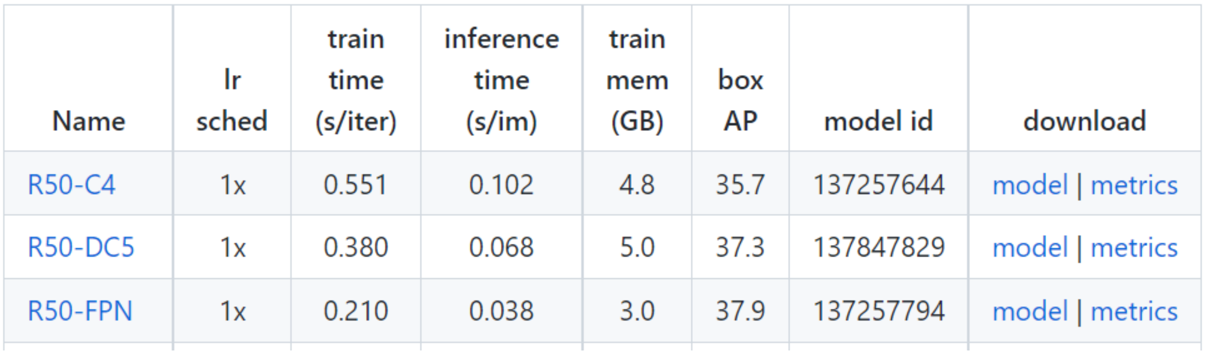
From them we can say that ~60% of the time it correctly identityes drinks and snacks, ~10% it mistakes it as the other type and ~20% of the time cannot find it. Interestingly enough there ae a lot of cases when it finds drink, but it actually a background. This because of the image of a bar with drinks - I did not marked all of the drinks in there because I thought that some of them are undetectable, but it seems like, that the model is able to do that.

But also sometimes it mistakes large rectangular objects as a drink (maybe because of the canned soda, that looks similary) like on this picture



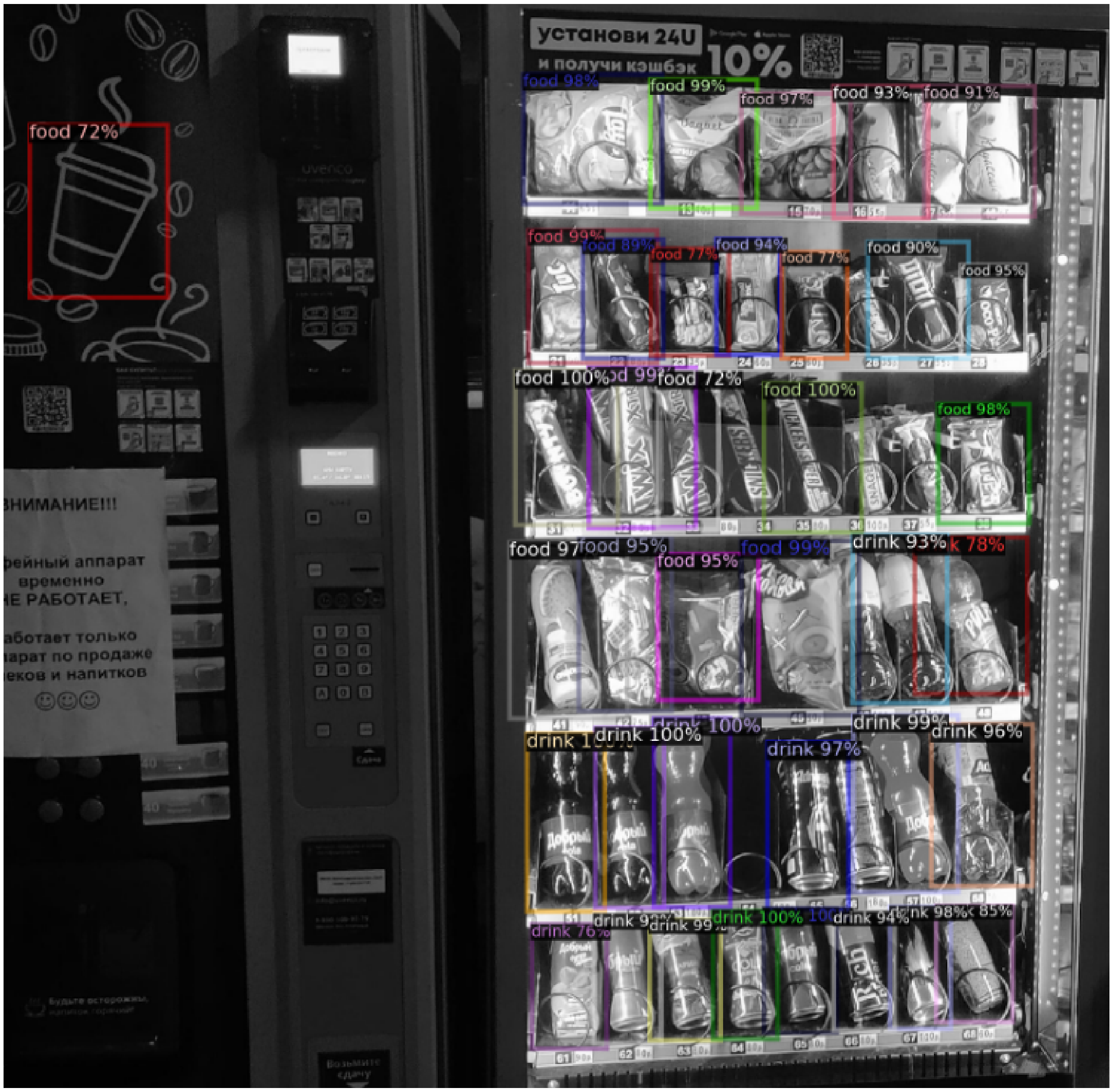
For Faster RCNN everything is better in terms of predictions. MOdel has higher probabilities for predictions (~98% in comparison to 0.5-0.7 for YOLO)

Except for the longer training part the model itself weight around 3 GB (or I hope so that the train mem mean). Anyway it should be more complex and weight more than YOLOv8 because of the time required and the difference in expected accuracy.



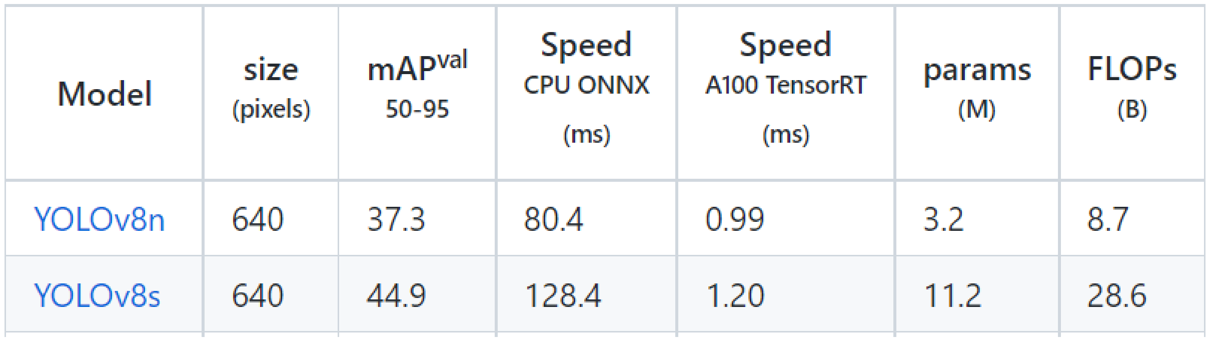
Name	lr sched	train time (s/iter)	inference time (s/im)	train mem (GB)	box AP	model id	download
R50-C4	1x	0.551	0.102	4.8	35.7	137257644	model | metrics
R50-DC5	1x	0.380	0.068	5.0	37.3	137847829	model | metrics
R50-FPN	1x	0.210	0.038	3.0	37.9	137257794	model | metrics

Just by looking at the result of Faster RCNN we can clearly see that it gets better results

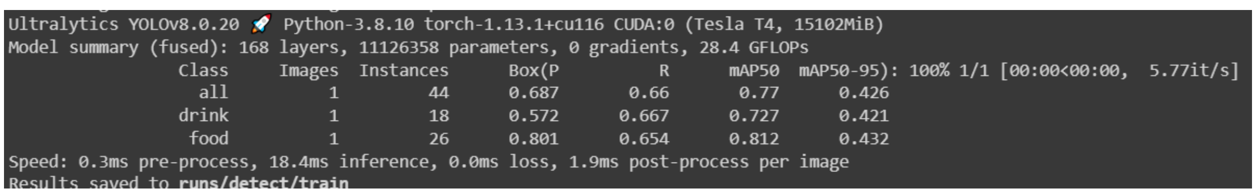


Yeah even with this small error with the image of a coffee cup. Model detects it not as a drink just because all drinks in my dataset represented as either bottles or aluminium cans, but the image is a rectangular simple shape as a bag of crisps. In any case the probability is pretty low - almost on the threshold.

In the defence of the YOLOv8s except for its fast time of training. Values for mAP that I've got are pretty close to the expected



Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6



```
Ultralytics YOLOv8.0.20 🚀 Python-3.8.10 torch-1.13.1+cu116 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 11126358 parameters, 0 gradients, 28.4 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  5.77it/s]
                   all          1         44      0.687       0.66       0.77      0.426
                 drink          1         18      0.572      0.667      0.727      0.421
                  food          1         26      0.801      0.654      0.812      0.432
Speed: 0.3ms pre-process, 18.4ms inference, 0.0ms loss, 1.9ms post-process per image
Results saved to runs/detect/train
```