Pose Bowl Detection Track (First Place Solution)

Ammar Ali Jaafar Mahmoud Aleksander Chigorin

1 Basic information for winner announcement

1. Name: Ammar Ali
   Hometown: Syria - Jableh
   Resident: Russia - Saint Petersburg
   UAE, Dubai

2. Name: Jaafar Mahmoud
   Hometown: Syria - Jableh
   Resident: Russia - Saint Petersburg

3. Name: Alexander Chigorin
   Hometown: Russian Federation
   Resident: UAE, Dubai, Town Square

2 Model documentation and write-up

2.1 Who are you (mini-bio) and what do you do professionally?

1. (Ammar) I am currently pursuing my PhD degree in Applied Computer Science at ITMO University, and I am in my third year of the program. In addition to my academic pursuits, I hold a senior researcher position at Polynome , where I focus on the development of multilingual large language models for rare languages, including Arabic. My expertise encompasses both computer vision and natural language processing.

2. (Jaafar) I am a researcher and developer specializing in robotic vision. I hold MSc in Intellectual Robotics and am expected to defend my PhD this year, focusing on robust localization. With around 5 years of professional experience, my expertise lies primarily in 3D perception, localization and mapping for Robots.

3. (Alexander) As the Research Director at VisionLabs UAE, I am responsible for overseeing multiple research initiatives. My primary focus is on advancing the core metrics of our projects, consistently achieving results

that are on par with or surpass the current state-of-the-art (SOTA). I actively work towards integrating these improvements into our products. My primary areas of research include object detection and human pose estimation.

2.2 What motivated you to compete in this challenge?

We are thrilled to participate in object detection competitions, as they provide a stimulating environment for us to showcase our skills and expertise. The recent competition we entered was particularly intriguing due to its unique time constraints and robustness challenge. It would be an immense honor for us if NASA were to incorporate our models into their space missions, as it would demonstrate the effectiveness and reliability of our technology in even the most extreme conditions.

2.3 High level summary of your approach: what did you do and why?

2.3.1 What ?

Our proposed solution primarily leverages Yolov8, with three distinct iterations (nano, small, and medium) that have been meticulously trained. Our approach employs a sequential ensemble strategy, taking into account two crucial factors:

1. The confidence threshold associated with the identified spaceship.
2. The unique condition of the competition data, which ensures that each image contains precisely one spaceship, neither more nor less.

Following the detection phase, we employ a refiner model (Yolov8m) that meticulously refines the detected object. This refiner has undergone rigorous training in two stages:

1. The initial stage involved training on approximately 300,000 synthetic images. The background for these images was generated using a diffusion model, while the spaceships was sourced from the provided metadata (nobackground images).
2. The second stage involved fine-tuning the refiner on the competition data with slightly less intense augmentations. This process ensures that the refiner model is optimized for the specific conditions of the competition.

2.3.2 Why ?

As we know that each image includes one spaceship we can do cascade ensemble based on the existence of the object. Hence the Nano model is not so accurate we decided to filter also on the confidence. The main goal of the detector is to detect the region of the object even if the mAP is low. After detecting the

area and cropping it we pass it to the refiner. It is very important the refiner is generalized and trained on wild number of different spaceships (the training set includes about 15 different space ships which is not enough). Therefore the generalization and robustness needed more data. The refiner doesn't need to be train on a high resolutions which allow us to fit yolov8m after any detector.
NOTE: we have split the data using StratifiedKFold based on the object bounding box area. After tuning the hyper-parameters we trained on the full dataset all experiments.

3 Do you have any useful charts, graphs, or visualizations from the process?

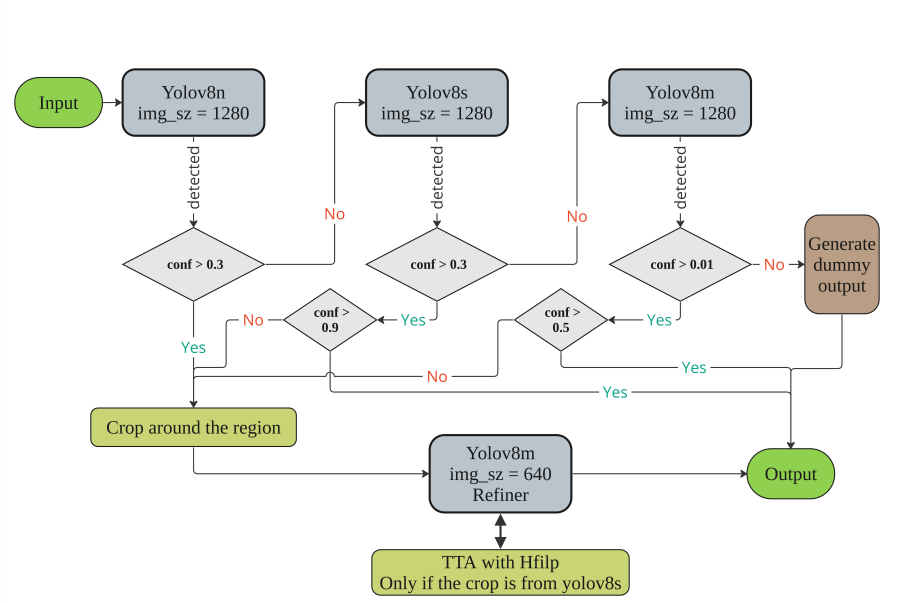

Figure 1: The main inference pipeline

Fig1 shows how we do sequential ensemble, please note the all detection models where quantized to int8 to accelerate inference, but the refiner used without quantization.

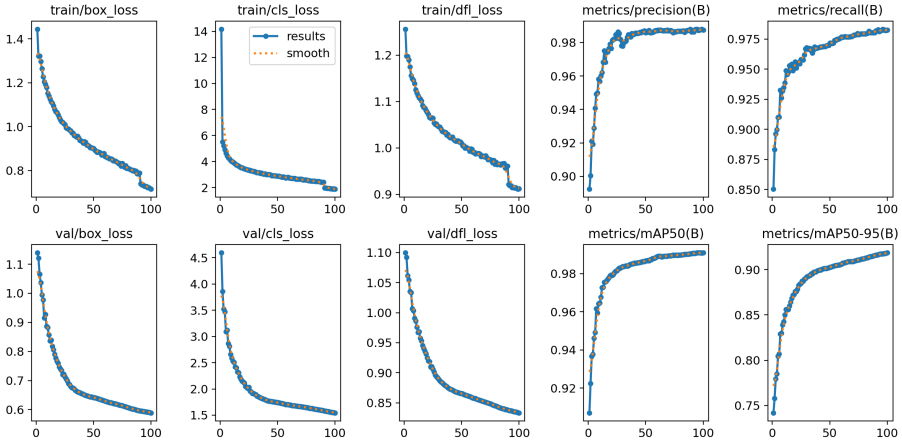

Figure 2: Training metrics for the Nano detector

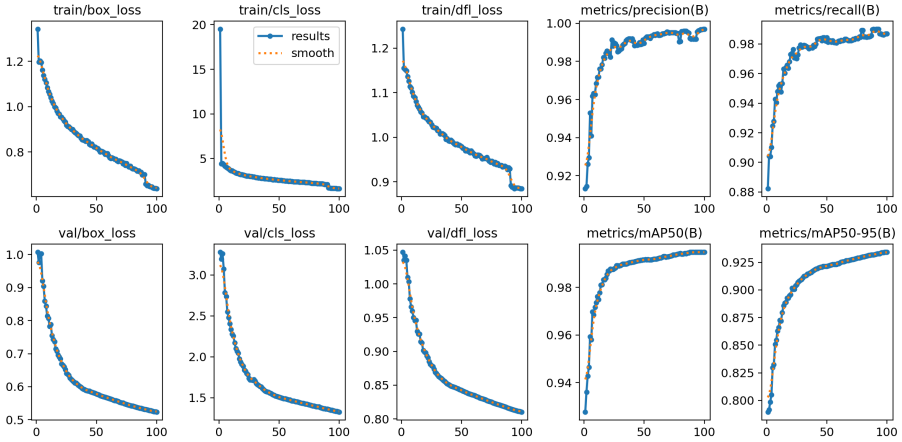

Figure 3: Training metrics for the small detector

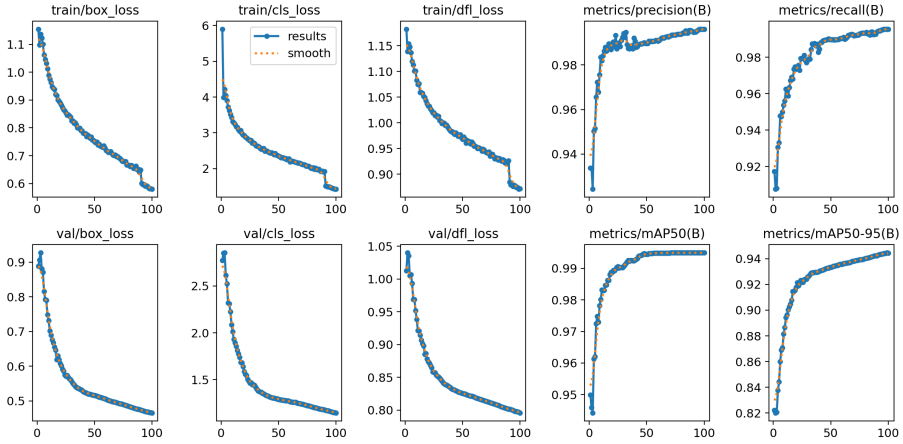

Figure 4: Training metrics for the medium, detector

Fig2, 3 and 4 show the metrics and loss changes during the training, all detectors where trained for 100 epochs, for other hyper-parameters refer to the code.

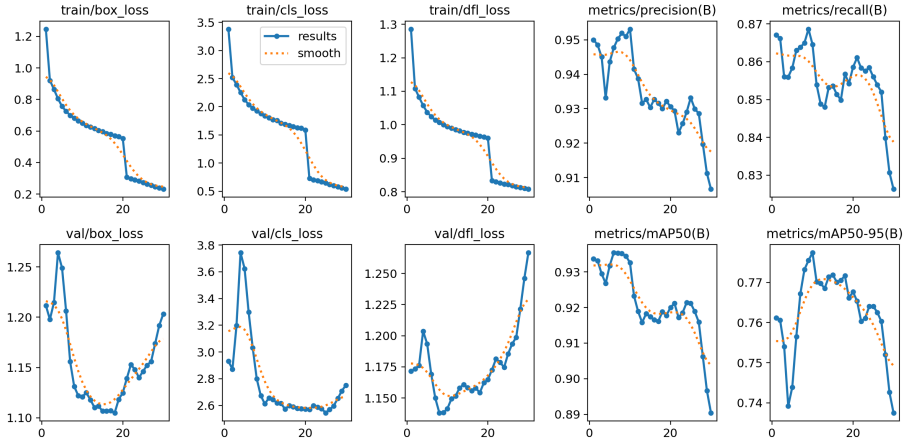

Figure 5: Training metrics for the refiner 1 stage while validating on the competition data

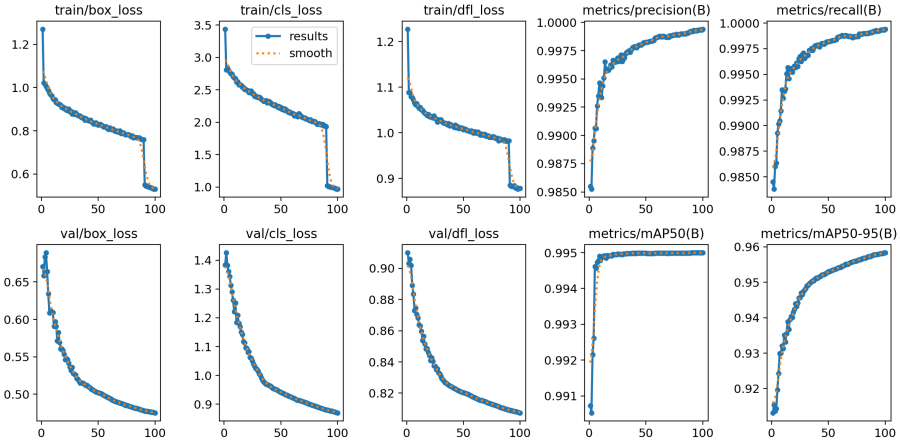

Figure 6: Training metrics for the refiner second stage

Figs 5 and 6, show the metrics and loss changes while training the refiner model.

Figure 7: Example of synthetic backgrounds

The data where generated using Kandinsky 3 [**?**].

4 Copy and paste the 3 most impactful parts of your code and explain what each does and how it helped your model.

```
prompts = [
    "Outter space with a part of the Earth visible",
    "Outter space with a very small part of the Earth visible",
    "Outter space with a part of the Mars visible",
    "Outter space with a very small part of the Mars visible",
    "Outter space with a part of the moon visible",
    "Outter space with a very small part of the moon visible",
```

```
    "Outter space",
    "Outter space with sunshine",
    "Outter space with Earth and moon visible and sunshine",
    "Outter space with Earth and moon visible",
    "Outter space with Earth visible and clouds",
    "Outter space with Earth visible and clouds top view",
    "Outter space with Earth visible and forests satallite image",
    "satallite image from the space for a country",
    "satallite image from the space for a country with clouds",
    "satallite image from the space for an ocean",
    "satallite image from the space for an ocean with clouds",
    "Outer space with Earth partially obscured by the curvature of a planet. satallite image
    "Outer space with a sliver of Earth peeking through the darkness. satallite image",
    "Outer space with Mars dominating the foreground, dwarfing Earth in the distance. satall
    "Outer space with a tiny glimpse of Mars in the vast expanse. satallite image",
    "Outer space with the moon prominently featured against the backdrop of stars. satallite
    "Outer space with a minuscule portion of the moon barely visible. satallite image",
    "An expansive view of outer space, devoid of any celestial bodies. satallite image",
    "Outer space illuminated by the warm glow of a distant star. satallite image",
    "Outer space with both Earth and the moon in the frame, basking in sunlight. satallite i
    "Outer space with Earth and the moon captured in perfect alignment. satallite image",
    "Outer space with Earth visible, adorned by wispy clouds. satallite image",
    "Outer space showcasing Earth from a bird's-eye view, surrounded by clouds. satallite im
    "Outer space capturing Earth's surface, showcasing lush forests and sprawling landscapes
    "A satellite image captured from outer space, focusing on a specific country's terrain a
]
```

The prompts we wrote to generate the background data.

```
a1 = conf/(conf + conf2)
a2 = conf2/(conf + conf2)
b1 = bbox
b2 = bbox2
newx1 = a1*b1[0] + a2*b2[0]
newy1 = a1*b1[1] + a2*b2[1]
newx2 = a1*b1[2] + a2*b2[2]
newy2 = a1*b1[3] + a2*b2[3]
# print(bbox)
bbox = [newx1, newy1, newx2, newy2]
```

Weighted box fusion for the refiner after the yolo small detector. We generated two boxes using TTA (image + HFlip). and merged then with the code above.

```
if len(result.boxes) > 0 :
    init_bbox = result.boxes.xyxy[0].tolist()
    anybox = init_bbox.copy()
    scale = 0.75
```

```
        init_bbox = expand_region(init_bbox, w, h, scale=scale)
        new_img = img[init_bbox[1]:init_bbox[3],init_bbox[0]:init_bbox[2],:]
        result2 = model2(new_img, verbose=False, imgsz= 640, conf=0.1)[0]
        if len(result2.boxes) > 0:
            bbox = result2.boxes.xyxy[0].tolist()
            bbox[0] += init_bbox[0]
            bbox[1] += init_bbox[1]
            bbox[2] += init_bbox[0]
            bbox[3] += init_bbox[1]
        else:
            flag = True

    else:
        flag = True

    if flag:
        flag = False
        result = model4(img, verbose=False, imgsz= 1280, conf=0.3)[0]
    # # get bbox coordinates if they exist, otherwise just get a generic box in center of an ima
        if len(result.boxes) > 0:
            init_bbox = result.boxes.xyxy[0].tolist()
            anybox = init_bbox.copy()
            if result.boxes.conf[0] > 0.9:
                bbox = init_bbox.copy()
            else:
                scale = 0.75
                init_bbox = expand_region(init_bbox, w, h)
                new_img = img[init_bbox[1]:init_bbox[3],init_bbox[0]:init_bbox[2],:]
                result2 = model2(new_img, verbose=False, imgsz= 640, conf=0.1)[0]
                result3 = model2(cv2.flip(new_img, 1), verbose=False, imgsz= 640, conf=0.1)[0]
                if len(result2.boxes) > 0:
                    bbox = result2.boxes.xyxy[0].tolist()
                    conf = result2.boxes.conf[0]
                    if len(result3.boxes) > 0:
                        bbox2 = result3.boxes.xyxy[0].tolist()
                        conf2 = result3.boxes.conf[0]
                        temp = bbox2[0]
                        bbox2[0] = new_img.shape[1] - bbox2[2]
                        bbox2[2] = new_img.shape[1] - temp
                    else:
                        bbox2 = bbox
                        conf2 = conf
                    a1 = conf/(conf + conf2)
                    a2 = conf2/(conf + conf2)
                    b1 = bbox
                    b2 = bbox2
```

```
                newx1 = a1*b1[0] + a2*b2[0]
                newy1 = a1*b1[1] + a2*b2[1]
                newx2 = a1*b1[2] + a2*b2[2]
                newy2 = a1*b1[3] + a2*b2[3]
                # print(bbox)
                bbox = [newx1, newy1, newx2, newy2]
            # if len(result2.boxes) > 0:
            #     bbox = result2.boxes.xyxy[0].tolist()
                bbox[0] += init_bbox[0]
                bbox[1] += init_bbox[1]
                bbox[2] += init_bbox[0]
                bbox[3] += init_bbox[1]
            else:
                flag = True

    else:
        flag = True

if flag:
    result = model3(img, verbose=False, imgsz= 1280, conf=0.01)[0]
    if len(result.boxes) > 0:
        # bbox = result.boxes.xyxy[0].tolist()
        init_bbox = result.boxes.xyxy[0].tolist()
        #anybox = init_bbox.copy()
        if result.boxes.conf[0] > 0.5:
            bbox = init_bbox.copy()
        else:
            init_bbox = expand_region(init_bbox, w, h)
            new_img = img[init_bbox[1]:init_bbox[3],init_bbox[0]:init_bbox[2],:]
            result = model2(new_img, verbose=False, imgsz= 640, conf=0.1)[0]
            if len(result.boxes) > 0:
                bbox = result.boxes.xyxy[0].tolist()
                bbox[0] += init_bbox[0]
                bbox[1] += init_bbox[1]
                bbox[2] += init_bbox[0]
                bbox[3] += init_bbox[1]
            else:
                bbox = init_bbox
```

The sequential ensemble between models helped to get almost same performance as if we used yolov8m + refiner for all images which requires x3 time more

5 Please provide the machine specs and time you used to run your model.

Specifications: Note that you can train all models on V100 except the medium model with resolution 1280. you will need to user lower batch size to fit in in RTX 3090 and therefore to change the hyper-parameters. It is better to use A100 or A6000 to reproduce the results for this model.

1. CPU: AMD EPYC 7742 64-Core Processor (8 workers used only)
2. GPU: NVIDIA A100-SXM4, VRAM 40 GB
3. Memory (RAM): 128 GB
4. OS: Linux
5. Docker: skypirate91/minigpt4:0.4

Training Duration:

1. yolov8 nano image-size 1280: 4.5 minutes for a single epoch training and 0.5 minute for validation in total 500 minutes.
2. yolov8 small image-size 1280: 5.5 minutes for a single epoch training and 0.5 minute for validation in total 600 minutes.
3. yolov8 medium image-size 1280: 9.2 minutes for a single epoch training and 0.8 minute for validation in total 1000 minutes.
4. yolov8 medium image-size 640 refiner stage1: 36 minutes for a single epoch training and 1.5 minute for validation in total 1125 minutes (30 epochs).
5. yolov8 medium image-size 640 refiner stage2: 3 minutes for a single epoch training and 1.5 minute for validation in total 450 minutes (100 epochs).

Inference Duration:
It took about 1 hour and 41 minutes on driven data env and specifications.

6 Anything we should watch out for or be aware of in using your model (e.g. code quirks, memory requirements, numerical stability issues, etc.)?

No, if you use the same docker container/ devices and libraries version Then the solution should be completely reproducible.

7 Did you use any tools for data preparation or exploratory data analysis that aren't listed in your code submission?

No

8 How did you evaluate performance of the model other than the provided metric, if at all?

The model was evaluated using the provided metric, mAP@0.5 and mAP@0.5:0.95.

9 What are some other things you tried that didn't necessarily make it into the final workflow (quick overview)?

1. Despite the slightly lower validation results, Yolov9C was utilized as a refiner. However, due to these results, it was not submitted.
2. A comprehensive evaluation of various versions of Yolo, ranging from Yolov5 to Yolov9, was conducted. The results indicated that Yolov8 demonstrated the most optimal performance when used as a detector.
3. A modified Yolo architecture, combining elements of Yolov8 and Yolov9, was developed and trained from scratch on the COCO dataset, followed by tuning on the competition data. Although the resulting model exhibited superior performance, it was not submitted due to its slower processing speed.
4. DeepSparse did not prove to be effective on the our CPUs, So it was not submitted.

10 If you were to continue working on this problem for the next year, what methods or techniques might you try in order to build on your work so far? Are there other fields or features you felt would have been very helpful to have?

1. Proceed with the development of the innovative architecture, which is a fusion of YOLOv9 and v8.

2. Explore the potential of RT-DETR in handling this task.

3. Devote more time to the generation of high quality and more realistic synthetic data.

11 What simplifications could be made to run your solution faster without sacrificing significant accuracy?

1. In order to optimize the performance and reduce memory consumption, it is recommended to quantize the refiner model to int8 precision.

2. It is worth considering the removal of small and medium detectors from the pipeline, as this modification will result in only a minimal decrease of approximately 1-2% in the metric, while significantly improving the pipeline's speed, nearly doubling it.

3. Implementing the deepsparse library for all components of the pipeline could potentially lead to even better results and improved performance.

https://arxiv.org/abs/2312.03511