Notes, links and information

For this assignment, I choose the inference option. Since inference is not that resource intensive, I did all of my work locally in vscode. All of my code for this homework can be found here.

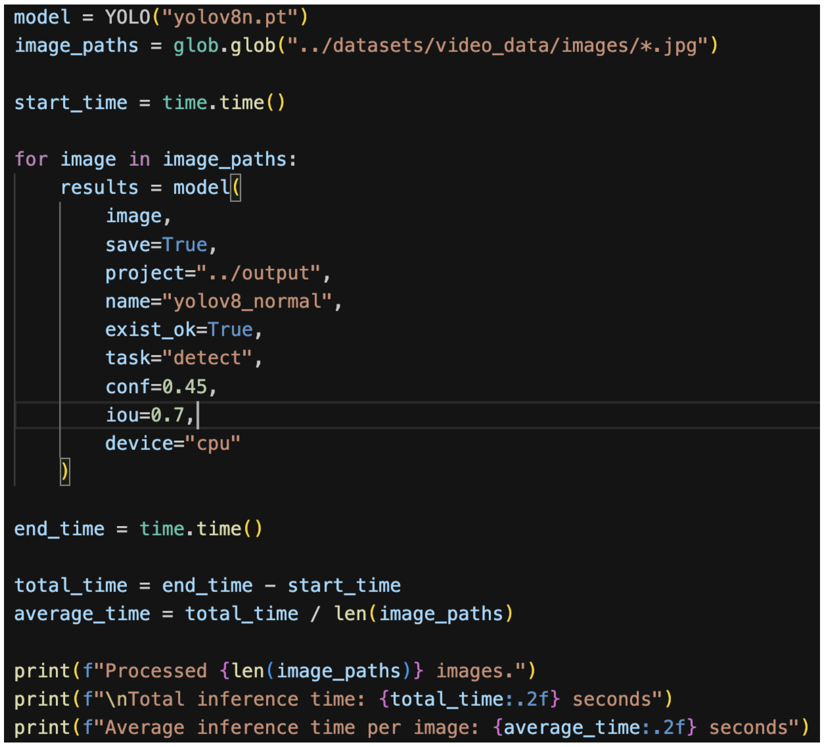
I did inference on yolov8 and fcos_resnet50_fpn. For yolov8, I tried inference using pytorch and for optimization, I chose onnx and openvino. As for fcos_resnet50_fpn, I did inference using pytorch and onnx. I will be comparing optimization vs no optimization, onnx vs openvino, and yolov8 vs fcos_resnet50_fpn. At first, I only looked at the speed difference but later on I realized I can also look at how good the model is via mAP50-95.

Coding, what I did, and analysis

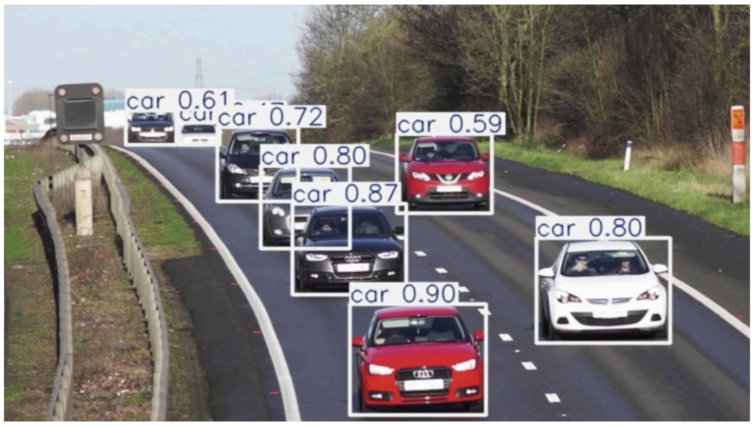
To start, I thought we needed to use the reference code so I used the first reference link to get some data to do inference with. I got a 1 minute traffic video here and used the reference code to extract 61 images. As for the other reference code, it was too confusing and I don't know how to make use of it so I opted to just code everything myself or find sample code elsewhere.

Yolov8 - default pytorch

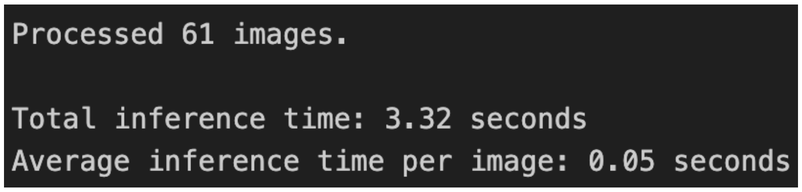
The first model I choose to do inference on is the yolov8 model. I chose this because we were learning about it in class. I found from the ultralytic website an example on how to use create and run inference on a yolo model.



```python
model = YOLO("yolov8n.pt")
image_paths = glob.glob("../datasets/video_data/images/*.jpg")

start_time = time.time()

for image in image_paths:
    results = model(
        image,
        save=True,
        project="../output",
        name="yolov8_normal",
        exist_ok=True,
        task="detect",
        conf=0.45,
        iou=0.7,
        device="cpu"
    )

end_time = time.time()

total_time = end_time - start_time
average_time = total_time / len(image_paths)

print(f"Processed {len(image_paths)} images.")
print(f"\nTotal inference time: {total_time:.2f} seconds")
print(f"Average inference time per image: {average_time:.2f} seconds")
```



One of the image outputs, I won't show anymore output as it's pretty much the same for all the inference that I did for this homework. Anyways, I just want to add that this is pretty cool what we can do with these detection models. It looks like magic.

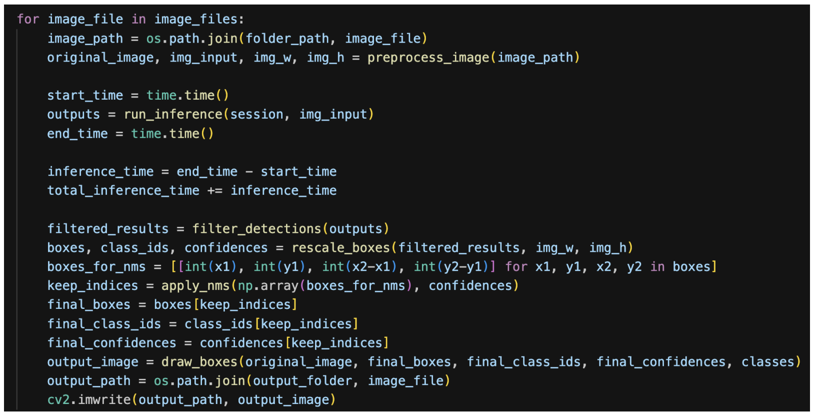


```
Processed 61 images.

Total inference time: 3.32 seconds
Average inference time per image: 0.05 seconds
```

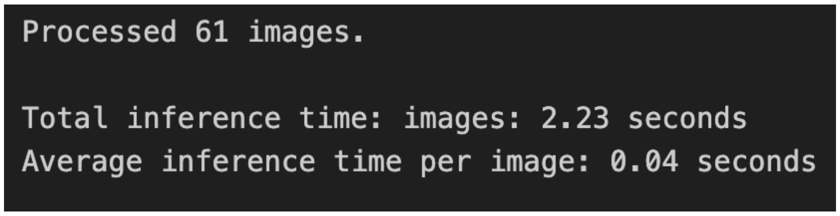
After modifying it to infer over my extract images folder, and adding some time tracking, I ran the code. From the output I was pretty surprised how fast it ran since it was 61 images and it only took 3.3 seconds to process all of them.

Yolov8 - onnx

After this, I tried to edit this code so it uses onnx to infer instead of the default pytorch. It did not work as I did not know what I was doing so I found a code someone did that uses onnx to infer with pytorch here. After fixing up the code to match my needs and incorporating more efficient code I found from the ultralytic github, I was able to get a working code that uses onnx.



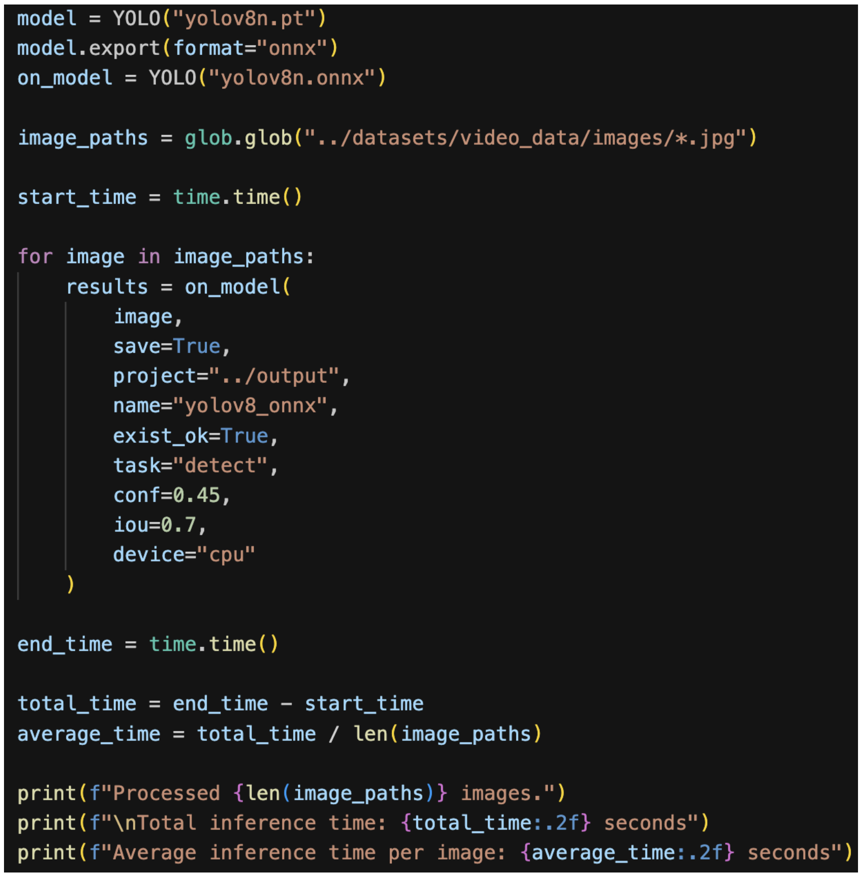
```
for image_file in image_files:
    image_path = os.path.join(folder_path, image_file)
    original_image, img_input, img_w, img_h = preprocess_image(image_path)

    start_time = time.time()
    outputs = run_inference(session, img_input)
    end_time = time.time()

    inference_time = end_time - start_time
    total_inference_time += inference_time

    filtered_results = filter_detections(outputs)
    boxes, class_ids, confidences = rescale_boxes(filtered_results, img_w, img_h)
    boxes_for_nms = [[int(x1), int(y1), int(x2-x1), int(y2-y1)] for x1, y1, x2, y2 in boxes]
    keep_indices = apply_nms(np.array(boxes_for_nms), confidences)
    final_boxes = boxes[keep_indices]
    final_class_ids = class_ids[keep_indices]
    final_confidences = confidences[keep_indices]
    output_image = draw_boxes(original_image, final_boxes, final_class_ids, final_confidences, classes)
    output_path = os.path.join(output_folder, image_file)
    cv2.imwrite(output_path, output_image)
```



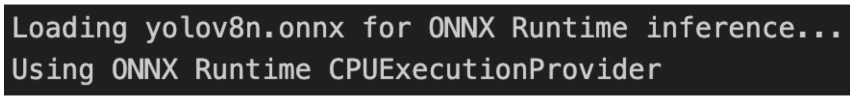
```
Processed 61 images.

Total inference time: images: 2.23 seconds
Average inference time per image: 0.04 seconds
```

From running and looking at the output, I can see that there is a difference with using onnx. The time to process each image went from 0.05 to 0.04. The dataset is kind of small so the difference is not that noticeable.

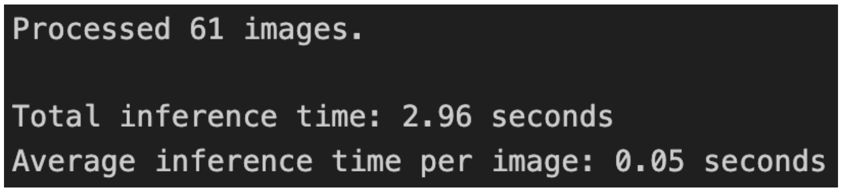
Later on while doing openvino optimization, I found out that ultralytic have methods that can detect supported model export formats and automatically use the optimization of that format internally. Because of this discovery, I redid the code for onnx and ran it through the dataset again.



```python
model = YOLO("yolov8n.pt")
model.export(format="onnx")
on_model = YOLO("yolov8n.onnx")

image_paths = glob.glob("../datasets/video_data/images/*.jpg")

start_time = time.time()

for image in image_paths:
    results = on_model(
        image,
        save=True,
        project="../output",
        name="yolov8_onnx",
        exist_ok=True,
        task="detect",
        conf=0.45,
        iou=0.7,
        device="cpu"
    )

end_time = time.time()

total_time = end_time - start_time
average_time = total_time / len(image_paths)

print(f"Processed {len(image_paths)} images.")
print(f"\nTotal inference time: {total_time:.2f} seconds")
print(f"Average inference time per image: {average_time:.2f} seconds")
```



```
Loading yolov8n.onnx for ONNX Runtime inference...
Using ONNX Runtime CPUExecutionProvider
```

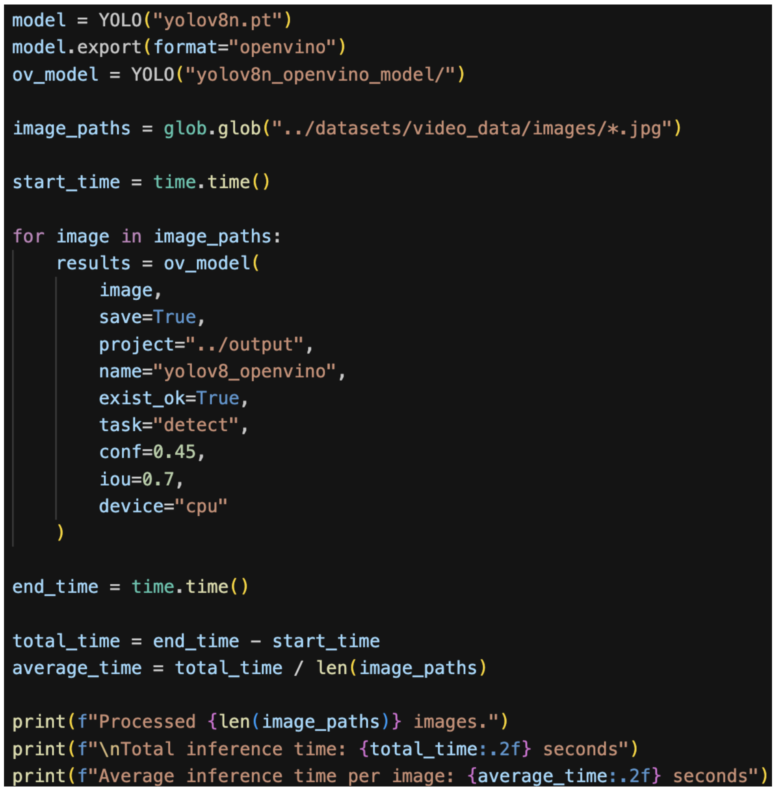


```
Processed 61 images.

Total inference time: 2.96 seconds
Average inference time per image: 0.05 seconds
```

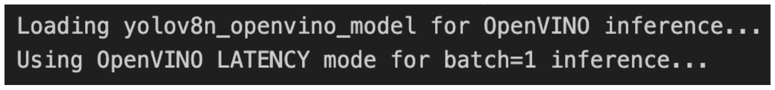
I ran this code and the first onnx code multiple times and concluded that the time is pretty similar. But this new one is better as it uses an already existing onnx implementation from ultralytic that the first one was also basically using but processes the image without it anyways.

Yolov8 - openvino

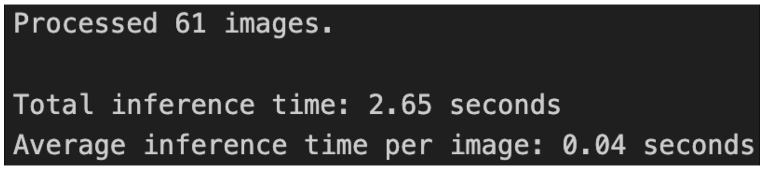
After doing the first implementation of onnx for yolov8, I moved on to the second optimization. Since I did onnx and openvino for the last homework, I decided to try to do a different inference optimizer. I tried to do torchcript and that wasn't working, then I tried to do tensorRT and that went nowhere too. Finally, I found a tutorial on the ultralytic website on how to use openvino to optimize yolo inference. Ultralytic has some supported model format that the library can detect and automatically use that optimizer internally when you infer. Since onnx is supported, I also went back and used this functionality to implement onnx on the yolov8 model.



```
model = YOLO("yolov8n.pt")
model.export(format="openvino")
ov_model = YOLO("yolov8n_openvino_model/")

image_paths = glob.glob("../datasets/video_data/images/*.jpg")

start_time = time.time()

for image in image_paths:
    results = ov_model(
        image,
        save=True,
        project="../output",
        name="yolov8_openvino",
        exist_ok=True,
        task="detect",
        conf=0.45,
        iou=0.7,
        device="cpu"
    )

end_time = time.time()

total_time = end_time - start_time
average_time = total_time / len(image_paths)

print(f"Processed {len(image_paths)} images.")
print(f"\nTotal inference time: {total_time:.2f} seconds")
print(f"Average inference time per image: {average_time:.2f} seconds")
```



```
Loading yolov8n_openvino_model for OpenVINO inference...
Using OpenVINO LATENCY mode for batch=1 inference...
```

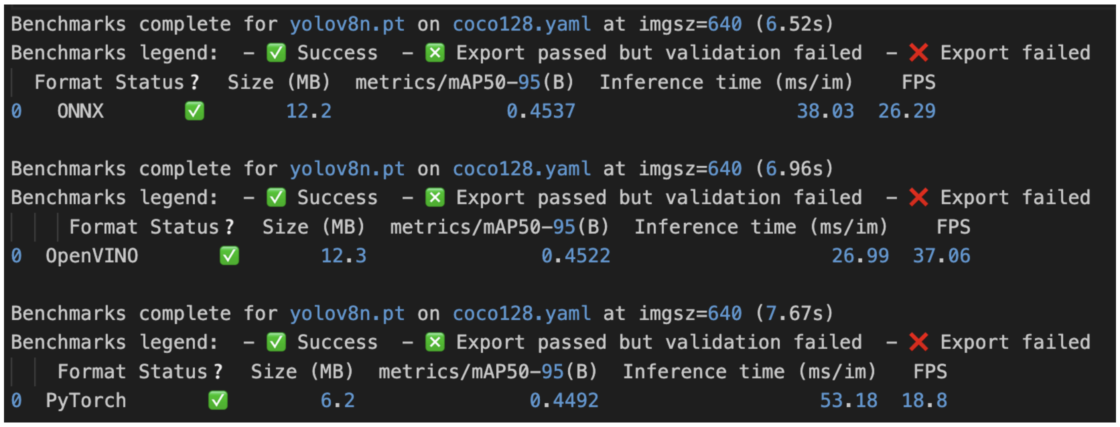


```
Processed 61 images.

Total inference time: 2.65 seconds
Average inference time per image: 0.04 seconds
```

After doing inference with openvino and looking at the output, I can see that it is faster than the default pytorch. I ran it multiple times and it consistently finished processing earlier than that of onnx.

More comparison on yolov8

After one of the lectures, someone asks the professor if we should compare the inference on its accuracy or on how well it can detect objects. His answer was not really clear but I will do it anyway. I found that the ultralytic library has a function call benchmark that will benchmark on your exported yolo model. It's pretty convenient as they have datasets that the function can automatically download and do the benchmark on it. Anyways, I proceeded to use this function on all three inferences: onnx, openvino, and pytorch.

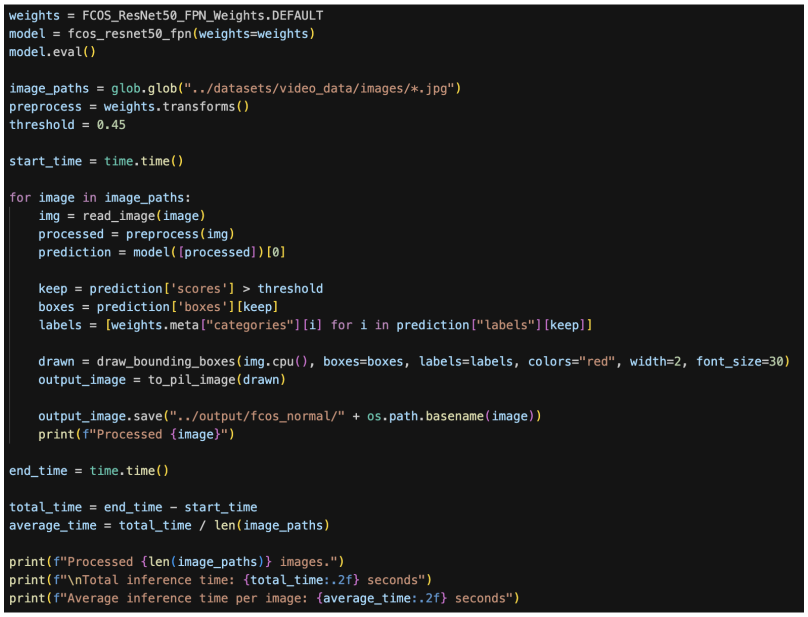


```
Benchmarks complete for yolov8n.pt on coco128.yaml at imgsz=640 (6.52s)
Benchmarks legend:  - ✅ Success  - ❎ Export passed but validation failed  - ❌ Export failed
   Format Status❔  Size (MB)  metrics/mAP50-95(B)  Inference time (ms/im)    FPS
0   ONNX       ✅       12.2               0.4537                    38.03  26.29

Benchmarks complete for yolov8n.pt on coco128.yaml at imgsz=640 (6.96s)
Benchmarks legend:  - ✅ Success  - ❎ Export passed but validation failed  - ❌ Export failed
      Format Status❔  Size (MB)  metrics/mAP50-95(B)  Inference time (ms/im)    FPS
0  OpenVINO       ✅       12.3               0.4522                    26.99  37.06

Benchmarks complete for yolov8n.pt on coco128.yaml at imgsz=640 (7.67s)
Benchmarks legend:  - ✅ Success  - ❎ Export passed but validation failed  - ❌ Export failed
     Format Status❔  Size (MB)  metrics/mAP50-95(B)  Inference time (ms/im)    FPS
0  PyTorch       ✅        6.2               0.4492                    53.18  18.8
```

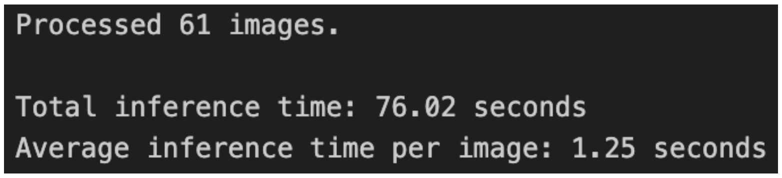
The dataset this is using is called coco128 which is a small subset (128 images) of the coco dataset. Since it comes with the true bounding box data, the benchmark function is able to compute the mAP50-95 score of the model. From the output, it is pretty easy to see the difference between each one. As expected, default pytorch inference of yolov8 is the slowest with an FPS of 18.8. Onnx comes second with an FPs of 26.29 and openvino is the fastest with an FPS of 37.06. This is almost double that of pytorch. Although it's very slight, the mAP50-95 score of onnx and openvino is higher than pytorch. I am not sure if this means anything however, as the difference is very small as well as the dataset size also being small. The result might be different if the dataset is way bigger.

Fcos_resnet50_fpn - pytorch

After finishing with yolov8, I went to work to do inference on another model. I chose fcos_resnet50_fpn for this next one. This model is a bit older than yolov8 but I think it should still be good. It is a torch vision model so there are not a lot of built in functions like yolov8 so its not too clear how to use this model. Luckily, I found an example usage of a torch vision model on the pytorch documentation website. After adapting the code to the model I am using and for what the homework is asking for, I got a working code for the pytorch inference.



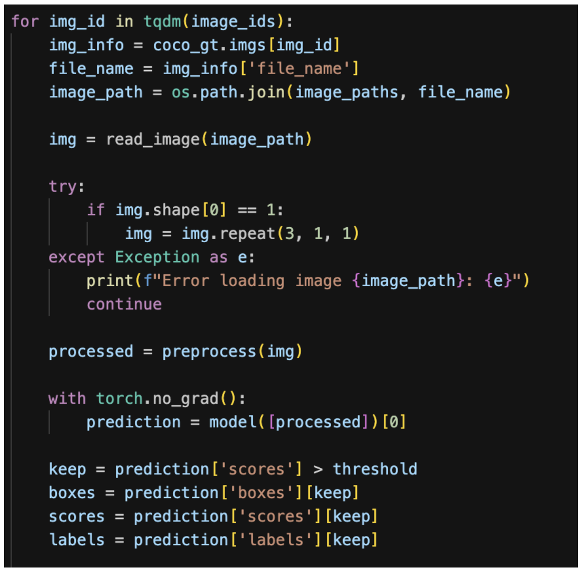
```python
weights = FCOS_ResNet50_FPN_Weights.DEFAULT
model = fcos_resnet50_fpn(weights=weights)
model.eval()

image_paths = glob.glob("../datasets/video_data/images/*.jpg")
preprocess = weights.transforms()
threshold = 0.45

start_time = time.time()

for image in image_paths:
    img = read_image(image)
    processed = preprocess(img)
    prediction = model([processed])[0]

    keep = prediction['scores'] > threshold
    boxes = prediction['boxes'][keep]
    labels = [weights.meta["categories"][i] for i in prediction["labels"][keep]]

    drawn = draw_bounding_boxes(img.cpu(), boxes=boxes, labels=labels, colors="red", width=2, font_size=30)
    output_image = to_pil_image(drawn)

    output_image.save("../output/fcos_normal/" + os.path.basename(image))
    print(f"Processed {image}")

end_time = time.time()

total_time = end_time - start_time
average_time = total_time / len(image_paths)

print(f"Processed {len(image_paths)} images.")
print(f"\nTotal inference time: {total_time:.2f} seconds")
print(f"Average inference time per image: {average_time:.2f} seconds")
```



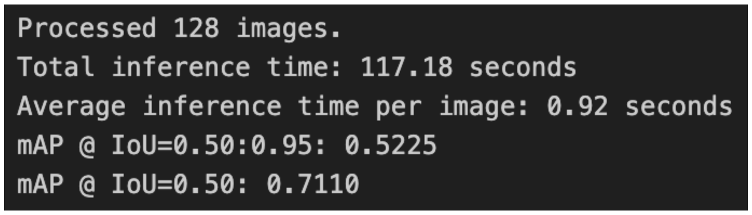
```
Processed 61 images.

Total inference time: 76.02 seconds
Average inference time per image: 1.25 seconds
```

This one is just a test of pytorch inference so I only did it with the dataset I extracted from the video I downloaded. From looking at the output, it's shocking compared to yolov8. The difference in inference time is night and day. The yolov8 pytorch model took 3.3 seconds while fcos took 76 seconds. This is a >20x seconds difference. At least looking at the image output, I can see that the model did just as well as yolov8 in detecting the objects in the image.

After this, I also implemented this code so that it can work with the coco128 dataset and compute the mAP50-95 score afterwards.



```python
for img_id in tqdm(image_ids):
    img_info = coco_gt.imgs[img_id]
    file_name = img_info['file_name']
    image_path = os.path.join(image_paths, file_name)

    img = read_image(image_path)

    try:
        if img.shape[0] == 1:
            img = img.repeat(3, 1, 1)
    except Exception as e:
        print(f"Error loading image {image_path}: {e}")
        continue

    processed = preprocess(img)

    with torch.no_grad():
        prediction = model([processed])[0]

    keep = prediction['scores'] > threshold
    boxes = prediction['boxes'][keep]
    scores = prediction['scores'][keep]
    labels = prediction['labels'][keep]
```

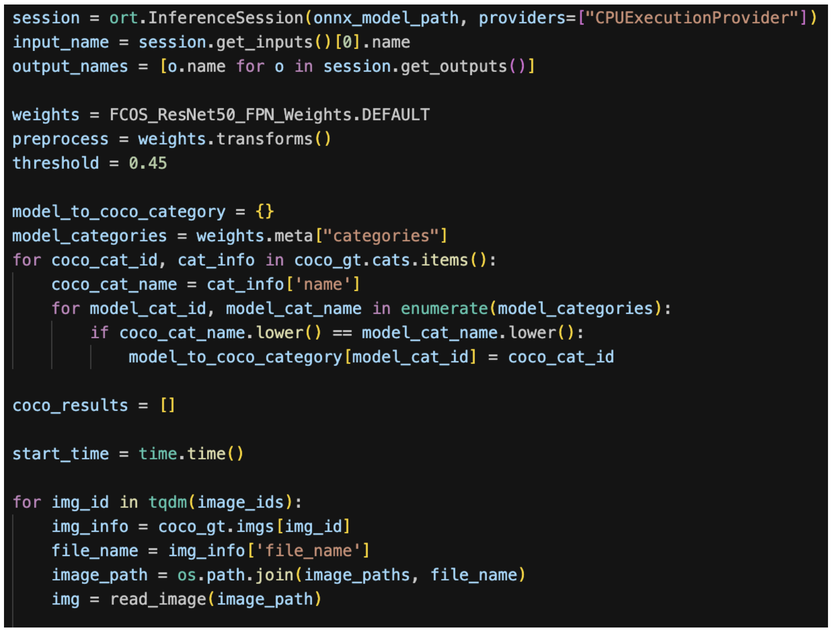


```
Processed 128 images.
Total inference time: 117.18 seconds
Average inference time per image: 0.92 seconds
mAP @ IoU=0.50:0.95: 0.5225
mAP @ IoU=0.50: 0.7110
```

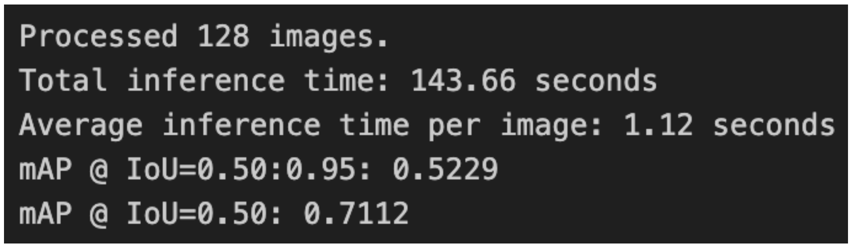
From the output, we can see that the inference time is still significantly slower than that of yolov8. The difference here though is that the mAP50-95 score is higher than that of yolov8. I guess this means that newer models like yolov8 sacrifice a bit of accuracy for way faster inference time.

Fcos_resnet50_fpn - onnx

For the last inference run, I edit the pytorch implementation to use onnx runtime for inference.



```python
session = ort.InferenceSession(onnx_model_path, providers=["CPUExecutionProvider"])
input_name = session.get_inputs()[0].name
output_names = [o.name for o in session.get_outputs()]

weights = FCOS_ResNet50_FPN_Weights.DEFAULT
preprocess = weights.transforms()
threshold = 0.45

model_to_coco_category = {}
model_categories = weights.meta["categories"]
for coco_cat_id, cat_info in coco_gt.cats.items():
    coco_cat_name = cat_info['name']
    for model_cat_id, model_cat_name in enumerate(model_categories):
        if coco_cat_name.lower() == model_cat_name.lower():
            model_to_coco_category[model_cat_id] = coco_cat_id

coco_results = []

start_time = time.time()

for img_id in tqdm(image_ids):
    img_info = coco_gt.imgs[img_id]
    file_name = img_info['file_name']
    image_path = os.path.join(image_paths, file_name)
    img = read_image(image_path)
```



```
Processed 128 images.
Total inference time: 143.66 seconds
Average inference time per image: 1.12 seconds
mAP @ IoU=0.50:0.95: 0.5229
mAP @ IoU=0.50: 0.7112
```

After inference using onnx and looking at the output, we can see that the inference time is actually higher than pytorch inference. I ran both code multiple times and the onnx inference consistently took about 20 more seconds to infer. The mAP50-95 score is marginally better but I don't think it is meaningful enough to say it's a benefit.

Conclusion

Yolov8 onnx and yolov8 openvino inference showed better inference time than default pytorch inference. Comparing them together however, openvino optimization performed better than onnx. Comparing yolov8 to fcos, yolov8 inference time is way faster than fcos, but it performs worse in terms of mAP50-95 score. Comparing yolov8 onnx inference to fcos onnx inference, we can see that while yolov8 benefits from using onnx optimization, fcos performs worse with onnx optimization.