COS30082 - Applied Machine Learning

Week 7 - Lab - Object Detection

The requirements of this lab are quite straightforward, which is just to find an image online or my own photo containing various objects, then I have to load the YOLOv8 model and detect the objects inside that image. Therefore, in order to get more familiar with ultralytics and the other tasks that YOLO models can perform, I will load all the models for each task (detection, segmentation, oriented bounding box, pose detection and classification) as well as performing an object detection task for a short video. The follow parts will demonstrate my works for this lab.
At the beginning, I will download ultralytics packages and import neccessary packages for this lab.

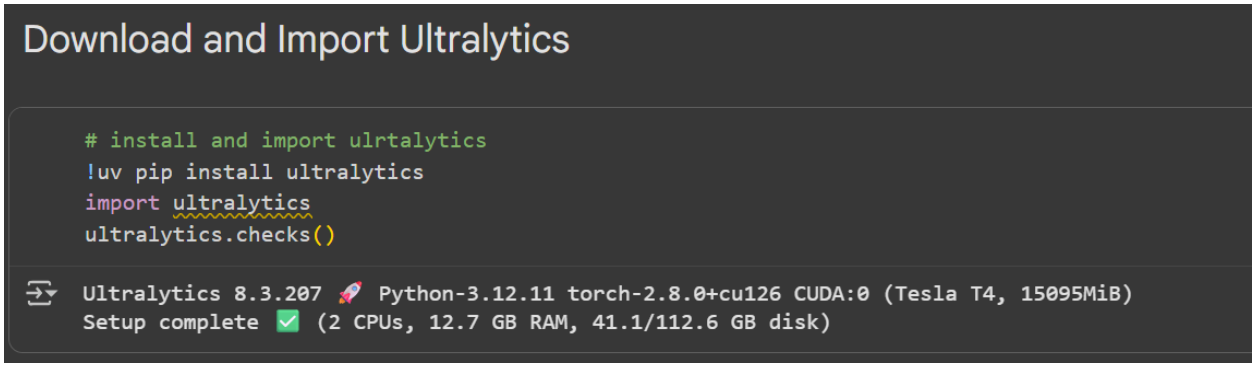

Figure 1: Download and import ultralytics successfully

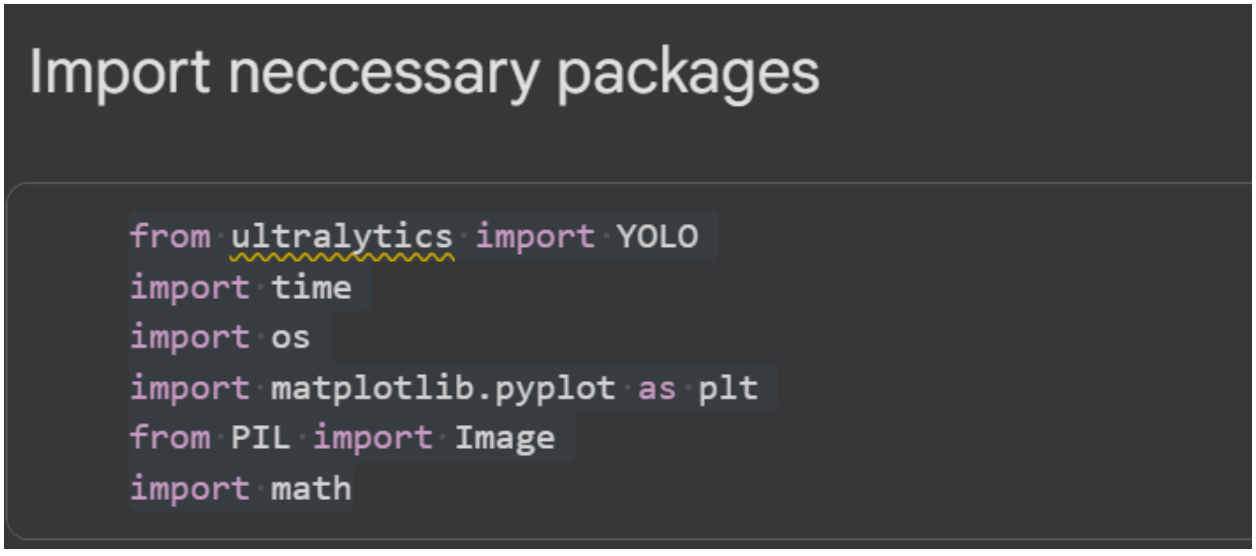

Figure 2: Import neccessary packages for this lab

For each task, ultralytics provide various model of different sizes (nano, small, medium, large, and extra large). If the model is larger, it offers higher accuracy but higher inference time. In this lab, I will load all the models and compare their performance to see the difference.

1. Detection Task

Figure 3: Image for detection task

```
# List of YOLOv8 model weights to test
models = [
    "yolov8n.pt",  # nano
    "yolov8s.pt",  # small
    "yolov8m.pt",  # medium
    "yolov8l.pt",  # large
    "yolov8x.pt"   # extra large
]
```

Figure 4: Lists of model used to test

```python
# Run detection for each model
for model_name in models:
    print(f"\nTesting {model_name}...")

    # Load model
    model = YOLO(model_name)

    # Inference and timing
    start_time = time.time()
    results = model(image_path, save=True, project="results", name=model_name.split('.')[0], exist_ok=True)
    end_time = time.time()

    # Count detections
    num_detections = len(results[0].boxes)
    detection_time = round(end_time - start_time, 3)

    # Get saved image path
    result_img_path = os.path.join(results[0].save_dir, os.path.basename(results[0].path))

    results_summary.append({
        "Model": model_name,
        "Detections": num_detections,
        "Inference Time (s)": detection_time,
        "Image Path": result_img_path
    })

    output_images.append(result_img_path)

    print(f"Done: {num_detections} objects detected in {detection_time:.3f} seconds.")
    print(f"Saved at: {result_img_path}")
```

Figure 5: Code for running inference

```python
# --- Display results summary ---
print("\n--- YOLOv8 Model Performance Comparison ---")
for r in results_summary:
    print(f"{r['Model']:<10} | Detections: {r['Detections']:<3} | Inference Time: {r['Inference Time (s)']}s")
```

```
--- YOLOv8 Model Performance Comparison ---
yolov8n.pt | Detections: 14  | Inference Time: 1.703s
yolov8s.pt | Detections: 14  | Inference Time: 0.183s
yolov8m.pt | Detections: 13  | Inference Time: 0.307s
yolov8l.pt | Detections: 12  | Inference Time: 0.466s
yolov8x.pt | Detections: 14  | Inference Time: 0.792s
```

Figure 6: Model performance comparison

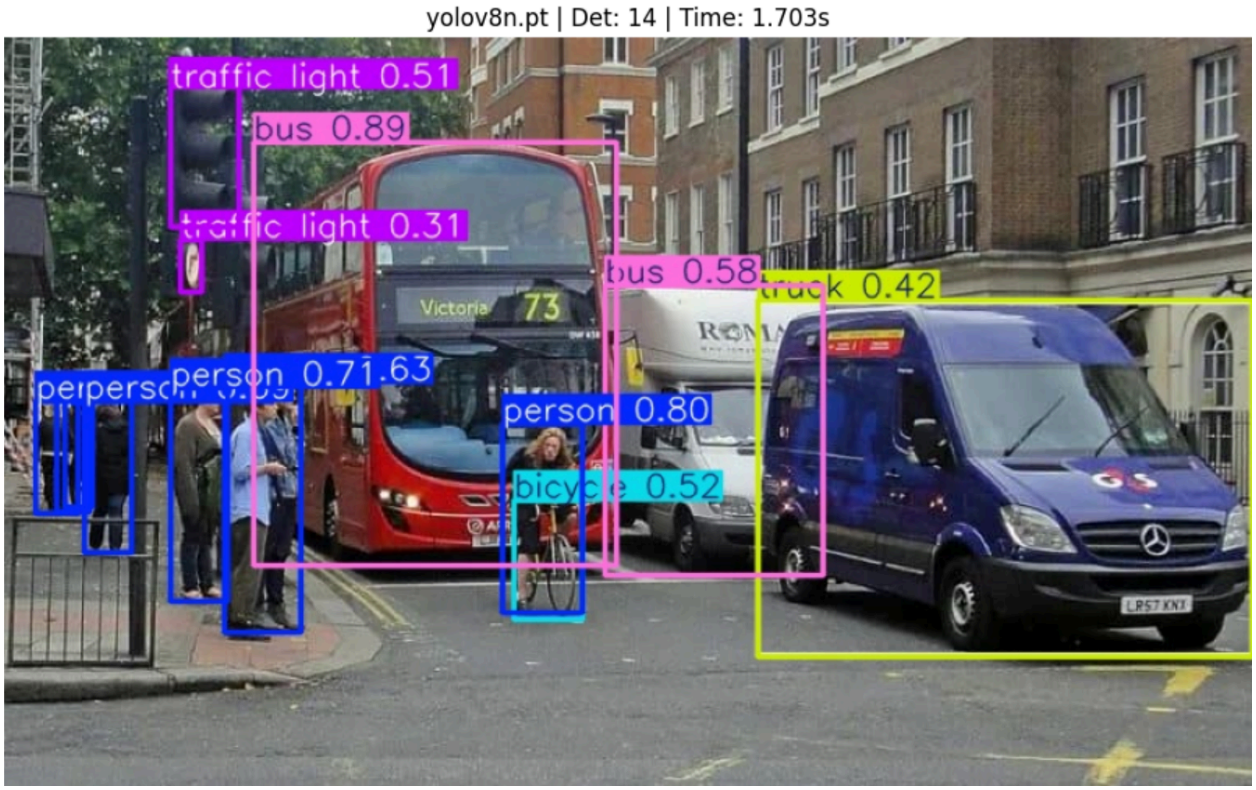

Figure 7: Results from nano model

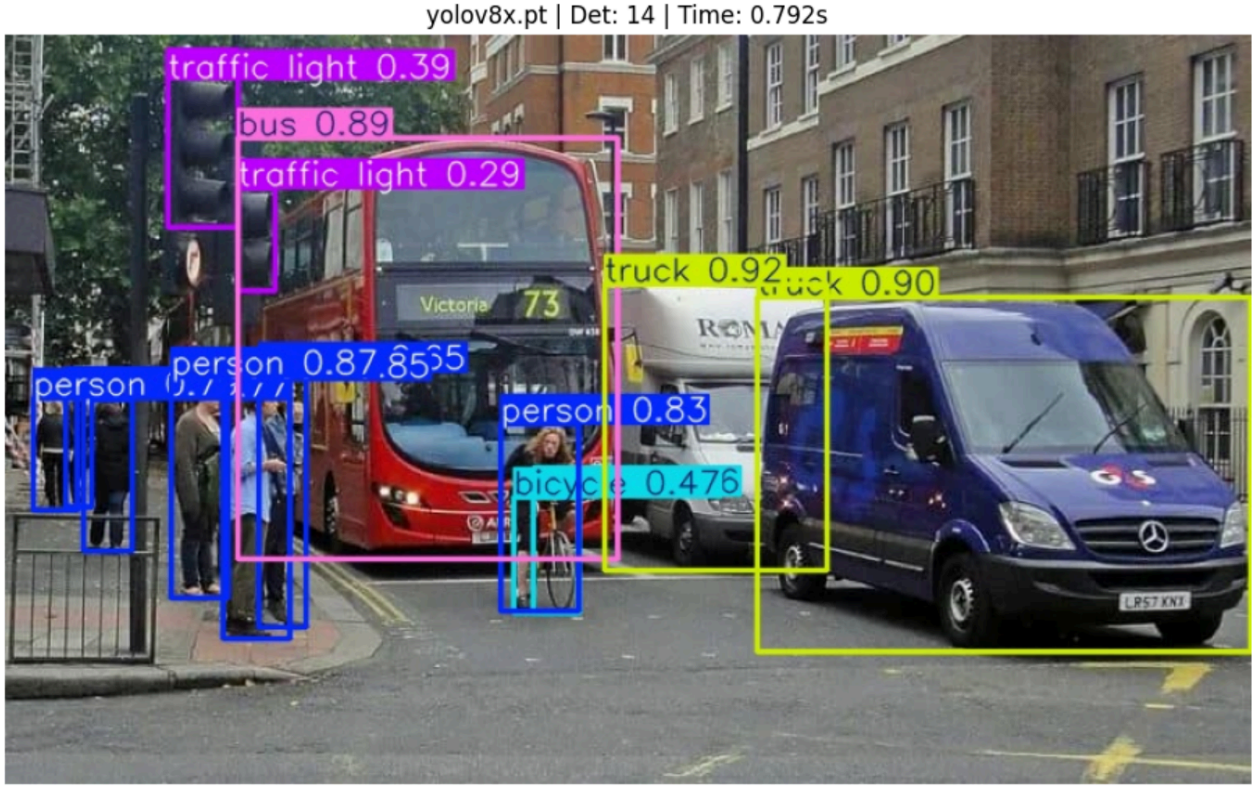

Figure 8: Results from extra large model

From the figures above, we can see that the inference time is higher for larger model. In terms of the accuracy, we can see some wrong detections in the nano model, shuch as the misclassification between truck and bus, traffic light and traffic sign.

2. Segmentation Task

Figure 9: Image used for segmentation task

```
# List of YOLOv8 segmentation models
models = [
    "yolov8n-seg.pt",
    "yolov8s-seg.pt",
    "yolov8m-seg.pt",
    "yolov8l-seg.pt",
    "yolov8x-seg.pt"
]
```

Figure 10: List of models used for segmentation

```python
# Array to store the results
results_summary = []
output_images = []

# Run inference for each model
for model_name in models:
    print(f"Running {model_name}...")

    # load the model
    model = YOLO(model_name)

    # Start timer
    start_time = time.time()

    # do the inference
    results = model(
        image_path,
        task="segment",
        save=True,
        project="results_seg",
        name=model_name.split('.')[0],
        exist_ok=True
    )

    # End timer
    end_time = time.time()

    # Number of objects being segmented
    num_segments = len(results[0].boxes) if hasattr(results[0], "boxes") and results[0].boxes is not None else 0

    # Calculate the inference time
    inference_time = round(end_time - start_time, 3)

    # Save the results
    result_img_path = os.path.join(results[0].save_dir, os.path.basename(results[0].path))
    output_images.append(result_img_path)

    results_summary.append({
        "Model": model_name,
        "Segments": num_segments,
        "Inference Time (s)": inference_time,
        "Image Path": result_img_path
    })

    print(f"{num_segments} objects segmented in {inference_time}s. Saved at {result_img_path}")
```

Figure 11: Code for doing the inference

```python
# Display performance summary
print("\nSegmentation Performance Summary:")
for r in results_summary:
    print(f"{r['Model']:<15} | Segments: {r['Segments']:<3} | Time: {r['Inference Time (s)']}s")
```

```
Segmentation Performance Summary:
yolov8n-seg.pt  | Segments: 7   | Time: 0.232s
yolov8s-seg.pt  | Segments: 6   | Time: 0.172s
yolov8m-seg.pt  | Segments: 4   | Time: 0.327s
yolov8l-seg.pt  | Segments: 5   | Time: 0.476s
yolov8x-seg.pt  | Segments: 5   | Time: 0.813s
```

Figure 12: Segmentation performance comparison

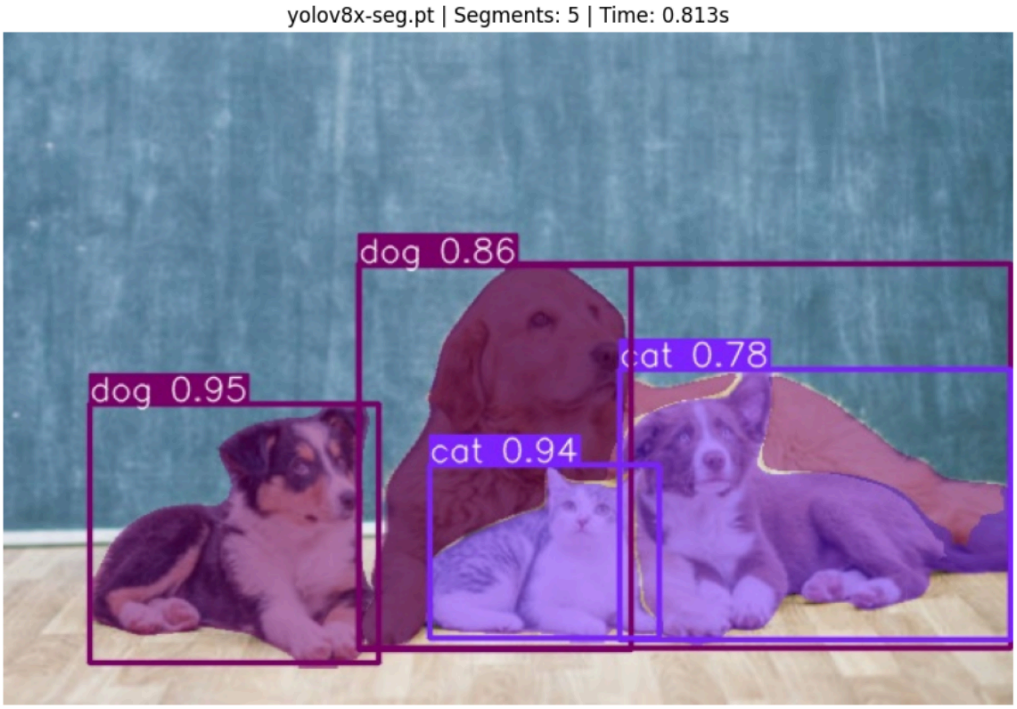

Figure 13: Result of the extra large model

From the figures for segmentation task, there is still wrong segmentation even for the extra large model. The inference time is still higher for larger model.

3. Pose Detection Task

Figure 14: Image used for pose detection.

```python
# List of YOLOv8 pose models
models = [
    "yolov8n-pose.pt",
    "yolov8s-pose.pt",
    "yolov8m-pose.pt",
    "yolov8l-pose.pt",
    "yolov8x-pose.pt",
    "yolov8x-pose-p6.pt"
]
```

Figure 15: List of the models for pose detection

```python
# Arrays to store the results
results_summary = []
output_images = []

# Run pose detection for each model
for model_name in models:
    print(f"Running {model_name}...")

    # Load the model
    model = YOLO(model_name)

    # Start timer
    start_time = time.time()

    # Running inference
    results = model(
        image_path,
        task="pose",
        save=True,
        project="results_pose",
        name=model_name.split('.')[0],
        exist_ok=True
    )

    # End timer
    end_time = time.time()

    # Calclate the number of poses
    num_poses = len(results[0].keypoints) if hasattr(results[0], "keypoints") and results[0].keypoints is not None else 0

    # Calculate the inference time
    inference_time = round(end_time - start_time, 3)

    # Save the results
    result_img_path = os.path.join(results[0].save_dir, os.path.basename(results[0].path))
    output_images.append(result_img_path)

    results_summary.append({
        "Model": model_name,
        "Poses": num_poses,
        "Inference Time (s)": inference_time,
        "Image Path": result_img_path
    })

    print(f"{num_poses} poses detected in {inference_time}s. Saved at {result_img_path}")
```

Figure 16: Code for pose detection inference

```
# Display performance summary
print("\nPose Detection Performance Summary:")
for r in results_summary:
    print(f"{r['Model']:<15} | Poses: {r['Poses']:<3} | Time: {r['Inference Time (s)']}s")
```

```
Pose Detection Performance Summary:
yolov8n-pose.pt | Poses: 3   | Time: 0.197s
yolov8s-pose.pt | Poses: 3   | Time: 0.154s
yolov8m-pose.pt | Poses: 3   | Time: 0.307s
yolov8l-pose.pt | Poses: 3   | Time: 0.477s
yolov8x-pose.pt | Poses: 3   | Time: 0.768s
yolov8x-pose-p6.pt | Poses: 3   | Time: 1.288s
```

Figure 17: Pose detection performance comparison

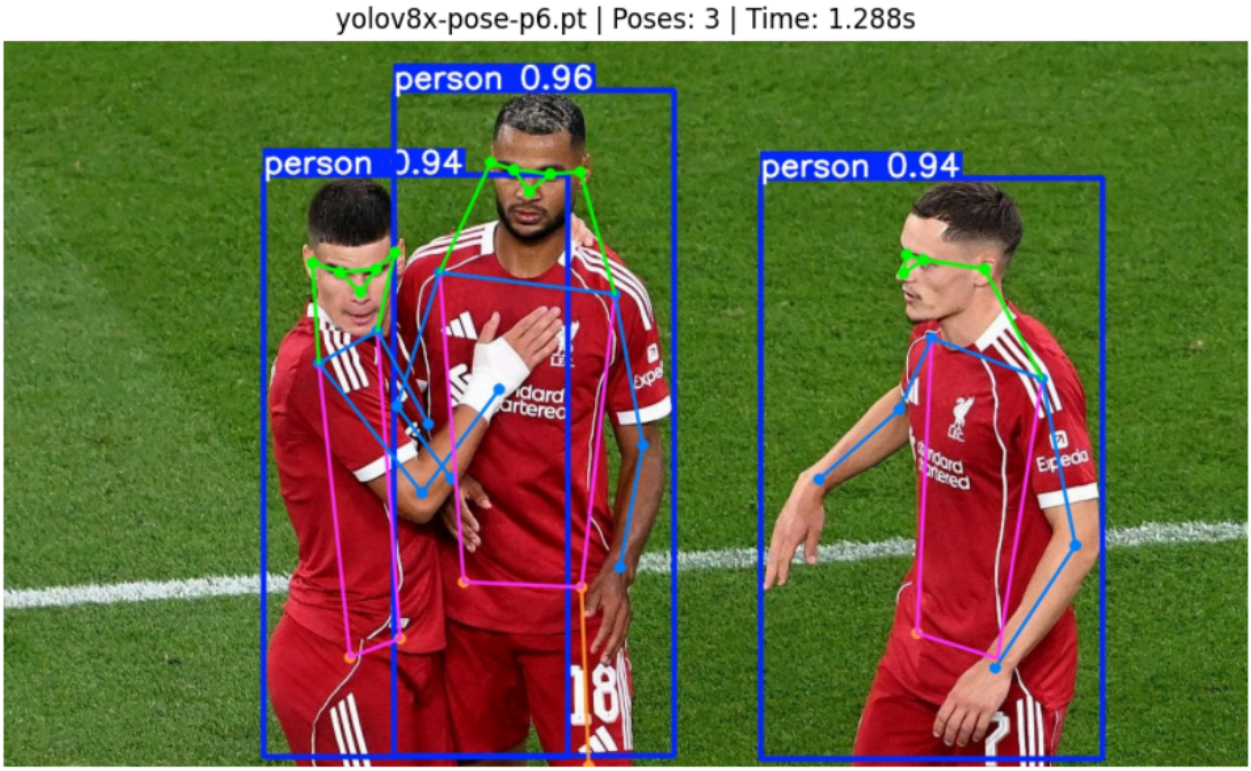

Figure 18: Results for pose detection

For pose detection task, because the image is quite simple, all the models provide the same output. However, the confidence scores for the larger models are higher.

4. OBB Task

Figure 19: Image used for OBB task

```
# List of YOLOv8 OBB models
models = [
    "yolov8n-obb.pt",
    "yolov8s-obb.pt",
    "yolov8m-obb.pt",
    "yolov8l-obb.pt",
    "yolov8x-obb.pt"
]
```

Figure 20: Lists of models used for OBB task

```python
# Create arrays to store results
results_summary = []
output_images = []

# Run object detection for each model
for model_name in models:
    print(f"Running {model_name}...")

    # Load the model
    model = YOLO(model_name)

    # Start timer
    start_time = time.time()

    # Running inference
    results = model(
        image_path,
        task="detect",
        save=True,
        project="results_obb",
        name=model_name.split('.')[0],
        exist_ok=True
    )

    # End timer
    end_time = time.time()

    # Count the number of objects
    num_objects = len(results[0].boxes) if hasattr(results[0], "boxes") and results[0].boxes is not None else 0

    # Calculate the inference time
    inference_time = round(end_time - start_time, 3)

    # Store the results
    result_img_path = os.path.join(results[0].save_dir, os.path.basename(results[0].path))
    output_images.append(result_img_path)

    results_summary.append({
        "Model": model_name,
        "Objects": num_objects,
        "Inference Time (s)": inference_time,
        "Image Path": result_img_path
    })

    print(f"{num_objects} objects detected in {inference_time}s. Saved at {result_img_path}")
```

Figure 21: Code for doing inference for OBB

```python
# Display performance summary
print("\nOBB Detection Performance Summary:")
for r in results_summary:
    print(f"{r['Model']:<15} | Time: {r['Inference Time (s)']}s")
```

```
OBB Detection Performance Summary:
yolov8n-obb.pt  | Time: 0.473s
yolov8s-obb.pt  | Time: 0.329s
yolov8m-obb.pt  | Time: 0.475s
yolov8l-obb.pt  | Time: 0.686s
yolov8x-obb.pt  | Time: 0.988s
```

Figure 22: OBB performance comparison

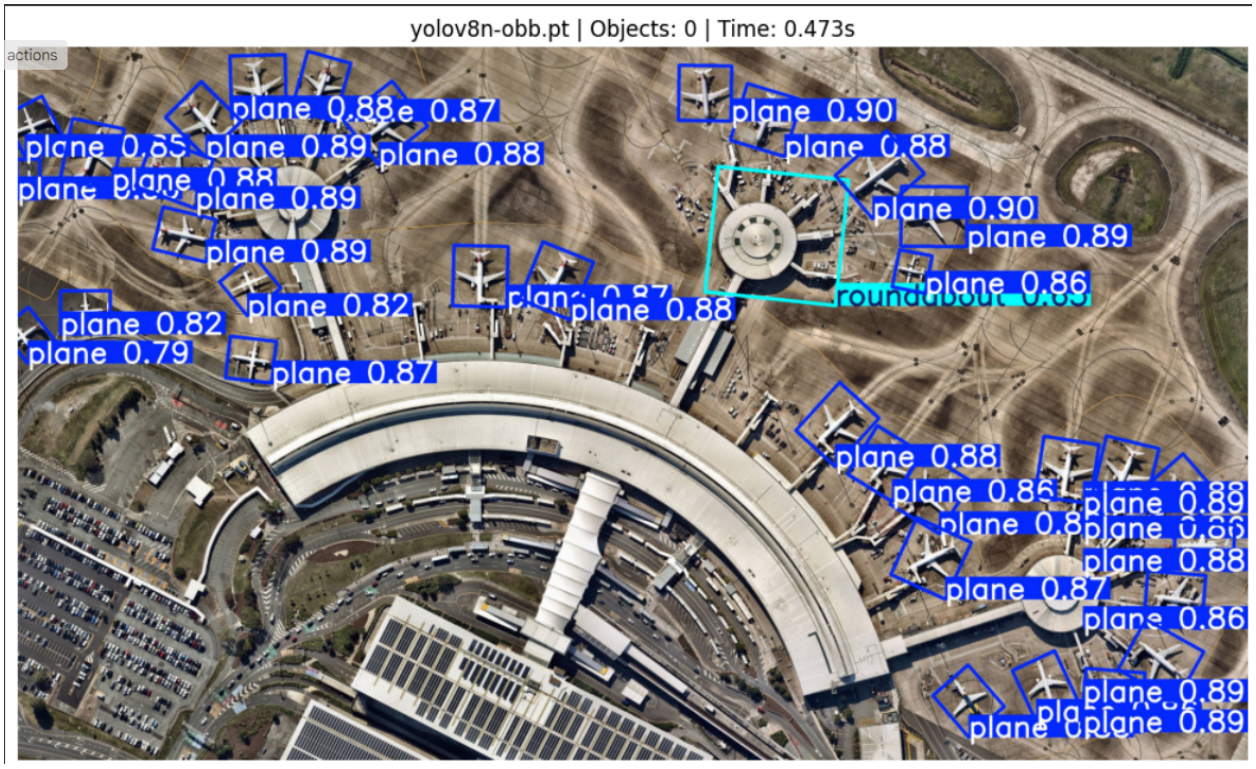

Figure 23: Result for the nano OBB model

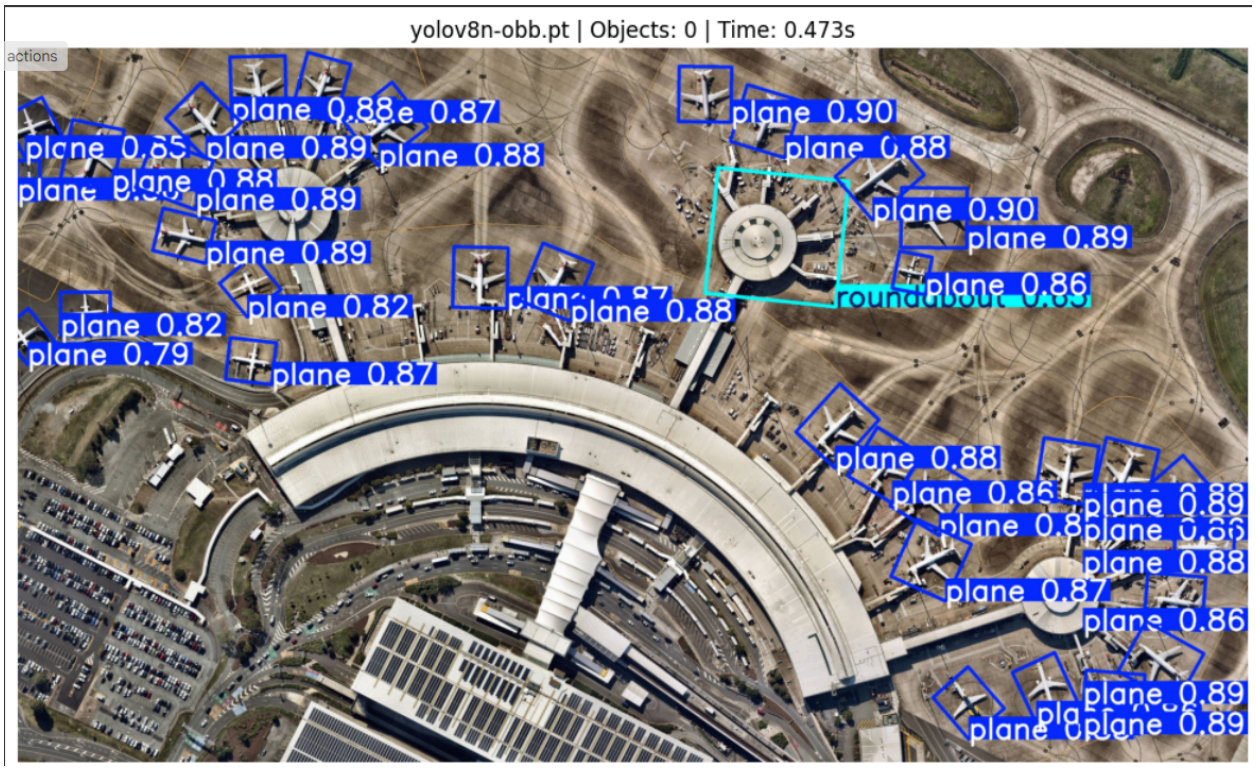

Figure 24: Result for extra large OBB model

5. Classification Task

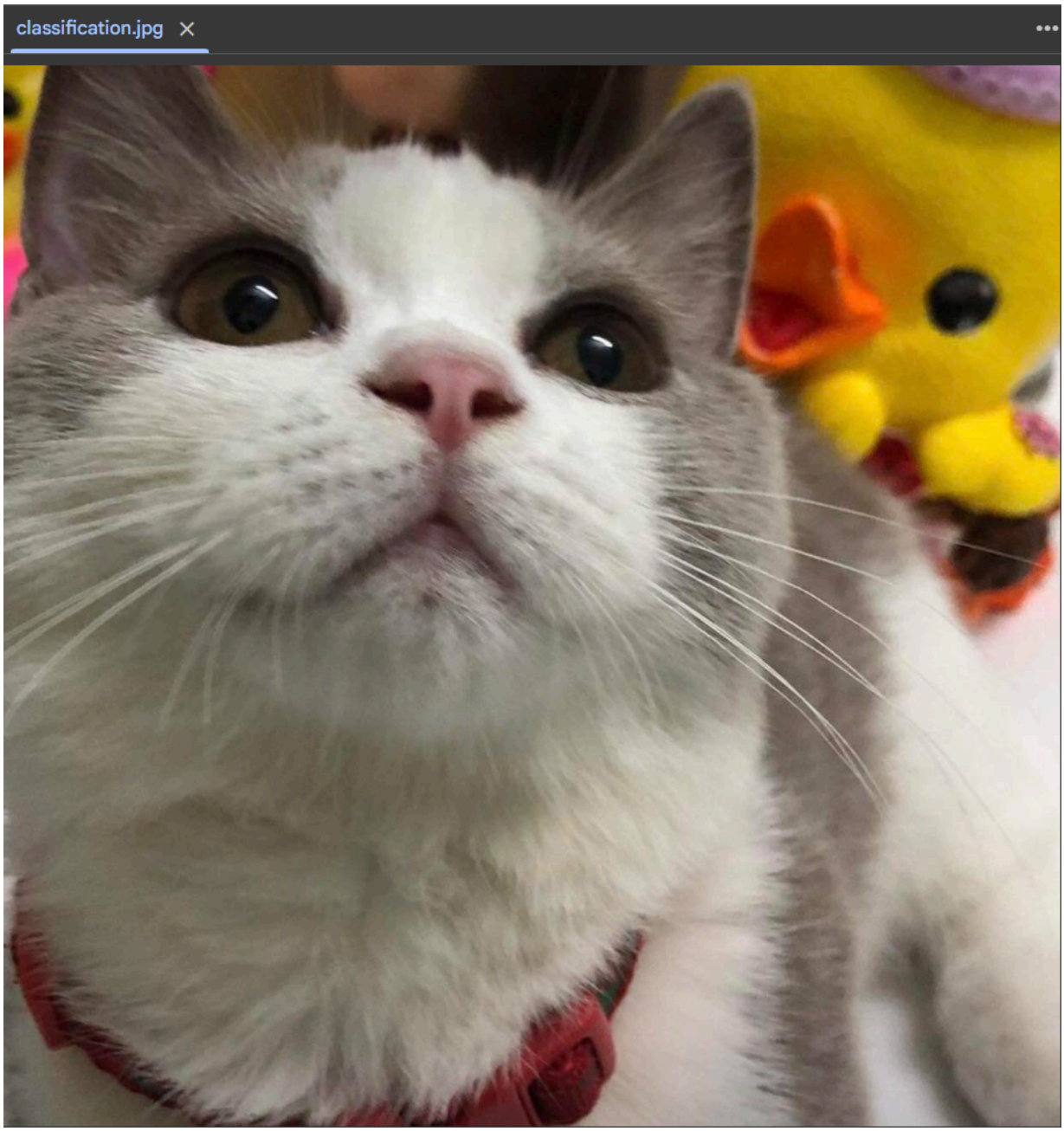

Figure 25: Image used for classification task

```python
# List of YOLOv8 classification models
models = [
    "yolov8n-cls.pt",
    "yolov8s-cls.pt",
    "yolov8m-cls.pt",
    "yolov8l-cls.pt",
    "yolov8x-cls.pt"
]
```

Figure 26: List of models used for classification task

```python
# Do inference for each model
for model_name in models:
    print(f"Running {model_name}...")

    # Load the model
    model = YOLO(model_name)


    # Start the timer
    start_time = time.time()

    # Running inference
    results = model(
        image_path,
        task="classify",
        save=True,
        project="results_cls",
        name=model_name.split('.')[0],
        exist_ok=True
    )

    # End timer
    end_time = time.time()

    # Get predicted class and confidence
    if hasattr(results[0], "probs") and results[0].probs is not None:
        class_idx = results[0].probs.top1
        predicted_class = results[0].names[class_idx]
        confidence = float(results[0].probs.top1conf)
    else:
        predicted_class = "Unknown"
        confidence = 0.0

    # Calculate the inference time
    inference_time = round(end_time - start_time, 3)

    # Store the results
    result_img_path = os.path.join(results[0].save_dir, os.path.basename(results[0].path))
    output_images.append(result_img_path)

    results_summary.append({
        "Model": model_name,
        "Class": predicted_class,
        "Confidence": confidence,
        "Inference Time (s)": inference_time,
        "Image Path": result_img_path
    })

    print(f"Predicted class: {predicted_class} (conf: {confidence:.2f}) in {inference_time}s. Saved at {result_img_path}")
```

Figure 27: Code on doing inference for doing inference

```python
# Display performance summary
print("\nClassification Performance Summary:")
for r in results_summary:
    print(f"{r['Model']:<15} | Class: {r['Class']:<15} | Time: {r['Inference Time (s)']}s")
```

```
Classification Performance Summary:
yolov8n-cls.pt  | Class: Persian_cat     | Time: 0.546s
yolov8s-cls.pt  | Class: Persian_cat     | Time: 0.143s
yolov8m-cls.pt  | Class: Persian_cat     | Time: 0.308s
yolov8l-cls.pt  | Class: Persian_cat     | Time: 0.521s
yolov8x-cls.pt  | Class: Persian_cat     | Time: 0.945s
```

Figure 28: Classification performance comparison

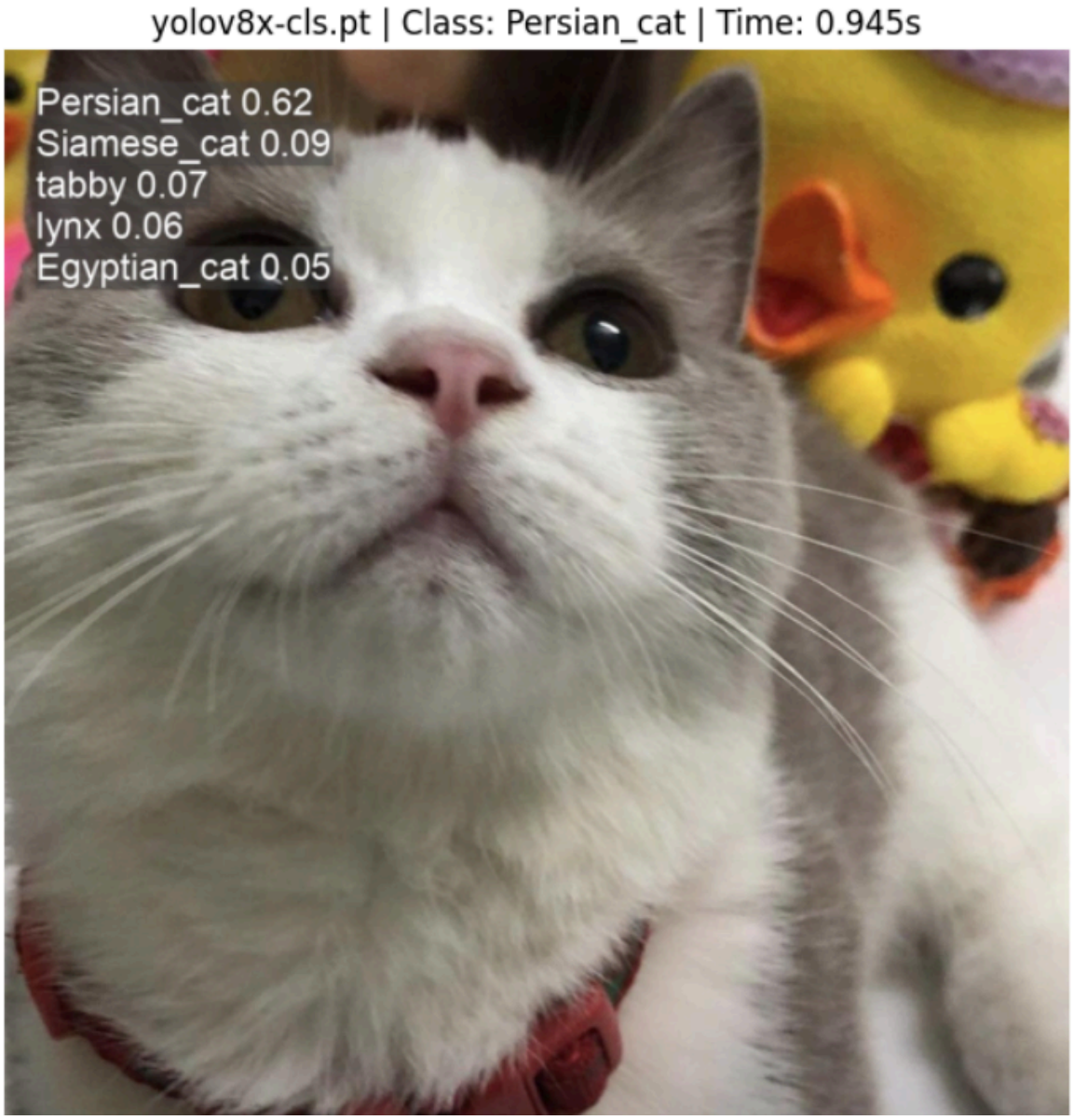

Figure 29: Result of the extra large model

5. Video Detection Task

For video detection, I will use the extra large YOLOv8 model (yolov8x.pt) and run this model on a short 5s video. The model successfully detect objects in the video for each frame.

```
# Load model
model = YOLO(model_name)

# Run detection on video
start_time = time.time()
results = model(
    video_path,
    save=True,
    project="results_video",
    name=model_name.split('.')[0],
    exist_ok=True
)
end_time = time.time()

# Calculate inference time
inference_time = round(end_time - start_time, 3)

# Save the outputs
output_video_path = os.path.join(results[0].save_dir, os.path.basename(results[0].path))

print(f"Video processed in {inference_time}s. Saved at {output_video_path}")
```

```
video 1/1 (frame 97/151) /content/videos/video.mp4: 384x640 4 persons, 17 cars, 4 motorcycles, 2 trucks, 1 traffic light, 31.4ms
video 1/1 (frame 98/151) /content/videos/video.mp4: 384x640 3 persons, 18 cars, 3 motorcycles, 2 trucks, 1 traffic light, 31.5ms
video 1/1 (frame 99/151) /content/videos/video.mp4: 384x640 4 persons, 18 cars, 3 motorcycles, 1 truck, 2 traffic lights, 31.5ms
video 1/1 (frame 100/151) /content/videos/video.mp4: 384x640 4 persons, 19 cars, 4 motorcycles, 3 trucks, 2 traffic lights, 31.4ms
video 1/1 (frame 101/151) /content/videos/video.mp4: 384x640 4 persons, 19 cars, 4 motorcycles, 2 trucks, 1 traffic light, 31.6ms
video 1/1 (frame 102/151) /content/videos/video.mp4: 384x640 5 persons, 18 cars, 3 motorcycles, 3 trucks, 1 traffic light, 30.1ms
video 1/1 (frame 103/151) /content/videos/video.mp4: 384x640 4 persons, 18 cars, 4 motorcycles, 3 trucks, 1 traffic light, 30.0ms
video 1/1 (frame 104/151) /content/videos/video.mp4: 384x640 4 persons, 19 cars, 3 motorcycles, 2 trucks, 1 traffic light, 30.1ms
video 1/1 (frame 105/151) /content/videos/video.mp4: 384x640 4 persons, 18 cars, 3 motorcycles, 3 trucks, 2 traffic lights, 30.3ms
video 1/1 (frame 106/151) /content/videos/video.mp4: 384x640 4 persons, 19 cars, 3 motorcycles, 3 trucks, 1 traffic light, 30.3ms
video 1/1 (frame 107/151) /content/videos/video.mp4: 384x640 4 persons, 17 cars, 3 motorcycles, 3 trucks, 1 traffic light, 30.2ms
video 1/1 (frame 108/151) /content/videos/video.mp4: 384x640 4 persons, 18 cars, 3 motorcycles, 3 trucks, 1 traffic light, 30.4ms
video 1/1 (frame 109/151) /content/videos/video.mp4: 384x640 3 persons, 18 cars, 3 motorcycles, 3 trucks, 1 traffic light, 30.5ms
video 1/1 (frame 110/151) /content/videos/video.mp4: 384x640 3 persons, 19 cars, 3 motorcycles, 2 trucks, 1 traffic light, 30.5ms
video 1/1 (frame 111/151) /content/videos/video.mp4: 384x640 4 persons, 19 cars, 3 motorcycles, 2 trucks, 1 traffic light, 30.5ms
video 1/1 (frame 112/151) /content/videos/video.mp4: 384x640 4 persons, 19 cars, 3 motorcycles, 2 trucks, 1 traffic light, 30.4ms
video 1/1 (frame 113/151) /content/videos/video.mp4: 384x640 4 persons, 20 cars, 4 motorcycles, 2 trucks, 1 traffic light, 30.6ms
video 1/1 (frame 114/151) /content/videos/video.mp4: 384x640 4 persons, 19 cars, 3 motorcycles, 2 trucks, 1 traffic light, 30.3ms
video 1/1 (frame 115/151) /content/videos/video.mp4: 384x640 5 persons, 18 cars, 3 motorcycles, 2 trucks, 1 traffic light, 30.5ms
video 1/1 (frame 116/151) /content/videos/video.mp4: 384x640 4 persons, 18 cars, 5 motorcycles, 1 truck, 1 traffic light, 31.1ms
video 1/1 (frame 117/151) /content/videos/video.mp4: 384x640 4 persons, 19 cars, 3 motorcycles, 2 trucks, 1 traffic light, 30.3ms
video 1/1 (frame 118/151) /content/videos/video.mp4: 384x640 4 persons, 20 cars, 3 motorcycles, 1 truck, 1 traffic light, 32.2ms
video 1/1 (frame 119/151) /content/videos/video.mp4: 384x640 4 persons, 18 cars, 3 motorcycles, 2 trucks, 1 traffic light, 31.3ms
video 1/1 (frame 120/151) /content/videos/video.mp4: 384x640 3 persons, 18 cars, 3 motorcycles, 1 truck, 1 traffic light, 30.5ms
video 1/1 (frame 121/151) /content/videos/video.mp4: 384x640 4 persons, 18 cars, 3 motorcycles, 1 truck, 1 traffic light, 30.4ms
video 1/1 (frame 122/151) /content/videos/video.mp4: 384x640 4 persons, 19 cars, 3 motorcycles, 2 trucks, 1 traffic light, 30.5ms
video 1/1 (frame 123/151) /content/videos/video.mp4: 384x640 4 persons, 19 cars, 3 motorcycles, 1 truck, 1 traffic light, 30.2ms
video 1/1 (frame 124/151) /content/videos/video.mp4: 384x640 3 persons, 20 cars, 3 motorcycles, 1 truck, 1 traffic light, 30.0ms
video 1/1 (frame 125/151) /content/videos/video.mp4: 384x640 4 persons, 19 cars, 5 motorcycles, 2 trucks, 1 traffic light, 30.1ms
video 1/1 (frame 126/151) /content/videos/video.mp4: 384x640 3 persons, 18 cars, 4 motorcycles, 1 truck, 1 traffic light, 30.1ms
```

Figure 30: Code for video detection and detection for each frame