PROJECT REPORT

COURSE: AI&ES

COURSE CODE: CT-361

DEPARTMENT: CSIT

Group Members:

Kumel Ahmed (CT-22034)

Wahaj Ahmed (CT-22032)

Shahzaib Ahmed (CT-22042)

Hasan Mohi Uddin (CT-22024)

Submitted To:

Sir Abdullah

NED UNIVERSITY OF ENGINNERING & TECHNOLOGY

Real-Time Object Detection and Numeration System for Autonomous Robots Using YOLOv8

Contents

1. Introduction

This project develops an advanced AI-powered vision system for autonomous robots that enables real-time object detection and contextual caption generation using the YOLOv8 deep learning model. The system processes images and live video feeds to accurately identify and classify objects in the robot's environment, providing essential perception capabilities for navigation, manipulation, and interaction.

By integrating this object detection pipeline, robots can dynamically understand their surroundings, recognize obstacles, detect relevant objects, and generate descriptive insights to assist in decision-making. The project includes data preprocessing, model training with custom datasets, performance evaluation, interactive inference through image uploads, and live detection via robot-mounted cameras or webcams.

2. Key Components and Functionalities

2.1 Dataset Loading and Processing

Description:

- **Purpose:** Prepare the dataset in YOLO format for training.
- **Input:** Open Images dataset with bounding box annotations.
- **Output:** Train/validation image-label pairs in YOLO format.

Key Code Components:

```
def load_dataset():
    # Load classes
    classes_df = pd.read_csv(CLASSES_PATH)
    class_dict = {row.LabelName: idx for idx, row in enumerate(classes_df.itertuples())}
    class_names = classes_df['DisplayName'].tolist()

    # Load annotations
    annotations_df = pd.read_csv(ANNOTATIONS_PATH)

    # Get image paths
    image_files = list(Path(TRAIN_DATA_PATH).glob("*.jpg"))
    image_ids = [img.stem for img in image_files]

    print(f"Loaded {len(class_names)} classes and {len(image_files)} images")
    return annotations_df, class_dict, class_names, image_files, image_ids
```

- **annotations_df:** DataFrame containing object bounding boxes and metadata.
- **class_dict:** Mapping of class label IDs to integer indices.
- **class_names:** List of class display names.

```
def convert_to_yolo_format(annotations_df, class_dict, image_files, image_ids, train_ratio=0.8):
    # Randomly split data
    random.seed(42)
    random.shuffle(image_ids)
    split_idx = int(len(image_ids) * train_ratio)
    train_ids = set(image_ids[:split_idx])
    val_ids = set(image_ids[split_idx:])

    for img_path in image_files:
        img_id = img_path.stem
        img_annotations = annotations_df[annotations_df['ImageID'] == img_id]

        if img_annotations.empty:
            continue

        subset = "train" if img_id in train_ids else "val"

        # Copy image
        #shutil.copy(img_path, os.path.join(YOLO_DIR, "images", subset, img_path.name))
        dst_path = os.path.join(YOLO_DIR, "images", subset, img_path.name)
        if not os.path.exists(dst_path):
            shutil.copy(img_path, dst_path)
        # Create annotation file
        label_path = os.path.join(YOLO_DIR, "labels", subset, f"{img_id}.txt")
        with open(label_path, "w") as f:
            for _, row in img_annotations.iterrows():
                if row['LabelName'] not in class_dict:
                    continue
                class_id = class_dict[row['LabelName']]
                x_min, x_max = float(row['XMin']), float(row['XMax'])
                y_min, y_max = float(row['YMin']), float(row['YMax'])

                # Normalized values — do NOT divide again
                x_center = (x_min + x_max) / 2
                y_center = (y_min + y_max) / 2
                width = x_max - x_min
                height = y_max - y_min

                # Optional: Skip tiny boxes (common with noise in Open Images)
                if width < 0.01 or height < 0.01:
                    continue

                f.write(f"{class_id} {x_center} {y_center} {width} {height}\n")
                print(f"Writing label file: {label_path}")


print(f"Converted {len(train_ids)} training images and {len(val_ids)} validation images")
return train_ids, val_ids
```

- **Conversion:** Normalizes bounding boxes (x_center, y_center, width, height) relative to image dimensions.
- **Filtering:** Skips extremely small boxes to reduce noise.

2.2 Dataset Configuration (YAML)

Description:

Generates a .yaml configuration required by YOLOv8 to locate the dataset and define classes.

```
def create_yaml_config(class_names):
    """Create YAML configuration file for YOLOv8."""
    yaml_content = {
        'train': os.path.join(YOLO_DIR, 'images', 'train'),
        'val': os.path.join(YOLO_DIR, 'images', 'val'),
        'nc': len(class_names),
        'names': class_names
    }

    yaml_path = os.path.join(YOLO_DIR, 'dataset.yaml')
    with open(yaml_path, 'w') as f:
        yaml_str = f"""train: {yaml_content['train']}
val: {yaml_content['val']}
nc: {yaml_content['nc']}
names: {yaml_content['names']}"""
        f.write(yaml_str)

    print(f"Created YAML configuration at {yaml_path}")
    return yaml_path
```

```
FORCE_PREPROCESS = False

if FORCE_PREPROCESS or not os.path.exists(os.path.join(YOLO_DIR, 'dataset.yaml')):
    annotations_df, class_dict, class_names, image_files, image_ids = load_dataset()
    train_ids, val_ids = convert_to_yolo_format(annotations_df, class_dict, image_files, image_ids)
    yaml_path = create_yaml_config(class_names)
else:
    yaml_path = os.path.join(YOLO_DIR, 'dataset.yaml')
    print(f"YOLO dataset already processed. Using existing config at {yaml_path}")
```

- **train:** Path to training images.
- **val:** Path to validation images.
- **nc:** Number of object classes.
- **names:** Human-readable class names.

2.3 Model Training

Description:

Fine-tunes the YOLOv8 model using the processed dataset.

```python
def train_yolo_model(yaml_path):  # added workers as parameter

    # Updated model path
    model_path = r'C:\Users\Hacking\AI Project\runs\detect\train6\weights\last.pt'

    # Load weights from the last model checkpoint for fine-tuning
    model = YOLO(model_path)

    results = model.train(
        data=yaml_path,
        epochs=25,
        imgsz=640,
        batch=16,
        save=True,
        resume=True,  # Resume if a checkpoint exists in the default run folder
        device=0 if torch.cuda.is_available() else 'cpu',
    )

    print(f"Training completed. Results saved to {model.trainer.save_dir}")
    return model
```

- **model_path:** Loads weights from a previous checkpoint (fine-tuning).
- **epochs, batch_size, img_size:** Standard training parameters.
- **resume:** Enables continuation of interrupted training.
- **lr0:** Initial learning rate for optimization.
- **workers:** Number of data loading threads.

2.4 Evaluation

```python
def evaluate_model(model, yaml_path):
    # Run validation
    val_results = model.val(data=yaml_path)

    # Print results
    print(f"Validation Results:")
    print(f"mAP50: {val_results.box.map50:.4f}")
    print(f"mAP50-95: {val_results.box.map:.4f}")

    return val_results
```

- **mAP50:** Mean Average Precision at IoU=0.5.
- **mAP50-95:** Mean AP across different IoU thresholds (0.5 to 0.95).

2.5 Visualizing Predictions

```
def visualize_predictions(model, num_images=3, conf=0.25):
    # Get random validation images
    val_dir = os.path.join(YOLO_DIR, "images", "val")
    val_images = list(Path(val_dir).glob("*.jpg"))

    if len(val_images) == 0:
        print("No validation images found!")
        return

    # Select random images
    random.shuffle(val_images)
    selected_images = val_images[:min(num_images, len(val_images))]

    # Display predictions
    plt.figure(figsize=(15, 5 * len(selected_images)))

    for i, img_path in enumerate(selected_images):
        # Run prediction
        results = model.predict(img_path, conf=conf)[0]

        # Get annotated image
        annotated_img = results.plot()
        annotated_img = cv2.cvtColor(annotated_img, cv2.COLOR_BGR2RGB)

        # Display image
        plt.subplot(len(selected_images), 1, i+1)
        plt.imshow(annotated_img)
        plt.title(f"Predictions on {img_path.name}")
        plt.axis('off')
```

```
    # Print detected objects
    boxes = results.boxes
    print(f"\nDetections in {img_path.name}:")
    for box in boxes:
        cls_id = int(box.cls.item())
        conf = box.conf.item()
        class_name = class_names[cls_id]
        print(f"  - {class_name}: {conf:.2f}")

plt.tight_layout()
plt.show()
```

- **Displays:** Images with bounding boxes and class names.

- **Confidence filter:** Removes low-probability predictions.

2.6 Generating Captions

```python
from collections import Counter

def generate_caption_from_boxes(boxes, class_names, conf_threshold=0.2):
    labels = [class_names[int(box.cls.item())] for box in boxes if box.conf.item() >= conf_threshold]
    counts = Counter(labels)
    if not counts:
        return "No significant objects detected."

    parts = [f"{v} {k.lower() + ('s' if v > 1 else '')}" for k, v in counts.items()]
    caption = "This image shows " + ', '.join(parts[:-1]) + (' and ' + parts[-1] if len(parts) > 1 else parts[0]) + "."
    return caption
```

- **Purpose:** Provides a human-readable summary of detected objects.
- **Logic:** Counts and groups similar objects, constructs a descriptive sentence.

2.7 Predicting a Single Image

```python
def predict_image(image_path, model, conf_threshold=0.25):
    # Run prediction
    results = model.predict(image_path, conf=conf_threshold)[0]

    # Get annotated image
    annotated_img = results.plot()
    annotated_img = cv2.cvtColor(annotated_img, cv2.COLOR_BGR2RGB)

    # Display image
    plt.figure(figsize=(10, 8))
    plt.imshow(annotated_img)
    plt.title("Object Detection Results")
    plt.axis('off')
    plt.show()

    # Print detected objects
    boxes = results.boxes
    print("Detected Objects:")
    detections = []
    for box in boxes:
        cls_id = int(box.cls.item())
        conf = box.conf.item()
        class_name = class_names[cls_id]
        print(f"  - {class_name}: {conf:.2f}")
        detections.append(f"{class_name}: {conf:.2f}")

    # Generate and print caption
    caption = generate_caption_from_boxes(boxes, class_names, conf_threshold)
    print("\nGenerated Caption:")
```

- **Loads image:** Draws predictions with matplotlib.
- **Displays:** Detected objects and the generated caption.

2.8 File Upload Inference (Jupyter)

```
def inference_from_upload():
    upload = widgets.FileUpload(accept='.jpg,.jpeg,.png', multiple=False, description='Upload Image')
    conf_slider = widgets.FloatSlider(value=0.25, min=0.1, max=0.9, step=0.05, description='Confidence:')
    button = widgets.Button(description='Detect Objects')
    output = widgets.Output()

    display(upload, conf_slider, button, output)

    def on_button_click(b):
        with output:
            output.clear_output()
            if not upload.value:
                print("Please upload an image first!")
                return

            # Save uploaded file correctly
            file_info = upload.value[0]
            file_name = file_info['name']
            file_data = file_info['content']
            temp_path = f"temp_{file_name}"
            with open(temp_path, 'wb') as f:
                f.write(file_data)

            predict_image(temp_path, model, conf_slider.value)
            os.remove(temp_path)
    button.on_click(on_button_click)
```

- **ipywidgets:** Allows interactive upload and real-time object detection in notebooks.
- **Usage:** Ideal for demo environments and non-programmers.

2.9 Live Webcam Detection

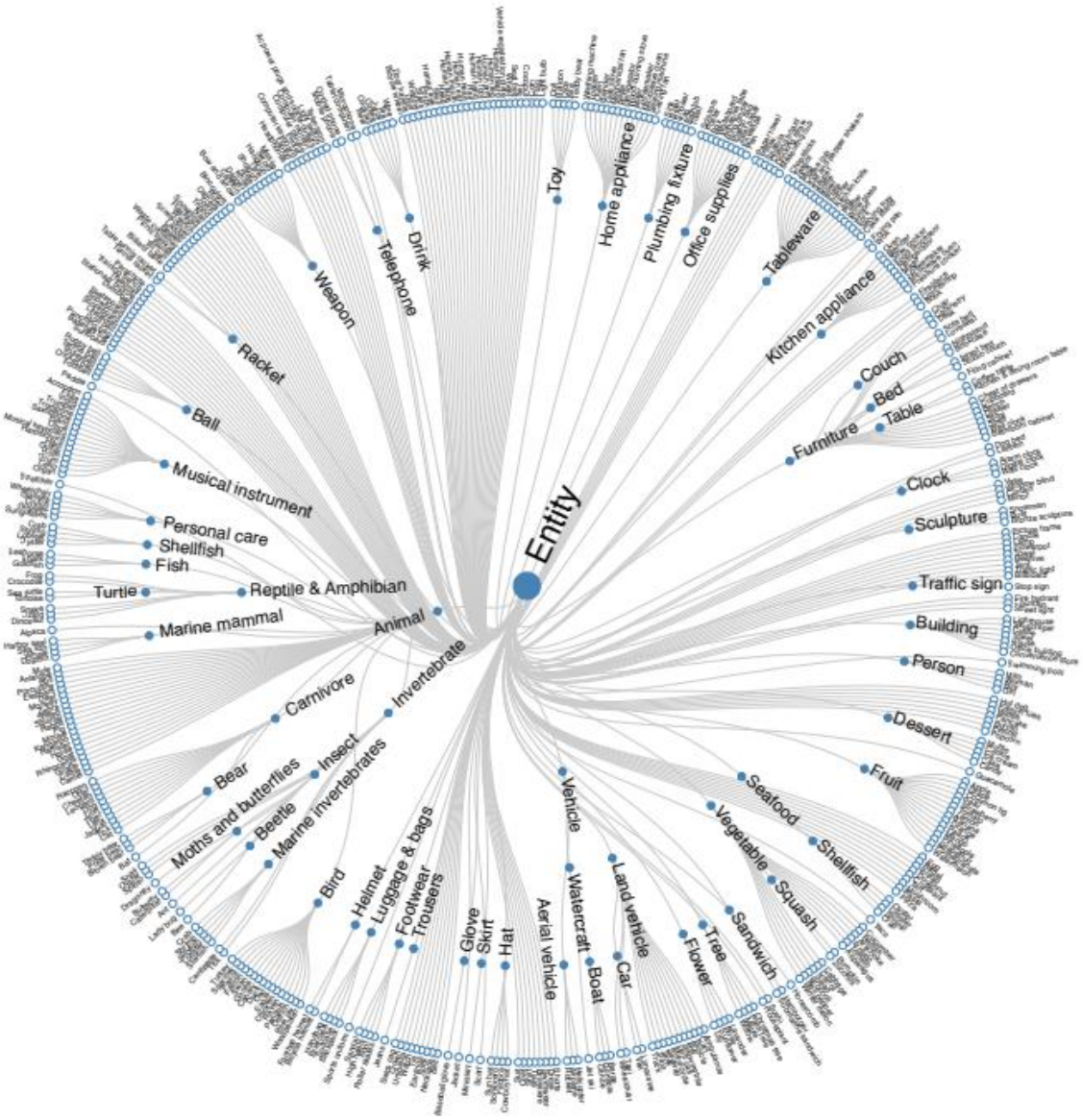
```
def start_webcam_detection(model, conf_threshold=0.25):
    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        print("Error: Could not open webcam.")
        return
    print("Webcam detection started. Press 'q' to exit.")
    while True:
        ret, frame = cap.read()
        if not ret:
            print("Error: Could not read frame.")
            break
        results = model.predict(frame, conf=conf_threshold)[0]
        annotated_frame = results.plot()

        # OPTIONAL: Add caption to frame
        caption = generate_caption_from_boxes(results.boxes, class_names, conf_threshold)
        cv2.putText(annotated_frame, caption, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0,255,0), 2)

        cv2.imshow('YOLOv8 Object Detection', annotated_frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()
```

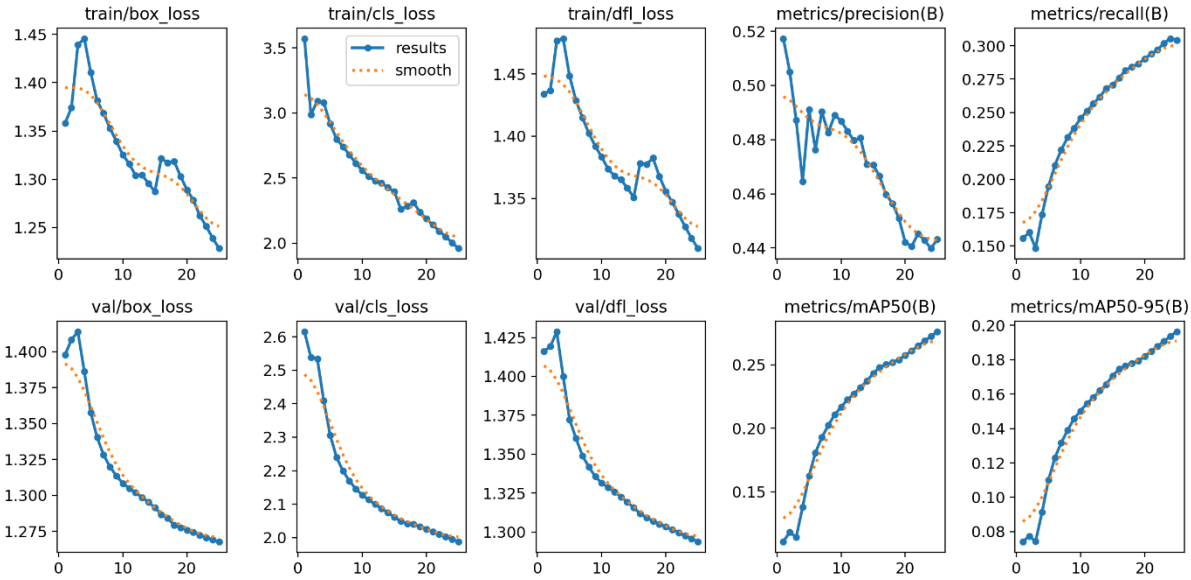
- **cv2.VideoCapture:** Captures live video stream.
- **Real-time inference:** Applies YOLO to each frame.

3. Dataset Classes:



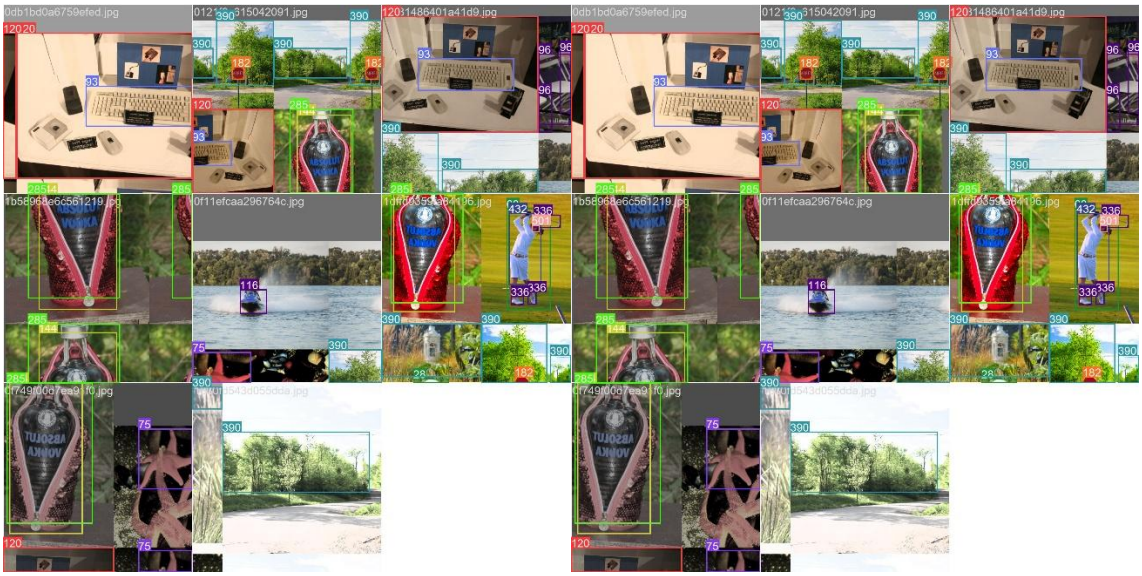
4. Batch Training:

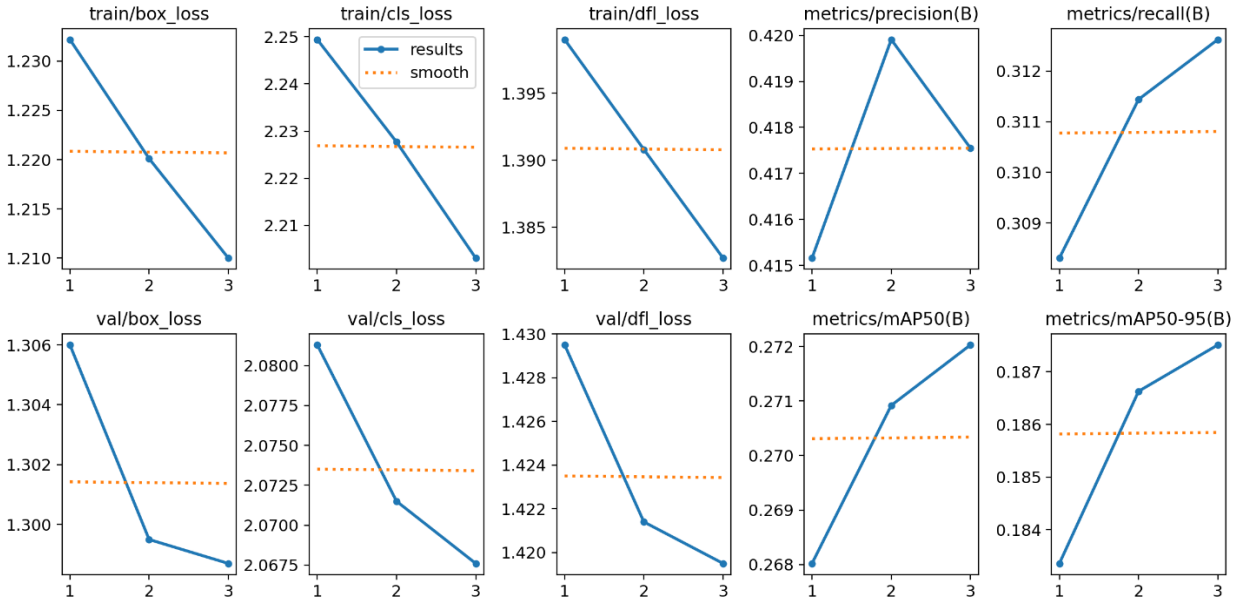
Screenshots for batch training of our Object Recognition and Numeration model.



5. Improving accuracy with Finetuning:

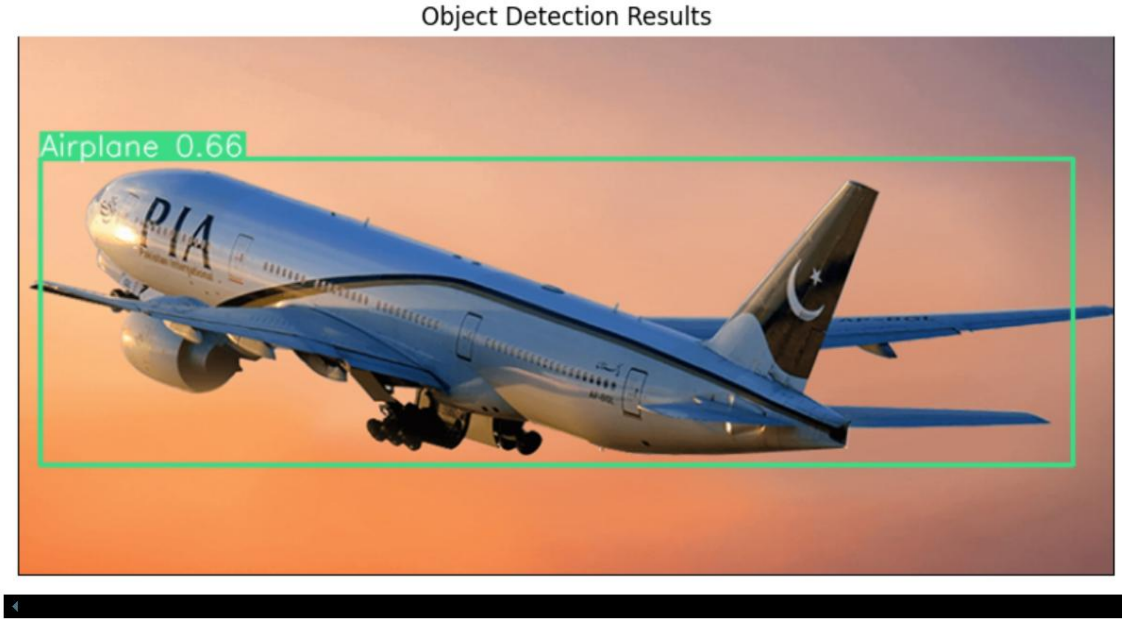
Improving the accuracy of object detection of our model through finetuning.





6. Image Output:

Screenshot of the image detection and numeration of object/humans done by our model.

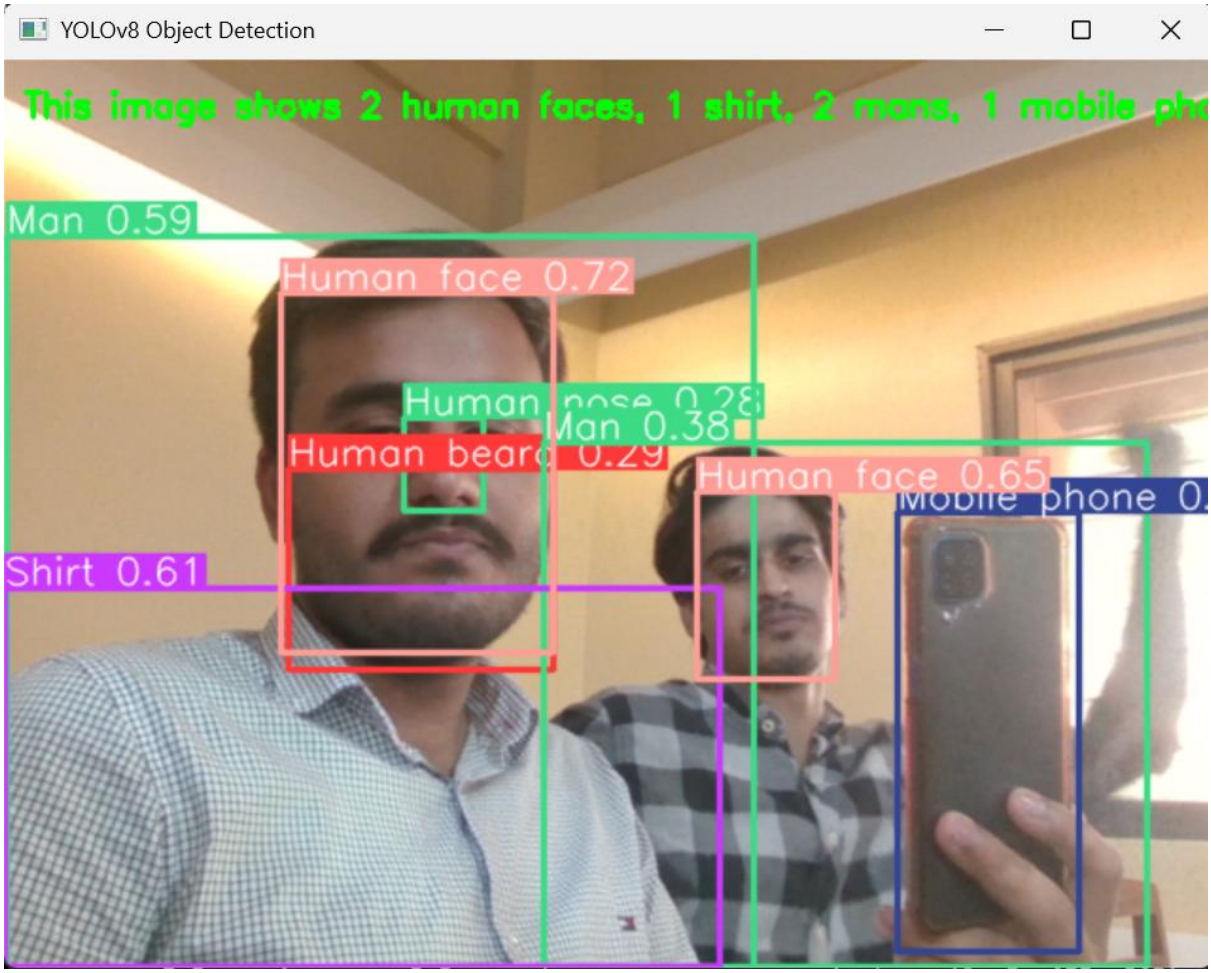



```
Detected Objects:
  - Airplane: 0.66

Generated Caption:
This image shows 1 airplane.
```

7. Real Time Output:

Screenshot of the real time detection and numeration of object/humans done by our model.



8. Conclusion

This object detection system leverages the power of YOLOv8 to build a versatile, accurate, and interactive pipeline. It supports training with custom datasets, evaluation, visual feedback, and real-time inference — making it suitable for both research and deployment in real-world applications.