Project ARI3129 - Object Detection & Localisation using Yolov8 *(Jupyter Notebook #2)*

Name: Andrea Filiberto Lucas
ID No: 0279704L

Automated Dataset Management with Roboflow and Folder Organization

This script automates the process of managing a dataset using Roboflow. It creates necessary directories, checks for installed dependencies, installs them if missing, and downloads the dataset. It also organizes the dataset into a structured folder hierarchy, ensuring everything is ready for further use.

```
import os
import subprocess
import shutil

# Constants for colored output
COLORS = {
    "green": "\033[92m",  # Green text
    "red": "\033[91m",    # Red text
    "reset": "\033[0m"    # Reset to default color
}

# Define the path to the Versions folder and the target subfolder
versions_path = os.path.abspath(os.path.join("..", "Versions"))
target_subfolder = os.path.join(versions_path, "MDD-AFL-Yolov8")

# Check if the Versions folder exists, if not, create it
if not os.path.exists(versions_path):
    os.makedirs(versions_path)
    print(f"[{COLORS['green']}✔{COLORS['reset']}] Folder created at:
{versions_path}")

# Check if the MDD-AFL-Yolov8 subfolder exists
if os.path.exists(target_subfolder):
    print(f"[{COLORS['green']}✔{COLORS['reset']}] The subfolder
'{target_subfolder}' already exists. Skipping download!")
else:
    # Check if roboflow is installed
    if importlib.util.find_spec("roboflow") is not None:
# type: ignore
        print(f"[{COLORS['green']}✔{COLORS['reset']}] Roboflow is
already installed!")
```

```
    else:
        # Install roboflow using pip
        try:
            subprocess.check_call(["pip", "install", "roboflow"])
            print(f"[{COLORS['green']}✔{COLORS['reset']}] Roboflow
successfully installed!")
        except subprocess.CalledProcessError as e:
            print(f"[{COLORS['red']}✖{COLORS['reset']}]Failed to
install Roboflow. Please check your setup.")
            raise e

    # Import and use Roboflow
    from roboflow import Roboflow
# type: ignore

    # Prompt the user for their API key
    print("Please enter your Roboflow API key to download the
dataset...")
    api_key = input("Please enter your Roboflow API key: ")

    # Initialize Roboflow with the provided API key
    rf = Roboflow(api_key=api_key)

    # Retrieve project and version
    project = rf.workspace("advanced-cv").project("maltese-domestic-
dataset")
    version = project.version(1)

    # Download the dataset
    dataset = version.download("yolov8")

    current_folder = os.getcwd()  # Get the current working directory
    original_folder = os.path.join(current_folder, "Maltese-Domestic-
Dataset--1")
    renamed_folder = os.path.join(current_folder, "MDD-AFL-Yolov8")
    target_folder = os.path.join(versions_path, "MDD-AFL-Yolov8")

    # Check if the original folder exists
    if os.path.exists(original_folder):
        # Rename the folder
        os.rename(original_folder, renamed_folder)

        # Move the renamed folder to ../Versions/
        shutil.move(renamed_folder, target_folder)
        print(f"[{COLORS['green']}✔{COLORS['reset']}] Folder
downloaded to: {target_folder}")
    else:
        print(f"[{COLORS['red']}✖{COLORS['reset']}]Folder
'{original_folder}' does not exist. No action taken.")
```

```
[✔] The subfolder '/Users/afl/Documents/University/Year 3/Lectures/SEM1/Advanced CV/Assignments/ARI3129-MDD/Versions/MDD-AFL-Yolov8' already exists. Skipping download!
```

Automated Library Installer in Python

This script automates checking and installing libraries from a JSON file. It verifies installations, installs missing libraries via `pip`, and provides clear, colored output for success or errors. With built-in error handling and preloaded common libraries, it simplifies dependency management in Python projects.

```
import json
import importlib.util

# Path to the JSON file
lib_file_path = os.path.join("..", "Libraries", "Task2_AFL_Lib.json")

# Read the libraries from the JSON file
try:
    with open(lib_file_path, 'r') as file:
        libraries = json.load(file)
except FileNotFoundError:
    print(f"{COLORS['red']}Error: Library file not found at {lib_file_path}{COLORS['reset']}")
    exit(1)
except json.JSONDecodeError:
    print(f"{COLORS['red']}Error: Failed to decode JSON from the library file.{COLORS['reset']}")
    exit(1)

# Function to check and install libraries
def check_and_install_libraries(libraries):
    for lib, import_name in libraries.items():
        # Check if the library is installed by checking its module spec
        if importlib.util.find_spec(import_name) is not None:
            print(f"[{COLORS['green']}✔{COLORS['reset']}] Library '{lib}' is already installed.")
        else:
            # If the library is not found, try to install it
            print(f"[{COLORS['red']}✖{COLORS['reset']}] Library '{lib}' is not installed. Installing...")
            try:
                subprocess.check_call(["pip", "install", lib])
                print(f"[{COLORS['green']}✔{COLORS['reset']}] Successfully installed '{lib}'.")
            except subprocess.CalledProcessError:
                print(f"[{COLORS['red']}✖{COLORS['reset']}] Failed to install '{lib}'. Please install it manually.")
```

```
# Execute the function to check and install libraries
check_and_install_libraries(libraries)

# Import necessary libraries
import time
import torch
import random
import pandas as pd
#type: ignore
import cv2
#type: ignore
import matplotlib.pyplot as plt
#type: ignore
import seaborn as sns
#type: ignore
import numpy as np
#type: ignore
import matplotlib.patches as patches
#type: ignore
import concurrent.futures
from concurrent.futures import ThreadPoolExecutor, as_completed
from tqdm import tqdm
#type: ignore
from ultralytics import YOLO
#type: ignore
from IPython.display import Image, display
#type: ignore
```

```
[✔] Library 'opencv-python' is already installed.
[✔] Library 'matplotlib' is already installed.
[✔] Library 'tqdm' is already installed.
[✔] Library 'ultralytics' is already installed.
[✔] Library 'torch' is already installed.
```

Determine the Available Device for Computation

This function identifies and selects the best available device for running deep learning models, prioritizing hardware acceleration via CUDA (GPU) or MPS (Metal Performance Shaders). If no hardware acceleration is available, it defaults to the CPU and notifies the user.

```
# Determine the available device: CUDA, MPS, or CPU.
def determine_device():
    if torch.cuda.is_available():
        device = "cuda"
        print(f"Using GPU: {torch.cuda.get_device_name(0)}")
    elif torch.backends.mps.is_available():
        device = "mps"
```

```
        print("Using MPS (Metal Performance Shaders) for
acceleration.")
    else:
        device = "cpu"
        print("Using CPU. Consider enabling GPU or MPS for faster
training.")
    return device
```

Training process for Yolov8 object detector.

Experiment Configuration for YOLOv8

Defines the path to the `data.yaml` file and key hyperparameters for conducting experiments with YOLOv8. These settings include the number of epochs, image size, batch size, and a customizable experiment name for version tracking and organization.

```
# Path to data.yaml
data_yaml_path =
os.path.join(os.path.abspath(os.path.join(os.getcwd(), os.pardir)),
"Versions", "MDD-AFL-Yolov8", "data.yaml")

# Prompt user for hyperparameters
try:
    epochs = int(input("Enter the number of epochs: "))
    experiment_number = int(input("Enter the experiment number: "))
    experiment_name = f"MDD-AFL-Yolov8_v{experiment_number}"
except ValueError:
    print("Invalid input. Please enter valid numbers for epochs and
experiment number.")
    raise

# Display selected parameters
print(f"Selected epochs: {epochs}")
print(f"Experiment name: {experiment_name}")

# Hyperparameters
imgsz = 640
batch_size = 8
```

```
Selected epochs: 100
Experiment name: MDD-AFL-Yolov8_v7
```

YOLOv8 Training and Validation Pipeline

This script initializes and trains a YOLOv8 model from scratch using a specified architecture (`yolov8s.yaml`, `yolov8n.yaml`). Key features include:

- **Timer**: Measures and reports total training time in hours, minutes, and seconds.

- **Training**: Configurable hyperparameters for epochs, image size, batch size, and experiment naming.

Error handling is incorporated for key stages, including model creation, training, and validation, ensuring robustness in experimentation workflows.

v1 - First Test (With Prototype Code)

This script trains a YOLOv8 model from scratch using a specified configuration file (yolov8n.yaml) and dataset configuration (data.yaml) located in the Versions/MDD-AFL-Yolov8 directory. The model is trained for 50 epochs with an image size of 640, a batch size of 8, and is named MDD-AFL-Yolov8_v1. After training, the model is evaluated on a validation dataset to generate performance metrics, such as precision, recall, and mAP.

```
# Path to data.yaml (relative to Yolov8_AFL.ipynb)
data_yaml_path =
os.path.join(os.path.abspath(os.path.join(os.getcwd(), os.pardir)),
"Versions", "MDD-AFL-Yolov8", "data.yaml")

# Create or load a YOLOv8 model (from scratch)
model = YOLO("yolov8n.yaml")  # specify your chosen architecture
(e.g., yolov8n.yaml)

# Train
model.train(
    data=data_yaml_path,
    epochs=50,
    imgsz=640,
    batch=8,
    name="MDD-AFL-Yolov8_v1",
    pretrained=False  # to train from scratch
)

# Evaluate
metrics = model.val()
print("Validation Metrics:", metrics)
```

v2 - First Good Result (Using Yolov8s)

This script trains a YOLOv8 model using the yolov8s.yaml configuration and a specified dataset for 55 epochs. It records the training duration and logs errors if the configuration file is missing or issues occur during training. After training, the model is evaluated on a validation dataset to calculate metrics such as precision, recall, and mAP, providing insights into its performance.

```
# Timer start
start_time = time.time()

# Create or load a YOLOv8 model (from scratch)
try:
    model = YOLO("yolov8s.yaml")  # specify architecture
```

```
except FileNotFoundError:
    print("Error: YOLOv8 configuration file 'yolov8s.yaml' not found.
Check your setup.")
    raise

# Train
try:
    model.train(
        data=data_yaml_path,
        epochs=55,
        imgsz=imgsz,
        batch=batch_size,
        name=experiment_name,
        pretrained=False,
    )
except Exception as e:
    print(f"Error during training: {e}")
    raise

# Timer end
end_time = time.time()
# Calculate elapsed time
elapsed_time = end_time - start_time
# Convert seconds to hours, minutes, and seconds
hours = int(elapsed_time // 3600)
minutes = int((elapsed_time % 3600) // 60)
seconds = int(elapsed_time % 60)
print(f"Training completed in {hours}h {minutes}m {seconds}s.")

# Evaluate
try:
    metrics = model.val()
    print("Validation Metrics:", metrics)
except Exception as e:
    print(f"Error during validation: {e}")
    raise
```

v3 - Results in Overfitting

This script trains a YOLOv8 model from scratch using the specified architecture (yolov8l.yaml) and dataset configuration. It tracks training time, performs training for 100 epochs with defined parameters like image size and batch size, and evaluates the model on a validation dataset to generate key metrics. Error handling is included for missing files or issues during training and validation, ensuring a robust workflow.

```
# Timer start
start_time = time.time()

# Create or load a YOLOv8 model (from scratch)
```

```
try:
    model = YOLO("yolov8l.yaml")  # specify architecture
except FileNotFoundError:
    print("Error: YOLOv8 configuration file 'yolov8n.yaml' not found.
Check your setup.")
    raise

# Train
try:
    model.train(
        data=data_yaml_path,
        epochs=100,
        imgsz=imgsz,
        batch=batch_size,
        name=experiment_name,
        pretrained=False,
    )
except Exception as e:
    print(f"Error during training: {e}")
    raise

# Timer end
end_time = time.time()
# Calculate elapsed time
elapsed_time = end_time - start_time
# Convert seconds to hours, minutes, and seconds
hours = int(elapsed_time // 3600)
minutes = int((elapsed_time % 3600) // 60)
seconds = int(elapsed_time % 60)
print(f"Training completed in {hours}h {minutes}m {seconds}s.")

# Evaluate
try:
    metrics = model.val()
    print("Validation Metrics:", metrics)
except Exception as e:
    print(f"Error during validation: {e}")
    raise
```

v4 - Added Early Stopping

This script trains a YOLOv8 model using the yolov8n.yaml configuration for a maximum of 200 epochs, with early stopping triggered after 31 epochs due to a patience threshold of 10. The model is trained from scratch with a dropout rate of 0.3 to improve generalization. Training time is tracked, and any errors during training or validation are logged. After training, the model is evaluated on a validation dataset to generate performance metrics, such as precision, recall, and mAP.

```
# Timer start
start_time = time.time()

# Create or load a YOLOv8 model (from scratch)
try:
    model = YOLO("yolov8n.yaml")  # specify architecture
except FileNotFoundError:
    print("Error: YOLOv8 configuration file 'yolov8n.yaml' not found.
Check your setup.")
    raise

# Train
try:
    model.train(
        data=data_yaml_path,
        epochs=200, # Stopped at 31 epochs (early stopping)
        imgsz=imgsz,
        batch=batch_size,
        name=experiment_name,
        pretrained=False,
        dropout=0.3,
        patience=10,
    )
except Exception as e:
    print(f"Error during training: {e}")
    raise

# Timer end
end_time = time.time()
# Calculate elapsed time
elapsed_time = end_time - start_time
# Convert seconds to hours, minutes, and seconds
hours = int(elapsed_time // 3600)
minutes = int((elapsed_time % 3600) // 60)
seconds = int(elapsed_time % 60)
print(f"Training completed in {hours}h {minutes}m {seconds}s.")

# Evaluate
try:
    metrics = model.val()
    print("Validation Metrics:", metrics)
except Exception as e:
    print(f"Error during validation: {e}")
    raise
```

v5 - Started using Learning Rate

This script trains a YOLOv8 model using the yolov8n.yaml configuration with specified hyperparameters, including a cosine learning rate scheduler, a dropout rate of 0.3, and a patience threshold of 10 for early stopping. Training is performed for a predefined number of

epochs, with plots generated to visualize progress. The model leverages a custom learning rate range (lr0=0.001 to lrf=0.00001) and dynamically selects the appropriate device for computation. Training time is tracked, and after completion, the model is evaluated on a validation dataset to output metrics such as precision, recall, and mAP. Errors during any stage are logged to ensure robustness.

```
# Timer start
start_time = time.time()

# Create or load a YOLOv8 model (from scratch)
try:
    model = YOLO("yolov8s.yaml")  # specify architecture
except FileNotFoundError:
    print("Error: YOLOv8 configuration file 'yolov8s.yaml' not found.
Check your setup.")
    raise

# Train
try:
    model.train(
        data=data_yaml_path,
        epochs=epochs,
        imgsz=imgsz,
        batch=batch_size,
        name=experiment_name,
        pretrained=False,
        plots=True,
        device=determine_device(),
        patience=10,
        dropout=0.3,
        cos_lr=True,
        lr0=0.001,
        lrf=0.00001,
    )
except Exception as e:
    print(f"Error during training: {e}")
    raise

# Timer end
end_time = time.time()
# Calculate elapsed time
elapsed_time = end_time - start_time
# Convert seconds to hours, minutes, and seconds
hours = int(elapsed_time // 3600)
minutes = int((elapsed_time % 3600) // 60)
seconds = int(elapsed_time % 60)
print(f"Training completed in {hours}h {minutes}m {seconds}s.")

# Evaluate
try:
```

```
    metrics = model.val()
    print("Validation Metrics:", metrics)
except Exception as e:
    print(f"Error during validation: {e}")
    raise
```

v6 - Best Performace (Result)

This script trains a YOLOv8 model using the yolov8n.yaml configuration with optimized hyperparameters, including a cosine learning rate scheduler (lr0=0.0001 to lrf=0.00001), a dropout rate of 0.4, and a patience threshold of 10 for early stopping. Training is performed for the specified number of epochs on the available device, and detailed plots are generated to monitor progress. The script tracks the total training duration and, after completion, evaluates the model on a validation dataset to produce performance metrics such as precision, recall, and mAP. This configuration yielded the best results in terms of model performance

```
# Timer start
start_time = time.time()

# Create or load a YOLOv8 model (from scratch)
try:
    model = YOLO("yolov8s.yaml")  # specify architecture
except FileNotFoundError:
    print("Error: YOLOv8 configuration file 'yolov8s.yaml' not found.
Check your setup.")
    raise

# Train
try:
    model.train(
        data=data_yaml_path,
        epochs=epochs,
        imgsz=imgsz,
        batch=batch_size,
        name=experiment_name,
        pretrained=False,
        plots=True,
        device=determine_device(),
        patience=10,
        dropout=0.4,
        cos_lr=True,
        lr0=0.0001,
        lrf=0.00001,
    )
except Exception as e:
    print(f"Error during training: {e}")
    raise

# Timer end
```

```
end_time = time.time()
# Calculate elapsed time
elapsed_time = end_time - start_time
# Convert seconds to hours, minutes, and seconds
hours = int(elapsed_time // 3600)
minutes = int((elapsed_time % 3600) // 60)
seconds = int(elapsed_time % 60)
print(f"Training completed in {hours}h {minutes}m {seconds}s.")

# Evaluate
try:
    metrics = model.val()
    print("Validation Metrics:", metrics)
except Exception as e:
    print(f"Error during validation: {e}")
    raise
```

Results

Ultralytics Results

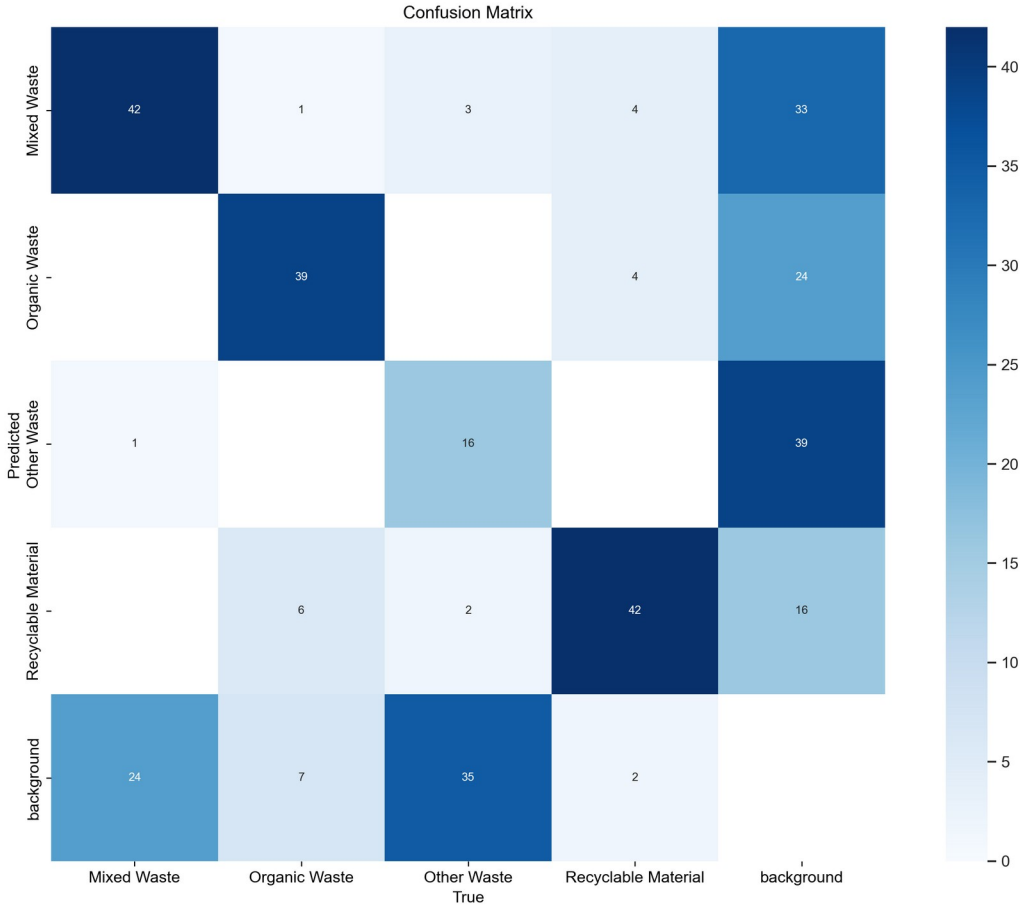
This script facilitates the selection and evaluation of YOLOv8 detection results stored in the `runs/detect` directory. It identifies folders ending with "R," lets the user choose one, and displays evaluation images within the selected folder (excluding "val_batch" files). Additionally, it checks the corresponding "non-R" folder for a `results.png` file and displays it if found.

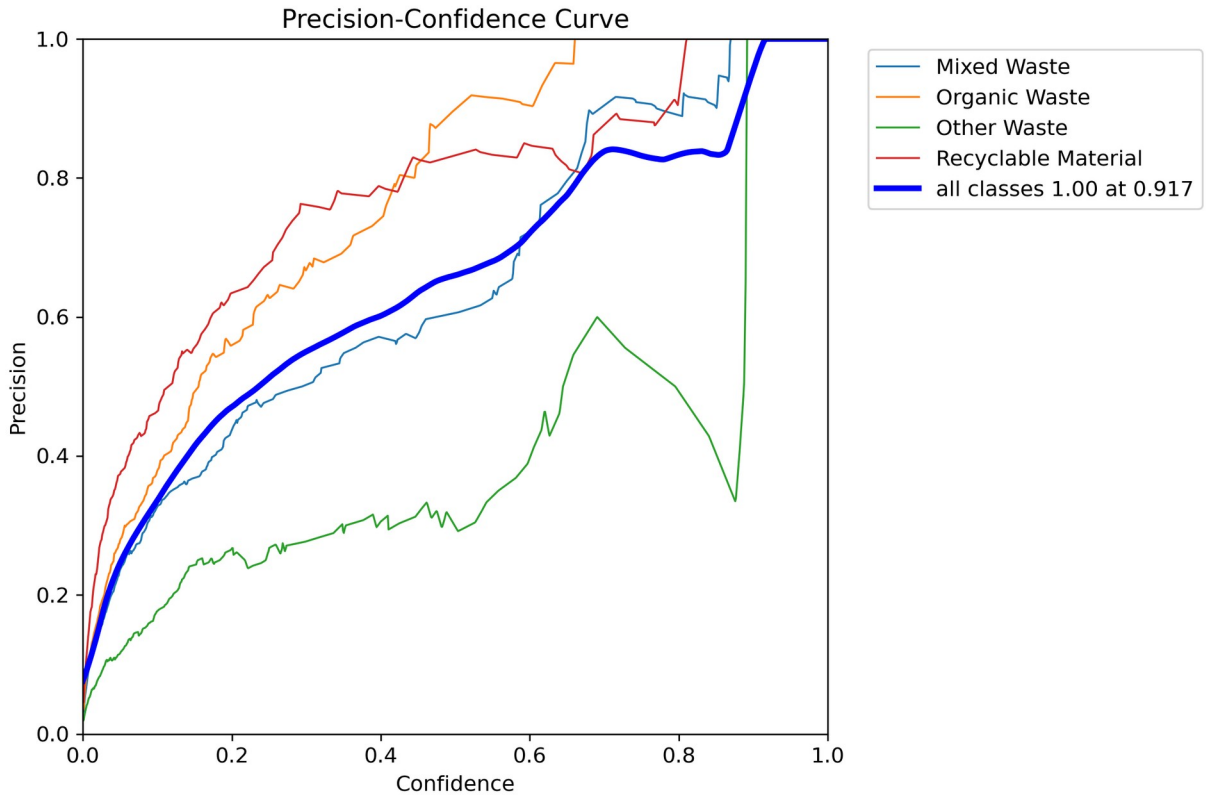
```
# Path to the "runs/detect" folder
detect_folder_path = os.path.join(os.getcwd(), "runs", "detect")

# Get all folders ending with "R" that also contain "AFL" and sort
them in ascending order
folders_ending_with_R = sorted([
    folder for folder in os.listdir(detect_folder_path)
    if folder.endswith("R")
    and "AFL" in folder  # Folder must contain "AFL"
    and os.path.isdir(os.path.join(detect_folder_path, folder))
])

# Check if there are any folders to choose from
if not folders_ending_with_R:
    print("No Results folders with 'AFL' found!")
else:
    # Display available folders for user selection
    print("Available folders:")
    for i, folder in enumerate(folders_ending_with_R, start=1):
        print(f"{i}. {folder}")

    # Prompt the user to select a folder
    try:
```

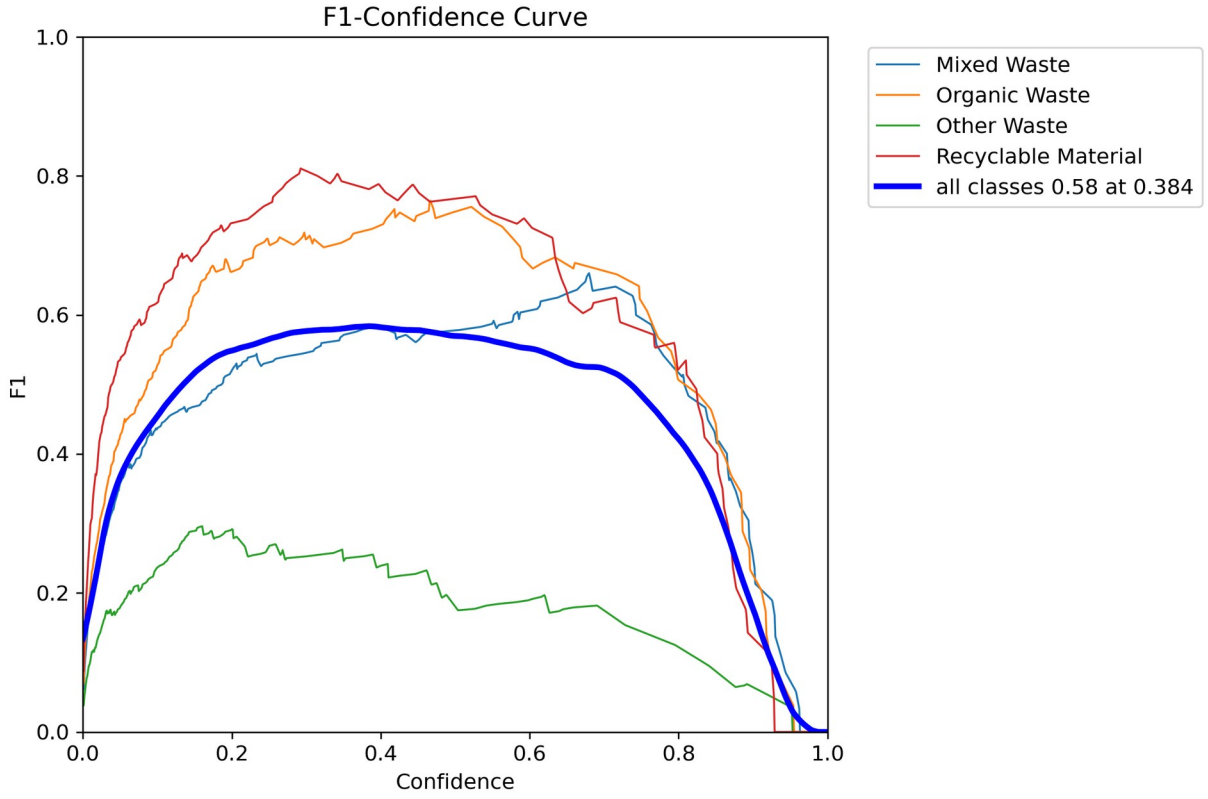
```
        choice = int(input("Enter the number corresponding to the
folder you want to use: ")) - 1
        if choice < 0 or choice >= len(folders_ending_with_R):
            raise IndexError
        selected_folder = folders_ending_with_R[choice]
    except (ValueError, IndexError):
        print("Invalid selection. Please restart and try again.")
        raise

    print(f"\nSelected folder: {selected_folder}")
    # Path to the selected folder
    eval_folder_path = os.path.join(detect_folder_path,
selected_folder)

    # List of evaluation images excluding "val_batch" files
    evaluation_images = [
        img for img in os.listdir(eval_folder_path)
        if img.lower().endswith(('.png', '.jpg')) and "val_batch" not
in img
    ]

    # Display images
    if not evaluation_images:
        print("No evaluation images found in the selected folder.")
    else:
        print("\nDisplaying evaluation images:")
        for img_file in evaluation_images:
            img_path = os.path.join(eval_folder_path, img_file)
            display(Image(filename=img_path))

    # Path to the non-R version of the selected folder
    non_r_folder = selected_folder.rstrip("R")
    non_r_folder_path = os.path.join(detect_folder_path, non_r_folder)

    # Check and display "results.png" if it exists
    results_png_path = os.path.join(non_r_folder_path, "results.png")
    if os.path.exists(results_png_path):
        print("\nDisplaying results.png from the non-R version of the
folder:")
        display(Image(filename=results_png_path))
    else:
        print("No results.png found in the non-R version of the
selected folder.")
```

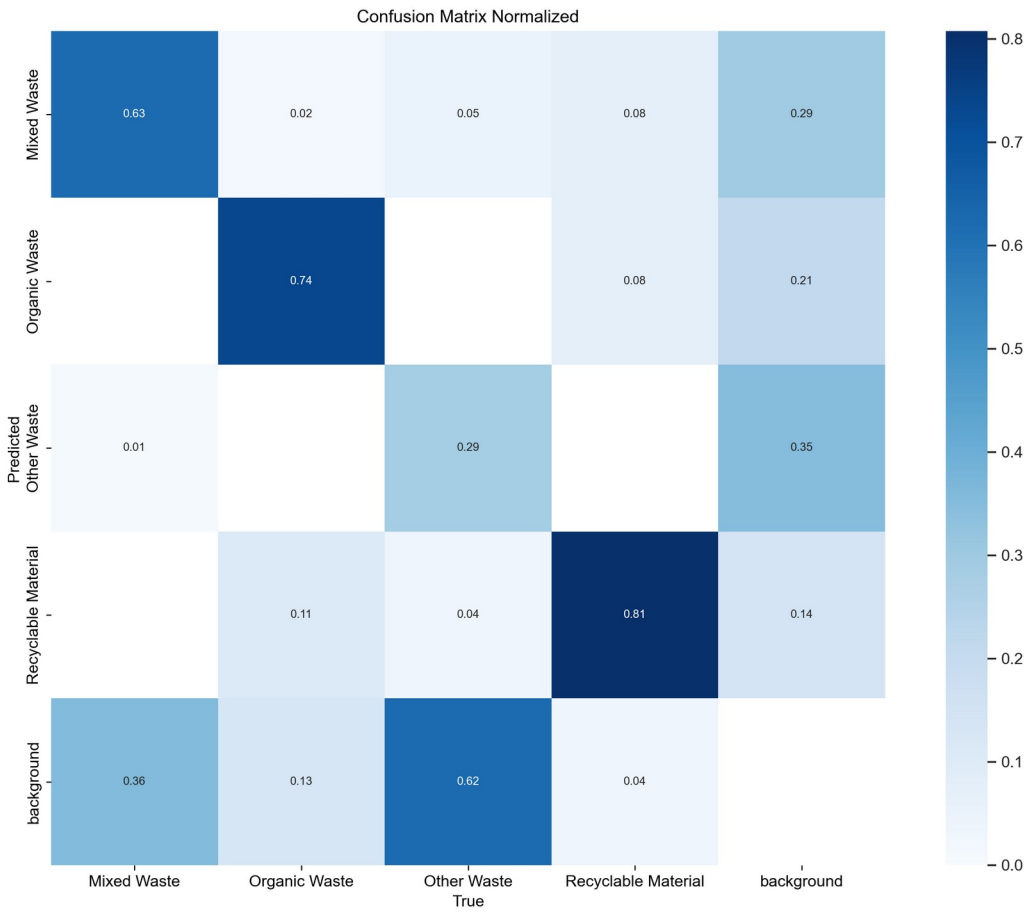
```
Available folders:
1. MDD-AFL-Yolov8_v1R
2. MDD-AFL-Yolov8_v2R
3. MDD-AFL-Yolov8_v3R
4. MDD-AFL-Yolov8_v4R
5. MDD-AFL-Yolov8_v5R
```

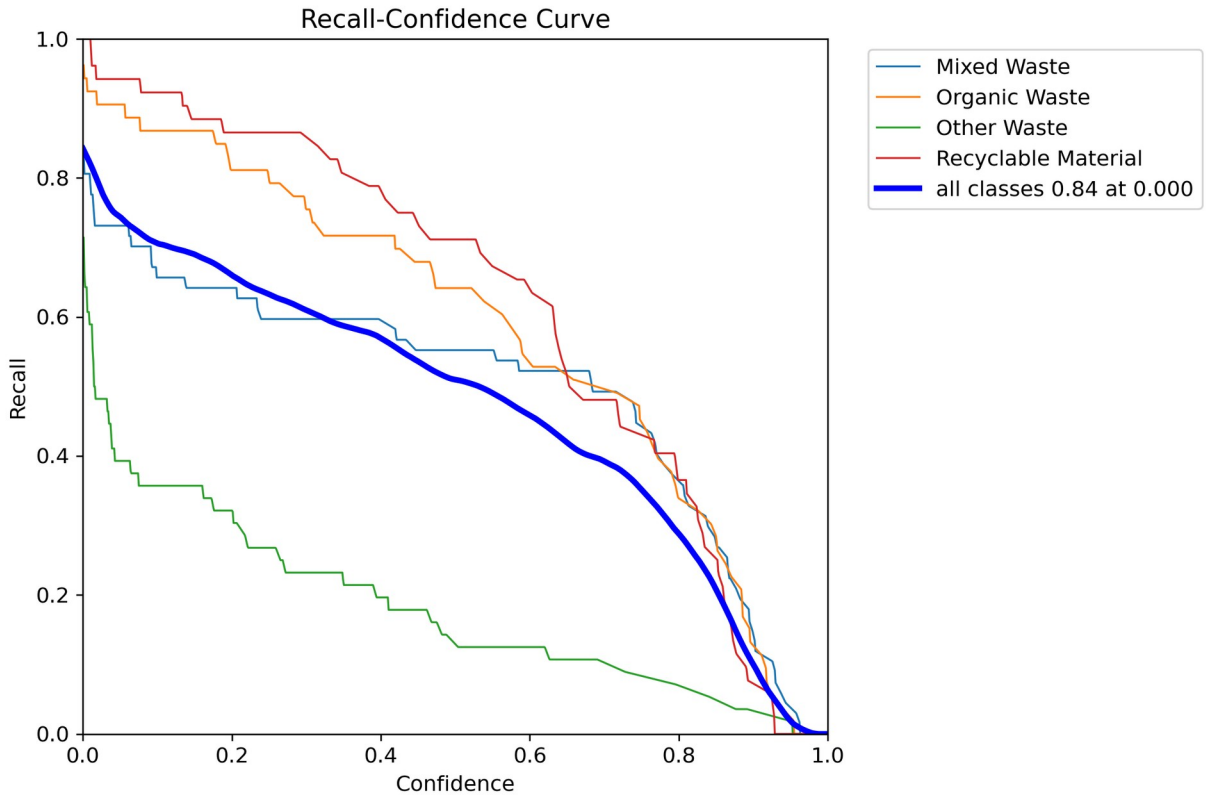
```
6. MDD-AFL-Yolov8_v6R

Selected folder: MDD-AFL-Yolov8_v6R

Displaying evaluation images:
```

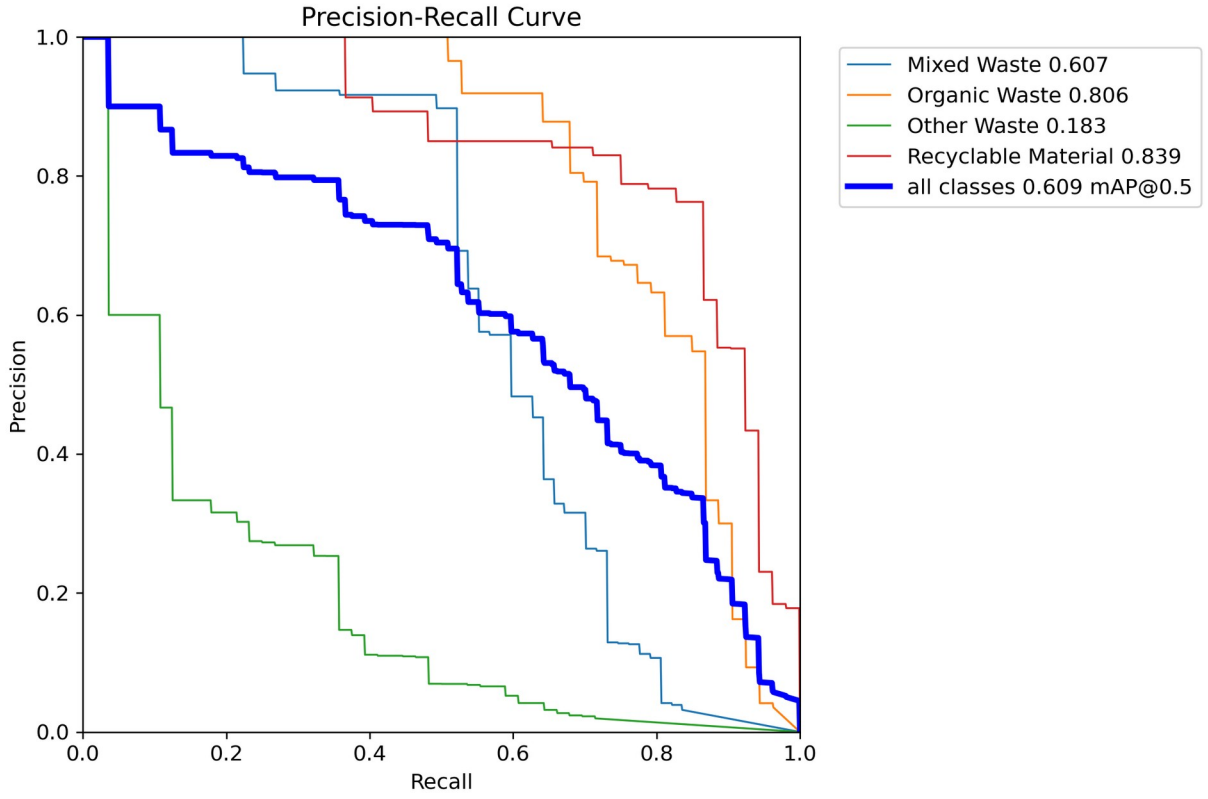



Precision-Confidence Curve
Precision
Confidence
0.0
0.2
0.4
0.6
0.8
1.0
Mixed Waste
Organic Waste
Other Waste
Recyclable Material
all classes 1.00 at 0.917

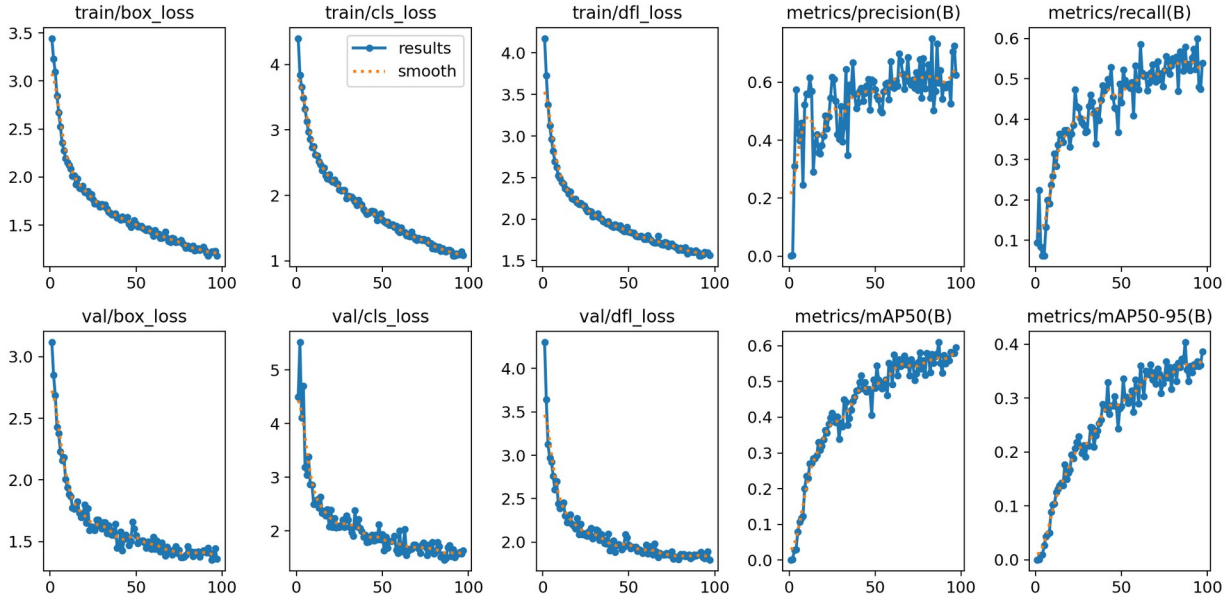

F1-Confidence Curve
F1
Confidence
0.0
0.2
0.4
0.6
0.8
1.0
Mixed Waste
Organic Waste
Other Waste
Recyclable Material
all classes 0.58 at 0.384


Confusion Matrix Normalized
Mixed Waste
Organic Waste
Other Waste
Recyclable Material
background
Predicted
True
0.63
0.02
0.05
0.08
0.29
0.74
0.08
0.21
0.01
0.29
0.35
0.11
0.04
0.81
0.14
0.36
0.13
0.62
0.04
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0


Recall-Confidence Curve
Recall
Confidence
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Mixed Waste
Organic Waste
Other Waste
Recyclable Material
all classes 0.84 at 0.000


Precision-Recall Curve
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Mixed Waste 0.607
Organic Waste 0.806
Other Waste 0.183
Recyclable Material 0.839
all classes 0.609 mAP@0.5

```
Displaying results.png from the non-R version of the folder:
```



YOLOv8 Version Selection and Weights Path Identification

This script lists all available YOLOv8 versions in the `runs/detect` directory (excluding folders ending with "R"), sorts them, and prompts the user to select one for testing. After a valid selection, it identifies the path to the corresponding weights file (`best.pt`) for the chosen version and confirms the selection. Robust error handling ensures smooth execution by addressing cases of missing versions or invalid user input.

```
# Define the detect path
detect_path = os.path.join(os.getcwd(), "runs", "detect")

# Get all valid versions in the "runs/detect" directory
# Ensure folders contain "AFL" in their name and do not end with "R"
available_versions = sorted(
    [
        folder
        for folder in os.listdir(detect_path)
        if os.path.isdir(os.path.join(detect_path, folder))
        and "AFL" in folder  # Check if "AFL" exists in the folder
name
        and not folder.endswith("R")  # Exclude folders ending with
"R"
    ]
)

# Raise an error if no valid versions are found
if not available_versions:
```

```
    raise FileNotFoundError("No valid versions found in 'runs/detect'
with 'AFL' in their name.")

# Display options and prompt the user for input
print("Available versions:")
for i, version in enumerate(available_versions, start=1):
    print(f"{i}. {version}")

try:
    selected_index = int(input("Enter the number of the version you
want to test: ")) - 1
    if selected_index < 0 or selected_index >=
len(available_versions):
        raise ValueError("Invalid selection.")
except ValueError:
    raise ValueError("Invalid input. Please enter a valid number.")

# Set the selected version and path to the weights
selected_version = available_versions[selected_index]
weights_path = os.path.join(detect_path, selected_version, "weights",
"best.pt")

# Output the selected version and the corresponding weights path
print(f"\nSelected version: {selected_version}")

# Verify the weights file exists
if not os.path.exists(weights_path):
    raise FileNotFoundError(f"Weights file not found: {weights_path}")
```

```
Available versions:
1. MDD-AFL-Yolov8_v1
2. MDD-AFL-Yolov8_v2
3. MDD-AFL-Yolov8_v3
4. MDD-AFL-Yolov8_v4
5. MDD-AFL-Yolov8_v5
6. MDD-AFL-Yolov8_v6

Selected version: MDD-AFL-Yolov8_v6
```

Code-based Metrics (Table Format)

This script analyzes the training and validation metrics for a selected YOLOv8 version. It reads the `results.csv` file from the specified version's folder, calculates averages for key metrics (e.g., precision, recall, mAP@50, mAP@50-95, and loss values), and displays both the final epoch values and their averages in a structured table. The script ensures robust error handling to manage missing files or invalid paths.

```
# Path to the selected version's folder
selected_version_path = os.path.join(detect_path, selected_version)
```

```python
# Path to results CSV file
results_csv_path = os.path.join(selected_version_path, "results.csv")

# Ensure the file exists
if not os.path.exists(results_csv_path):
    raise FileNotFoundError(f"'results.csv' not found in the selected
version: {selected_version_path}")

# Load the CSV file
results_df = pd.read_csv(results_csv_path)

# Compute averages for all metrics and losses
average_metrics = {
    "Metric": [
        "Precision", "Recall", "mAP@50", "mAP@50-95",
        "Train Box Loss", "Train Classification Loss",
        "Validation Box Loss", "Validation Classification Loss"
    ],
    "Final Value": [
        results_df['metrics/precision(B)'].iloc[-1],
        results_df['metrics/recall(B)'].iloc[-1],
        results_df['metrics/mAP50(B)'].iloc[-1],
        results_df['metrics/mAP50-95(B)'].iloc[-1],
        results_df['train/box_loss'].iloc[-1],
        results_df['train/cls_loss'].iloc[-1],
        results_df['val/box_loss'].iloc[-1],
        results_df['val/cls_loss'].iloc[-1]
    ],
    "Average Value": [
        results_df['metrics/precision(B)'].mean(),
        results_df['metrics/recall(B)'].mean(),
        results_df['metrics/mAP50(B)'].mean(),
        results_df['metrics/mAP50-95(B)'].mean(),
        results_df['train/box_loss'].mean(),
        results_df['train/cls_loss'].mean(),
        results_df['val/box_loss'].mean(),
        results_df['val/cls_loss'].mean()
    ]
}

# Convert metrics dictionary to DataFrame
metrics_df = pd.DataFrame(average_metrics)

# Display the title and the table
print(f" YOLO Model Training & Validation Metrics for Version:
{selected_version}")
display(metrics_df)
```

```
 YOLO Model Training & Validation Metrics for Version: MDD-AFL-
Yolov8_v6
```

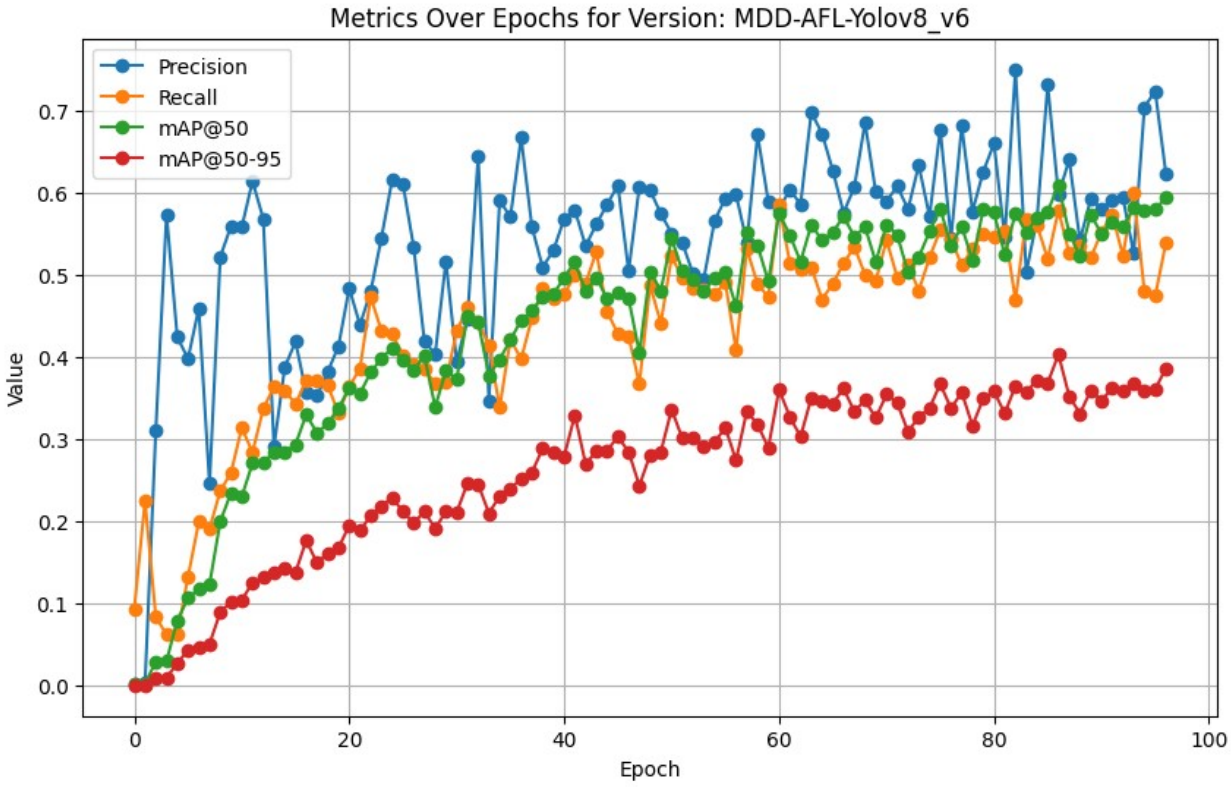
```
                          Metric  Final Value  Average Value
0                      Precision      0.62361       0.539661
1                         Recall      0.53942       0.435585
2                          mAP@50      0.59498       0.437615
3                       mAP@50-95      0.38640       0.259626
4                  Train Box Loss      1.17962       1.632396
5       Train Classification Loss      1.08770       1.813742
6            Validation Box Loss      1.36062       1.615706
7  Validation Classification Loss      1.63531       2.062133
```

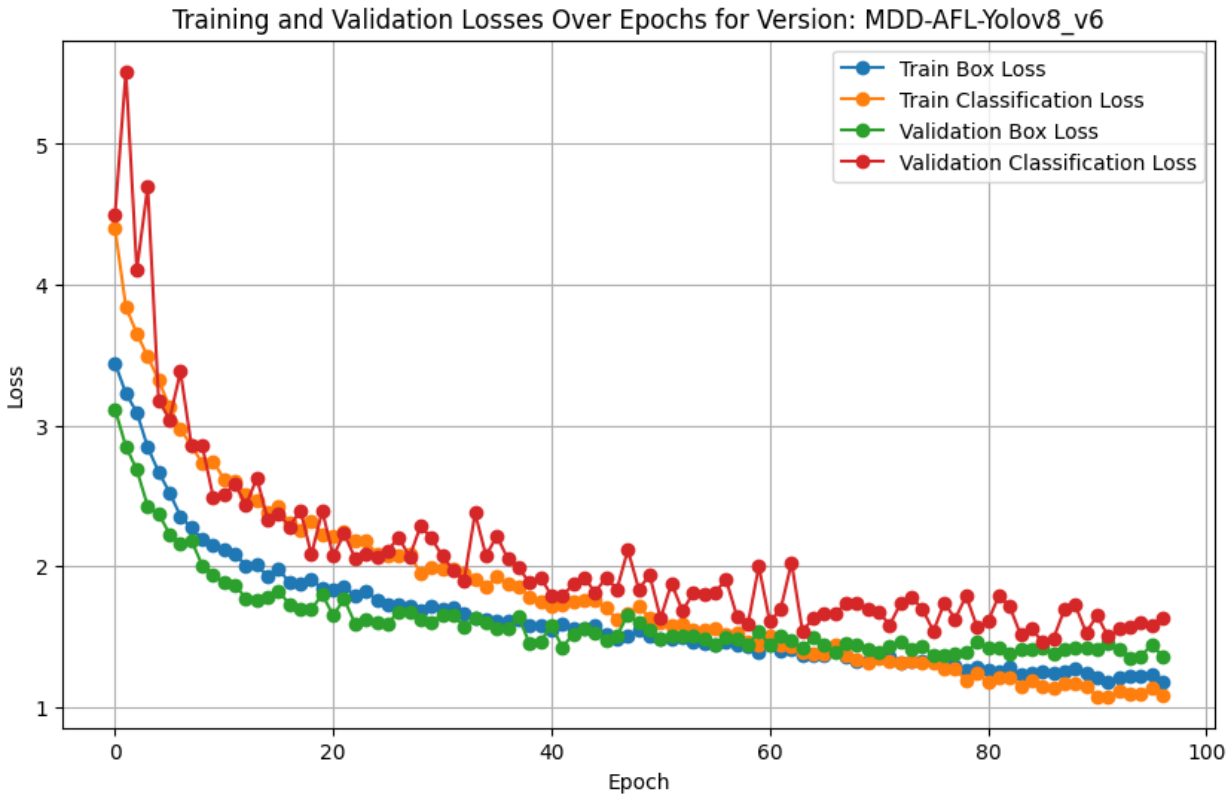
YOLOv8 Metrics and Loss Visualization for Selected Version

This script reads the `results.csv` file from the selected YOLOv8 version's folder and visualizes key metrics and losses over epochs. It plots Precision, Recall, mAP@50, and mAP@50-95 to assess the model's performance throughout training. Additionally, it visualizes the training and validation losses for box and classification tasks, helping to track the model's convergence and potential overfitting. The plots are displayed with clear labels and legends for easy interpretation.

```
# Path to the selected version's folder
selected_version_path = os.path.join(detect_path, selected_version)

# Path to results CSV file
results_csv_path = os.path.join(selected_version_path, "results.csv")

# Ensure the file exists
if not os.path.exists(results_csv_path):
    raise FileNotFoundError(f"'results.csv' not found in the selected
version: {selected_version_path}")

# Load the CSV file
results_df = pd.read_csv(results_csv_path)

# Plot Precision, Recall, mAP@50, and mAP@50-95 over epochs
plt.figure(figsize=(10, 6))
plt.plot(results_df.index, results_df['metrics/precision(B)'],
label='Precision', marker='o')
plt.plot(results_df.index, results_df['metrics/recall(B)'],
label='Recall', marker='o')
plt.plot(results_df.index, results_df['metrics/mAP50(B)'],
label='mAP@50', marker='o')
plt.plot(results_df.index, results_df['metrics/mAP50-95(B)'],
label='mAP@50-95', marker='o')
plt.title(f"Metrics Over Epochs for Version: {selected_version}")
plt.xlabel("Epoch")
plt.ylabel("Value")
plt.legend()
plt.grid(True)
plt.show()
```

```
# Plot Training and Validation Losses over epochs
plt.figure(figsize=(10, 6))
plt.plot(results_df.index, results_df['train/box_loss'], label='Train
Box Loss', marker='o')
plt.plot(results_df.index, results_df['train/cls_loss'], label='Train
Classification Loss', marker='o')
plt.plot(results_df.index, results_df['val/box_loss'],
label='Validation Box Loss', marker='o')
plt.plot(results_df.index, results_df['val/cls_loss'],
label='Validation Classification Loss', marker='o')
plt.title(f"Training and Validation Losses Over Epochs for Version:
{selected_version}")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)
plt.show()
```



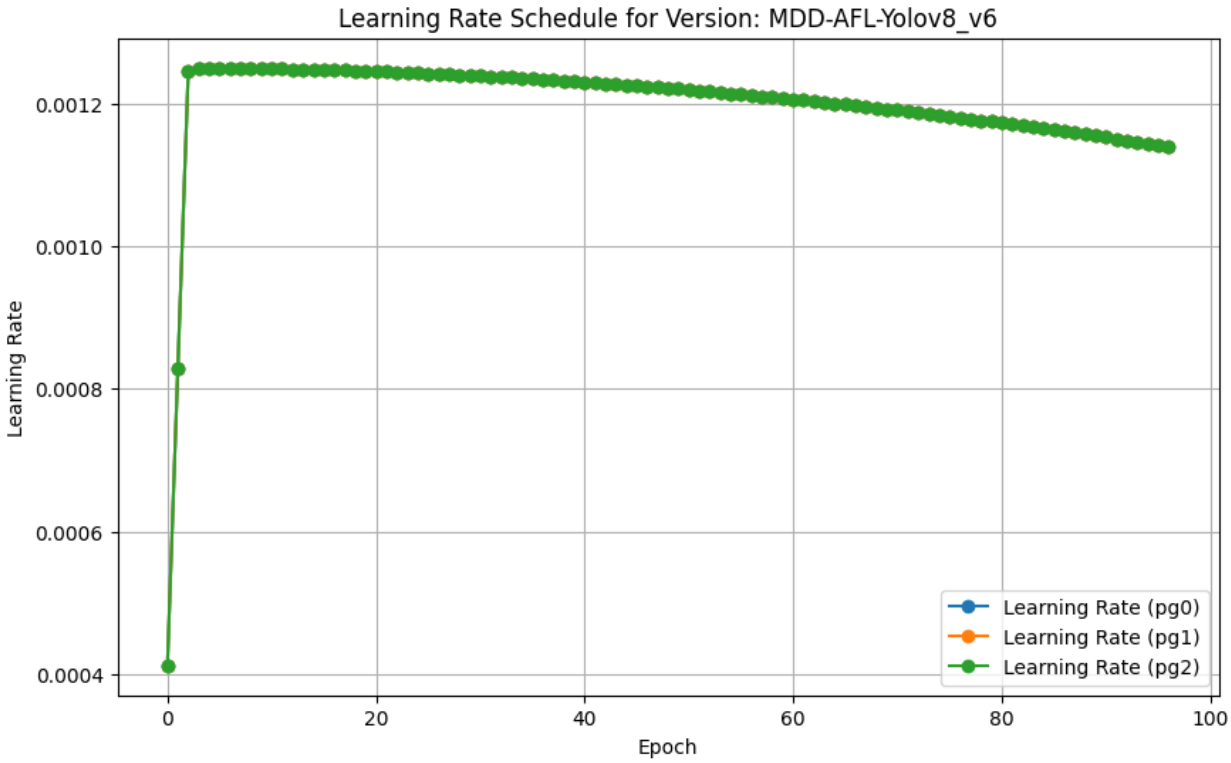


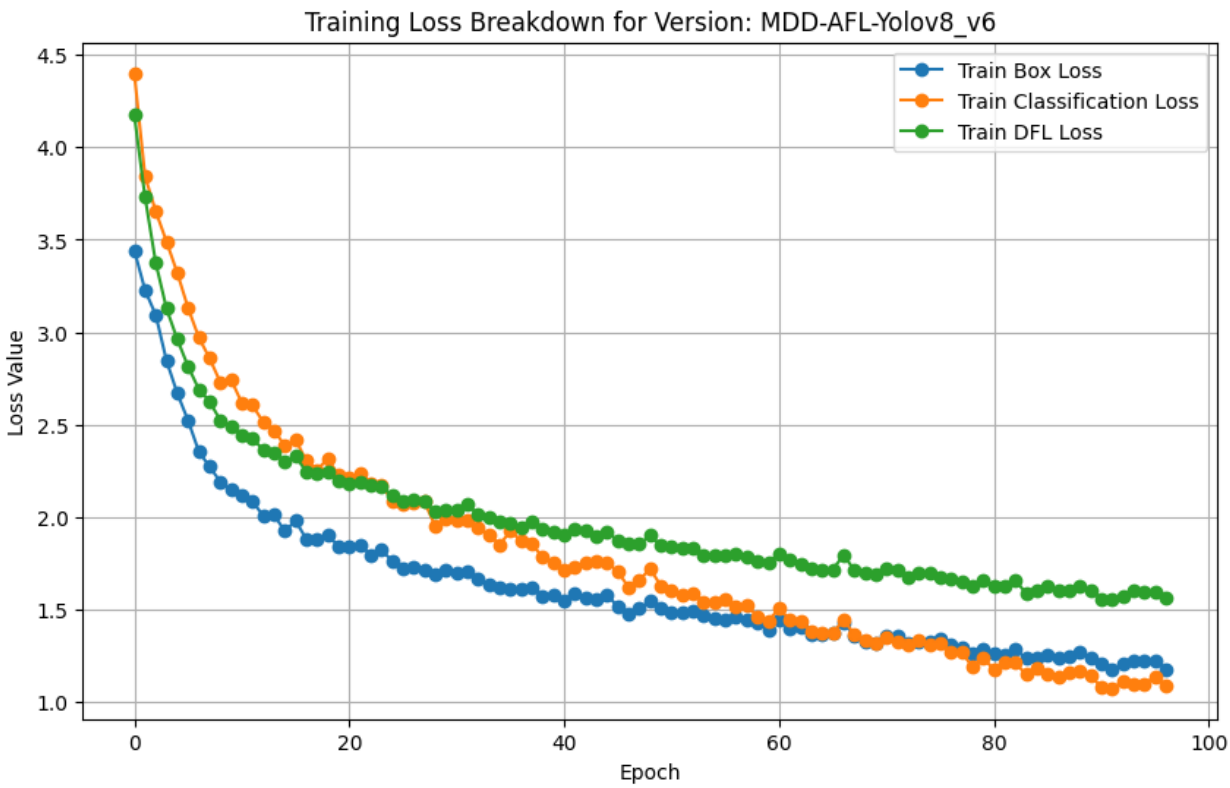
YOLOv8 Learning Rate and Loss Visualization for Selected Version

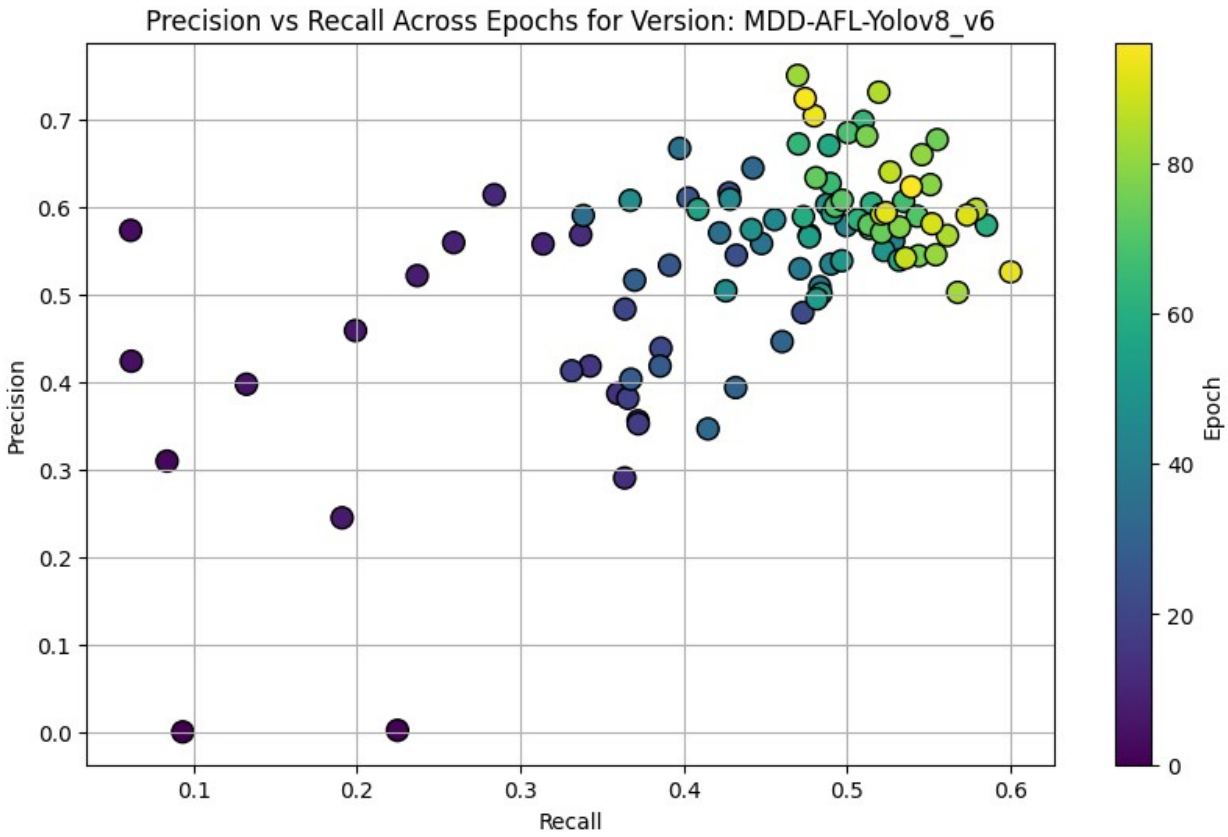
This script reads the `results.csv` file from the selected YOLOv8 version's folder and generates three visualizations. First, it plots the learning rate schedule for different parameter groups (`pg0`, `pg1`, `pg2`) over epochs. Second, it visualizes the breakdown of training losses, including box loss, classification loss, and DFL loss. Finally, a scatter plot shows the relationship between precision and recall across epochs, with color coding for epoch progression. These visualizations provide insight into the model's learning rate adjustments, loss behavior, and performance trade-offs.

```
# Path to the selected version's folder
selected_version_path = os.path.join(detect_path, selected_version)

# Path to results CSV file
results_csv_path = os.path.join(selected_version_path, "results.csv")

# Ensure the file exists
if not os.path.exists(results_csv_path):
    raise FileNotFoundError(f"'results.csv' not found in the selected
version: {selected_version_path}")

# Load the CSV file
results_df = pd.read_csv(results_csv_path)

# Plot Learning Rate Schedule
```

```
plt.figure(figsize=(10, 6))
plt.plot(results_df.index, results_df['lr/pg0'], label='Learning Rate
(pg0)', marker='o')
plt.plot(results_df.index, results_df['lr/pg1'], label='Learning Rate
(pg1)', marker='o')
plt.plot(results_df.index, results_df['lr/pg2'], label='Learning Rate
(pg2)', marker='o')
plt.title(f"Learning Rate Schedule for Version: {selected_version}")
plt.xlabel("Epoch")
plt.ylabel("Learning Rate")
plt.legend()
plt.grid(True)
plt.show()

# Plot Loss Breakdown
plt.figure(figsize=(10, 6))
plt.plot(results_df.index, results_df['train/box_loss'], label='Train
Box Loss', marker='o')
plt.plot(results_df.index, results_df['train/cls_loss'], label='Train
Classification Loss', marker='o')
plt.plot(results_df.index, results_df['train/dfl_loss'], label='Train
DFL Loss', marker='o')
plt.title(f"Training Loss Breakdown for Version: {selected_version}")
plt.xlabel("Epoch")
plt.ylabel("Loss Value")
plt.legend()
plt.grid(True)
plt.show()

# Plot Precision vs. Recall
plt.figure(figsize=(10, 6))
plt.scatter(results_df['metrics/recall(B)'],
results_df['metrics/precision(B)'], c=results_df.index,
cmap='viridis', s=100, edgecolor='k')
plt.colorbar(label="Epoch")
plt.title(f"Precision vs Recall Across Epochs for Version:
{selected_version}")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.grid(True)
plt.show()
```


Learning Rate Schedule for Version: MDD-AFL-Yolov8_v6
Learning Rate
Epoch
0.0004
0.0006
0.0008
0.0010
0.0012
0
20
40
60
80
100
Learning Rate (pg0)
Learning Rate (pg1)
Learning Rate (pg2)


Training Loss Breakdown for Version: MDD-AFL-Yolov8_v6
Loss Value
Epoch
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
0
20
40
60
80
100
Train Box Loss
Train Classification Loss
Train DFL Loss



Visualisation

Image Processing Helper Functions

This script includes several functions for handling YOLOv8 model loading, image processing, and visualization. The `load_model` function loads a YOLOv8 model from a specified path, while the `load_image` function loads and converts an image to RGB format. The `ensure_rgb` function ensures that images are in the correct RGB format, handling grayscale and RGB/BGR images appropriately.

```
# Load the model
def load_model(weights_path):
    if not os.path.exists(weights_path):
        raise FileNotFoundError(f"Weights file not found at
{weights_path}.")
    return YOLO(weights_path)

# Load the image
def load_image(image_path):
    if not os.path.exists(image_path):
        raise FileNotFoundError(f"Image file not found at
{image_path}.")
    image = cv2.imread(image_path)  # Load image in BGR format
```

```
    return cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # Convert to RGB

# Ensure images are in RGB format
def ensure_rgb(image):
    if len(image.shape) == 3 and image.shape[2] == 3:  # Check if the
image has 3 channels
        return cv2.cvtColor(image, cv2.COLOR_BGR2RGB) if not
np.array_equal(
            image, cv2.cvtColor(cv2.cvtColor(image,
cv2.COLOR_RGB2BGR), cv2.COLOR_BGR2RGB)
        ) else image
    elif len(image.shape) == 2:  # Grayscale image
        return cv2.cvtColor(image, cv2.COLOR_GRAY2RGB)  # Convert
grayscale to RGB
    else:
        raise ValueError("Unsupported image format. Image must have 1
(grayscale) or 3 (RGB/BGR) channels.")
```

Display Original and Predicted Images Side by Side

This function displays the **original image** and the **predicted image with bounding boxes** side by side. It filters out bounding boxes with confidence below a specified threshold, and the images are shown for easy comparison of the model's predictions against the original input.

```
def display_images_side_by_side(original, predicted, results,
confidence_threshold=0.35, titles=("Original Test Image", "Predicted
Image with Bounding Boxes")):
    # Ensure both images are in RGB format
    original_rgb = ensure_rgb(original)

    # Filter results to exclude bounding boxes with confidence below
threshold
    filtered_boxes = []
    boxes = results[0].boxes  # Access the bounding boxes from the
results
    if boxes is not None:
        for box in boxes:
            x1, y1, x2, y2 = box.xyxy[0].tolist()  # Extract
coordinates
            confidence = box.conf.item()           # Extract
confidence as a scalar
            class_id = int(box.cls.item())         # Extract class ID
as a scalar
            if confidence >= confidence_threshold:
                filtered_boxes.append((x1, y1, x2, y2, confidence,
class_id))

    predicted_rgb = ensure_rgb(predicted)
```

```
    # Display images side by side
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.imshow(original_rgb)
    plt.title(titles[0])
    plt.axis("off")
    plt.subplot(1, 2, 2)
    plt.imshow(predicted_rgb)
    plt.title(titles[1])
    plt.axis("off")
    plt.tight_layout()
    plt.show()
```

Crop and Display Bounding Boxes in a Grid

This function crops the image based on detected bounding boxes with confidence above a specified threshold and displays them in a grid. Each cropped image is labeled with its class name and confidence score, allowing for easy inspection of individual detected objects.

```
def crop_and_display_boxes_grouped(original_image, results,
confidence_threshold=0.36):
    # Get bounding boxes and their corresponding class names
    boxes = results[0].boxes
    confidences = boxes.conf.cpu().numpy()
    class_ids = boxes.cls.cpu().numpy()

    # Initialize a list to store cropped images
    cropped_images = []
    cropped_titles = []

    # Loop through each bounding box and crop the region
    for i, (box, confidence, class_id) in enumerate(zip(boxes,
confidences, class_ids)):
        if confidence >= confidence_threshold:
            # Get the coordinates of the bounding box
            x1, y1, x2, y2 = box.xyxy[0].tolist()

            # Crop the image around the bounding box
            cropped_image = original_image[int(y1):int(y2),
int(x1):int(x2)]

            # Map class ID to class name
            class_name = model.names[int(class_id)]

            # Store the cropped image and its title
            cropped_images.append(cropped_image)
            cropped_titles.append(f"{class_name} - Conf:
{confidence:.2f}")
```

```
    # If there are cropped images, display them in a grid
    if cropped_images:
        num_images = len(cropped_images)
        grid_size = int(np.ceil(np.sqrt(num_images)))  # Calculate
grid size for display

        # Plot cropped images in a grid
        plt.figure(figsize=(grid_size * 3, grid_size * 3))
        for i, (cropped_image, title) in enumerate(zip(cropped_images,
cropped_titles)):
            plt.subplot(grid_size, grid_size, i + 1)
            plt.imshow(cropped_image)
            plt.title(title)
            plt.axis("off")
        plt.tight_layout()
        plt.show()
```

Visual Analytics

This script defines a function `analyze_detections` that processes the results from a YOLOv8 model, analyzing the detected objects in an image. It extracts bounding boxes, class IDs, and confidence scores, then maps the class IDs to class names using the model's predefined classes. The function performs a breakdown of detected objects, displaying the total number and counts for specific classes, such as "Mixed Waste," "Organic Waste," "Other Waste," and "Recyclable Material."

The function also visualizes the analysis through two types of plots:

1. **Bar Chart**: Displays the count of detected objects per class, with color-coded bars for easy identification.
2. **Scatter Plot**: Plots confidence scores for each detection, color-coded by class, allowing for quick assessment of model certainty. These visualizations help in evaluating detection accuracy and confidence.

```
def analyze_detections(results, model):
    # Get detected boxes and classes
    boxes = results[0].boxes  # Bounding box tensor
    num_objects = len(boxes)  # Total number of detected objects

    if num_objects == 0:
        print("No (0) objects detected in the image.")
        return

    # Extract class IDs and confidence scores
    class_ids = boxes.cls.cpu().numpy()
    confidences = boxes.conf.cpu().numpy()

    # Map class IDs to class names using model's class names
    class_names = [model.names[int(cls_id)] for cls_id in class_ids]
```

```
    # Define fixed classes (ensure consistency in bar chart even if
some classes are missing)
    fixed_classes = ["Mixed Waste", "Organic Waste", "Other Waste",
"Recyclable Material"]
    class_counts = {class_name: 0 for class_name in fixed_classes}
    for class_name in class_names:
        if class_name in class_counts:
            class_counts[class_name] += 1

    # Display analytics
    print("\n=== Image Analytics ===")
    print(f"Total Objects Detected: {num_objects}")
    print("Object Breakdown:")
    for class_name, count in class_counts.items():
        print(f"  - {class_name}: {count}")
    print("Confidence Scores:")
    for i, (class_name, confidence) in enumerate(zip(class_names,
confidences)):
        print(f"  {i + 1}. {class_name} - Confidence:
{confidence:.2f}")

    # Plot bar chart for class counts
    colors = {
        "Mixed Waste": "lightcoral",  # Pastel red
        "Organic Waste": "lightgreen",  # Pastel green
        "Other Waste": "sandybrown",  # Pastel orange
        "Recyclable Material": "lightskyblue"  # Pastel blue
    }

    plt.figure(figsize=(8, 5))
    plt.bar(class_counts.keys(), class_counts.values(),
color=[colors[class_name] for class_name in class_counts.keys()])
    plt.title('Object Count per Class')
    plt.xlabel('Classes')
    plt.ylabel('Count')
    plt.yticks(range(0, max(class_counts.values()) + 2, 1))  # Ensure
y-axis increments by 1
    plt.show()

    # Scatter plot for confidence scores
    plt.figure(figsize=(10, 6))
    scatter_colors = [colors[class_name] for class_name in
class_names]
    plt.scatter(class_names, confidences, c=scatter_colors, alpha=0.7,
edgecolor='k')
    plt.title('Confidence Scores by Class')
    plt.xlabel('Classes')
    plt.ylabel('Confidence Scores')
    plt.ylim(0, 1.05)  # Confidence scores range from 0 to 1
```

```
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.show()
```

YOLOv8 Model Prediction and Evaluation on Test Images

This script loads a YOLOv8 model using the selected `best.pt` weights file and processes a set of test images from a designated folder. It checks if the test folder exists and contains valid image files, then iterates over each image to perform predictions. The script filters out bounding boxes with a confidence score below 0.36, renders the filtered results, and displays both the original and predicted images side by side.

Additionally, it analyzes the detection results, providing insights into the number of objects detected, the class breakdown, and confidence scores. Visualizations, such as side-by-side comparisons and cropped object grids, help evaluate the model's performance on test images.

```
# Path to the selected version's folder
weights_path = os.path.join(os.getcwd(), "runs", "detect",
selected_version, "weights", "best.pt")
test_folder_path = os.path.abspath(os.path.join(os.getcwd(), "..",
"Test"))

print(f"Selected weights file: {weights_path}")

# Validate and process all images in the test folder
if not os.path.exists(test_folder_path):
    raise FileNotFoundError(f"Test folder not found at:
{test_folder_path}")

test_images = [img for img in os.listdir(test_folder_path) if
img.lower().endswith(('.png', '.jpg', '.jpeg'))]
if not test_images:
    raise ValueError(f"No valid image files found in the test folder:
{test_folder_path}")

# Determine the device
device = determine_device()  # Ensure this function is defined
elsewhere in your code

# Load the YOLO model
model = load_model(weights_path)

# Ensure plots are displayed inline
%matplotlib inline

# Loop through each test image
for test_image in test_images:
    test_image_path = os.path.join(test_folder_path, test_image)
    print(f"Processing test image: {test_image_path}")

    # Perform prediction on the current test image
```

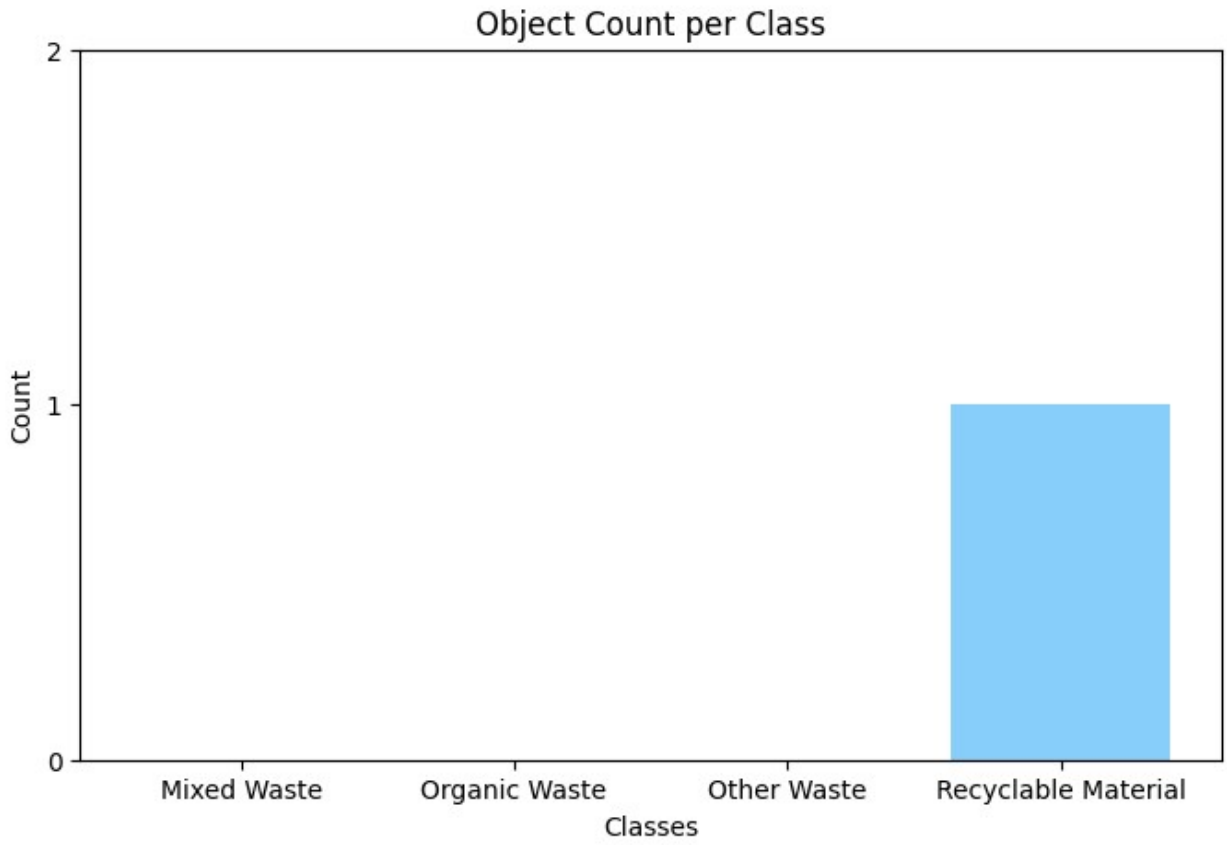
```
    results = model.predict(source=test_image_path, device=device)

    # Load and process the images
    original_image = load_image(test_image_path)  # Ensure this loads
the image in RGB

    # Filter out boxes with confidence less than 0.35
    filtered_boxes = results[0].boxes[results[0].boxes.conf > 0.36]

    # Create a copy of results and replace boxes with filtered ones
    results[0].boxes = filtered_boxes

    # Render the filtered results
    predicted_image = cv2.cvtColor(results[0].plot(),
cv2.COLOR_BGR2RGB)  # Convert rendered image to RGB

    # Display the original and predicted images side by side
    display_images_side_by_side(original_image, predicted_image,
results)

    # Analyze and display detection analytics
    analyze_detections(results, model)

    # Crop and display the detected bounding boxes in a grouped grid
    crop_and_display_boxes_grouped(original_image, results)
```

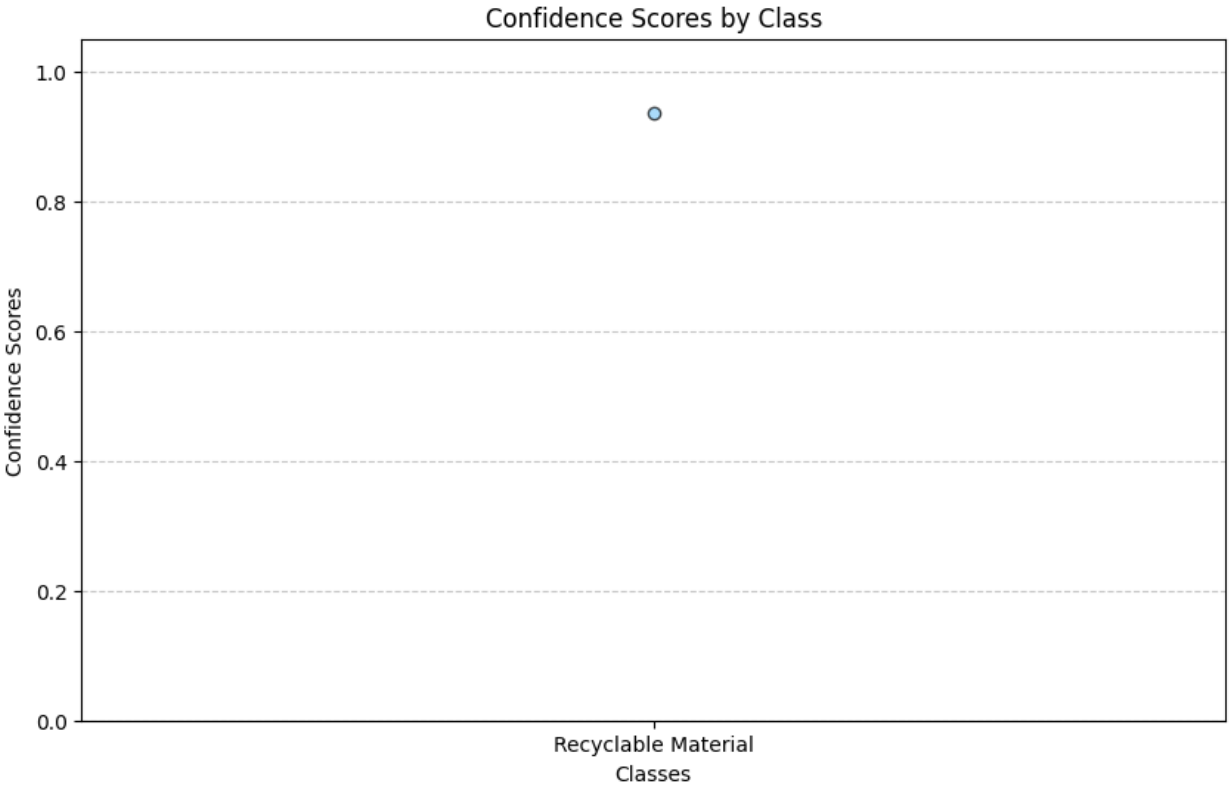
```
Selected weights file: /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced CV/Assignments/ARI3129-MDD/Jupyter
Notebooks/runs/detect/MDD-AFL-Yolov8_v6/weights/best.pt
Using MPS (Metal Performance Shaders) for acceleration.
Processing test image: /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced CV/Assignments/ARI3129-MDD/Test/AFL_T4.jpeg

WARNING ⚠ NMS time limit 2.050s exceeded
image 1/1 /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced CV/Assignments/ARI3129-MDD/Test/AFL_T4.jpeg:
640x480 1 Recyclable Material, 367.1ms
Speed: 98.2ms preprocess, 367.1ms inference, 6948.9ms postprocess per
image at shape (1, 3, 640, 480)
```

Original Test Image

Predicted Image with Bounding Boxes


```
=== Image Analytics ===
Total Objects Detected: 1
Object Breakdown:
  - Mixed Waste: 0
  - Organic Waste: 0
  - Other Waste: 0
  - Recyclable Material: 1
Confidence Scores:
  1. Recyclable Material - Confidence: 0.94
```


Object Count per Class
Count
2
1
0
Mixed Waste
Organic Waste
Other Waste
Recyclable Material
Classes


Confidence Scores by Class
Confidence Scores
1.0
0.8
0.6
0.4
0.2
0.0
Recyclable Material
Classes

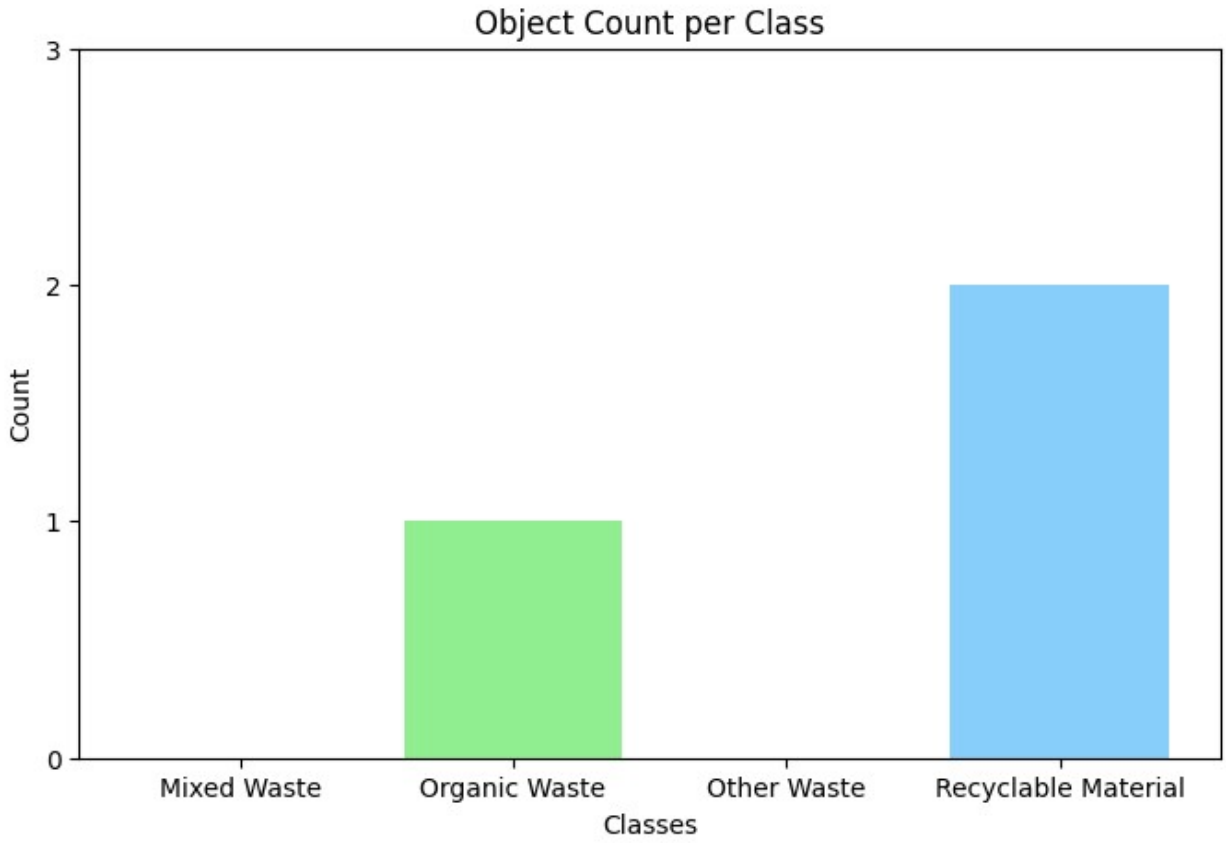
Recyclable Material - Conf: 0.94

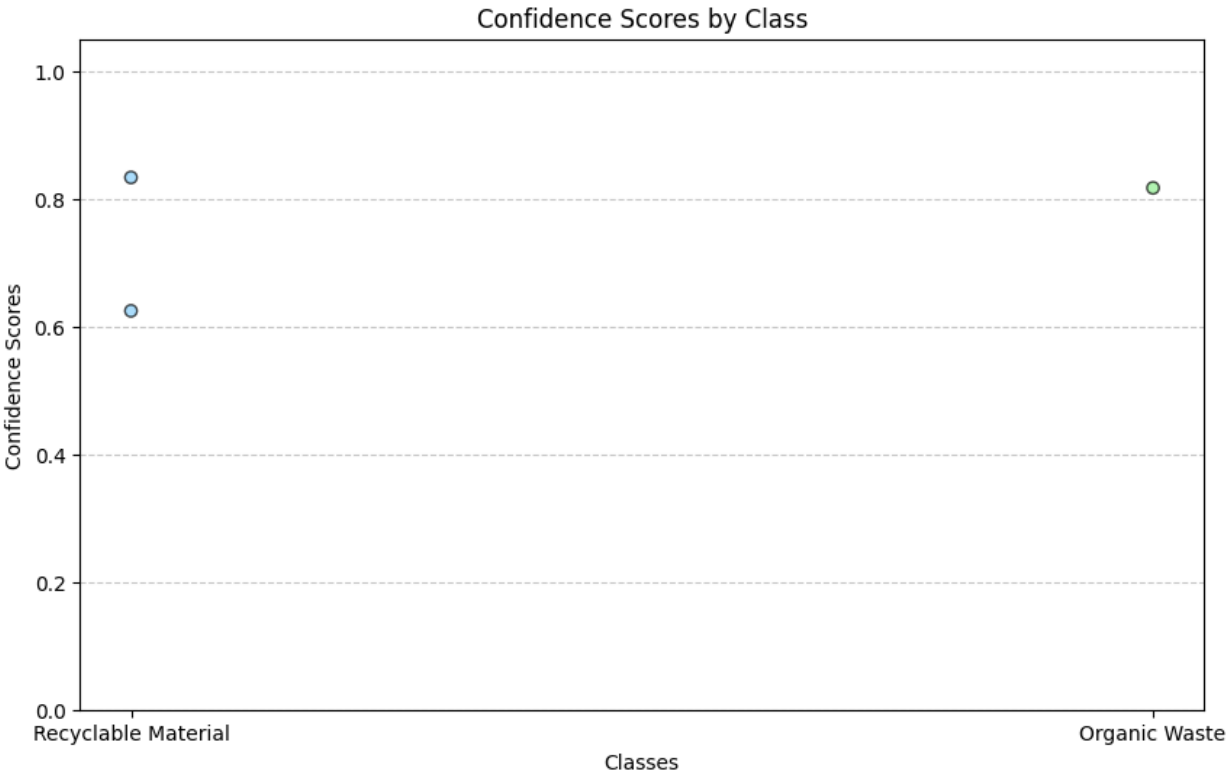
```
Processing test image: /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced CV/Assignments/ARI3129-MDD/Test/AFL_T5.jpeg

image 1/1 /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced CV/Assignments/ARI3129-MDD/Test/AFL_T5.jpeg:
640x480 1 Organic Waste, 2 Recyclable Materials, 106.0ms
Speed: 7.3ms preprocess, 106.0ms inference, 257.5ms postprocess per
image at shape (1, 3, 640, 480)
```

Original Test Image

Predicted Image with Bounding Boxes

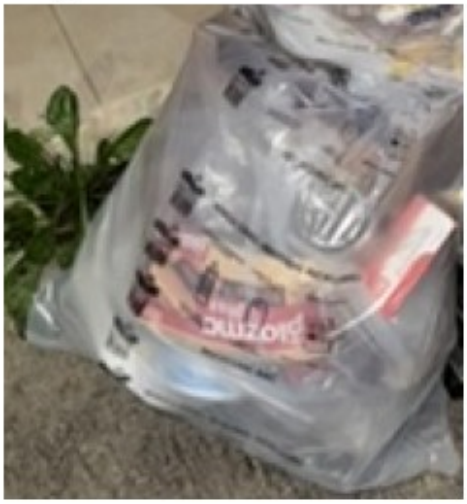
```
=== Image Analytics ===
Total Objects Detected: 3
Object Breakdown:
  - Mixed Waste: 0
  - Organic Waste: 1
  - Other Waste: 0
  - Recyclable Material: 2
Confidence Scores:
  1. Recyclable Material - Confidence: 0.83
  2. Organic Waste - Confidence: 0.82
  3. Recyclable Material - Confidence: 0.62
```



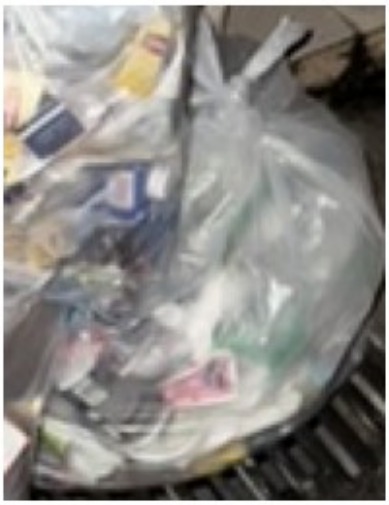

Confidence Scores by Class
1.0
0.8
0.6
0.4
0.2
0.0
Confidence Scores
Recyclable Material
Organic Waste
Classes

Recyclable Material - Conf: 0.83

Organic Waste - Conf: 0.82



Recyclable Material - Conf: 0.62

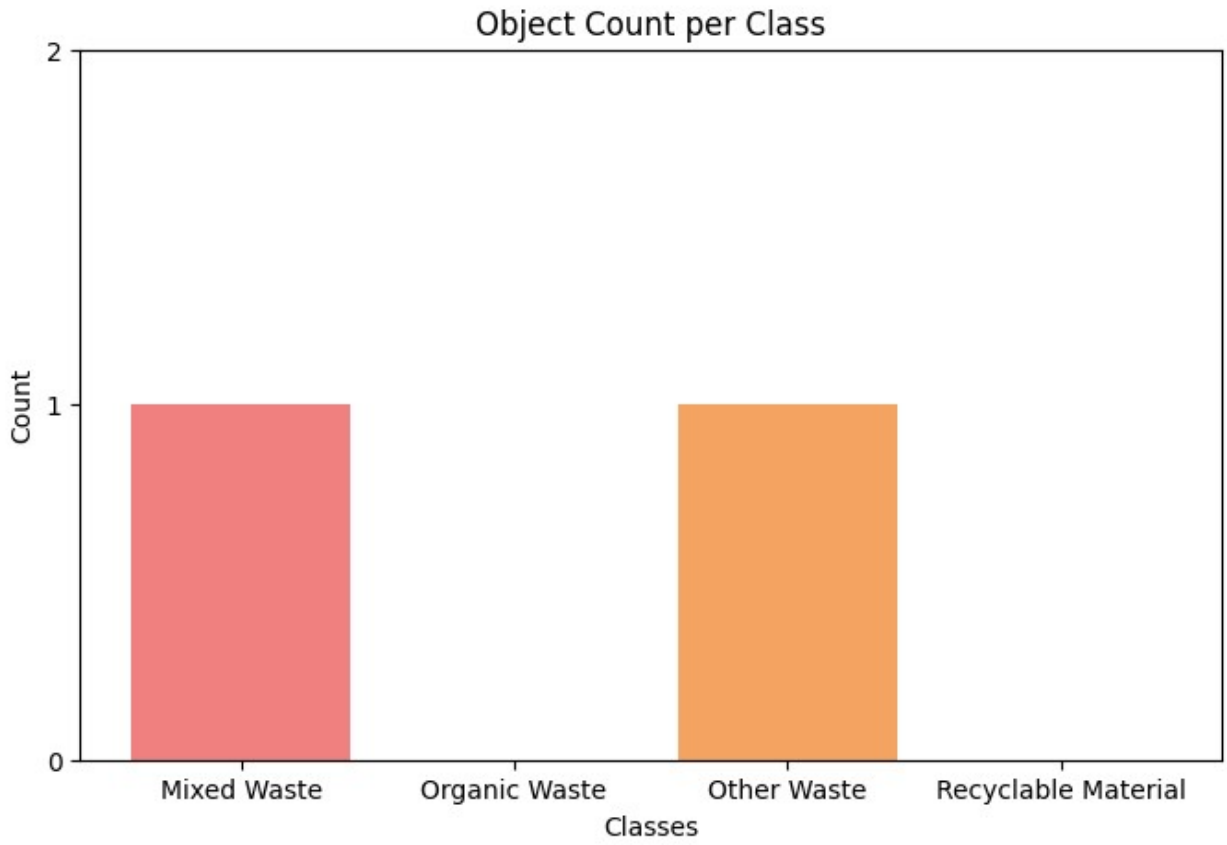


```
Processing test image: /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced CV/Assignments/ARI3129-MDD/Test/AFL_T2.jpeg

image 1/1 /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced CV/Assignments/ARI3129-MDD/Test/AFL_T2.jpeg:
640x480 1 Mixed Waste, 2 Other Wastes, 64.3ms
Speed: 7.8ms preprocess, 64.3ms inference, 156.5ms postprocess per
image at shape (1, 3, 640, 480)
```

Original Test Image

Predicted Image with Bounding Boxes


```
=== Image Analytics ===
Total Objects Detected: 2
Object Breakdown:
  - Mixed Waste: 1
  - Organic Waste: 0
  - Other Waste: 1
  - Recyclable Material: 0
Confidence Scores:
  1. Mixed Waste - Confidence: 0.96
  2. Other Waste - Confidence: 0.47
```


Object Count per Class
Count
2
1
0
Mixed Waste
Organic Waste
Other Waste
Recyclable Material
Classes


Confidence Scores by Class
Confidence Scores
1.0
0.8
0.6
0.4
0.2
0.0
Mixed Waste
Other Waste
Classes

Mixed Waste - Conf: 0.96

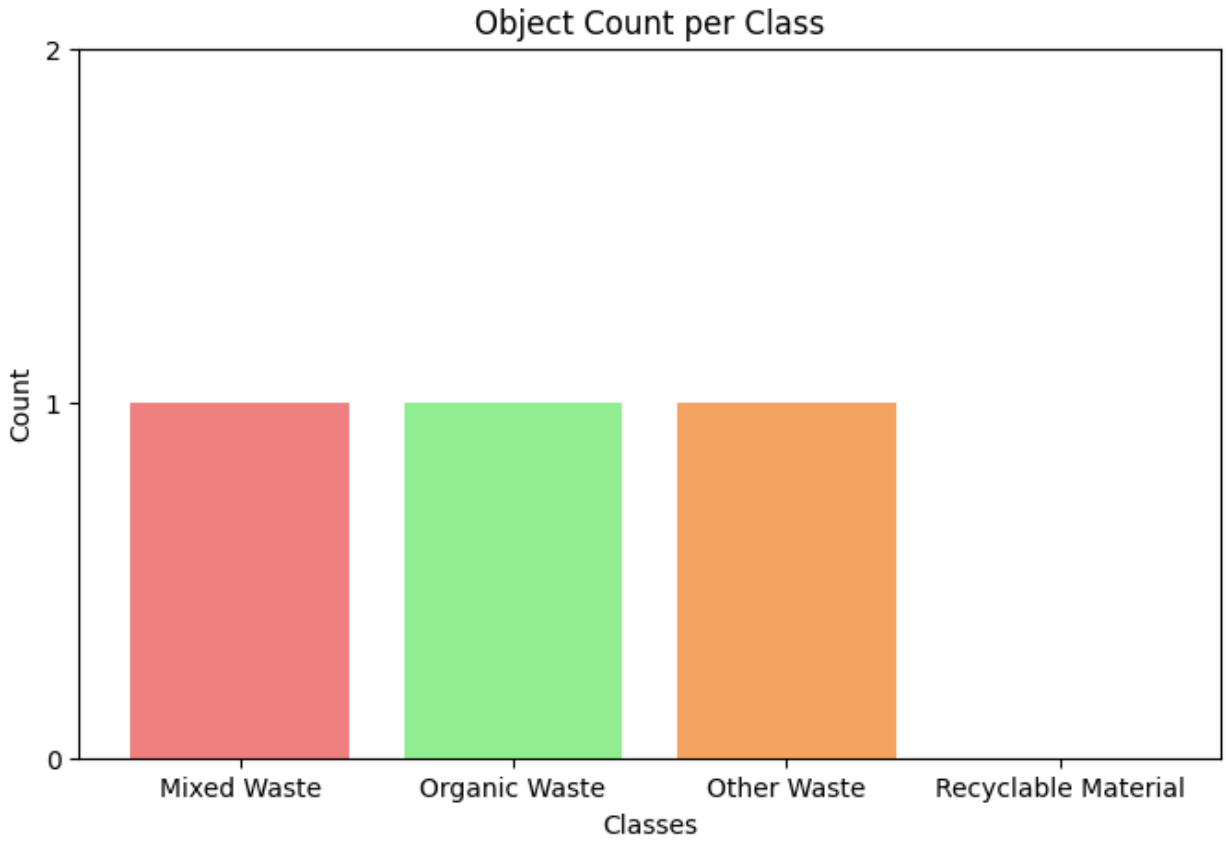
Other Waste - Conf: 0.47

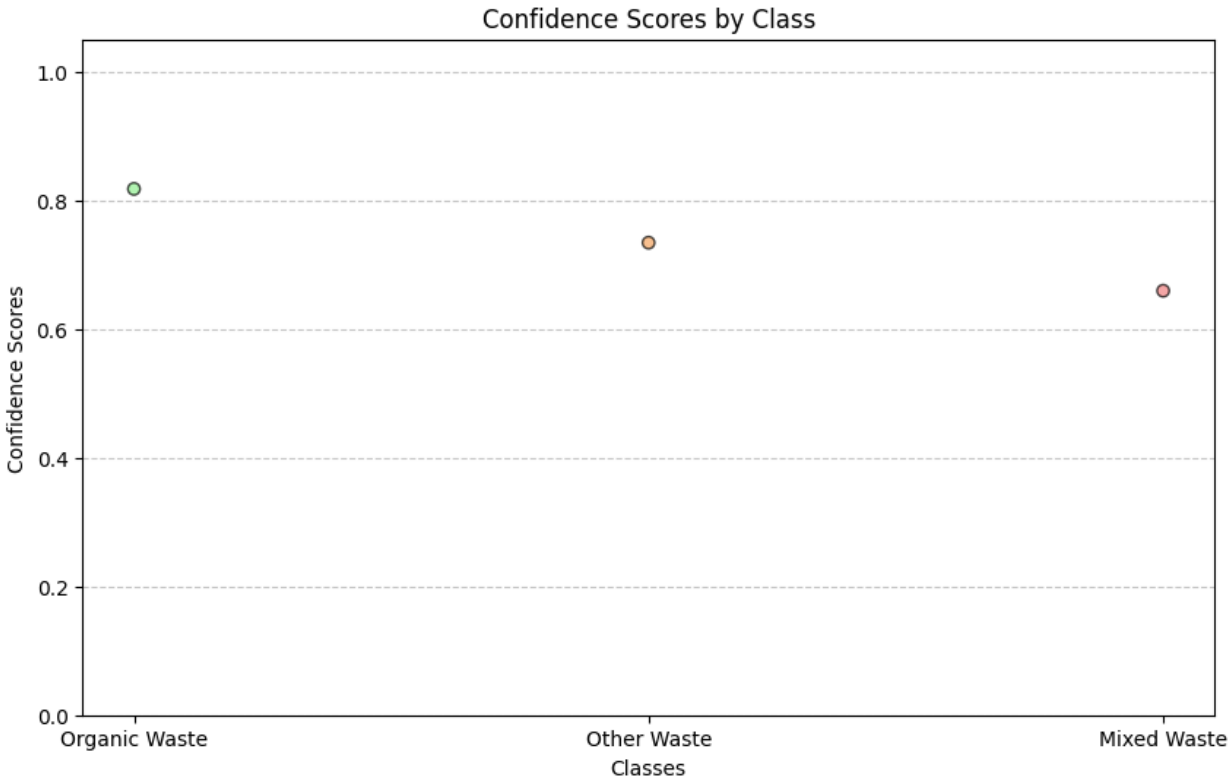
```
Processing test image: /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced CV/Assignments/ARI3129-MDD/Test/AFL_T3.jpeg

image 1/1 /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced CV/Assignments/ARI3129-MDD/Test/AFL_T3.jpeg:
640x480 1 Mixed Waste, 1 Organic Waste, 1 Other Waste, 77.5ms
Speed: 6.5ms preprocess, 77.5ms inference, 154.7ms postprocess per
image at shape (1, 3, 640, 480)
```

Original Test Image

Predicted Image with Bounding Boxes

```
=== Image Analytics ===
Total Objects Detected: 3
Object Breakdown:
  - Mixed Waste: 1
  - Organic Waste: 1
  - Other Waste: 1
  - Recyclable Material: 0
Confidence Scores:
  1. Organic Waste - Confidence: 0.82
  2. Other Waste - Confidence: 0.73
  3. Mixed Waste - Confidence: 0.66
```




Confidence Scores by Class
1.0
0.8
0.6
0.4
0.2
0.0
Confidence Scores
Organic Waste
Other Waste
Mixed Waste
Classes

Organic Waste - Conf: 0.82

Other Waste - Conf: 0.73

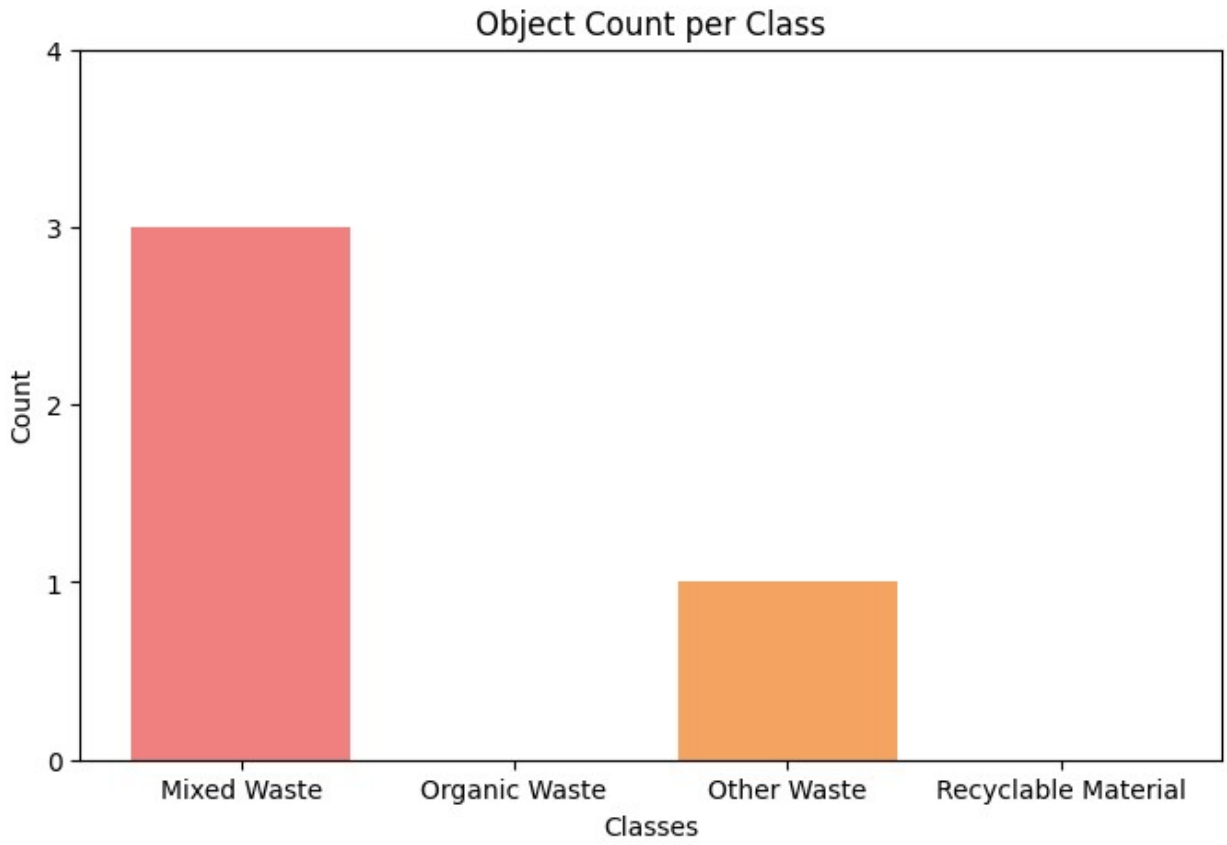
Mixed Waste - Conf: 0.66

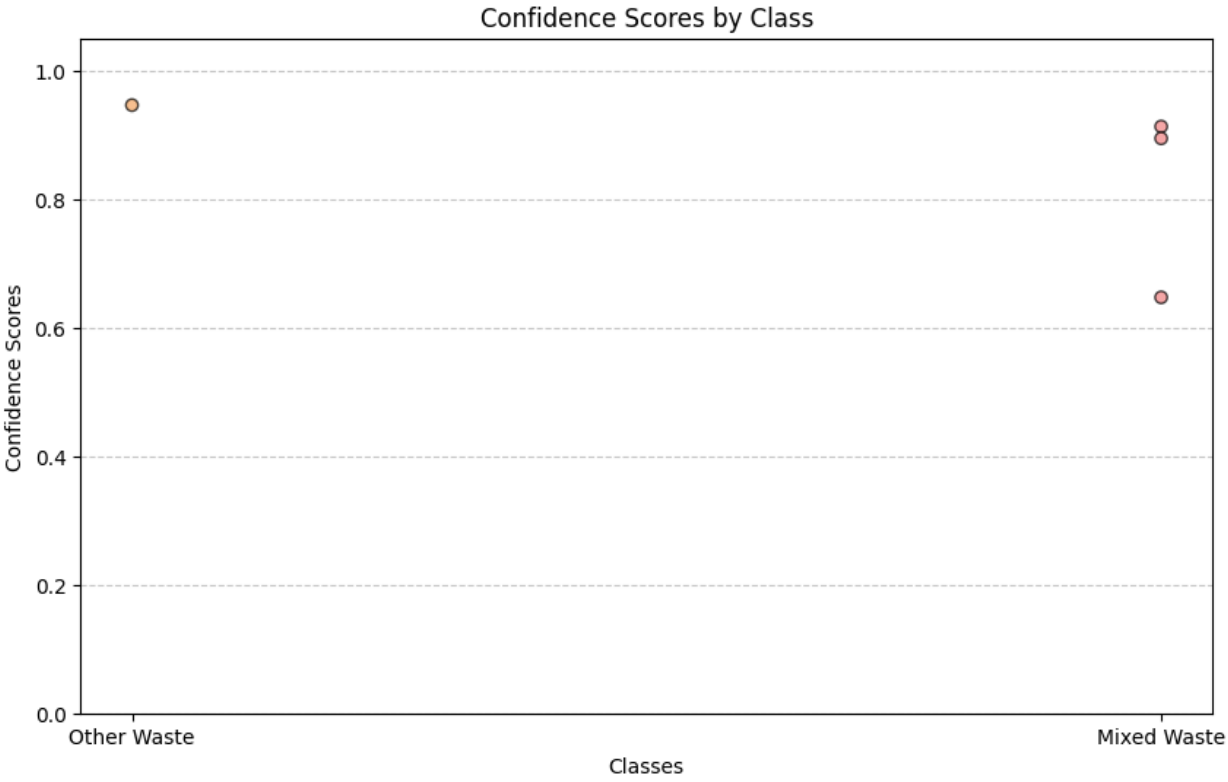
```
Processing test image: /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced CV/Assignments/ARI3129-MDD/Test/AFL_T3N.jpeg

WARNING ⚠ NMS time limit 2.050s exceeded
image 1/1 /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced CV/Assignments/ARI3129-MDD/Test/AFL_T3N.jpeg:
640x480 4 Mixed Wastes, 1 Organic Waste, 2 Other Wastes, 30.0ms
Speed: 5.0ms preprocess, 30.0ms inference, 2720.6ms postprocess per
image at shape (1, 3, 640, 480)
```

Original Test Image

Predicted Image with Bounding Boxes


```
=== Image Analytics ===
Total Objects Detected: 4
Object Breakdown:
  - Mixed Waste: 3
  - Organic Waste: 0
  - Other Waste: 1
  - Recyclable Material: 0
Confidence Scores:
  1. Other Waste - Confidence: 0.95
  2. Mixed Waste - Confidence: 0.91
  3. Mixed Waste - Confidence: 0.89
  4. Mixed Waste - Confidence: 0.65
```


Object Count per Class
Count
Classes
Mixed Waste
Organic Waste
Other Waste
Recyclable Material
0
1
2
3
4

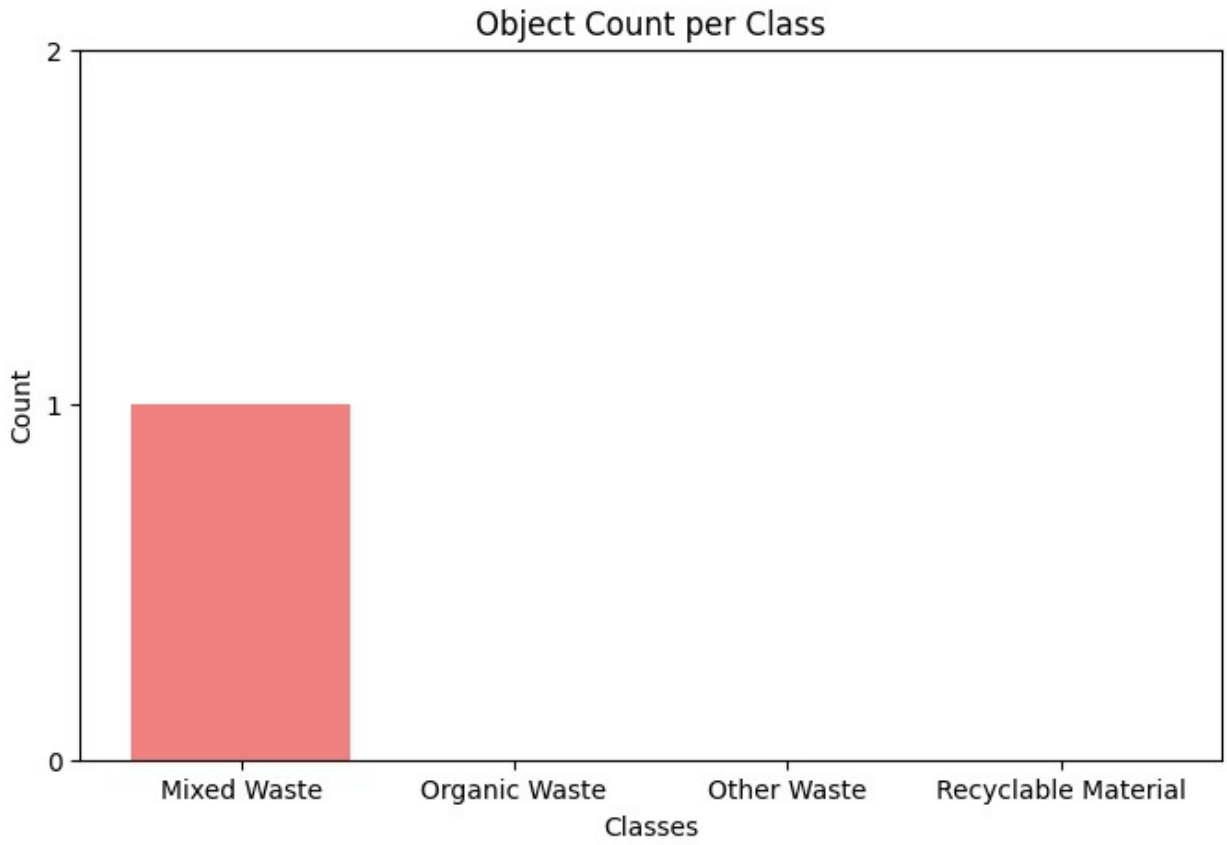

Confidence Scores by Class
Confidence Scores
Classes
Other Waste
Mixed Waste
0.0
0.2
0.4
0.6
0.8
1.0

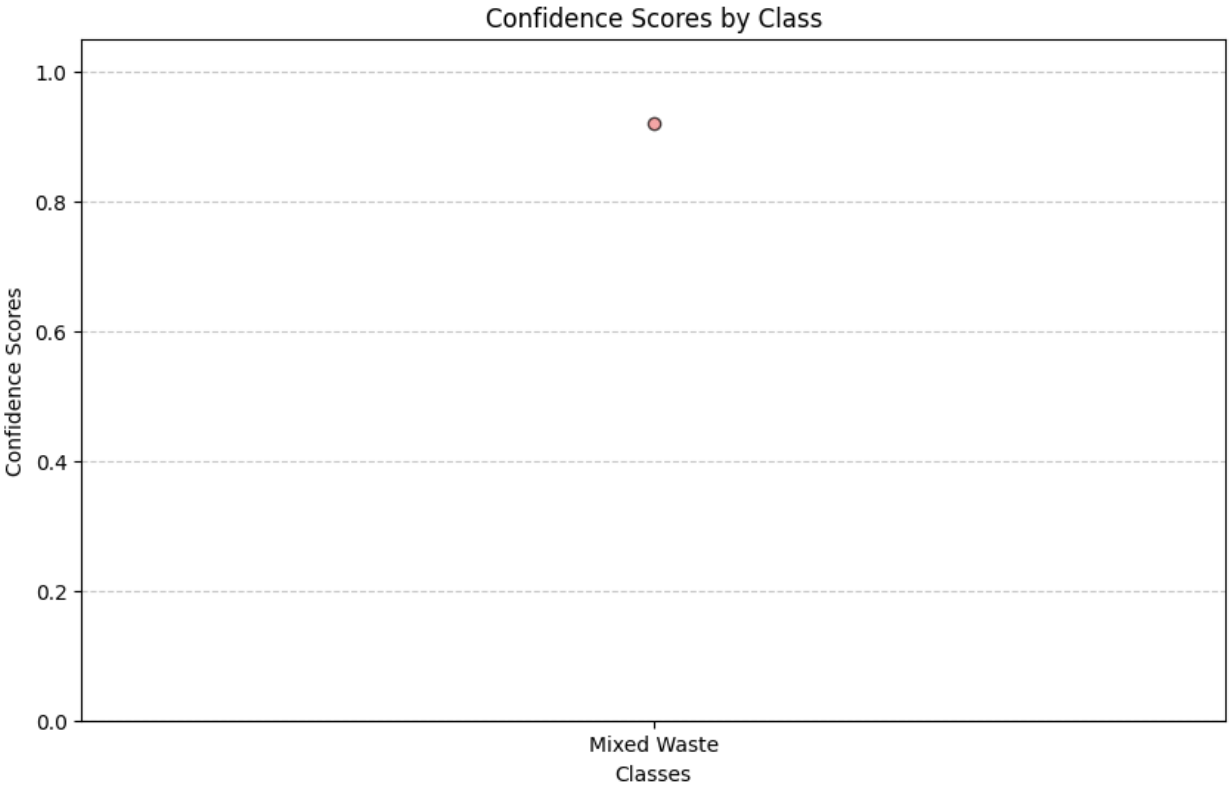

```
Processing test image: /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced CV/Assignments/ARI3129-MDD/Test/AFL_T1N.jpeg

image 1/1 /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced CV/Assignments/ARI3129-MDD/Test/AFL_T1N.jpeg:
640x480 1 Mixed Waste, 155.7ms
Speed: 8.0ms preprocess, 155.7ms inference, 240.1ms postprocess per
image at shape (1, 3, 640, 480)
```

Original Test Image

Predicted Image with Bounding Boxes


```
=== Image Analytics ===
Total Objects Detected: 1
Object Breakdown:
  - Mixed Waste: 1
  - Organic Waste: 0
  - Other Waste: 0
  - Recyclable Material: 0
Confidence Scores:
  1. Mixed Waste - Confidence: 0.92
```


Object Count per Class
Count
2
1
0
Mixed Waste
Organic Waste
Other Waste
Recyclable Material
Classes


Confidence Scores by Class
Confidence Scores
1.0
0.8
0.6
0.4
0.2
0.0
Mixed Waste
Classes

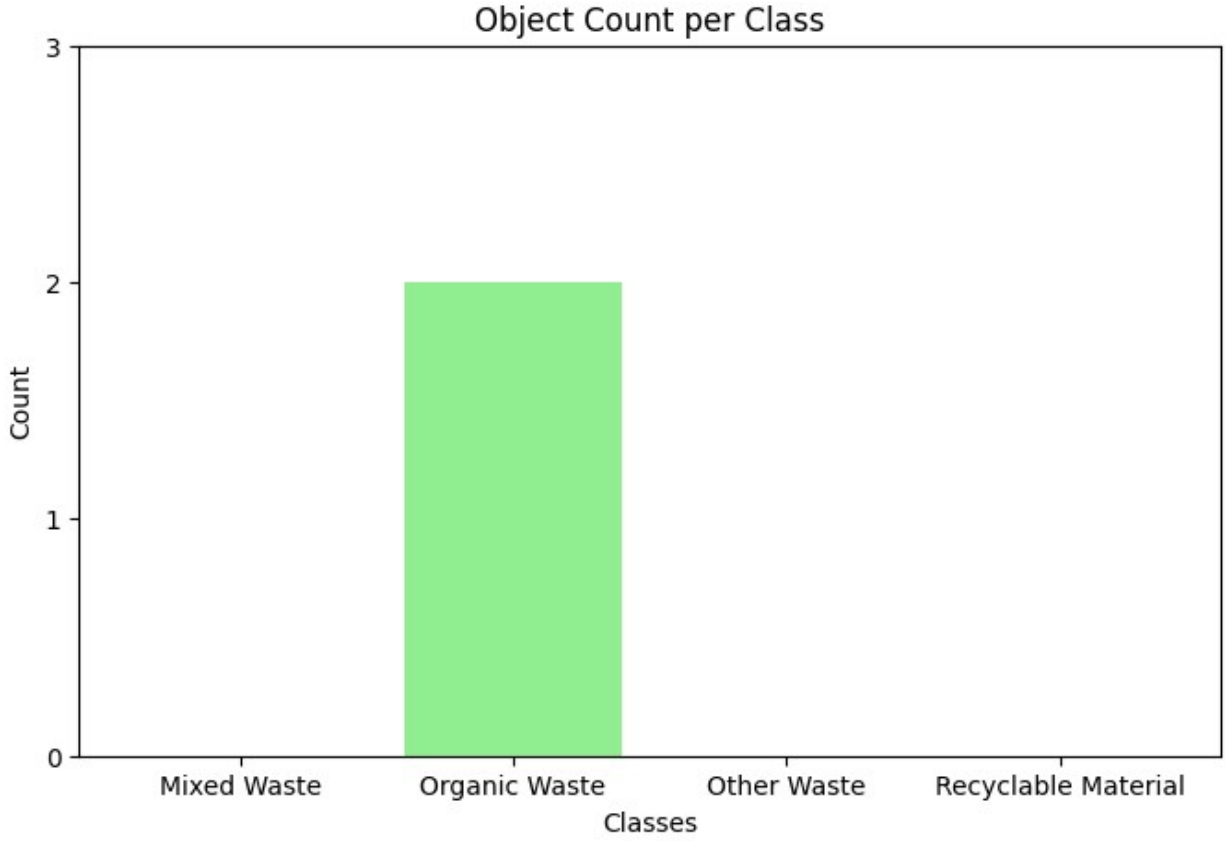
Mixed Waste - Conf: 0.92

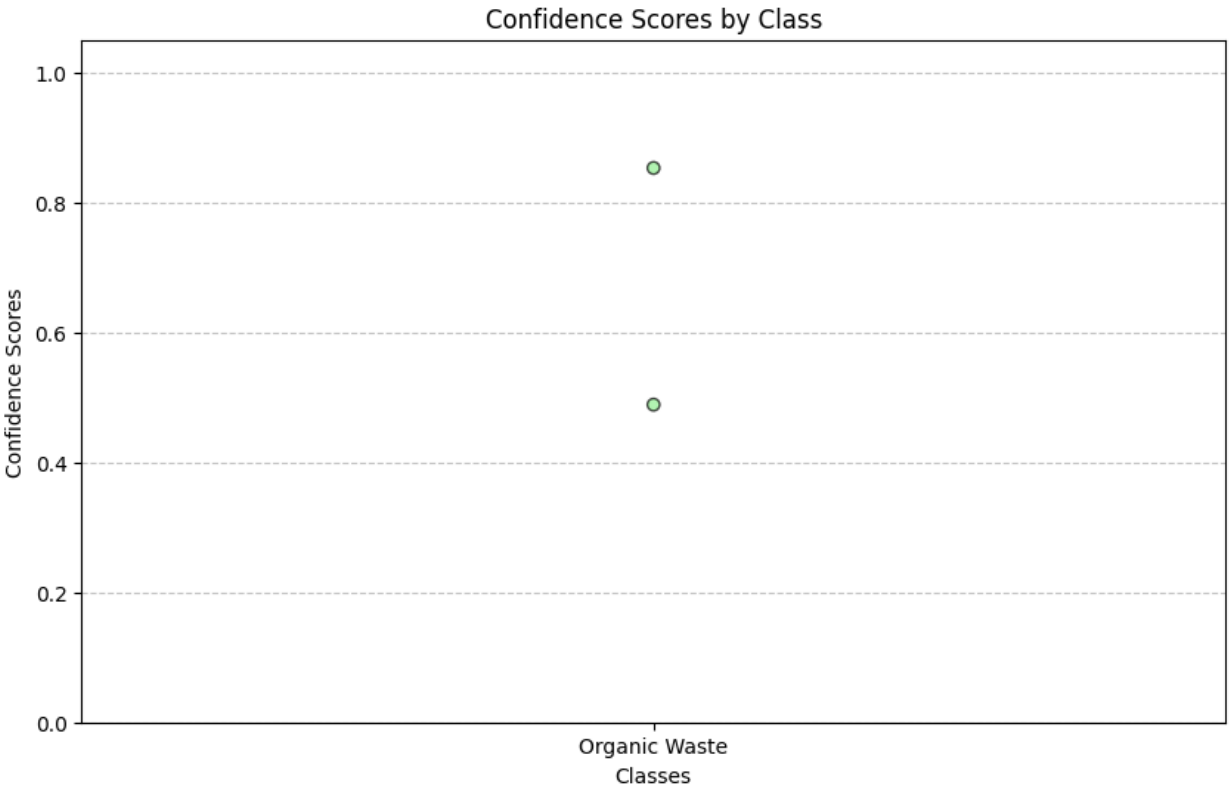
```
Processing test image: /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced CV/Assignments/ARI3129-MDD/Test/AFL_T1.jpeg

image 1/1 /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced CV/Assignments/ARI3129-MDD/Test/AFL_T1.jpeg:
640x480 3 Organic Wastes, 114.0ms
Speed: 7.8ms preprocess, 114.0ms inference, 13.2ms postprocess per
image at shape (1, 3, 640, 480)
```

Original Test Image

Predicted Image with Bounding Boxes

```
=== Image Analytics ===
Total Objects Detected: 2
Object Breakdown:
  - Mixed Waste: 0
  - Organic Waste: 2
  - Other Waste: 0
  - Recyclable Material: 0
Confidence Scores:
  1. Organic Waste - Confidence: 0.85
  2. Organic Waste - Confidence: 0.49
```





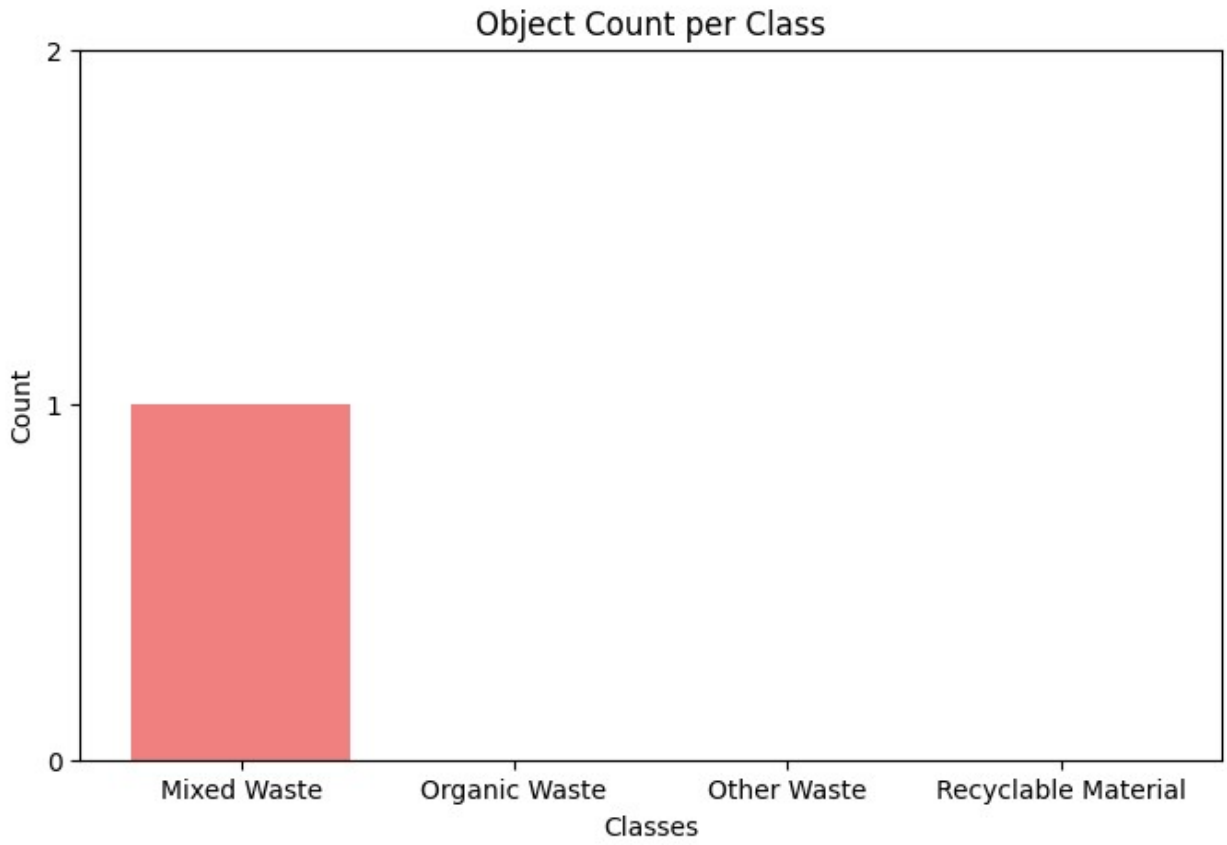

```
Processing test image: /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced CV/Assignments/ARI3129-MDD/Test/AFL_T5N.jpeg

image 1/1 /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced CV/Assignments/ARI3129-MDD/Test/AFL_T5N.jpeg:
640x480 1 Mixed Waste, 31.5ms
```

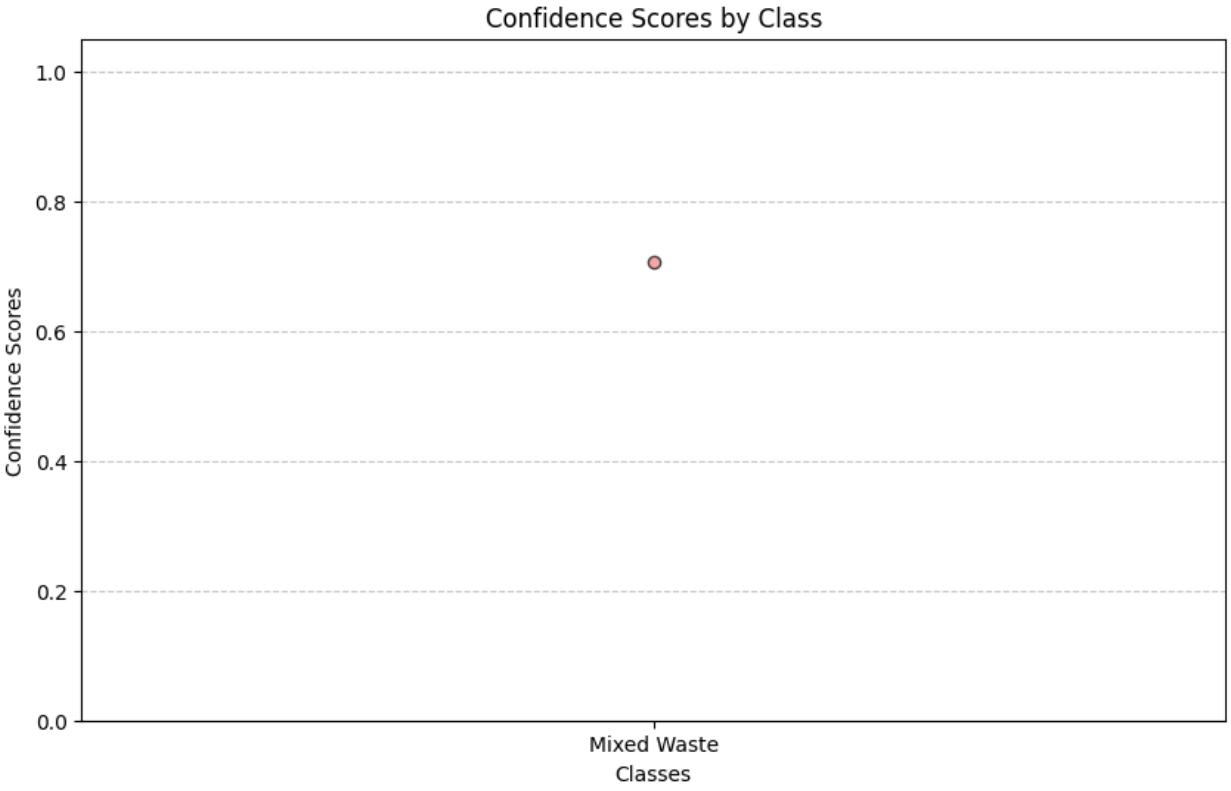
```
Speed: 4.6ms preprocess, 31.5ms inference, 163.0ms postprocess per
image at shape (1, 3, 640, 480)
```

Original Test Image

Predicted Image with Bounding Boxes


```
=== Image Analytics ===
Total Objects Detected: 1
Object Breakdown:
  - Mixed Waste: 1
  - Organic Waste: 0
  - Other Waste: 0
  - Recyclable Material: 0
Confidence Scores:
  1. Mixed Waste - Confidence: 0.71
```

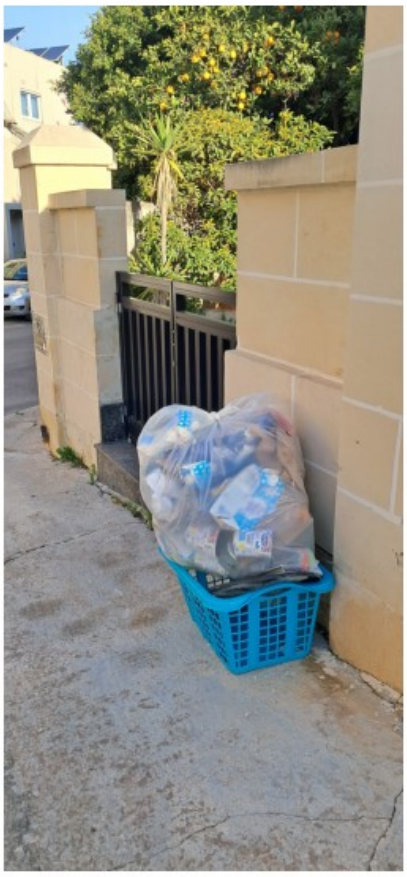

Object Count per Class
Count
2
1
0
Mixed Waste
Organic Waste
Other Waste
Recyclable Material
Classes


Confidence Scores by Class
Confidence Scores
1.0
0.8
0.6
0.4
0.2
0.0
Mixed Waste
Classes

Mixed Waste - Conf: 0.71

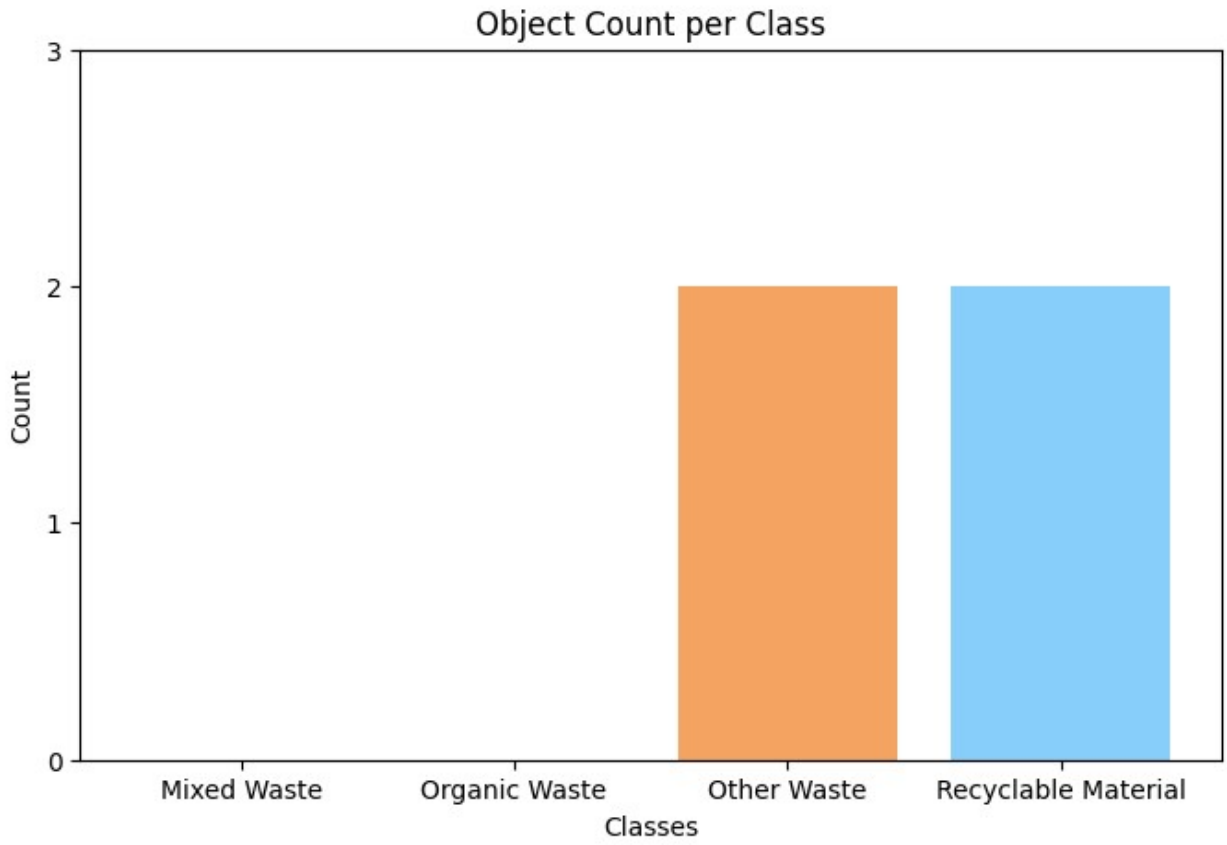
```
Processing test image: /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced CV/Assignments/ARI3129-MDD/Test/SDM_T2.jpg

image 1/1 /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced CV/Assignments/ARI3129-MDD/Test/SDM_T2.jpg:
640x320 3 Other Wastes, 2 Recyclable Materials, 719.3ms
Speed: 29.3ms preprocess, 719.3ms inference, 175.5ms postprocess per
image at shape (1, 3, 640, 320)
```

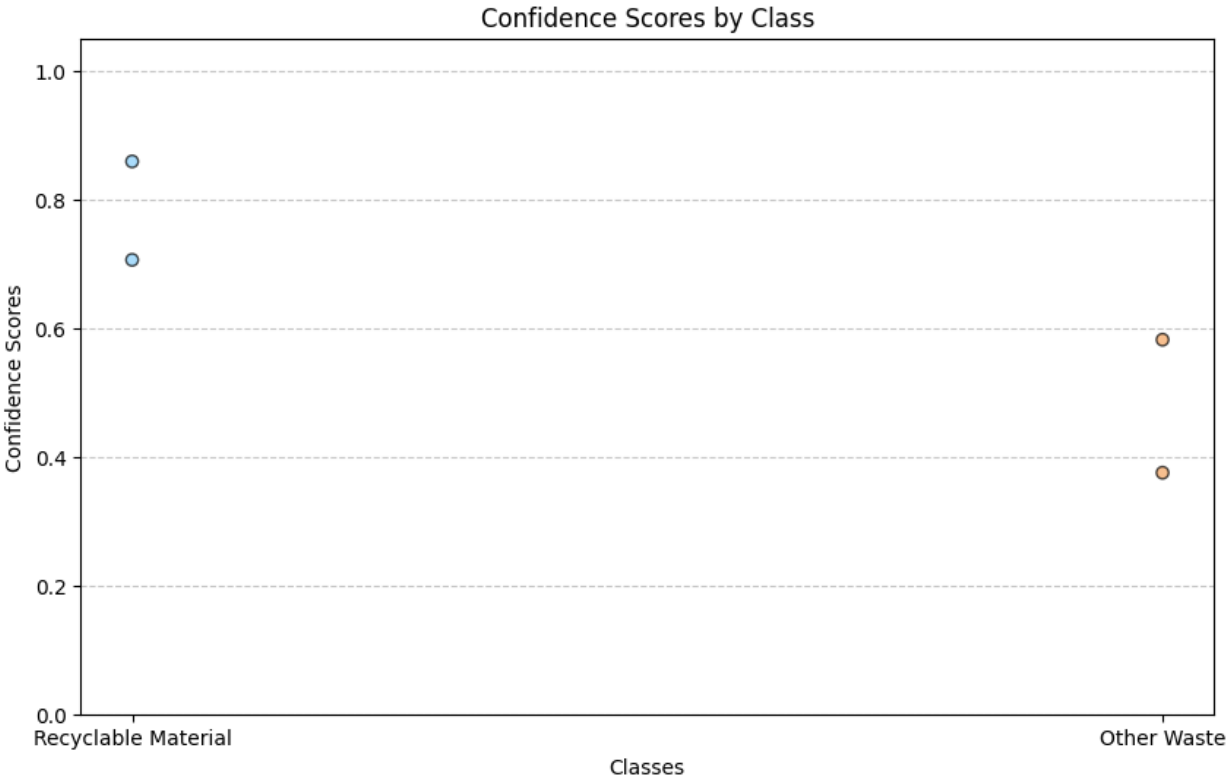
Original Test Image



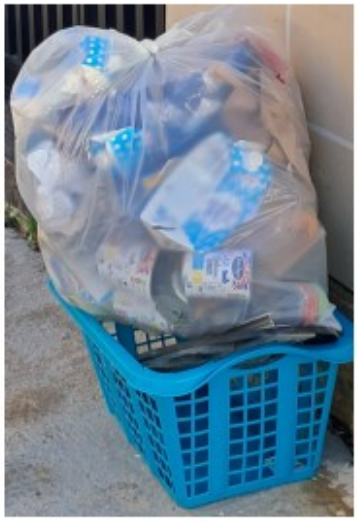
Predicted Image with Bounding Boxes


```
=== Image Analytics ===
Total Objects Detected: 4
Object Breakdown:
  - Mixed Waste: 0
  - Organic Waste: 0
  - Other Waste: 2
  - Recyclable Material: 2
Confidence Scores:
  1. Recyclable Material - Confidence: 0.86
  2. Recyclable Material - Confidence: 0.71
  3. Other Waste - Confidence: 0.58
  4. Other Waste - Confidence: 0.38
```


Object Count per Class
Count
Classes
Mixed Waste
Organic Waste
Other Waste
Recyclable Material
0
1
2
3


Confidence Scores by Class
Confidence Scores
Classes
Recyclable Material
Other Waste
0.0
0.2
0.4
0.6
0.8
1.0

Recyclable Material - Conf: 0.86



Recyclable Material - Conf: 0.71

Other Waste - Conf: 0.58

Other Waste - Conf: 0.38

```
Processing test image: /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced CV/Assignments/ARI3129-MDD/Test/AFL_T2N.jpeg

image 1/1 /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced CV/Assignments/ARI3129-MDD/Test/AFL_T2N.jpeg:
640x480 (no detections), 210.6ms
Speed: 7.9ms preprocess, 210.6ms inference, 4.5ms postprocess per
image at shape (1, 3, 640, 480)
```

Original Test Image

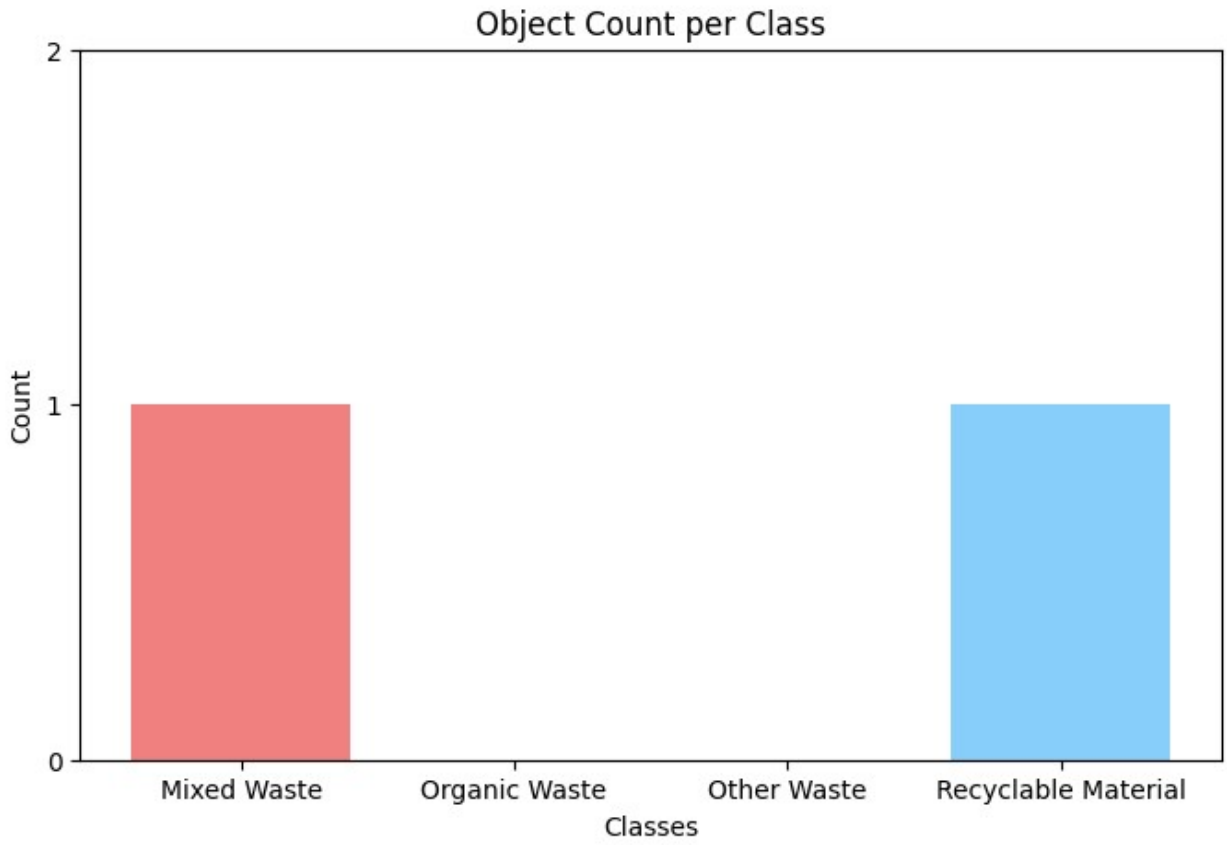
Predicted Image with Bounding Boxes

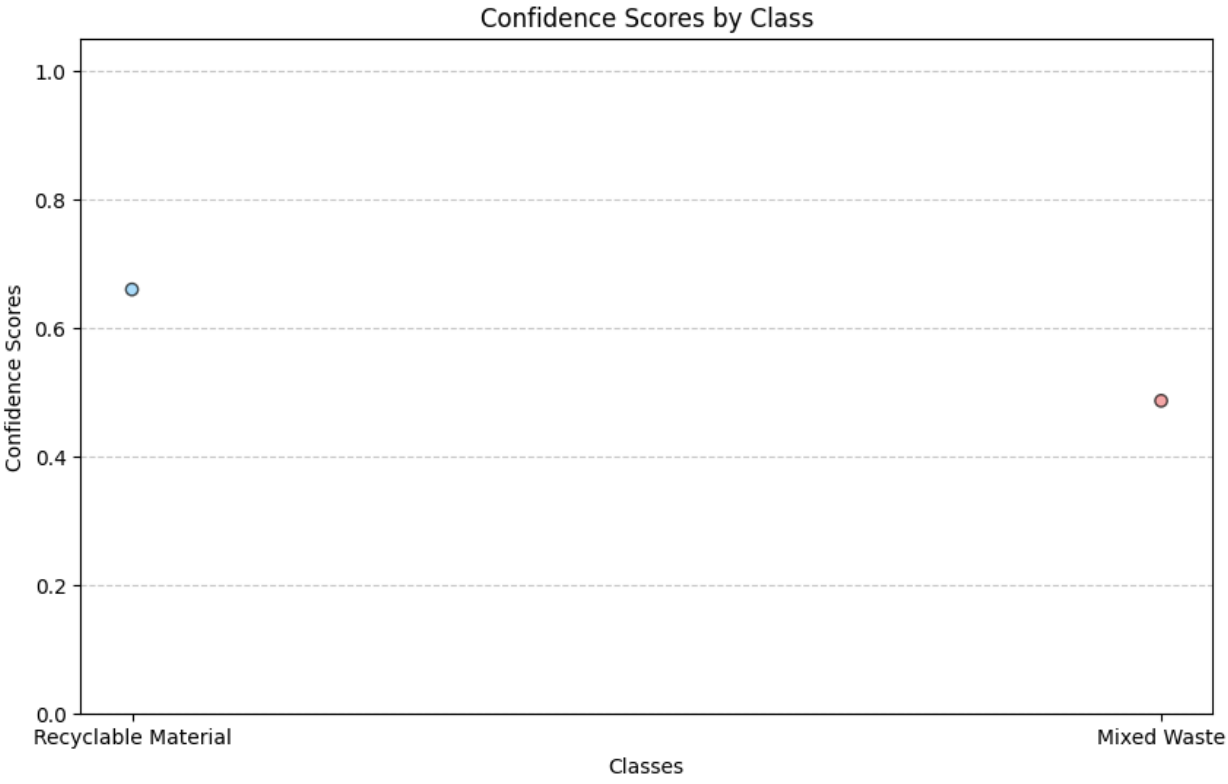
```
No (0) objects detected in the image.
Processing test image: /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced CV/Assignments/ARI3129-MDD/Test/SDM_T1.jpg

image 1/1 /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced CV/Assignments/ARI3129-MDD/Test/SDM_T1.jpg:
640x320 1 Mixed Waste, 1 Recyclable Material, 27.9ms
Speed: 5.2ms preprocess, 27.9ms inference, 193.4ms postprocess per
image at shape (1, 3, 640, 320)
```

Original Test Image

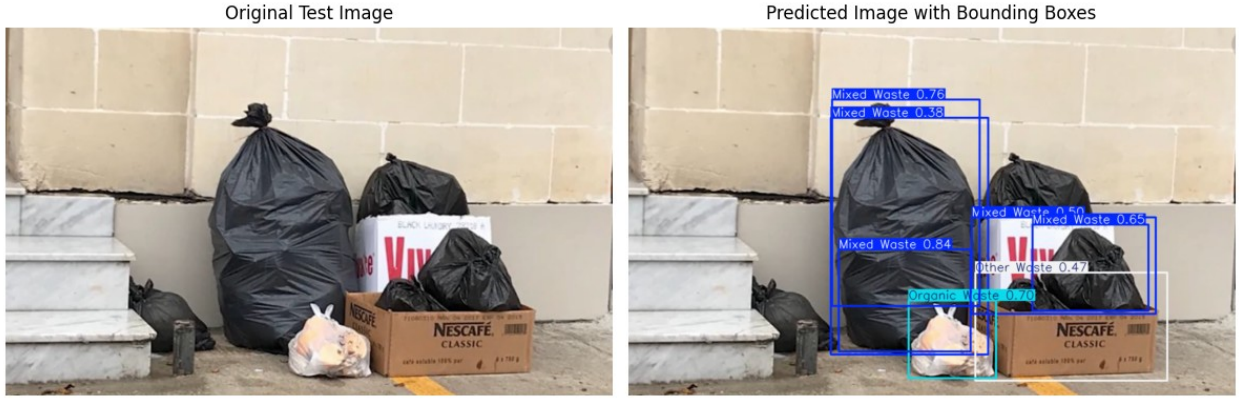
Predicted Image with Bounding Boxes


```
=== Image Analytics ===
Total Objects Detected: 2
Object Breakdown:
  - Mixed Waste: 1
  - Organic Waste: 0
  - Other Waste: 0
  - Recyclable Material: 1
Confidence Scores:
  1. Recyclable Material - Confidence: 0.66
  2. Mixed Waste - Confidence: 0.49
```


Object Count per Class
Count
2
1
0
Mixed Waste
Organic Waste
Other Waste
Recyclable Material
Classes

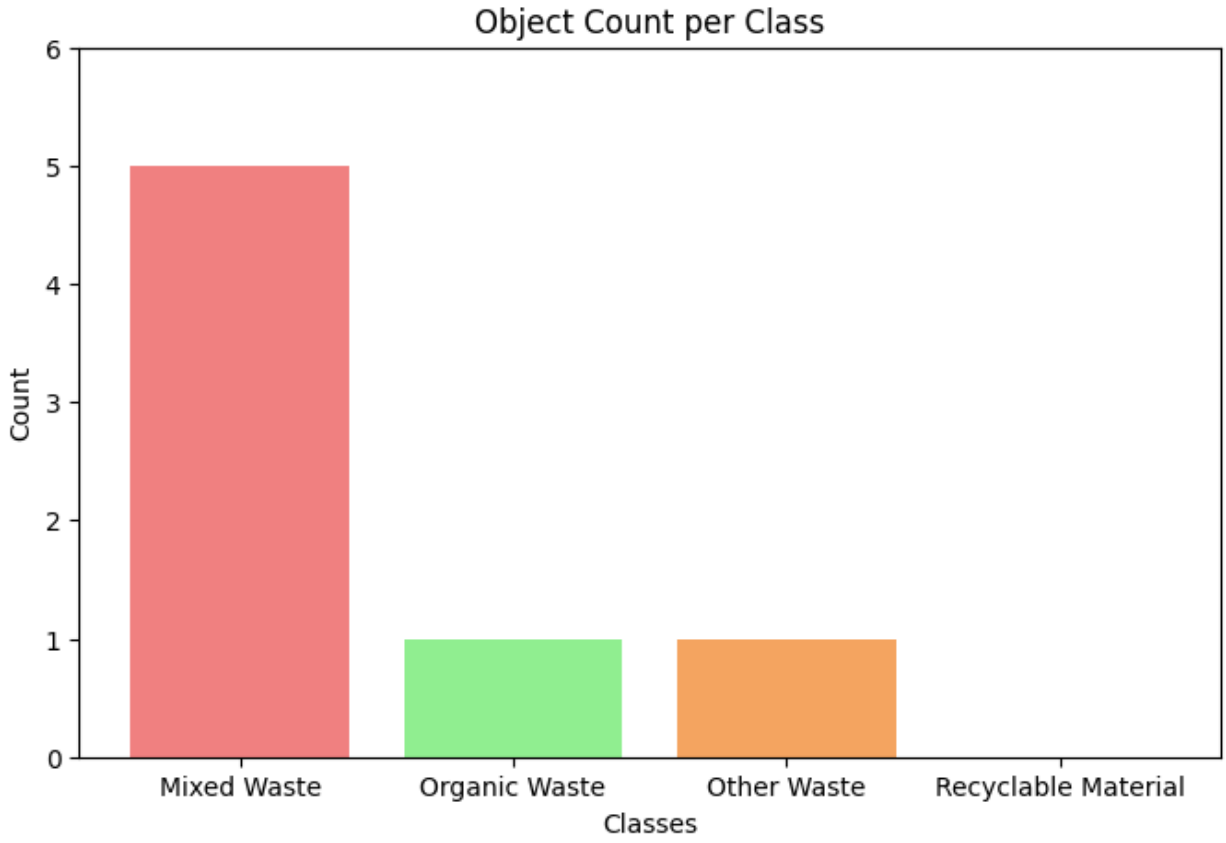

Confidence Scores by Class
Confidence Scores
1.0
0.8
0.6
0.4
0.2
0.0
Recyclable Material
Mixed Waste
Classes

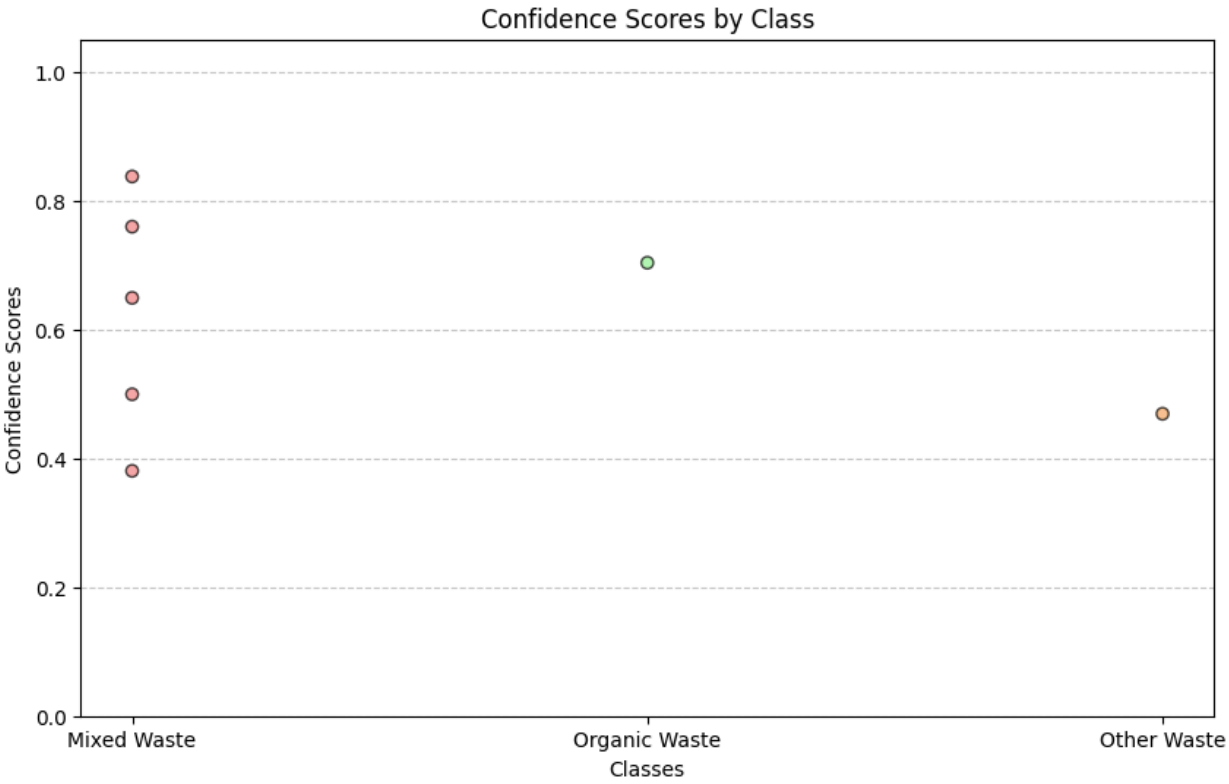

```
Processing test image: /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced CV/Assignments/ARI3129-MDD/Test/AFL_T4N.jpeg

image 1/1 /Users/afl/Documents/University/Year
3/Lectures/SEM1/Advanced CV/Assignments/ARI3129-MDD/Test/AFL_T4N.jpeg:
416x640 6 Mixed Wastes, 1 Organic Waste, 1 Other Waste, 1 Recyclable
Material, 444.3ms
Speed: 60.0ms preprocess, 444.3ms inference, 197.1ms postprocess per
image at shape (1, 3, 416, 640)
```




```
=== Image Analytics ===
Total Objects Detected: 7
Object Breakdown:
  - Mixed Waste: 5
  - Organic Waste: 1
  - Other Waste: 1
  - Recyclable Material: 0
```

```
Confidence Scores:
  1. Mixed Waste - Confidence: 0.84
  2. Mixed Waste - Confidence: 0.76
  3. Organic Waste - Confidence: 0.70
  4. Mixed Waste - Confidence: 0.65
  5. Mixed Waste - Confidence: 0.50
  6. Other Waste - Confidence: 0.47
  7. Mixed Waste - Confidence: 0.38
```




Confidence Scores by Class
Confidence Scores
1.0
0.8
0.6
0.4
0.2
0.0
Mixed Waste
Organic Waste
Other Waste
Classes

Mixed Waste - Conf: 0.84

Mixed Waste - Conf: 0.76

Organic Waste - Conf: 0.70

Mixed Waste - Conf: 0.65

Mixed Waste - Conf: 0.50

Other Waste - Conf: 0.47

Mixed Waste - Conf: 0.38