```
# Install PyTorch and torchvision
!pip install torch torchvision

# Install additional dependencies if necessary
!pip install <other-libraries>
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (2.1.0+cu121)
Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dist-packages (0.16.0+cu121)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch) (3.13.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from torch) (4.9.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch) (3.2.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch) (3.1.3)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch) (2023.6.0)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-packages (from torch) (2.1.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from torchvision) (1.25.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from torchvision) (2.31.0)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.10/dist-packages (from torchvision) (9.4.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch) (2.1.5)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision) (3.3
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision) (2024.2.2
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch) (1.3.0)
/bin/bash: -c: line 1: syntax error near unexpected token `newline'
/bin/bash: -c: line 1: `pip install <other-libraries>'
```

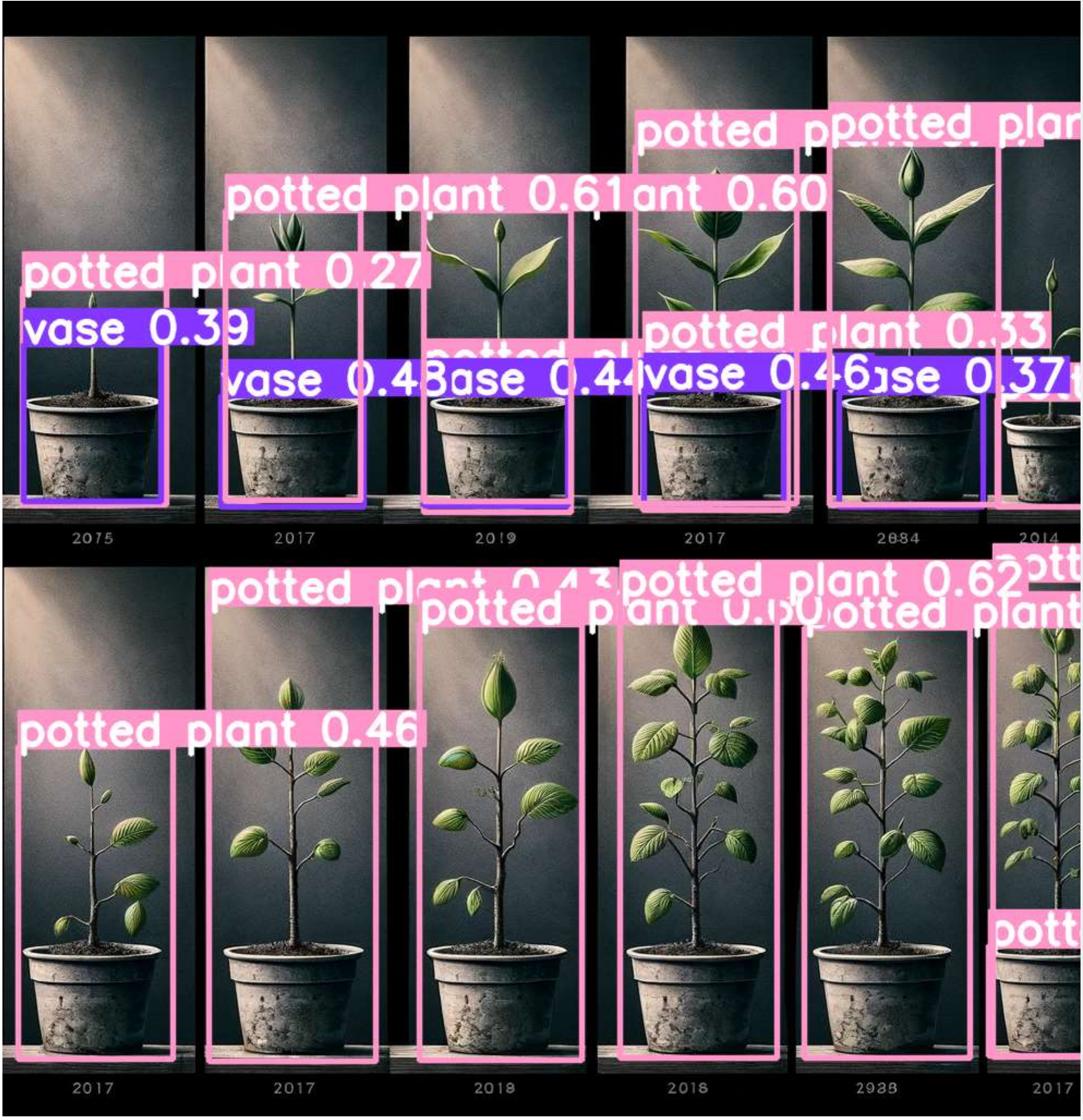
```
import torch
from PIL import Image
import matplotlib.pyplot as plt
from google.colab import files
import os

# Load the pre-trained YOLOv5 model
model = torch.hub.load('ultralytics/yolov5', 'yolov5s', pretrained=True)

def process_image(image_path):
    # Load an image
    img = Image.open(image_path)

    # Inference
    results = model(img)

    # Results
    results.print()  # Print results to console
    results.show()  # Show the image with bounding boxes

    return results

# Upload files
uploaded = files.upload()

# Assuming the uploaded files are in the current working directory, list them
image_files = [name for name in uploaded.keys()]

# Process each image
for image_path in image_files:
    print(f"Processing {image_path}...")
    results = process_image(image_path)
```

```
Using cache found in /root/.cache/torch/hub/ultralytics_yolov5_master
YOLOv5 🚀 2024-2-29 Python-3.10.12 torch-2.1.0+cu121 CPU

Fusing layers...
YOLOv5s summary: 213 layers, 7225885 parameters, 0 gradients, 16.4 GFLOPs
Adding AutoShape...
```

Choose Files image,jpg.webp

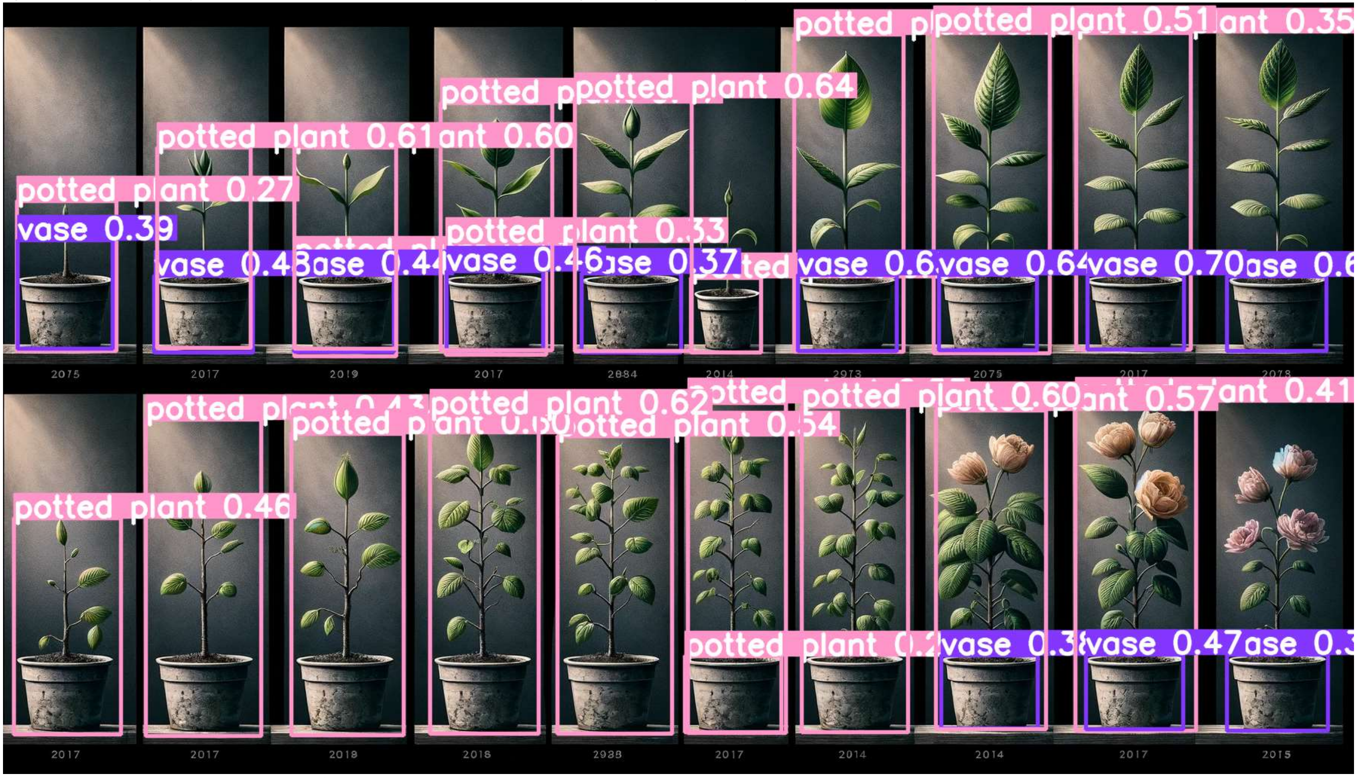
- **image,jpg.webp**(image/webp) - 570232 bytes, last modified: 2/28/2024 - 100% done

```
Saving image,jpg.webp to image,jpg (4).webp
Processing image,jpg (4).webp...
image 1/1: 1024x1792 21 potted plants, 12 vases
Speed: 104.4ms pre-process, 252.2ms inference, 26.6ms NMS per image at shape (1, 3, 384, 640)
```



```
# Example: Counting detected objects of a certain class across images

# Initialize a dictionary to hold counts of objects
object_counts = {}

# Loop through processed images
for image_path in image_files:
    results = process_image(image_path)

    # Extract detected object names for this image
    detected_objects = results.pandas().xyxy[0]['name']

    # Count occurrences of each object
    for object_name in detected_objects:
        if object_name in object_counts:
            object_counts[object_name] += 1
        else:
            object_counts[object_name] = 1

# Print out counts
print(object_counts)
```

```
image 1/1: 1024x1792 21 potted plants, 12 vases
Speed: 75.1ms pre-process, 248.1ms inference, 3.2ms NMS per image at shape (1, 3, 384, 640)
```

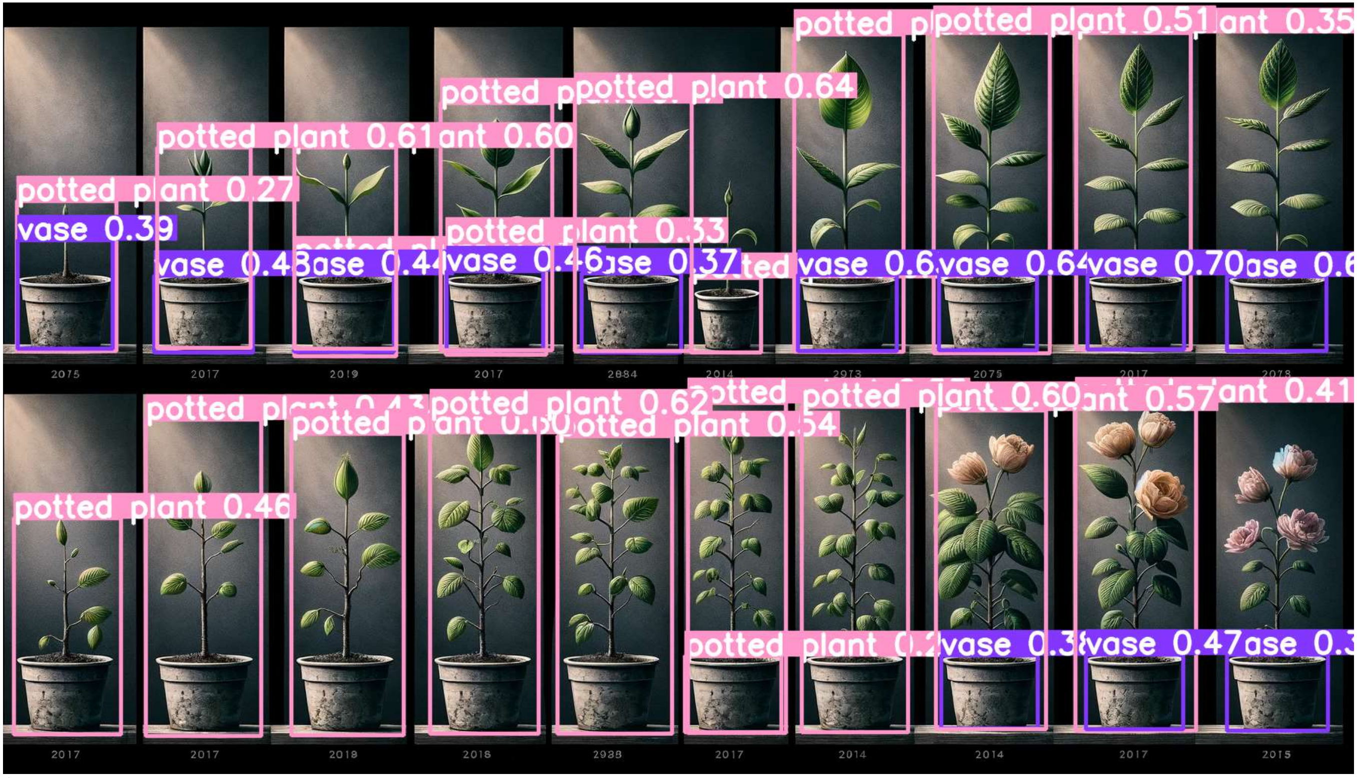


```
{'vase': 12, 'potted plant': 21}
```

```
# Import necessary libraries
import matplotlib.pyplot as plt
import numpy as np

# Assuming you have a list of object counts per image
# Example data: Number of objects detected in each image
object_counts = [5, 7, 6, 9, 10, 11, 15, 13, 14, 16]  # Replace with your actual data

# Image indices or timestamps for the x-axis
# If you have timestamps, you might need to convert them to a suitable format
image_indices = np.arange(1, len(object_counts) + 1)  # This generates a simple sequence

# Plotting
plt.figure(figsize=(10, 6))  # Set the figure size
plt.plot(image_indices, object_counts, marker='o', linestyle='-', color='b')  # Plot data
plt.title('Object Counts Over Images')  # Title of the plot
plt.xlabel('Image Index')  # X-axis label
plt.ylabel('Count of Objects')  # Y-axis label
plt.xticks(image_indices)  # Ensure every image index is marked
plt.grid(True)  # Show grid
plt.show()  # Display the plot
```

```
results = process_image(image_path)
results.print()  # Print results to console
```

```
image 1/1: 1024x1792 21 potted plants, 12 vases
Speed: 71.2ms pre-process, 237.4ms inference, 1.5ms NMS per image at shape (1, 3, 384, 640)
```



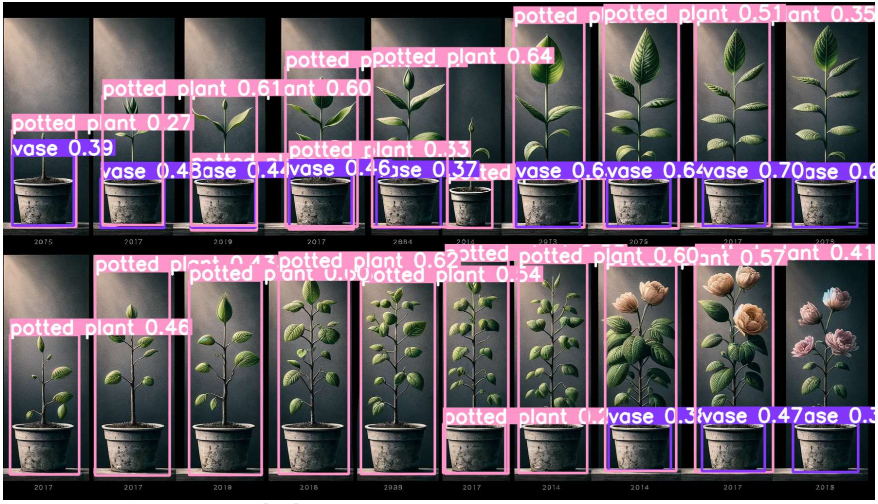
```
image 1/1: 1024x1792 21 potted plants, 12 vases
Speed: 71.2ms pre-process, 237.4ms inference, 1.5ms NMS per image at shape (1, 3, 384, 640)
```

```
print("Object counts:", object_counts)
```

```
Object counts: [5, 7, 6, 9, 10, 11, 15, 13, 14, 16]
```

```
# Assuming 'image_files' is a list of image paths
object_counts = []  # List to hold the count of objects per image
object_counts = []  # List to hold the count of objects per image

for image_path in image_files:
    results = process_image(image_path)  # Process the image
    detected_objects = results.pandas().xyxy[0]  # Extract detection results
    count = len(detected_objects)  # Get the number of objects detected
    object_counts.append(count)  # Append the count to our list

# Print the collected data to verify
print("Collected object counts:", object_counts)
```

```
image 1/1: 1024x1792 21 potted plants, 12 vases
Speed: 64.9ms pre-process, 231.8ms inference, 1.4ms NMS per image at shape (1, 3, 384
```



```
Collected object counts: [33]
```