```
import zipfile
import os

zip_path = 'archive.zip'
extract_folder = 'pothole_images'

try:
    with zipfile.ZipFile(zip_path, 'r') as zip_ref:
        zip_ref.extractall(extract_folder)
    print(f"Extraction complete. Files extracted to '{extract_folder}'")
except zipfile.BadZipFile:
    print("BadZipFile error: The zip file is corrupted or incomplete.")
except Exception as e:
    print(f"An error occurred: {e}")
```

```
!pip install ultralytics
```

```
import warnings
warnings.filterwarnings('ignore')
import os
import shutil
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import random
import cv2
import yaml
from PIL import Image
from collections import deque
from ultralytics import YOLO
from IPython.display import Video
```

```
Creating new Ultralytics Settings v0.0.6 file ✅
View Ultralytics Settings with 'yolo settings' or at '/root/.config/Ultralytics/settings.json'
Update Settings with 'yolo settings key=value', i.e. 'yolo settings runs_dir=path/to/dir'. For help see https://docs.ultralytics.cor
```

```
sns.set(rc={'axes.facecolor': '#ffe4de'}, style='darkgrid')
```

```
model = YOLO('yolov8n-seg.pt')
```

```
Downloading https://github.com/ultralytics/assets/releases/download/v8.3.0/yolov8n-seg.pt to 'yolov8n-seg.pt'...
100%|██████████| 6.74M/6.74M [00:00<00:00, 359MB/s]
```

```
# Define the dataset_path
dataset_path = '/content/pothole_images/Pothole_Segmentation_YOLOv8'

# Set the path to the YAML file
yaml_file_path = os.path.join(dataset_path, 'data.yaml')

# Load and print the contents of the YAML file
with open(yaml_file_path, 'r') as file:
    yaml_content = yaml.load(file, Loader=yaml.FullLoader)
    print(yaml.dump(yaml_content, default_flow_style=False))
```

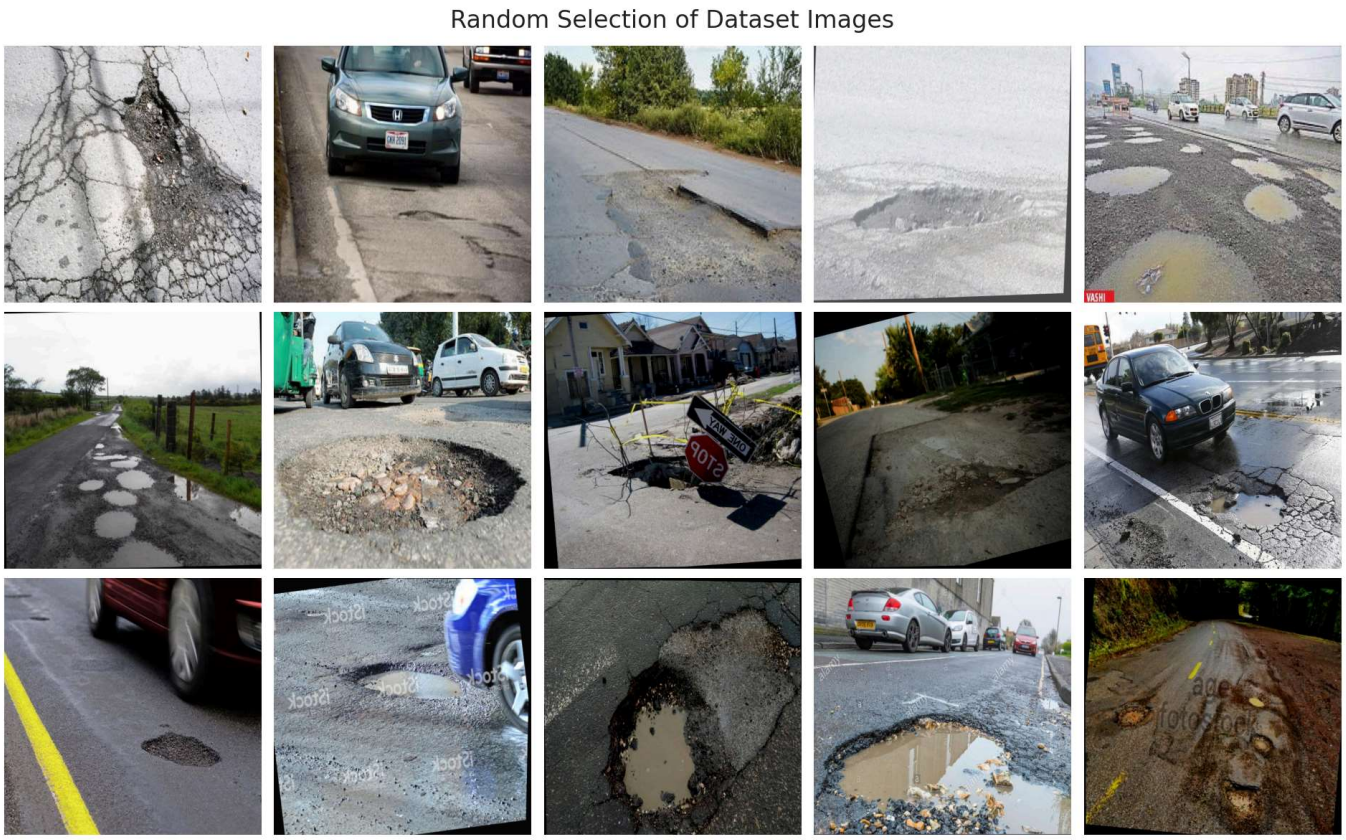
```
names:
- Pothole
nc: 1
roboflow:
  license: CC BY 4.0
  project: pothole_segmentation_yolov8
  url: https://universe.roboflow.com/farzad/pothole_segmentation_yolov8/dataset/1
  version: 1
  workspace: farzad
train: ../train/images
val: ../valid/images
```

```
# Set paths for training and validation image sets
train_images_path = os.path.join(dataset_path, 'train', 'images')
valid_images_path = os.path.join(dataset_path, 'valid', 'images')

# Initialize counters for the number of images
num_train_images = 0
num_valid_images = 0
```

```
# Initialize sets to hold the unique sizes of images
train_image_sizes = set()
valid_image_sizes = set()

# Check train images sizes and count
for filename in os.listdir(train_images_path):
    if filename.endswith('.jpg'):
        num_train_images += 1
        image_path = os.path.join(train_images_path, filename)
        with Image.open(image_path) as img:
            train_image_sizes.add(img.size)

# Check validation images sizes and count
for filename in os.listdir(valid_images_path):
    if filename.endswith('.jpg'):
        num_valid_images += 1
        image_path = os.path.join(valid_images_path, filename)
        with Image.open(image_path) as img:
            valid_image_sizes.add(img.size)

# Print the results
print(f"Number of training images: {num_train_images}")
print(f"Number of validation images: {num_valid_images}")

# Check if all images in training set have the same size
if len(train_image_sizes) == 1:
    print(f"All training images have the same size: {train_image_sizes.pop()}")
else:
    print("Training images have varying sizes.")

# Check if all images in validation set have the same size
if len(valid_image_sizes) == 1:
    print(f"All validation images have the same size: {valid_image_sizes.pop()}")
else:
    print("Validation images have varying sizes.")
```

```
Number of training images: 720
Number of validation images: 60
All training images have the same size: (640, 640)
All validation images have the same size: (640, 640)
```

```
# Set the seed for the random number generator
random.seed(0)

# Create a list of image files
image_files = [f for f in os.listdir(train_images_path) if f.endswith('.jpg')]

# Randomly select 15 images
random_images = random.sample(image_files, 15)

# Create a new figure
plt.figure(figsize=(19, 12))

# Loop through each image and display it in a 3x5 grid
for i, image_file in enumerate(random_images):
    image_path = os.path.join(train_images_path, image_file)
    image = Image.open(image_path)
    plt.subplot(3, 5, i + 1)
    plt.imshow(image)
    plt.axis('off')

# Add a suptitle
plt.suptitle('Random Selection of Dataset Images', fontsize=24)

# Show the plot
plt.tight_layout()
plt.show()

# Deleting unnecessary variable to free up memory
del image_files
```



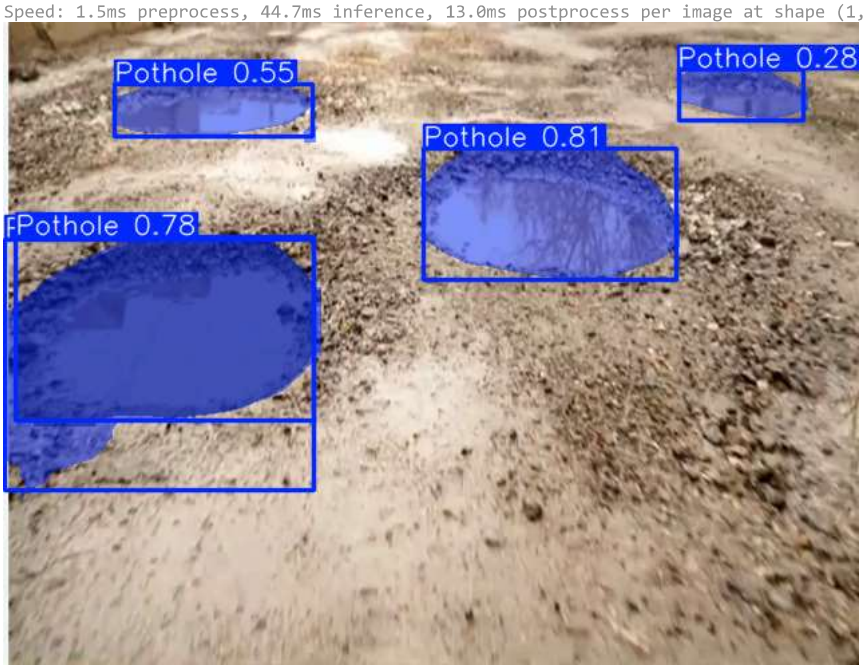

```
results = model.train(
    data=yaml_file_path,     # Path to the dataset configuration file
    epochs=100,              # Number of epochs to train for
    imgsz=640,               # Size of input images as integer
    patience=15,             # Epochs to wait for no observable improvement for early stopping of training
    batch=16,                # Number of images per batch
    optimizer='auto',        # Optimizer to use, choices=[SGD, Adam, Adamax, AdamW, NAdam, RAdam, RMSProp, auto]
    lr0=0.0001,              # Initial learning rate
    lrf=0.01,                # Final learning rate (lr0 * lrf)
    dropout=0.25,            # Use dropout regularization
    device=0,                # Device to run on, i.e. cuda device=0
    seed=42                  # Random seed for reproducibility
)
```

```
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.0001' and 'momentum=0.937' and determining best 'optimizer', 'lr0' and 'moment
```

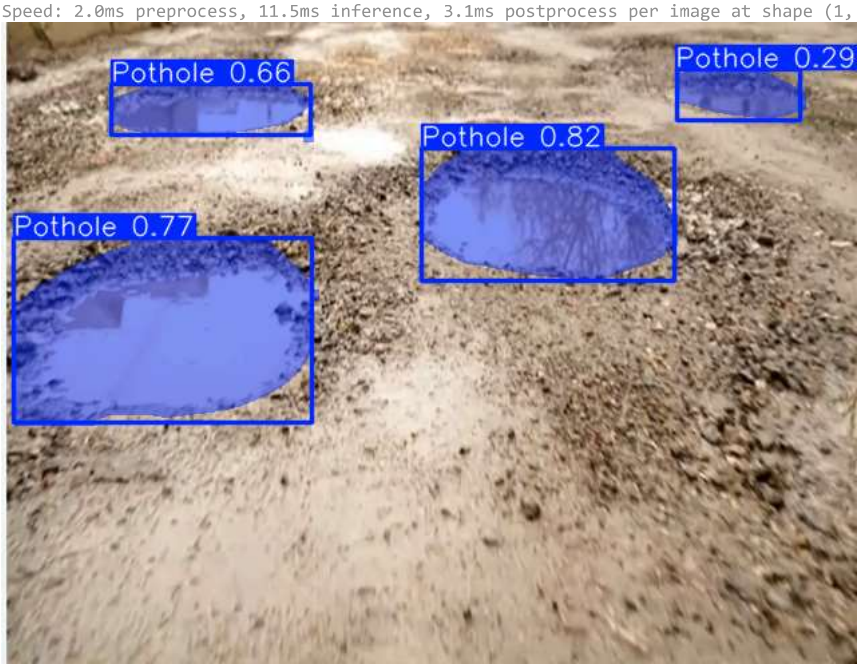
```
optimizer: AdamW(lr=0.002, momentum=0.9) with parameter groups 66 weight(decay=0.0), 77 weight(decay=0.0005), 76 bias(decay=0.0)
Image sizes 640 train, 640 val
Using 2 dataloader workers
Logging results to runs/segment/train
Starting training for 100 epochs...

      Epoch    GPU_mem   box_loss   seg_loss   cls_loss   dfl_loss  Instances       Size
      1/100      2.67G      1.478      2.914      2.339      1.438         62        640: 100%|██████████| 45/45 [00:22<00:00,  2
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95)     Mask(P          R      mAP50  mAP50-

      Epoch    GPU_mem   box_loss   seg_loss   cls_loss   dfl_loss  Instances       Size
      2/100      3.29G      1.478      2.523      1.843      1.423         92        640: 100%|██████████| 45/45 [00:18<00:00,  2
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95)     Mask(P          R      mAP50  mAP50-

      Epoch    GPU_mem   box_loss   seg_loss   cls_loss   dfl_loss  Instances       Size
      3/100       3.3G      1.515      2.554      1.757      1.448         84        640: 100%|██████████| 45/45 [00:17<00:00,  2
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95)     Mask(P          R      mAP50  mAP50-

      Epoch    GPU_mem   box_loss   seg_loss   cls_loss   dfl_loss  Instances       Size
      4/100      3.33G      1.511      2.446      1.647      1.445         60        640: 100%|██████████| 45/45 [00:17<00:00,  2
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95)     Mask(P          R      mAP50  mAP50-

      Epoch    GPU_mem   box_loss   seg_loss   cls_loss   dfl_loss  Instances       Size
      5/100      3.34G      1.498      2.396      1.573      1.415         71        640: 100%|██████████| 45/45 [00:17<00:00,  2
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95)     Mask(P          R      mAP50  mAP50-

      Epoch    GPU_mem   box_loss   seg_loss   cls_loss   dfl_loss  Instances       Size
      6/100      3.36G      1.446      2.374      1.499      1.394         83        640: 100%|██████████| 45/45 [00:16<00:00,  2
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95)     Mask(P          R      mAP50  mAP50-
```

```
from google.colab.patches import cv2_imshow
# Load the trained model
trained_model = YOLO('/content/runs/segment/train/weights/best.pt')  # Adjust if saved elsewhere

# Path to video file (you can use 0 for webcam)
video_path = '/content/pothole_images/Pothole_Segmentation_YOLOv8/sample_video-2.mp4'  # Change to 0 for webcam

# Create a VideoCapture object
cap = cv2.VideoCapture(video_path)

# Define codec and create VideoWriter object to save output (optional)
save_output = True
if save_output:
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    out = cv2.VideoWriter('pothole_detection_output2.mp4', fourcc, 20.0, (640, 480))

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    # Resize frame for consistency (optional)
    resized_frame = cv2.resize(frame, (640, 480))

    # Run detection
    results = trained_model(resized_frame)

    # Plot results on frame
    annotated_frame = results[0].plot()  # YOLOv8 returns list of results

    # Display the frame
    cv2_imshow(annotated_frame)

    # Save the frame if needed
    if save_output:
        out.write(annotated_frame)

    # Press 'q' to quit early
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
if save_output:
    out.release()
cv2.destroyAllWindows()
```

```
0: 480x640 5 Potholes, 44.7ms
Speed: 1.5ms preprocess, 44.7ms inference, 13.0ms postprocess per image at shape (1, 3, 480, 640)
```




```
0: 480x640 4 Potholes, 11.5ms
Speed: 2.0ms preprocess, 11.5ms inference, 3.1ms postprocess per image at shape (1, 3, 480, 640)
```




```
0: 480x640 4 Potholes, 13.7ms
Speed: 2.3ms preprocess, 13.7ms inference, 3.5ms postprocess per image at shape (1, 3, 480, 640)
```