```
import warnings
warnings.filterwarnings('ignore')
!pip install ultralytics
# Import necessary libraries
import os
import shutil
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import cv2
import yaml
from PIL import Image
from ultralytics import YOLO
from IPython.display import Video
```

```
Collecting ultralytics
  Downloading ultralytics-8.3.32-py3-none-any.whl.metadata (35 kB)
Requirement already satisfied: numpy>=1.23.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (1.26.4)
Requirement already satisfied: matplotlib>=3.3.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (3.8.0)
Requirement already satisfied: opencv-python>=4.6.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (4.10.0.84)
Requirement already satisfied: pillow>=7.1.2 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (11.0.0)
Requirement already satisfied: pyyaml>=5.3.1 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (6.0.2)
Requirement already satisfied: requests>=2.23.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (2.32.3)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (1.13.1)
Requirement already satisfied: torch>=1.8.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (2.5.1+cu121)
Requirement already satisfied: torchvision>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (0.20.1+cu121)
Requirement already satisfied: tqdm>=4.64.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (4.66.6)
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages (from ultralytics) (5.9.5)
Requirement already satisfied: py-cpuinfo in /usr/local/lib/python3.10/dist-packages (from ultralytics) (9.0.0)
Requirement already satisfied: pandas>=1.1.4 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (2.2.2)
Requirement already satisfied: seaborn>=0.11.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (0.13.2)
Collecting ultralytics-thop>=2.0.0 (from ultralytics)
  Downloading ultralytics_thop-2.0.11-py3-none-any.whl.metadata (9.4 kB)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics) (4.54.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics) (1.4.7)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics) (24.2)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.1.4->ultralytics) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.1.4->ultralytics) (2024.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.23.0->ultralytics) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.23.0->ultralytics) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.23.0->ultralytics) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.23.0->ultralytics) (2024.8.30)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->ultralytics) (3.16.1)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->ultralytics) (4.12.2)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->ultralytics) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->ultralytics) (3.1.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->ultralytics) (2024.10.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->ultralytics) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from sympy==1.13.1->torch>=1.8.0->ultralytics) (1.3.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib>=3.3.0->ultralytics) (1.16.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch>=1.8.0->ultralytics) (3.0.2)
Downloading ultralytics-8.3.32-py3-none-any.whl (887 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 887.0/887.0 kB 15.2 MB/s eta 0:00:00
Downloading ultralytics_thop-2.0.11-py3-none-any.whl (26 kB)
Installing collected packages: ultralytics-thop, ultralytics
Successfully installed ultralytics-8.3.32 ultralytics-thop-2.0.11
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.com/quickstart/#ultralytics-settings.
```

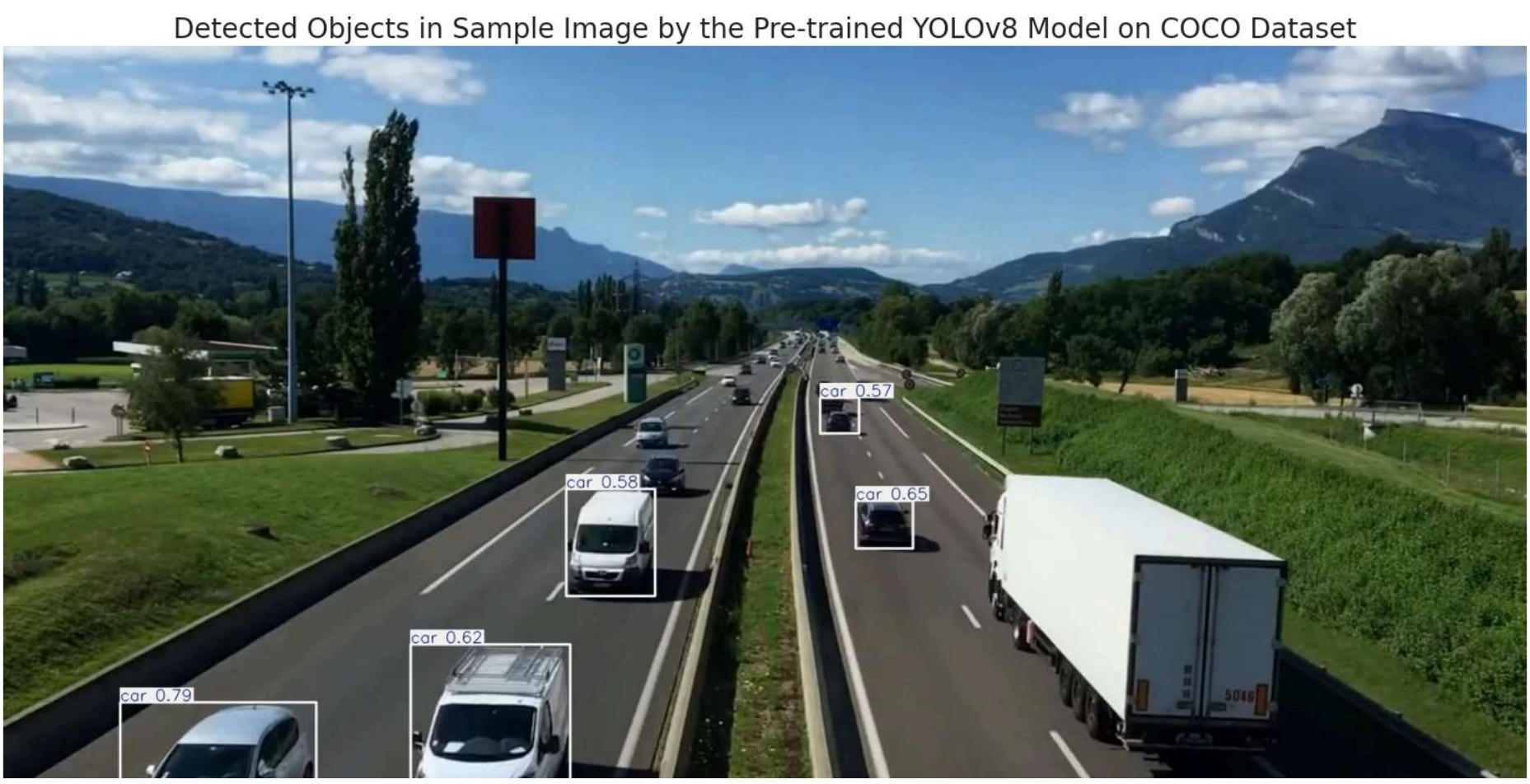
```
# Configure the visual appearance of seaborn plots
sns.set(rc={'axes.facecolor': '#eae8fa'}, style='darkgrid')


# Loading YOLO Pretrained model
model = YOLO('yolov8n.pt')
```

```
Downloading https://github.com/ultralytics/assets/releases/download/v8.3.0/yolov8n.pt to 'yolov8n.pt'...
100%|██████████| 6.25M/6.25M [00:00<00:00, 73.0MB/s]
```

```
import zipfile
import os

file_name = "/content/drive/MyDrive/Untitled folder/archive (2).zip"  # Replace with the name of your uploaded ZIP file

with zipfile.ZipFile(file_name, 'r') as zip_ref:
    zip_ref.extractall("extracted_folder")  # Specify the folder where files will be extracted
    print("Files extracted successfully!")
```

```
Files extracted successfully!
```

```
# Path to the image file
image_path = '/content/extracted_folder/Vehicle_Detection_Image_Dataset/sample_image.jpg'

# Perform inference on the provided image(s)
results = model.predict(source=image_path,
                        imgsz=640,
                        verbose=False,# Resize image to 640x640 (the size pf images the model was trained on)
                        conf=0.5)   # Confidence threshold: 50% (only detections above 50% confidence will be considered)

# Annotate and convert image to numpy array
sample_image = results[0].plot(line_width=2)

# Convert the color of the image from BGR to RGB for correct color representation in matplotlib
sample_image = cv2.cvtColor(sample_image, cv2.COLOR_BGR2RGB)

# Display annotated image
plt.figure(figsize=(20,15))
plt.imshow(sample_image)
plt.title('Detected Objects in Sample Image by the Pre-trained YOLOv8 Model on COCO Dataset', fontsize=20)
plt.axis('off')
plt.show()
```




```
# Define the dataset_path
dataset_path = '/content/extracted_folder/Vehicle_Detection_Image_Dataset'

# Set the path 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))
```

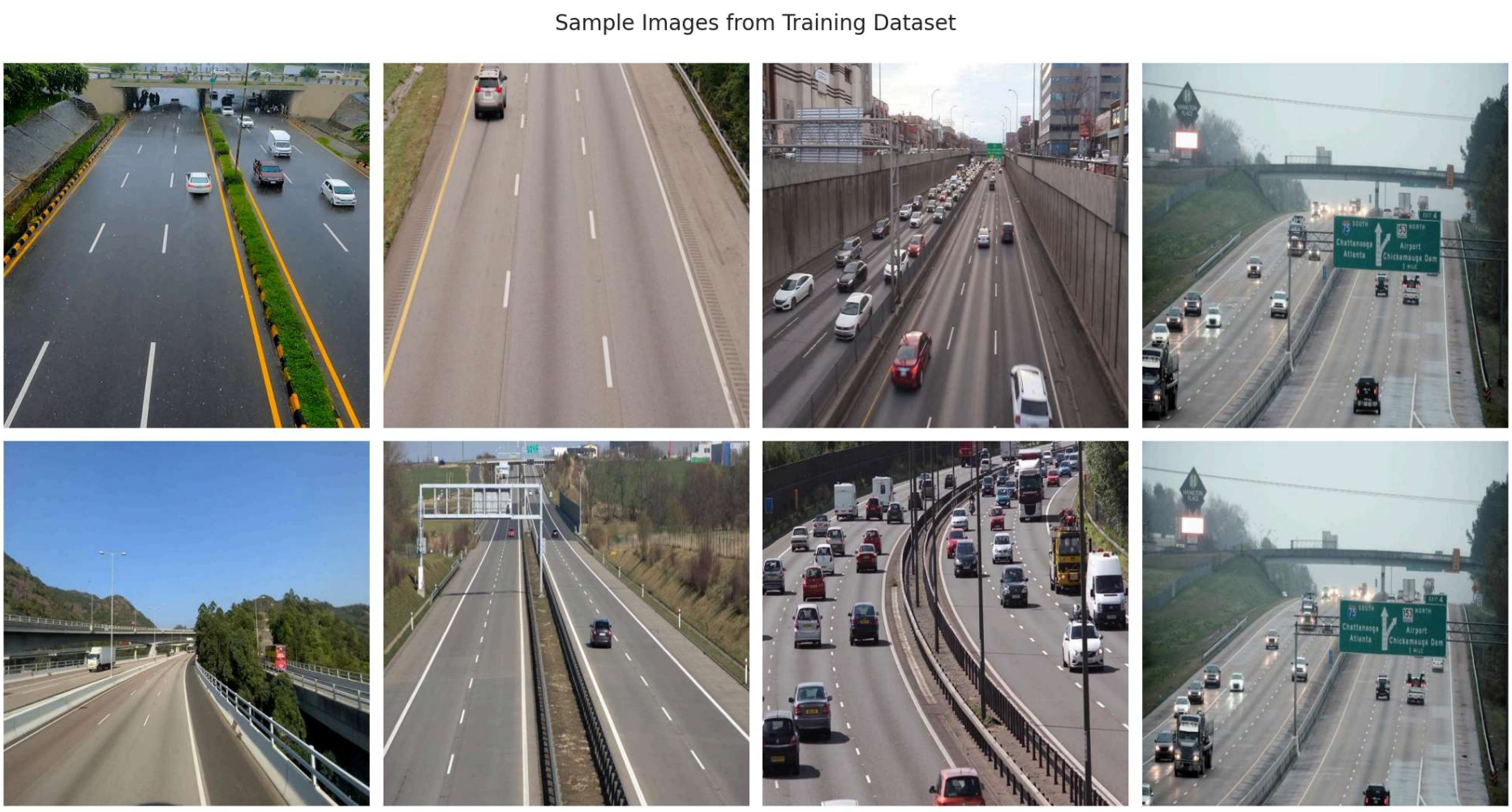
```
names:
- Vehicle
nc: 1
roboflow:
  license: CC BY 4.0
  project: vehicle_detection_yolov8
  url: https://universe.roboflow.com/farzad/vehicle_detection_yolov8/dataset/3
  version: 3
  workspace: farzad
train: ../train/images
val: ../valid/images
```

```
# Set paths for trainig and validation images sets
trian_image_path = os.path.join(dataset_path, 'train', 'images')
validation_image_path = os.path.join(dataset_path, 'valid', 'images')

# Initialize counters for the number of images
num_trian_images = 0
num_valid_images = 0

# Initialize sets to hold the unique sizes of images
train_image_size = set()
valid_image_size = set()

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

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

# Print the resutls
print(f"Number of training images:, {num_trian_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_size) == 1:
    print(f"All training images have the same size: {train_image_size.pop()}")
else:
    print("Training images have verying sizes.")

# Check if all images is validation set have same size
if len(valid_image_size) == 1:
    print(f"All validation images have the same size: {valid_image_size.pop()}")

else:
    print("Validation images have verying sizes")
```

```
Number of training images:, 536
Number of validation images: 90
All training images have the same size: (640, 640)
All validation images have the same size: (640, 640)
```

```
# List all jpg images in the directory

image_files = [file for file in os.listdir(trian_image_path) if file.endswith('.jpg')]

# Select 8 images as eqal intervals
num_images = len(image_files)
selected_images = [image_files[i] for i in range(0, num_images, num_images // 8)]
```

```
# Create a 2x4 subplot
fig,axes = plt.subplots(2,4,figsize=(20,11))

# Display each of the selected images
for ax, img_file in zip(axes.ravel(), selected_images):
    img_path =os.path.join(trian_image_path, img_file)
    image = Image.open(img_path)
    ax.imshow(image)
    ax.axis('off')

plt.suptitle('Sample Images from Training Dataset',fontsize=20)
plt.tight_layout()
plt.show()
```




```
import torch
print(torch.cuda.is_available())  # Should return True if GPU is enabled
```

```
False
```

```
results = model.train(
    data=yaml_file_path,
    epochs=20,  # Reduced to 20 epochs
    imgsz=640,
    device='cpu',  # Use 'cpu' if no GPU is available, or '0' for the first GPU
    patience=20,  # Optional: Adjust patience for early stopping
    batch=32,
    optimizer='auto',
    lr0=0.0001,
    lrf=0.1,
    dropout=0.1
)
```

```
20 epochs completed in 3.928 hours.
Optimizer stripped from runs/detect/train/weights/last.pt, 6.2MB
Optimizer stripped from runs/detect/train/weights/best.pt, 6.2MB

Validating runs/detect/train/weights/best.pt...
Ultralytics 8.3.32 🚀 Python-3.10.12 torch-2.5.1+cu121 CPU (Intel Xeon 2.20GHz)
Model summary (fused): 168 layers, 3,005,843 parameters, 0 gradients, 8.1 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 2/2 [00:31<00:00, 15.62s/it]
                   all         90        937      0.916      0.902      0.965      0.706
Speed: 9.0ms preprocess, 322.5ms inference, 0.0ms loss, 1.2ms postprocess per image
Results saved to runs/detect/train
```

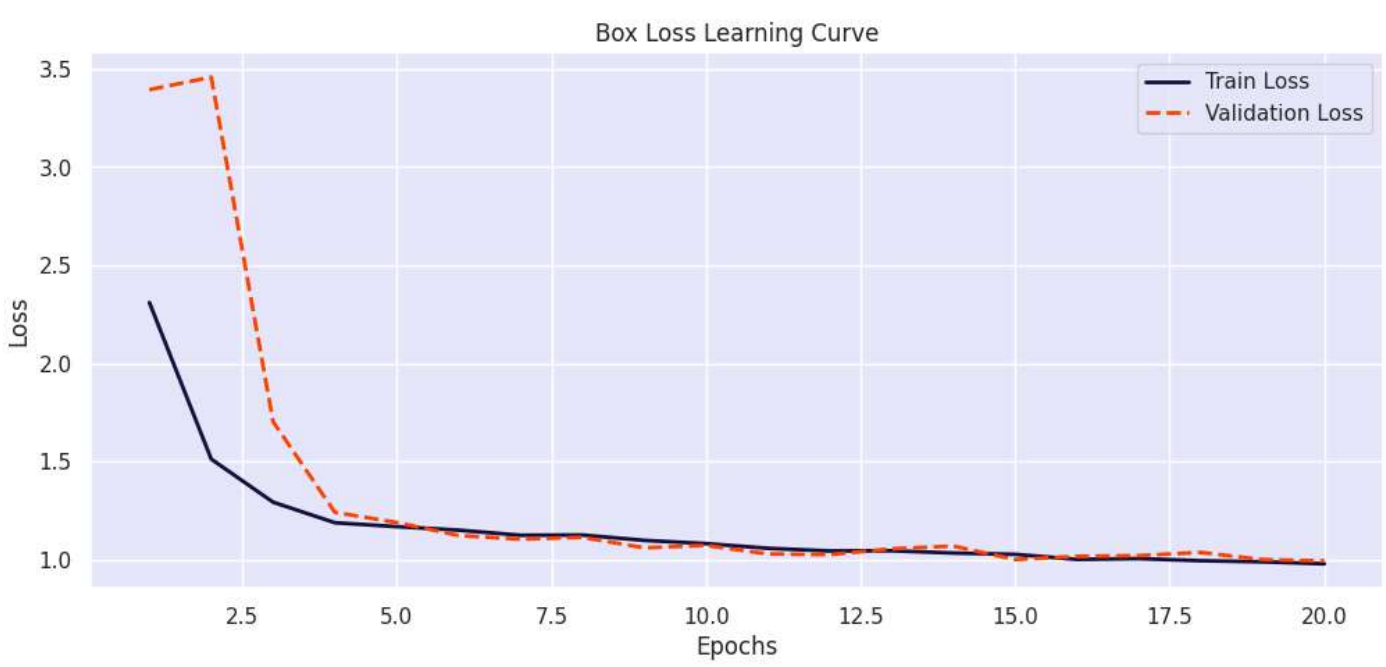
```
# Define the path to the Directory
post_training_files_path ='/content/runs/detect/train'

# list the files in the directory
!ls {post_training_files_path}
```

```
args.yaml                                       PR_curve.png        train_batch1.jpg
confusion_matrix_normalized.png                 R_curve.png         train_batch2.jpg
confusion_matrix.png                            results.csv         val_batch0_labels.jpg
events.out.tfevents.1731898754.6b1137575b3d.214.0  results.png      val_batch0_pred.jpg
F1_curve.png                                    train_batch0.jpg    val_batch1_labels.jpg
labels_correlogram.jpg                          train_batch170.jpg  val_batch1_pred.jpg
labels.jpg                                      train_batch171.jpg  weights
P_curve.png                                     train_batch172.jpg
```

```
# Construct the path to the best model weights file using os.path.join
best_model_path = os.path.join(post_training_files_path, 'weights/best.pt')

# Load the best model weights into the YOLO model
best_model = YOLO(best_model_path)

# Validate the best model using the validation set with default parameters
metrics = best_model.val(split='val')
```

```
Ultralytics 8.3.32 🚀 Python-3.10.12 torch-2.5.1+cu121 CPU (Intel Xeon 2.20GHz)
Model summary (fused): 168 layers, 3,005,843 parameters, 0 gradients, 8.1 GFLOPs
val: Scanning /content/extracted_folder/Vehicle_Detection_Image_Dataset/valid/labels.cache... 90 images, 0 backgrounds, 0 corrupt: 100%|██████████| 90/90 [00:00<?, ?it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 6/6 [00:29<00:00,  4.85s/it]
                   all         90        937      0.916      0.902      0.965      0.706
Speed: 3.5ms preprocess, 302.8ms inference, 0.0ms loss, 1.2ms postprocess per image
Results saved to runs/detect/val
```

```
# Define a function to plot learning curves for loss values
def plot_learning_curve(df, train_loss_col, val_loss_col, title):
    plt.figure(figsize=(12, 5))
    sns.lineplot(data=df, x='epoch', y=train_loss_col, label='Train Loss', color='#141140', linestyle='-', linewidth=2)
    sns.lineplot(data=df, x='epoch', y=val_loss_col, label='Validation Loss', color='orangered', linestyle='--', linewidth=2)
    plt.title(title)
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()


# Create the full file path for 'results.csv' using the directory path and file name
results_csv_path = os.path.join(post_training_files_path, 'results.csv')

# Load the CSV file from the constructed path into a pandas DataFrame
df = pd.read_csv(results_csv_path)

# Remove any leading whitespace from the column names
df.columns = df.columns.str.strip()

# Plot the learning curves for each loss
plot_learning_curve(df, 'train/box_loss', 'val/box_loss', 'Box Loss Learning Curve')
plot_learning_curve(df, 'train/cls_loss', 'val/cls_loss', 'Classification Loss Learning Curve')
plot_learning_curve(df, 'train/dfl_loss', 'val/dfl_loss', 'Distribution Focal Loss Learning Curve')
```




```
# Interface on validationset
valid_image_path = os.path.join(dataset_path, 'valid','images')

# list all jpg images in the directory
image_files = [file for file in os.listdir(valid_image_path) if file.endswith('.jpg')]

# select 9 images at equal intervals
num_images = len(image_files)
selected_images = [image_files[i] for i in range(0, num_images, num_images // 9)]

# Initailize the subplot
fig, axes = plt.subplots(3,3,figsize=(20,21))
plt.title('Validattion set inferences', fontsize=24)


# Perform inferences on each selected image and display it
for i, ax, in enumerate(axes.flatten()):
    image_path = os.path.join(valid_image_path, selected_images[i])
    results = best_model.predict(source = image_path, imgsz = 640, conf=0.5)
    annotated_image = results[0].plot(line_width=1)
    annotated_image_rgb =cv2.cvtColor(annotated_image, cv2.COLOR_BGR2RGB)
    ax.imshow(annotated_image_rgb)
    ax.axis('off')

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

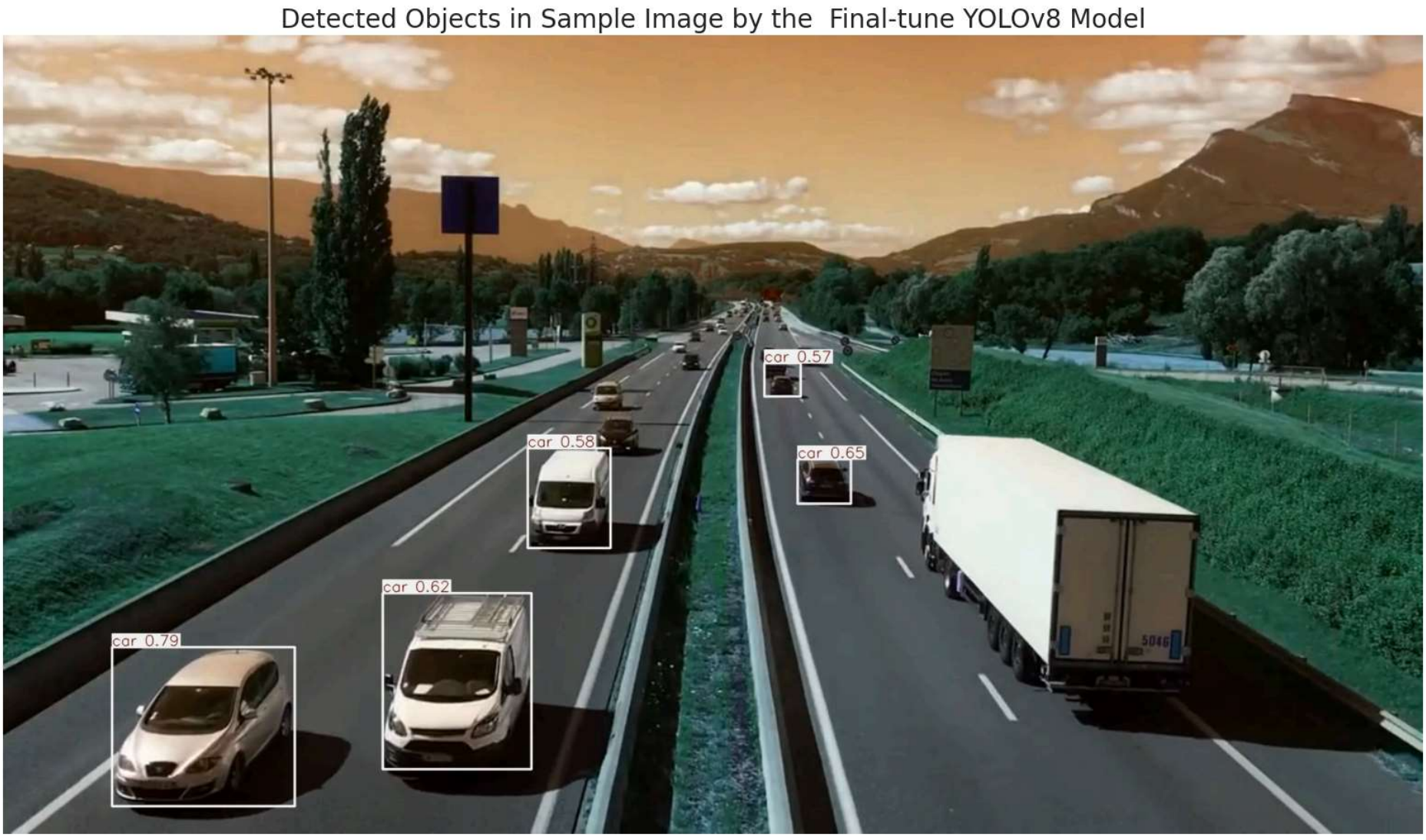
```
image 1/1 /content/extracted_folder/Vehicle_Detection_Image_Dataset/valid/images/16_mp4-1_jpg.rf.3493f4b7618e207609847857a20dbaff.jpg: 640x640 14 Vehicles, 263.6ms
Speed: 3.7ms preprocess, 263.6ms inference, 1.4ms postprocess per image at shape (1, 3, 640, 640)

image 1/1 /content/extracted_folder/Vehicle_Detection_Image_Dataset/valid/images/8_mp4-14_jpg.rf.e15e22de0316fce6d319640eae7a04c3.jpg: 640x640 15 Vehicles, 252.3ms
Speed: 4.5ms preprocess, 252.3ms inference, 1.1ms postprocess per image at shape (1, 3, 640, 640)

image 1/1 /content/extracted_folder/Vehicle_Detection_Image_Dataset/valid/images/10_mp4-9_jpg.rf.72a35ba2f76343afc7a101e720dbbb81.jpg: 640x640 22 Vehicles, 276.4ms
Speed: 3.4ms preprocess, 276.4ms inference, 1.1ms postprocess per image at shape (1, 3, 640, 640)

image 1/1 /content/extracted_folder/Vehicle_Detection_Image_Dataset/valid/images/3_mp4-17_jpg.rf.fdfd6a4ddf7c5f516a940b2452857d67.jpg: 640x640 1 Vehicle, 241.7ms
Speed: 3.4ms preprocess, 241.7ms inference, 1.0ms postprocess per image at shape (1, 3, 640, 640)

image 1/1 /content/extracted_folder/Vehicle_Detection_Image_Dataset/valid/images/test_mp4-13_jpg.rf.98cd77f75c4492f8f103aaf4ce2ca8f8.jpg: 640x640 14 Vehicles, 245.4ms
Speed: 3.2ms preprocess, 245.4ms inference, 1.1ms postprocess per image at shape (1, 3, 640, 640)

image 1/1 /content/extracted_folder/Vehicle_Detection_Image_Dataset/valid/images/test2_mp4-8_jpg.rf.c10dd559d44a868fd6f6877fd734815a.jpg: 640x640 21 Vehicles, 269.0ms
Speed: 3.3ms preprocess, 269.0ms inference, 1.7ms postprocess per image at shape (1, 3, 640, 640)

image 1/1 /content/extracted_folder/Vehicle_Detection_Image_Dataset/valid/images/10_mp4-0_jpg.rf.08b3bd34bbb73fb80c2d662c34474a98.jpg: 640x640 21 Vehicles, 247.3ms
Speed: 3.7ms preprocess, 247.3ms inference, 1.2ms postprocess per image at shape (1, 3, 640, 640)

image 1/1 /content/extracted_folder/Vehicle_Detection_Image_Dataset/valid/images/15_mp4-6_jpg.rf.f8ad77cac61af6ee1b551af09789bc24.jpg: 640x640 4 Vehicles, 244.6ms
Speed: 5.4ms preprocess, 244.6ms inference, 1.0ms postprocess per image at shape (1, 3, 640, 640)

image 1/1 /content/extracted_folder/Vehicle_Detection_Image_Dataset/valid/images/14_mp4-15_jpg.rf.9640ab027504af3733b76827756764eb.jpg: 640x640 2 Vehicles, 376.5ms
Speed: 3.9ms preprocess, 376.5ms inference, 1.7ms postprocess per image at shape (1, 3, 640, 640)
```

Validattion set inferences

```
sample_image_path = '/content/extracted_folder/Vehicle_Detection_Image_Dataset/sample_image.jpg'
# Perform inference on the provided image using best model
results = best_model.predict(source=sample_image_path, imgsz=640,conf=0.7)

# Annotate and convert image to numpy array
smaple_image = results[0].plot(line_width=2)

# convert the color of the image from BGR to RGB for correct color representation in matplotlib
sample_image = cv2.cvtColor(sample_image, cv2.COLOR_BGR2RGB)

# Display annotated image
plt.figure(figsize=(20,15))
plt.imshow(sample_image)
plt.title('Detected Objects in Sample Image by the  Final-tune YOLOv8 Model', fontsize=20)
```

```
plt.axis('off')
plt.show()
```

```
image 1/1 /content/extracted_folder/Vehicle_Detection_Image_Dataset/sample_image.jpg: 384x640 5 Vehicles, 245.6ms
Speed: 5.9ms preprocess, 245.6ms inference, 1.3ms postprocess per image at shape (1, 3, 384, 640)
```




```
# Define the path to the sample video in the dataset
dataset_video_path = '/content/extracted_folder/Vehicle_Detection_Image_Dataset/sample_video.mp4'

# Define the destination path in the working directory
video_path = '/content/runs/sample_video.mp4'

# Copy the video file from its original location in the dataset to the current working directory in Kaggle for further processing
shutil.copyfile(dataset_video_path, video_path)

# Initiate vehicle detection on the sample video using the best performing model and save the output
best_model.predict(source=video_path, save=True)
```

```
keypoints: None
masks: None
names: {0: 'Vehicle'}
obb: None
orig_img: array([[[254, 253, 242],
        [254, 253, 242],
        [254, 253, 242],
        ...,
        [210, 179, 162],
        [210, 179, 162],
        [210, 179, 162]],

       [[254, 253, 242],
        [254, 253, 242],
        [254, 253, 242],
        ...,
        [210, 179, 162],
        [210, 179, 162],
        [210, 179, 162]],

       [[255, 253, 243],
        [255, 252, 242],
        [255, 252, 242],
        ...,
        [210, 179, 162],
        [208, 177, 160],
        [208, 177, 160]],

       ...,

       [[ 94, 100, 104],
        [ 93,  99, 103],
        [ 89,  95,  99],
        ...,
        [ 88, 100, 105],
        [ 90, 102, 107],
        [ 94, 106, 111]],

       [[ 97, 103, 107],
        [ 96, 102, 106],
        [ 88,  94,  98],
        ...,
        [ 91, 103, 108],
        [ 95, 107, 112],
        [ 97, 109, 114]],

       [[ 97, 103, 107],
        [ 95, 101, 105],
        [ 88,  94,  98],
        ...,
        [ 89, 101, 106],
        [ 92, 104, 109],
        [ 98, 110, 115]]], dtype=uint8)
orig_shape: (720, 1280)
path: '/content/runs/sample_video.mp4'
probs: None
save_dir: 'runs/detect/predict'
speed: {'preprocess': 3.9031505584716797, 'inference': 150.7854461669922, 'postprocess': 0.9720325469970703}]
```

```
# Convert .avi video to .mp4
!ffmpeg -y -loglevel panic -i /content/runs/detect/predict/sample_video.avi processed_sample_video.mp4

# Embed and display the .mp4 video
from IPython.display import Video

Video("/content/processed_sample_video.mp4", embed=True, width=960)
```


```
import cv2
import numpy as np

# Define the threshold for considering traffic as heavy
heavy_traffic_threshold = 10

# Define the vertices for the quadrilaterals
vertices1 = np.array([(465, 350), (609, 350), (510, 630), (2, 630)], dtype=np.int32)
vertices2 = np.array([(678, 350), (815, 350), (1203, 630), (743, 630)], dtype=np.int32)

# Define the vertical range for the slice and lane threshold
x1, x2 = 325, 635
lane_threshold = 609

# Define the positions for the text annotations on the image
text_position_left_lane = (10, 50)
text_position_right_lane = (820, 50)
intensity_position_left_lane = (10, 100)
intensity_position_right_lane = (820, 100)

# Define font, scale, and colors for the annotations
font = cv2.FONT_HERSHEY_SIMPLEX
font_scale = 1
font_color = (255, 255, 255)    # White color for text
background_color = (0, 0, 255)  # Red background for text

# Open the video
cap = cv2.VideoCapture('/content/runs/sample_video.mp4')  # Ensure the input path is correct

# Check if the video file was opened successfully
if not cap.isOpened():
    raise IOError("Error opening video file. Check the file path!")

# Get the frame width and height
frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

# Define the codec and create VideoWriter object
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('traffic_density_analysis.avi', fourcc, 20.0, (frame_width, frame_height))

# Read until video is completed
while cap.isOpened():
    ret, frame = cap.read()
    if ret:
        # Create a copy of the original frame to modify
        detection_frame = frame.copy()

        # Black out the regions outside the specified vertical range
        detection_frame[:x1, :] = 0  # Black out from top to x1
        detection_frame[x2:, :] = 0  # Black out from x2 to the bottom of the frame

        # Perform inference on the modified frame
        results = best_model.predict(detection_frame, imgsz=640, conf=0.4)
        processed_frame = results[0].plot(line_width=1)

        # Restore the original top and bottom parts of the frame
        processed_frame[:x1, :] = frame[:x1, :].copy()
        processed_frame[x2:, :] = frame[x2:, :].copy()

        # Draw the quadrilaterals on the processed frame
        cv2.polylines(processed_frame, [vertices1], isClosed=True, color=(0, 255, 0), thickness=2)
        cv2.polylines(processed_frame, [vertices2], isClosed=True, color=(255, 0, 0), thickness=2)

        # Retrieve the bounding boxes from the results
        bounding_boxes = results[0].boxes

        # Initialize counters for vehicles in each lane
        vehicles_in_left_lane = 0
        vehicles_in_right_lane = 0

        # Loop through each bounding box to count vehicles in each lane
        for box in bounding_boxes.xyxy:
            # Check if the vehicle is in the left lane based on the x-coordinate of the bounding box
            if box[0] < lane_threshold:
                vehicles_in_left_lane += 1
            else:
                vehicles_in_right_lane += 1
```

```
            # Determine the traffic intensity for the left lane
            traffic_intensity_left = "Heavy" if vehicles_in_left_lane > heavy_traffic_threshold else "Smooth"
            # Determine the traffic intensity for the right lane
            traffic_intensity_right = "Heavy" if vehicles_in_right_lane > heavy_traffic_threshold else "Smooth"

            # Add a background rectangle for the left lane vehicle count
            cv2.rectangle(processed_frame, (text_position_left_lane[0]-10, text_position_left_lane[1] - 25),
                          (text_position_left_lane[0] + 460, text_position_left_lane[1] + 10), background_color, -1)

            # Add the vehicle count text on top of the rectangle for the left lane
            cv2.putText(processed_frame, f'Vehicles in Left Lane: {vehicles_in_left_lane}', text_position_left_lane,
                        font, font_scale, font_color, 2, cv2.LINE_AA)

            # Add a background rectangle for the left lane traffic intensity
            cv2.rectangle(processed_frame, (intensity_position_left_lane[0]-10, intensity_position_left_lane[1] - 25),
                          (intensity_position_left_lane[0] + 460, intensity_position_left_lane[1] + 10), background_color, -1)

            # Add the traffic intensity text on top of the rectangle for the left lane
            cv2.putText(processed_frame, f'Traffic Intensity: {traffic_intensity_left}', intensity_position_left_lane,
                        font, font_scale, font_color, 2, cv2.LINE_AA)

            # Add a background rectangle for the right lane vehicle count
            cv2.rectangle(processed_frame, (text_position_right_lane[0]-10, text_position_right_lane[1] - 25),
                          (text_position_right_lane[0] + 460, text_position_right_lane[1] + 10), background_color, -1)

            # Add the vehicle count text on top of the rectangle for the right lane
            cv2.putText(processed_frame, f'Vehicles in Right Lane: {vehicles_in_right_lane}', text_position_right_lane,
                        font, font_scale, font_color, 2, cv2.LINE_AA)

            # Add a background rectangle for the right lane traffic intensity
            cv2.rectangle(processed_frame, (intensity_position_right_lane[0]-10, intensity_position_right_lane[1] - 25),
                          (intensity_position_right_lane[0] + 460, intensity_position_right_lane[1] + 10), background_color, -1)

            # Add the traffic intensity text on top of the rectangle for the right lane
            cv2.putText(processed_frame, f'Traffic Intensity: {traffic_intensity_right}', intensity_position_right_lane,
                        font, font_scale, font_color, 2, cv2.LINE_AA)

            # Write the processed frame to the output video
            out.write(processed_frame)
        else:
            break

    # Release the video capture and video write objects
    cap.release()
    out.release()
```

```
Speed: 3.8ms preprocess, 154.2ms inference, 1.0ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 2 Vehicles, 152.4ms
Speed: 3.6ms preprocess, 152.4ms inference, 1.0ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 2 Vehicles, 158.3ms
Speed: 4.2ms preprocess, 158.3ms inference, 1.0ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 2 Vehicles, 187.9ms
Speed: 5.0ms preprocess, 187.9ms inference, 1.0ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 2 Vehicles, 155.8ms
Speed: 5.3ms preprocess, 155.8ms inference, 1.0ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 2 Vehicles, 155.2ms
Speed: 3.9ms preprocess, 155.2ms inference, 1.0ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 2 Vehicles, 159.0ms
Speed: 6.2ms preprocess, 159.0ms inference, 1.0ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 2 Vehicles, 158.7ms
Speed: 3.6ms preprocess, 158.7ms inference, 1.6ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 2 Vehicles, 183.1ms
Speed: 4.6ms preprocess, 183.1ms inference, 1.0ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 2 Vehicles, 162.3ms
Speed: 5.2ms preprocess, 162.3ms inference, 1.0ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 2 Vehicles, 161.9ms
Speed: 3.6ms preprocess, 161.9ms inference, 1.0ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 2 Vehicles, 185.8ms
Speed: 3.5ms preprocess, 185.8ms inference, 1.4ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 2 Vehicles, 157.5ms
Speed: 5.2ms preprocess, 157.5ms inference, 1.4ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 2 Vehicles, 170.2ms
Speed: 6.2ms preprocess, 170.2ms inference, 1.0ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 2 Vehicles, 152.6ms
Speed: 6.1ms preprocess, 152.6ms inference, 1.1ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 2 Vehicles, 160.6ms
Speed: 4.2ms preprocess, 160.6ms inference, 1.0ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 2 Vehicles, 177.4ms
Speed: 3.9ms preprocess, 177.4ms inference, 1.1ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 2 Vehicles, 155.4ms
Speed: 4.7ms preprocess, 155.4ms inference, 1.0ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 2 Vehicles, 175.2ms
Speed: 5.1ms preprocess, 175.2ms inference, 1.0ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 2 Vehicles, 158.2ms
Speed: 4.7ms preprocess, 158.2ms inference, 1.0ms postprocess per image at shape (1, 3, 384, 640)
```

```
# Convert the .avi video generated by our traffic density estimation app to .mp4 format for compatibility with notebook display
!ffmpeg -y -loglevel panic -i /content/traffic_density_analysis.avi traffic_density_analysis.mp4

# Embed and display the processed sample video within the notebook
Video("traffic_density_analysis.mp4", embed=True, width=960)
```

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

0.04 / 0.30

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.