```
!pip install ultralytics opencv-python cvzone numpy
```

```
Collecting ultralytics
  Downloading ultralytics-8.3.158-py3-none-any.whl.metadata (37 kB)
Requirement already satisfied: opencv-python in /usr/local/lib/python3.11/dist-packages (4.11.0.86)
Collecting cvzone
  Downloading cvzone-1.6.1.tar.gz (25 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (2.0.2)
Requirement already satisfied: matplotlib>=3.3.0 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (
Requirement already satisfied: pillow>=7.1.2 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (11.2
Requirement already satisfied: pyyaml>=5.3.1 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (6.0.
Requirement already satisfied: requests>=2.23.0 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (2
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (1.15.
Requirement already satisfied: torch>=1.8.0 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (2.6.0
Requirement already satisfied: torchvision>=0.9.0 in /usr/local/lib/python3.11/dist-packages (from ultralytics)
Requirement already satisfied: tqdm>=4.64.0 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (4.67.
Requirement already satisfied: psutil in /usr/local/lib/python3.11/dist-packages (from ultralytics) (5.9.5)
Requirement already satisfied: py-cpuinfo in /usr/local/lib/python3.11/dist-packages (from ultralytics) (9.0.0)
Requirement already satisfied: pandas>=1.1.4 in /usr/local/lib/python3.11/dist-packages (from ultralytics) (2.2.
Collecting ultralytics-thop>=2.0.0 (from ultralytics)
  Downloading ultralytics_thop-2.0.14-py3-none-any.whl.metadata (9.4 kB)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.3
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.3.0->
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.3.
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=3.3
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib>
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.1.4->ultr
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.1.4->ul
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from request
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests>=2.23.0->u
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests>=2.2
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests>=2.2
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.0->ultralyti
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/python3.11/dist-packages (from torch>
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.0->ultralyti
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.0->ultralytics
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from torch>=1.8.0->ultralytics
Collecting nvidia-cuda-nvrtc-cu12==12.4.127 (from torch>=1.8.0->ultralytics)
  Downloading nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cuda-runtime-cu12==12.4.127 (from torch>=1.8.0->ultralytics)
  Downloading nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cuda-cupti-cu12==12.4.127 (from torch>=1.8.0->ultralytics)
  Downloading nvidia_cuda_cupti_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cudnn-cu12==9.1.0.70 (from torch>=1.8.0->ultralytics)
  Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cublas-cu12==12.4.5.8 (from torch>=1.8.0->ultralytics)
  Downloading nvidia_cublas_cu12-12.4.5.8-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cufft-cu12==11.2.1.3 (from torch>=1.8.0->ultralytics)
  Downloading nvidia_cufft_cu12-11.2.1.3-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-curand-cu12==10.3.5.147 (from torch>=1.8.0->ultralytics)
  Downloading nvidia_curand_cu12-10.3.5.147-py3-none-manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cusolver-cu12==11.6.1.9 (from torch>=1.8.0->ultralytics)
  Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cusparse-cu12==12.3.1.170 (from torch>=1.8.0->ultralytics)
  Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.whl.metadata (1.6 kB)
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/lib/python3.11/dist-packages (from to
```

```
# Create a directory for weights if it doesn't exist
!mkdir -p Yolo-Weights

# Download YOLOv8l weights
!wget -P Yolo-Weights https://github.com/ultralytics/assets/releases/download/v8.2.0/yolov8l.pt
```

```
--2025-06-21 18:43:38--  https://github.com/ultralytics/assets/releases/download/v8.2.0/yolov8l.pt
Resolving github.com (github.com)... 20.27.177.113
```

```
Connecting to github.com (github.com)|20.27.177.113|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asset-2e65be/521807533/10638fb7-d214-403
--2025-06-21 18:43:38--  https://objects.githubusercontent.com/github-production-release-asset-2e65be/521807533/10
Resolving objects.githubusercontent.com (objects.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.1
Connecting to objects.githubusercontent.com (objects.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 87792836 (84M) [application/octet-stream]
Saving to: 'Yolo-Weights/yolov8l.pt'

yolov8l.pt          100%[===================>]  83.73M  34.2MB/s    in 2.4s

2025-06-21 18:43:42 (34.2 MB/s) - 'Yolo-Weights/yolov8l.pt' saved [87792836/87792836]
```

```
# Example of creating a Videos directory in Colab if needed
!mkdir -p Videos
# Then manually upload cars.mp4 into this 'Videos' folder


from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_
```

Start coding or generate with AI.

```
!wget -O sort.py https://raw.githubusercontent.com/abewley/sort/master/sort.py
```

```
--2025-06-21 18:46:04--  https://raw.githubusercontent.com/abewley/sort/master/sort.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.110.133, 185.199.109.133, 185.199.111.1
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.110.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 11739 (11K) [text/plain]
Saving to: 'sort.py'

sort.py             100%[===================>]  11.46K  --.-KB/s    in 0.003s

2025-06-21 18:46:05 (3.73 MB/s) - 'sort.py' saved [11739/11739]
```

```
import numpy as np
from ultralytics import YOLO
import cv2
import cvzone
import math
from sort import * # Make sure sort.py is in the same directory as your notebook

# --- Mount Google Drive (Recommended) ---
# Run this in a separate cell FIRST if your files are on Drive
from google.colab import drive
drive.mount('/content/drive')

# --- Define Paths ---
# Adjust these paths based on where your files are.
# Example for Google Drive:
VIDEO_PATH = "/content/drive/MyDrive/YoloCarCounter/cars.mp4"
MASK_PATH = "/content/drive/MyDrive/YoloCarCounter/mask.png"
GRAPHICS_PATH = "/content/drive/MyDrive/YoloCarCounter/graphics.png"
YOLO_WEIGHTS_PATH = "Yolo-Weights/yolov8l.pt" # Assuming you downloaded this to /content/Yolo-Weights/


# --- Verify File Existence (Crucial Debugging Step) ---
```

```
import os

print(f"Checking video file: {VIDEO_PATH}")
if not os.path.exists(VIDEO_PATH):
    print(f"ERROR: Video file not found at {VIDEO_PATH}")
    print("Please ensure the video file is uploaded/mounted correctly and the path is accurate.")
    # Exit or handle gracefully if video is essential
    exit() # Or raise an error
else:
    print("Video file found.")

print(f"Checking mask file: {MASK_PATH}")
if not os.path.exists(MASK_PATH):
    print(f"ERROR: Mask file not found at {MASK_PATH}")
    # Handle missing mask, maybe use a blank mask or exit
    mask = np.zeros((1080, 1920, 3), dtype=np.uint8) # Create a black mask as fallback
    print("Using a black fallback mask.")
else:
    print("Mask file found.")

print(f"Checking graphics file: {GRAPHICS_PATH}")
if not os.path.exists(GRAPHICS_PATH):
    print(f"ERROR: Graphics file not found at {GRAPHICS_PATH}")
    # Handle missing graphics, maybe use a blank image or exit
    imgGraphics = np.zeros((100, 100, 4), dtype=np.uint8) # Create a transparent fallback image
    print("Using a blank fallback graphics image.")
else:
    print("Graphics file found.")

print(f"Checking YOLO weights: {YOLO_WEIGHTS_PATH}")
if not os.path.exists(YOLO_WEIGHTS_PATH):
    print(f"ERROR: YOLO weights not found at {YOLO_WEIGHTS_PATH}")
    print("Please ensure the weights are downloaded correctly.")
    exit() # Or raise an error
else:
    print("YOLO weights found.")


# --- Initialize Video Capture ---
cap = cv2.VideoCapture(VIDEO_PATH)

# Check if video was opened successfully
if not cap.isOpened():
    print(f"ERROR: Could not open video file: {VIDEO_PATH}")
    print("Possible reasons:")
    print("1. File path is incorrect or file does not exist (check previous checks).")
    print("2. Video codec not supported by OpenCV in Colab environment.")
    print("3. Video file is corrupted.")
    exit()
else:
    print("Video opened successfully.")
    # Get video properties to confirm it's reading something
    frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = int(cap.get(cv2.CAP_PROP_FPS))
    frame_count_total = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    print(f"Video Resolution: {frame_width}x{frame_height}, FPS: {fps}, Total Frames: {frame_count_total}")
    if frame_count_total <= 0:
        print("WARNING: Video reports 0 or negative total frames. It might be empty or unreadable.")


# --- Load YOLO Model ---
model = YOLO(YOLO_WEIGHTS_PATH)

classNames = ["person", "bicycle", "car", "motorbike", "aeroplane", "bus", "train", "truck", "boat",
```

```
                "traffic light", "fire hydrant", "stop sign", "parking meter", "bench", "bird", "cat",
                "dog", "horse", "sheep", "cow", "elephant", "bear", "zebra", "giraffe", "backpack", "umbrella",
                "handbag", "tie", "suitcase", "frisbee", "skis", "snowboard", "sports ball", "kite", "baseball bat",
                "baseball glove", "skateboard", "surfboard", "tennis racket", "bottle", "wine glass", "cup",
                "fork", "knife", "spoon", "bowl", "banana", "apple", "sandwich", "orange", "broccoli",
                "carrot", "hot dog", "pizza", "donut", "cake", "chair", "sofa", "pottedplant", "bed",
                "diningtable", "toilet", "tvmonitor", "laptop", "mouse", "remote", "keyboard", "cell phone",
                "microwave", "oven", "toaster", "sink", "refrigerator", "book", "clock", "vase", "scissors",
                "teddy bear", "hair drier", "toothbrush"
                ]

# Load mask and graphics only if they were found, otherwise use fallbacks
if os.path.exists(MASK_PATH):
    mask = cv2.imread(MASK_PATH)
else:
    # If mask was not found, a black mask was already created as fallback
    pass

if os.path.exists(GRAPHICS_PATH):
    imgGraphics = cv2.imread(GRAPHICS_PATH, cv2.IMREAD_UNCHANGED)
else:
    # If graphics not found, a blank image was already created as fallback
    pass


# Tracking
tracker = Sort(max_age=20, min_hits=3, iou_threshold=0.3)

limits = [400, 297, 673, 297]
totalCount = []

# For displaying images in Colab
from IPython.display import display, Image
import io
from PIL import Image as PIL_Image

frame_count = 0
display_interval = 30 # Display every 30 frames (adjust as needed)

print("Starting video processing loop...")
while True:
    success, img = cap.read()
    if not success:
        print("DEBUG: cap.read() returned False. End of video or error reading frame.")
        break

    # Ensure img has 3 channels for bitwise_and if mask is grayscale
    if len(img.shape) == 2: # Grayscale image
        img = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)

    # Resize mask if it doesn't match the image dimensions
    if mask.shape[:2] != img.shape[:2]:
        print(f"WARNING: Mask resolution {mask.shape[:2]} does not match video frame resolution {img.shape[:2]}. Resiz
        mask = cv2.resize(mask, (img.shape[1], img.shape[0]))

    imgRegion = cv2.bitwise_and(img, mask)

    # Ensure imgGraphics has alpha channel for overlay, if not, add one
    if imgGraphics.shape[2] == 3: # If it's a 3-channel (BGR) image
        imgGraphics = cv2.cvtColor(imgGraphics, cv2.COLOR_BGR2BGRA) # Convert to BGRA

    # Ensure imgGraphics dimensions are appropriate or resize
    if imgGraphics.shape[0] > img.shape[0] or imgGraphics.shape[1] > img.shape[1]:
        print(f"WARNING: Graphics image resolution {imgGraphics.shape[:2]} is larger than video frame {img.shape[:2]}.
        imgGraphics = cv2.resize(imgGraphics, (img.shape[1], img.shape[0]))
```

```
        img = cvzone.overlayPNG(img, imgGraphics, (0, 0)) # overlay on the original img, not imgRegion
        results = model(imgRegion, stream=True)

        detections = np.empty((0, 5))

        for r in results:
            boxes = r.boxes
            for box in boxes:
                # Bounding Box
                x1, y1, x2, y2 = box.xyxy[0]
                x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
                w, h = x2 - x1, y2 - y1

                # Confidence
                conf = math.ceil((box.conf[0] * 100)) / 100
                # Class Name
                cls = int(box.cls[0])
                currentClass = classNames[cls]

                if currentClass in ["car", "truck", "bus", "motorbike"] and conf > 0.3:
                    currentArray = np.array([x1, y1, x2, y2, conf])
                    detections = np.vstack((detections, currentArray))

        resultsTracker = tracker.update(detections)

        cv2.line(img, (limits[0], limits[1]), (limits[2], limits[3]), (0, 0, 255), 5)
        for result in resultsTracker:
            x1, y1, x2, y2, id = result
            x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
            # print(result) # Uncomment for detailed tracking output
            w, h = x2 - x1, y2 - y1
            cvzone.cornerRect(img, (x1, y1, w, h), l=9, rt=2, colorR=(255, 0, 255))
            cvzone.putTextRect(img, f' {int(id)}', (max(0, x1), max(35, y1)),
                               scale=2, thickness=3, offset=10)

            cx, cy = x1 + w // 2, y1 + h // 2
            cv2.circle(img, (cx, cy), 5, (255, 0, 255), cv2.FILLED)

            if limits[0] < cx < limits[2] and limits[1] - 15 < cy < limits[1] + 15:
                if totalCount.count(id) == 0:
                    totalCount.append(id)
                    cv2.line(img, (limits[0], limits[1]), (limits[2], limits[3]), (0, 255, 0), 5)

        cv2.putText(img,str(len(totalCount)),(255,100),cv2.FONT_HERSHEY_PLAIN,5,(50,50,255),8)

        # Display image in Colab (replace cv2.imshow)
        img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        pil_img = PIL_Image.fromarray(img_rgb)
        if frame_count % display_interval == 0:
            buffered = io.BytesIO()
            pil_img.save(buffered, format="JPEG")
            display(Image(buffered.getvalue()))

        frame_count += 1

    cap.release()
    print(f"\nProcessing finished. Total cars counted: {len(totalCount)}")
```

```
End of video or failed to read frame.
Total cars counted: 0
```

```
!pip install filterpy
```

```
Collecting filterpy
  Downloading filterpy-1.4.5.zip (177 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 178.0/178.0 kB 3.9 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from filterpy) (2.0.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from filterpy) (1.15.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (from filterpy) (3.10.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->filte
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib->filterpy)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->filt
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->filt
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->filter
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib->filterpy) (1
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->filte
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib->f
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->mat
Building wheels for collected packages: filterpy
  Building wheel for filterpy (setup.py) ... done
  Created wheel for filterpy: filename=filterpy-1.4.5-py3-none-any.whl size=110460 sha256=3b282ff18aa7cc845603ec73
  Stored in directory: /root/.cache/pip/wheels/12/dc/3c/e12983eac132d00f82a20c6cbe7b42ce6e96190ef8fa2d15e1
Successfully built filterpy
Installing collected packages: filterpy
Successfully installed filterpy-1.4.5
```

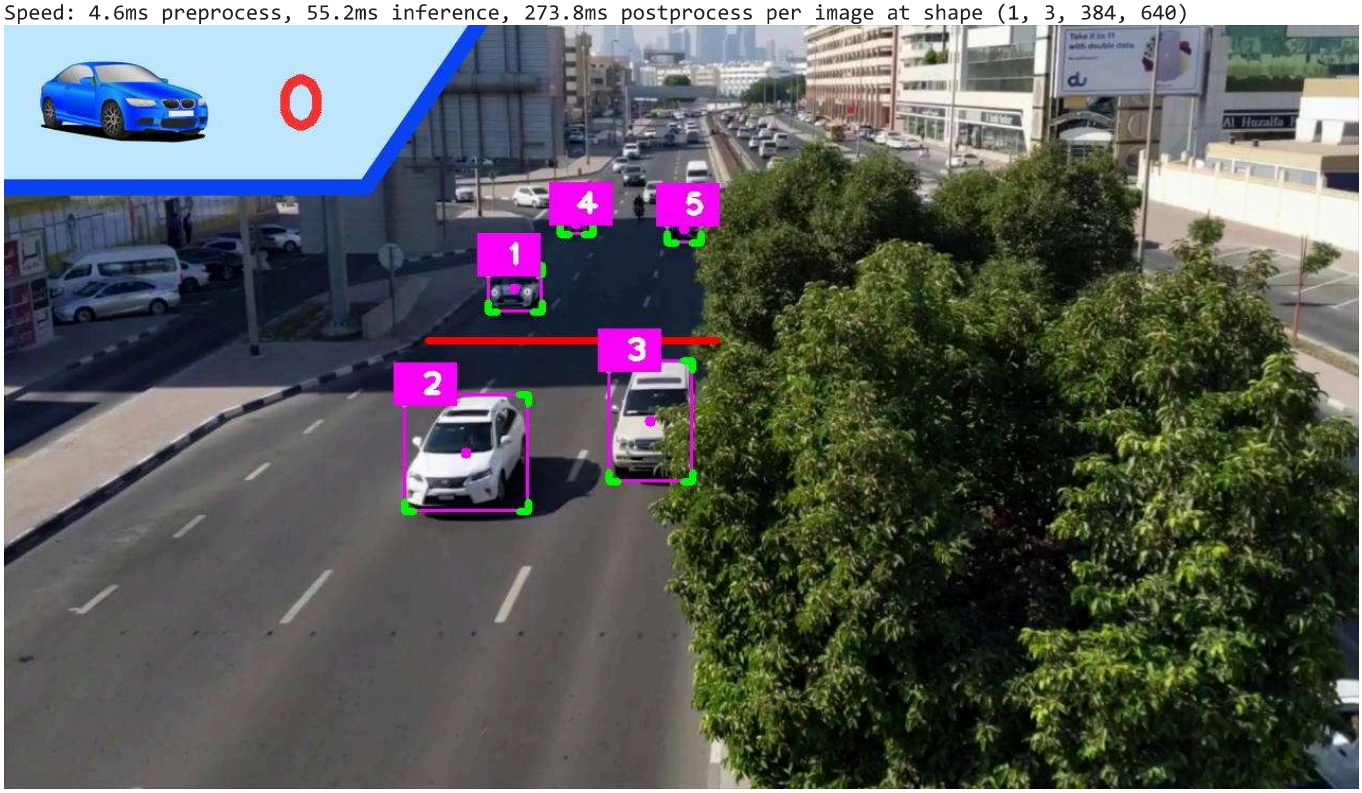
```
import numpy as np
from ultralytics import YOLO
import cv2
import cvzone
import math
from sort import * # Make sure sort.py is in the same directory as your notebook

# --- Mount Google Drive (Only if you still need it for other files or future use) ---
# If your files are all in /content/, you DON'T need to run this.
# from google.colab import drive
# drive.mount('/content/drive')

# --- Define Paths ---
# IMPORTANT: Paths based on your input: /content/cars.mp4, /content/mask.png, /content/graphics.png, and /content/Yolo
VIDEO_PATH = "/content/cars.mp4"
MASK_PATH = "/content/mask.png"
GRAPHICS_PATH = "/content/graphics.png"
YOLO_WEIGHTS_PATH = "/content/Yolo-Weights/yolov8l.pt" # Explicitly set to /content/Yolo-Weights/


# --- Verify File Existence (Crucial Debugging Step) ---
import os

print(f"Checking video file: {VIDEO_PATH}")
if not os.path.exists(VIDEO_PATH):
    print(f"ERROR: Video file not found at {VIDEO_PATH}")
    print("Please ensure the video file is uploaded/mounted correctly and the path is accurate.")
    # Exit or handle gracefully if video is essential
    exit() # Or raise an error
else:
    print("Video file found.")

print(f"Checking mask file: {MASK_PATH}")
if not os.path.exists(MASK_PATH):
    print(f"ERROR: Mask file not found at {MASK_PATH}")
    # Handle missing mask, maybe use a blank mask or exit
    mask = np.zeros((1080, 1920, 3), dtype=np.uint8) # Create a black mask as fallback
    print("Using a black fallback mask.")
else:
    print("Mask file found.")
```

```
print(f"Checking graphics file: {GRAPHICS_PATH}")
if not os.path.exists(GRAPHICS_PATH):
    print(f"ERROR: Graphics file not found at {GRAPHICS_PATH}")
    # Handle missing graphics, maybe use a blank image or exit
    imgGraphics = np.zeros((100, 100, 4), dtype=np.uint8) # Create a transparent fallback image
    print("Using a blank fallback graphics image.")
else:
    print("Graphics file found.")

print(f"Checking YOLO weights: {YOLO_WEIGHTS_PATH}")
if not os.path.exists(YOLO_WEIGHTS_PATH):
    print(f"ERROR: YOLO weights not found at {YOLO_WEIGHTS_PATH}")
    print("Please ensure the weights are downloaded correctly. Run the wget command again if unsure:")
    print("!mkdir -p /content/Yolo-Weights && !wget -P /content/Yolo-Weights https://github.com/ultralytics/assets/rel
    exit() # Or raise an error
else:
    print("YOLO weights found.")


# --- Initialize Video Capture ---
cap = cv2.VideoCapture(VIDEO_PATH)

# Check if video was opened successfully
if not cap.isOpened():
    print(f"ERROR: Could not open video file: {VIDEO_PATH}")
    print("Possible reasons:")
    print("1. File path is incorrect or file does not exist (check previous checks).")
    print("2. Video codec not supported by OpenCV in Colab environment.")
    print("3. Video file is corrupted.")
    exit()
else:
    print("Video opened successfully.")
    # Get video properties to confirm it's reading something
    frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = int(cap.get(cv2.CAP_PROP_FPS))
    frame_count_total = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    print(f"Video Resolution: {frame_width}x{frame_height}, FPS: {fps}, Total Frames: {frame_count_total}")
    if frame_count_total <= 0:
        print("WARNING: Video reports 0 or negative total frames. It might be empty or unreadable.")


# --- Load YOLO Model ---
model = YOLO(YOLO_WEIGHTS_PATH)

classNames = ["person", "bicycle", "car", "motorbike", "aeroplane", "bus", "train", "truck", "boat",
              "traffic light", "fire hydrant", "stop sign", "parking meter", "bench", "bird", "cat",
              "dog", "horse", "sheep", "cow", "elephant", "bear", "zebra", "giraffe", "backpack", "umbrella",
              "handbag", "tie", "suitcase", "frisbee", "skis", "snowboard", "sports ball", "kite", "baseball bat",
              "baseball glove", "skateboard", "surfboard", "tennis racket", "bottle", "wine glass", "cup",
              "fork", "knife", "spoon", "bowl", "banana", "apple", "sandwich", "orange", "broccoli",
              "carrot", "hot dog", "pizza", "donut", "cake", "chair", "sofa", "pottedplant", "bed",
              "diningtable", "toilet", "tvmonitor", "laptop", "mouse", "remote", "keyboard", "cell phone",
              "microwave", "oven", "toaster", "sink", "refrigerator", "book", "clock", "vase", "scissors",
              "teddy bear", "hair drier", "toothbrush"
              ]

# Load mask and graphics only if they were found, otherwise use fallbacks
# These lines now use the updated MASK_PATH and GRAPHICS_PATH variables
if os.path.exists(MASK_PATH):
    mask = cv2.imread(MASK_PATH)
else:
    # If mask was not found, a black mask was already created as fallback
    pass
```

```
if os.path.exists(GRAPHICS_PATH):
    imgGraphics = cv2.imread(GRAPHICS_PATH, cv2.IMREAD_UNCHANGED)
else:
    # If graphics not found, a blank image was already created as fallback
    pass


# Tracking
tracker = Sort(max_age=20, min_hits=3, iou_threshold=0.3)

limits = [400, 297, 673, 297]
totalCount = []

# For displaying images in Colab
from IPython.display import display, Image
import io
from PIL import Image as PIL_Image

frame_count = 0
display_interval = 30 # Display every 30 frames (adjust as needed)

print("Starting video processing loop...")
while True:
    success, img = cap.read()
    if not success:
        print("DEBUG: cap.read() returned False. End of video or error reading frame.")
        break

    # Ensure img has 3 channels for bitwise_and if mask is grayscale
    if len(img.shape) == 2: # Grayscale image
        img = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)

    # Resize mask if it doesn't match the image dimensions
    if mask.shape[:2] != img.shape[:2]:
        print(f"WARNING: Mask resolution {mask.shape[:2]} does not match video frame resolution {img.shape[:2]}. Resiz
        mask = cv2.resize(mask, (img.shape[1], img.shape[0]))

    imgRegion = cv2.bitwise_and(img, mask)

    # Ensure imgGraphics has alpha channel for overlay, if not, add one
    if imgGraphics is not None and imgGraphics.shape[2] == 3: # If it's a 3-channel (BGR) image
        imgGraphics = cv2.cvtColor(imgGraphics, cv2.COLOR_BGR2BGRA) # Convert to BGRA

    # Ensure imgGraphics dimensions are appropriate or resize
    # Make sure imgGraphics is not None before trying to check shape or resize
    if imgGraphics is not None and (imgGraphics.shape[0] > img.shape[0] or imgGraphics.shape[1] > img.shape[1]):
        print(f"WARNING: Graphics image resolution {imgGraphics.shape[:2]} is larger than video frame {img.shape[:2]}.
        imgGraphics = cv2.resize(imgGraphics, (img.shape[1], img.shape[0]))


    # Only attempt overlay if imgGraphics is not None
    if imgGraphics is not None:
        img = cvzone.overlayPNG(img, imgGraphics, (0, 0)) # overlay on the original img, not imgRegion
    else:
        print("WARNING: imgGraphics is None, skipping overlay.")


    results = model(imgRegion, stream=True)

    detections = np.empty((0, 5))

    for r in results:
        boxes = r.boxes
        for box in boxes:
            # Bounding Box
```

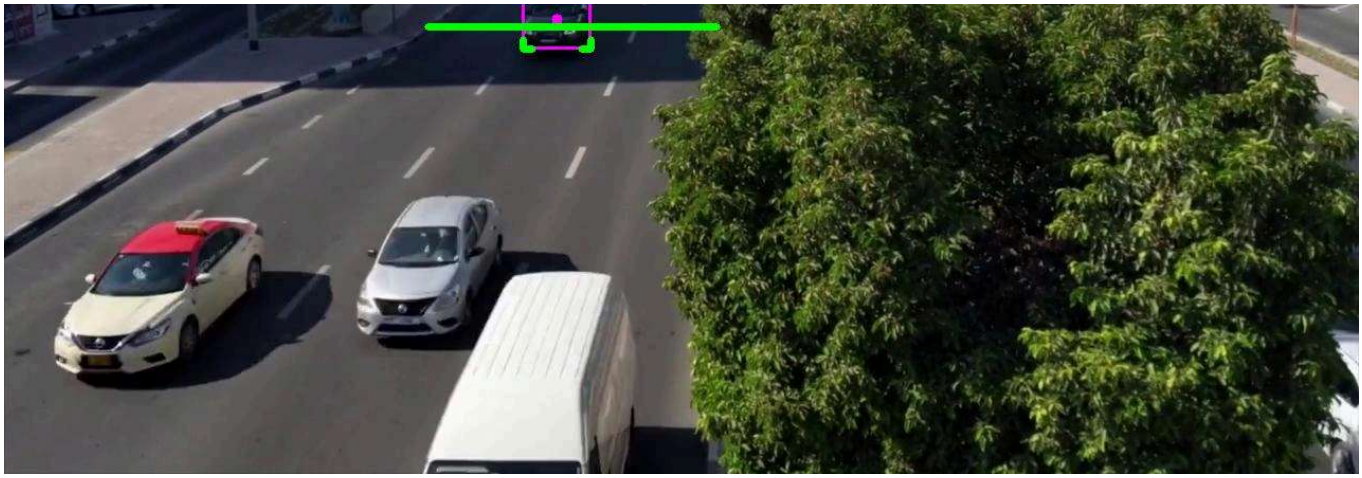
```
            x1, y1, x2, y2 = box.xyxy[0]
            x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
            w, h = x2 - x1, y2 - y1

            # Confidence
            conf = math.ceil((box.conf[0] * 100)) / 100
            # Class Name
            cls = int(box.cls[0])
            currentClass = classNames[cls]

            if currentClass in ["car", "truck", "bus", "motorbike"] and conf > 0.3:
                currentArray = np.array([x1, y1, x2, y2, conf])
                detections = np.vstack((detections, currentArray))

    resultsTracker = tracker.update(detections)

    cv2.line(img, (limits[0], limits[1]), (limits[2], limits[3]), (0, 0, 255), 5)
    for result in resultsTracker:
        x1, y1, x2, y2, id = result
        x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
        # print(result) # Uncomment for detailed tracking output
        w, h = x2 - x1, y2 - y1
        cvzone.cornerRect(img, (x1, y1, w, h), l=9, rt=2, colorR=(255, 0, 255))
        cvzone.putTextRect(img, f' {int(id)}', (max(0, x1), max(35, y1)),
                           scale=2, thickness=3, offset=10)

        cx, cy = x1 + w // 2, y1 + h // 2
        cv2.circle(img, (cx, cy), 5, (255, 0, 255), cv2.FILLED)

        if limits[0] < cx < limits[2] and limits[1] - 15 < cy < limits[1] + 15:
            if totalCount.count(id) == 0:
                totalCount.append(id)
                cv2.line(img, (limits[0], limits[1]), (limits[2], limits[3]), (0, 255, 0), 5)

    cv2.putText(img,str(len(totalCount)),(255,100),cv2.FONT_HERSHEY_PLAIN,5,(50,50,255),8)

    # Display image in Colab (replace cv2.imshow)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    pil_img = PIL_Image.fromarray(img_rgb)
    if frame_count % display_interval == 0:
        buffered = io.BytesIO()
        pil_img.save(buffered, format="JPEG")
        display(Image(buffered.getvalue()))

    frame_count += 1

cap.release()
print(f"\nProcessing finished. Total cars counted: {len(totalCount)}")
```

```
Checking video file: /content/cars.mp4
Video file found.
Checking mask file: /content/mask.png
Mask file found.
Checking graphics file: /content/graphics.png
Graphics file found.
Checking YOLO weights: /content/Yolo-Weights/yolov8l.pt
YOLO weights found.
Video opened successfully.
Video Resolution: 1280x720, FPS: 30, Total Frames: 368
Starting video processing loop...
WARNING: Mask resolution (480, 950) does not match video frame resolution (720, 1280). Resizing mask.

0: 384x640 5 cars, 55.2ms
Speed: 4.6ms preprocess, 55.2ms inference, 273.8ms postprocess per image at shape (1, 3, 384, 640)
```



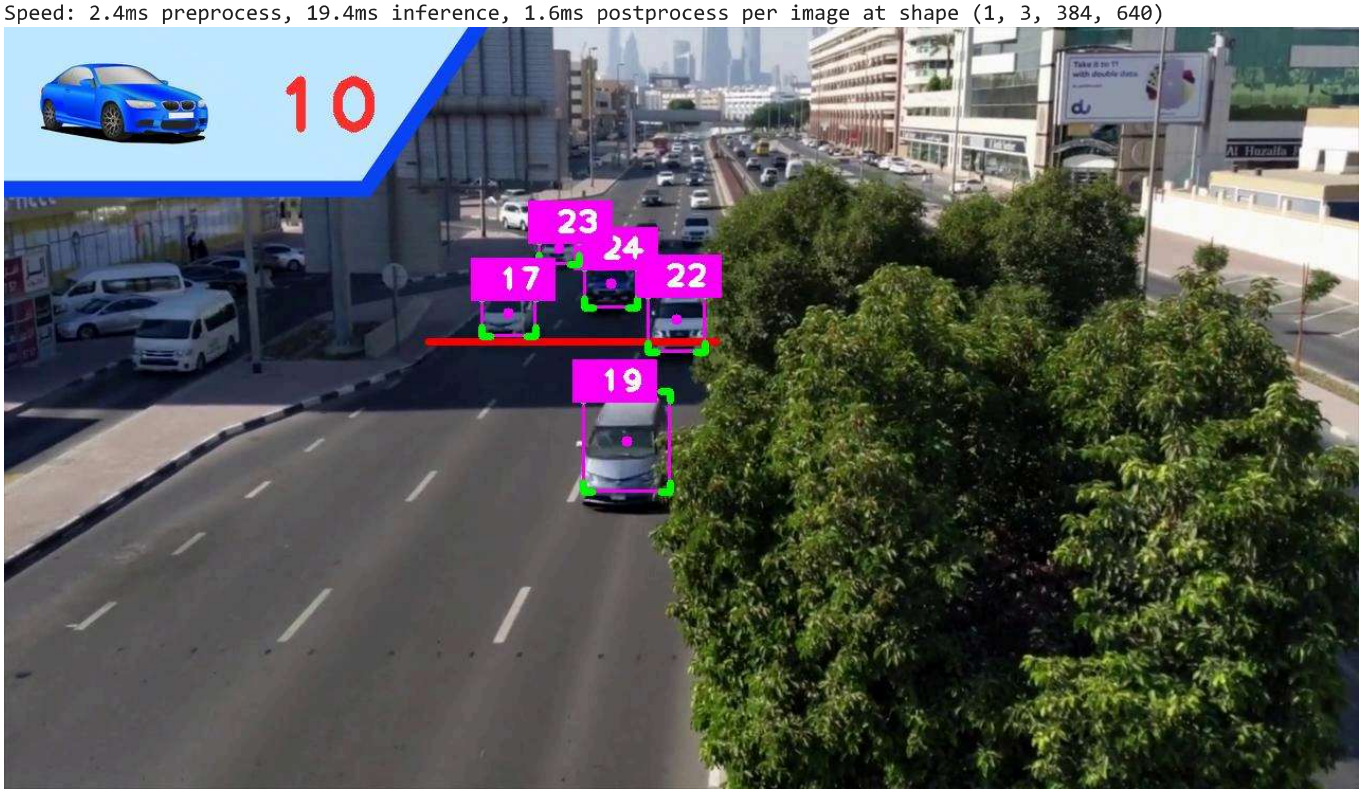

```
0: 384x640 5 cars, 40.2ms
Speed: 2.7ms preprocess, 40.2ms inference, 2.1ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 5 cars, 40.1ms
Speed: 2.6ms preprocess, 40.1ms inference, 1.7ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 1 person, 5 cars, 40.1ms
Speed: 2.4ms preprocess, 40.1ms inference, 1.7ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 1 person, 5 cars, 40.1ms
Speed: 4.2ms preprocess, 40.1ms inference, 1.9ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 5 cars, 28.0ms
Speed: 2.3ms preprocess, 28.0ms inference, 2.8ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 4 cars, 28.0ms
Speed: 2.8ms preprocess, 28.0ms inference, 1.9ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 5 cars, 28.0ms
Speed: 2.2ms preprocess, 28.0ms inference, 1.8ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 5 cars, 28.0ms
Speed: 2.4ms preprocess, 28.0ms inference, 1.7ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 5 cars, 21.6ms
Speed: 2.4ms preprocess, 21.6ms inference, 1.6ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 5 cars, 21.8ms
```

```
Speed: 2.3ms preprocess, 21.8ms inference, 1.9ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 5 cars, 21.2ms
Speed: 2.3ms preprocess, 21.2ms inference, 1.7ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 6 cars, 21.2ms
Speed: 2.7ms preprocess, 21.2ms inference, 1.6ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 6 cars, 21.0ms
Speed: 5.1ms preprocess, 21.0ms inference, 1.6ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 6 cars, 1 motorcycle, 19.0ms
Speed: 2.1ms preprocess, 19.0ms inference, 1.6ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 6 cars, 1 motorcycle, 19.2ms
Speed: 5.6ms preprocess, 19.2ms inference, 1.7ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 6 cars, 1 motorcycle, 19.1ms
Speed: 5.5ms preprocess, 19.1ms inference, 1.8ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 6 cars, 1 motorcycle, 19.2ms
Speed: 2.8ms preprocess, 19.2ms inference, 1.7ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 5 cars, 1 motorcycle, 1 truck, 19.0ms
Speed: 2.8ms preprocess, 19.0ms inference, 1.9ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 1 person, 3 cars, 1 motorcycle, 1 truck, 18.8ms
Speed: 4.8ms preprocess, 18.8ms inference, 1.9ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 1 person, 5 cars, 1 motorcycle, 1 truck, 18.6ms
Speed: 2.5ms preprocess, 18.6ms inference, 1.6ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 1 person, 4 cars, 1 motorcycle, 1 truck, 19.0ms
Speed: 6.3ms preprocess, 19.0ms inference, 1.7ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 1 person, 4 cars, 1 motorcycle, 1 truck, 18.6ms
Speed: 2.5ms preprocess, 18.6ms inference, 1.8ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 1 person, 4 cars, 1 motorcycle, 1 truck, 18.7ms
Speed: 2.1ms preprocess, 18.7ms inference, 1.7ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 1 person, 4 cars, 1 motorcycle, 1 truck, 18.7ms
Speed: 2.4ms preprocess, 18.7ms inference, 1.7ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 4 cars, 1 truck, 19.8ms
Speed: 2.5ms preprocess, 19.8ms inference, 1.7ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 4 cars, 2 trucks, 18.6ms
Speed: 2.4ms preprocess, 18.6ms inference, 1.5ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 4 cars, 1 bus, 1 truck, 18.8ms
Speed: 2.5ms preprocess, 18.8ms inference, 1.5ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 5 cars, 1 truck, 19.1ms
Speed: 2.5ms preprocess, 19.1ms inference, 1.7ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 5 cars, 19.2ms
Speed: 2.4ms preprocess, 19.2ms inference, 2.0ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 4 cars, 23.0ms
Speed: 2.5ms preprocess, 23.0ms inference, 2.7ms postprocess per image at shape (1, 3, 384, 640)
```



```
0: 384x640 4 cars, 18.3ms
Speed: 4.6ms preprocess, 18.3ms inference, 2.3ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 4 cars, 18.8ms
Speed: 2.8ms preprocess, 18.8ms inference, 1.7ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 4 cars, 21.9ms
Speed: 2.6ms preprocess, 21.9ms inference, 1.9ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 4 cars, 18.7ms
Speed: 2.4ms preprocess, 18.7ms inference, 1.7ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 4 cars, 19.0ms
Speed: 2.4ms preprocess, 19.0ms inference, 2.1ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 4 cars, 23.5ms
Speed: 4.0ms preprocess, 23.5ms inference, 1.6ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 5 cars, 18.3ms
Speed: 4.5ms preprocess, 18.3ms inference, 1.8ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 5 cars, 18.4ms
Speed: 2.6ms preprocess, 18.4ms inference, 1.6ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 5 cars, 19.8ms
Speed: 2.6ms preprocess, 19.8ms inference, 2.0ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 4 cars, 18.9ms
Speed: 3.7ms preprocess, 18.9ms inference, 1.9ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 4 cars, 18.7ms
Speed: 2.5ms preprocess, 18.7ms inference, 1.7ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 4 cars, 19.0ms
Speed: 2.7ms preprocess, 19.0ms inference, 1.7ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 5 cars, 19.0ms
Speed: 2.4ms preprocess, 19.0ms inference, 1.5ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 5 cars, 19.1ms
Speed: 6.2ms preprocess, 19.1ms inference, 1.9ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 5 cars, 18.8ms
Speed: 2.6ms preprocess, 18.8ms inference, 2.0ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 6 cars, 18.9ms
Speed: 2.6ms preprocess, 18.9ms inference, 1.7ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 6 cars, 18.4ms
Speed: 2.4ms preprocess, 18.4ms inference, 1.5ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 6 cars, 18.9ms
Speed: 2.5ms preprocess, 18.9ms inference, 1.6ms postprocess per image at shape (1, 3, 384, 640)
```

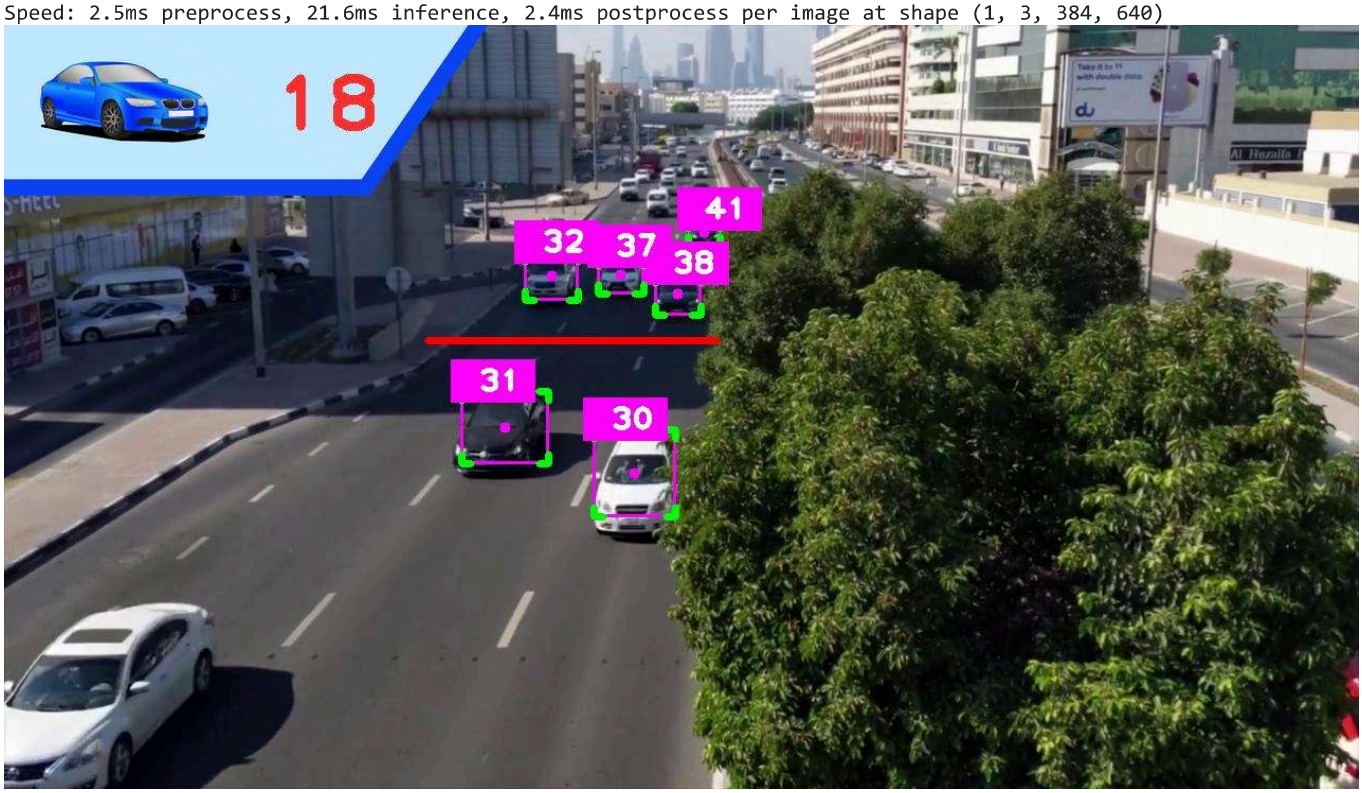
```
0: 384x640 6 cars, 18.8ms
Speed: 2.5ms preprocess, 18.8ms inference, 1.5ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 6 cars, 18.7ms
Speed: 2.2ms preprocess, 18.7ms inference, 1.6ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 6 cars, 18.7ms
Speed: 2.8ms preprocess, 18.7ms inference, 1.8ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 6 cars, 18.8ms
Speed: 2.6ms preprocess, 18.8ms inference, 1.7ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 5 cars, 19.2ms
Speed: 2.7ms preprocess, 19.2ms inference, 1.7ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 6 cars, 19.0ms
Speed: 2.3ms preprocess, 19.0ms inference, 1.6ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 5 cars, 23.8ms
Speed: 2.5ms preprocess, 23.8ms inference, 2.8ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 6 cars, 18.3ms
Speed: 2.4ms preprocess, 18.3ms inference, 1.8ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 7 cars, 18.3ms
Speed: 5.9ms preprocess, 18.3ms inference, 1.6ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 7 cars, 18.8ms
Speed: 2.4ms preprocess, 18.8ms inference, 1.7ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 8 cars, 18.4ms
Speed: 2.1ms preprocess, 18.4ms inference, 1.9ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 8 cars, 19.4ms
Speed: 2.4ms preprocess, 19.4ms inference, 1.6ms postprocess per image at shape (1, 3, 384, 640)
```




```
0: 384x640 7 cars, 18.2ms
Speed: 2.3ms preprocess, 18.2ms inference, 3.3ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 8 cars, 1 truck, 18.2ms
Speed: 2.2ms preprocess, 18.2ms inference, 1.6ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 7 cars, 18.2ms
Speed: 2.5ms preprocess, 18.2ms inference, 1.7ms postprocess per image at shape (1, 3, 384, 640)
```

```
0: 384x640 7 cars, 1 truck, 19.1ms
Speed: 2.5ms preprocess, 19.1ms inference, 1.8ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 7 cars, 1 truck, 18.4ms
Speed: 2.2ms preprocess, 18.4ms inference, 1.5ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 6 cars, 1 truck, 18.2ms
Speed: 2.6ms preprocess, 18.2ms inference, 1.5ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 6 cars, 1 truck, 18.2ms
Speed: 2.5ms preprocess, 18.2ms inference, 1.6ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 9 cars, 1 truck, 18.5ms
Speed: 2.4ms preprocess, 18.5ms inference, 1.8ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 9 cars, 1 truck, 19.1ms
Speed: 2.9ms preprocess, 19.1ms inference, 1.8ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 8 cars, 1 truck, 18.5ms
Speed: 2.7ms preprocess, 18.5ms inference, 2.5ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 8 cars, 1 truck, 18.2ms
Speed: 2.4ms preprocess, 18.2ms inference, 1.5ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 7 cars, 1 truck, 18.9ms
Speed: 2.3ms preprocess, 18.9ms inference, 1.6ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 7 cars, 19.1ms
Speed: 2.4ms preprocess, 19.1ms inference, 1.7ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 7 cars, 18.3ms
Speed: 2.4ms preprocess, 18.3ms inference, 1.6ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 7 cars, 1 truck, 18.5ms
Speed: 2.3ms preprocess, 18.5ms inference, 1.6ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 7 cars, 19.2ms
Speed: 2.4ms preprocess, 19.2ms inference, 2.1ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 8 cars, 19.0ms
Speed: 2.5ms preprocess, 19.0ms inference, 1.7ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 8 cars, 19.9ms
Speed: 2.4ms preprocess, 19.9ms inference, 2.4ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 8 cars, 19.0ms
Speed: 3.0ms preprocess, 19.0ms inference, 2.5ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 8 cars, 19.1ms
Speed: 2.4ms preprocess, 19.1ms inference, 1.8ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 9 cars, 18.5ms
Speed: 2.5ms preprocess, 18.5ms inference, 2.4ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 8 cars, 18.4ms
Speed: 2.2ms preprocess, 18.4ms inference, 1.5ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 6 cars, 18.4ms
Speed: 2.4ms preprocess, 18.4ms inference, 1.7ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 6 cars, 1 truck, 19.4ms
Speed: 2.4ms preprocess, 19.4ms inference, 2.1ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 7 cars, 1 truck, 19.9ms
Speed: 2.5ms preprocess, 19.9ms inference, 1.6ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 7 cars, 18.6ms
Speed: 2.5ms preprocess, 18.6ms inference, 1.7ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 8 cars, 19.0ms
Speed: 2.4ms preprocess, 19.0ms inference, 1.7ms postprocess per image at shape (1, 3, 384, 640)
```

```
0: 384x640 7 cars, 19.0ms
Speed: 2.3ms preprocess, 19.0ms inference, 2.5ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 6 cars, 19.0ms
Speed: 2.3ms preprocess, 19.0ms inference, 2.7ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 7 cars, 21.6ms
Speed: 2.5ms preprocess, 21.6ms inference, 2.4ms postprocess per image at shape (1, 3, 384, 640)
```




```
0: 384x640 8 cars, 21.4ms
Speed: 2.3ms preprocess, 21.4ms inference, 2.3ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 8 cars, 21.8ms
Speed: 2.4ms preprocess, 21.8ms inference, 4.4ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 7 cars, 20.4ms
Speed: 2.5ms preprocess, 20.4ms inference, 2.3ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 8 cars, 18.7ms
Speed: 2.4ms preprocess, 18.7ms inference, 2.7ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 8 cars, 19.0ms
Speed: 2.3ms preprocess, 19.0ms inference, 2.1ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 8 cars, 19.5ms
Speed: 4.0ms preprocess, 19.5ms inference, 2.1ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 8 cars, 18.7ms
Speed: 2.4ms preprocess, 18.7ms inference, 3.5ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 8 cars, 24.1ms
Speed: 2.4ms preprocess, 24.1ms inference, 5.2ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 8 cars, 20.8ms
Speed: 2.4ms preprocess, 20.8ms inference, 2.5ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 9 cars, 20.2ms
Speed: 2.5ms preprocess, 20.2ms inference, 2.5ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 9 cars, 20.3ms
Speed: 2.2ms preprocess, 20.3ms inference, 2.1ms postprocess per image at shape (1, 3, 384, 640)

0: 384x640 7 cars, 1 truck, 19.4ms
```