National College of Ireland

Configuration Manual

MSc Research Project
MSc Cloud Computing

Jatin Nikhil Dilliraj Saraswathy
Student ID: 23127228

School of Computing
National College of Ireland

Supervisor: Mr. Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Jatin Nikhil Dilliraj Saraswathy
Student ID:	23127228
Programme:	MSc Cloud Computing **Year:** 2024-25
Module:	MSc Research Project
Lecturer:	Mr. Vikas Sahni
Submission Due Date:	03/01/25
Project Title:	Traffic signal optimisation and control using deep learning framework deployed over cloud for a connected Traffic Management
Word Count:	1016 **Page Count:** 8

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Jatin Nikhil
Date:	03/01/25

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Jatin Nikhil Dilliraj Saraswathy
Student ID: 23127228

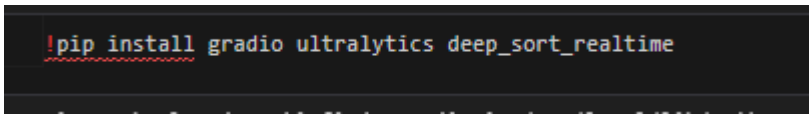
1 System Requirements

RAM: 16-32 GB SSD
OS: Windows 11 / Ubuntu 20.0.x +
Environment: AWS/Google Colab

2 Steps to Execute:

Step 1: Enter the URL
Step 2: Input a video or traffic image
Step 3: Observe the Gradio output
Step 4: Check the vehicles count

3 Vehicle Count Config File



The code snippet installs three Python libraries using the `pip` command:

1. `gradio`: A library to build web-based user interfaces for machine learning models.
2. `ultralytics`: A package related to the YOLO (You Only Look Once) object detection framework.
3. `deep_sort_realtime`: A library for real-time multi-object tracking using the Deep SORT algorithm.

```python
import cv2
import pandas as pd
import gradio as gr
from ultralytics import YOLO
import numpy as np

# Initialize the YOLO model
model = YOLO('yolo11n.pt')

# Define vehicle classes to track
vehicle_classes = ['car', 'motorcycle', 'bus', 'truck']

# Function to process video and count vehicles crossing the red line
def process_video_with_counting(video_path):
    """
    Process a video to detect and count vehicles crossing a red line.
    """
    cap = cv2.VideoCapture(video_path)

    # Initialize variables
    count = 0
    counter_down = set()

    # Output video setup
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    out = cv2.VideoWriter('output_annotated_video.mp4', fourcc, cap.get(cv2.CAP_PROP_FPS),
                          (int(cap.get(cv2.CAP_PROP_FRAME_WIDTH)), int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))))

    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break
        count += 1

        # Perform object detection
        results = model(frame)
        if len(results) > 0:
            a = results[0].boxes.data
            a = a.detach().cpu().numpy()
            px = pd.DataFrame(a).astype("float")

            list = []
            for index, row in px.iterrows():
                x1 = int(row[0])
                y1 = int(row[1])
                x2 = int(row[2])
                y2 = int(row[3])
                d = int(row[5])
                c = model.names[d]
                if c in vehicle_classes:  # Filtering only vehicle types
                    list.append([x1, y1, x2, y2])
```

This code snippet processes a video to detect and count vehicles crossing a red line.

1. **Imports and Initialization:**
   - Libraries like `cv2`, `pandas`, and `gradio` are imported for video processing, data handling, and building interfaces.
   - A YOLO model is initialized with a pre-trained weight file (`yolov8n.pt`).
   - Vehicle classes of interest are defined as `['car', 'motorcycle', 'bus', 'truck']`.
2. **`process_video_with_counting` function:**
   - This function takes a video file path as input and analyzes each frame to detect vehicles.

- Sets up the video writer to save annotated output (`output_annotated_video.mp4`).
- Reads each frame and processes it using the YOLO model for object detection.

3. **Object Detection:**
   - The YOLO model returns bounding box predictions for objects in the frame.
   - Filters out predictions to keep only those matching the defined vehicle classes.
   - Coordinates of detected vehicles are stored in a list for further processing.

```
            # Process detected vehicles in the frame
            for x1, y1, x2, y2 in list:
                # Get the center of the bounding box
                cx = (x1 + x2) // 2
                cy = (y1 + y2) // 2

                # Check if the vehicle crosses the red line (Y=308)
                y = 308
                offset = 7
                if y < (cy + offset) and y > (cy - offset):
                    # Count this vehicle as crossing the red line
                    counter_down.add(f"{cx}_{cy}")  # Using a unique center point as ID for counting

        # Draw the red line
        red_color = (0, 0, 255)
        cv2.line(frame, (282, 308), (1004, 308), red_color, 3)
        cv2.putText(frame, 'Red line', (280, 308), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1, cv2.LINE_AA)

        # Display count of vehicles that crossed
        downwards = len(counter_down)
        cv2.putText(frame, f'Going down - {downwards}', (60, 40), cv2.FONT_HERSHEY_SIMPLEX, 0.5, red_color, 1, cv2.LINE_AA)

        # Write the frame to output video
        out.write(frame)

    cap.release()
    out.release()

    # Return annotated video and vehicle count
    return 'output_annotated_video.mp4', f'Vehicles going down: {len(counter_down)}'

def count_and_detect_vehicles(image):
    """
    Detect and count vehicles in an image.
    """
    # Perform object detection
    results = model(image)

    # Count vehicles
    vehicle_counts = {}

    # Create a copy of the image for annotation
    annotated_image = image.copy()

    # Draw bounding boxes and count vehicles
    for result in results:
        boxes = result.boxes

        for box in boxes:
            # Get class details
            cls = model.names[int(box.cls[0])]

            # Check if it's a vehicle class
            if cls in vehicle_classes:
                # Increment count
                vehicle_counts[cls] = vehicle_counts.get(cls, 0) + 1
```

his part of the code builds on the vehicle detection logic to count vehicles crossing a specific line (red line) in a video and also includes functionality to process images for vehicle detection. Here's a concise explanation:

1. Processing Detected Vehicles in Frames:

- For each detected vehicle's bounding box, the center point `(cx, cy)` is calculated.
- If the center point crosses the defined red line (at `y=308` with an offset), the vehicle is counted and logged in `counter_down` to ensure each vehicle is counted only once.

2. Annotating the Frame:

- The red line is drawn on the video frame to visualize the detection threshold.
- Displays the count of vehicles that have crossed the line (`Going down`).
- The annotated frame is written to the output video.

3. Finalizing Video Output:

- Releases video resources and returns the annotated video file and the total vehicle count.

4. `count_and_detect_vehicles` Function:

- Detects vehicles in a single image using the YOLO model.
- Counts the detected vehicles and groups them by type (e.g., car, bus).

- Annotates the image with bounding boxes and the count of detected vehicles.

```
                # Get bounding box coordinates
                x1, y1, x2, y2 = box.xyxy[0]
                x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)

                # Draw rectangle
                color = (np.random.randint(0, 255),
                         np.random.randint(0, 255),
                         np.random.randint(0, 255))
                cv2.rectangle(annotated_image, (x1, y1), (x2, y2), color, 2)

                # Put class label
                label = f"{cls}"
                cv2.putText(annotated_image, label, (x1, y1 - 10),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.9, color, 2)

    # Prepare vehicle count text
    count_text = "\n".join([f"{vehicle}: {count}" for vehicle, count in vehicle_counts.items()])

    return annotated_image, count_text

def interface():
    """
    Create a Gradio interface for vehicle counting on images and videos.
    """
    def process_input(input_type, input_value):
        if input_type == "image":
            # Read the uploaded image
            image = cv2.imread(input_value)
            if image is None:
                return None, None, "Error: Unable to read the uploaded image."
            annotated_image, vehicle_count = count_and_detect_vehicles(image)
            return annotated_image, None, vehicle_count
        elif input_type == "video":
            # Handle video input and ensure proper format
            if isinstance(input_value, str):
                annotated_video, vehicle_count = process_video_with_counting(input_value)
                return None, annotated_video, vehicle_count
            else:
                return None, None, "Error: Unsupported video format."
        else:
            return None, None, "Error: Invalid input type."

    demo = gr.Interface(
        fn=process_input,
        inputs=[
            gr.Radio(["image", "video"], label="Input Type", value="image"),
            gr.File(type="filepath", label="Upload Image/Video")
        ],
        outputs=[
            gr.Image(label="Annotated Image (for Images Only)"),
            gr.Video(label="Annotated Video (for Videos Only)"),
            gr.Textbox(label="Vehicle Count")
        ],
        title="Vehicle Counting and Crossing Detection",
        description="Detect and count vehicles in images or videos. Vehicles crossing a red line in videos will be tracked."
    )
    return demo

# Launch the Gradio interface
if __name__ == "__main__":
    app = interface()
    app.launch(debug=True)
```

This part of the code integrates the detection and counting logic into a Gradio web interface for ease of use. Here's a concise breakdown:

1. Bounding Box and Annotations:

- For each detected object, a bounding box is drawn with a randomly chosen color.
- The class label (e.g., "car", "bus") is displayed above the bounding box.
- The vehicle count by type is prepared as a summary text (`count_text`).

2. Gradio Interface (`interface` function):

- Defines a `process_input` function to handle both image and video inputs:
  - **Image:** Reads and processes the uploaded image for vehicle detection, returns the annotated image and vehicle count.
  - **Video:** Processes the uploaded video for vehicle detection and tracking, returns the annotated video and count.
  - Handles unsupported file types with an error message.
- Sets up a Gradio interface with:
  - Inputs: A radio button to select between "image" or "video" and a file upload component.
  - Outputs: Displays the annotated image/video and a textbox with the vehicle count.

3. Launching the Gradio App:

- The `interface` function is called, and the Gradio app is launched with debug mode enabled.

References

Ntakolia, C., Lyridis, D.V. A n − D ant colony optimization with fuzzy logic for air traffic flow management. Oper Res Int J 22, 5035–5053 (2022).

H R, Deekshetha & Madhav, A. & Tyagi, Amit. (2022). Traffic Prediction Using Machine Learning. 10.1007/978-981-16-9605-3_68.

Zarindast, Atousa & Poddar, Subhadipto & Sharma, Anuj. (2022). A Data-Driven Method for Congestion Identification and Classification. Journal of Transportation Engineering, Part A: Systems. 148. 10.1061/JTEPBS.0000654.

Kranti Kumar, Manoj Kumar, Pritikana Das,Traffic congestion forecasting using multilayered deep neural network,Transportation Letters,2023,ISSN 1942-7867,

Elsagheer, Samir & AlShalfan, Khaled. (2021). Intelligent Traffic Management System Based on the Internet of Vehicles (IoV). Journal of Advanced Transportation. 2021. 1-23. 10.1155/2021/4037533.