SOURCE CODE:

```
import cv2
import torch
import time
import numpy as np
import tkinter as tk
from PIL import Image, ImageTk
from threading import Thread

# Load the YOLOv5 model from Torch Hub
model = torch.hub.load('ultralytics/yolov5', 'yolov5s')

# Define the labels for vehicles and autorickshaw
VEHICLE_LABELS = ['car', 'motorbike', 'bus', 'truck', 'bicycle', 'autorickshaw']

# Define road areas (these values may need to be adjusted)
road_areas = [
    (0, 0, 640, 480),  # Road 1
    (640, 0, 1280, 480),  # Road 2
    (0, 480, 640, 960),  # Road 3
    (640, 480, 1280, 960)  # Road 4
]

# Duration to check each road (in seconds)
check_duration = 20
green_light_duration = 5  # Seconds each light stays green
```

```
class TrafficDensityApp:
    def __init__(self, root):  # Corrected from _init_ to __init__
        self.root = root
        self.root.title("Traffic Density Detection")

        # Set the window size to 1024x720 to accommodate both camera and traffic lights
        self.root.geometry("1024x720")
        self.root.configure(bg="#f0f0f0")

        # Set button style with white font color
        button_style = {"font": ("Helvetica", 12, "bold"), "bg": "#4CAF50", "fg": "white", "relief": "raised"}

        # Frame for Start/Stop buttons
        self.control_frame = tk.Frame(root, bg="#f0f0f0")
        self.control_frame.pack(side="left", padx=10, pady=10)

        self.start_button = tk.Button(self.control_frame, text="Start Processing", command=self.start_detection, **button_style)
        self.start_button.grid(row=0, column=0, pady=5)

        self.stop_button = tk.Button(self.control_frame, text="Stop Detection", command=self.stop_detection, state=tk.DISABLED, **button_style)
        self.stop_button.grid(row=1, column=0, pady=5)

        # Frame for Manual Processing buttons
        self.manual_frame = tk.Frame(root, bg="#f0f0f0")
        self.manual_frame.pack(side="right", padx=10, pady=10)
```

```
        self.manual_start_button = tk.Button(self.manual_frame, text="Start Manual Processing", command=self.start_manual_processing, **button_style)
        self.manual_start_button.grid(row=0, column=0, pady=5)

        self.manual_stop_button = tk.Button(self.manual_frame, text="Stop Manual Processing", command=self.stop_manual_processing, state=tk.DISABLED, font=("Helvetica", 12, "bold"), bg="red", fg="white", relief="raised")
        self.manual_stop_button.grid(row=1, column=0, pady=5)

        # Display label for results
        self.result_label = tk.Label(root, text="Result: ", font=("Helvetica", 12), bg="#f0f0f0", fg="#333")
        self.result_label.pack(pady=5)

        # Canvas to show the camera feed with adjusted height and width
        self.canvas = tk.Canvas(root, width=640, height=480, bg="#d0d0d0")  # Increased camera window size
        self.canvas.pack(side="left", padx=10, pady=10)

        # Canvas for traffic lights with more compact layout
        self.traffic_light_canvas = tk.Canvas(root, width=250, height=400, bg="#ffffff", highlightthickness=0)
        self.traffic_light_canvas.pack(side="right", padx=10, pady=10)

        # Drawing traffic light indicators with road labels and adjusted positioning
        self.traffic_lights = []
        for i in range(4):
            y_position = 50 + i * 90
```

```
            self.traffic_light_canvas.create_text(125, y_position - 15, text=f"Road {i + 1}", font=("Helvetica", 10, "bold"), fill="#333")
            red_light = self.traffic_light_canvas.create_oval(100, y_position, 150, y_position + 40, fill="red")
            green_light = self.traffic_light_canvas.create_oval(160, y_position, 210, y_position + 40, fill="gray")
            self.traffic_lights.append((red_light, green_light))

        self.running = False
        self.cap = None
        self.manual_running = False

    def update_traffic_lights(self, active_index):
        for i, (red_light, green_light) in enumerate(self.traffic_lights):
            if i == active_index:
                self.traffic_light_canvas.itemconfig(red_light, fill="gray")
                self.traffic_light_canvas.itemconfig(green_light, fill="green")
            else:
                self.traffic_light_canvas.itemconfig(red_light, fill="red")
                self.traffic_light_canvas.itemconfig(green_light, fill="gray")
        self.root.update_idletasks()

    def reset_traffic_lights(self):
        for red_light, green_light in self.traffic_lights:
            self.traffic_light_canvas.itemconfig(red_light, fill="red")
            self.traffic_light_canvas.itemconfig(green_light, fill="gray")
        self.root.update_idletasks()
```

```
def start_detection(self):
    self.start_button.config(state=tk.DISABLED)
    self.manual_start_button.config(state=tk.DISABLED)
    self.stop_button.config(state=tk.NORMAL)
    self.running = True

    camera_thread = Thread(target=self.detect_traffic_density)
    camera_thread.start()

def stop_detection(self):
    self.running = False
    if self.cap:
        self.cap.release()
    self.start_button.config(state=tk.NORMAL)
    self.manual_start_button.config(state=tk.NORMAL)
    self.stop_button.config(state=tk.DISABLED)
    self.reset_traffic_lights()

def start_manual_processing(self):
    if not self.manual_running:
        self.manual_running = True
        self.manual_start_button.config(state=tk.DISABLED)
        self.manual_stop_button.config(state=tk.NORMAL)
        manual_thread = Thread(target=self.manual_processing)
        manual_thread.start()

def stop_manual_processing(self):
```

```
    self.manual_running = False
    self.manual_start_button.config(state=tk.NORMAL)
    self.manual_stop_button.config(state=tk.DISABLED)

def manual_processing(self):
    while self.manual_running:
        for i in range(4):
            if not self.manual_running:
                break
            self.update_traffic_lights(i)
            time.sleep(green_light_duration)
    self.reset_traffic_lights()

def detect_traffic_density(self):
    self.cap = cv2.VideoCapture(0)
    road_vehicle_counts = [0] * len(road_areas)

    while self.running:
        for i in range(len(road_areas)):
            if not self.running:
                break

            start_time = time.time()
            frame_counts = []
            saved_frame = None

            while time.time() - start_time < check_duration and self.running:
```

```
            ret, frame = self.cap.read()
            if not ret:
                print("Failed to capture video frame")
                break

            rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            results = model(rgb_frame)
            detections = results.pandas().xyxy[0].to_dict(orient="records")

            vehicle_count = 0
            for detection in detections:
                if detection['name'] in VEHICLE_LABELS:
                    vehicle_count += 1
            frame_counts.append(vehicle_count)

            for detection in detections:
                if detection['name'] in VEHICLE_LABELS:
                    xmin, ymin, xmax, ymax = int(detection['xmin']),
int(detection['ymin']), int(detection['xmax']), int(detection['ymax'])
                    cv2.rectangle(frame, (xmin, ymin), (xmax, ymax), (0, 255, 0),
2)
                    label = f"{detection['name']} {int(detection['confidence'] *
100)}%"
                    cv2.putText(frame, label, (xmin, ymin - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 2)
            x1, y1, x2, y2 = road_areas[i]
            cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 0, 0), 2)
```

```
                cv2.putText(frame, f"Road: {i + 1}", (20, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)
                average_count = int(np.mean(frame_counts))
                cv2.putText(frame, f"Avg Count: {average_count}", (20, 70),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)


                img = Image.fromarray(cv2.cvtColor(frame,
cv2.COLOR_BGR2RGB))
                imgtk = ImageTk.PhotoImage(image=img)
                self.canvas.create_image(0, 0, anchor="nw", image=imgtk)
                self.root.update_idletasks()
            road_vehicle_counts[i] = average_count


        sorted_counts = sorted(enumerate(road_vehicle_counts), key=lambda x:
x[1], reverse=True)
        for index, count in sorted_counts:
            self.update_traffic_lights(index)
            time.sleep(green_light_duration)


        self.reset_traffic_lights()


# Create the main window and run the app
root = tk.Tk()
app = TrafficDensityApp(root)
root.mainloop()
```