Real-time Traffic Monitoring System based on Deep Learning and YOLOv8

Saif B. Neamah and Abdulamir A. Karim

Department of Computer Sciences, University of Technology, Baghdad, Iraq

***Abstract*—Computer vision applications are important nowadays because they provide solutions to critical problems that relate to traffic in a cost-effective manner to reduce accidents and preserve lives. This paper proposes a system for real-time traffic monitoring based on cutting-edge deep learning techniques through the state-of-the-art you-only-look-once v8 algorithm, benefiting from its functionalities to provide vehicle detection, classification, and segmentation. The proposed work provides various important traffic information, including vehicle counting, classification, speed estimation, and size estimation. This information helps enforce traffic laws. The proposed system consists of five stages: The preprocessing stage, which includes camera calibration, ROI calculation, and preparing the source video input; the vehicle detection stage, which uses the convolutional neural network model to localize vehicles in the video frames; the tracking stage, which uses the ByteTrack algorithm to track the detected vehicles; the speed estimation stage, which estimates the speed for the tracked vehicles; and the size estimation stage, which estimates the vehicle size. The results of the proposed system running on the Nvidia GTX 1070 GPU show that the detection and tracking stages have an average accuracy of 96.58% with an average error of 3.42%, the vehicle counting stage has an average accuracy of 97.54% with a 2.46% average error, the speed estimation stage has an average accuracy of 96.75% with a 3.25% average error, and the size estimation stage has an average accuracy of 87.28% with a 12.72% average error.**

***Index Terms* – Computer vision, Deep learning, Object detection, Traffic monitoring, YOLOv8.**

I. Introduction

The emergence of smart cities and intelligent traffic systems nowadays aims at enhancing human lives by enabling efficient use of infrastructure resources, reducing risks such as traffic collisions, and improving drivers' and pedestrians' safety. The need for these types of systems is increasing across various domains such as transportation, public safety, and infrastructure management. In addition to the growth of cities, the increase in human population, and the number of vehicles, vehicular traffic data represents the most vital data source in smart city management systems. An effective analysis of this data can yield significant benefits for both citizens and governments. Traffic problems have increased as cities become larger. With the increasing number of vehicles, several network protocols have been developed to address this problem. These protocols require a data source that can continuously feed the network with traffic data provided by real-time traffic analysis systems (Dias, et al., 2023) (Farooq and Kanwal, 2023).

Road traffic accidents are one of the leading causes of mortality globally. In Iraq, traffic accidents have increased considerably, especially after 2003, as a result of the growth in the economy and population. The number of vehicles as of 2015 was 5.775 million, according to the 2018 World Health Organization (WHO) road safety report (WHO, 2018).

II. Related Works

There are a number of studies that are conducted on vehicle detection and vehicle speed estimation, but very few on vehicle size estimation. Some of these studies are described below:

In 2020, Berna, Swathi, and Devi, presented in their work entitled *"Distance and Speed Estimation of Moving Object Using Video Processing"* a distance and speed estimation in real-time with 90% precision using a fixed camera and a centroid method. Object detection is done by performing frame differencing and thresholding, and the speed is estimated by dividing the manually measured distance by the time difference, and the vehicle length is also estimated (Berna, Swathi, and Devi, 2020).

In 2020, Costa, Rauen and Fronza, presented in their work entitled *"Car Speed Estimation Based on Image Scale Factor"* a vehicle speed estimation with a longitudinal trajectory relative to the camera without reference markers. They used the image scale factor (ISF) in pixels to calculate the distance from the vehicle to the camera in different video

ARO-The Scientific Journal of Koya University
Vol. XI, No. 2 (2023), Article ID: ARO.11327. 14 pages
DOI: 10.14500/aro.11327
Received: 07 August 2023; Accepted: 25 October 2023
Regular research paper: Published: 16 November 2023
Corresponding author's e-mail: saif.b.neamah@uotechnology.edu.iq

frames, and they proposed to combine the ratio of $\frac{fw}{W}$ and $\frac{fh}{H}$ where f is the camera focal length, w, h are the sensor's width and height, and W, H are the image frame width and height into a single ISF, then estimate the distance and time difference to estimate the vehicle speed. The result of their work shows a maximum deviation of 2.2% for vehicle speed estimation (Costa, Rauen, and Fronza, 2020).

In 2021, Lin, Jeng, and Lioa, presented in their work entitled *"A Real-time Vehicle Counting, Speed Estimation, and Classification System Based on Virtual Detection Zone and YOLO"* the implementation of a real-time traffic monitoring system to count vehicles and estimate their speed with a classification facility. They based their work on the Gaussian mixture model (GMM) and you only look once v4 (YOLOv4). The GMM and a virtual detection zone are used for vehicle counting and detection. The GMM is used to perform background subtraction and foreground segmentation. The YOLOv4 model is used for vehicle classification, with a classification accuracy of 98.91% and an average absolute percentage error of vehicle speed estimation of 7.6% (Lin, Jeng and Lioa, 2021).

In 2022, Gupta et al. presented in their work entitled *"Vehicle Speed Detection System in Highway"* the implementation of a vehicle speed detection system using a video-based approach. The Haar cascade approach is used for vehicle detection and a correlation tracker, and speed estimation, they estimated the pixel per meter (PPM) ratio manually and calculated the speed by dividing the distance by the recorded time when the vehicle enters and leaves a region of interest. The detection accuracy was 95%, and for speed measurement, it was 92% (Gupta, et al., 2022).

In 2022, Shihab, Ghani, and Mohammed, presented in their work entitled *"Machine Learning Techniques for Vehicle Detection"* the implementation of a vehicle detection and classification system using two methods based on Haar cascade and YOLOv3. The detection results they got were 86.9% for the Haar cascade approach and 91.31% for the YOLOv3 approach, concluding that YOLO-based algorithms have better detection results than Haar cascade-based methods. YOLO-based methods are robust to different lighting conditions (Shihab, Ghani, and Mohammed, 2022).

This paper implements a real-time system based on the most recent CNN-based deep learning models that give more accurate detection results with more critical information regarding traffic, including vehicle counting, vehicle classification, vehicle speed estimation, and vehicle size estimation. The vehicle size can be useful in enforcing speed limit laws on roads, as we know truck speed limits, for example, differ from salon car speed limits. The built-in classification facility in our proposed system can estimate the vehicle speed and vehicle count based on the vehicle class. The vehicle size estimation was not thoroughly researched in previous studies. In this paper, we present a new way to estimate vehicle size in real time. Table I summarizes the related works and their limitations compared with our work.

III. Traditional Traffic Monitoring Systems

A variety of hardware technologies are available for collecting traffic data to facilitate traffic surveillance systems, including sensors, induction loops, and microwave radars. All hardware technologies have their limitations. Induction loops only affect the point of measurements, limiting their spatial coverage, and with significant traffic density, their accuracy can degrade. The majority of surface sensors have high installation and maintenance costs. Handheld radar guns that rely on the principle of the Doppler effect, in addition to their equipment's high cost, require an on-site operator and a line of sight to perform the speed estimation, can only be applied to one vehicle at a time, and suffer from shadowing where more than one wave is reflected from vehicles with different heights (Koyuncu and Koyuncu, 2018).

TABLE I
Summary of Related Works Limitations

Work	Year	Related works (limitations and comparison)
Distance and speed estimation of moving object using video processing	2020	Vehicle detection is based on the frame-differencing method. It is not clear if the system can track multiple vehicles; no vehicle size information is provided, and there is no vehicle counting facility. Our system tracks multiple vehicles and estimates the vehicle size with a counting facility.
Car speed estimation based on ISF	2020	It is not clear if the system works on multiple vehicles, and the system is not robust to motion blur; no vehicle size information is provided, and there is no counting facility. Our system works on multiple vehicles, is robust to motion blur, and the size information is estimated with a counting facility.
A real-time vehicle counting, speed estimation, and classification system based on a virtual detection zone	2021	There is no estimation of the vehicle's size. The system used an outdated YOLOv4 as a pre-trained object detector and classifier. Our system estimates the vehicle size, and the detection accuracy is higher based on the YOLOv8 model.
Vehicle speed detection system on highway	2022	The system used Haar cascades for vehicle detection and a correlation tracker. The detection and tracking are not powerful enough to deal with heavy traffic. There is no estimation of vehicle size information and no vehicle counting facility. Our system uses a deep-learning-based method that is more robust in vehicle detection and suitable for real-time applications. It also provides a courting facility with size estimation.
Machine learning techniques for vehicle detection	2022	The system is limited to multi-object detection and classification; there is no tracking applied and no counting facility; neither vehicle speed nor size information is provided. Used an outdated YOLOv3 model. Our system also provides multiple vehicle tracking and speed estimation in real time, based on the recent YOLOv8 model.

ISF: Image scale factor

IV. Deep Learning-based Traffic Monitoring Systems

Deep learning and convolutional neural network (CNN)-based object detection techniques can automatically extract features from input images and are more resilient to changes in illumination, shadows, and partial occlusions. The two primary methodologies of detection are the one-stage approach and the two-stage approach, which dominate the field of object detection. One-stage detectors, such as you only look once (YOLO) and single-shot detectors (SSD), approach object identification as a regression problem in which the bounding box coordinates and the object classes are predicted directly. However, two-stage detectors, like the region-based CNN (R-CNN), have two phases. Using a search approach, such as a selective method or a region proposal network, the first step is to produce a large number of region suggestions. The second step is to submit these region proposals for classification and bounding box regression. Two-stage detectors often have greater detection rates than one-stage detectors, but since there are more processes, they also take longer processing time (Liu, et al., 2021; Yasir and Ali, 2021).

V. CNNs

The CNN, also called ConvNet, is a very popular deep learning architecture that can learn directly from the input data without the need for manual human feature extraction. It includes *multiple* convolution and pooling layers. CNNs are specifically intended to deal with a variety of 2-dimensional shapes and are widely employed in the applications of computer vision, image segmentation, and object detection. The rapid development of GPU technology made CNNs so popular. In fact, one of the bottlenecks of deep neural networks is that training takes a long time because of the many hidden units in the network. But as GPUs became faster, this bottleneck was overcome (Sarker, 2021; Raheem and AbdulHussain, 2020).

In CNNs, the states of each layer are arranged according to a spatial grid structure; these spatial relationships are inherited from one layer to the next because each feature value is based on a small local spatial region in the previous layer. Each layer in the convolutional network is a 3-dimensional grid structure that has a *height*, *width*, and *depth*. The depth of a single layer refers to the number of *channels* in each layer; in the case of colored RGB images, the depth is three. Fig. 1 demonstrates the traditional CNN model (Alzubaidi, et al., 2021; Awotunde, et al., 2023).

VI. Object Detection with CNN

The problem of object detection can be efficiently solved with deep learning models. Object detection based on CNNs consists of two tasks: recognizing and localizing objects in the image. Recognition is a classification task that involves providing category information and the probability of the target. The other is a positioning task that involves finding the specific location of the target by utilizing bounding boxes with labels. There are various algorithms for object detection using CNNs, which are mainly divided into two main categories (Alzubaidi, et al., 2021) (AlNujaidi, AlHabib and AlOdhieb, 2023).

1. *Two-stage algorithms*: Like the R-CNN series, which generates at the first stage set ROIs that represent a set of category-independent bounding boxes in the image; at the second stage, it makes corrections based on the bounding box region to improve the final detection results. The two-stage algorithms give more accurate results, but they are more computationally expensive as they require the classification

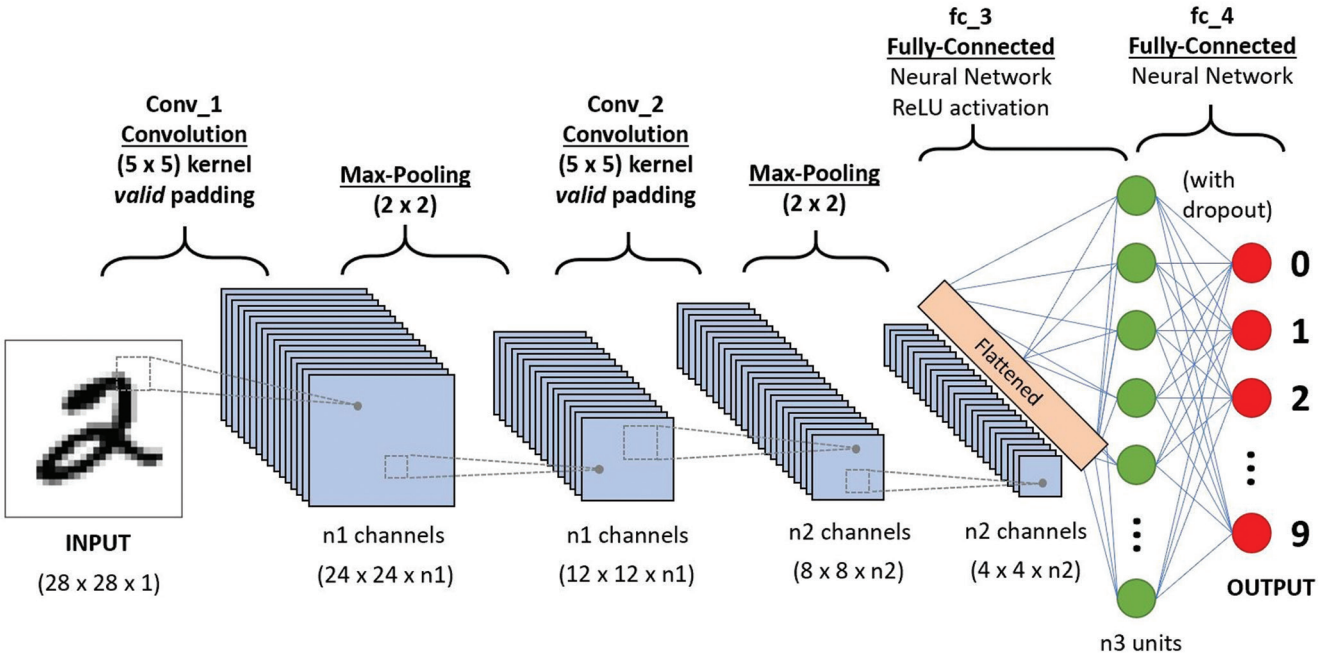

Fig. 1. Traditional CNN model. n is the number of image color channels.

network to be applied to a large number of ROIs, so this approach is slower than the one-stage algorithms (AlNujaidi, AlHabib and AlOdhieb, 2023) (Alzubaidi, et al., 2021).

2. *One-stage algorithms*: They merge the classification and regression into a single pass, like the YOLO series and SSD, divide the image into a fixed grid, and apply a classification network to each segment. The one-stage algorithms are fast but give less accurate detection results than the R-CNN algorithms (AlNujaidi, AlHabib and AlOdhieb, 2023) (Hakim and Fadhil, 2021).

VII. YOLO One-stage DNN Model

YOLO is a real-time object detection algorithm that uses a single CNN to predict the bounding boxes and class labels of objects in an image. *YOLOv1* was introduced in 2015, which frames the object detection problem as a regression problem instead of a classification problem that classifies each pixel in the image. To eliminate duplicate detections, YOLO divides the input image into a grid and predicts the bounding boxes for the same class along with its confidence values; the output is then followed by a non-maximum suppression (NMS). There are 24 CNN layers in YOLOv1 architecture, then two fully connected layers. All layers use leaky ReLU except for the last one, which uses a linear activation function. YOLO used (1*1) convolutional layers to reduce the number of feature maps and the network parameters. The loss function used is composed of multiple sum-squared errors. *YOLOv2* was introduced in 2016 with various improvements, including detecting over 9000 object categories, adding batch normalization, using anchor boxes, using the Darknet-19 architecture composed of 19 convolutional layers and five max-pooling layers. *YOLOv3* was introduced in 2018 and evolved from Darknet-19 to Darknet-53 with residual connections. *YOLOv4* was introduced in 2020 with weighted residual connections, cross-stage partial connections (CSP), cross-mini-batch normalization, self-adversarial training, and mish-activation function as the backbone. *YOLOv5* was introduced in 2021 and implemented under the PyTorch framework. It has a focus layer to reduce the number of parameters and a CSP, which extends shallow information in the focus layer to maximize functionality. *YOLOv6* was introduced in 2022 with a plain single-path backbone for small models and efficient multi-branch blocks for large models; this was done by proposing two scaled, re-parametrizable backbones and necks to accommodate models of different sizes. *YOLOv7,* which was introduced in 2022, proposed a planned re-parameterized model by merging multiple computational modules into one at the inference stage and made some architectural reforms. *YOLOv8* was introduced in 2023 and outperformed all previous models, as illustrated in Fig. 2 (Hussain, 2023) (Jocher, Chaurasia and Qiu, 2023).

VIII. YOLOv8 Architecture

YOLOv8 proposed a new backbone network with a new anchor-free detection head, which means it predicts directly the center of an object instead of the offset from a known anchor box. It also proposes a new loss function. The basic architecture of YOLOv8 consists of two major parts: the backbone for extracting feature maps and the head for detection, as illustrated in Fig. 3 (King, 2023).

The backbone contains a series of convolutional layers for different image resolutions and sizes, and then the features detected are passed through the advanced head for detection based on a loss function. New convolutional layers are used. The stem's first (6*6) convolution is replaced by a (3*3); the main building block was changed, and C2f replaced the YOLOv5 C3. The bottleneck is the same as in YOLOv5, but the first convolution's kernel was changed from (1*1) to (3*3) in the neck, and features are concatenated directly without forcing the same channel dimensions. This reduces the parameter count and the overall size of the tensors (Jocher, Chaurasia and Qiu, 2023).

IX. Bytetrack Multi-object Tracking Algorithm

A state-of-the-art algorithm based on a combination of feature pyramids, anchor-free detection, and multiple-scale training makes it compatible with YOLOv8. The algorithm tracks by associating every bounding box instead of only the high-scoring ones. Data association is the core of multi-object tracking, which first computes the similarity between trackelets and detection boxes. A tracklet is a sequence of detections that are likely to belong to the same object. The similarity metrics may include location, motion, and

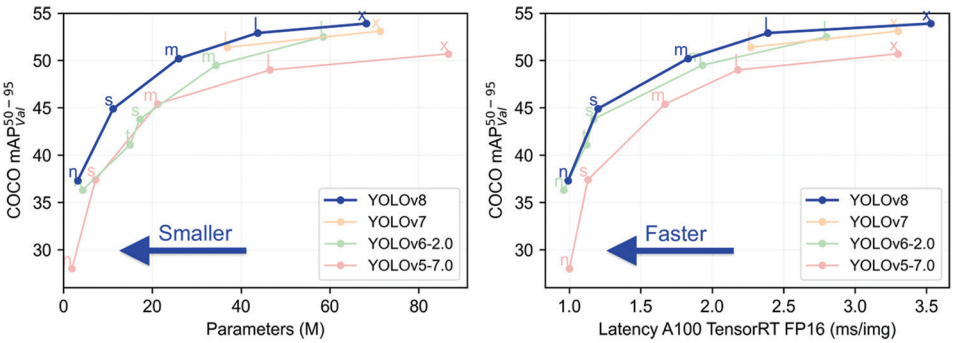

Fig. 2. YOLOv8 evaluation against recent YOLO models.

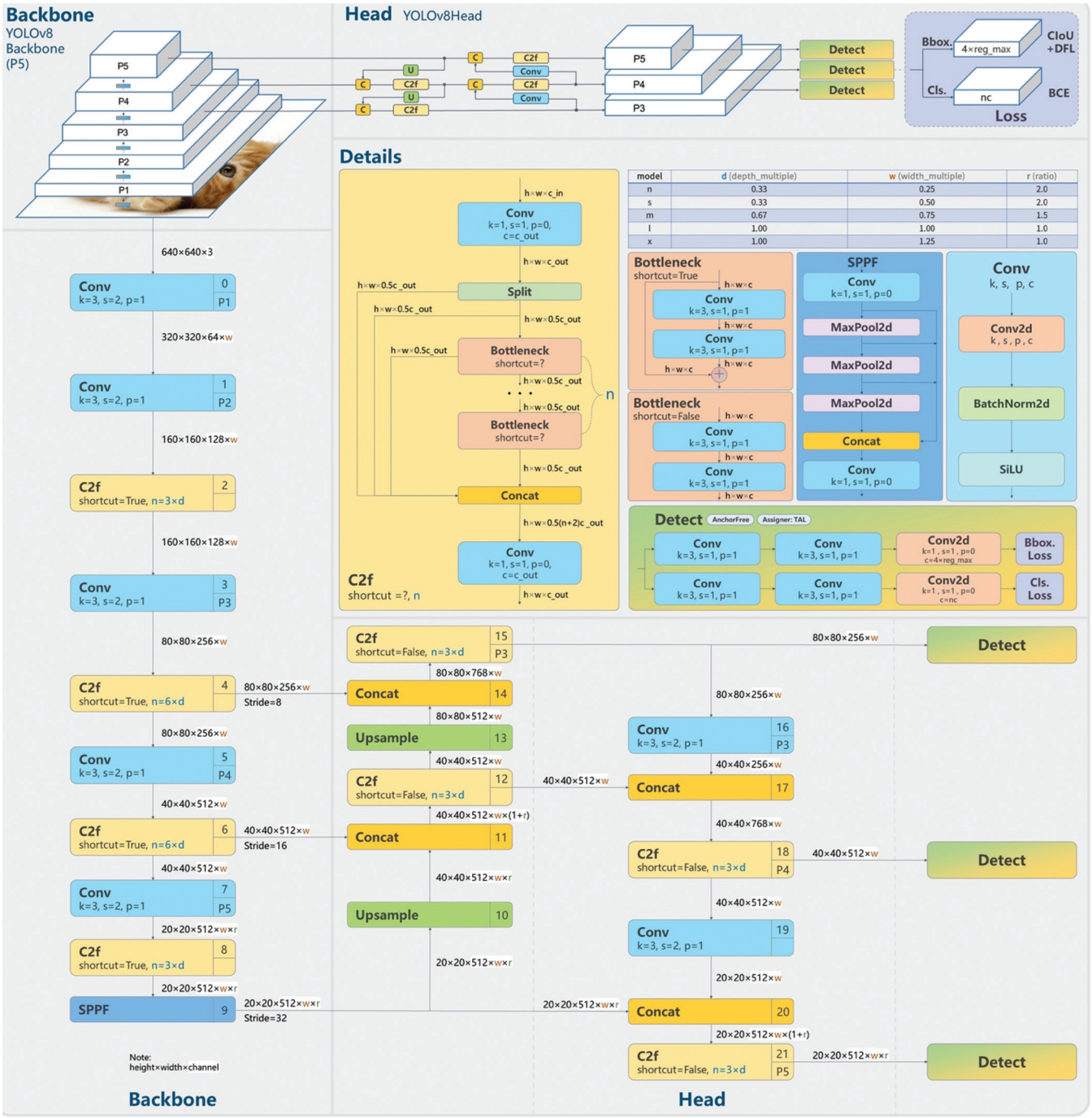

Fig. 3. YOLOv8 architecture.

appearance features. The ByteTrack algorithm tracks objects with high or low values of confidence by presenting a BYTE method for each detection box. First, the high-scoring boxes are tracked based on motion similarity or appearance similarity, and a Kalman filter is adopted to predict the location of the objects in the next frame. The similarity can be computed by the IoU or Re-ID feature distance between the predicted box and the detection box. After similarity computation, the matching strategy assigns identities to the objects using the Hungarian algorithm. Algorithm 1 provides the pseudocode for the BYTE algorithm (Zhang, et al., 2022).

X. The Proposed System Architecture

The general architecture of the proposed system consists of several stages that start with the preprocessing stage, which includes the camera calibration and calculation of the PPM ratio, calculating the ROIs, determining the video source, preparing to read the live video feed, dividing the live video into a series of frames, and resizing the video frames into the standard resolution (640*640). Then comes the detection stage for specified classes of vehicles that implement the YOLOv8 pretrained classification and segmentation model, and then a tracking algorithm is applied to the detected vehicles to track their movements. The tracking stage is

Algorithm 1: BYTE Algorithm

Input: V: video file; **D**: object detector; **θ**: detection score threshold
Output: T: tracks

Begin:
Step 1: Initialize T=0
Step 2: for frame f_k in V do
$D_k \leftarrow Det\ (f_k)$
$D_{high} \leftarrow 0$
$D_{low} \leftarrow 0$
Step 3: for d in D_k do
If d. score >θ then
$D_{high} \leftarrow D_{high} \cup \{d\}$
else
$D_{low} \leftarrow D_{low} \cup \{d\}$
Step 4: for t in T
$t.KalmanFilter(t)$
Step 5: Associate T and D_{high} using similarity1
Step 6: $D_{remain} \leftarrow$ remaining boxes from D_{high}
Step 7: $T_{remain} \leftarrow$ remaining tracks from T
Step 8: Associate T_{remain} in D_{low} using similarity2
Step 9: $T_{re\text{-}remain} \leftarrow$ remaining tracks from T
Step 10: $T \leftarrow T \backslash T_{re\text{-}main}$ // delete unmatched tracks
Step 11: for d in D_{remain} do
$T \leftarrow T \cup \{d\}$
end for
return
End

essential for the speed estimation stage; it assigns track IDs to the detected vehicles. Finally, size estimation is performed to calculate the size of segments provided by the segmentation model according to the calibration parameters. The following block diagram in Fig. 4 illustrates a general block diagram of the proposed system.

The system also includes a vehicle counting process, which counts the vehicles passing through the ROIs in real-time based on the center of the detected vehicle passing within the coordinates of the ROI lines. The counter is displayed on the video output, and the total number of vehicles passed during the observation period is recorded in the log file in CSV format. The tracking trial drawing process, which is an optional process that draws a tracking trail behind the tracked vehicles to give the viewer a sense of tracking, the output of the proposed system is presented as a video stream that is displayed to the viewer as long as the system is running. The stream can also be saved to the hard disk as a video file in MP4 format. The proposed system will also produce a log file that will contain all the important information calculated by the system, including the timestamp for the system's first run, each car's estimated speed and estimated size associated with the car tracking ID and class ID, the timestamp for each calculation, and the total car count number.

XI. The Proposed System's Stages

The proposed system consists of five stages; each stage leads to the next one, and every stage contains specific details that will be explained in the following sub-sections.

A. The Preprocessing Stage

The first stage contains three main steps, which are the camera calibration step, the ROI calculation step, and the input preparation step. The camera calibration step is essential for the speed and size estimation stages. The calibration is done using a reference object with known dimensions (one known car dimension, e.g.,) at the speed and size estimation ROIs. It is necessary to calculate the PPM ratio, which is used to estimate the real distance in the ROI and the real size of the detected vehicles. The second step is the ROI calculation. In the proposed system, two ROIs are specified, namely, ROI1 and ROI2, and three lines, namely, Line 1, Line 2, and Line 3, as illustrated in Fig. 5. The picture was taken by a phone camera at Mohammed Al Qassim Highway Street in Baghdad, Iraq.

The reason behind dividing the ROI into two regions is that we want to get the most accurate results possible, so the speed is calculated in ROI1, then calculated again in ROI2, and the average speed will be taken as the final result for the vehicle's estimated speed. The final step at this stage is

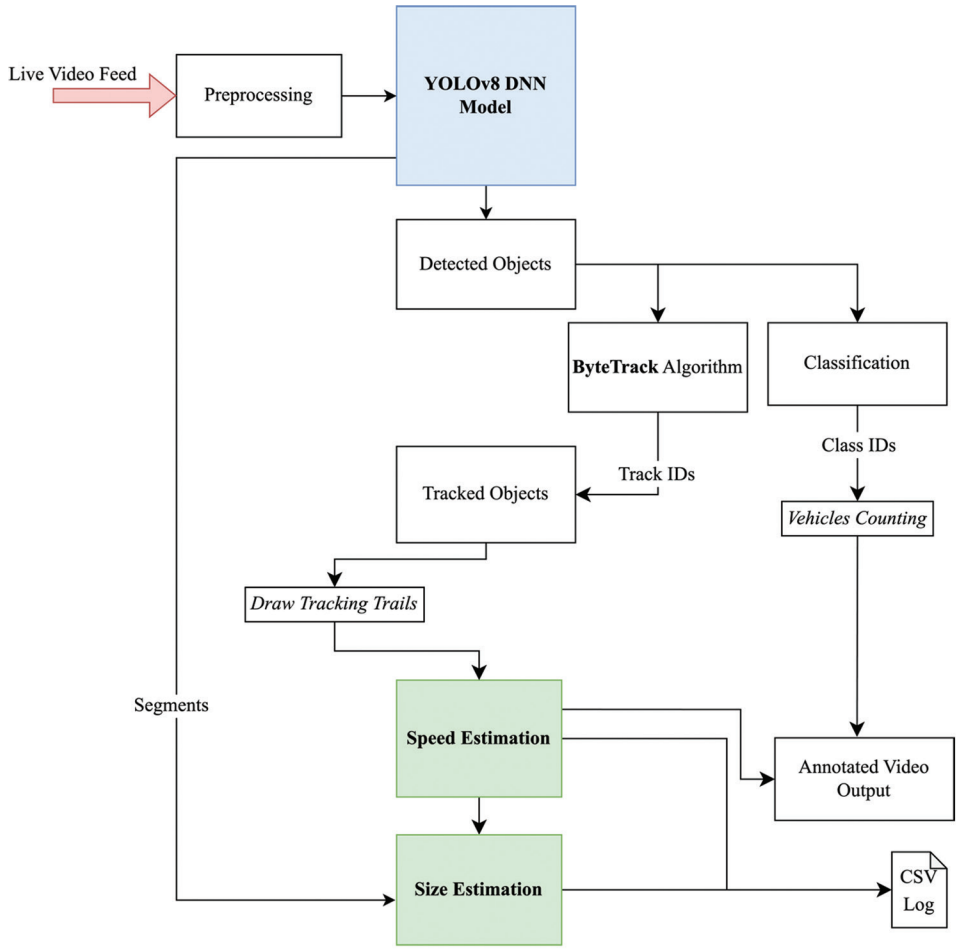

Fig. 4. The proposed system's block diagram.

Fig. 5. The proposed system's ROIs.

the input preparation step, where the source video input is determined and resized to (640*640) resolution, and then the video frames are extracted.

B. The Vehicle Detection Stage

The detection stage deploys the pretrained YOLOv8 DNN models. The detection algorithm consists of multiple layers of convolution (Conv), coarse-to-fine (C2f), concatenation (Concat), upsampling (Upsample), spatial pixel per feature (SPPF), and finally the segmentation process. The Conv layer involves the standard convolution operation of a sliding window with predefined kernels, a stride of 1, no padding, and the SiLU activation function, followed by batch normalization to improve the overall learning process. The C2f process involves a convolution operation with a kernel of size (1*1), no padding, and a stride of 1, then the output is entered into a split operation that is fed to the bottleneck; the bottleneck itself consists of two convolutional layers with residual connections. Then the concatenation operation is performed, and finally another convolution is performed.

The SPPF is spatial pyramid pooling (fast), similar to the SPP used in YOLOv5, which involves two convolution layers and three max-pooling layers, used to capture multi-scale information and improve the detection performance of the network, enabling the network to detect objects at different sizes more accurately. The upsample process involves increasing the size of the matrices by using a transposed convolution process with a stride of 2 and a padding of 1. It increases the spatial resolution of the feature maps, which helps improve object localization. The Concat process involves the concatenation of a list of tensors into a tensor of one dimension. This will enable feature fusion, model flexibility, and the handling of multiple inputs. Fig. 6 demonstrates the multiple layers of the DNN segmentation model.

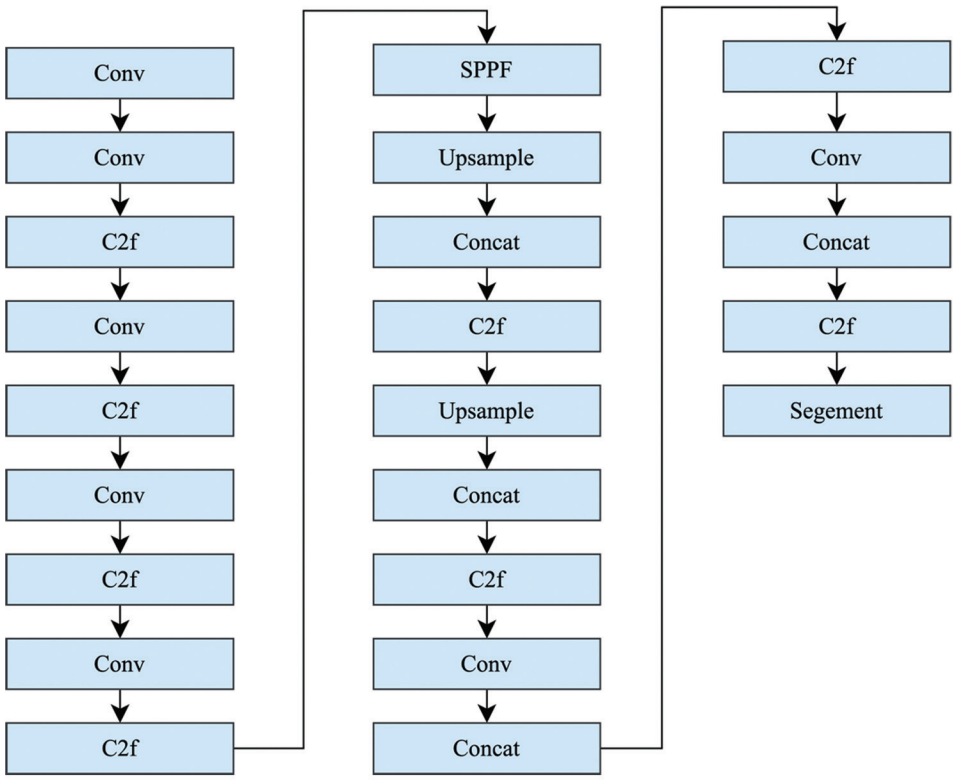

Fig. 6. YOLOv8 segmentation model layers.

C. The Vehicle Tracking Stage

The tracking stage depends on the results of the detection stage as the tracker tracks the detected bounding boxes of the vehicles. The ByteTrack algorithm is applied to track the moving vehicles in the scene based on their detected bounding boxes, and then the algorithm will assign a track identification number (*track ID*) for each tracked vehicle. The tracking stage is essential to the proposed system as it will be used in the vehicle counting process and the speed estimation stage for calculating the speed of the tracked vehicle in the scene. The vehicle counting process starts after the detection stage and during the tracking stage. The counting process is performed in the predefined ROIs. Line 1 of ROI1 will be used to define the vehicle's entering point, and Line 2 will define the vehicle's leaving point. At the entry line, the center of each detected vehicle is calculated using the following equation:

$$(cx, cy) = \frac{(x + (x + w))}{2}, \frac{(y + (h + y))}{2} \quad (1)$$

Where: *(x, y)* is the detected object's bounding box top left corner coordinates. *(w, h)* is the detected object's bounding box width and height. *(cx, cy)* is the center coordinate of the detected object.

Several important pieces of information are stored in a queue, including the car ID, the time of entering in seconds (TIME1), and the vehicle's calculated center (CENTER1). This information will be used later in the speed estimation stage. When the vehicle approaches Line 1 by a predefined *offset*, it will be considered that the vehicle has entered ROI1, and a vehicle counter will increase. The offset is manually fine-tuned and is essential to the system's performance because, in real-time systems that employ heavy processing of DNN models, the system will encounter some cases of missed detections during the vehicle's movement that can be solved with this offset to expand the detection zone. Fig. 7 demonstrates the three lines in the ROI with offset values.

D. The Speed Estimation Stage

The object counting process provides this stage with three queues (CARS_Q1, CARS_Q2, and CARS_Q3), each corresponding to a line in the ROI. The queues contain the car ID, the timestamp, and the car center coordinates. The speed will be calculated two times in the proposed system: The first speed calculation will be between Line 1 and Line 2, and the second is between Line 2 and Line 3. Finally, the two calculations are averaged to produce the final estimated speed that will be displayed on the screen and stored in the log file along with the car ID number and the timestamp. Equations 2 and 4 use the Euclidean distance to calculate the distance between two centers, and Equations 3 and 5 divide the results by a calibration parameter (PPM) ratio.

$$distance1 = \sqrt{\begin{matrix}(CENTER2.x - CENTER1.x)^2 \\ +(CENTER2.y - CENTER1.y)^2\end{matrix}} \quad (2)$$

$$real\ distance1 = \frac{distance1}{PPM} \quad (3)$$

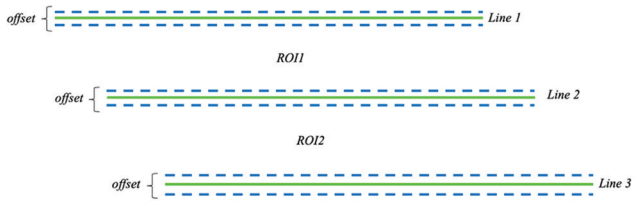

Fig. 7. The detection lines offsets.

$$distance2 = \sqrt{\begin{matrix}(CENTER3.x - CENTER2.x)^2 \\ +(CENTER3.y - CENTER2.y)^2\end{matrix}} \quad (4)$$

$$real\ distance2 = \frac{distance2}{PPM} \quad (5)$$

The time difference is also calculated for the two regions as the difference between the time recorded at Line 2 and the time recorded at Line 1, as in Equations 6, and the difference between the time recorded at Line 3 and Line 2, as in Equation 7.

$$\Delta Time1 = Time2 - Time1 \quad (6)$$

$$\Delta Time2 = Time3 - Time2 \quad (7)$$

Finally, the speed is calculated for the two regions as in Equations 8 and 9, and then the average speed is calculated between the two speeds as in Equation 10.

$$Speed1 = \frac{real\ distance1}{\Delta Time1} \quad (8)$$

$$Speed2 = \frac{real\ distance2}{\Delta Time2} \quad (9)$$

$$Average\ Speed = \frac{Speed1 + Speed2}{2} \quad (10)$$

The average speed is calculated in meters per second units; to display the speed in kilometers per hour (kmh) or miles per hour (mph), unit conversion equations 11 and 12 are applied.

$$Speed_{kmh} = Average\ speed\ *3.6 \quad (11)$$

$$Speed_{mph} = Speed_{kmh} * 1.6093470879 \quad (12)$$

The details of the speed estimation at ROI1 and ROI2 are demonstrated in Algorithm 2.

E. The Size Estimation Stage

The size estimation stage depends on the output of the pretrained segmentation model of YOLOv8 in the detection stage, which generates segments (masks) for each vehicle. The following steps are performed to estimate the area and size: Step 1: The generated mask, which is a series of points that surround the detected vehicle, is used to draw a polygon filling the detected vehicle using the *fillPolly* OpenCV method. Step 2: The area of the polygon is calculated using the *contourArea* OpenCV method, as in Equation 13.

$$pixel_{area} = contourArea(mask) \quad (13)$$

Step 3: The calculated area is divided by PPM to get an estimation of the real area of the detected vehicle, as Equation 14 shows.

$$real_{area} = \frac{pixel_{area}}{PPM} \quad (14)$$

As the vehicle gets closer to the camera, its image will increase in area, so we need to have a fixed region for measuring the area of the vehicle in that region. In the proposed system, the area of the vehicle is measured if the center of the vehicle's bounding box reaches line 2. The estimated area is stored in the log file alongside the car ID. Step 4: The size of the vehicle is a three-dimensional measurement, and since the proposed system utilizes a single camera, one dimension of the detected vehicle will be lost since images are stored in computers as two-dimensional data structures. The least variable vehicle dimension is the vehicle width, and since our proposed system performs vehicle classification into four vehicle classes (car, bus, truck, and motorcycle), we assumed that the width of these vehicle types is approximately equal for each class; vehicles usually vary greatly in length and height. Through an Internet search, we found that the average salon car width is approximately 1.5 m. The average bus width is approximately 2 m. The average truck width is approximately 2.5 m, and the average motorcycle width is approximately 0.5 m. Estimating the size of the vehicle requires calculating the vehicle's length and height. Using the top-side view camera position and acquiring the vehicle mask through the segmentation stage, we can find the two dimensions by taking the point with the lowest y value and the point with the highest y value to represent the two points of the vehicle length. Applying the Euclidian distance, we will obtain the vehicle's length. Similarly, from these points, we can find the height, then apply the size equation 15 to get the approximated size in pixels, and then apply equation 16 to estimate the real size in meters by dividing the size in pixels over the PPM ratio.

$$pixel_{size} = vehicle_{length} * vehicle_{width} * vehicle_{height} \quad (15)$$

$$real_{size} = \frac{pixel_{size}}{PPM} \quad (16)$$

Fig. 8 demonstrates the approximated vehicle's length and height from the segmentation polygon points. Algorithm 3 demonstrates the steps to estimate the size of detected vehicles.

XII. Results and Discussion

The results of the proposed system are presented on the system's output screen with all the annotations applied to it by the multiple stages of the proposed system. Fig. 9 shows the proposed system's screen of results.

The detection stage will draw a bounding box around the detected vehicles; the bounding box of each vehicle will be annotated with the vehicle's tracking ID assigned by the ByteTrack tracking algorithm, the class name extracted from

Algorithm 2: The Speed Estimation Algorithm

Input: CAR_ID, **CAR_Q1, CAR_Q2, CAR_Q3:** list of queues, **PPM**
Output: SPEED_Q1, SPEED_Q2: queues with estimated speed values

---

Start:
Step 1: SPEED_Q1 = [ ], SPEED_Q2 = [ ]
offset = 20
speed = speed1 = speed2 = 0
Step 2: check5 = cy <= (L2.start.y1 + offset) // object at Line 2
check6 = cy >= (L2.start.y1 - offset)
check7 = cx <= L2.end.x2
Step 3: if check5 and check6 and check7
For each object in **CARS_Q1**:
Step 4: dx = CENTER2.x – CENTER1.x
dy = CENTER2.y – CENTER1.y
Step 5: distance1 = math.sqrt(math.pow(dx, 2) + math.pow(dy, 2))
Step 6: real_distance1 = distance1 / PPM
Step 7: delta_time1 = TIME2 – TIME1
Step 8 speed1 = (real_distance1 / delta_time1) * 3.6 // km/h
Step 9: SPEED_Q1.append([CAR_ID, speed1])
End For
Step 10: check8 = cy <= (L3.start.y1 + offset) // object at Line 3
check9 = cy >= (L3.start.y1 - offset)
check10 = cx <= L3.end.x2
Step 11: if check8 and check9 and check10
Step 12: For each object in **CARS_Q2**:
Step 13: dx = CENTER3.x – CENTER2.x
dy = CENTER3.y – CENTER2.y
Step 14: distance2 = math.sqrt(math.pow(dx, 2) + math.pow(dy, 2))
Step 15: real_distance2 = distance2 / PPM
Step 16: delta_time2 = TIME3 – TIME2
Step 17: speed2 = (real_distance2 / delta_time2) * 3.6 // km/h
Step 18: SPEED_Q2.append([CAR_ID, speed2])
End For
Step 19: if CAR_ID in SPEED_Q1 and CAR_ID in SPEED_Q2
speed = int(SPEED_Q1[CAR_ID] + SPEED_Q2[CAR_ID] / 2)
speed_mph = int(speed * 1.6093470879)
now = datetime.now( )
Step 20: display (CAR_ID, speed, now) // show car id, speed, time on screen
log (CAR_ID, speed, now). // store car id, speed, time on log file
log (CAR_ID, speed_mph, now). // store car id, speed, time on log file
End

Fig. 8. Vehicle's height and length.

the COCO dataset after performing classification by the CNN model, and the confidence score. Instance segmentation is also provided in the proposed system, and it will produce a segment mask for each detected vehicle; the segment masks are filled with red to reflect the vehicle's blob. The speed estimation will also be shown at the top center of the screen. The text consists of the estimated speed, the vehicle ID, and a timestamp. This text will dynamically change as every vehicle passes through the ROI.

The classification process will assign different colors to each vehicle class as demonstrated in Fig. 10.

The proposed system was evaluated on test videos taken at different locations and lengths with different camera positions to evaluate the detection, tracking, classification, and counting processes, while the system was running on an

Algorithm 3: The Area and Size Estimation Algorithm

Input: CAR_ID, **CLASS_IDS**, **MASKS**, **PPM**
Output: SIZE_Q: queue with estimated size values

Start:
SIZE_Q = [ ]; offset = 20
Step 1: For each mask in **MASKS**:
x1 = mask [0][0] // get the first point
y1 = mask [0][1]
Step 2: check_mask1 = y1 <= (L2.start.y1 + offset) // object at Line 2
check_mask2 = y1 >= (L2.start.y1 - offset)
check_mask3 = x1 <= L2.end.x2
Step 3: if check_mask1 and check_mask2 and check_mask3:
Step 4: pixels_area = contourArea(mask)// calculate mask area
Step 5: real_area = pixels_area / PPM
Step 6: **if** CAR_ID not in SIZE_Q:
SIZE_Q.append([CAR_ID, real_area])
now = datetime.now( )
log (CAR_ID, real_area, now) // store car id, area, time on log file
Step 7: p1 = min(mask[1]) // vehicle length points
p2 = max(mask[1])
v_length = math.sqrt((p2[0] – p1[0])**2+(p2[1] – p1[1])**2)
Step 8: p1 = min(mask[0]) // vehicle height points
p2 = max(mask[6])
v_height = math.sqrt((p2[0] – p1[0])**2+(p2[1] – p1[1])**2)
Step 9: v_width = 1.5 // default for salon cars
If CLASS_ID == 3:
v_width = 0.5 // motorcycle class
else if CLASS_ID == 5:
v_width = 2 // bus class
else if CLASS_ID == 7:
v_width = 2.5 // truck class
End if
Step 10: real_size = (v_length * v_height * v_width) / PPM
now = datetime.now( )
log (CAR_ID, real_size, now) // store car id, size, time on log file
End For
End

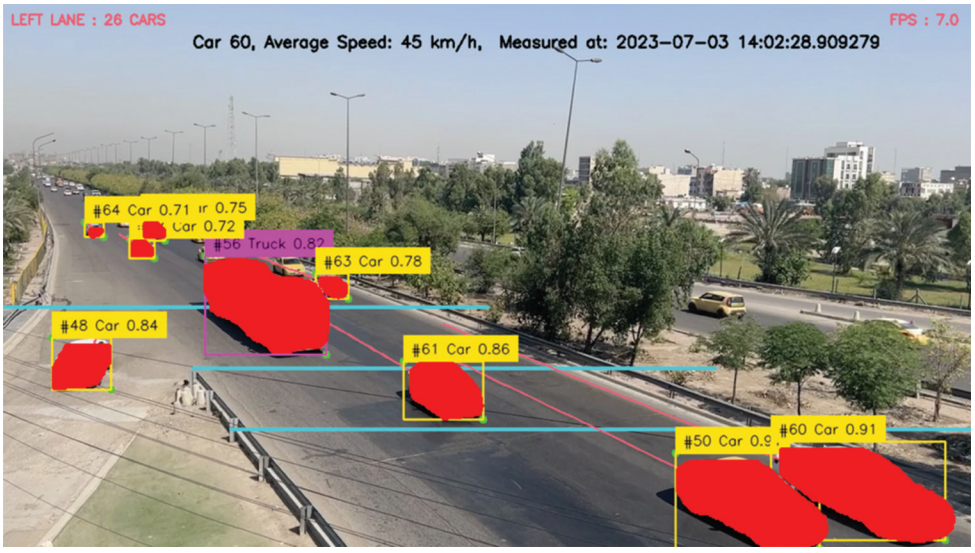

Fig. 9. System's output screen.

http://dx.doi.org/10.14500/aro.11327

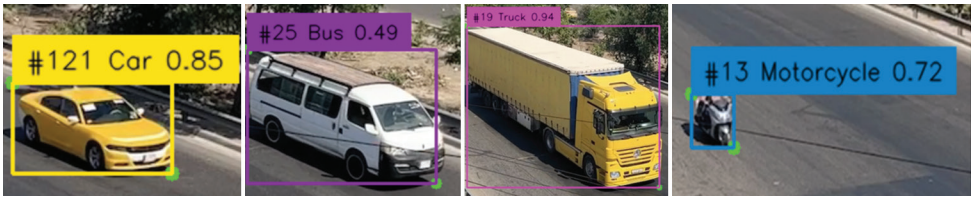

Fig. 10. System's vehicle classification.

Table II
Vehicle detection with tracking evaluation

Test video	Model size	Ground-truth vehicles	System's detections	FPS	Accuracy (%)	Error (%)
Video1	Nano	70	67	48.8	95.7	4.3
Video1	Small	70	68	42	97.1	2.9
Video1	Medium	70	69	28.7	98.5	1.5
Video1	Large	70	69	21.5	98.5	1.5
Video1	Extra-Large	70	69	15.9	98.5	1.5
Video2	Nano	45	41	50.2	91.1	8.9
Video2	Small	45	43	43.8	95.5	4.5
Video2	Medium	45	43	29.6	95.5	4.5
Video2	Large	45	43	21.7	95.5	4.5
Video2	Extra-Large	45	44	16	97.7	2.3
Video3	Nano	28	25	50	89.2	10.8
Video3	Small	28	26	43.3	92.8	7.2
Video3	Medium	28	27	29.3	96.4	3.6
Video3	Large	28	28	21.7	100	0
Video3	Extra-Large	28	28	16	100	0
Video4	Nano	50	48	48	96	4
Video4	Small	50	49	41.5	98	2
Video4	Medium	50	49	28.5	98	2
Video4	Large	50	50	21	100	0
Video4	Extra-Large	50	50	15.6	100	0
Video5	Nano	37	34	46	91.8	8.2
Video5	Small	37	35	40	94.5	5.5
Video5	Medium	37	36	28	97.2	2.8
Video5	Large	37	36	21	97.2	2.8
Video5	Extra-Large	37	37	15.7	100	0
Averages					**96.58**	**3.42**

Table III
Vehicle counting evaluation

Test video	Model size	Ground-truth count	System's count	FPS	Accuracy (%)	Error (%)
Video1	Nano	60	61	49.4	98.3	1.7
Video1	Small	60	60	42.6	100	0
Video1	Medium	60	60	28.9	100	0
Video1	Large	60	61	21.5	98.3	1.7
Video1	Extra-Large	60	62	16	96.6	3.4
Video2	Nano	45	45	50.8	100	0
Video2	Small	45	44	43.6	97.7	2.3
Video2	Medium	45	43	29.3	95.5	4.5
Video2	Large	45	45	21.6	100	0
Video2	Extra-Large	45	45	16	100	0
Video4	Nano	40	40	47.6	100	0
Video4	Small	40	43	41	93	7
Video4	Medium	40	44	28.1	90	10
Video4	Large	40	43	21.1	93	7
Video4	Extra-Large	40	43	15.7	93	7
Video5	Nano	21	22	44.6	95.4	4.6
Video5	Small	21	21	39.8	100	0
Video5	Medium	21	21	27.5	100	0
Video5	Large	21	21	20.7	100	0
Video5	Extra-Large	21	21	15.5	100	0
Averages					97.54	2.46

Table IV
Vehicle classification evaluation

Class	Precision (%)	Recall (%)	Accuracy (%)	F1-score (%)
Car	95	97.9	94.7	96.3
Bus	100	36.3	94.7	53.2
Truck	81.4	100	96.2	89.7
Motorcycle	100	66.6	99.7	79.9

Nvidia GTX 1017 GPU with different YOLOv8 model sizes as listed in the following Tables II-IV.

For speed estimation evaluation, the VS13 dataset (Djukanović, et al., 2022) with known ground truth vehicle speed values is used to evaluate the system, and the results are demonstrated in Table V.

The size estimation stage is also evaluated by known ground truth values of known vehicle sizes; the result obtained is shown in Table VI.

The results of the proposed system showed excellent estimations for the multiple system stages. For the detection and tracking stage, the results we tested on multiple video samples and different CNN model sizes showed an average accuracy of 96.58% and an average error of 3.42%. For the vehicle counting process, the results show an average accuracy of 97.54% with a 2.46% average error. For the vehicle classification stage, the results from a single test video show an accuracy of 94.7% for the salon car class, 94.7% for the bus class, 96.2% for the truck class, and 99.7% for the motorcycle class. The precision values for the car, bus, truck, and motorcycle were 95%, 100%, 81.4%, and 100%, respectively. The recall values obtained were 97.9%, 36.3%, 100%, and 66.6%. The f1-score values were 96.3%, 53.2%, 89.7%, and 79.9% for the four vehicle classes. For the speed estimation stage, the results obtained had an average accuracy of 96.75% and a 3.25% average error. Finally, for the size estimation stage, the results obtained compared with ground-truth values of vehicle size gave 87.28% average accuracy with a 12.72% average error. The low accuracy in the bus class is because there are no buses in the test videos, only minibuses, and the model is trained on large regular buses; minibuses are classified as salon cars.

Table V
Vehicle speed estimation evaluation

Test video	Ground-truth speed value (km/h)	System's speed estimation (km/h)	FPS	Accuracy (%)	Error (%)
Eval30	30	28	29	93.3	6.7
Eval42	42	41.4	29	98.5	1.5
Eval56	56	53.8	29	96	4
Eval64	64	62.4	29	97.5	2.5
Eval71	71	68.4	29	96.3	3.7
Eval88	88	90	29	97.7	2.3
Eval94	94	90.6	29	96.3	3.7
Eval101	101	99.4	29	98.4	1.6
Averages				**96.75**	**3.25**

Table VI
Vehicle size estimation evaluation

Vehicle ID	Ground-truth size value (m^3)	System's size estimation (m^3)	FPS	Accuracy (%)	Error (%)
1	11.53	15	30	76.8	33.2
5	13.7	14	30	97.8	2.2
11	16.52	16	30	96.8	3.2
15	14.37	13	30	90.4	9.6
24	9.18	9	30	98	2
37	14.18	13	30	91.6	8.4
39	14.18	14	30	98.7	1.3
65	16.68	7	30	41.9	58.1
97	10.14	11	30	92.2	7.8
118	19.03	24	30	79.2	20.8
Averages				87.28	12.72

XIII. Conclusions and Future Works

The proposed system provided traffic monitoring information in real-time using a combination of cutting-edge deep learning technology and state-of-the-art algorithms. The system gave excellent results in all of its stages compared with previous works, with high accuracy and low error rates.

The system was implemented with various adjustable parameters such as frame skipping, enabling and disabling segmentation, and device-agnostic code that made the system run on different hardware configurations and GPUs, which made the system flexible and configurable for any video input type, resolution, or weather condition. The classification stage in the proposed system is useful in enforcing different speed limits based on the vehicle class. The counting process can also count vehicles in general or count them based on their classes for more specific information.

The proposed system estimates the vehicle's size with a novel approach based on segmentation masks generated from the YOLOv8 segmentation model, fixing one dimension (the vehicle's width) and calculating the other two (the vehicle's length and height) from the polygon points of the segmentation mask. The vehicle size information is useful for traffic control, as speed limits differ for different vehicle sizes. For example, trucks have speed limits that are different from those of salon cars. Size information can also help detect large vehicles that are not allowed to drive on certain roads at certain times of the day. The vehicle height is also important for driving in tunnels and under bridges.

As suggestions for future work to further develop the system, an automatic ROI detection algorithm can be implemented, and for size estimation, a stereo camera can be deployed to get more accurate estimations. In addition, the system can be integrated with an OCR algorithm for reading the vehicle's license plate for vehicles with speed limit violations.

References

AlNujaidi, K., AlHabib, G., and AlOdhieb, A., 2023. Spot-the-camel: Computer vision for safer roads. *International Journal of Artificial Intelligence and Applications* (*IJAIA*), 14(2), pp.1-10.

Alzubaidi, L., Zhang, J., Humaidi, A.J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M.A., Al-Amidie, M., and Farhan, L., 2021. Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data*, 8(1), p.53.

Awotunde, J.B., Jimoh, R.G., Imoize, A.L., Abdulrazaq, A.T., Li, C.T., and Lee, C.C., 2023. An enhanced deep learning-based deepfake video detection and classification system. *Electronics*, 12(1), p.87.

Berna, S.J., Swathi S., Devi, C.Y., and Varalakshmi, D., 2020. Distance and speed estimation of moving object using video processing. *International Journal for Research in Applied Science and Engineering Technology (IJRASET)*, 8(5), pp.2605-2612.

Costa, L.R., Rauen, M.S., and Fronza, A.B., 2020. Car speed estimation based on image scale factor. *Forensic Science International*, 310, p.110229.

Dias, T., Fonseca, T., Vitorino, J., Martins, A., Malpique, S., and Praça, I., 2023. *From Data to Action: Exploring AI and IoT-Driven Solutions for Smarter Cities*. Springer, Cham.

Djukanović, S., Bulatović, N., and Čavor, I., 2022. *A Dataset for Audio-video Based Vehicle Speed Estimation*. Cornell University, New York, pp.1-4.

Farooq, M.S., and Kanwal, S., 2023. Traffic Road Congestion System using by the Internet of Vehicles (IoV), *Networking and Internet Architecture*, Cornell University, New York, arXiv:2306.00395, pp.1-9.

Gupta, U., Kumar, U., Kumar, S., Shariq, M., and Kumar, R., 2022. Vehicle speed detection system in highway. *International Research Journal of Modernization in Engineering Technology and Science*, 4(5), pp.406-411.

Hakim, H., and Fadhil, A., 2021. Survey: Convolution neural networks in object detection. *Journal of Physics: Conference Series*, 1804, pp.1-18.

Hussain, M., 2023. YOLO-v1 to YOLO-v8, the rise of YOLO and its complementary nature toward digital manufacturing and industrial defect detection. *Machines*, 11, p.677.

Jocher, G., Chaurasia, A., and Qiu, J., 2023. *YOLO by Ultralytics* (*Version 8.0.0*). Available from: https://github.com/ultralytics/ultralytics [Last accessed on 2023 Aug 07].

Jocher, G., Chaurasia, A., and Qiu, J., 2023. *YOLOv8 Docs by Ultralytics* (*Version 8.0.0*). Available from: https://docs.ultralytics.com [Last accessed on 2023 Aug 07].

King, R., 2023. *Github Repo MMYOLO*. Available from: https://github.com/open-mmlab/mmyolo/tree/main/configs/yolov8 [Last accessed on 2023 Aug 07].

Koyuncu, H., and Koyuncu, B., 2018. Vehicle speed detection by using camera and image processing software. *The International Journal of Engineering and Science* (*IJES*), 7(9), pp.64-72.

Lin, C.J., Jeng, S.Y., and Lioa, H.W., 2021. A real-time vehicle counting, speed estimation, and classification system based on virtual detection zone and YOLO. *Mathematical Problems of Applied System Innovations for IoT Applications*, 2021, p.1577614.

Liu, C., Huynh, D., Sun, C., Reynolds, M., and Atkinson, S., 2021. A vision-based pipeline for vehicle counting, speed estimation, and classification. *IEEE Transactions on Intelligent Transportation Systems*, 22(12), pp.7547-7560.

Raheem, F.A., and AbdulHussain, A.A., 2020. Deep learning convolution neural networks analysis and comparative study for static alphabet ASL hand gesture recognition. *Journal of Xidian University*, 14(4), pp.1871-1881.

Sarker, I.H., 2021. Deep learning: A comprehensive overview on techniques, taxonomy, applications and research Directions. *SN Computer Science*, 2, p.420.

Shihab, M.R., Ghani, R.F., and Mohammed, A.J., 2022. Machine learning techniques for vehicle detection. *Iraqi Journal of Computers, Communications, Control and Systems Engineering* (*IJCCCE*), 22(4), pp.1-12.

WHO, 2018. *Global Status Report on Road Safety 2018*. World Health Organization, Geneva.

Yasir, M.A., and Ali, Y.H., 2021. Review on real time background extraction: Models, applications, environments, challenges and evaluation approaches. *International Journal of Online and Biomedical Engineering*, 17(2), pp.37-68.

Zhang, Y., Sun, P., Jiang, Y., Yu, D., Weng, F., Yuan, Z., Luo, P., Liu, W., and Wang, X., 2022. ByteTrack: Multi-Object Tracking by Associating Every Detection Box, *Computer Vision and Pattern Recognition*, Cornell University, New York, arXiv:2110.06864, pp.1-14.