Hindawi
Journal of Advanced Transportation
Volume 2021, Article ID 5551976, 15 pages
https://doi.org/10.1155/2021/5551976

Research Article

Towards AI-Based Traffic Counting System with Edge Computing

Duc-Liem Dinh, Hong-Nam Nguyen, Huy-Tan Thai, and Kim-Hung Le

University of Information Technology-VNU-HCM, Ho Chi Minh City, Vietnam

Correspondence should be addressed to Kim-Hung Le; hunglk@uit.edu.vn

Received 23 February 2021; Revised 25 May 2021; Accepted 17 June 2021; Published 28 June 2021

Academic Editor: Jinjun Tang

The recent years have witnessed a considerable rise in the number of vehicles, which has placed transportation infrastructure and traffic control under tremendous pressure. Yielding timely and accurate traffic flow information is essential in the development of traffic control strategies. Despite the continual advances and the wealth of literature available in intelligent transportation system (ITS), there is a lack of practical traffic counting system, which is readily deployable on edge devices. In this study, we introduce a low-cost and effective edge-based system integrating object detection models to perform vehicle detecting, tracking, and counting. First, a vehicle detection dataset (VDD) representing traffic conditions in Vietnam was created. Several deep learning models for VDD were then examined on two different edge device types. Using this detection, we presented a lightweight counting method seamlessly combining with a traditional tracking method to increase counting accuracy. Finally, the traffic flow information is obtained based on counted vehicle categories and their directions. The experiment results clearly indicate that the proposed system achieves the top inference speed at around 26.8 frames per second (FPS) with 92.1% accuracy on the VDD. This proves that our proposal is capable of producing high-accuracy traffic flow information and can be applicable to ITS in order to reduce labor-intensive tasks in traffic management.

1. Introduction

In the past few years, the intelligent transportation system has become a fundamental branch of smart city construction, playing an essential role in public transport planning, management, and security [1]. A typical ITS is operated on the basis of a traffic flow analyzer that estimates the number of vehicles on the road in a specific interval. Deeply investigating this information may reveal useful information about the current traffic conditions (e.g., traffic density and congestion level) or predicting abnormal events (e.g., traffic jams and lane occupancy). During different periods, various vehicle movement patterns are also inspected to identify slow or immobile vehicles, resulting in congestion. This could be alleviated by using adaptive signal timing of traffic lights, which is adaptively calculated from current traffic flow information of intersections [2]. Counting the number of vehicles is thus an integral component of ITS needed to improve traffic efficiency.

Despite the fact that the widespread use of Internet of Things (IoT) aims to connect every physical object to the Internet, building a connected environment in transportation from a wireless sensor network has been facing major challenges and is still far from reality [3]. Special sensors (e.g., magnetic coil and ultrasonic detection) are commonly deployed along the roadside to observe traffic conditions. However, the deployment of these sensors requires expensive costs and alteration in transportation infrastructure. Their sensory data is most likely limited in both quantity and quality due to transmission issues (e.g., disconnected and interfered by environmental factors) [4]. Hence, traffic counting from low-cost surveillance cameras has emerged as a particularly attractive candidate for automation of traffic flow control in ITS because of the escalation in advanced artificial intelligence (AI) technologies in computer vision. Compared with the sensory-based method, a video-based traffic counting system (VTCS) not only simplifies deployment and maintenance processes but also seamlessly integrates with AI to detect, track, and identify each vehicle and its behaviors [5]. For example, the vehicle reidentification (Reid) application combines with AI to identify vehicle owners' information via recognizing

vehicle license plate. Consequently, a VTCS using AI is undoubted in the traffic flow analyzer of ITS.

Deep learning (DL), a subtechnique of AI, is the most commonly used approach to increase vehicle detection performance [6]. It makes use of many nonlinear hidden layers for supervised/unsupervised feature extraction, transformation, pattern analysis, and classification, where each successive layer takes the output from the previous layer as input. The features in data are automatically revealed instead of manually adding in traditional machine learning algorithms. The main limitation relates to the fact that executing DL algorithms requires massive computation capability and training data [7]. These algorithms are thus primarily performed in data centers on the cloud. Although integrating DL into ITSs under such a cloud-centric paradigm has drawn more attention from other researchers in the last several years, it still encounters innumerable challenges to fulfill smart city requirements: (1) Transmitting collected data from end devices to cloud is inevitably time-consuming, whereas critical applications stringently require low latency to instantly react with abnormal behaviors. (2) Transmitting data over the network and storing these data on distant servers may raise numerous issues related to security- and privacy-preserving, especially sensitive data about user's behaviors. One viable solution to these problems is edge computing, considered as complementary of cloud computing [8]. Data processing is preferred to cloud servers to execute on edge devices. Moving these computing tasks closer to data sources may considerably reduce latency and security risks.

In this paper, we introduce an AI-based traffic counting framework running in edge devices. The framework aims at effectively obtaining traffic flow information via three steps: vehicle detecting, tracking, and counting. For the purpose of accurately detecting various vehicle types, several common object detection approaches are retrained and optimized to be compatible with constrained devices before thoroughly examining on two representative single-board computers (GPU and TPU). With regard to the counting step, we design a lightweight counting method based on the cosine similarity concept that is seamlessly integrated with the traditional tracking method named Deep SORT. In addition, we established a VDD dataset representing traffic conditions in Vietnam, mostly consisting of motorcycles to validate the efficiency of our proposal. The experiment results show that the proposed framework is capable of producing high-accuracy traffic flow information in real time (26.8 frames per second with 92.1% accuracy on the VDD dataset). An interesting point to note here is that the implementation of our proposal can also be applied to multiple surveillance scenarios such as crowd monitoring and counting visitors in commercial areas. Our main contributions can be summarized as follows:

(i) We present an AI-based traffic counting framework operating directly on edge devices to obtain traffic flow information effectively. A vehicle object detection dataset (VDD) representing traffic conditions in Vietnam is also created to evaluate the performance of our proposal.

(ii) To select the appropriate vehicle detection model for edge devices, we optimize, retrain, and examine various object detection approaches on two types of single-board computers (SBCs) (Google Coral Dev Board and Nvidia Jetson Nano).

(iii) We propose a lightweight counting method based on the cosine similarity concept that is well suited with the traditional tracking method.

(iv) We show that the proposed framework can obtain high-accuracy traffic flow information in real time while operating on SBCs (acting as edge devices).

(v) Our framework is packed into a Docker container that is publicly available for the research and development of open communities (https://github.com/MrLiem/SSD_MobileDet_SORT).

The remainder of the paper is organized as follows. In Section 2, we present the related works. We then introduce the proposed system in Section 3. The experiment results are reported in Section 4. Finally, Section 5 concludes the article.

2. Related Works

The recent years have seen the rapid improvement of AI and its application across smart agriculture [9] and critical infrastructure development [10] and particularly in VTCS, which is widely used in ITS and attracted more attention from the research community in the last few years. This is an inevitable result of the emergence of advanced image processing and AI technologies. Based on the survey in [11], we may divide the vision-based vehicle counting method into two categories: regression-based methods and detection-based methods.

2.1. The Regression-Based Methods. The regression-based methods aim to exploit global image characteristics (e.g., color histogram and pixel density) to identify and count the number of vehicles. The authors in [12] introduce the Maximum Excess over SubArrays (MESA) distance concept for density estimation. Improving the MESA, L. Fiaschi et al. [13] propose an ensemble of randomized regression trees consisting of dense features as an input for the counting process. In [14], the authors introduce a cascade regression approach to measure and classify vehicles based on low-level features for each foreground segment. Similarly, Liu et al. [15] present a Hierarchical Classification based Regression (HCR) model that employs the codec metadata of compressed videos for extracting the batch of low-level features. The study in [16] incorporates a locally temporal regression method with the spatial regression. Reference [17] designs a convolution neural network (CNN) to regress vehicle spatial density images. Meanwhile, Zhang et al. count the number of vehicles by using an FCN-rLSTM network to extract spatial-temporal information from density maps [18].

2.2. The Detection-Based Methods. In the ITS, most detection-based methods identify the vehicle in the video by analyzing image features. In [19, 20], scale-invariant feature

transform (SIFT) and speeded-up robust features (SURF) are formed from image keys and interest points. These extracted features are demonstrated to be effective for detecting vehicles. Likewise, Dalal et al. [21] extract histogram of oriented gradients (HOG) features using silhouette contours, especially the head, shoulders, and feet. Reference [22] creates a set of Haar-like features, including four edge features, eight line features, and two center-surround features. These features combined with SVM or AdaBoost could significantly increase the performance of the detection model. However, the mentioned approaches are unstable and easily affected by large rotations in images. Their detection accuracy is thus strongly affected when changing the camera angle.

To overcome this problem, deep learning-based object detections are applied by Girshick et al. [23] to combine the region proposal method with CNNs (R-CNN) for searching all possible regions of the object. To lighten the feature extraction process, the authors in [24] extract the feature maps only once from the entire image before training the detectors. Reference [25] improves R-CNN and SPP-net by applying Fast R-CNN, which supports single-stage training with multitask loss function and bounding-box regression. Ren et al. [26] integrate a selective search mechanism in Faster R-CNN to accelerate the inference operation. The evaluation shows that Faster R-CNN is 10x faster than Fast R-CNN and 250x faster than R-CNN while maintaining similar accuracy. Despite being 250x faster than R-CNN, Faster R-CNN is still insufficient for real-time context with the detection speed lower than 10 FPS [27]. Liu et al. [28] propose Single Shot MultiBox Detector (SSD) replacing RPN by uniform extraction. This enhancement improves the detection speed up to 19 FPS (SSD512) and 46 FPS (SSD300). In a similar attempt, a serial of You Only Look Once (YOLO) algorithms [29–33] developed by Joseph Redmon could improve both the object's detection speed and accuracy. These algorithms use a convolutional network to process multiple bounding boxes and return class probabilities for those boxes on entire images.

For the tracking method, Tomasi et al. [34] propose Kanade-Lucas-Tomasi (KLT) feature tracker built from pyramid representation. Meanwhile, the authors in [35] propose the spatiotemporal relations that can be applied in a video to extract the movement trajectory information of vehicles. In [36–38], KLT is used to extract vehicle trajectories and count vehicles. The authors in [39] combine Kalman Filter and Hungarian algorithm to build Simple Online and Real-time Tracking (SORT) supporting multi-object tracking with high performance. Wojke et al. propose an extension of SORT named Deep SORT [40] which joins the appearance information in data association to improve the tracker's robustness.

Several studies further estimate the traffic flow parameters for ITS, such as vehicle category, density, speed, and traffic accidents. Reference [41] calculates the average speed of a traffic stream, density, and volume from counting vehicle results. Gao et al. [42] classify the vehicle size into lite, small, medium, and large based on vehicles' motion flow. Works in [43] build a real-time monitoring framework using ontology and latent Dirichlet allocation (OLDA) and bidirectional long short-term memory (Bi-LSTM) which has been proven to have higher accuracy compared to the previous models. Meanwhile, several works use relevant text data instead of images or videos for assessing the traffic flow parameters. The data could be social networking data retrieved from the most relevant documents, reviews, and tweets from social media and news articles. For example, in [44], the authors propose the word embedding model to interpret semantic meanings and a low-dimensional vector of each word in the sentence. A fuzzy ontology-based lexicon method is also exploited to increase the word embedding model's accuracy.

Besides these traffic flow variables, many other studies also focus on improving information security while transferring data from edge devices to the cloud in ITS. There are a number of ways in which the data reliability can be extended and enhanced. For example, Li et al. [10] apply the watermark embedding and extraction technique to secure the exchange data between devices and the cloud. A secure management paradigm used for big data context is discussed in [45]. Reference [46] presents a mobile edge architecture supporting software defined system. The authors in [47] build reliable and stable clusters for transmitting data based on trust degree estimation. In a similar attempt, a new AI-based routing protocol [48] is introduced to increase the efficiency and security of data in ITS. Meanwhile, the authors in [49–51] exploit the blockchain technology for authentication and authorization, suitable for large-scale intelligent applications.

3. Traffic Counting System

In this section, we broadly discuss how the proposed traffic counting system works. First of all, Figure 1 shows the architecture overview of our system, including its functional components. In our vision, a low-cost camera integrating with an edge device (single-board computers) could act as an AI camera that effectively counts the number of running vehicles on the street. Then, the counting result is transferred to a server on the cloud where the traffic information is analyzed and interactively visualized. A block diagram given in Figure 2 summarizes the detecting, tracking, and counting vehicle operations on edge devices in the proposed system. Compared with the traditional cloud paradigm, our approach significantly reduces the end-to-end latency and consumed network resources. In more detail, our system is composed of the following main components:

(a) *USB Camera.* It is directly plugged into a single-board computer for constantly capturing the traffic vehicles and extracting to high-quality video streaming, which is the input of object detection models running directly on edge devices. In our experiment, the output video resolution is set at 720p.

(b) *Edge Device.* It is a central processing component used to perform computational tasks. Edge devices could be single-board computers with sufficient

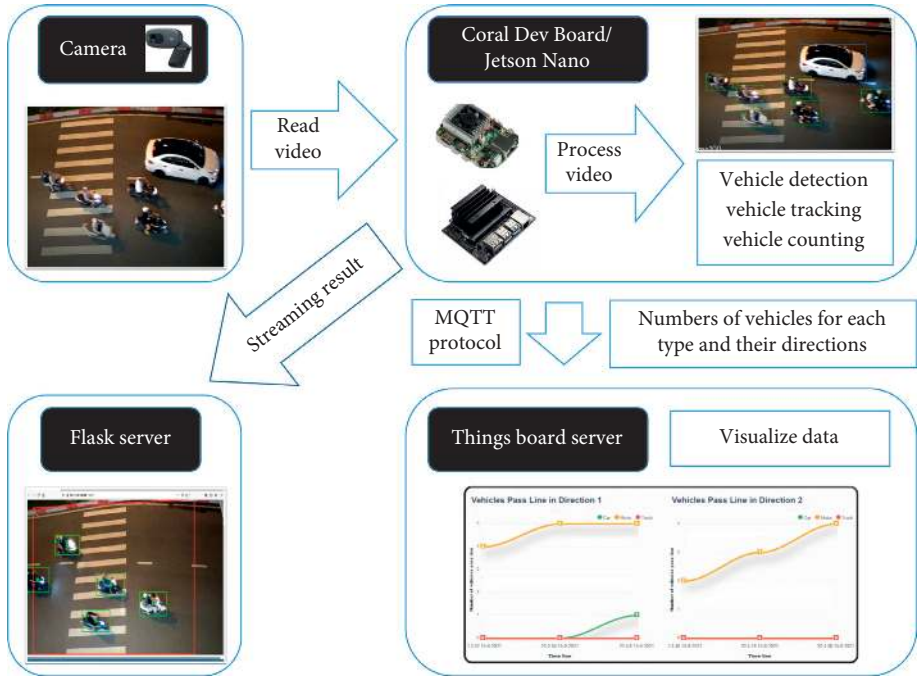

Figure 1: The architecture overview of the proposed edge-based system for traffic flow detection. Vehicle detecting, tracking, and counting are directly performed on edge devices (e.g., Google Coral Dev Board and Nvidia Jetson Nano).

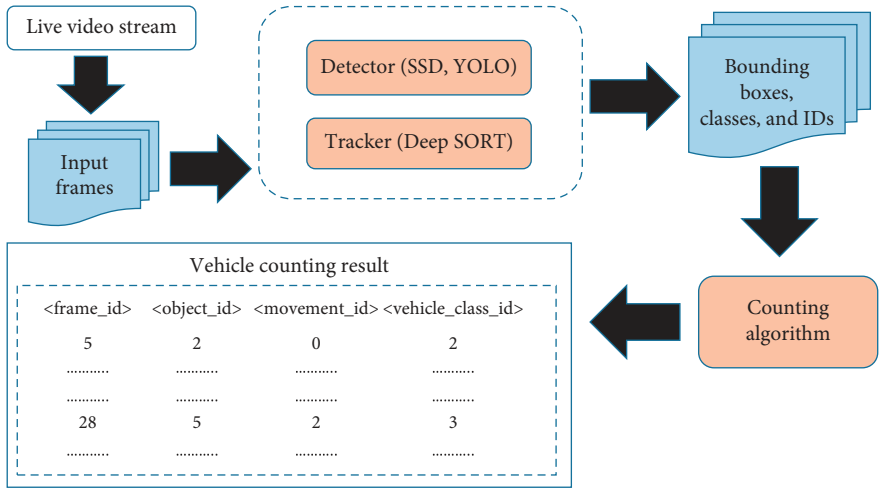

Figure 2: The block diagram depicts the data flow in the proposed system. The video input goes through detector and tracker components to create bounding boxes and identification before counting by our counting algorithm.

computational power (e.g., Nvidia Jetson Nano and Google Coral Dev Board) to count the number of vehicles in observed videos in two steps: (1) identifying vehicles in the video frames by using a deep learning model and (2) assigning all vehicles specific IDs and counting their occurrences. Then, the counting results are transmitted to a cloud server for further processing via MQTT protocol.

(c) *ThingsBoard Server.* It is a cloud server used to store and visualize the traffic information collected from several edge devices [52]. The server offers the administrators a general view of traffic conditions in different places.

(d) *Flask.* This component is a micro web framework written in Python. It is used to build a Live Video Streaming Server to stream live videos from edge devices to web browsers for administration tasks.

3.1. Vehicle Detecting. In this study, we retrained existing models using the transfer learning technique instead of building from scratch. Training a model with no computed weights or bias may require a vast amount of computing time and training data that is insufficient for edge computing [53]. In contrast, transfer learning enables reusing a model that has been already trained for a specific task. The model is further trained by a smaller dataset described in detail about a particular context.

The model's efficiency is thus significantly increased and tightly fits in the defined context. In the proposed system, we employed TensorFlow object detection API (available at https://github.com/tensorflow/models/tree/master/research/object_detection) to retrain the model with a self-created Vietnamese vehicle dataset. In detail, we examined six models for operating inference object detection task on single-board computers: SSD MobileNet V1 300×300, SSD MobileDet 320×320, SSD MobileNet V2 320×320, SSD MobileNet V2 FPNLite 320×320, SSD MobileNet V2 FPNLite 640×640, and SSD MobileNet V1 FPN 640×640 models with checkpoint available on TensorFlow Detection Model Zoo [54].

To decrease the model size and resource consumption (CPU and RAM), we quantized the models with entirely 8-bit fixed-point numbers from 32-bit floating-point numbers (including weights and activation outputs). This quantization makes the model smaller and faster without significantly affecting the neural network's inference accuracy. We applied either quantization-aware training (recommended) or full-integer posttraining quantization for total compatibility with the Edge TPU. In more detail, quantization-aware training utilizes "fake" quantization nodes to replicate the effect of 8-bit values in the neural network graph during the training process [55]. The model has better tolerance for lower precision values because the 8-bit weights are adjusted through training instead of converting later. Full-integer posttraining quantization estimates the range of floating-point values in each network layer using a small subset of data (extracted from either training or evaluating datasets) [56].

In general, selecting the best vehicle detection model according to both accuracy and speed is likely to be impossible to obtain. This may be explained by the fact that there are two-stage detection methods (e.g., R-CNN, Fast R-CNN, Faster R-CNN, and Mask R-CNN) detecting objects through two steps: (1) the model suggests a set of regions of interest based on a region proposal network; (2) the region proposals are dispatched to the channel for bounding-box regression as well as object classification. Such models achieve the highest accuracy rates, but they are often slow and require significant computational resources. On the other hand, the single-stage detectors such as YOLO (You Only Look Once) and SSD (Single Shot MultiBox Detector) ignore the region proposal phase and the detection process is executed directly over a dense number of possible locations. They are thus faster and simpler than former approaches, but their detection accuracy is slightly decreased. To find the suitable vehicle detection models running on edge devices, we empirically evaluate several object detection methods (including YOLO family and SSD architecture) on Coral Dev Board and Nvidia Jetson Nano devices. The comparison results are reported in detail in Section 4.

3.1.1. YOLO. You Only Look Once (YOLO) is a network specifically designed for fast and accurate real-time object detection. Its detection accuracy is comparable with common object detection algorithms, such as RetinaNet and Faster-RCNN [57]. YOLO network is lightweight and compact, making it suitable for real-time embedded applications, especially executing directly the classification and the localization in the image as a regression problem [29]. Since 2016, different YOLO versions have been released, such as YOLO9000 [30], YOLOv4 [32], YOLOv4-tiny [58], and YOLOv5 [33]. These versions gradually increase the general framework's accuracy. YOLOv4 and YOLOv5 not only have high detection accuracy but also perform well with small targets. Smaller models have also been released (e.g., YOLOv4-tiny and YOLOv5-small), suitable for AI applications on the edge devices and embedded systems.

In contrast to other region proposal classification networks (Fast R-CNN) which perform detection on different region proposals in an image, YOLO does not seek interested regions in the input image. It typically splits the image into a 19×19 grid cell in charge of predicting K bounding boxes. The overall architecture of the algorithm is illustrated in Figure 3. The network contains 24 convolutional layers and then comes through two fully connected layers. Instead of using the inception modules implemented by GoogLeNet, the 1×1 reduction layers followed by 3×3 convolutional layers are simply utilized.

3.1.2. Single Shot MultiBox Detectors. Single Shot MultiBox Detector (SSD) is one of the most well-known deep learning object detection methods. In SSD, the multiscale feature maps for detection are employed, as shown in Figure 4. As a result, with 300×300 and 500×500 images, SSD achieves 72.1% mAP and 75.1% mAP, respectively, on VOC2007 test at 58 FPS on an Nvidia Titan X, which outperforms Faster R-CNN models. SSD algorithm has been developed by researchers on many different backbone models as a feature extractor. The backbone model is typically a pretrained image classification network such as MobileNet V1, V2, Inception, and MobileDet. MobileNet V1 is an efficient neural network model for mobile and embedded vision applications. It is based on depthwise separable convolution that comprises two steps: depthwise convolution and pointwise convolution. A single filter is applied to each input feature map channel by depthwise convolution, and then the pointwise convolution utilizes the 1×1 convolution to combine the outputs of the depthwise layer. Using MobileNet, there is nine times less computation overhead than the standard convolution with only a slight reduction in the accuracy [60]. MobileNet V2 is an upgraded version of MobileNet V1 by using inverted residual blocks with bottlenecking features [61]. Its number of parameters is significantly lower than that of the original MobileNet V1. MobileDet is a novel image classification model architecture attaining state-of-the-art results on various mobile accelerators. It is a step backward from depthwise convolution, which is considered as more efficient for edge devices and mobile processors. The experimental results show that MobileDet surpasses MobileNet V3 on the COCO object detection tasks by 1.7 mAP at comparable mobile CPU inference latencies and MobileNet V2 + SSDLite by 1.9 mAP on CPU [62].

3.2. Vehicle Tracking and Counting. Frame-by-frame analysis of incoming video streams does not guarantee temporal

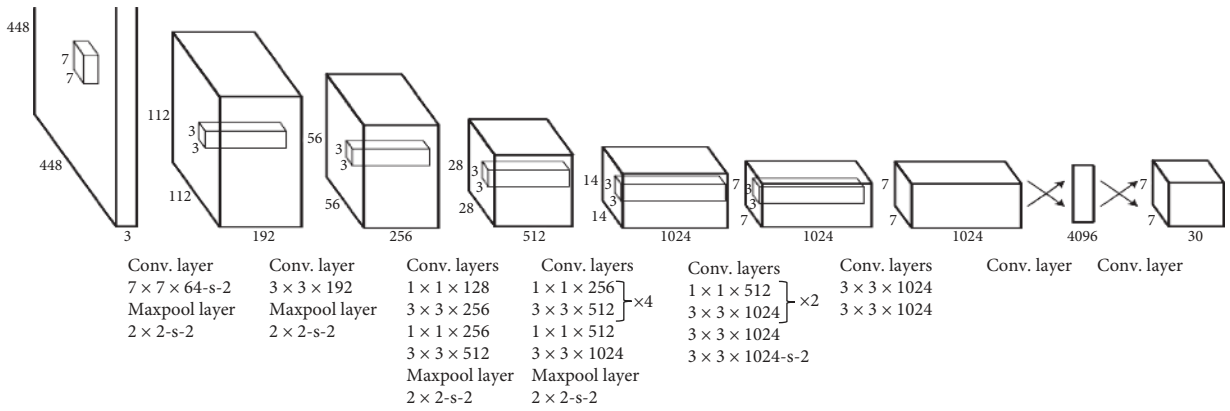

Figure 3: YOLO network architecture (from [31]).

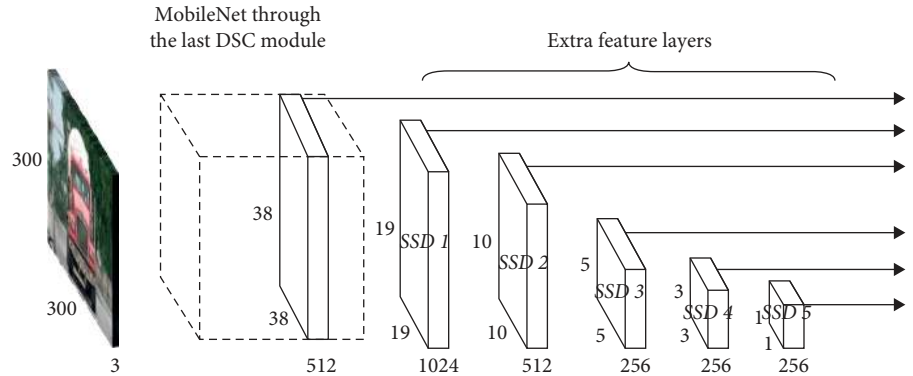

Figure 4: MobileNet-SSD network architecture (from [59]).

continuity in the determined objects, and thus counting function cannot be immediately performed. Time-consistent labeling of objects must be achieved between frames to provide a reliable counting method for moving objects. In addition, the method has to prevent undercounting (unidentified objects are mistakenly identical) and overcounting (identical objects are misidentified as distinct). In order to achieve this goal, we employed an extension of a multiobject tracking method named Deep SORT (the Simple Online and Realtime Tracking with a Deep Association metric). An overview of the Deep SORT's components is illustrated in Figure 5. Deep SORT uses a combination of Kalman Filter and Hungarian algorithm to track the objects between frames consistently. Kalman Filter predicts trajectory tracks, while the Hungarian algorithm is responsible for matching the predicted trajectory tracks with the objects in the frames. The outstanding feature in Deep SORT is to integrate an appearance descriptor, which is a wide residual network, with two convolutional layers followed by six residual blocks [63]. This descriptor makes the tracker more robust against object misses and occlusions, effectively reducing the number of identity switches. Consequently, Deep SORT is well suitable for real-time scenarios.

Combining with the tracking method, we developed a novel algorithm to effectively count different vehicle types after leaving the Region-of-Observation (ROO). The proposed algorithm consists of two steps:

(i) **Step 1** is to determine whether the vehicle is inside or outside the ROO. First of all, a horizontal line to the right of each point of the bounding box is defined and extended to infinity. We then count the number of times the line intersects with polygon edges. A point is considered to be inside the polygon if either counting of intersections is odd or it lies on an edge polygon. In contrast, the point is outside. In case at least one point of the bounding box is within the polygon, we may conclude that the bounding box is in the ROO area, as shown in Figure 6.

(ii) **Step 2** is to determine the direction of movement (DOM) based on trajectory.

For one-way and two-way roads, the vehicle's DOM is selected by calculating the cosine similarity score between the trajectory of each vehicle and the predefined DOMs. As shown in Figure 7, let x and y be two vector pairs (AB, CD) and (AB, EF). We compute the cosine similarity score between these vectors with the following equation:

$$\text{similarity} = \cos(\phi) = \frac{x.y}{|x| \cdot |y|}, \tag{1}$$

where $|x|$ and $|y|$ are the Euclidean norms of vectors x and y. We then compare them to find the vehicle's

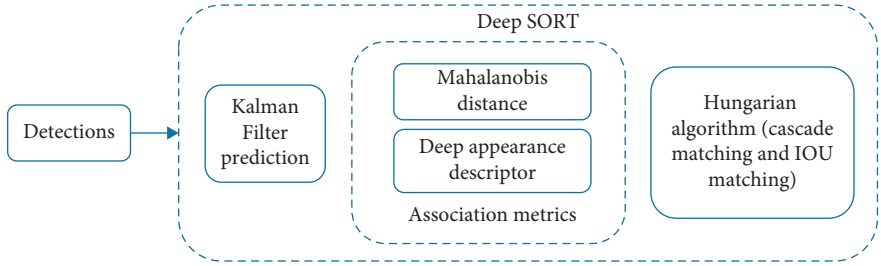

Figure 5: A block diagram depicts data flow in Deep SORT to track the vehicle between frames. Kalman Filter predicts trajectory tracks, while the Hungarian algorithm matches the predicted trajectory tracks with the detected objects in the frames.

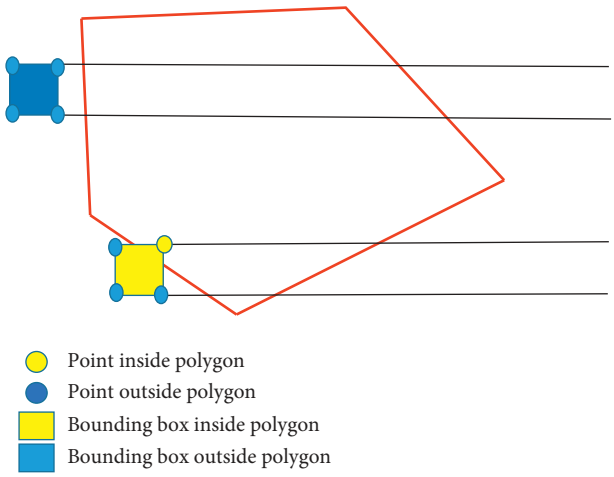

Figure 6: The mechanism is used to check whether a bounding box is inside ROO or not.

DOM. Points C and D are the coordinates of the center box of the first and last frames of objects entering the ROO area. A set A, B, E, F is a list of predefined points to describe DOMs.

For intersections, the cosine similarity approach is ineffective because there are many similar directions. Therefore, we proceed to compute the minimum distance from the center point of the bounding box to the edges of the polygon of the first and last frames that appeared in the ROO area to determine the direction of the vehicle's movement. Figure 8 shows the process of determining the vehicle's DOM based on calculating the minimum distance method. Assuming that there is a vehicle in ROO with the black and white circle corresponding to the center point of the first frame and the last frame of the recorded object, the min distances between each of the two center points and the edges of the polygon illustrated by two yellow lines in plot (b) were calculated. We can see that the distance from the black point (the center point of the first frame) to edge A is the smallest distance implying that the vehicle comes from the direction of edge A; similarly the vehicle exits the ROO at the direction of edge D because the distance from the white point (center point of the last frame) to edge D is the most minimal. Finally, we can deduce that the vehicle moves from edge A to edge D, corresponding to arrow number 2 in plot (a).

4. Results and Discussion

4.1. Dataset Generation. This section presents the vehicle detection dataset (VDD) produced for vehicle detection in real-world conditions. The dataset consists of 22 videos of urban roads and intersection scenes recorded using several traffic cameras. These videos are 23 minutes in length [64]. The cameras are set up on the roadside and capture vehicles moving into or leaving the different roads and intersections. Videos in VDD are taken from various angles, periods, and weather conditions. For each video, we split it into an 18-minute video for training and a 5-minute video for testing. We then cut training videos into image frames and label them every 3 seconds. We obtain about 2,000 annotated images of approximately 10,000 objects. Furthermore, we adopt the data augmentation technique to increase the training set's diversity by applying transformations such as image rotation and modifying brightness. The dataset set after augmentation is about 4700 annotated images. In the final step, we convert these images and XML files to be compatible with TensorFlow TFRecord for the training process.

To enhance the counting accuracy, we classify the videos into two scenes: one-way or two-way roads and intersections. We also define a Region-of-Observation (ROO) and the direction of movement (DOM) for each video. ROO is represented as a polygon, limiting the space to focus on processing to traffic detection. DOM determines the possible directions of movement of vehicles. Figure 9 illustrates six representative scenes from the dataset with predefined ROO (polygons with a red border) and DOM (possible directions of movement indicated in the form of arrows). A list of points for drawing the ROO area and direction arrows is provided for each video.

4.2. Index of Performance. We evaluated the performance of a model based on the following criteria: accuracy, FPS, memory used, CPU usage, model size, GPU usage, and TPU compatibility. Regarding the detection accuracy, we applied COCO detection metrics officially used in several COCO competitions. This metric is slightly different from Pascal VOC metrics in terms of implementation and additional statistical reports such as mAP (mean Average Precision) for IoU from 0.5 to 0.95 as well as precision/recall statistics for small, medium, and large objects.

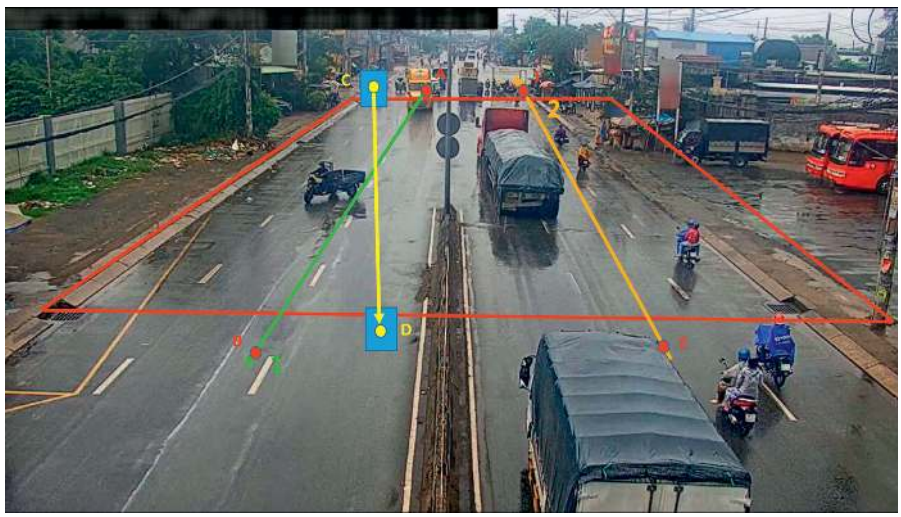

Figure 7: Illustration of calculating cosine similarity score in one-way and two-way road scenarios.

(a) (b)

Figure 8: Illustration of min distance calculation method in intersection scenarios: (a) an intersection with six moving directions labeled from 1 to 6 (POO and predefined DOM) and (b) illustration of the min distances and distance to edges used to count the vehicles moving following direction number 2 (calculating min distance).

(a) (b)

Figure 9: Continued.

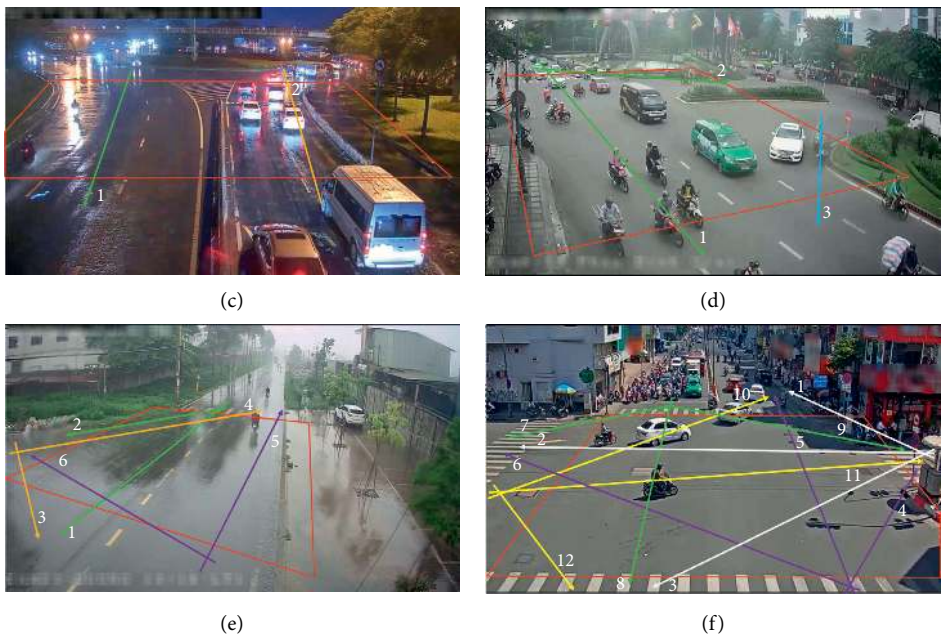

Figure 9: Different scenarios captured from VDD with predefined ROO and DOM. (a) VDD Scene-1, (b) VDD Scene-2, (c) VDD Scene-3, (d) VDD Scene-4, (e) VDD Scene-5, and (f) VDD Scene-6.

For the counting task, we separately evaluated the efficiency of the proposed counting algorithm from the detection process. When a vehicle leaves the ROO region, our system updates and logs the current counting result. The detected vehicle must belong to one of the four selected vehicle types, and the vehicle must move in one of the DOMs. We then summarize the total number of each type of vehicle and their directions when exiting the ROO in each testing video. Let $\hat{y}$ and y be the system's counting result and the actual result from ground truth. Finally, we calculate the error rate according to the following formula:

$$\text{Error} = \frac{|y - \hat{y}|}{y}. \tag{2}$$

4.3. Evaluation Results. The proposed traffic counting system was evaluated in two types of edge devices—first, a Coral Dev Board supported TPU (Section 4.3.1) and then Jetson Nano Board supported GPU (Section 4.3.2). The counting result was also reported at the end of the section.

4.3.1. On Coral Dev Board. We evaluate running models on embedded CPU: quad-core Cortex-A53 @ 1.5 GHz in case of using or without using Edge TPU coprocessor to compare the results in two cases. First, we retain the available models with our datasets until the models were converged. By checking the weight file saved during training, we select the highest accuracy for each model. For compatibility with the Edge TPU, all models are quantized to the nearest 8-bit fixed-point numbers using the TensorFlow Model Optimization Toolkit [65]. The first two models are quantized using the quantization-aware training technique, and the following four models are applied using the full-integer posttraining quantization method. These models are then compiled with the Edge TPU compiler to run on Edge TPU [66]. Next, we run the evaluation models on our traffic videos (1280×720 resolution) and obtain the results presented in Table 1. We can see that, after being converted to an 8-bit fully quantized TFLite model, the model size has been significantly reduced (2 to 4 times depending on the model) compared to the original size. The model size is reported about 3–5 Mb to be suitable for deploying on embedded devices. Among the models, the smallest model after being converted is the SSD MobileDet 320×320 with 3.3 Mb (TFLite model) and 4.31 Mb (TFLite model compatible with Edge TPU).

As shown from the evaluation, many models achieve high accuracy, such as SSD MobileDet 320×320, SSD MobileNet V2 FPNLite 320×320, SSD MobileNet V2 FPNLite 640×640, and SSD MobileNet V1 FPN 640×640 with mean Average Precision mAP@0.5 out of 0.9. The highest and lowest detection accuracies are reported to be 0.963 and 0.768 at models SSD MobileNet V1 FPN 640×640 and SSD MobileNet V1 300×300, respectively. The other important factor in evaluating a Traffic AI camera's efficiency in real time is frames per second (FPS). Our experiments show that the models running on Edge TPU have a much higher speed than the ones running on embedded CPU only. SSD MobileNet V1 300×300, SSD MobileDet 320×320, and SSD MobileNet V2 320×320 have the highest detection speed with 27.5, 26.8, and 25 FPS, respectively. These figures demonstrate that these models are suitable for real-time applications. SSD MobileNet V2 FPNLite 320×320 and SSD MobileNet V2 FPNLite 640×640 give

Table 1: Experimental results about running models on Coral Dev Board.

Model	Model size (Mb)		Accuracy		FPS		Memory (%)	CPU usage (%)	
	TFLite	Edge TPU TFLite	mAP@0.5 : 0.95	mAP@ 0.5	TFLite	Edge TPU TFLite	TFLite and Edge TPU TFLite (%)	TFLite	Edge TPU TFLite
SSD MobileNet V1 300×300	5.37	5.75	0.445	0.768	2.6	27.5	17.8	26%	34.7%
SSD MobileDet 320×320	**3.33**	**4.31**	**0.621**	**0.921**	**1.8**	**26.8**	**19.7**	**25.8%**	**33.3%**
SSD MobileNet V2 320×320	4.58	5.18	0.462	0.794	3.2	25	18.3	26.2%	32.1%
SSD MobileNet V2 FPNLite 320×320	3.68	3.88	0.623	0.906	2.45	7.9	18	26.9%	26.7%
SSD MobileNet V2 FPNLite 640×640	4.26	4.74	0.754	0.951	0.63	2.3	19.1	26.4%	24.8%
SSD MobileNet V1 FPN 640×640	30	30.7	0.781	0.963	0.07	0.08	26.8	25.2%	25.1%

not much higher FPS when running on Edge TPU (7.9 and 2.3 compared to 2.45 and 0.63). SSD MobileNet V1 FPN 640×640 almost shows no difference between Edge TPU and embedded CPU with low FPS of only 0.08 when running on Edge TPU. By deeper analysis, we figure out that, after converting to the TFLite model and compiling to run on Edge TPU, not all models are compatible with operations supported on TFLite [67] and Edge TPU [68].

In case the models could not satisfy all the requirements to run on Edge TPU, they could be further compiled to enhance the performance. At the first point in the model diagram where the unsupported operation occurs, the compiler splits the chart into two parts. The first part of the diagram that contains only supported operations is compiled into a custom operation that executes on Edge TPU, and the others are performed on CPU. Currently, the Edge TPU compiler cannot partition the model more than once. When an unsupported operation happens, the operations and the following tasks are executed on the CPU regardless of the subsequent supported operations. That means the percentage of operations performed on Edge TPU and its order may affect the model's inference time running on the Coral Dev Board. When the models are executed on the board, memory and CPU usage are stable. Average consumed memory ranges from 17% to 27%, and CPU usage is below 35% when running inference on Coral Dev Board (memory: 1 GB, CPU: quad-core Cortex-A53, Cortex M4F). During the inference phase, the average CPU temperature fluctuates around 55 degrees Celsius, and the temperature of TPU remains around 41 degrees Celsius.

From our evaluation results, we highly recommend using the SSD MobileDet 320×320 model in the vehicle detection context. Although this model does not give the highest accuracy, it provides the highest performance with an accuracy of 92.1% and the average inference time of 37,313 ms (equivalent to average FPS of 26.8). Initially, we retrain the SSD MobileDet model at about 150000 steps. During the training, we use TensorBoard to observe the changes in the loss and the accuracy of each stage's weight on the test set. At about 100000 steps, the model tends to converge, mAP@0.5 peaks at 92.1% and then stops increasing, and total loss at this step reaches 0.312, as shown in Figures 10 and 11 . We extract the saved weight file at step 100000, and then we convert the model to a TFLite format before compiling it to Edge TPU.

4.3.2. On Jetson Nano. We retrained the models SSD MobileNet V2, SSD Inception V2, YOLOv4, Tiny YOLOv4, and YOLOv5-small with our datasets and obtained parameters (mAP) after training. Unlike Coral Dev Board that performs inference in INT8 precision, Jetson Nano uses FP16 or FP32 precision. As a result, there is no need for the quantization step for the inference. However, these models should be optimized and restructured using TensorRT for high throughput, low latency, and a low device-memory footprint. The workflow consists of the following steps: we convert the TensorFlow model to the UFF format and the Darknet model to the ONNX format, which is compatible with the TensorRT engine. We reported several performance parameters of models (e.g., throughput, accuracy, memory usage, and CPU-GPU) in Table 2.

According to Table 2, at mAP@0.5, YOLOv4 (0.939), YOLOv5-small (0.937), and Tiny YOLOv4 (0.899) are among the most accurate models, but their FPS are lower than those of SSD MobileNet V2 and SSD Inception V2. To achieve the model's highest mAP, we train each model with the distinctive number of loops due to their architectures' differences. In particular, the models based on Darknet's framework are trained between 5 and 7 hours. Meanwhile, TensorFlow models take 10 to 12 hours to reach the most accurate mAP. During the training for YOLO's models, it is crucial to adjust the parameters such as batch size, subdivisions, and max batches to be compatible with the datasets. We print out each model's FPS and recognize that SSD MobileNet V2, SSD Inception V2, and Tiny YOLOv4 are groups of models with much higher FPS and reported about 24 FPS, 19 FPS, and 20.04 FPS, respectively. We then visualize all stats relevant to throughput and mAP before concluding that Tiny YOLOv4 and SSD Inception V2 models are the most suitable solutions.

It is noticeable from Figure 12 that the models associated with the high level of accuracy hardly linked with equivalent

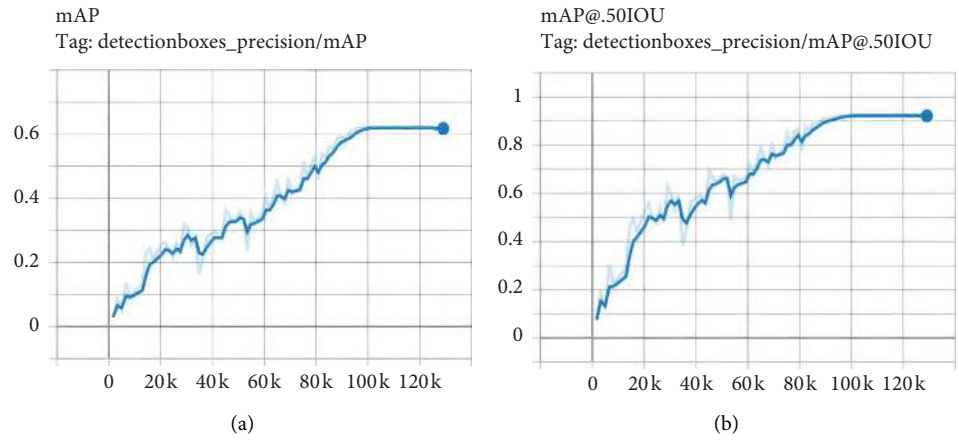

Figure 10: Accuracy of SSD MobileDet model during training. (a) mAP@0.5 : 0.95; (b) mAP@0.5.

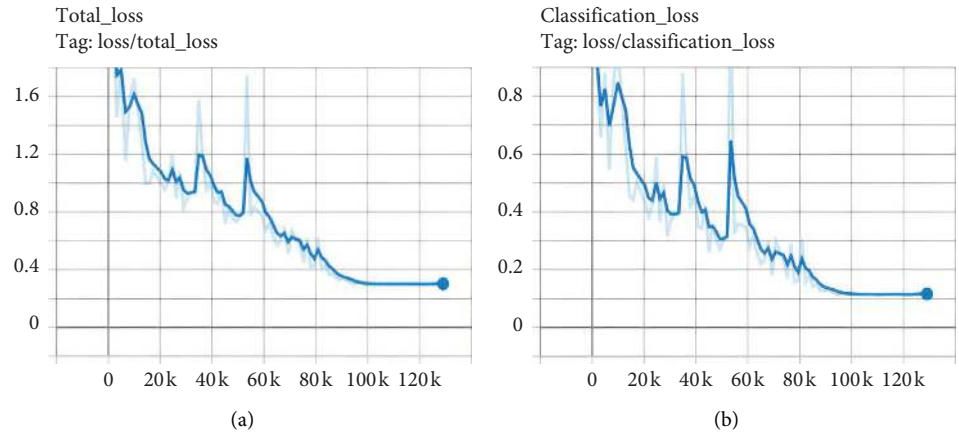

Figure 11: SSD MobileDet training loss curve. (a) Total loss. (b) Classification loss.

Table 2: Experimental results about running models on Jetson Nano.

#	Model	Framework	Size (MB)	mAP@0.5	mAP@0.5:0.95	MEM (GB)	FPS	CPU (%)	GPU (%)
1	YOLOv4 (416 × 416)	Darknet	296.2	0.939	0.694	2.516	4.09	27.6	99
2	**Tiny YOLOv4 (416 × 416)**	**Darknet**	**32.7**	**0.899**	**0.619**	**1.905**	**20.04**	**23.3**	**88**
3	YOLOv5-small (640 × 640)	Darknet	14.5	0.937	0.632	3.1	4.02	58.8	87
4	SSD MobileNet V2 (300 × 300)	TensorFlow	67.8	0.796	0.449	2.217	24	30.1	94
5	**SSD Inception V2 (300 × 300)**	**TensorFlow**	**100.2**	**0.840**	**0.481**	**2.043**	**19**	**33.2**	**99**

height in FPS; on the other hand, the models involved with relatively low accuracy level ended up in high FPS. In both CPU and GPU modes, the obtained values are not too different. Discovering the proper model for vehicle detection is about finding the appropriate trade-off between accuracy and performance, satisfying the requirements. In Figure 12, among the various models investigated in this research, Tiny YOLOv4 and SSD Inception V2 are the best trade-offs between mAP and throughput (FPS).

4.3.3. Counting Result. We run the proposed counting algorithm on the preprepared testing videos. Figure 13 is a screenshot of Camera 06 belonging to the set of testing videos, and counting results are shown in Table 3. Based on the results, the error parameter has an absolute value stretching from 6.4% to 25%. Generally, we found that the larger the sample volume, the lower the absolute number of the error. For example, in direction 1, the vehicle numbers were counted as 16, 165, 3, and 35. The errors detected using the experimental counting algorithm were 20%, 6.4%, 25%, and 14.6%, respectively. The correlation between sample volumes and error rate suggests that our counting accuracy could be increased when the sample volume is large enough.

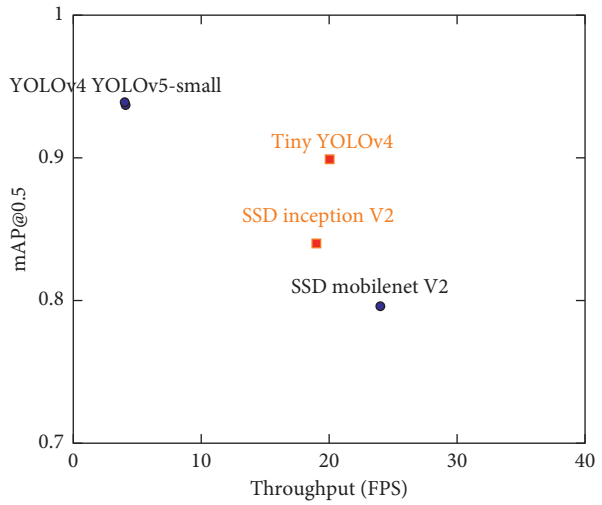

Figure 12: The mean average precision of evaluated models across throughput.

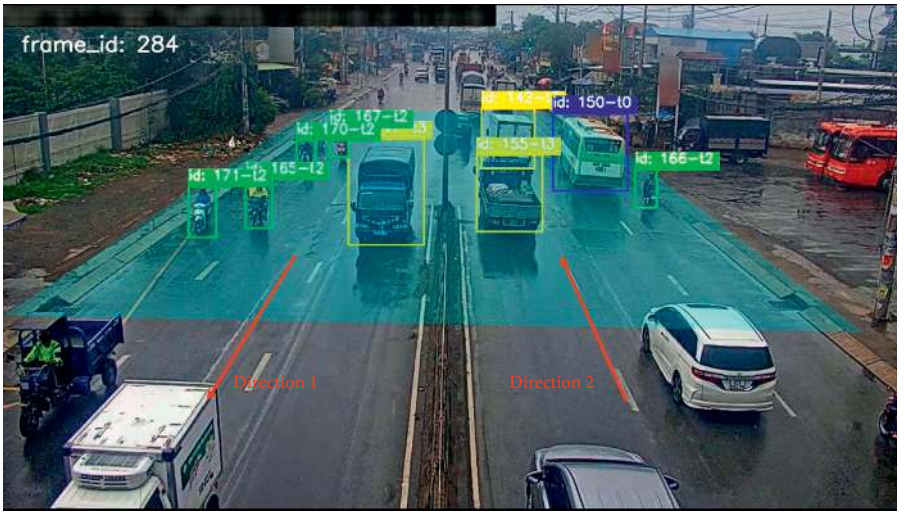

Figure 13: Tracking result running on board of Camera 06 belonging to the set of testing videos.

Table 3: Evaluating experimental traffic counting result on Camera 06.

Type of vehicle	Direction 1			Direction 2		
	Experimental	Actual	Error (%)	Experimental	Actual	Error (%)
Type 1 (car, taxi,. . ..)	16	20	−20	25	31	−19.4
Type 2 (motorcycle, bicycle,. . .)	165	155	6.4	109	121	−11.6
Type 3 (bus, coach bus,. . .)	3	4	−25	4	5	−20
Type 4 (truck, container,. . .)	35	41	−14.6	31	38	−18.4

5. Conclusion

In this study, we introduced an edge-based traffic counting system. The performance evaluation of various deep learning object detection models, which directly perform real-time vehicle detection in Jetson Nano and Coral Dev Board, is also analyzed and discussed. The evaluation results cover several performance indices (e.g., model size, accuracy, FPS, memory, and CPU usage) that are useful for selecting minimum hardware cost to deploy a traffic counting application to edge devices. Moreover, a counting algorithm and optimization methods are adopted to reduce model size while maintaining high detection accuracy. The experiment results show that SSD MobileDet running on Coral Dev Board gives the best correlation between accuracy (92.1%) and inference time (37,313 ms) compared to the other models, but Jetson Nano is more popular, compatible with more types of architect models, and also cheaper than Coral Dev Board ($99 versus $129.99 of Coral Dev). Our future work will focus on further optimizing the vehicle detection model while maintaining low inference time. This could be done by employing novel object detection methods, such as

vision transformer architecture. We also plan to include layering plate identification on the top of current vehicle detection to recognize the vehicle identity that is very useful for traffic management features.

Data Availability

The captured traffic video data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research was funded by Vietnam National University Ho Chi Minh City (VNU-HCM) under Grant no. DSC2021-26-04.

References

[1] Y. Lin, P. Wang, and M. Ma, "Intelligent transportation system (its): concept, challenge and opportunity," in *Proceedings of the 2017 IEEE 3rd International Conference on Big Data Security on Cloud (Bigdatasecurity), IEEE International Conference on High Performance and Smart Computing (Hpsc), and IEEE International Conference on Intelligent Data and Security (Ids)*, pp. 167–172, IEEE, Beijing, China, May 2017.

[2] M. A. Khan, S. Abbas, Z. Hasan, and A. Fatima, "Intelligent transportation system (ITS) for smart-cities using Mamdani Fuzzy inference system," *International Journal of Advanced Computer Science and Applications*, vol. 19, no. 2, 2018.

[3] B. B. Gupta and M. Quamara, "An overview of internet of things (IoT): architectural aspects, challenges, and protocols," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 21, 2018.

[4] S. H. Shah and I. Yaqoob, "A survey: internet of things (IOT) technologies, applications and challenges," in *Proceedings of the 2016 IEEE Smart Energy Grid Engineering (SEGE)*, pp. 381–385, IEEE, Oshawa, Canada, August 2016.

[5] A. Sumalee and H. W. Ho, "Smarter and more connected: future intelligent transportation system," *Iatss Research*, vol. 42, no. 2, pp. 67–71, 2018.

[6] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep learning for IoT big data and streaming analytics: a survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 2923–2960, 2018.

[7] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic, "Deep learning applications and challenges in big data analytics," *Journal of Big Data*, vol. 2, no. 1, pp. 1–21, 2015.

[8] Y. Ai, M. Peng, and K. Zhang, "Edge computing technologies for internet of things: a primer," *Digital Communications and Networks*, vol. 4, no. 2, pp. 77–86, 2018.

[9] N. Kumar, V. Poonia, B. B. Gupta, and M. K. Goyal, "A novel framework for risk assessment and resilience of critical infrastructure towards climate change," *Technological Forecasting and Social Change*, vol. 165, p. 120532, 2021.

[10] D. Li, L. Deng, B. Bhooshan Gupta, H. Wang, and C. Choi, "A novel CNN based security guaranteed image watermarking generation scenario for smart city applications," *Information Sciences*, vol. 479, pp. 432–447, 2019.

[11] A. Hanif, A. B. Mansoor, and A. S. Imran, "Performance analysis of vehicle detection techniques: a concise survey," in *Trends and Advances in Intelligent Systems and Computing*, pp. 491–500, Springer, Berlin, Germany, 2018.

[12] V. Lempitsky and A. Zisserman, "Learning to count objects in images," in *Proceedings of the 23rd International Conference on Neural Information Processing Systems-Volume 1*, pp. 1324–1332, Vancouver, Canada, December 2010.

[13] L. Fiaschia, U. Koethe, R. Nair, and F. A. Hamprecht, "Learning to count with regression forest and structured labels," in *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, Tsukuba, Japan, November 2012.

[14] M. Liang, X. Huang, C.-H. Chen, X. Chen, and A. Tokuta, "Counting and classification of highway vehicles by regression analysis," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 5, pp. 2878–2888, 2015.

[15] X. Liu, Z. Wang, J. Feng, and H. Xi, "Highway vehicle counting in compressed domain," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3016–3024, Las Vegas, NV, USA, June 2016.

[16] Z. Wang, X. Liu, J. Feng, J. Yang, and H. Xi, "Compressed-domain highway vehicle counting by spatial and temporal regression," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 29, 2019.

[17] H. Tayara, K. G. Soo, and K. T. Chong, "Vehicle detection and counting in high-resolution aerial images using convolutional regression neural network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3016–3024, Las Vegas, NV, USA, June 2016.

[18] S. Zhang, G. Wu, J. P. Costeira, and J. M. F. Moura, "FCN-rLSTM: deep spatio-temporal neural networks for vehicle counting in city cameras," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Venice, Italy, October 2017.

[19] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[20] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: speeded up robust features," in *Proceedings of the Computer Vision-ECCV 2006*, pp. 404–417, Springer, Graz, Austria, May 2006.

[21] H. Tayara, K. G. Soo, and K. T. Chong, "Histograms of oriented gradients for human detection," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, San Diego, CA, USA, June 2005.

[22] R. Lienhart and J. Maydt, "An extended set of Haar-like features for rapid object detection," in *Proceedings of the International Conference on Image Processing*, pp. 900–903, Rochester, NY, USA, September 2002.

[23] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 580–587, Columbus, OH, USA, June 2014.

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 9, pp. 1904–1916, 2015.

[25] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1440–1448, Santiago, Chile, December 2015.

[26] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, pp. 1137–1149, 2016.

[27] Z. Dai, H. Song, X. Wang et al., "Video-based vehicle counting framework," *IEEE Access*, vol. 7, pp. 64460–64470, 2019.

[28] W. Liu, D. Anguelov, D. Erhan et al., "SSD: single shot multiBox detector," in *Proceedings of the Computer Vision-ECCV 2016*, pp. 21–37, Springer, Amsterdam, The Netherlands, October 2016.

[29] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 779–788, Las Vegas, NV, USA, June 2016.

[30] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7263–7271, Honolulu, HI, USA, July 2017.

[31] J. Redmon and A. Farhadi, "YOLOv3: an incremental improvement," in *Proceedings of the Computer Vision and Pattern Recognition*, Salt Lake City, UT, USA, June 2018.

[32] A. Bochkovskiy, C. Y. Wang, and H. Y. M. Liao, "YOLOv4: optimal speed and accuracy of object detection," 2020, https://arxiv.org/abs/2004.10934.

[33] G. Jocher, K. Nishimura, T. Mineeva, and R. Vilariño, YOLOv5, 2020.

[34] A. Dame and E. Marchand, "Optimal detection and tracking of feature points using mutual information," in *Proceedings of the 16th IEEE International Conference on Image Processing (ICIP)*, pp. 3601–3604, Cairo, Egypt, November 2009.

[35] C. Choi, T. Wang, C. Esposito, B. B. Gupta, and K. Lee, "Sensored semantic annotation for traffic control based on knowledge inference in video," *IEEE Sensors Journal*, vol. 21, p. 1, 2021.

[36] R. Zhao and X. Wang, "Counting vehicles from semantic regions," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 2, pp. 1016–1022, 2013.

[37] R. Ke, Z. Li, S. Kim, J. Ash, Z. Cui, and Y. Wang, "Real-time bidirectional traffic flow parameter estimation from aerial videos," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 4, 2016.

[38] U. Gopikrishnan and R. Jose, "DriveCare: a real-time vision based driver drowsiness detection using multiple convolutional neural networks with kernelized correlation filters (MCNN-KCF)," in *Proceedings of the 2020 IEEE 5th International Conference on Computing Communication and Automation (ICCCA)*, Greater Noida, India, October 2020.

[39] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," pp. 3464–3468, 2016, https://arxiv.org/abs/1602.00763.

[40] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," pp. 3645–3649, 2017, https://arxiv.org/abs/1703.07402.

[41] R. Ke, Z. Li, J. Tang, Z. Pan, and Y. Wang, "Real-time traffic flow parameter estimation from UAV video based on ensemble classifier and optical flow," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, pp. 54–64, 2018.

[42] Z. Gao, R. Zhai, P. Wang et al., "Synergizing appearance and motion with low rank representation for vehicle counting and traffic flow analysis," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, pp. 2675–2685, 2017.

[43] F. Ali, A. Ali, M. Imran, R. A. Naqvi, M. H. Siddiqi, and K.-S. Kwak, "Traffic accident detection and condition analysis based on social networking data," *Accident Analysis & Prevention*, vol. 151, p. 105973, 2021.

[44] F. Ali, D. Kwak, P. Khan et al., "Transportation sentiment analysis using word embedding and ontology-based topic modeling," *Knowledge-Based Systems*, vol. 174, pp. 27–42, 2019.

[45] C. L. Stergiou, K. E. Psannis, and B. B. Gupta, "IoT-based big data secure management in the fog over a 6G wireless network," *IEEE Internet of Things Journal*, vol. 8, no. 7, 2020.

[46] Y. Jararweh, M. Alsmirat, M. Al-Ayyoub et al., "Software-defined system support for enabling ubiquitous mobile edge computing," *The Computer Journal*, vol. 60, no. 10, pp. 1443–1457, 2017.

[47] F. Mirsadeghi, M. K. Rafsanjani, and B. B. Gupta, "A trust infrastructure based authentication method for clustered vehicular ad hoc networks," *Peer-to-Peer Networking and Applications*, vol. 13, pp. 1–17, 2020.

[48] H. Fatemidokht, M. K. Rafsanjani, B. B. Gupta, and C.-H. Hsu, "Efficient and secure routing protocol based on artificial intelligence algorithms with UAV-assisted for vehicular ad hoc networks in intelligent transportation systems," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–13, 2021.

[49] C. Esposito, M. Ficco, and B. B. Gupta, "Blockchain-based authentication and authorization for smart city applications," *Information Processing & Management*, vol. 58, no. 2, p. 102468, 2021.

[50] Mamta, B. B. Gupta, K. -C. Li, V. C. M. Leung, K. E. Psannis, and S. Yamaguchi, "Blockchain-assisted secure fine-grained searchable encryption for a cloud-based healthcare cyber-physical system," *IEEE/CAA Journal of Automatica Sinica*, 2021.

[51] A. Al-Qerem, M. Alauthman, A. Almomani, and B. B. Gupta, "IoT transaction processing through cooperative concurrency control on fog-cloud computing environment," *Soft Computing*, vol. 24, no. 8, pp. 5695–5711, 2020.

[52] L. T. De Paolis, V. De Luca, and R. Paiano, "Sensor data collection and analytics with thingsboard and spark streaming," in *Proceedings of the 2018 IEEE Workshop on Environmental, Energy, and Structural Monitoring Systems (EESMS)*, pp. 1–6, IEEE, Salerno, Italy, June 2018.

[53] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning," in *Artificial Neural Networks and Machine Learning*, pp. 270–279, Springer, Cham, Switzerland, 2018.

[54] Tensorflow, "Tensorflow 2 detection model Zoo," 2020, https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md.

[55] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: a whitepaper," 2018, https://arxiv.org/abs/1806.08342.

[56] H. Wu, P. Judd, X. Zhang, M. Isaev, and P. Micikevicius, "Integer quantization for deep learning inference: principles and empirical evaluation," 2020, https://arxiv.org/abs/2004.09602.

[57] T. Y. Ross and G. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2980–2988, Honolulu, HI, USA, July 2017.

[58] Z. Jiang, L. Zhao, S. Li, and Y. Jia, "Real-time object detection method based on improved YOLOv4-tiny," 2020, https://arxiv.org/abs/2011.04244.

[59] S. Arabi, A. Haghighat, and A. Sharma, "A deep learning based solution for construction equipment detection: from

development to deployment," 2019, https://arxiv.org/abs/1904.09021.

[60] A. G. Howard, M. Zhu, B. Chen et al., "Efficient convolutional neural networks for mobile vision applications," 2017, https://arxiv.org/abs/1704.04861.

[61] A. Howard, A. Zhmoginov, L. C. Chen, M. Sandler, and M. Zhu, "Inverted residuals and linear bottlenecks: mobile networks for classification, detection and segmentation," 2018, https://arxiv.org/abs/1801.04381.

[62] Y. Xiong, H. Liu, S. Gupta et al., "Mobiledets: searching for object detection architectures for mobile accelerators," 2020, https://arxiv.org/abs/2004.14525.

[63] X. Hou, Y. Wang, and L. P. Chau, "Vehicle tracking using deep SORT with low confidence track filtering," in *Proceedings of the 2019 16th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pp. 1–6, IEEE, Taipei, Taiwan, September 2019.

[64] "AI challenge Hochiminh City example," 2020, https://github.com/hcmcaic/ai-challenge-2020.

[65] Tensorflow, "TensorFlow model optimization," 2021, https://www.tensorflow.org/model_optimization/guide.

[66] Google, "Edge TPU compiler," Google LLC, Mountain View, CA, USA, 2020, https://coral.ai/docs/edgetpu/compiler/#system-requirements.

[67] Tensorflow, "TensorFlow lite and TensorFlow operator compatibility," 2021, https://www.tensorflow.org/lite/guide/ops_compatibility.

[68] Google, "All operations supported by the Edge TPU and any known limitations," Google LLC, Mountain View, CA, USA, 2020, https://coral.ai/docs/edgetpu/models-intro/#supported-operations.