Deep Learning Model on Embedded Board for Vehicle Detection and Vehicle Tracking

Luong Thanh Tra, Nguyen Minh Nguyen, Jongtae Lim,
Hyungsik Shin, Seongwon Cho

Abstract

This paper proposes a deep learning model to detect and track the vehicle on an embedded device such as Odroid, Orange Pi, etc. This system includes two main parts: vehicle detection and vehicle tracking. Since deep learning has achieved high accuracy over the classical image processing method, object detectors can detect vehicles in the street and highway. It can be normal to run the computer detection program with graphic processor unit (GPU) support, but it is challenging to run it on the embedded board with no GPU support and low central processing unit (CPU) performance. This paper focuses on balancing edge-computing-based deep learning object detection's accuracy and performance using additional techniques such as quantization, edge TPU, and multiple threads. SSDLite with MobileNet backbone is chosen due to its lighter than other networks but still obtain good performance compare with Yolo.

Keywords : Optimize model| Vehicle detection| Vehicle tracking| Embedded device| SSDLite| MobileNet

I. Introduction

When Vehicle detection is a very critical part of traffic surveillance. Vehicle recognition and localization in route observance video scenes are of tidy significance to intelligent traffic management and control. The old way to detect vehicles by using computer vision algorithms has achieved outstanding performance. However, many noises make that program worse, such as background clutter, light shadow, and vehicle speed. With the trendy installation of closed-circuit television (CCTV) cameras, enormous traffic video footage has been collected for analysis. Typically, we can get more data of a more distant paved road at a high viewing angle. We can now train the deep learning model for detecting vehicles because of that big data, which can reduce traffic jams, accidents or control the highway's flow. In the past, because of the lack powerful embedded board, we must build a server to process the input footage from the CCTV camera, and it can take money to do that. Therefore, if an embedded board can detect or track vehicles, it can reduce server costs and the time of data processing. We can say that the embedded systems are getting better and better every moment by looking at hardware, software, and methods. Thanks to nanotechnology, the power of the embedded board can be seen as a more miniature computer. Therefore, like a

* This research was supported by Ministry of SMEs and Startups(RS-2024-00448168, RS-2024-00434571), Ministry of the Interior and Safety(RS-2024-00461780).

Manuscript : 2024. 12. 30
Confirmation of Publication:2025.01.20
Corresponding Author : Seongwon Cho e-mail: swcho@hongik.ac.kr

computer, it can run as multiple threads and cut down the computational time. There are many commercial embedded boards such as Odroid, Raspberry pi, Jetson. The proposed method used in this paper is to choose a "matching" deep learning model for deploying the whole system on an embedded device. The system uses images or video sequences as input and giving out the vehicle object and the tracking id as outputs.

II. Related Research

The primary purpose of this step is to detect and track vehicles from the input frame or image. There are many object detection algorithms and object tracking algorithms that we can apply to this problem. However, because we have to deploy into the embedded device that is less powerful than PC, the "correct" algorithm has to be chosen carefully.

2.1. Object Detection

In the past, a system used to detect an object have to take a classifier for that specific object and evaluate it at multiple separate areas and ratios in the evaluate image. For example, the R-CNN model uses region proposal algorithms to process multiple tasks before showing the predicted detection. Firstly, the methods generate all of the possible bounding boxes in a picture and run a classifier. After getting the classification, post-processing is accustomed to cleaning the predicted bounding boxes, erasing duplicate predictions, and recalculating scores on other objects in the image. As we can see, these pipelines are very complicated. Therefore, it takes much latency and is tough to optimize because every unit must be trained individually.

YOLO [1] (you only look once) algorithm is extremely straightforward compared to the R-CNN model. The figure below explains the essential step of this model. YOLO is a single and straightforward convolutional neural network that predicts the multiple bounding boxes and class probability. Follow the step-by-step in the figure above, the system reshapes an image to the fixed input size such as 240x240, runs end-to-end CNN on that image, and finally sets the threshold for the predicted results. Yolo can train the whole image, so we do not need to cut out the bounding boxes of objects to feed into the model.

Moreover, because of the single CNN, this model can straight optimize detection performance. The YOLO model or the unified model proved that it has many advantages compared with the traditional methods. However, it slows down the system and does not suitable for an embedded device.

Despite the state-of-the-art hardware, it is kind of too slow to reach real-time detection projects. SSD [2] does not resample features or pixels for the hypotheses bounding box and keeps the accuracy the same to increase the speed. The essential enhancement for improving the speed derives from getting rid of the proposal bounding box and consecutive features or pixels resampling phase. SSD improvements involve three main things. Firstly, a small convolutional filter is used to predict classes and offset locations of the bounding box. Secondly, multiple

independent filters or predictors are used to detect multiple aspect ratios. Finally, the SSD model applies these predictors to multiple feature maps in the later phases to detect at multiple scales.

The lightweight MobileNet [3–5] model is built from a streamlined deep neural network architecture. This deep neural network model uses two hyper-parameters which are width multiplier and resolution multiplier to set up tiny and low latency models. Based on these ideas, this model can deploy into embedded systems and mobile devices. The prior works show that we must use the smaller model by building a model suitable for resource restrictions such as size, latency, and computation. While the prior works focus on the size only and do not care about speed, MobileNet chooses to shrink the network's width.

Based on the related work above, we are sure that using the YOLO algorithm is unsuitable for embedded and mobile devices. Therefore, using the SSD algorithm and MobileNet algorithm is more sufficient for the embedded system, which this thesis mainly focuses on. SSDLite is the lite version of the SSD model, a mobile-friendly alternative to regular SSD. This lite model replaces all regular convolution with depthwise separable convolutions in the SSD prediction layers. This design model seems to be suitable for the design of MobileNet, and it takes less computational cost than usual. SSDLite replaces many heavy layers, so the parameters and latency reduce significantly, as shown Table 1.

The experiment also illustrates that the MobileNet + SSDLite is more suitable for the embedded device than ordering algorithms such as YOLO and SSD because of the low latency and parameters. Based on the table information, we choose the SSDLite MobileNet (figure 1) as our backbone because it is ten times smaller and twenty times more efficient whistle still got the same mAP as the YOLOv2 COCO dataset.

Table 1. The comparison of the parameters and latency between SSD and SSDLite(Madds stands for Multiply-Add operations, representing the cumulative count of multiply-and-add operations)

	Params	Madds
SSD	14.8M	1.25B
SSDLITE	**2.1M**	**0.35B**

2.2. Object Tracking

Object tracking is an algorithm to track single or multiple objects overtimes in the input frame sequence. This algorithm is more profound than object detection because we need to care about image (video) sequences instead of a single one. We cannot easily split the frame from video and process it like a single image because we must focus on more features and noises besides detecting the bounding box: the ID of an object must remain the same through frames, when the object is overlapped or disappear in some frame, the system can still detect the correct ID of that object, and the problem about the speed in the realtime project. Object tracking can be divided into two primary approaches: single object tracking and multiple object tracking.

• Single object tracking (SOT) focuses on detecting only one object in the entire video. Because it tracks only one object, we need to provide the initial bounding box of that object.

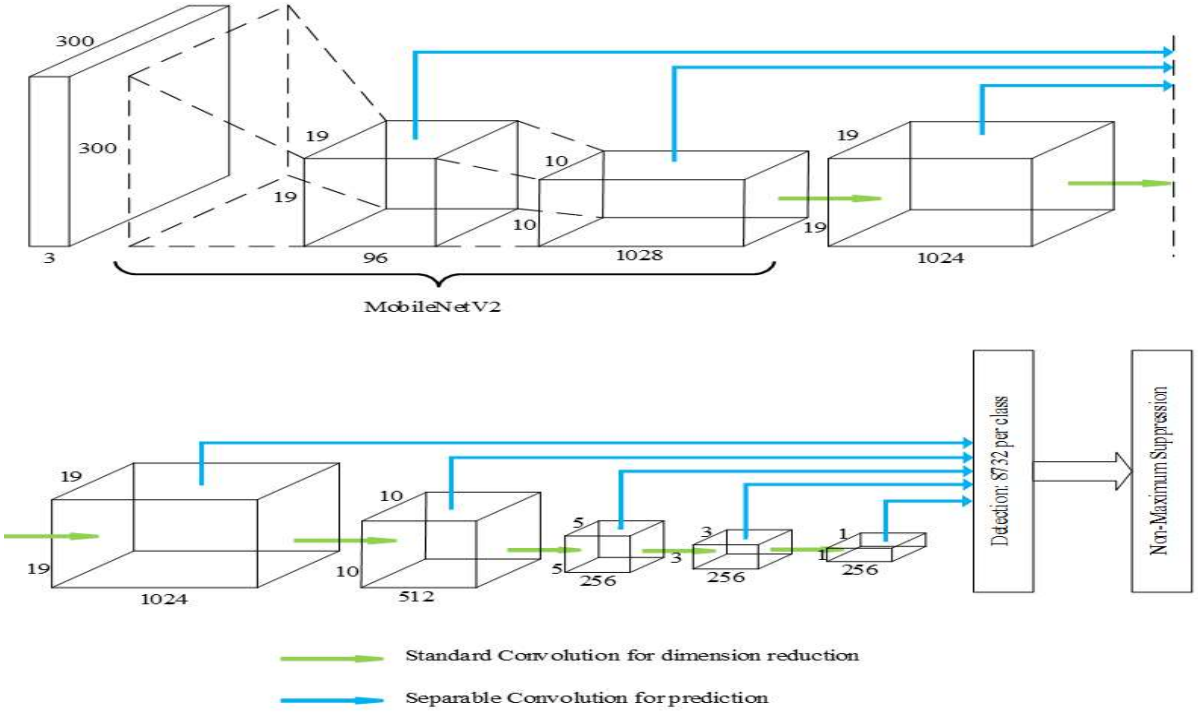

Fig. 1. SSDLiteMobileNetV2 architecture

• Multiple object tracking (MOT) tracks multiple objects that appear in the video, including new objects. MOT is more complicated than SOT and receives more notice from researchers.

The centroid tracking algorithm is built from the OpenCV library for object tracking. Besides, object tracking has three processes which are:

• Getting an initial set of a detected object which can be an input of detected bounding box coordinates

• With each initial detection, mark a unique ID for them

• Tracking objects moving in frames and remain the assignment of unique IDs

By applying a unique ID for each object, we can count them in the sequence. This type of algorithm is fundamental to order image processing or computer vision algorithm. The centroid tracking is based on the Euclidean distance between two centroids of the existing objects and the new detected object's centroids in the subsequent frames.

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \qquad (1)$$

where:

p, q: two points in Euclidean n−space

q_i, p_i: Euclidean vectors, starting from an initial point

n: number of spaces, n−space

However, there are three main drawbacks of this centroid tracking algorithm.

The first problem is that this approach needs to be run on every frame. There is no issue when running the tracking algorithms on every frame if we use fast object detectors such as Haar cascades or color thresholding. However, the processing pipeline speed significantly slows down if we use a deep learning object detector neural network such as

YOLO, SSD on a resource-constrained system.

The second problem is that the centroid tracking algorithm assumes that a unique object's centroid must lie close together from the frame by frame. This assumption only works when we represent a 3D world with 2D frames, which means it is impossible when objects overlap. In that case, the object ID may switch from one to another.

The third problem, which is mainly related to the vehicle tracking project, is its speed. If the behind and also the front vehicle accelerate speed, so in the subsequent, the centroid of the behind vehicle is minimum Euclidean distance compare with the front vehicle.

DeepSORT [7-8] is developed to solve the problem of many ID switches. It uses deep learning neural networks to extract the object's features to increase accuracy in the mapping stage. Moreover, a matching cascade was built to contact the object that disappears in some frames.

In multiple object tracking, especially in tracking-by-detection algorithms, two main factors affect tracking performance:

• Data association focuses on data connection, especially criteria to choose the proper connection of new detection to tracking objects stored

• Track life cycle management focuses on the life cycle of a tracked object that has been stored, including when it initializes the tracking stage, when it stops, and when it deletes the tracked object.

In DeepSORT, data association is solved by the Hungarian algorithm, the same as SORT. However, the connection is based on IOU and focuses on other factors such as distance between new detection coordinates and tracked points and the cosine distance between two feature vectors.

III. Optimization

3.1. Quantization

Quantization is a technique for storing tensors and performing computations at lower bit widths than floating-point accuracy. A quantized model rather than accomplishes some of the tensor's operations with floating-point values executes with integers. This idea approves for a more condensed model representation, and the use performance is higher in many hardware systems by vectorized operations. The model that supports INT8 quantization reduces memory bandwidth requirements and size four times that traditional FP32 models. Quantization is generally a technique to increase the inference's speed. Moreover, quantized operators only support the forward pass.

Quantizing a model means converting all of the floating-point 32 bit (such as activation outputs or weights) to the nearest fixed point 8-bit numbers. Based on this method, the model is much smaller and faster. Even though the 8-bit model can be less accurate, the neural network's inference precise is not reasonably affected. There are two forms of quantized technique which are quantization aware training that quantize weights and activations during model training and post-training quantization that quantize them after model training.

3.2. Edge TPU device

The Edge TPU only supports a fixed set of neural network architectures and operations to afford high inference time with a low-power cost.

Fig. 2. Coral Edge TPU

The Edge TPU can deal with deep feed-forward neural networks like CNN (convolutional neural network). However, it only supports TensorFlow lite models quantized to 8-bit bandwidths and compiled for the Edge TPU specifically. Besides. TensorFlow lite model is a lightweight version that is created for embedded and mobile devices. We can get a low-latency inference in a tiny binary size in which interpreter kernels and the TFLite model are much smaller. So we converted the trained SSDLite MobileNet V2 to TFLite model using post-training quantization.

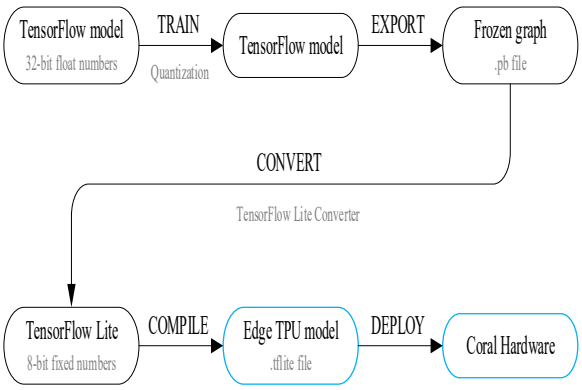

Fig. 3. The Edge TPU's workflow of creating a model

IV. Experimental Results

We trained the vehicles detecting and vehicles tracking model using a computer system equipped with an Intel i9-9900K CPU and an NVIDIA GeForce RTX 2080 Ti GPU. And I used PyTorch version 1.7.0 and Python version 3.8.

4.1. Vehicle detection

MIO-TCD [6] localization dataset includes 137,743 high-resolution pictures at different day times and different months by almost thousands of CCTV cameras. Those pictures have been chosen from a wide range of vehicle detection challenges. This dataset includes one or more foreground objects in 11 labels.

After we combine our lab data and the MIO-TCD data, the data structure is shown in the table below.

Table 2. Our data structure

1500 lab data train	21820 MIO-TCD data train	4372 MIO-TCD data dev	1489 MIO-TCD data test

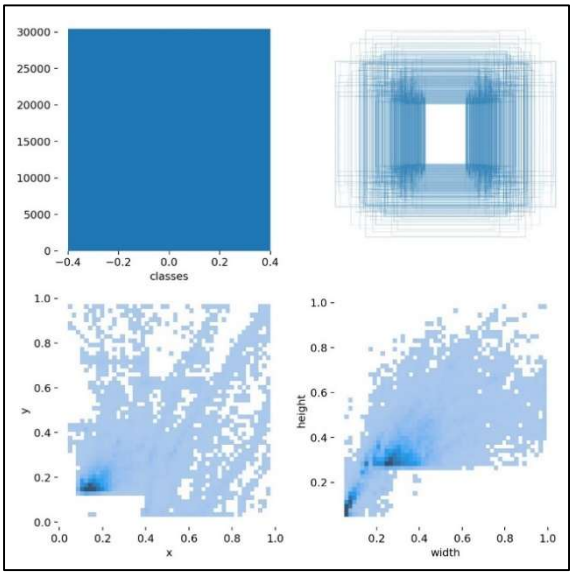

Fig. 4. Vehicle dataset visualization

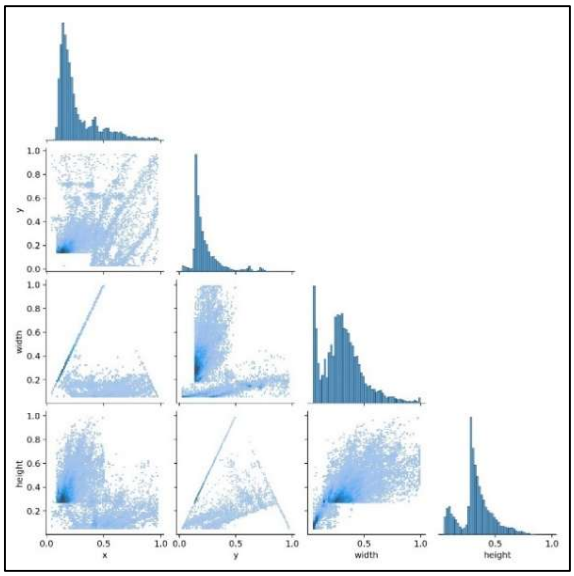

Fig. 5. Labels correlogram

After training SSDMobileNetV2 with our dataset of around 48,000 steps, we got the below results. First of all, the total loss is quite good, which is around 0.9. The second thing that we care about is the mAP value, which is close to 0.88.

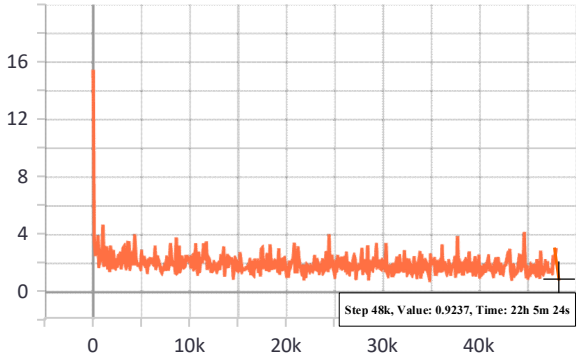

Fig. 6. SSDMobileNet total loss

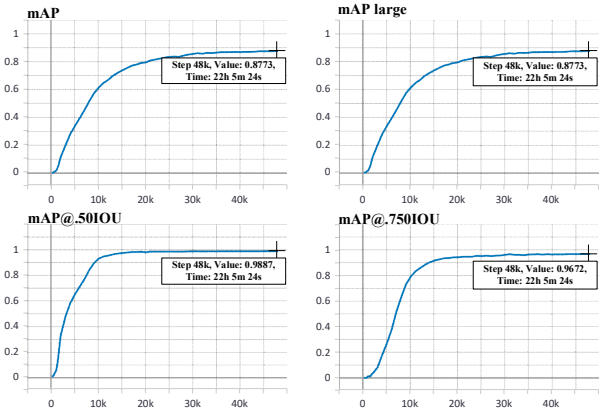

Fig. 7. SSDMobileNet mAP graphs

As mentioned above, we deploy our program on PC and OrangePi3 board to test our model performance and speed.

The table below shows the inference time of the SSDMobileNetV2 model and YOLO model in our PC. When getting the inference time result of these two algorithms, we do not use GPU and multiple CPU threads to detect.

Table 3. SSDMobileNetV2 and YOLO inference time in PC

PC	SSDMobileNetV2	YOLO
Inference time	~0.03s	~0.06s

Base on that inference result, it seems to be that the SSDMobileNetV2 speed is much faster than YOLO without GPU supported, which is not appear in an embedded device. We also run the program on the OrangePi3 board to get the inference time on the real-life embedded board. The comparison of quantized model inference time and the none one is also shown in the table below.

Table 4. SSDMobileNetV2 and YOLO inference time in PC

OrangePi3 board	Inference time with Coral edge TPU
SSDMobileNetV2 without quantization	~1.3s
SSDMobileNetV2 with quantization	~0.06s
YOLO with quantization	~1.0s

4.2. Vehicle Tracking

We used the dataset of the re-identification vehicle for our training called VeRi [9-11]. This comprehensive range dataset is used for semantic analysis of vehicles in real life. The VeRi dataset's

target is to contribute an overall benchmark to validate large-scale computer vision algorithms' performance and facilitate a large-scale of a current research topic related to "vehicles."

The VeRi dataset includes 50,000 images of 776 vehicles in real-world scenes covering a 1.0 km^2 zone in 24 hours. Therefore, this dataset is more scalable for real-life applications. The pictures are captured in a real-world abandoned surveillance scene and labeled with many attributes such as brands, types, bounding boxes, colors. Therefore, the complicated models can be trained and evaluated for vehicle Re-Id. Each vehicle is taken by 2 ~18 CCTV cameras in different resolutions, viewpoints, occlusions, illuminations, which contribute high recurrence percentage in the real-life environment. The dataset is also labeled with spatiotemporal information and satisfactory license plates such as plate bounding boxes, plate numbers, the distances between neighboring cameras, and vehicles' timestamps.

After getting datasets for detecting vehicles, we started to train the model for tracking vehicles called the DeepSORT model. The accuracy is outstanding at 0.9609.

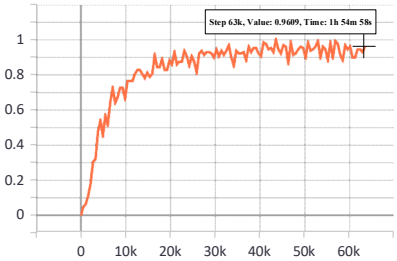

Fig. 8. DeepSort accuracy graph

As we can see, the inference time of tracking vehicles on an embedded device is twice as in a PC. We can see that SSDMobileNetV2+DeepSORT model is faster than YOLO+DeepSORT model.

Table 5. The total time of tracking program on YOLO and SSDMobileNetV2 backbone model

	PC	OrangePi3 board
SSDMobileNetV2+ DeepSORT	~0.07s	~1.7s
YOLO + DeepSORT	0.15s	18s

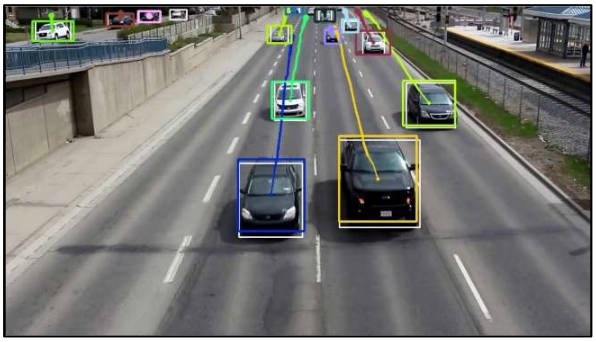

Fig. 9. Vehicle tracking results

V. Conclusion

Vehicle detecting and tracking programs are popular right now, but we must add more techniques to reach a high detecting stage and tracking accuracy. In this thesis, the effective model SSDMobileNetV2 is used to increase inference time on the embedded device. Moreover, the quantization technique is added when training the model suitable for the embedded device and Coral edge TPU to increase inference speed. Besides those advantages, the SSDMobileNetV2 model is not deeper as the current state-of-art detection model, so it is hard to recognize the small object and miss the detection in some frames when it has many vehicles. However, our model outperforms the standard model YOLO in the inference time in both PCs without using GPU and embedded devices.

The DeepSORT model is used to get better performance results than the SORT model's previous version in the tracking

stage. The DeepSORT model trains on the VeRi vehicle dataset can get numerous features to define and track the vehicle, such as color, type, and centroid distance. After testing our program in some highway videos, it is excellent, and it can be applied in a real-life environment without hesitation. However, because the process to get a result from the model is too long in the embedded board, it needs to develop more.

The enhancement of our project is based on two designed models. First, it can be great to reduce the detecting model's depth, leading to a faster model. Moreover, optimize the code to get better results in the inference time is also a solution. Second, we may need to add quantization for the DeepSORT model training stage to deploy this model to the embedded board.

In conclusion, the combination of SSDMobileNet V2 and DeepSORT is a reasonable solution for vehicle detection and tracking on the embedded board.

REFERENCES

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *in Proc. Of IEEE Conf on Computer Vision and Pattern Recognition,* pp. 779-788, May 2016.

[2] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Y. Fu, and A. C. Berg, "Ssd: Single shot multiBox detector," *in Proc. Of 14th European Conf. on Computer Vision-ECCV*, pp. 21-37, Amsterdam, The Netherlans, Oct. 2016.

[3] G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications", arXiv preprint arXiv:1704.04861, Apr. 2017.

[4] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," *in Proc. of IEEE Conf. on Computer Vision and Pattern Recognition,* pp. 4510-4520, Mar. 2019.

[5] Howard, M. Sandler, G. Chu, L. C. Chen, B. Chen, M. Tan and H. Adam, "Searching for mobilenetv3", *in Proc. of IEEE/CVF Conf. on Computer Vision*, pp. 1314-1324, Nov. 2019.

[6] Z. Luo, F. B. Charron, C. Lemaire, J. Konrad, S. Li, A. Mishra, A. Achkar, J. Eichel and P.M Jodoin, "MIO-TCD: A new benchmark dataset for vehicle classification and localization," *IEEE Trans. Image Processing,* vol. 27, no. 10, pp. 5129-5141, Oct. 2018.

[7] Bewley, Z. Ge, L. Ott, F. Ramos and B. Upcroft, "Simple online and realtime tracking," *in Proc. of IEEE Conf. on Image Processing*, pp. 3464-3468, Sep. 2017.

[8] N. Wojke, A. Bewley and D. Paulus, "Simple online and realtime tracking with a deep association metric," *in Proc. of 2017 IEEE Conf. on Image Processing*, pp. 3645-3649, Sep. 2017.

[9] X. Liu, W. Liu, T. Mei and H. Ma, "Provid: Progressive and multimodal vehicle reidentification for large-scale urban surveillance," *IEEE Trans. Multimedia.*, vol. 20, no. 3, pp. 645-658, Mar. 2018.

[10] X. Liu, W. Liu, T. Mei and H. Ma, "A deep learning-based approach to progressive vehicle re-identification for urban surveillance," *in Proc. of 14th European Conf. on Computer Vision*, pp.869-884, Amsterdam, The Netherlands, Oct. 2016.

[11] X. Liu, W. Liu, H. Ma and H. Fu, "Large-scale vehicle re-identification in urban surveillance videos," *in Proc. of IEEE*

Conf. on Multimedia and Expo, pp. 1−6, Jul. 2016.

Authors

Luong Thanh Tra

He received B.S degree from Viet Nam National University Ho Chi Minh City, Vietnam and M.S degree from Hongik University, Korea.

Nguyen Minh Nguyen

He received his B.S. and MS degrees from Ton Duc Thang University and Sejong University, respectively. He is a graduate student of Hongik University Ph.D. program.

Jong Tae Lim

He received his B.S. degree from Seoul National University, Korea, He received his Ph.D degree from University of Michigan, Ann Arbor, MI, USA in 2001. He has been a professor of Hongik University, Seoul, Korea.

Hyungsik Shin

He received his B.S. degree from Seoul National University, Korea in 2003. He received his Ph.D degree from Stanford University, USA in 2001. He has been a professor of Hongik University, Seoul, Korea.

Seongwon Cho

He received his B.S. degree from Seoul National University, Korea in 1982, He received his MS and Ph.D degrees from Purdue University, West Lafayette, Indiana, USA in 1987 and 1992, respectively. He has been a professor of Hongik University, Seoul, Korea.