Smart Media Journal / Vol.13, No.01 / ISSN:2287-1322
https://dx.doi.org/10.30693/SMJ.2023.13.01.24

Parking Lot Occupancy Detection using Deep Learning and Fisheye Camera for AIoT System

To Xuan Dung, Seongwon Cho

Abstract

The combination of Artificial Intelligence and the Internet of Things (AIoT) has gained significant popularity. Deep neural networks (DNNs) have demonstrated remarkable success in various applications. However, deploying complex AI models on embedded boards can pose challenges due to computational limitations and model complexity. This paper presents an AIoT-based system for smart parking lots using edge devices. Our approach involves developing a detection model and a decision tree for occupancy status classification. Specifically, we utilize YOLOv5 for car license plate (LP) detection by verifying the position of the license plate within the parking space.

Keywords : AIoT | YOLOv5| Decision Tree| Fisheye Camera

I. INTRODUCTION

In recent years, the increasing number of vehicles has made an efficient Parking Management System (PMS) essential for large buildings, enabling them to provide real-time information on the availability of parking slots. Traditionally, PMS relies on expensive sensor-based techniques, such as ultrasonic sensors [1], magnetic sensors [2] [3], or a combination of both [4] [5], mounted on each parking slot to detect the presence of a vehicle. While these approaches offer high accuracy, they entail additional costs in terms of sensor expenses, installation, and maintenance. More recently, vision-based solutions [6] [7] have emerged as a cost-effective alternative to conventional PMS systems that rely on hardware sensors attached to each parking slot. However, in existing research, cars are typically detected for monitored parking spaces. In our study, we employ a Fisheye camera to detect license plates, enabling us to further recognize the license plate number allows us to precisely monitor when a car enters a parking space. Fisheye lenses have gained popularity due to their ability to provide natural, wide, and omnidirectional coverage, which traditional cameras with narrow fields of view (FoV) cannot achieve. In parking lot monitoring systems, fisheye cameras offer advantages by effectively reducing the number of cameras required to cover broader views

* This work was supported by Hongik University and Ministry of SMEs and Startups

Manuscript : 2023.12.16
Revised : 2024.01.15
Confirmation of Publication: 2024.01.29
Corresponding Author : Seongwon Cho e-mail : swcho@hongik.ac.kr

of cars and parking spaces. However, fisheye cameras present distorted views that require image undistortion and unwarping techniques or dedicated designs to handle these distortions during processing. It is worth noting that, to the best of our knowledge, there is currently no open dataset available for fisheye car object detection in surveillance applications. Additionally, when a car is parked, only the front of the car is visible. In this paper, we propose the use of deep Convolutional Neural Network [8] (CNN) for fisheye cameras to detect the license plate of the car, enabling us to accurately verify the position of the car. The decentralization of our system offers clear advantages, including reduced communication overhead and the elimination of computing bottlenecks. As a result, the system scales better as the number of monitored parking spaces increases. Our research presents a smart parking lot monitoring system that utilizes deep learning, specifically YOLOv5 [9]. We believe that our approach is advantageous compared to systems using ground sensors, such as magnetic sensors placed on every parking space. With a single fisheye camera, we can simultaneously monitor multiple parking lots at a significantly lower cost than installing and maintaining sensors in each parking lot.

II. RELATED WORK

The A comprehensive and diverse dataset is indeed crucial for the advancement of parking monitoring systems. In our study, we utilized a high-resolution, diverse, and large-scale parking lot dataset specifically collected for implementing our parking lot solution.

The choice of algorithms is crucial for achieving good object detection performance. Over the years, significant advancements [10] [11] [12] have been made in the field of detection models. The success of AlexNet at the ImageNet Large Scale Visual Recognition Challenge in 2012 [13] was a game changer in deep learning-based detection. This led to the development of twostage detectors, which generate proposals and classify them as potential objects. In recent years, one-stage detectors have gained prominence, classifying each region of interest as an object or background within a single detection pipeline.

There has been a growing emphasis on developing smaller networks for mobile applications, prioritizing fast inference times and high accuracy. Several notable models have emerged in this domain. MobileNetV2 + SSDLite, introduced in 2018, is an improved version of the MobileNet classification network, combined with the SSDLite [14] detection framework. Tiny-YOLOv4, developed as a fast variant of YOLOv4 [15], offers efficient object detection capabilities. MobileDet [16], a TensorFlow-based detection model, enhances performance on nonGPU devices such as CPU, DSP, and Edge TPU. Additionally, YOLOv5, based on YOLOv3 [17], incorporates improved augmentation and auto learning bounding box anchors.

In our study, we chose YOLOv5 to compare its performance with other published models. While there have been

studies utilizing the Coral TPU or deploying state-of-the-art detection networks on embedded devices, a comprehensive exploration of using the Coral TPU with SOTA models is lacking [18] [19] [20].

In summary, a comprehensive dataset and the selection of appropriate algorithms are crucial for advancing parking monitoring systems. Our study utilized a diverse and high-resolution dataset, along with the YOLOv5 model, to achieve accurate and efficient object detection. Further research is needed to explore the utilization of SOTA models such as YOLOv7 and YOLOv8 with the Coral TPU.

III. METHODOLOGY

1. YOLOv5

Ultralytics YOLOv5 builds upon the success of previous YOLO versions, introducing new features and improvements to enhance performance and flexibility even further. YOLOv5 is designed to be fast, accurate, and easy to use, making it an excellent choice for a wide range of object detection, instance segmentation and image classification tasks. It has three components, including the input layer, backbone network, neck network, and output detection layer. An illustration of this structure can be seen in Figure 1.

1.1. Backbone

The backbone network plays a crucial role in feature extraction from input images. In the case of YOLOv5, it leverages Cross Stage Partial Networks [21] (CSPNet) and Focus as its backbone to effectively identify important aspects of the input image. CSPNet addresses the issue of redundant network optimization gradient information within the backbone network and reduces the redundancy while enhancing the learning ability of CNNs. The backbone network utilizes the feature map from the base layer and employs a dense block to propagate the duplicated feature map to the next level, thus separating the feature map from the base layer.

1.2. Neck

The In this study, the CSP2 structure is adopted to enhance the fusion of network characteristics. The Neck component is commonly used to construct feature pyramids, as mentioned in [22], which helps models achieve effective object scaling generalization. By incorporating a Neck module, the network becomes capable of recognizing the same object at different sizes and scales. The Neck is designed to optimize the features extracted by the backbone network and typically consists of bottom-up and top-down pathways.

Traditionally, the Neck incorporates an up-sampling and down sampling block to efficiently re-process and utilize the feature maps extracted at different stages by the backbone. This allows for effective feature aggregation. Unlike single-shot detectors (SSD) [23], which do not involve a feature layer aggregation process, the Neck plays a critical role in the architecture of target detection models.

By leveraging the Neck component in the network architecture, this study aims to enhance the fusion of network

characteristics, enabling better object scaling generalization and improving the detection performance.

1.3. Head

The main role of the Head is to determine the location and category of the detected objects using the feature maps extracted from the backbone network. In object detection, there are generally two types of

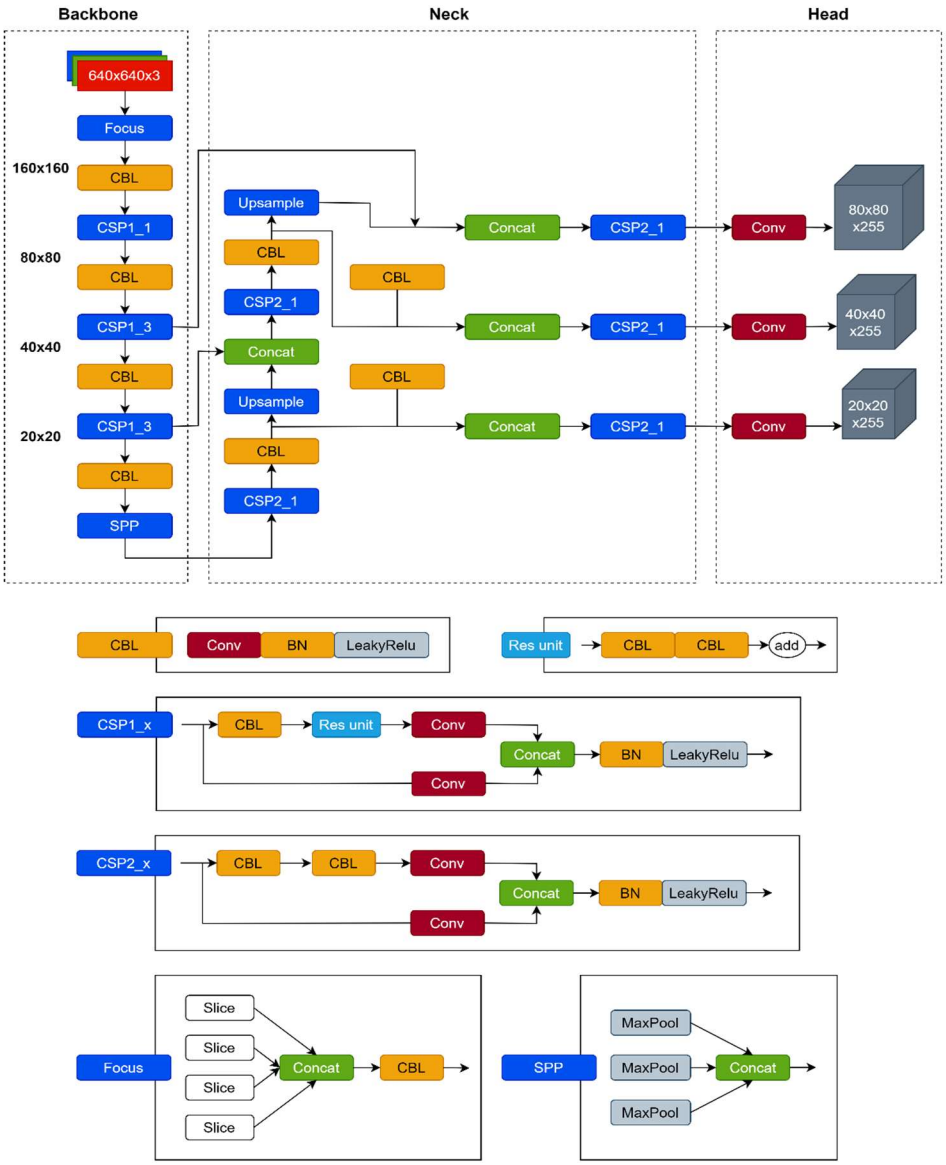

Fig. 1. Structural architecture network of YOLOv5

The Head component in the object detection process is responsible for the final detection and classification. After anchor boxes are applied to the feature maps, the Head generates the final output vectors, which include class probabilities and bounding boxes.

heads: one-stage object detectors and two-stage object detectors. The RCNN (Region-based Convolutional Neural Network) series is a prominent example of two-stage detectors, which have historically been dominant in the field of object detection.

In the YOLOv5 model, the Head is similar to the YOLOv3 [24] and YOLOv4 [25] models. It is primarily used in the final stage of the detection process, as mentioned in [26]. Once the anchor boxes are applied to the feature map, the Head generates the final output vector, which consists of class probabilities, object scores, and bounding boxes. Overall, the Head component in YOLOv5 is responsible for the crucial task of determining the location and category of objects in the input image. It completes the object detection pipeline by generating the final output vector that represents the detected objects and their associated information.

2. Quantization

In edge and embedded technologies, limited memory and computational capabilities pose challenges. To mitigate the strain on these constrained resources, optimization techniques for TensorFlow models have been employed. One commonly adopted method, especially with the Edge TPU Accelerator Module, is model quantization. Quantization is a valuable approach in AI modeling as it effectively reduces latency, power consumption, and model size while maintaining reasonably high accuracy levels

The deployment of deep neural networks (DNNs) to the Edge TPU involves a multi-step process (Figure 2), as illustrated in Figure 2. Initially, a deep learning model is transformed into the TensorFlow Lite (tflite) format. The model, initially represented in float32 precision, is then quantized to int8 or uint8 format [27]. This quantization process reduces the precision of the model weights and activations, making them more compact and suitable for efficient execution on low-power devices.

Once the model is quantized, the tflite file is further processed by the Edge TPU compiler. This compilation step optimizes the model specifically for the Edge TPU, resulting in a specialized tflite format tailored for inference on the Edge TPU. The optimized model is designed to take full advantage of the hardware capabilities of the Edge TPU, enabling efficient and fast inference on edge devices.

By employing model quantization and the Edge TPU compiler, the deployment of deep learning models to the Edge TPU becomes feasible, providing a balance between resource utilization, inference speed, and accuracy.

These optimization techniques enable efficient execution of AI models on edge and embedded devices, opening up possibilities for deploying sophisticated AI applications in resource-constrained environments.

3. Decision Tree

The complete decision tree algorithm is shown in Figure 3. Fisheye lenses provide a wide field of view, but they introduce distortion to the captured images, as

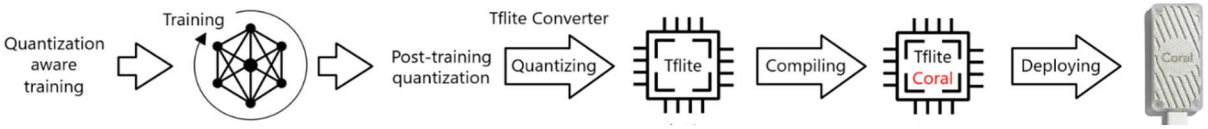

Fig. 2. Deployment of neural networks to the Edge TPU

shown in Figure 4. This distortion can pose challenges for object detection and training models. To address this issue, we propose using the LaRecNet algorithm [21] to undistort the raw images captured by the fisheye camera.

Figure 5 demonstrates the undistorted images after applying the LaRecNet algorithm. By correcting the distortion, we obtain more accurate representations of the scene, which improves the performance of our detection and tracking system.

Once the images are undistorted, we employ the YOLOv5s deep learning algorithm to detect license plates (LP) in the frames. To ensure reliable detection, we set a confidence threshold of 0.90. The detection process is performed frame by frame, enabling real-time identification of license plates with high accuracy.

Simultaneously, we define parking spaces by specifying four points in each frame: (x1, y1), (x2, y2), (x3, y3), and (x4, y4). If a license plate is detected within a defined parking space, we initiate tracking using the Simple Online and Realtime Tracking (SORT) algorithm [29]. This tracking mechanism allows us to monitor the movement of detected license plates over time, enabling efficient parking space management.

To facilitate tracking and management, we assign unique IDs to newly detected license plates entering the parking lot. This ID assignment enables us to accurately track the movement and occupancy of parking spaces, ensuring effective utilization of parking resources.

By combining the undistortion process using LaRecNet, YOLOv5s for license plate detection, deep Soft for tracking, and the utilization of unique IDs for license plates, our proposed system offers an efficient and accurate solution for object detection and tracking in parking lots using fisheye cameras.

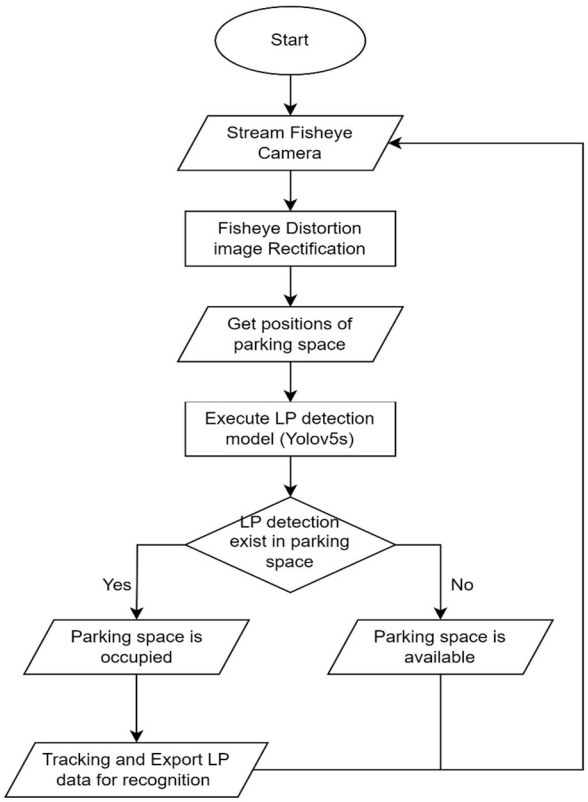

Fig. 3. Parking lot occupancy decision tree

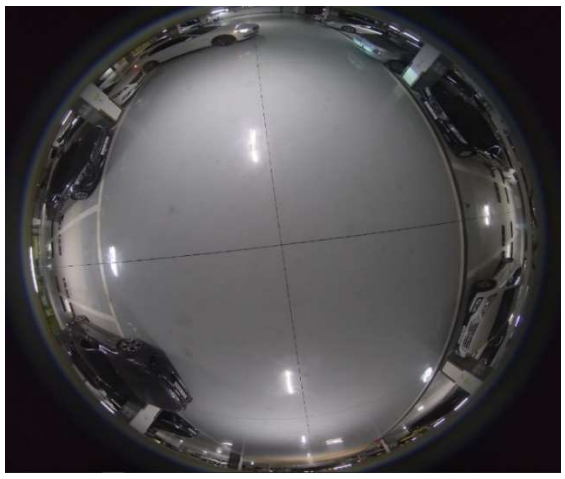

Fig. 4. Distortion image from Fisheye lenses

Fig. 5. Undistorted image

Fig. 6. NVIDIA Jetson Nano mainboard

Fig. 7. Accelerator Google Coral Edge TPU

IV. EXPERIMENT AND RESULTS

1. Dataset

The dataset for this study comprises 10,000 undistorted, annotated, and labeled images collected from Hongik University Parking Lot, each with a resolution of 2560x1280 pixels. Each image within this dataset contains 1 to 6 bounding boxes representing license plate object classes. The dataset was divided into three subsets: 70% for training, 10% for validation, and 20% for testing. This common partitioning approach is employed to prevent overfitting and assess the model's performance.

2. Hardware

The models were trained on a computer with the configuration: CPU AMD Ryzen Threadripper 2950X @ 4.40 GHz (16 threads x 32 core), 64GB DDR4 2666MHZ for RAM, GPU NVIDIA GeForce GTX 2080 Ti 12GB x 2, Linux Ubuntu 20.04.4 LTS and Python 3.9.12. In terms of the embedded board, NVIDIA Jetson Nano in Figure5 (CPU: ARM® Cortex® A57 MPCore (Quad-Core) Processor,Maximum Operating Frequency: 1.43GHz, Maxwell GPU 128-core GPU, Maximum Operating Frequency: 921MHz) was used for the experimental process, with the assistance of the accelerator Google Coral Edge TPU in Figure 6.

3. Evaluation metrics

In our experimental evaluation, we utilized several metrics to assess the performance of the models: Precision (the ratio of correctly predicted bounding boxes to the total number of predicted bounding boxes), Recall (the ratio of correctly predicted bounding boxes to the total number of ground truth bounding boxes), F1-score (The metrics measures the balance between precision and recall. When the value of F1-score is high, this means both the precision and recall are

high. A lower F1−score means a greater imbalance between precision and recall), Accuracy (represents the overall correctness of the model's predictions and is calculated as the ratio of the sum of true positives and true negatives to the total number of samples).

$$Precision\ (P) = \frac{TP}{TP + FP}$$

$$\text{Recall}\ (R) = \frac{TP}{TP + FN}$$

$$\text{F1 score}\ = \frac{2 * P * R}{P + R}$$

$$Accuracy\ = \frac{TP}{(TP + FP + FN)}$$

TP (True Positives): The number of correctly predicted bounding boxes for the single class. FP (False Positives): The number of predicted bounding boxes that do not match the ground truth for the single class. FN (False Negatives): The number of ground truth bounding boxes that were not detected by the model for the single class.

In addition to these evaluation metrics, we also considered the inference time and model file size. Inference time refers to the duration taken by the model to process an input image and make predictions. A shorter inference time is desirable, especially for embedded systems with limited computational capabilities. Model file size indicates the storage space required to store the model's parameters and architecture. Minimizing the model file size is important for efficient deployment and management of the model, particularly on resource−constrained devices.

4. Experimental Results

Table 1 illustrates the performance comparison of various techniques, presenting reported results. Our solution consistently outperforms other methods in both accuracy and speed. Specifically, YOLOv5s demonstrates superior accuracy compared to other techniques, ranging from 2.1% to 20.7% when tested on images from the test set. In our test set, YOLOv5s achieves an impressive 94.3% accuracy with an inference time of 0.071 seconds, surpassing the best compared method, YOLOv4, which attains 92.2% accuracy with an inference time of 0.094 seconds. This makes YOLOv5s 2.1% more accurate and 0.023 seconds faster, with the model input size being the highest at 640x640 pixels. On the other hand, SSD MobileNetV2, with the highest inference time of 0.042 seconds, exhibits a lower accuracy of only 73.6%. Despite its faster runtime, the model's input size is the lowest at 300x300 pixels. When compared with our YOLOv5 model, SSD MobileNetV2 is 20.7% less accurate and only 0.029 seconds faster.

Table 1. Model comparison

Model	Input size	Precision	Recall	F1score	Accuracy	Speed (ms)	File Size (MB)
EfficientDet Lite0 [30]	320x320	0.957	0.869	0.911	83.6 %	0.172	6.12
EfficientDet Lite1 [30]	384x384	0.962	0.871	0.914	84.2 %	0.261	8.24
EfficientDet Lite2 [30]	448x448	0.981	0.908	0.943	89.2 %	0.537	10.75
SSD Mobilenetv2 [14]	300x300	0.946	0.768	0.848	73.6 %	0.042	7.21
SSDLite MobileDet [16]	320x320	0.953	0.794	0.866	76.4 %	0.462	6.42
YOLOv3 [17]	608x608	0.971	0.872	0.919	85.0 %	0.153	10.64
YOLOv4 [15] [31]	640x640	0.986	0.934	0.959	92.2 %	0.094	9.85
YOLOv5s [9]	640x640	0.991	0.952	0.971	94.3 %	0.071	8.04

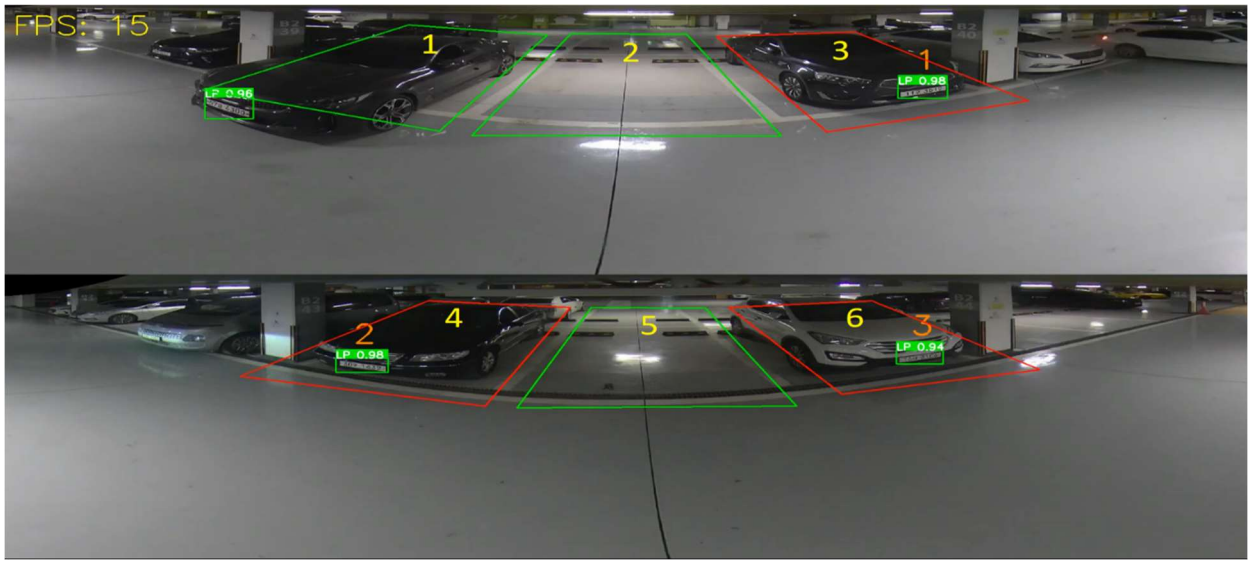

Fig. 8. Real time testing results

Figure 8 showcases the testing setup at a real-time parking lot where the evaluation phase took place. The figure displays the availability of parking spaces, with spaces 1, 2, and 5 indicated as green color, signifying their availability. As shown, a car was about to enter parking space number 1 but was not fully inside, resulting in our model detecting the license plate position outside the designated parking space. Parking spaces 3, 4, and 6 already had cars parked shown in red color, and our model successfully tracked and assigned IDs to monitor their status.

V. CONCLUSION

In this study, we have successfully applied the YOLOv5s deep learning algorithm in conjunction with a proposed decision tree to develop a smart parking lot system capable of detecting the occupancy status of parking spaces and monitoring the position of parked cars.

The YOLOv5s algorithm, known for its efficiency and accuracy in object detection, was chosen as the backbone of our system. We quantized the model to optimize its performance and facilitate its deployment on resource-constrained devices. The algorithm was successfully run on the Jetson Nano mainboard and the Google Coral Edge TPU accelerator, demonstrating its versatility across different hardware platforms.

To make the final determination of the parking space status, we introduced a decision tree. This decision tree takes into account various factors, such as the presence of license plates, the position of the detected vehicles, and the defined parking space boundaries. By combining the outputs of the YOLOv5s algorithm and the decision tree, we were able to accurately assess the availability of parking spaces in real-time.

The results of our comparative analysis against other techniques clearly indicated the superiority of the proposed algorithm. Our approach achieved robustness in accurately detecting parking space occupancy, while also demonstrating faster processing times and a smaller

model file size compared to alternative methods. This highlights the efficiency and effectiveness of our system in practical applications.

Looking ahead, our future research endeavors will focus on

exploring more optimized methods to further enhance the speed and reduce the size of the model without compromising its performance. This will involve investigating techniques such as model compression, knowledge distillation, and network architecture design. By continuously improving the efficiency and effectiveness of our system, we aim to provide even more reliable and efficient solutions for smart parking lot management.

REFERENCES

[1] Mingkai Chen and Tianhai Chang, "*A parking guidance and information system based on wireless sensor network*," *2011 IEEE International Conference on Information and Automation, Shenzhen, China,* pp. 601–605, 2011.

[2] E. Sifuentes, O. Casas and R. Pallas–Areny, "*Wireless Magnetic Sensor Node for Vehicle Detection With Optical Wake–Up*," *in IEEE Sensors Journal*, vol. 11, no. 8, pp. 1669–1676, Aug. 2011.

[3] Z. Zhang, M. Tao and H. Yuan, "*A Parking Occupancy Detection Algorithm Based on AMR Sensor,*" in *IEEE Sensors Journal*, vol. 15, no. 2, pp. 1261–1269, Feb. 2015.

[4] Alam, M., Moroni, D., Pieri, G., Tampucci, M., Gomes, M., Fonseca, J., Ferreira, J. and Leone, G.R., 2018. Real–time smart parking systems integration in distributed ITS for smart cities. *Journal of Advanced transportation*, 2018.

[5] Joseph, J., Patil, R.G., Narahari, S.K.K., Didagi, Y., Bapat, J. and Das, D., 2014. *Wireless sensor network based smart parking system. Sensors & Transducers,* vol .162*, no.* 1, pp. 5–10, 2014.

[6] Cai, B.Y., Alvarez, R., Sit, M., Duarte, F. and Ratti, C., 2019. Deep learning–based video system for accurate and real–time parking measurement. *IEEE Internet of Things Journal,* vol. 6, no. 5, pp.7693–7701.

[7] Cho, W., Park, S., Kim, M.J., Han, S., Kim, M., Kim, T., Kim, J. and Paik, J., 2018, January. Robust parking occupancy monitoring system using random forests. *In 2018 International Conference on Electronics, Information, and Communication (ICEIC),* pp. 1–4.

[8] S. Albawi, T. A. Mohammed, and S. Al–Zawi, "Understanding of a convolutional neural network," *in 2017 international conference on engineering and technology (ICET)*, pp. 1–6, Ieee, 2017.

[9] G. Jocher, "YOLOv5 by ultralytics." https://github.com/ ultralytics/yolov5, 2020 (accessed sep., 10, 2023).

[10] Z.–Q. Zhao, P. Zheng, S.–t. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 11, pp. 3212–3232, 2019.

[11] X. Wu, D. Sahoo, and S. C. Hoi, "Recent advances in deep learning for object detection," *Neurocomputing*, vol. 396, pp. 39–64, 2020.

[12] L. Jiao, F. Zhang, F. Liu, S. Yang, L. Li, Z. Feng, and R. Qu, "A survey of deep learning–based object detection," *IEEE access*, vol. 7, pp. 128837–128868, 2019.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[14] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: *Inverted residuals and linear bottlenecks," in Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

[15] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," arXiv preprint arXiv:2004.10934, 2020.

[16] Y. Xiong, H. Liu, S. Gupta, B. Akin, G. Bender, Y. Wang, P.-J. Kindermans, M. Tan, V. Singh, and B. Chen, "Mobiledets: Searching for object detection architectures for mobile accelerators," *in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 3825–3834, 2021.

[17] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[18] A. Ghosh, S. A. Al Mahmud, T. I. R. Uday, and D. M. Farid, "Assistive technology for visually impaired using tensor flow object detection in raspberry pi and coral usb accelerator," *in 2020 IEEE Region 10 Symposium (TENSYMP)*, pp. 186–189, IEEE, 2020.

[19] A. Yazdanbakhsh, K. Seshadri, B. Akin, J. Laudon, and R. Narayanaswami, "An evaluation of edge tpu accelerators for convolutional neural networks," *arXiv e-prints*, pp. arXiv–2102, 2021.

[20] V. Mazzia, A. Khaliq, F. Salvetti, and M. Chiaberge, "Realtime apple detection system using embedded systems with hardware accelerators: An edge ai application," *IEEE Access*, vol. 8, pp. 9102–9114, 2020.

[21] C.-Y. Wang, H.-Y. M. Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh, and I.-H. Yeh, "Cspnet: A new backbone that can enhance learning capability of cnn," *in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pp. 390–391, 2020.

[22] Z. Li and F. Zhou, "Fssd: feature fusion single shot multibox detector," *arXiv preprint arXiv:1712.00960*, 2017.

[23] D. Biswas, H. Su, C. Wang, A. Stevanovic, and W. Wang, "An automatic traffic density estimation using single shot detection (ssd) and mobilenet-ssd," *Physics and Chemistry of the Earth*, Parts A/B/C, vol. 110, pp. 176–184, 2019.

[24] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," arXiv preprint arXiv:1804.02767, 2018.

[25] D. Wu, S. Lv, M. Jiang, and H. Song, "Using channel pruning-based yolo v4 deep learning algorithm for the real-time and accurate detection of apple flowers in natural environments," *Computers and Electronics in Agriculture*, vol. 178, p. 105742, 2020.

[26] B. Jiang, R. Luo, J. Mao, T. Xiao, and Y. Jiang, "Acquisition of localization confidence for accurate object detection," *in Proceedings of the European conference on computer vision (ECCV)*, pp. 784–799, 2018.

[27] "Post-training quantization." https://www.tensorflow.org/lite/performance/post training quantization(accessed sep., 20, 2023).

[28] Z.-C. Xue, N. Xue, and G.-S. Xia, "Fisheye distortion rectification from deep straight lines," arXiv preprint arXiv:2003.11386, 2020.

[29] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," *in 2017 IEEE international conference on image processing (ICIP)*, pp. 3645–3649, IEEE, 2017.

[30] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient

object detection," *in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10781–10790, 2020.

[31] WongKinYiu,"Yolov4. "https://github.com/WongKinYiu/PyTorch YOLOv4, 2020.

[32] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," *in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10781–10790, 2020.

Authors

To Xuan Dung

He received B.S degree in control and automation engineering from Hanoi University of Science and Technology, Vietnam and M.S degree from Hongik University, 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.