ODAR: A Lightweight Object Detection Framework for Autonomous Driving Robots

Le Hoang Duong¹, Huynh Thanh Trung¹, Pham Minh Tam², Gwangzeen Ko³, Jung Ick Moon³, Jun Jo⁴, Nguyen Quoc Viet Hung¹

¹Griffith University, Australia
²Hanoi University of Science and Technology, Vietnam
³Electronics and Telecommunications Research Institute, Korea

Abstract—Object detection is an emerging and essential problem in recent years, which has been widely applied in many aspects of daily life such as video surveillance, self-driving robots, and automatic payment. The rapid development of deep learning models allows object detectors to work in real-time with high accuracy. However, such a sophisticated model often requires robust computing infrastructure such as powerful graphics processing units (GPUs). This requirement might cause a severe issue for embedded systems with small, power-efficient artificial intelligence (AI) systems like Jetson Nano, which are often restricted in both memory storage and computing sheer power. In this work, we aim to address this challenge by proposing a lightweight object detection framework that is specialized for the Internet of Things (IoT) devices with low-power processors such as Jetson Nano. In order to detect the object with different size, our framework employs a backbone residual CNN-based network as the feature extractor. We then design a multi-layer model to combine the feature at different levels of granularity, before using the processed feature to locate and classify the object. We also apply augmentation techniques to enhance the robustness of the framework to adversarial factors. Extensive experiments on real devices in many scenarios, such as autonomous cars or wireless robot recharging systems, showed that our technique can achieve nearly on par results with the state-of-the-art YOLOv5 while requires only one-fourth of computation power.

Index Terms—object detection, autonomous robot driving, image processing, deep neural network

I. INTRODUCTION

Object detection has been attracting increasing amounts of attention in recent years due to its wide range of applications and recent technological breakthroughs. This task is under extensive investigation in both academia and real world applications, such as monitoring security, autonomous driving, transportation surveillance, drone scene analysis, and robotics vision. Among many factors and efforts that lead to the fast evolution of object detection techniques, notable contributions should be attributed to the development of deep convolution neural networks and Graphics Processing Units (GPUs) computing power. Most of the state-of-the-art object detectors utilize deep learning networks as their backbone and detection network to extract features from input images (or videos), classification and localization respectively.

However, the modern deep learning architecture used in the latest detectors often requires heavy computing power and large memory space for model storing [1, 2, 3]. This raises the challenge for deployment on low-power systems like IoT devices, which has restricted resources in term of both memory and processing power. The tradition solutions often urge the devices to send the captured information to a powerful master server or Cloud for processing, then the processed data is sent back to the devices. Such methods are applicable for many systems, but requires a fast and stable connection as well as is prone to latency and noise.

In this work, we address the problem of object detection for a wireless robot recharging system, where the robot is required to quickly recognize the obstacles and adjust its moving to reach its charging station. We employ the edge computing strategy [4], [5], [6], [7], where the computing is processed at the edge of the network (IoT devices). This allows the device to have fast reaction in various scenario and avoid any latency by transmission. To handle the restricted resources issue, we propose a real-time, lightweight object detection framework for autonomous robot. In order to detect the object with different size, the framework leverage a light backbone residual CNN-based network as the feature extractor. The extracted features are aggregated at different level of granularity, then are combined by a multi-scale BiFPN model. The output features are then consumed to produce the box and class prediction. To simplify the model, our framework applies the object vs non-object setting as the robot need to avoid any possible obstacle. Some sophisticated augmentation such as Mosaic augmentation [8] is employed to enhance robustness of our framework to small objects and adversarial factors. The salient contributions of our work are summarized as follows:

- We proposed ODAR, a lightweight, real-time Object Detection model that assists Autonomously drive for Robots. Inspired from the SOTA detector Yolo [9], [10], our framework was designed to adapt with restricted resources while guaranteeing high accuracy.
- We developed a light residual CNN network that allows the feature at different resolution can be learnt efficiently while require less resource than the existing backbone structure. Then, our proposed model employed the multi-scale fusion model BiFPN [11] to bring the features together to predict the bounding box and object label.
- We constructed a dataset using real devices, which is a wireless robot recharging system where the robot autonomously find the way to charging place. The dataset...
contains 36,000 images with the objects ranging from large size (e.g. book, cable) to small size (e.g. coin), which allows us to fully examine the capability of the techniques in detecting the small objects.

- We compared our technique vs other baselines using the constructed datasets and popular external datasets (e.g. COCO [12]) in various scenarios. The empirical result shows that our framework not only achieves better results compared to baseline techniques but also exhibits robustness to various adversarial factors such as illumination, image quality and object size.

In the remainder, the report is organised as follows. We first survey the literature of object detection techniques in Section II then introduces our model in Section III. Section IV reports the experimental results, before Section V concludes the paper.

II. LITERATURE STUDY

In this section, we give a brief survey of object detection literature. The techniques are organized into two categories: Traditional Techniques and Deep Learning Techniques.

A. Traditional Techniques

Most of the early object detection algorithms were developed based on handcrafted features [13], [14], [15], [16], [17]. Due to the lack of deep learning techniques and the computing power, designing sophisticated feature by expert knowledge was nearly the only choice, along with variety of speed up tricks to fully exploit the limited computing resources [18], [19], [20], [21], [22], [23], [24]. P. Viola and M. Jones [13] was the first to propose a real-time detector of human faces, namely VJ detector. The detector employed a straightforward strategy, which is to apply sliding windows at all possible locations and scales. The next version [25] improved the detection speed by integrating new functions: “integral image” and “feature selection”. The integral image is a speed up box filtering technique using Haar wavelet [26]. For feature selection, VJ detector used Adaboost algorithm [27] to select a small set of features from a huge set of random features pools. Histogram of Oriented Gradients (HOG) detector [28] proposed important improvements of the scale-invariant feature transform [29] and shape contexts [30]. HOG detector employed a dense grid system and overlapping local contrast normalization to balance the feature invariance (e.g. transition, scale) and improve the accuracy. The technique also rescaled the input image for multiple times while maintaining the detection window size to detect small objects. The improve variants of HOG detector [31], [32] followed a “divide and conquer” strategy, where the detection of an object was treated as an ensemble of detections of object parts.

B. Deep Learning Techniques

With the development of deep learning models and computing resources, great progress has been made in object detection. The handcrafted feature engineering which consumes human effort and problem specific was replaced by the automatic deep learning feature extractor, mostly convolutional neural network models (CNNs) [33], [34], [35], [36], [37].

Two-stage detectors. Girshick et al. [38] proposed R-CNN, the first region based CNN object detector. This pioneer work proved that a CNN could lead to significant higher performance than the traditional feature-based techniques such as HOG Detector. After selectively searching for category-independent region proposals, R-CNN applied a five-layer convolutional network on each proposal to extract a 4096-dimensional feature vector. The features then were fed to a set of object class-specific linear SVMs, following by a bounding box regressor. Girshick et al. [39] went beyond R-CNN by extracting features from entire input images and passed only the region of interest (RoI) to save the computation for each region proposal. Also, a RoI pooling layer was employed to obtain a fixed size feature map from region proposals of different size. Faster R-CNN [40] replaced the slow RoI selective search process by a novel CNN-based region proposal network (RPN) to efficiently predict region proposals in various scales. Furthermore, a multi-scale anchors system was used to simplify the RoI search by removing the need of multiple scales of input images or features. Mask R-CNN [41] extended Faster R-CNN by using the ResNet-FPN [42] as a backbone network to extract RoI feature.

One-stage detectors. While the two-stage detectors follow the multi-step, coarse-to-fine strategy; the latest one-stage detectors perform the task in an end-to-end, one-step manner [43], [44], [45], [46], [47]. YOLO [48] was the first technique following this scheme, with two main improvements comparing to the Faster R-CNN. First, the technique predicts significant less RoIs (100 vs. 2000 of Faster R-CNN). Second, YOLO applied a unified architecture including CNN-based feature extractors and bounding box/class predictors. The feature extraction network contained 24 convolutional layers followed by 2 fully connected layers, then the whole network was used for prediction using regression loss. YOLOv2 [9] proposed a new feature extraction backbone namely Darknet-19, which included 19 convolution layers and 5 maxpool layers. The new backbone reduced significantly amount of operations needed while enhanced the quality of extracted features. Besides, YOLOv2 applied popular techniques such as batch normalization, high resolution classifier and dimension clusters to enhance the detection accuracy. YOLOv3 [10] used independent logistic classifiers to allows the detector to handle with more complex datasets that contains overlapping object labels. YOLOv3 also replaced Darknet-19 in YOLOv2 by a deeper and robust feature extractor, called Darknet-53, including 53 residual convolutional layers. Yolov4 implemented new features, including Weighted-Residual-Connections (WRC), Cross-Stage-Partial-connections (CSP), Cross mini-Batch Normalization (CmBN) and Self-adversarial-training (SAT) [8]. These improvements helped to increase the processing speed and enhance the accuracy on the large project comparing to YOLOv3.
Comparing to the existing techniques, our technique is tailored for IOT devices with restricted computing power while require accurate and real-time reaction. Our framework proposes a brand new backbone residual CNN-based, and designs a multi-layer model to combine the feature at different level of granularity. We also simplify the detection setting as a binary classification, and apply augmentations to enhance the robustness to adversarial factors.

III. PROPOSED FRAMEWORK

The Figure 1 demonstrates the overview of our framework. As shown, the framework consists of three main components. The backbone is a convolutional neural network that aggregates and forms image features at different granularities. The neck is a series of layers to mix and combine image features to pass them forward to prediction. The heads consumes features from the neck and takes box and class prediction steps.

Backbone. For the backbone module, we employ a residual CNN network that consists of successive convolutional layers, which is shown in Figure 2. Our technique inspires from EfficientNet [11], a convolutional network architecture that performs better pattern extraction while guarantees the efficiency of the model. The concept of the model is to use a multi-objective neural architecture search that optimizes at the time accuracy and efficiency criteria. We also go beyond the existing work by adding the residual connection to refrain the details from being washed out over deep architecture.

In more details, the network starts with two convolutional layers of the size 3 × 3, the second one using the stride of 2, which performs lightweight filtering. Then, the network continues with multiple convolution layer. Instead of the standard convolution, our technique leverages depth-wise separable layer to reduce the computational cost while guarantees the quality of pattern extraction. For each convolution block, a residual connection is added to aid the gradient flow during the backpropagation and allows the flow of memory (or information) from initial layers to last layers. After the stacked layers, the output feature goes through a average pooling layer, followed by a fully connected layer at the end of the network. The final extractor architecture is simpler than the recent head architecture used in the latest YOLO versions such as Darknet53 [10] and CSPNet [49] to make our framework adapt to the IOT environment setting. Note that this new network even uses less convolutional layer than Darknet-19 used in YOLOv2 [9] (16 comparing to 19), but achieve even higher result than the next version YOLOv3, which will be used in YOLOv2 [9] (16 comparing to 19), but achieve even higher result than the next version YOLOv3, which will be later proved in our experimental results.

Neck. Like the latest Yolo versions, we choose PA-NET [10] as the neck network for feature aggregation. We implement the BiFPN variant [11] due to its lightweight and state-of-the-art performance. BiFPN is a multi-scale feature fusion model which aims to aggregate features at different resolutions using the extracted pattern from the backbone network. Formally, given a list of multi-scale features $P_{\text{in}} = P_{\text{in}}^l_1, P_{\text{in}}^l_2, \ldots$, where $P_{\text{in}}^l$ represents the features at the scale $l_i$, the network attempts to find a transformation $f$ that is able to effectively aggregate different features, results in a list of new output features: $P_{\text{out}} = f(P_{\text{in}})$. Our neck network takes level 3-5 input features $P_{\text{in}}^l_3, \ldots, P_{\text{in}}^l_5$, where $P_{\text{in}}^l_i$ captures the feature level with the resolution of $\frac{1}{2^l}$ of the input images. For example, given the input resolution is 1024 x 1024, $P_{\text{in}}^l_5$ represents feature level 3 (1024/2^3 = 128) with resolution 128 x 128, while $P_{\text{in}}^l_3$ depicts the feature at level 5 with the resolution of 32 x 32. The neck network then fuses the multi-scale features in a top-down manner:

$$P_{\text{out}}^l = \text{Conv}(P_{\text{in}}^l + \text{Resize}(P_{\text{out}}^{l+1}))$$

where Resize is a upsampling/downsampling operation for resolution matching, and Conv is a convolutional operation for feature processing. The fusion of feature at different level not only facilitates the detection of the small objects, but also helps to accurately detect the large object using the signal from its parts.

Head. The head module takes the features extracted from the neck network to takes box and class prediction steps.

For bounding box prediction, our system leverages dimension clusters as anchor boxes. In more details, the model predicts 4 coordinates for each bounding box, $t_x, t_y, t_w, t_h$, which represents the X-position of the box center, Y-position of the box center, box width and box height, respectively. Then, given the ground-truth for a coordinate prediction is $\hat{t}$, the error is calculated by subtracting the groundtruth by the prediction values: $\hat{t} - t$. The loss then can be calculated over data batches using sum of squared error loss as follows:

$$L_b = \frac{1}{N} \sum_{i=1}^{N}(\hat{t}_i - t_i)^2$$

Besides, we apply a threshold of 0.5 to determine whether the predicted box is correctly for an existing object. If the overlap between prediction box and the ground-truth is under the threshold, we ignore the prediction, following [40]. Unlike [40], our system only assigns one bounding box prior for each ground truth object. If a bounding box prior is not assigned to a ground truth object it incurs no loss for coordinate or class predictions, only objectness.

For class prediction, each box predicts the classes the bounding box may contain using multilabel classification. To simplify the model, we treat the label to be 0 or 1, which represents for object and non-object setting. The problem thus simplify the model, we treat the label to be 0 or 1, which represents for object and non-object setting. The problem thus becomes the binary classification problem. As a result, we use binary cross-entropy loss for the class predictions:

$$L_c = -\frac{1}{N} \sum_{i=1}^{N} (y_i \log(p(y_i)) + (1 - y_i) \log(p(1 - y_i)))$$

where $y$ is the label and $p(y)$ is the predicted probability for all $N$ samples. The bounding box loss and the classification loss then are combined by the weighted-sum strategy to generate the final loss:

$$L = \lambda L_B + (1 - \lambda) L_C$$
with $\lambda$ being the balancing hyperparameter.

**Data Augmentation.** With each training batch, we pass training data through a data loader, which augments data online. The data loader implements three kinds of augmentations: illumination change, color space adjustments and mosaic augmentation. Mosaic augmentation [8], which combines four images into four tiles of random ratio, is especially useful to help the model to handle “small object problem”. For illumination factors, we address the most common ones which is brightness and contrast. For brightness, we multiply a pixel intensity by a brightness factor $\tau$ and then clip the scaled value to a valid range of $[0, 255]$:

$$x = \begin{cases} \tau x, & \text{if } \tau x < 255 \\ 255, & \text{if } \tau x > 255 \end{cases} \quad (5)$$

We selected five brightness factors including $\{0.5, 0.75, 1, 1.5, 2\}$, which could create significant differences between images. For contrast, we change the contrast of the images using the formula:

$$x = \begin{cases} \text{avg} + \psi(x - \text{avg}), & \text{if } 0 < \text{avg} + \psi(x - \text{avg}) < 255 \\ 255, & \text{if } \text{avg} + \psi(x - \text{avg}) > 255 \\ 0, & \text{if } \text{avg} + \psi(x - \text{avg}) < 0 \end{cases} \quad (6)$$

where $x$ is value of pixel in image, $\text{avg}$ is average of value of all pixel in image, and $\psi$ is contrast factor of $\{0.5, 0.75, 1, 1.5, 2\}$.

**IV. E MVIRICAL EVALUATION**

**A. Experimental Settings**

**Datasets.** To fully evaluate the performance of our proposed technique to the real-world setting, we constructed a dataset using real device with objects varying in size, shapes, colors and materials (see Figure 4). Each object is located in 10 different positions in the frame. We also augmented the data with illumination factors (e.g. brightness, contrast). The final datasets include 36k images in total.

Besides, we also leverage the popular external dataset COCO (Common Objects in Context) [12] to evaluate the performance of our model. The dataset contains a total of 300,000 images in 91 various categories.
Baselines. We consider several state-of-the-art deep learning based object detection techniques as follows.

- **EfficientDet**: is a state-of-the-art object detector proposed by Google Brain, which implements a fast feature fusion and proposes a compound scaling method to scales the resolution, depth and width for the whole architecture.

- **YOLOv3**: is a one-stage object detector that employs a deep residual CNN network for feature extractor and a multi-label classifier to tackle the problem of overlapping label object detection.

- **YOLOv5**: is the latest version of the popular detection framework YOLO, which can achieves high-accurate detection result by leveraging the sophisticated backbone CSP-Darknet and imagery augmenters.

Metrics. To evaluate the accuracy of the technique, we employ the mAP@k metric, which calculate the average precision of the overlap between the predicted and the ground-truth boundaries, a.k.a Intersection over Union (IoU); and k being the overlap threshold. We use two popular threshold 0.5 and 0.95, which means the used metrics are mAP@[0.5, 0.95].

Another metric for evaluating the performance of the Foreign Object Detection (FOD) technique is the processing speed. We measure the detection speed as the number of frames or images which a technique can process per second.

Reproducibility Environment. We develop the system prototype as shown in Figure 5. This system operates based on Jetson Nano development kit acts as a main processing unit which is powered by Quad-core ARM A57 at 1.43 GHz CPU, 128-core Maxwell graphic card and 04 GB RAM 64-bit LPDDR4 together with the Raspberry Pi Camera V2 to detect foreign objects. To test the accuracy of the system in different illumination conditions, the system is located in the simulated environment with a range of light intensity by installing 2 light tubes. In order to measure the light intensity, we build a tool which is powered by Arduino Mega (based on 8-bit Atmel Microcontroller ATmega2560) and light sensor (based on Light Dependent Resistor).

B. Efficiency Comparison

In this experiment, we examined the efficiency of our technique in terms of processing time and computation resources required. We considered this one of the most essential criteria beside the detection accuracy, given the restrictive environment for embedded systems such as our wireless robot recharging system. For computation resources, we count the total parameters required in each deep learning models of our proposed framework and the other baselines. For detection speed, we calculate the processing time our robot needed for each image in milliseconds.

The experimental result is shown in Table I. It can be seen that our technique achieves the fastest processing time and has least number of parameters among 4 techniques (YOLOv3, YOLOv5 and EfficientDet) with 6.3 milliseconds and 2,065,686 parameters respectively. Our model ODAR use only one-fourth of number of parameters comparing to the YOLO models (v3 and v5) and two-thirds of those of EfficientDet. We also process the images significantly faster than YOLOv5 and EfficientDet, and slightly faster than YOLOv3. Despite achieving impressive efficiency, our model could achieve nearly equivalent result when it comes to accuracy, which is presented in the next experiments.

<table>
<thead>
<tr>
<th>Model</th>
<th>time (ms)</th>
<th>number of parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>YOLOv3</td>
<td>6.8</td>
<td>8,669,876</td>
</tr>
<tr>
<td>YOLOv5</td>
<td>8.1</td>
<td>7,276,605</td>
</tr>
<tr>
<td>EfficientDet</td>
<td>9.4</td>
<td>3,880,067</td>
</tr>
<tr>
<td>ODAR</td>
<td>6.3</td>
<td>2,065,686</td>
</tr>
</tbody>
</table>

C. End-to-end Comparison

In this experiment, we examined the end-to-end performance of object detection techniques on our dataset and COCO dataset. We used the mean average precision (mAP@0.95 and mAP@0.5, mAP over the Intersection over Union (IOU) threshold = 0.95 and 0.5) which compares the ground-truth bounding box to the detected box and returns
a score. The higher the score, the more accuracy the model achieves in its detection. The results are shown in Table II.

Overall, the scores of mAP@0.5 of EfficientDet-d0, which are applied for our datasets, stands in the highest position among four techniques with the score of 0.998. The other three techniques (our technique, YOLOv3 and YOLOv5) together stand in the second position with the score of 0.997. In terms of mAP@0.95, our technique comes in the second position right after the state-of-the-art YOLOv5 model. When we use COCO dataset, scores of mAP@0.5 of all techniques is significantly reduced. Our technique comes in the third place with 0.492 higher than those of YOLOv3 with 0.352. When it comes to scores of mAP@0.95, our technique stands in the third position with 0.319 that nearly comes close to the state-of-the-art technique YOLOv5 with 0.366.

D. Influence of brightness

This experiment studied the object detection sensitivity to the brightness. Figure 6 shows the results of an experiment in which the brightness factor of the visual content varied from 0.5 to 2. In terms of all four techniques (our model ODAR, YOLOv5, YOLOv3 and EfficientDet-d0), when the brightness value varied from 0.5 to 2, OADR model delivers the highest results for mAP@0.5 and mAP@0.95 of our dataset. When it comes to COCO dataset shown in Figure 7, YOLOv5 model produces the highest scores in all values of brightness for mAP@0.5 and mAP@0.95. ODAR model stands in the third position right after EfficientDet-d0 in this category. As we can see, with the least of number of total parameters, ORAD model running on our dataset has the lowest level of object detection sensitivity to brightness.

E. Influence of contrast

We then studied the effects of contrast factor to the object detection accuracy for all four techniques. We also use mAP@0.5 and mAP@0.95 metrics to evaluate them. Figure 8 shows the results of an experiment in which the contrast factor of the visual content was varied from 0.5 to 2 for our dataset. Similarly to brightness related results for mAP@0.5 and mAP@0.95, our ODAR model and state-of-the-art YOLOv5 model together produce the highest accuracy among all four techniques in all contrast levels. Specially, all the scores of our ODAR model and YOLOv5 are equal except when the score at brightness level 2 for mAP@0.5 and at brightness level 1.5 for mAP@0.95, YOLOv5 model just goes a bit higher than our ODAR model.

Again, we also evaluate the object detection accuracy for all four techniques with COCO dataset, this has been shown in Figure 9. Our ODAR model score also comes in the third place among all techniques for the mAP@0.5 and mAP@0.95. The last position belongs to YOLOv3 model in this category. As we can see again, with the least of number of total parameters, our ODAR model running on our dataset has the lowest level of object detection sensitivity to contrast and is nearly equal to the state-of-the-art YOLOv5 does.

F. Influence of image compression

Image compression is the last property we investigated. This technique helps reducing the image size significantly while still keeping its quality at an acceptable level. We explore some different settings of image compression in YOLOv5,
TABLE II: End-to-end comparison in bounding box detection

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Yolov3 mAP@0.5</th>
<th>mAP@0.95</th>
<th>Yolov5 mAP@0.5</th>
<th>mAP@0.95</th>
<th>Efficientdet-d0 mAP@0.5</th>
<th>mAP@0.95</th>
<th>Our technique mAP@0.5</th>
<th>mAP@0.95</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our dataset</td>
<td>0.997</td>
<td>0.773</td>
<td>0.997</td>
<td>0.801</td>
<td>0.998</td>
<td>0.744</td>
<td>0.997</td>
<td>0.799</td>
</tr>
<tr>
<td>COCO</td>
<td>0.352</td>
<td>0.176</td>
<td>0.557</td>
<td>0.366</td>
<td>0.511</td>
<td>0.332</td>
<td>0.492</td>
<td>0.319</td>
</tr>
</tbody>
</table>

Fig. 9: Detection Sensitivity to contrast in COCO dataset

YOLOv3, EfficientDet-d0 model and our ODAR model. As the results shown in Figure 10 for our dataset, the scores of our model and YOLOv5 are equal and together achieve the highest level among four models in all level of image compression ratio except when the score at compression level from 50 to 90 per cent for mAP@0.95, YOLOv5 model just goes a bit higher than our model. When we change our dataset to the COCO dataset for evaluation which is shown in Figure 11, our ODAR model score also comes in the third place among all four techniques for the mAP@0.5 and mAP@0.95. The last position belongs to YOLOv3 model.

Through results, with the least of number of parameters, our model running on our dataset has the lowest level of object detection sensitivity to image compression and is nearly equal to the state-of-the-art YOLOv5 does.

Fig. 11: Sensitivity to image compression in COCO dataset

V. CONCLUSION

In this research, we have surveyed the literature and proposed a lightweight real-time deep learning based framework for the object detection problem in self-driving robots. We inspired and went beyond the latest one-stage deep learning detector YOLOv5 by tailoring the architecture to adapt well with the restricted environment for IoT devices. In more details, we developed a brand new backbone module using residual convolutional network that balances between efficiency and accuracy. The neck was also redesigned to aggregates features with different granularity while guarantees the processing speed and low computation requirement. We also applied a data augmentation in training to make the model robust to adversarial factors such as illumination noises and small objects problem. We constructed a dataset using real objects and hardware devices in our wireless robot charging system, and also used the popular external COCO dataset to fully evaluate our system. The extensive experiments justified the solid performance of our framework with on-par accuracy with the SOTA baselines while is much more efficient in terms of processing time and computing power. Our technique is also robust to various adversarial factors such as brightness and image resolution.

ACKNOWLEDGEMENTS

This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT). (No. IITP2020-0-00618, Development of commercialization technology for ultra small, high efficiency wireless charging for 1kW class robot).