Insulator Detection via CNN for UAS Onboard Computers

Alif Ijlal Wafi[1], Rinaldi Munir[2]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: alif.iw@outlook.com[1], rinaldi@informatika.org[2]

***Abstract*—This paper proposes the usage of single-stage CNN models for detecting insulators in aerial images and measures their applicability in low-power computing settings that often found in UAS onboard systems. In addition to methods in literature, we also design another network based on YOLOv2 modified with SPP (spatial pyramid pooling) block and CIoU loss as our baseline. Our results shows that while both using SPP block and optimizing the bounding box regression function increases the overall detection accuracy without significant cost, network architectures that is specifically designed for edge devices are much more suitable on said environments. One of such design is SF-YOLO, with computation cost of 3,842 BFLOP (29% lower than YOLOv3 tiny, 86% lower than ours) while retaining AP_{50} score higher than 0.9, and thus can be further used for autonomous navigation subsystems with proper edge devices.**

***Keywords*—UAS; insulator; autonomous; object detection; deep learning; onboard**

I. Introduction

UAS has been utilized widely in power grid inspections due to its cost effectiveness. Recently, several studies have been made to further increase the workflow efficiency by automating the whole inspection procedure. Said systems are expected to be able to autonomously navigate the transmission lines with minimal instructions other than tower coordinates. To achieve this, the UAV should be able to visually detect transmission towers [1], power lines [2], and their individual components to make sure all points of interest are captured. Captured images could then be further analyzed in ground control systems readily available in many UAS-based inspection services. In these inspections, insulators are one of the critical components due to its nature as both the physical construct support and the insulation component in the electrical system.

Several insulator methods have been proposed in various literature, each with overlooked caveats that we would like to describe as follows. Binary images are used to localize insulators by detecting their pattern resemblance to insulator strings and discs, either heuristically [3] or with SVM based learning [4]. Hewever, it is often found that insulator aerial images are captured in angles such that the string inbetween insulator caps are not visible. This case is much prominent when inspecting disc faults, such as medial cracks and holes. Keypoint based methods are also used by BoW (bag of words) filtering and segmenting the remaining keypoints [5]. Because of its heavy reliance to high number of initial keypoints, and by extension, requires corner keypoint detectors, this method is especially hard to replicate in scale invariant settings such as aerial images. Active contour model is also proposed in [6] to snap initial contours to insulators. Aside from the fact that it shares the same weakness with binary images as neither of them cannot differentiate foreground objects by themselves, these models also needs intial, manual input to proceed and thus unsuitable for automation purposes.

CNN based methods were also proposed with VGG-16 [7] and YOLOv3 modifier with with ResNet-50 and SPP blocks [8], both reporting good performance in various conditions. However, [7] suffers from data over-augmentation (stitching 60 insulator foregrounds into 1056 backgrounds). Both studies also use desktop/server hardware which does not represent its applicability to onboard computers on UAV. This is especially important for applications in which live video transmission to ground control is not possible, such as rural areas without 4G coverage. In this paper, we propose a modified YOLOv2 as a baseline architecture for said onboard applications. Furthermore, our resulting benchmarks can also be used as a suitable reference for similar future research in low power computing environments.

II. Proposed Method

A. Dataset Construction

To optimize the CNN training, a proper insulator image dataset with varying types, sizes and colors is required. Background variety is also equally important to ensure the model could be operated in any inspection environments. The dataset is made using the data from EPRI [9], Opole [10], and CPLID [7] public dataset. The dataset is then randomly split with stratified holdout validation scheme into training and validation sets with a ratio of 9:1. This is due to variance discrepancy between datasets, such as similar backgrounds and/or single insulator type as described in figure 1. This dataset is then further augmented by $\pm 30°$ rotation and flipping in horizontal, vertical, and diagonal direction. Furthermore, we also collect image samples from national power grid company, PT PLN, as testing dataset to represent the model performance in the actual inspection scenario.

TABLE I. DATA USAGE FROM RESPECTIVE SOURCES

Source	Image count	
	Used	**Total**
EPRI	1600	1600
Opole	30	2630
CPLID	40	600
PLN	40	40

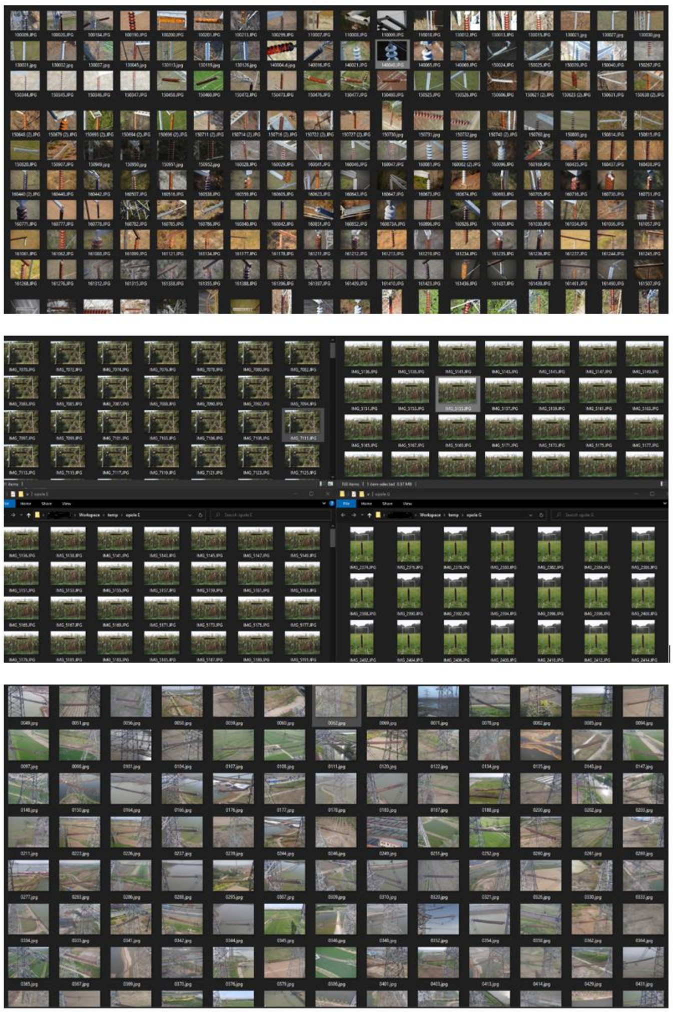


Fig. 1. From above, dataset source from EPRI, Opole, CPLID. Enlarge on the digital document for clarity.

B. *Modified YOLOv2 with SPP and CIoU*

CNN architectures derived from YOLO [11] are single stage models, meaning the the resulting output layer consists both the bounding box of detected objects and their class probability. On the contrary, double stage models such as Faster-RCNN [12] separate the region proposal and class inference into different networks. Hence, single stage models are favored in cases that needs faster inference time while retaining passable detection accuracy. Figure 2 illustrates the single stage model.

We chose YOLOv2 [13] as the starting point of our baseline architecture due to its significantly smaller layer depth compared to its successors, namely YOLOv3 [14] and YOLOv4 [15]. Since YOLOv2 and upwards are all aimed to be able to handle datasets consisting thousands of object classes, using earlier versions of these architectures should not mean it would perform significantly worse in our single object class problem. We then modify YOLOv2 to include SPP block at the end of feature extraction as described in table II. Referring to the work of [16], the addition of these pooling blocks are meant to increase the network performance not by merely deepening the network, but with combining local features from various scale spaces.

TABLE II. SPP BLOCK CONFIGURATION ON YOLOV2

No.	Type	Filters	Size/*Stride*	Output	Desc.
	Image input	-	-	416x416x3	
1	Darknet-19			13x13x1024	
2	Conv.	1024	3x3/1	13x13x1024	
3	Conv.	512	1x1/1	13x13x512	
4	Maxpool		5x5x1	13x13x512	SPP
5	Route 3			13x13x512	
6	Maxpool		9x9/1	13x13x512	
7	Route 3			13x13x512	
8	Maxpool		13x13/1	13x13x512	
9	Route 3, 4, 6, 8			13x13x2048	SPP-end
10	Route-15			26x26x512	
11	Conv.	64	1x1/1	26x26x512	
12	Reorg.		/2	13x13x256	
13	Route 12, 9			13x13x2304	
14	Conv.	1024	3x3/1	13x13x1024	
15	Conv	30	1x1/1	13x13x30	YOLO detection

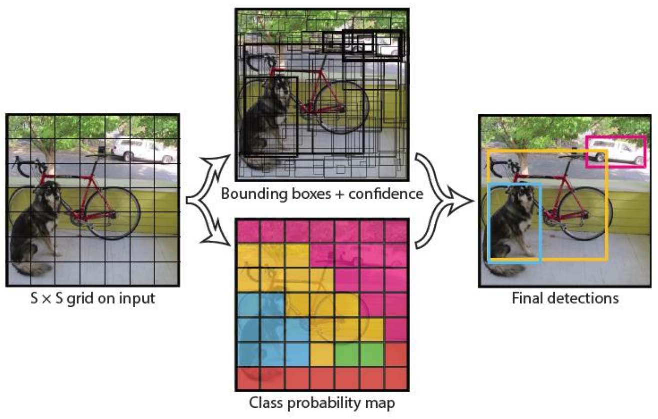


Fig 2. YOLO function model as a single stage function [11].

The default bounding box regression function often found in YOLO implementations is known as IoU loss (1), which is a simple loss function based solely on an overlap ratio between predicted box and the ground truth. We instead used a recently developed CIoU loss function [17] which handles cases such that these bounding boxes does not intersect. Furthermore, CIoU loss optimizes the regression by improving the penalty function and taking aspect ratio difference into account. By defining C as the minimum bounding box that covers both prediction and ground truth boxes, v as the aspect ratio metric, and α as the tradeoff parameter which prioritizes regression to overlapping boxes, CIoU can be written as follows:

$$L_{IoU} = 1 - \frac{|pred \cap grtruth|}{|pred \cup grtruth|} \tag{1}$$

$$C_{IoU} = 1 - IoU + \frac{(dist|center(pred) \cap center(grtruth)|)^2}{(diag(C))^2} + av \tag{2}$$

$$v = \frac{4}{\pi^2}(arctan\frac{width_{grtrut\ h}}{height_{grtrut\ h}} - \frac{width_{pred}}{height_{pred}})^2 \tag{3}$$

$$a = \frac{v}{(1 - IoU) + v} \tag{4}$$

III. Evaluation

A. Implementation Environment

Both our proposed baseline network and others described in literature are implemented in Darknet framework. Being the framework on which YOLO was born, Darknet trivializes the network setup with its configuration file systems. The experiment can also be reproduced as easily via exporting said files. Google Colabs environment was used to train and evaluate the networks, while their performance benchmarks were measured in a personal laptop with NVIDIA GTX950m and a single board computer Nanopi-M4. Aside from the fact that Google Colabs is a free cloud service in which available GPU differs from time to time, the hardware mentioned is much closer to the onboard computers. Respectively, they both represent Mini-PCs and entry level onboard computers often carried by UAV.

B. Evaluation Metric

The networks are measured by their model accuracy and performance in low power computing environment. AP (average precision) is used as the overall correctness measure between resulting and ground truth bounding boxes. Following the COCO evaluation metric [18], we retrieve AP in multiple IoU threshold to compute the final AP score. IoU can be described as overlap-union ratio between bounding boxes:

$$IoU = \frac{detection\ \cap ground\ truth}{detection\ \cup ground\ truth} \tag{5}$$

For instance, an IoU threshold of 1.00 would mean only positive detections with the exact box coordinates as the ground truth would count as a true positive. The final AP score is then computed as the average of AP taken in IoU threshold of 0.50 to 0.95:

$$AP = \frac{1}{10}\begin{pmatrix} AP_{50} + AP_{55} + AP_{60} + AP_{65} + AP_{70} + \\ AP_{75} + AP_{80} + AP_{85} + AP_{90} + AP_{95} \end{pmatrix} \tag{6}$$

With AP_{50} or AP@0.50 denotes the average precision on 0.50 IoU threshold. These scores are measured in each threshold by averaging precision numbers on the PR-curve, with the recall axis being the confidence score:

$$AP@IoU = \frac{1}{11}\sum_{recall\ \in(0,0.1,..0.9,1)} precision(recall) \tag{7}$$

Figure 3 is an example of a PR-curve in the literature. It should be noted that earlier studies often measure average precision only on the 0.50 IoU threshold. While AP_{50} should be enough in most usage cases such as autonomous navigation, it does not represent the overall model performance. Furthermore, strict measures such as IoU $\geq$ 0.75 might be useful to other cases, such as whether the model output can be directly fed to a fault detection system.

The performance benchmark of each network architecture is measured by its inference time on each image frame, memory requirement, and floating-point operations number. The inference time is then compared to the minimum visual navigation standard used in many robotic applications which is 30 FPS, or 33ms for each frame. Floating-point operations (FLOP) and RAM usage can then be used as a feasibility reference to whether the network can be applied to onboard systems, and by extension as a reference to whether the hardware used is suitable to use CNNs reliably.

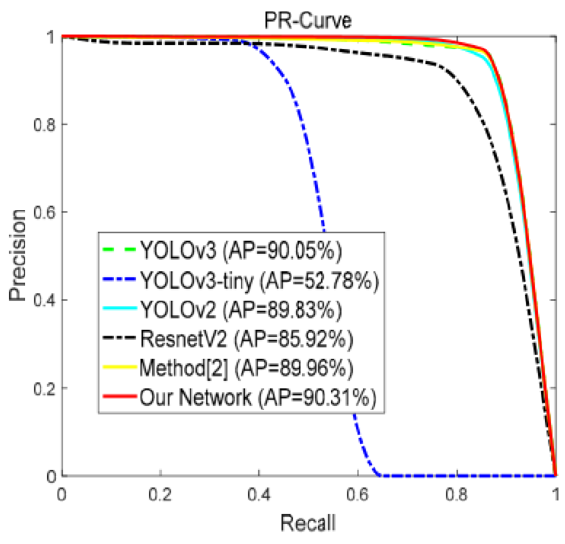


Fig 3. A sample PR-curve in the work of [8] with an IoU threshold of 0.5.

C. *Experiment Results*

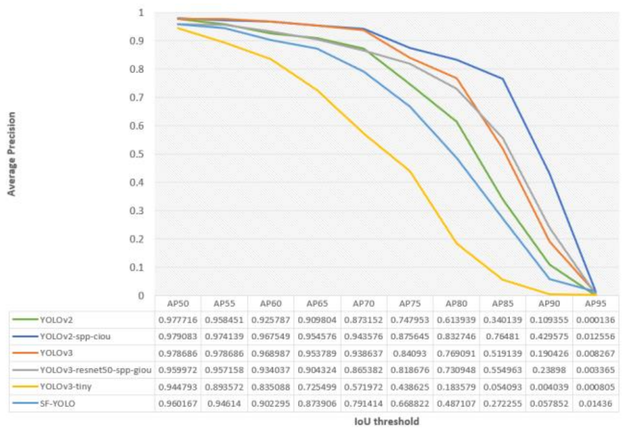
We compared our baseline network to the architectures found in [8], namely YOLOv2, YOLOv3, YOLOv3-ResNet50, and YOLOv3-tiny. For a fair comparison to the YOLOv3-tiny network, we add another comparison with SF-YOLO medium [19] as a network specifically designed for use in embedded environments.

TABLE III. VARIOUS NETWORK MODEL METRICS IN TRAINING AND VALIDATION SETS

Architecture	AP	FLOP (*Bn.*)	RAM	Inference Time	
				T864 (s)	950m (ms)
[13]	0.645	29.338	950MB	33.3	83.2
YOLOv2-spp-cIoU	0.773	29.539	970MB	33.2	86.8
[14]	0.714	65.304	1.5GB	64.3	172.4
[8]	0.696	64.506	1.8GB	65.1	188.6
YOLOv3-tiny	0.465	5.448	271MB	5.5	23.2
[19]	0.597	3.842	218MB	4.1	22.2

Results in table III shows that both the addition of SPP block and moving the bounding box regression function to CIoU results in increased accuracy without significant additional load. Figure 4 also shows that the proposed baseline architecture performs well even in a strict IoU threshold of 0.75, which means the output of said network can potentially be fed directly to insulator fault detection systems. However, both YOLOv2 and YOLOv3 derived networks are deemed not suitable to use in onboard environments. While their memory requirement can be handled by onboard computers, both do not satisfy the minimum FPS required on visual based navigation systems. Hence, small architectures such as YOLOv3-tiny and SF-YOLO is favored in this context. Unlike the results shown in [8], we found that YOLOv3-tiny performs well in IoU threshold of 0.50.

As shown by [19], SF-YOLO can run faster while having an even better detection performance than YOLOv3-tiny due to its deeper network, consisting of 46 convolution layers. This is done by limiting the filter depth on each layer by 256 or less, which is the upper limit of number of GPU cores found in edge devices. Our benchmark results on the single board computer does not able to achieve this however, because the MALI-T864 is a basic video processor unit as opposed to edge devices with dedicated GPU for deep learning. The difference between our hardware and [19] is compiled in table IV.



	AP50	AP55	AP60	AP65	AP70	AP75	AP80	AP85	AP90	AP95
YOLOv2	0.977716	0.958451	0.925787	0.909804	0.873152	0.747953	0.613939	0.340139	0.109355	0.000136
YOLOv2-spp-ciou	0.979083	0.974139	0.967549	0.954576	0.943576	0.875645	0.832746	0.76481	0.429575	0.012556
YOLOv3	0.978686	0.978686	0.968987	0.953789	0.938637	0.84093	0.769091	0.519139	0.190426	0.008267
YOLOv3-resnet50-spp-giou	0.959972	0.957158	0.934037	0.904324	0.865382	0.818676	0.730948	0.554963	0.23898	0.003365
YOLOv3-tiny	0.944793	0.893572	0.835088	0.725499	0.571972	0.438625	0.183579	0.054093	0.004039	0.000805
SF-YOLO	0.960167	0.94614	0.902295	0.873906	0.791414	0.668822	0.487107	0.272255	0.057852	0.01436

Fig. 4. Average precision measurement in various IoU threshold. Enlarge the digital document for clarity.

TABLE IV. COMPARISON OF THE AUTHORS' HARDWARE TO PROPER EDGE DEVICES

	Nanopi-M4v1	Jetson NANO	Jetson TX2	Jetson Navier NX
Driver	OpenCL	CUDA	CUDA	CUDA
GPU	4-core Mali Midgard, @650MHz	128-core NVIDIA Maxwell, @921MHz	256-core NVIDIA Pascal, @1300MHz	384-core NVIDIA Volta, @1866 MHz
RAM	4GB	4GB	8GB	8GB
Throughput (FP16)	80 GFLOPs	472 GFLOPs	1.3 TFLOPs	2.8 TFLOPs
SF-YOLO Inference	4.1s	29.4ms	12.9ms	12.8ms

D. *Detection on PLN Dataset*

The image dataset from the PT PLN is used to represent the model performance on unseen data in our local environment setting. Evaluation is then conducted after retraining the networks with both training and validation dataset. Following the metrics measured in the training phase, insulator locations in aerial images taken in Jakarta, Indonesia can also be inferred by the network.

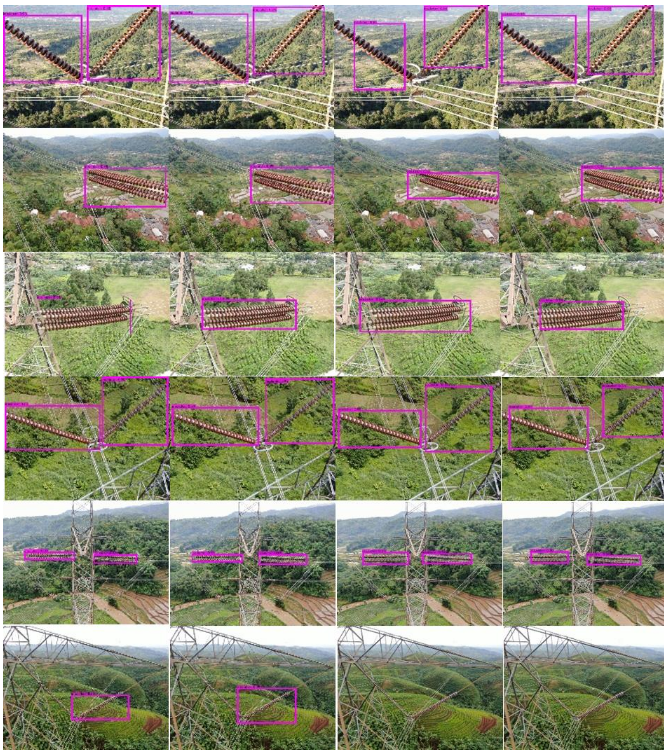


Fig. 5. Detection results in PLN dataset, from left: YOLOv2, our baseline network, YOLOv3-tiny, SF-YOLO. Enlarge the digital document for clarity.

Figure 5 shows that the usage of CIoU loss function results in bounding boxes that is much closer to the actual object compared to IoU loss. However, all models cannot reliably detect insulators that are excessively smaller compared to the rest of the image, mainly due to shooting distance. This is caused not only by the dataset largely consisting of insulators being close enough to inspect the faults visually, but also by the number of object scales that can be detected by CNNs (For instance, YOLOv2 have 2 scales, YOLOv3 and YOLOv4 have 3 scales) to limit the computation load while ensuring enough generalization. Aside from these limitations, the networks can still be used for autonomous navigation purposes when used in tandem with power line detection systems, as insulators should be directly attached into these cables.

IV. Conclusion

Insulator on aerial images can be detected using onboard computers with CNN based methods. However, our results shows that it is essential to use proper devices with edge GPU. The network architecture used is equally important to ensure its feasibility to the target hardware. When designing such networks for use in onboard computers, we recommend a rule-of-thumb of computation load being no more than 6GFLOP, while having filter depth of less than the number of cores available in the targeted edge GPU. The usage of SPP blocks and CIoU regression function is also recommended to maximize the network performance and to compensate the lack of feature depth within each layer.

The object detection method both on this paper and the earlier literatures only presents on how to identify insulator locations in aerial images. However, to ensure every required shooting angle is achieved, additional orientation classes are needed (e.g., insulator-top, insulator-side for horizontally oriented string insulators). The detection method could also be further expanded for other components in transmission tower when public datasets become available in the future.

Acknowledgement

The authors would like to thank PT. PLN for their helpful information regarding inspection practices and resulting image dataset on electrical transmission systems in Indonesia.

Reference

[1] A. Ceron, I. Mondragon and F. Prieto, "Real-time Transmission Tower Detection from Video Based on A Feature Descriptor," *IET Computer Vision,* 2016.

[2] Y. Zhang, X. Yuan and S. Chen, "Automatic Power Line Inspection Using UAV Images," *Remote Sensing,* 2017.

[3] J. Han, Z. Yang, Q. Zhang, C. Chen, H. Li, S. Lai, G. Hu, C. Xu, H. Xu, D. Wang and R. Chen, "A Method of Insulator Faults Detection in Aerial Images for High-Voltage Transmission Lines Inspection," 2019.

[4] Z. Zhao, G. Xu and Y. Qi, "Representation of Binary Feature Pooling for Detection of Insulator Strings in Infrared Images," *IEEE Transactions on Dieletcrics and Electrical Insulation vol. 23,* 2016.

[5] S. Liao and J. An, "A Robust Insulator Detection Algorithm Based on Local Features and Spatial Orders for Aerial Images," *IEEE Geoscience and Remote Sensing Letters vol. 12,* 2015.

[6] Q. Wu and J. An, "An Active Contour Model Based on Texture Distribution for Extracting Inhomogeneous Insulators From Aerial Images," *IEEE Transactions on Geoscience and Remote Sensing vol. 52,* 2014.

[7] X. Tao and D. Zhang, "Detection of Power Line Insulator Defects Using Aerial Images Analyzed Wirh Convolutional Neural Networks," *IEEE Transactions on Systems, Manm and Cybernetics: Systems vol. 50,* 2018.

[8] J. Han, Z. Yang, H. Xu, G. Hu, C. Zhang, H. Li, S. Lai and H. Zeng, "Search Like an Eagle: A Cascaded Model for Insulator Missing Faults Detection in Aerial Images," 2020.

[9] E. P. R. Institute, "Insulator Defect Image Dataset v1.1," 07 02 2020. [Online]. Available: https://www.epri.com/research/products/000000003002017949/. [Accessed 2021].

[10] M. Tomaszewski, B. Ruszczak and P. Michalski, "The collection of images of an insulator taken outdoors in varying lighting conditions with additional laser spots," *Data in Brief,* 2018.

[11] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2015.

[12] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," 2015.

[13] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," 2016.

[14] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," 2018.

[15] A. Bochkovskiy, C.-Y. Wang and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," 2020.

[16] Z. Huang and J. Wang, "DC-SPP-YOLO: Dense Connection and Spatial Pyramid Pooling Based YOLO for Object Detection," 2019.

[17] Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye and D. Ren, "Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression," 2019.

[18] T.-Y. Lin, M. Maire, B. Serge, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick and P. Dollar, "Microsoft COCO: Common Objects in Context," 2015.

[19] B.-G. Han, J.-G. Lee, K.-T. Lim and D.-H. Choi, "Design of a Scalable and Fast YOLO for Edge-Computing Devices," *Sensors,* 2020.