JOIV : Int. J. Inform. Visualization, 6(3) - September 2022 726-735

INTERNATIONAL JOURNAL ON INFORMATICS VISUALIZATION

journal homepage : www.joiv.org/index.php/joiv

Exploration of The Impact of Kernel Size for YOLOv5-based Object Detection on Quadcopter

Rissa Rahmania [a], Felix Corputty [b], Suryo Adhi Wibowo [b,*], Dany Eka Saputra [a], Annisa Istiqomah [a]

[a] *Computer Science Departement, School of Computer Science, Bina Nusantara University, Bandung Campus, Jakarta, Indonesia*
[b] *School of Electrical Engineering, Telkom University, Bandung, Indonesia*
Corresponding author: [*]suryoadhiwibowo@telkomuniversity.ac.id

***Abstract*— Drones or quadcopters have been widely used in various fields based on deep learning, especially object detection. However, drone vision characteristics such as occlusion and small objects are still being explored for performance in terms of accuracy and speed detection. The YOLO architecture is very commonly used for cases requiring high-speed detection. To overcome the limitations of drone vision, in this paper, we explore the size of the YOLOv5s backbone kernel in the shallowest convolutional layer to achieve better performance. The kernel is a filter that has a main role in the feature map, and it defines the size of the convolution matrix, and the resulting features in the shallowest convolutional layer are more representative of the case of object detection and recognition. The techniques can be divided into three major categories: (1) data preprocessing, which involves augmentation and normalization of the data, (2) kernel size exploration in the shallowest convolutional layer of the YOLOv5s, and (3) model implementation in the real environment using the quadcopter. The dataset consisted of four classes representing dragon fruit, snake fruit, banana, and pineapple, with a total of 8000 data. Exploration results with kernel size give promising results. Kernel sizes 5 and 7 give an *mAP* of 0.988. Through these results, modification of the kernel size provides an opportunity for more in-depth investigations, such as with the epoch parameter, padding scheme, and other optimization techniques.**

***Keywords*— YOLOv5; object detection; kernel size; quadcopter; deep learning.**

Manuscript received 1 May. 2022; revised 19 Jun. 2021; accepted 12 Jul. 2022. Date of publication 30 Sep. 2022.

I. INTRODUCTION

An unmanned aerial vehicle (UAV), known as a drone, is an airplane with a controlling system and its mechanism using a remote control or person controlled independently by a computer. Internet of Things (IoT) networks in the future of 6G communications require drones to act from sensors until performing the required actions [1]. Drones are an important research topic, and in recent years drone research continues to be carried out in various research areas such as positioning, navigation, controls, imaging, communications, sensors, materials, batteries, circuits, and motors. In the UAV platform, the camera is used as a sensor part in both Fixed-wing, Helicopter, Quadcopter, and Octocopter [2].

In the field of imaging, computer vision on drones has become an interesting topic because it has been proven that it can be used in various application fields such as reconnaissance [3], surveillance [4], agriculture [2], and site mapping [5], and others. Research on matters that are closely related to drone needs has actually been entirely researched, such as visual tracking [6], [7], along with the optimization of one of the methods, Histogram of Oriented Gradients (HOG) [8]. However, challenging drone vision characteristics are still issues, and optimal solutions are still needed. Issues related to drone characteristics on UAV datasets are small objects [9][10], occlusion [11], moving objects [12], and data augmentation [7]. This topic continues to be explored for implementing the research goal of an effective and efficient autonomous drone.

To improve detection performance in drone vision, research on real-time object detection has been accomplished, such as by Zhang et al. [13] developing SlimYOLOv3 to overcomdrones' limited memory and computing power. In addition, Wu et al. [14] modified YOLOv3 to detect cars, trucks, buses, and pedestrians for effective and robust implementation in real-time object detection. While missed detection still occurred. In other research by Zhang et al. [15], overcoming issues from drones by performing keyframe extraction, Baykara et al. [16] implemented TinyYOLO for efficient multiple object detection, and Lee et al. [17] involved cloud servers by using Fast YOLO on local and

Faster RCNN on a cloud. However, an error occurred during real-time testing, which implies non-effectiveness for drones in real-time conditions. Then, to achieve better metrics, in different cases, Wiranata et al. [18] performed a padding scheme technique to detect vehicles using the AlexNet architecture. However, satisfactory accuracy does not meet real-time object detection requirements. The architecture involves a convolutional neural network (CNN) eight layers deep.

In order to conquer real-time performance, the one-stage detectors are used in much previous research, specifically deep learning for drone vision in real-time object detection. You Only Look Once (YOLO) [19], [20] is one of the methods that is widely used because it predicts the object in a single step without using region proposals. However, optimal accuracy still needs to be explored with all kinds of techniques. Because of this fact, exploration of kernel size and stride in the YOLOv5s backbone are proposed. The rest of this paper is organized as follows. Section II describes the materials and the proposed method. The results and discussion are described in Section III. Section IV provides some concluding remarks.

II. Materials and Method

This section describes the quadcopter, deep learning, and transfer learning concepts. The quadcopter is used in this research, and the YOLOv5s method is implemented for object detection. The basic concept of deep learning with a Convolutional Neural Network (CNN) and transfer learning for using the method as a pre-trained model is explained.

A. Quadcopter

Historically, drones have been widely used for the military, which later increased their use in non-military applications. In Fig.1, it is shown that UAVs based on their wings, are divided into two types, namely Fixed-Wing and Rotary-Wing. The easiest example of Fixed-Wing is an airplane. Meanwhile, the quadcopter is one of the UAV platform types with Rotary-wing, which is included in the multi-rotor type [21]. Compared to other types of UAVs, the quadcopter is relatively small and carries a small payload, around 1.25 kg. This type is usually used for mapping and reconnaissance.

Quadcopter has many types of sensor preferences in the camera section. Some of them are RGB cameras, Thermal cameras, Multi-spectral cameras, and Visible-light cameras. This type of drone can also communicate using WiFi, Wireless Radio, and Xbee [2].

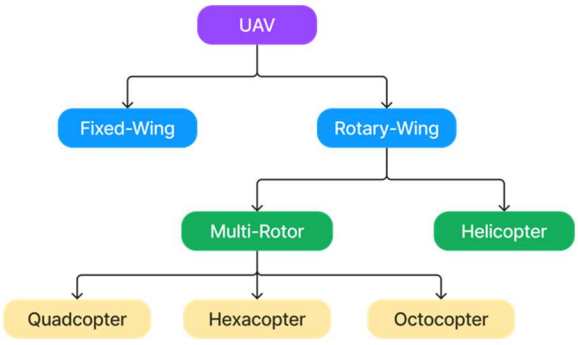


Fig. 1 Platform types of UAV.

The use of a quadcopter for object detection applications requires a computing platform installed onboard. The platform is able to use Arduino [22], Raspberry Pi, NVIDIA Jetson [23], and others. Therefore, several things that need to be considered in its implementation are the platform type, hardware components, and communications.

B. Deep Learning-based Object Detection

Convolutional Neural Network (CNN) is a method that is very commonly used specifically for Deep Learning. Deep learning is a part of Machine Learning that uses neural networks with deep structures to solve learning problems and the human brain system. The basic concept of deep learning methods is to perform a series of data science processes, from collecting datasets and labeling objects to conducting training, testing, and validation.

Datasets concerned with UAVs, such as VisDrone [24], MOR-UAV [25], and others, can be used openly to exp their objects' characteristics. Naturally, the bird-view taken from the drone camera brings up the characteristics of the object described in Section I. Then, the training process begins with filtering and continues with the convolution process. In the convolution process, each neuron receives an input, then performs a dot operation with *weight* and *bias*. In this process, *stride* and *padding* are determined. In addition, it involves *activation functions* that are in the hidden layer and output layer, which play a role in making decisions. Then in the pooling phase, the pixel image is selected using the average pooling or max-pooling technique [26]. Therefore, the system can perform classification, detection, or recognition.

Specifically for object detection, there are two types of detection schemes: wo-Stage Detector [27], [28] and a One-Stage Detector. Object detection with a Two-Stage Detector scheme has two stages: Region Proposal Network (RPN) and Classification Stage. R-CNN, Fast R-CNN, Faster R-CNN, and Cascade R-CNN are methods that are already used for UAV datasets [29]. In comparison, the One-Stage Detector concept can directly provide bounding box and object classification because it uses a single fully convolutional network. The well-known One-Stage Detector architectures are YOLO and SSD [30].

In this research, the architecture that has been built on the One-Stage Detector is used and then adapted and fine-tuned. This process is also known as transfer learning. To use the pre-trained model, it is necessary to first identify the characteristics of the architecture and the dataset that has been trained. In the adaptation process, load models are performed, and architectural modifications are made according to the data they have. At this stage, the architectural parameters and the resulting output need to be considered. For instance, adjusting the number of classes that can be detected by the architecture to be used. Since this information can be detected in the classifier step of architecture, it is possible to adapt the classifier to the dataset. Furthermore, retraining is carried out with a low learning rate in fine-tuning the selected architecture.

The transfer learning concept needs to ensure the overfit condition because the new dataset must apply to the pre-trained model. In this paper, the concept of transfer learning is performed on the YOLOv5s architecture. Therefore, this paper will focus on discussing YOLOv5s and the transfer

learning process, which specifically refers to the shallowest part of the YOLOv5s architecture.

C. YOLOv5

In contrast to the sliding window and region-based proposal techniques, the YOLO architecture processes entire parts of the image during the training and testing stages [19]. The basic concept of the YOLO algorithm is to execute the algorithm shown in Fig. 2. The input of YOLO is a preprocessed image along with the object labeling information. Preprocessed images include *resizing* and *augmentation* if needed. YOLO detects the coordinate of the objects with the help of a grid cell system that is set through a variable named *batch size*.

The cell will later help the system find out the location of the object in the image with the help of y_{train}, the result of object labeling. These results contain information on the P_c variable, which indicates the availability of an object in the image, B_x and B_y are the x dan y coordinates to determine the center point of the object. Then, B_w and B_H define the *width* and *height* of the bounding box. The variables C_1 and C_2 represent class information. If there is no object in the image, then the value of P_c would be zero, followed by other variables.

YOLOv5 is an extension of YOLOv4 [20]. The YOLOv5 has several types, specifically YOLOv5n, YOLOv5s, YOLOv5m, YOLOv5l, and YOLOv5x. The YOLOv5 implements a cross-stage partial (CSP) model based on a dense network (DenseNet), thus making this architecture a major contribution to inference speed and model size [31]. YOLOv5 is open-source and can be accessed at [32]. The training process is executed by involving y_{train} in the network from YOLOv5, which is shown in Fig. 2. The YOLOv5 architecture is divided into three parts, consisting of the *backbone*, *neck*, and *head*. In the *backbone* section, feature extraction is performed by CSPDarknet [33] and spatial pyramid pooling (SPP) [34].

Algorithm 1: Object Detection YOLO Algorithm

Input: preprocessed images (x_{train}), label object information (y_{train})
Output: image with a bounding box (*mAP*, *precision*, *recall*)
Step 1 (Divide image into grid cell system)
    Initialize step size on each image (x_{train})
Step 2 (Detect the center of object on each cell)
    Compute *y_train* ($P_c, B_x, B_y, B_h, B_w, C_1, C_2$)
        Detect object in the image (P_c)
        Detect height and weight of rectangle bounding box (B_h, B_w)
        Detect the location of the center point of each object in the cell of an image (B_x, B_y)
        Detect the class (C_1, C_2)
Step 3 (Training Process)
    Compute *y_train* into *Backbone*, *Neck*, and *Head*
Step 4 (Prediction Process)
    Compute *mAP*, *precision*, and *recall* on each bounding box with Non-Max Suppression

Fig. 2 Object detection of YOLO Algorithm.

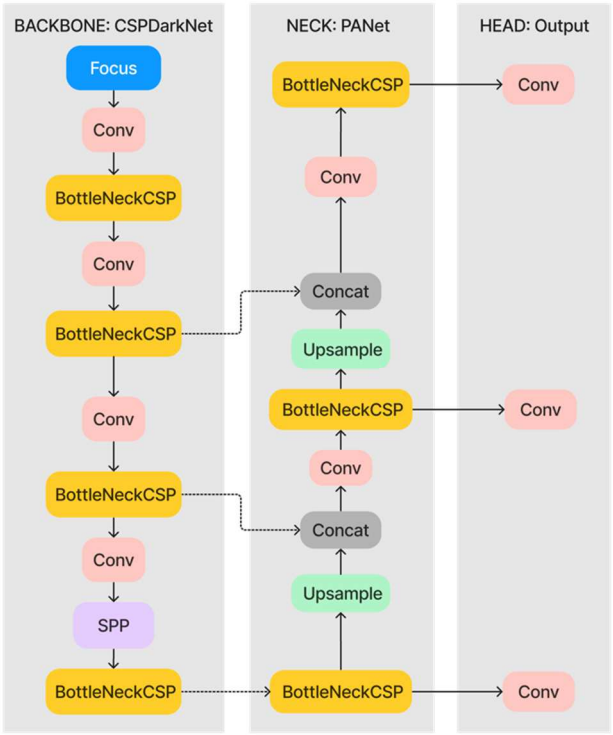


Fig. 3 Architecture of YOLOv5.

The role of CSPDarknet is to reduce the amount of computation. Where involves Partial Dense blocks, which are connected by transition layers. Then on the *neck*, the image feature is combined through several layers to arrive at a prediction. Then, in the *head* section, predictions of bounding boxes, class, coordinate, score, and size are made. This process is a one-forward pass stage that is not repeated.

In the concept of transfer learning that can be performed on the YOLOv5s architecture, the amount and the characteristics of the dataset that has been trained on the YOLOv5 architecture need to be considered. The YOLOv5 itself has been trained using COCO datasets [35] with the number of images per class ≥ 1500, 10000 labels per class, and variations in images based on time, weather, lighting, angles, and different cameras. If the new datasets are in a small number of data, where the characteristics are similar to those that have been trained in the pre-trained YOLOv5s model, the classifier step of the pre-trained model can be adjusted. Further, the backbone step can be improved or modified. Indeed, the thing to be avoided is small datasets with different characteristics from the dataset in the pre-trained YOLOv5s model. Therefore, the prediction results will be directly obtained through the *mean average precision* (*mAP*), *precision* (PR), and *recall* (RC) parameters with the Non-max Suppression technique [36].

D. Kernel Size

In CNN, the convolutional layer is defined by several parameters, such as *kernel* size, stride, and padding [37]. This paper will focus on *kernel* size and *stride* to improve performance. The *kernel* is a filter that has a main role in the feature map and defines the convolution size. In comparison, *stride* measures the kernel size shift in an image either downwards (rows) or to the right (columns). By default, the

stride is one. However, this can be adjusted to pass through more than one element. Fig. 3 shows the role of the kernel and *stride* in providing the output. In Fig. 4(a), the computation output in the lower leftmost cell is 0×0+6 ×1+0×2+0×3=6. While the output cell size [37] is obtained by using the equation,

$$\left\lfloor \frac{x_h - k_h + p_h + s_h}{s_h} \right\rfloor \times \left\lfloor \frac{x_w - k_w + p_w + s_w}{s_w} \right\rfloor \quad (1)$$

where $x, k, p, s, h,$ and w are the input image, *kernel*, padding, *stride*, height, and weight of matrix dimensions, respectively.

III. Results and Discussion

This section explains the proposed method, data systematics, and performance parameters. The trained YOLOv5s model will be implemented into a quadcopter. With this implementation, the quadcopter will perform the real-time inference process for object recognition. The data systematics used in this paper are described briefly, and the performance parameters used for system analysis will also be presented.

A. Proposed Method

In this subsection, the proposed method is explained. The quadcopter has the task of recognizing objects in the form of fruits, including *dragon fruit*, *snake fruit*, *banana*, and *pineapple*. For the system to detect and recognize these objects, the YOLOv5s method is used in this research. The initial process carried out is transfer learning by training on the pre-trained model from YOLOv5s. The training used in this paper uses a group of training data $I_t(x, y)$. The training data will be normalized to a size of 640×640 as input to the YOLOv5s *backbone*. Furthermore, in the *backbone*, there are several convolutional layers and CSP. While each step involves a matrix with a size of $h \times w \times c$, where h, w, and c are the height, weight, and class of the object. The convolutional layer serves as feature extraction, which will be processed in the next layer.

In this research, the *kernel* size exploration was carried out at the shallowest convolutional layer (shallow convolutional layer). The Shallow convolutional layer is the earliest convolutional layer. In this layer, the resulting features are more representative of the case of object detection and recognition. Therefore, in this layer, an exploration of the size of the kernel that affects the formation of convolutional features is carried out. Furthermore, the output from the layer will be processed in the next convolutional layer until the process in the *backbone* (CSPDarknet) is complete. The output of the CSPDarknet *backbone* process is then processed at the Path Aggregation Network Layer, which is the *neck* part of the YOLOv5s Architecture. Parameters and processes performed in the network layer use the default. From this process, the convolution process is then carried out again using a convolutional layer with a size of 256×85, 512×85, and 1024×85, which will then get the outputs of 20×20×255, 40×40×255, and 80×80×255, respectively.

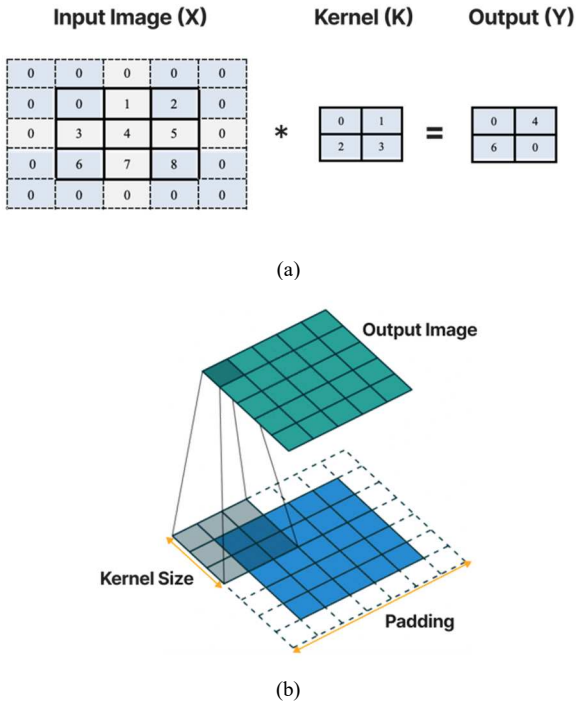


Fig. 4 Illustration of (a) convolution process with stride 3 and (b) kernel operation.

The model that has been trained is then stored and the loading process is carried out on a personal computer (PC) as a processor for the quadcopter. Using the API, the connection process between a PC and a quadcopter is done through WiFi intermediaries. After all these processes run well, then the inference process can be carried out in real-time. The framework of this research is shown in Fig. 5.

B. Data Systematic and Requirement System

In this subsection, data systematics and system requirements are explained. The dataset used in this research consisted of four classes: dragon fruit, snake fruit, banana, and pineapple, which were taken directly. Table 1 represents the dataset information used. The dataset that is used is a self-built dataset, which is taken by the quadcopter. This dataset then entered the data augmentation proses in the form of random affine (rotation, scale), mix-up, and copy-paste. Augmentation with a mix-up is performed by combining several classes in one image. In comparison, copy-paste applied the same image but changed the focus of taking objects on the captured image.

In addition, for the training process, from augmented processing, 2000 data for each class were obtained from the training process. With a total of 8000 data, to speed up the training process, use the NVIDIA DGX A-100 AI supercomputer with a Graphical Processing Unit (GPU) capacity of 40GB for each core.

TABLE I
NUMBER OF CLASSES IN THE DATASET

Class	Number
Dragon Fruit	104
Snake Fruit	197
Banana	130
Pineapple	163

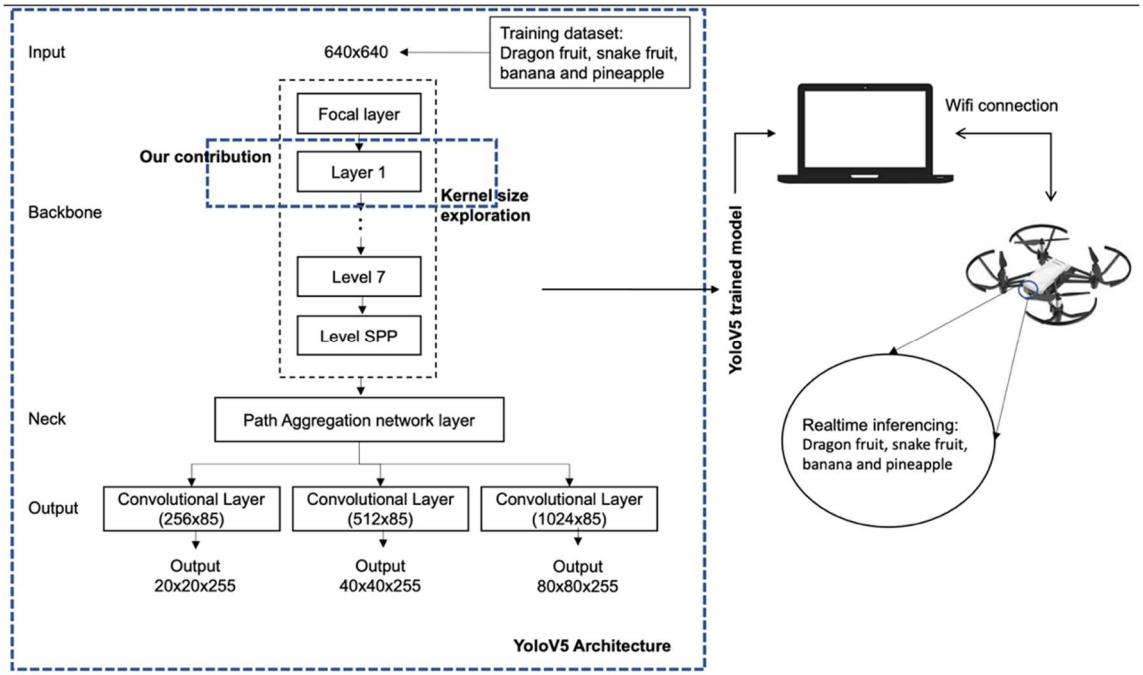


Fig. 5 Our proposed method framework.

Based on the results of the augmentation data and the type of object, this data with pre-trained YOLOv5s has a very similar dataset with a large number of datasets. Therefore, the purpose of transfer learning is to modify the backbone and fine-tune the model to improve performance. Furthermore, for the quadcopter used in this research, the DJI Tello 2 is used. The DJI Tello 2 is a low-level quadcopter with specifications for 5MP photos, 720p30 video, 98×92.5×41mm, and a battery capacity of 1190mAh with a voltage of 3.8volt. This quadcopter can be used indoors or outdoors and meets the quality requirements of laboratory-scale research. The kernel sizes used in this research were 5, 6, 7, 8, and 9 with *stride* 4. The illustration of this research is represented in Fig. 6.

C. Performance Parameter

To find out and analyze whether the model generated from this research is well performed or not, several performance parameters such as *precision* (2), *recall* (3), and *mAP* (4) [37] are used with the equation,

$$PR = \frac{TP}{TP+FP} \tag{2}$$

$$RC = \frac{TP}{TP+FN} \tag{3}$$

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i \, , AP = \int_0^1 p(r)dr \tag{4}$$

where $PR, RC, TP, FP, FN, N,$ and AP is *precision*, *recall*, *true positive*, *false positive*, *false negative*, number of classes, and mean average precision, respectively. For $p(r)$ is a function of *precision*. Precision is the ratio of *TP* to the sum of *TP* and *FP*.

TP is a condition where the objects have been labeled from the training data in an image, and the location of the labels is very similar to the bounding box generated from the training/validation/testing results. Meanwhile, *FP* is a condition where there are objects labeled from the training data in the image, and the labels' location is not the same as the bounding box generated from the results of training/validation/testing. Then, *FN* is a condition where objects should be detected, but there are no bounding boxes in the image.

D. Result

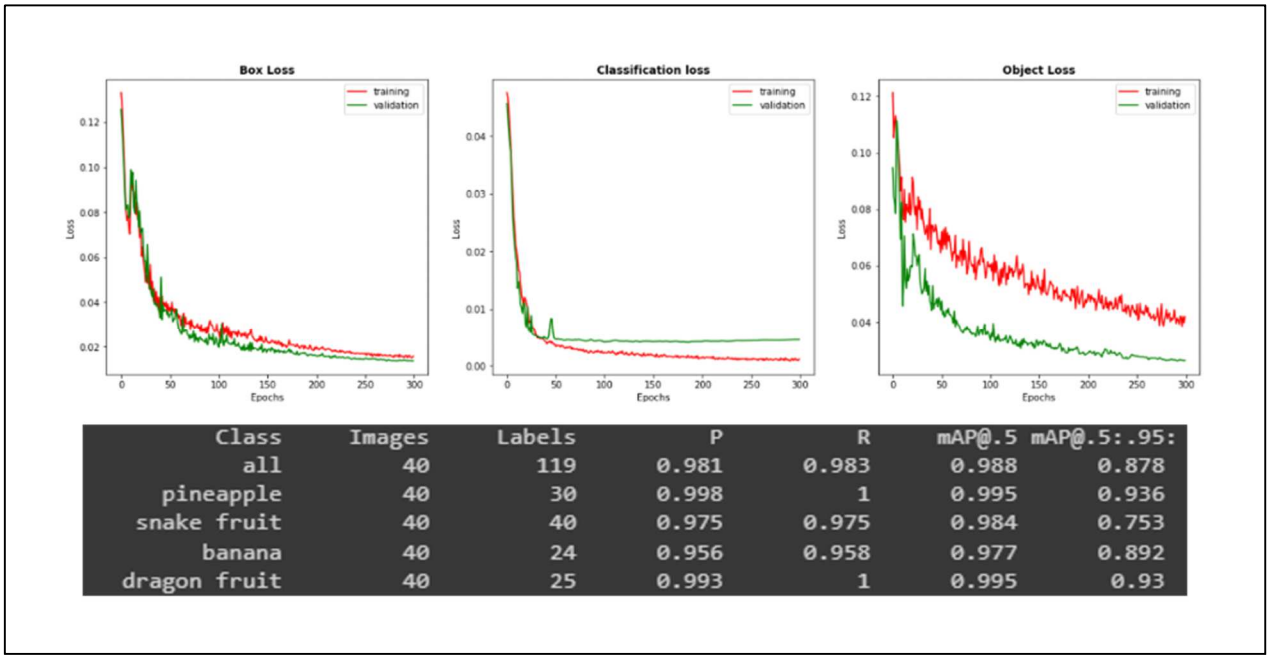
In the first experiment, the *kernel* size used was 5. Fig.7 represents the graphic results of the exploration consisting of *box loss*, *classification loss*, and *object loss*. The three graphics are the result of *loss* after doing 300 *epochs* and are marked with red and green lines for training and validation, respectively. In all three, it is shown that the *loss* value tends to continue to decrease. This graphic shows a very well performance. The results of these three graphics also occur in Fig. 7-Fig. 12.

From these results, *kernel* size 5 has excellent performance with *precision*, *recall*, and *mAP* values of 0.981, 0.983, and 0.988, respectively. While for the second experiment using *kernel* size 6, the results for *precision*, *recall*, and *mAP* were 0.981, 0.983, and 0.978, respectively. These results can be seen in Fig. 8. In Fig. 9, the results of the third experiment using *kernel* size 7 are represented.

Fig. 6 Our research implementation in real environment

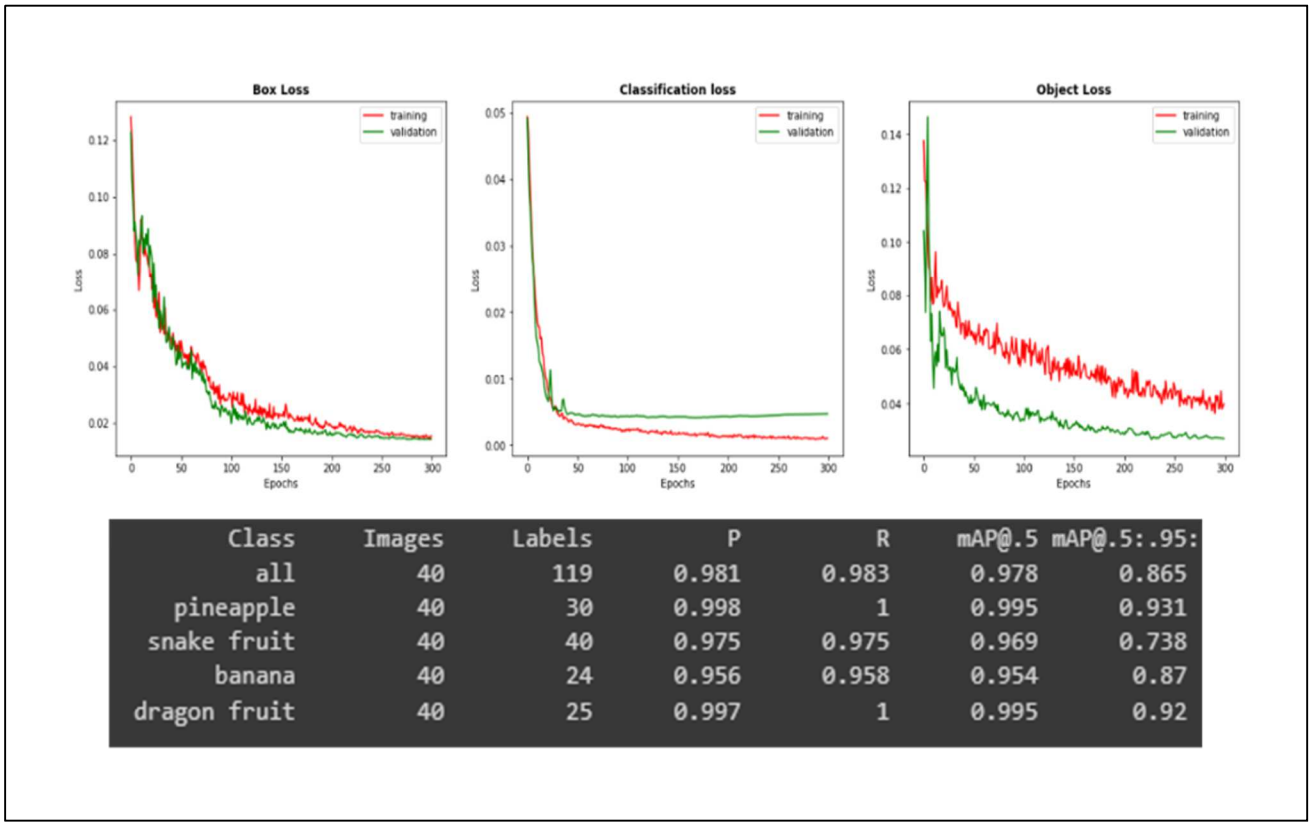
The results obtained from the experiment are the *precision*, *recall*, and *mAP* values are 0.981, 0.983, and 0.988,

respectively. For *kernel* size 8, the results are shown in Fig. 10, the *precision*, *recall*, and *mAP* values are 0.982, 0.983, and 0.984, respectively. In the last experiment, using *kernel* size 9, the values of *precision*, *recall*, and *mAP* were 0.978, 0.983, and 0.988, respectively. These results can be seen in Fig. 11. While using the default YOLOv5s, the *precision*, *recall*, and *mAP* results obtained are 0.982, 0.983, and 0.980. These results are represented in Fig. 12. Based on the results shown in Fig.7-Fig.12, the precision, recall, and mAP values show excellent results. The selection of the best kernel is based on the *mAP* score, where the best *mAP* value is generated when the *kernel* size is 5 and 7, which is 0.988.



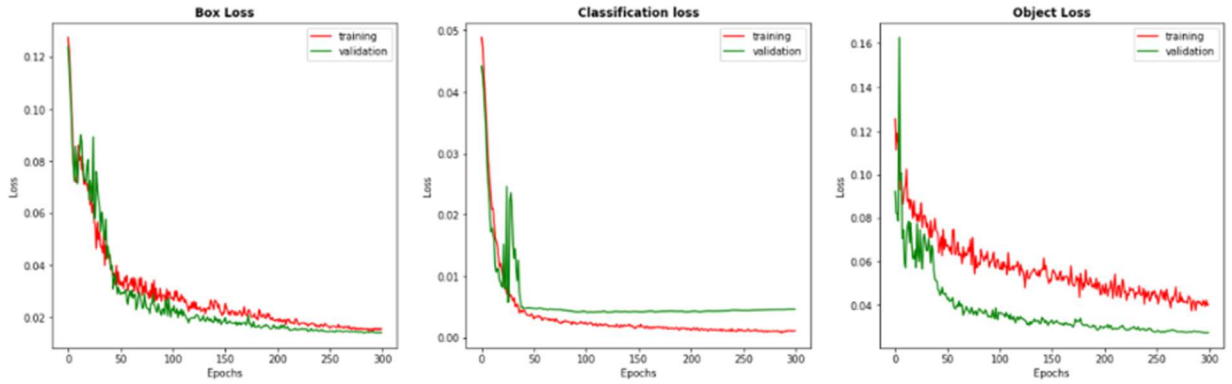
Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95:
all	40	119	0.981	0.983	0.988	0.878
pineapple	40	30	0.998	1	0.995	0.936
snake fruit	40	40	0.975	0.975	0.984	0.753
banana	40	24	0.956	0.958	0.977	0.892
dragon fruit	40	25	0.993	1	0.995	0.93

Fig. 7 Box loss, classification loss, object loss, performance parameters of precision (P), recall (R) and mAP: kernel size (5) and stride (4)



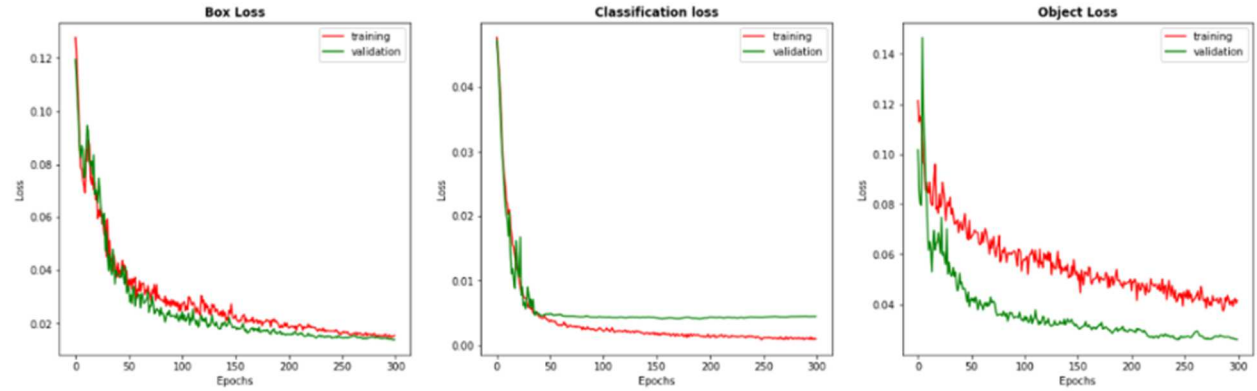
Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95:
all	40	119	0.981	0.983	0.978	0.865
pineapple	40	30	0.998	1	0.995	0.931
snake fruit	40	40	0.975	0.975	0.969	0.738
banana	40	24	0.956	0.958	0.954	0.87
dragon fruit	40	25	0.997	1	0.995	0.92

Fig. 8 Box loss, classification loss, object loss, performance parameters of precision (P), recall (R) and mAP: kernel size (6) and stride (4)



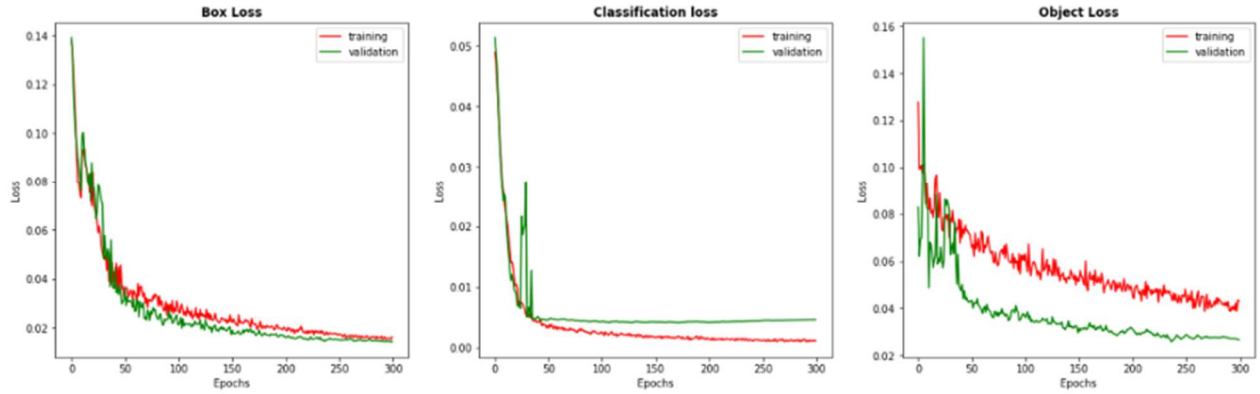
Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95:
all	40	119	0.981	0.983	0.988	0.876
pineapple	40	30	0.998	1	0.995	0.931
snake fruit	40	40	0.975	0.975	0.974	0.739
banana	40	24	0.956	0.958	0.986	0.908
dragon fruit	40	25	0.996	1	0.995	0.925

Fig. 9 Box loss, classification loss, object loss, performance parameters of precision (P), recall (R) and mAP: kernel size (7) and stride (4)



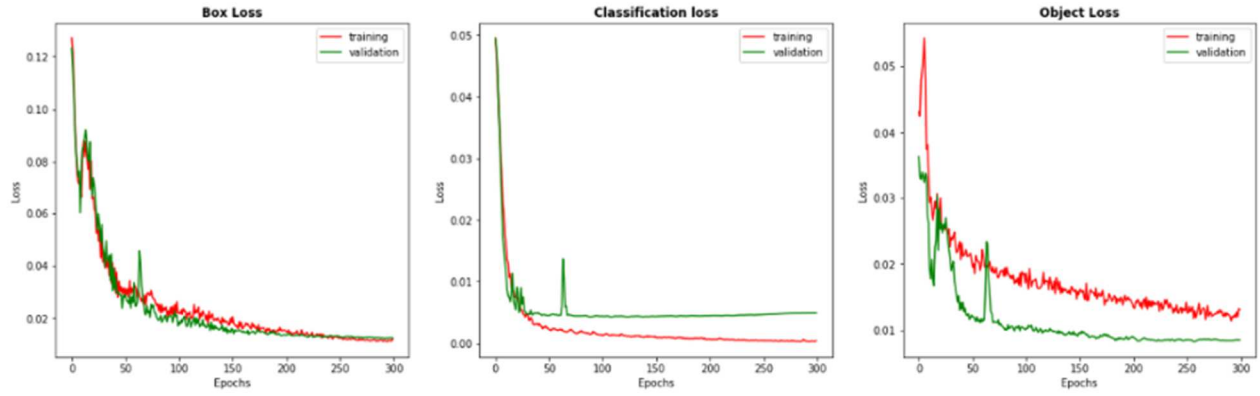
Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95:
all	40	119	0.982	0.983	0.984	0.874
pineapple	40	30	0.999	1	0.995	0.944
snake fruit	40	40	0.975	0.975	0.987	0.736
banana	40	24	0.956	0.958	0.961	0.908
dragon fruit	40	25	0.998	1	0.995	0.906

Fig. 10 Box loss, classification loss, object loss, performance parameters of precision (P), recall (R) and mAP: kernel size (8) and stride (4)



Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95:
all	40	119	0.978	0.983	0.988	0.88
pineapple	40	30	0.998	1	0.995	0.918
snake fruit	40	40	0.975	0.975	0.978	0.759
banana	40	24	0.955	0.958	0.982	0.912
dragon fruit	40	25	0.985	1	0.995	0.932

Fig. 11 Box loss, classification loss, object loss, performance parameters of precision (P), recall (R) and mAP: kernel size (9) and stride (4)



Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95:
all	40	119	0.982	0.983	0.98	0.874
pineapple	40	30	0.998	1	0.995	0.93
snake fruit	40	40	0.974	0.975	0.981	0.763
banana	40	24	0.956	0.958	0.948	0.888
dragon fruit	40	25	1	1	0.995	0.917

Fig. 12 Box loss, classification loss, object loss, performance parameters of precision (P), recall (R), and mAP: YOLOv5 Default

IV. Conclusion

We have experimented with kernel size to improve object detection performance for drones. Kernel size with an odd size shows better results than kernel size with an even size. While from the experiments carried out in this research, kernel sizes 5 and 7 gave optimal results with *precision*, *recall*, and *mAP* values of 0.981, 0.983, and 0.988, respectively. Compared to the default from YOLOv5s, our proposed method has *mAP* advantage of 0.008, close to 0.01. The results of the *box loss*, *classification loss*, and *object loss* graphs for kernel sizes 5 and 7 also give better results, with the difference between the training and validation lines having

a shape similar to a small error gap. Therefore, the graph does not represent the overfitting curve.

Based on the analysis, kernel sizes 5 and 7 have a better impact on performance than the default network. Modification of kernel size plays a role in the feature map process. The YOLOv5s can be optimized for speed detection needs. Moreover, this experiment can be expanded by experimenting with a more in-depth investigation, such as the epoch parameter, padding scheme, kernel design for detecting edges, and other optimization techniques.

REFERENCES

[1] R. Shrestha, R. Bajracharya, and S. Kim, "6G Enabled Unmanned Aerial Vehicle Traffic Management: A Perspective," *IEEE Access*, vol. 9, pp. 91119–91136, 2021, doi: 10.1109/ACCESS.2021.3092039.

[2] J. Kim, S. Kim, C. Ju, and H. il Son, "Unmanned aerial vehicles in agriculture: A review of perspective of platform, control, and applications," *IEEE Access*, vol. 7. Institute of Electrical and Electronics Engineers Inc., pp. 105100–105115, 2019. doi: 10.1109/ACCESS.2019.2932119.

[3] Z. Liu, C. Liu, W. Zhao, and A. Li, "A User-Priority-Driven Multi-UAV Cooperative Reconnaissance Strategy," *International Journal of Aerospace Engineering*, vol. 2021, pp. 1–14, Oct. 2021, doi: 10.1155/2021/9504056.

[4] S. K. Niranjan, REVA University, Institute of Electrical and Electronics Engineers. Bangalore Section, and Institute of Electrical and Electronics Engineers, *Proceedings of the International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE 2020) : October 9-10, 2020, Virtual Conference*.

[5] W. Jiang, Y. Zhou, L. Ding, C. Zhou, and X. Ning, "UAV-based 3D reconstruction for hoist site mapping and layout planning in petrochemical construction," *Automation in Construction*, vol. 113, May 2020, doi: 10.1016/j.autcon.2020.103137.

[6] S. A. Wibowo, H. Lee, E. K. Kim, and S. Kim, "Collaborative Learning based on Convolutional Features and Correlation Filter for Visual Tracking," *International Journal of Control, Automation and Systems*, vol. 16, no. 1, pp. 335–349, Feb. 2018, doi: 10.1007/s12555-017-0062-x.

[7] S. A. Wibowo, H. Lee, E. K. Kim, and S. Kim, "Visual tracking based on complementary learners with distractor handling," *Mathematical Problems in Engineering*, vol. 2017, 2017, doi: 10.1155/2017/5295601.

[8] S. A. Wibowo, H. Lee, E. K. Kim, and S. Kim, "Convolutional Shallow Features for Performance Improvement of Histogram of Oriented Gradients in Visual Object Tracking," *Mathematical Problems in Engineering*, vol. 2017, 2017, doi: 10.1155/2017/6329864.

[9] M. Liu, X. Wang, A. Zhou, X. Fu, Y. Ma, and C. Piao, "Uav-yolo: Small object detection on unmanned aerial vehicle perspective," *Sensors (Switzerland)*, vol. 20, no. 8, Apr. 2020, doi: 10.3390/s20082238.

[10] X. Zhang, E. Izquierdo, and K. Chandramouli, "Dense and Small Object Detection in UAV Vision based on Cascade Network." [Online]. Available: http://www.goldmansachs.com/our-thinking/technology-driving-

[11] Z. Pi, Y. Lian, X. Chen, Y. Wu, Y. Li, and L. Jiao, "A Novel Spatial and Temporal Context-Aware Approach for Drone-Based Video Object Detection," 2020. doi: 10.1109/ICCVW.2019.00027.

[12] K. M. Abughalieh, B. H. Sababha, and N. A. Rawashdeh, "A video-based object detection and tracking system for weight sensitive UAVs," *Multimedia Tools and Applications*, vol. 78, no. 7, pp. 9149–9167, Apr. 2019, doi: 10.1007/s11042-018-6508-1.

[13] P. Zhang, Y. Zhong, and X. Li, "SlimYOLOv3: Narrower, Faster and Better for Real-Time UAV Applications." [Online]. Available: https://github.com/PengyiZhang/SlimYOLOv3.

[14] Q. Wu and Y. Zhou, "Real-Time Object Detection Based on Unmanned Aerial Vehicle," 2019. doi: 10.1109/DDCLS.2019.8908984.

[15] J. Zhang, X. Liang, M. Wang, L. Yang, and L. Zhuo, "Coarse-to-fine object detection in unmanned aerial vehicle imagery using lightweight convolutional neural network and deep motion saliency," *Neurocomputing*, vol. 398, pp. 555–565, Jul. 2020, doi: 10.1016/j.neucom.2019.03.102.

[16] H. C. Baykara, E. Biyik, G. Gul, D. Onural, and A. S. Ozturk, "Real-time detection, tracking and classification of multiple moving objects in uav videos," in *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI*, Jun. 2018, vol. 2017-November, pp. 945–950. doi: 10.1109/ICTAI.2017.00145.

[17] J. Lee, J. Wang, D. Crandall, S. Sabanovic, and G. Fox, "Real-time, cloud-based object detection for unmanned aerial vehicles," in *Proceedings - 2017 1st IEEE International Conference on Robotic Computing, IRC 2017*, May 2017, pp. 36–43. doi: 10.1109/IRC.2017.77.

[18] A. Wiranata, S. A. Wibowo, R. Patmasari, R. Rahmania, and R. Mayasari, "Investigation of Padding Schemes for Faster R-CNN on Vehicle Detection," 2018. doi: 10.1109/ICCEREC.2018.8712086.

[19] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," Jun. 2015, [Online]. Available: http://arxiv.org/abs/1506.02640.

[20] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," Apr. 2020, [Online]. Available: http://arxiv.org/abs/2004.10934.

[21] B. Custers Editor, "The Future of Drone Use Opportunities and Threats from Ethical and Legal Perspectives." [Online]. Available: http://www.springer.com/series/8857.

[22] H. Takano *et al.*, "Visible Light Communication on LED-equipped Drone and Object-Detecting Camera for Post-Disaster Monitoring," in *IEEE Vehicular Technology Conference*, Apr. 2021, vol. 2021-April. doi: 10.1109/VTC2021-Spring51267.2021.9448902.

[23] S. Hossain and D. J. Lee, "Deep learning-based real-time multiple-object detection and tracking from aerial imagery via a flying robot with GPU-based embedded devices," *Sensors (Switzerland)*, vol. 19, no. 15, Aug. 2019, doi: 10.3390/s19153371.

[24] D. Du *et al.*, "VisDrone-DET2019: The Vision Meets Drone Object Detection in Image Challenge Results." [Online]. Available: http://www.aiskyeye.com/.

[25] M. Mandal, L. K. Kumar, and S. K. Vipparthi, "MOR-UAV: A Benchmark Dataset and Baselines for Moving Object Recognition in UAV Videos," in *MM 2020 - Proceedings of the 28th ACM International Conference on Multimedia*, Oct. 2020, pp. 2626–2635. doi: 10.1145/3394171.3413934.

[26] 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. Institute of Electrical and Electronics Engineers Inc., pp. 3212–3232, Nov. 01, 2019. doi: 10.1109/TNNLS.2018.2876865.

[27] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," Nov. 2013, [Online]. Available: http://arxiv.org/abs/1311.2524.

[28] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," Jun. 2015, [Online]. Available: http://arxiv.org/abs/1506.01497.

[29] L. Jiao *et al.*, "A Survey of Deep Learning-based Object Detection," Jul. 2019, doi: 10.1109/ACCESS.2019.2939201.

[30] W. Liu *et al.*, "SSD: Single Shot MultiBox Detector," Dec. 2015, doi: 10.1007/978-3-319-46448-0_2.

[31] A. A. Abdelhamid, S. R. Alotaibi, and A. Mousa, "Deep learning-based prototyping of android gui from hand-drawn mockups," *IET Software*, vol. 14, no. 7, pp. 816–824, Dec. 2020, doi: 10.1049/iet-sen.2019.0378.

[32] G. Jocher, "YOLOv5," 2020. https://github.com/ultralytics/yolov5 (accessed Jul. 08, 2022).

[33] C.-Y. Wang, H.-Y. M. Liao, I.-H. Yeh, Y.-H. Wu, P.-Y. Chen, and J.-W. Hsieh, "CSPNet: A New Backbone that can Enhance Learning Capability of CNN," Nov. 2019, [Online]. Available: http://arxiv.org/abs/1911.11929.

[34] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition," Jun. 2014, doi: 10.1007/978-3-319-10578-9_23.

[35] T.-Y. Lin *et al.*, "Microsoft COCO: Common Objects in Context," May 2014, [Online]. Available: http://arxiv.org/abs/1405.0312.

[36] J. Hosang, R. Benenson, and B. Schiele, "Learning non-maximum suppression," 2017. Accessed: Jul. 08, 2022. [Online]. Available: https://arxiv.org/abs/1705.02950.

[37] L. Alzubaidi *et al.*, "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions," *Journal of Big Data*, vol. 8, no. 1, Dec. 2021, doi: 10.1186/s40537-021-00444-8.