Jurnal Teknik Informatika (JUTIF)
Vol. 4, No. 1, Februari 2023, hlm. 109-115

DOI: https://doi.org/10.20884/1.jutif.2023.4.1.821
p-ISSN: 2723-3863
e-ISSN: 2723-3871

IMAGE DETECTION IN THE AIMBOT PROGRAM USING YOLOV4-TINY

Arief Kelik Nugroho[*1], Ipung Permadi[2], Ahmad Habiballah[3]

[1,2,3]Informatika, Fakultas Teknik, Universitas Jenderal Soedirman, Indonesia
Email: [1]arief.nugroho@unsoed.ac.id, [2]ipung.permadi@unsoed.ac.id, [3]ahmad.habiballah@gmail.com

(Naskah masuk: 10 Januari 2023, Revisi : 30 Januari 2023, diterbitkan: 10 Februari 2023)

Abstract

Cheats are a way for players to gain an unfair advantage. The rise of cheats in online games encourages game producers to increase the security of their games by implementing an anti-cheat system. However, the currently widely circulated anti-cheat system only monitors incoming and outgoing raw data. With the widespread use of image detection systems, we can fool most of today's anti-cheat systems. This can be done by capturing the image that appears on the screen and then processing it through the image detection system. From the process, it can be seen whether there are opponents that appear on the screen. If there is, the program will move the mouse to the place where the enemy is and shoot it. This program is built on the core of the YOLOv4-tiny image detection system.

Keywords: *Aimbot, Convolutional Neural Networks, First Person Shooter, Games, Image, Yolo.*

DETEKSI CITRA PADA PROGRAM AIMBOT MENGGUNAKAN *YOLOV4-TINY*

Abstrak

Cheat adalah cara bagi pemain untuk mendapatkan keuntungan yang tidak adil. Maraknya cheat dalam game online mendorong produsen game untuk meningkatkan keamanan game mereka dengan menerapkan sistem anti-cheat. Namun, sistem anti-cheat yang beredar luas saat ini hanya memantau data mentah yang masuk dan keluar. Dengan meluasnya penggunaan sistem deteksi gambar, kita dapat menipu sebagian besar sistem anti-cheat saat ini. Hal ini dapat dilakukan dengan menangkap gambar yang muncul di layar dan kemudian memprosesnya melalui sistem image detection. Dari proses tersebut, bisa dilihat apakah ada lawan yang muncul di layar. Jika ada, maka program akan menggerakkan mouse ke tempat musuh berada dan menembaknya. Program ini dibangun di atas inti dari sistem deteksi gambar YOLOv4-tiny.

Kata kunci: *Aimbot, Citra, Jaringan Saraf Tiruan Konvolusional, Game, Penembak Orang Pertama, Yolo.*

1. INTRODUCTION

As technology develops, games are becoming more and more popular. One of these game genres is FPS (First Person Shooter). In this game, players are required to be able to move the mouse quickly, agilely, and accurately in order to beat other players. However, not all players can do that well enough. Some of them even rely on cheats to gain an unfair advantage over their opponents. There are many kinds of cheats.

One of the many kinds of cheats that exist is aimbot. Aimbot is a computer program designed to help players target their opponents automatically. Most of the existing cheats make changes to the raw data that goes in and out when the game in question is being run by taking advantage of weaknesses in the game's security system.

Game developers use many ways to quell existing cheats[1], one of which is by implementing a system to detect suspicious incoming and outgoing data flows and strengthen the existing security system in their games. Players who are found to have used cheats will be punished according to the provisions that apply to each game. This anti-cheat system is complete enough to crush the cheats that roam around.

Therefore, to find a way to outsmart the current anti-cheat system, and to encourage the advancement of security technology in existing games, a cheat is made that does not change the raw data in the game in question.

In this program, an aimbot application will be made without being detected by the existing security system by creating a program that can take images that appear on the screen and then process them on an image detection algorithm that will determine whether there are enemies in the image. When the program detects an enemy, the program then sends an input signal to the computer to move the cursor

towards the enemy. Thus, the aimbot program is created without the need to read and change the raw data that enters and exits the game in question.

Aimbot is a program that is often used in multiplayer shooting games where users can target their opponents automatically[2]. This gives users a great advantage as they don't have to move the mouse quickly and precisely to outperform their opponents.

In the development process, Python is the programming language that is most often used in machine learning thanks to its developers and community who have created many useful libraries for scientific calculations and machine learning[3]. Python was chosen because the YOLO algorithm used in this program was created using Python. Python itself has various modules that can be combined. In this program, the Multiple Screen Shots (MSS) module is used to capture the screen and pyautogui to move the mouse. MSS was chosen because of its fast process and can be integrated with other modules such as Python Image Library (PIL) and NumPy. Pyautogui was chosen because it has various options on mouse movement

You Only Look Once (YOLO) is an algorithm for detecting an object contained in an image or video. YOLO itself uses Convolutional Neural Networks as the main structure[4], [5]. YOLO was chosen because this neural network architecture works faster than other detection methods[6]. The YOLO version used in this application is YOLOv4-tiny. This version was chosen because the available hardware is not strong enough to run the YOLOv4 version[7], [8].

2. METHOD

The research method used in this study can be seen in Figure 1.

Figure 1. Research Method.

2.1. Identification of problems

The main problem with existing cheats is that the operating methods do not differ much from each other[9], [10]. This makes the existing security system in a game can detect it easily. Therefore we need a new method that cannot be detected by the security system.

2.2. Data collection

In the data collection process, the default dataset in YOLOv4-tiny is used. This dataset has been through a training process so that it can be directly used in the program to be created. This dataset contains common objects that are commonly found, one of which is human objects. This human object will later be used to detect opponents in the program to be created.

2.3. Program Development

In the development process, several methods were found that were suitable to be applied to this program, namely the MSS module to capture images on the screen quickly and the PyAutoGUI module to move the mouse.

Images that have been captured by MSS will be processed by OpenCV to be cut and compressed. The image is then processed using the YOLOv4-tiny neural architecture network. The result of the processed image will be sent to the python program to convert it into mouse input using PyAutoGUI.

MSS stands for Multiple Screen Shots and is a module in python that can capture images that appear on the screen. MSS is very light and fast compared to other image capture modules. MSS itself also integrates well with OpenCV.

PyAutoGUI is a module in python that can provide input, either keyboard or mouse, to the computer[8]. This allows the python application that we create to interact with other applications. in PyAutoGUI, there is a fail-safe function to pause each PyAutoGUI call. To ensure the program can send mouse input quickly and continuously, this fail-safe feature is disabled at the start of the program.

2.4. Program Testing

This stage is carried out to test whether the program created can work as expected. Testing is done by running the program while the game is running and seeing if the program can detect the opponent and move the cursor towards the opponent.

The test was carried out on a Lenovo brand laptop with the IdeaPadS340 model with an AMD Ryzen 5 3500U Processor with a Radeon Vega Mobile Gfx 2.10 GHz GPU which has a RAM size of 8 GB.

The game used to be tested is Counter Strike: Condition Zero. This game was chosen because it is an old game so it does not require a lot of memory and computational processes to run, given the limitations of existing hardware and the need for a fairly large computational process by the program created.

3. HASIL DAN PEMBAHASAN

3.1. Network Structure

YOLOv4-tiny uses CSPDarknet53-tiny on its backbone network using the CSPBlock module found in the cross stage partial network, replacing the ResBlock module in the residual network. The CSPBlock module divides the feature map into two parts, and combines the two parts with a cross-stage residual edge. This allows gradient paths to be spread over two different network paths to increase the

correlation difference contained in the gradient information. The CSPBlock module can strengthen the learning ability of convolutional networks compared to the ResBlock module by increasing the calculation time of about 10-20%, this process increases accuracy. To reduce the number of calculations that exist, this process eliminates the computational bottleneck that has a lot of computations in the CSPBlock module. This increases the accuracy of the YOLOv4-tiny method in the case of constant or reduced computations[11], [12]

YOLOv4-tiny uses the LeakyReLU function as an activation function on CSPDarknet53-tiny without using the Mish activation function used in YOLOv4[13]. The LeakyReLU function can be seen in equetion 1

$$y = \begin{cases} x_i & x_i \geq 0 \\ \frac{x_i}{a_i} & x_i < 0 \end{cases} \quad (1)$$

Where : $a_i \in (1, +\infty)$ constant parameters.

As part of the feature aggregation, the YOLOv4-tiny method uses the pyramid network feature to extract feature maps with different scales to increase object detection speed, without using spatial pyramid pooling and path aggregation networks used in the YOLOv4 method. YOLOv4-tiny itself uses feature maps with different sizes, namely 13×13 and 26×26 to predict the detection results[13], [14]. If the input size is 416×416 and the feature classification is 80[15], the YOLOv4-tiny structure can be shown in Figure 2.

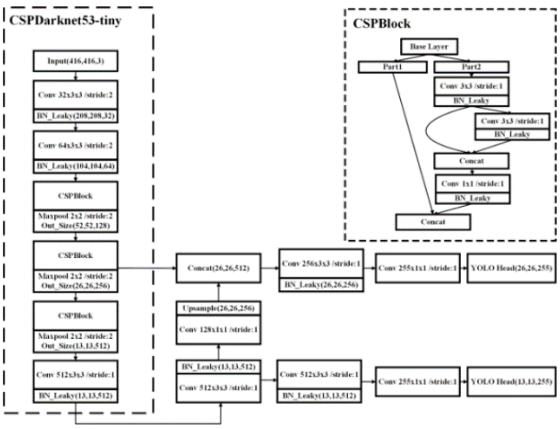


Figure 2. YOLOv4-tiny structure.

3.2. Prediction Process

The prediction process in YOLOv4-tiny is the same as the method used in YOLOv4[12][13]. All inputs obtained will be resized to be uniform. Then the input image will be divided into a grid of S×S size. on each grid there is a bounding box totaling B which is used to detect objects. This causes the program to give an output of S×S×B for each image obtained. If the center of an object is in a grid, then the bounding box in the grid will predict the object.

To reduce redundancy in the prediction process, a confidence threshold is created. If the confidence value of a bounding box is greater than the confidence threshold, the bounding box will be saved[16]. If the confidence value is lower, the bounding box will be deleted. The formula for calculating the confidence value of a bounding box can be seen in equetion 2.

$$C_i^j = P_{i,j} * IOU_{pred}^{truth} \quad (2)$$

Where:
C_i^j : confidence value of bounding box ke-j grid i.
$P_{i,j}$: object function value.

If the value the object in the box *j* is 1 in *grid i*, $P_{i,j}$. otherwise, $P_{i,j}$ is 0. IOU_{pred}^{truth} represents the cross between the existing shading between the predicted box and the ground truth box. The higher the objectness value obtained, the closer the predicted box will be to the ground truth box. The loss function used in YOLOv4-tiny is the same as in YOLOv4. The loss function itself consists of three parts which can be stated in equetion 3.

$$loss = loss_1 + loss_2 + loss_3 \quad (3)$$

With loss_1 as the confidence loss function, loss_2 as a classification loss function, and loss_3 as a bounding box regression loss function[17].

The problem that arises when adapting a domain using a discriminator is that the discriminator assigns the same importance to different samples. This makes some parts difficult to transfer and can cause negative values to appear. To solve this problem, the CDAN method is used which applies entropy to the network which can be seen in equetion 4.

$$loss_1 = -\sum_{i=0}^{S^2}\sum_{j=0}^{B} W_{ij}^{obj}\left[\hat{C}_i^j log\left(C_i^j\right) + \left(1 - \hat{C}_i^j\right) log\left(1 - C_i^j\right)\right] - \lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\left(1 - W_{ij}^{obj}\right)\left[\hat{C}_i^j log\left(C_i^j\right) + \left(1 - \hat{C}_i^j\right) log\left(1 - C_i^j\right)\right] \quad (4)$$

Where S^2 is the number of grids in the input image, B is the number of bounding boxes in a grid, W_{ij}^{obj} is a function on the object. If *bounding box* j in grid i detect object, then the value W_{ij}^{obj} is 1. Otherwise W_{ij}^{obj} is 0. C_i^j is a *confidence score* in the predicted box dan $\hat{C}_i^j$ is *confidence score* in the *truth box*. Whereas forλ_{noobj} is a weight value.

The classification loss function itself can be seen in equetion 5

$$loss_2 = -\sum_{i=0}^{S^2}\sum_{j=0}^{B} W_{ij}^{obj}\sum_{c=1}^{c}\left[\hat{p}_i^j(c)\,log\left(p_i^j(c)\right) - \left(1-\hat{p}_i^j(c)\right)log\left(1-p_i^j(c)\right)\right] \quad (5)$$

Where $p_i^j(c)$ as the predictive probability and $\hat{p}_i^j(c)$ sas the probability of the truth of the objects included in the c classification that are in the bounding box j and grid i.

For the bounding box regression loss function, it can be seen in equetion 6.

$$loss_3 = 1 - IOU + \frac{\rho^2\left(b,b^{gt}\right)}{c^2} + \frac{16}{\pi^4}\frac{\left(arctan\frac{w^{gt}}{h^{gt}} - arctan\frac{w}{h}\right)^4}{1-IOU+\frac{4}{\pi^2}\left(arctan\frac{w^{gt}}{h^{gt}} - arctan\frac{w}{h}\right)^2} \quad (6)$$

Where:

IOU as shading above the unit between the predicted bounding box and the truth bounding box.

w^{gt} and h^{gt} as the width and height of the truth bounding box.

w and h as width and height *predicted bounding box*.

$\rho^2(b,b^{gt})$ shows the Euclidean distance between the midpoints of the predicted bounding box and the truth bounding box.

C as the minimum diagonal distance between a box that can contain a predicted bounding box and a truth bounding box.

The first step that needs to be done in implementing this application is to prepare the dataset. The dataset used is the default YOLOv4-tiny dataset that has gone through the training process. Once the dataset is ready, the program can be run. When the program is running, a small screen will appear to monitor the ongoing detection. On this screen, a bounding box will appear showing what and where the detected object is. In addition, the program detection speed in frames per second will also appear on the terminal. The display screen along with the terminal can be seen in Figure 3.

After the program runs, the game you want to play can be run. The program will capture the image that appears in the center of the screen. This captured image will be processed by a program which will produce output in the form of movement and mouse input.

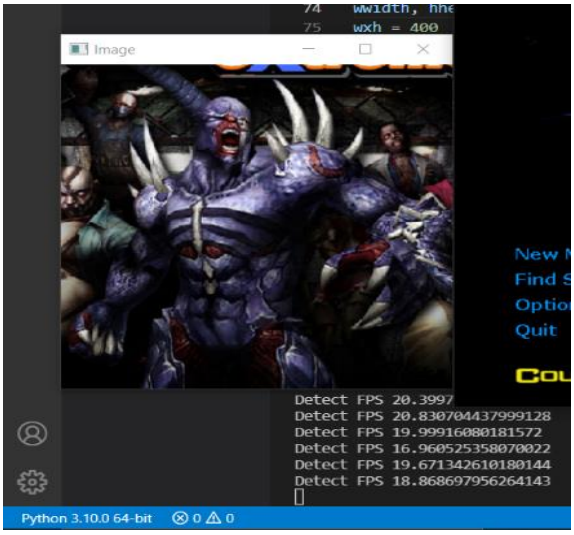


Figure 3. Screenshot and terminal

3.3. Evaluation

After testing the program on the game Counter Strike: Condition Zero, it is known that the program can run smoothly with speeds ranging from 18-20 fps. The program can also detect enemies that appear and shoot them automatically. The average percentage of detection success obtained from 5 different maps is 50.56%. This value is obtained by calculating the average number of times the opponent is detected by the program manually within 5 minutes on each map. The percentage of detection success on 5 different maps can be seen in Figures 4, 5, and 6

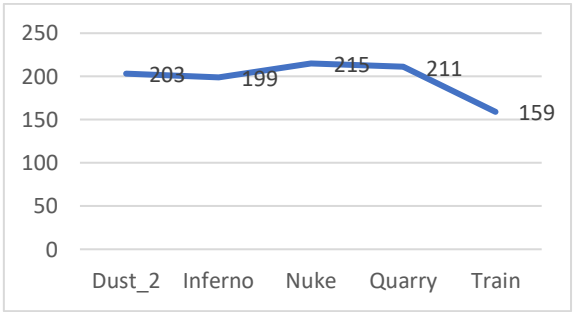


Figure 4. The number of enemies that appear on the screen for each map

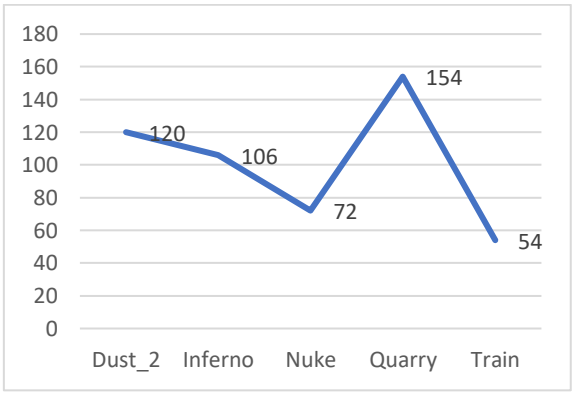


Figure 5. Number of enemies detected by the program on each map

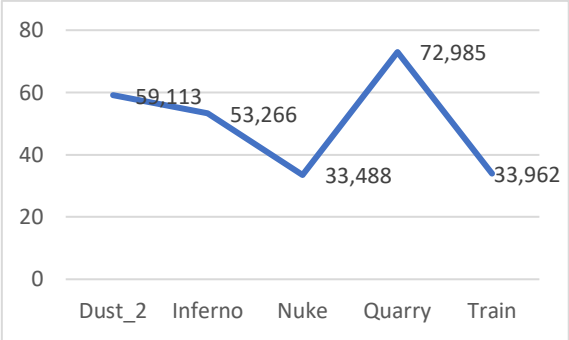


Figure 6. Percentage of program detection on each map

From the data obtained, it can be concluded that the Quarry map has the largest detection percentage, while the Nuke map has the smallest detection percentage. This is influenced by the existing texture on the map. Some places on certain maps have a texture that resembles the texture of the opponent so that the program is difficult to distinguish between the two. as can be seen in Figure 4.

Figure 4. The program cannot detect the opponent because the texture is similar to the texture of the walls and floors

In addition to similar textures, in certain places on the map there are places that have minimal lighting. This also greatly affects the detection as can be seen in Figure 5.

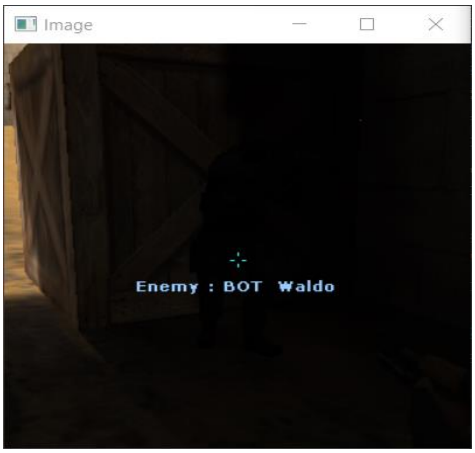


Figure 5. The program cannot detect the opponent due to the lack of lighting in that place

Another problem that arises in this program is that the program has difficulty recognizing enemies that are at a distance. This is because the program compresses the captured image which causes the enemy's image to become blurred so that the YOLOv4-tiny algorithm cannot recognize the enemy as shown in Figure 6.

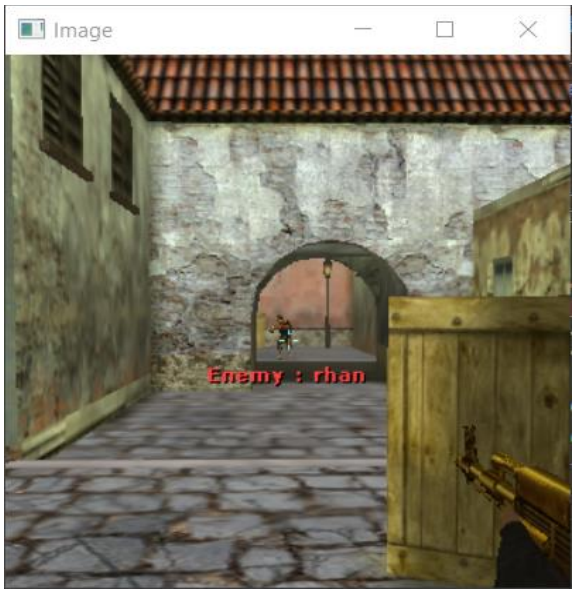


Figure 6. The program cannot detect an opponent who is at a considerable distance

Another problem that arises is that the program sometimes detects objects incorrectly. An object that is not an opponent is sometimes detected as an opponent. This results in players sometimes shooting at the ground or walls that are considered enemies. The program also often makes players shoot their own friends because they too are detected as humans. This can be seen in Figure 7 and Figure 8.

Figure 7. The program detects his own rifle as a player

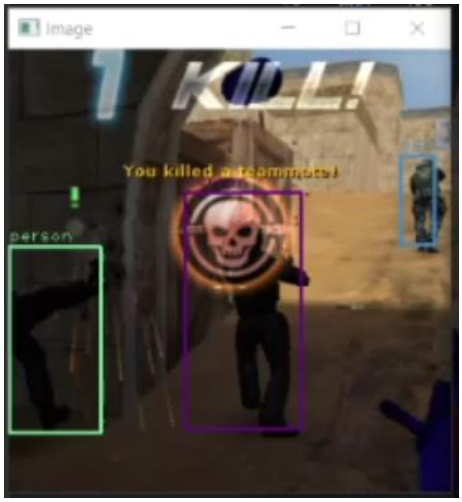
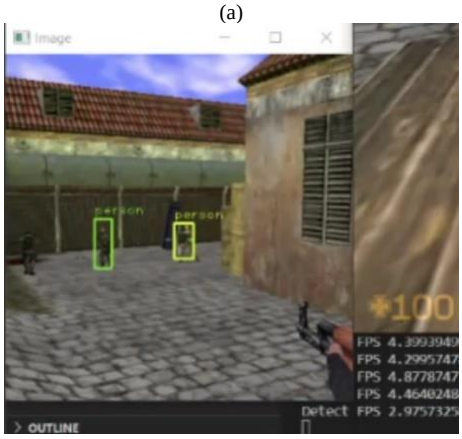


Figure 8. The program can't tell the difference between friends and foes, so it shoots its own comrades

Another problem that arises is that the program prioritizes detecting other objects that are not opponents. This happens because the dataset is still general and has not been specific to human-shaped objects. This causes a drastic decrease in accuracy on maps that have a lot of common objects as happened on the Train map as can be seen in Figure 9.

(a)



(b)

Figure 9. a) The program detects objects other than the opponent and tends to prioritize those objects b) Decreased FPS when law

In addition, the program also experienced a significant decrease in performance when detecting more than 1 opponent. This is because the non-maximal suppression (NMS) process used when the program detects an object really drains a lot of computer performance, which results in decreased detection speed [9].

4. CONCLUSION

The problems that arise as well as heavy hardware requirements make this program can not be used smoothly on common devices. In the future, further development is needed to perfect this program to increase the speed and accuracy of detection. The testing process is better by using more adequate hardware to increase the possibility of enemy detection without sacrificing detection speed. For further development in the future, optimization can be done by creating its own dataset specifically for detecting human objects. This is necessary so that the program can recognize character objects more optimally and to reduce errors when the program performs the detection process. In addition, a more effective method can be found in applying the non-maximum suppression process to speed up the image processing when an enemy is detected and create a system to distinguish friend and foe so that incidents of shooting teammates can be avoided.

REFERENCES

[1] S. Laato, S. Rauti, L. Koivunen, and J. Smed, "Technical cheating prevention in location-based games," in *ACM International Conference Proceeding Series*, 2021. doi: 10.1145/3472410.3472449.

[2] S. Raschka and V. Mirjalili, *Python Machine Learning: Machine Learning & Deep Learning with Python, Scikit-Learn and TensorFlow 2, Third Edition*, no. January 2010. 2019.

[3] J. Redmon and A. Angelova, "Real-time grasp detection using convolutional neural networks," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2015, vol. 2015-June, no. June. doi: 10.1109/ICRA.2015.7139361.

[4] Z. Jiang, L. Zhao, L. I. Shuaiyang, and J. I. A. Yanfei, "Real-time object detection method for embedded devices," *ArXiv*, vol. 3, 2020.

[5] P. Ren, L. Wang, W. Fang, S. Song, and S. Djahel, "A novel squeeze YOLO-based real-time people counting approach," *International Journal of Bio-Inspired Computation*, vol. 16, no. 2, 2020, doi: 10.1504/ijbic.2020.109674.

[6] W. He, Z. Huang, Z. Wei, C. Li, and B. Guo, "TF-YOLO: An improved incremental network for real-time object detection,"

Applied Sciences (Switzerland), vol. 9, no. 16, 2019, doi: 10.3390/app9163225.

[7] L. v. Yingli *et al.*, "A comparative study of different machine learning algorithms in predicting the content of ilmenite in titanium placer," *Applied Sciences (Switzerland)*, vol. 10, no. 2, 2020, doi: 10.3390/app10020635.

[8] D. Xu and Y. Wu, "Improved YOLO-V3 with densenet for multi-scale remote sensing target detection," *Sensors (Switzerland)*, vol. 20, no. 15, 2020, doi: 10.3390/s20154276.

[9] I. Permadi, A. K. Nugroho, and M. R. Rachmat, "PREDICTION OF THE AMOUNT OF PEPPER HARVEST BY USING FUZZY ASSOCIATIVE MEMORY," *Jurnal Teknik Informatika (Jutif)*, vol. 3, no. 1, pp. 177–182, 2022.

[10] Q. Oktiriani, A. K. Nugroho, and E. Maryanto, "FRONTEND DEVELOPMENT IN THE FINAL STUDY MANAGEMENT SYSTEM (SIPEDA) AT THE ENGINEERING FACULTY OF JENDERAL SOEDIRMAN UNIVERSITY," *Jurnal Teknik Informatika (Jutif)*, vol. 3, no. 2, pp. 321–329, 2022.

[11] X. Zhang, Y. Zhang, B. He, and G. Li, "Research on remote sensing image aircraft target detection techonlogy based on YOLOv4-tiny," *Guangxue Jishu/Optical Technique*, vol. 47, no. 3, 2021.

[12] X. Liu, Y. Zhang, F. Bao, K. Shao, Z. Sun, and C. Zhang, "Kernel-blending connection approximated by a neural network for image classification," *Comput Vis Media (Beijing)*, vol. 6, no. 4, 2020, doi: 10.1007/s41095-020-0181-9.

[13] X. Hou, J. Ma, and S. Zang, "Airborne infrared aircraft target detection algorithm based on YOLOv4-Tiny," in *Journal of Physics: Conference Series*, 2021, vol. 1865, no. 4. doi: 10.1088/1742-6596/1865/4/042007.

[14] T. Jiang and J. Cheng, "Target Recognition Based on CNN with LeakyReLU and PReLU Activation Functions," in *Proceedings - 2019 International Conference on Sensing, Diagnostics, Prognostics, and Control, SDPC 2019*, 2019. doi: 10.1109/SDPC.2019.00136.

[15] W. Fang, L. Wang, and P. Ren, "Tinier-YOLO: A Real-Time Object Detection Method for Constrained Environments," *IEEE Access*, vol. 8, 2020, doi: 10.1109/ACCESS.2019.2961959.

[16] Y. Chen, Q. Guo, X. Liang, J. Wang, and Y. Qian, "Environmental sound classification with dilated convolutions," *Applied Acoustics*, vol. 148, 2019, doi: 10.1016/j.apacoust.2018.12.019.

[17] Y. Ji, H. Zhang, Z. Zhang, and M. Liu, "CNN-based encoder-decoder networks for salient object detection: A comprehensive review and recent advances," *Inf Sci (N Y)*, vol. 546, 2021, doi: 10.1016/j.ins.2020.09.003.