



Optimizing Hand Gesture Recognition Using CNN Model Supported by Raspberry pi for Self-Service Technology

Abdul Haris Rangkuti^{a,*}, Varyl Hasbi Athala^a, Farrel Haridhi Indallah^a, Fajar Febriansyah^a

^a Informatics Department, School of Computer Science, Bina Nusantara University, Jakarta-11480, Indonesia

Corresponding author: *rangku2000@binus.ac.id

Abstract— This study describes the optimization of hand gesture recognition on Raspberry Pi 4 technology has advanced over the past years, and some computers are now able to compute much more complex problems like real-time object detection. However, for small devices, optimization is required to run in real-time with acceptable performance in terms of latency and low-cost effect on accuracy. Low latency is a requirement for most technology, especially when integrating real-time object detection as input into Self-Service Technology on Raspberry Pi for the store. This research was conducted on 288 pictures with six types of chosen hand gestures for command inputs that have been configured in the Self-Service Technology as a training dataset. In the experiment carried out, 5 CNN object detection models were used, namely YOLOv3-Tiny-PRN, YOLOv4-Tiny, MobileNetV2-Yolov3-NANO, YOLO-Fastest-1.1, and YOLO-Fastest-1.1-XL. Based on the experiment after optimization, the FPS and inference time metrics have improved performance. The performance improves due to a gained average value of FPS by 3 FPS and a reduced average value of inference time by 119,260 ms. But such an improvement also comes with a reduction in overall accuracy. The rest of the parameters have a reduced score on Precision, Recall, F1-Score, and some for IoU. Only YOLO-Fastest-1.1-XL have an improved value of IoU by about 0.58%. Some improvements in the CNN and dataset might improve the performance even more without sacrificing too much on the accuracy, but it's most likely suitable for another research as a continuation of this topic.

Keywords—Darknet; hand gesture; Opencv; object detection; YOLO; Raspberry Pi.

Manuscript received 20 Jul. 2022; revised 29 Aug. 2022; accepted 7 Sep. 2022. Date of publication 31 Mar. 2023. International Journal on Informatics Visualization is licensed under a Creative Commons Attribution-Share Alike 4.0 International License.



I. INTRODUCTION

Technology has advanced over the past years. A computer is one of the technologies that got improved. The number of transistors with each new CPU generation increased without rapidly changing the die size [1]. Such advancements allow some computers to compute much more complex problems on different sizes of machines. With such ability, these computers can run many tasks at the same time. Tasks like encoding video, encrypting files, and large and complex number computations are examples that require a lot of processing power [2]. However, there are still some new computers that are too slow to run complex problems. This problem is usually on small devices. Because the smaller the size, the slower it is due to compensating package size. These slower devices can only run complex problems when the system is optimized to run on the computer. Fortunately, hardware acceleration is often implemented into the semiconductor to accelerate processing in the supported

environments. Thus, computers can execute commands better and faster, even on a small device.

Rapid technology advancements also allow Artificial Intelligence (AI) to integrate into Computers. Real-time object detection is an example of AI implementation. Object detection works by generating potential bounding boxes and then running a classifier on these proposed boxes [3]. This study uses hand gestures image as the training and to give some message. The image can be used for identification, recognition, or classification of traditional clothes [4]. These data get extracted in the training process for their distinct characteristics. These characteristics are what define an object. Therefore, it could better recognize them within the frame. After training the AI, the weight files contain all the extracted features for testing the real-time object detection. Therefore, the machine will be able to recognize human hand gestures with the help of a camera. Then, it can do any programmed execution just from detected human hand gestures. Such interaction is called Human and Computer Interaction (HCI).

Over the years, human-computer interaction technology has focused on the design interface aspect and how humans interact with machine. Self-service technology is a technological interface that customers have to interact with to produce an independent service that is different from the usual direct service involving an employee [5]. There are many implementations of self-service technology. One of them is self-service technology for ordering products. This technology has benefits for the store owner and the user. Not only that, it could reduce operating costs and improve the quality of service. It is because the nature of this technology only involves computers in the ordering system. This study enables human-computer interaction by establishing a touchless system using hand gestures. Such a combination prevents direct contact between different users. Therefore, this technology could help create a more hygienic environment in stores.

The main focus of this study is to produce self-service technology that can be used for a raspberry pi environment with acceptable performance. A self-service technology application tests the performance by simulating the system's response feel and latency. The load gets heavier as the self-service technology also runs in parallel with the hand gesture recognition system, and it impacts the processor performance of the raspberry pi worse. Therefore, this study uses several small CNN models such as YOLOv3-Tiny-PRN, YOLOv4-Tiny, MobileNetV2-Yolov3-NANO, YOLO-Fastest-1.1, and YOLO-Fastest-1.1-XL to test accuracy and inference performance. Although these CNNs are for microprocessors like raspberry pi, it is still required to tweak the network to get the most optimal performance. Improving such can generate hints for recommendation systems, stand-alone process management, and human input reduction [6]. This means that there must be a balance between performance and accuracy. Therefore, optimizations are combined to form research on optimizing hand gesture recognition using raspberry pi for Self Service Technology in a store.

II. MATERIALS AND METHOD

A. The Materials

The technology of hand gesture recognition has been implemented as one of the solutions to different kinds of problems. One of the implementations of this technology is used in a study to use hand gesture recognition to send a signal for help. Using a Deep Neural Network as one of the system's machine learning types results in an accuracy of 98.79% in recognizing the hand gesture. This study concluded that using hand gesture recognition would be simplified and accelerate the process of asking for help, create a safe and fast way by using only one hand, and reduce the risk of unwanted things happening that could further increase the danger to the person involved [7]. The satisfying result of this study shows that using hand gesture recognition could increase the performance of a certain process. This research will implement the hand gesture recognition system in a self-service ordering system.

There are a number of papers that conducts study on the topic of hand gesture recognition. There is one study that compares different algorithms that are applicable to sign language hand gesture recognition. This study provides an in-

depth analysis that offers important insights to the researcher, developers, and any other interested parties on sign language hand recognition algorithms [8]. Besides using hand recognition technology to detect and identify the meaning of sign language, hand recognition technology is used in many different applications. Using hand gesture recognition for human-robot interaction [9]–[11], medical treatment [12]–[14], and automotive human-machine interaction [15] are a few of the many different uses of hand recognition technology.

A study was conducted to analyze the performance of YOLOv4, YOLOv4-tiny-PRN, YOLOv3, and YOLOv3-tiny on three types of Accelerator-based SDCs, which are NVIDIA Jetson Nano, NVIDIA Jetson Xavier NX and Raspberry Pi 4B (RPi). The conclusion of this study suggests that two aspects must be considered before choosing the hardware for implementing a system intelligence. The first aspect is that even though ASIC accelerators are low-performance, they are SBC-friendly. One of the experiments in this study concluded that the mean confidence on RPi + NCS2 is 0%, while the other devices have mean confidence of 57.9%. The second aspect that needs to be considered is when implementing smart applications to an SBCs-based GPU. Because of the memory sharing between CPU and GPU, the architecture and other related parameters must be designed carefully to achieve satisfying accuracy and speed results [16].

In another study under the title “Real-time object detection method for embedded devices,” there is an experiment to compare the performance of two types of YOLO methods for object detection (YOLOv3-tiny and YOLOv4-tiny) on embedded devices. When using Raspberry Pi 3B as the embedded device, the results for YOLOv3-tiny and YOLOv4-tiny are 0.18 FPS for YOLOv3-tiny and 0.19 FPS for YOLOv4-tiny. With only a difference of 0.01 FPS, YOLOv4-tiny proved to have better FPS than YOLOv3-tiny when used in a Raspberry Pi 3B device [3]. Comparing each model's performance with the same environment device will reveal which model is far more superior or optimal than the other when they are used in the same device. While in this study, there will be data showing which CNN models will perform well in an embedded device (Raspberry Pi) where the device is being used in self-service technology.

Another study was conducted to analyze the performance difference between YOLOv4 and YOLOv4-Tiny in detecting images, recorded videos, and real-time video. These models are converted to TFlite models for implementation in Mobile Applications using Android Studio. In the training stage, the YOLOv4-Tiny took less than an hour for 1000 iterations compared to 2 hours when using YOLOv4 [17]. As for the testing stage, YOLOv4 achieved 96.92% of accuracy on real-time video at 5071 ms while the YOLOv4-Tiny achieved 74.72% of accuracy on real-time video at 491 ms. These results show that YOLOv4-tiny is still effective enough to detect objects with a much better inference time than YOLOv4 with an expense of slight decreases in accuracy. Therefore, this study concludes that YOLOv4-Tiny is a better choice than YOLOv4 for real-time object detection applications.

B. Research Method

Figure 1 describes the research on the optimization of self-service technology in providing services in dealing with

infectious diseases. The process of optimizing self-service technology by utilizing hand gesture recognition technology has three stages consisting of a preparation stage, a training stage, and a testing stage. The preparation stage needs the six data. These data are required to prepare the things to learn and the network architecture for the training stage. In the training stage, the darknet framework uses the prepared six data for the training. After the training stage is complete, the testing stage uses the file from the training stage.

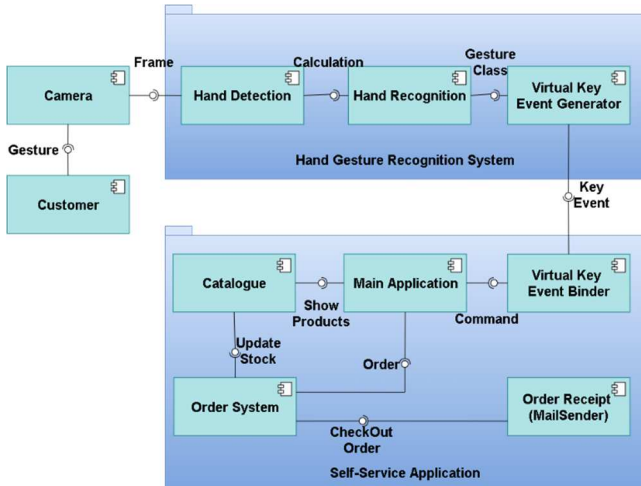


Fig. 1 Research Component Diagram Study of Optimizing Hand Gesture Recognition

Figure 1 shows the proposed system of self-service technology. This stage uses the proposed system to simulate the effects of the optimization in the real world. It starts by checking every frame on the webcam, then making a bounding box on the target object. Then finally, apply the gesture as a command to the Tkinter UI. For more details, the stages of each process can be seen in the following explanation.

Preparation

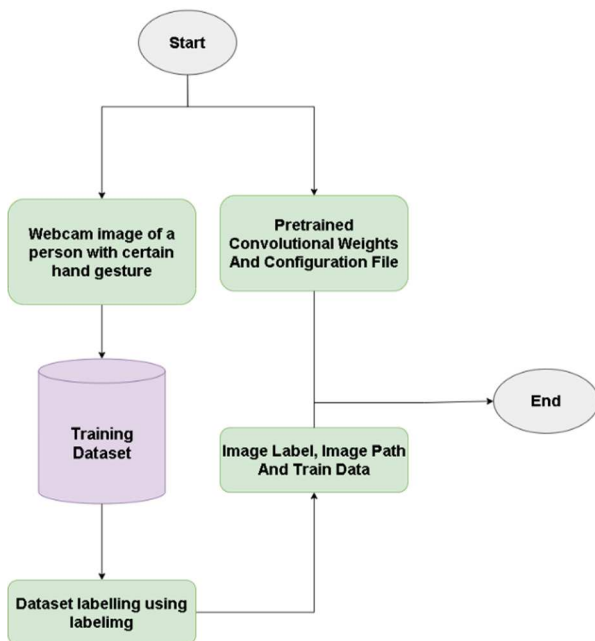


Fig. 2 Preparation Phase Flowchart

1) *Preparing images of hand gestures and train files:* Figure 2 describes that before starting the process, it is necessary to prepare things for the experiment. In figure 2, it is required to have photos, pre-trained weight files and configuration, image labels, image paths, and train data. First, this experiment requires photos of a person with certain hand gestures.

In this case, all images were obtained using a webcam. These images fall into six categories, as in Table I:

TABLE I
HAND GESTURE IMAGE SAMPLES







Hand Gesture	Category
	Hi Gesture
	Fist Gesture
	Three Fingers Gesture
	Ok Gesture
	L-Shaped Gesture
	C-Shaped Gesture

Table I informs that each category has 48 pictures with a different person, hands, contrast, brightness, lighting, noise, image size, and image ratio. The only requirement is to have a picture with at least one hand doing a gesture. Also, all images must be in the format .jpg. After that, no post-processing was made to the image. This is done to ensure the AI learns from different variables and becomes better at guessing in difficult cases. After that, images are labeled to output the text label file. Then, the paths of the images must be generated into a single text file. The information about the path and total class numbers are then written in a "trainer.data" file. Last, the pre-trained weight file must also be prepared and configured to the class total.

2) *Dataset Setup:* Figure 3 Inform that this study uses six chosen hand gestures as the category for the hand gestures. Each of these categories consists of 48 photos.

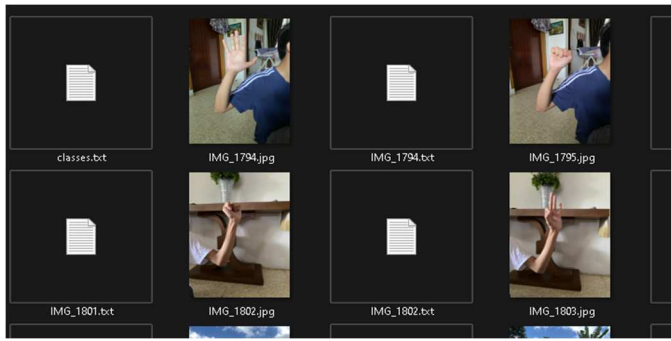


Fig. 3 Hand Gesture Image Dataset

These photos vary in lighting, image size, image ratio, background, hands, person, and distance. Therefore, a total of 288 images were collected for this experiment. This is to help the machine recognize the hands better. Then after collecting the images, all of them must be labeled. Labeling is a requirement in the topic of object detection. Labeling provides a good description of image content through visual mapping features with semantic and spatial labels [18]. Therefore, it helps the machine to learn objects in the image. This process also outputs a single text file that contains the coordinate, size, and corresponding class number of the hand gesture in each image. The dataset also contains a single text file containing class names and numbers. Figure 3 is the screenshot of some of the dataset that is ready for training.

3) *Training convolutional neural network using darknet:* Figure 4 informs the training stage, which outputs weight files. As stated before, it requires files which are images for train, image label, class identity, train data, model pre-trained weight, and a matching model configuration before training. This stage aims to use the six data that were previously prepared in the preparation stage to be trained using Darknet.

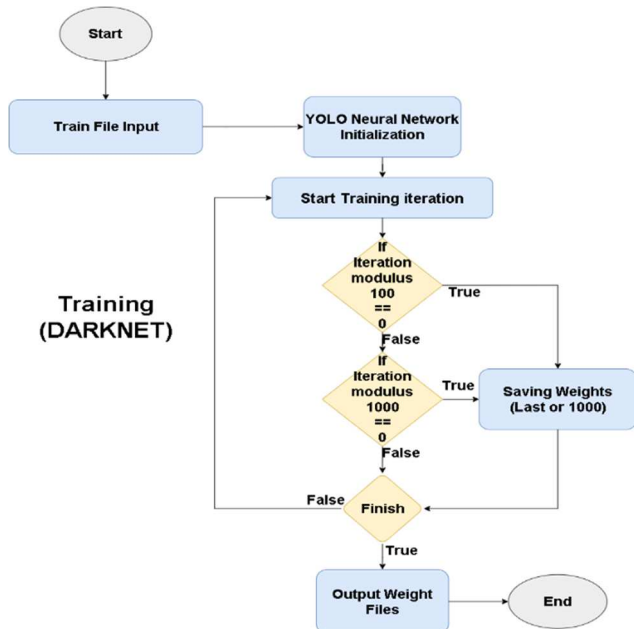


Fig. 4 Training darknet Phase Flowchart

The process will continue until the iteration modulus is completed or the 1000th iteration modulus is finished. The results of the training phase are weight files. According to

Budagyan and Abagyan [19], “weights are determined by choice of a target-object-space, which depends heavily on the nature of the objects in the training set and the predicted property.” The definition of weights provided by Budagyan and Abagyan [10] shows that the weight data is not always the same. So, to get the most accurate data possible, the training will be done using the same device and training data set. The result would be in a less ambiguous weight file.

To support the experiment, the darknet framework was used to help only in training the model. All models were tested, and these hyperparameters were set the same as in Table II.

TABLE II
TRAINING CONFIGURATION FOR ALL MODELS

Hyperparameters	Value
Batch	32
Subdivisions	24
Max_Batch	12000
Classes	6

Based on this experiment, Table II informs that each category image has different characteristics. For the machine to learn about human gestures, an artificial neural network in the form of a Convolutional Neural Network (CNN) is required. CNN is a variation of multi-layer perceptron human neural networks inspired by visual cortex research on cats’ visual senses [20]. This experiment uses five different CNNs. This experiment uses some model CNN to understand the optimal result in recognition. Table III compares CNNs that were used to optimize hand gesture recognition.

TABLE III
CNN MODEL COMPARISON TO OPTIMAL THE HAND GESTURE RECOGNITION

CNN Model	Parameters	BFLOPS	Network Size
YOLOv3-Tiny-PRN	4.7M	3.409	416
YOLOv4-Tiny	5.77M	6.795	416
MobileNetV2-YOLOv3-NANO	-	0.482	320
YOLO-Fastest-1.1	0.35M	0.226	320
YOLO-Fastest-1.1-XL	0.925M	0.695	320

Table III, which is from studies [21], [22], inform some CNN model will extract a different number of features, the base layers for every CNN model are all the same, and every model will consist of three main layers which order are the Convolutional Layer, Pooling Layer, and Fully Connected Layer [23]. Even though each CNN models have the same layers, the performance of each model will be different from one another. Different CNN models mean different amounts of features that can be extracted [24]. This study uses YOLO Architecture, a single CNN that simultaneously predicts multiple bounding boxes and the corresponding class probabilities [25]. Also, each model has its unique architecture, as shown in Table III, and each CNN model has different parameters, BFLOPS, and network.

4) *Hand gesture recognition system:* Figure 5 illustrates that after the training is complete, the system can start the testing phase by importing the required files for testing. These files are the trained weight file, configuration file, and trainer.data file.

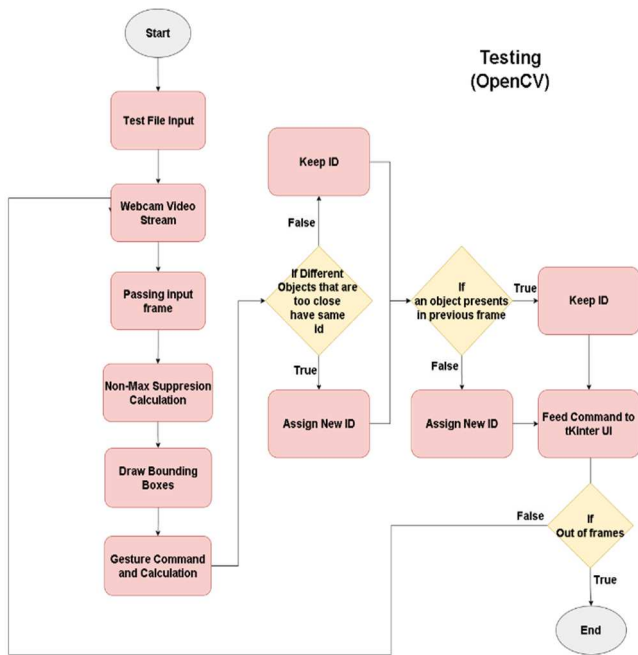


Fig. 5 Testing Phase to recognize hand gesture

The testing process starts by taking a frame from the webcam. Then, calculate the prediction using non-max suppression (NMS). The result is for making the bounding box on the target object. The bounding box usually comes with other information like class and coordinate, which is important for Gesture Command and Calculation. It is to generate the appropriate virtual key for the UI the receive. Then, some condition cases algorithm decides which behavior to apply within the case. Finally, The Tkinter UI should respond to the virtual key according to the predefined bindings. As in figure 5, this whole process runs in a loop. For more details, the following section will explain more about converting gesture input into command.

5) *Converting gesture input into the command:* Figure 6 informs the experiment stages, and the hand gesture recognition system is implemented into the self-service technology. This concept of gesture recognition is how a gesture can be converted into a command, just like how our eyes turn visual input into thoughts [26]. For the system to perform as previously stated, the AI needs to convert the gesture input into a command so that the UI can respond.

Figure 6 shows that the AI converts the hand gesture input into a virtual keyboard event input. Then, by binding the UI with several keys based on the number of hand gesture types (See Table IV), the UI can accept generated inputs from the virtual key event. Each virtual keyboard input would react to different UI commands (in Table IV). The UI has predefined commands that execute when a certain key is detected. This technique would make the self-service interface respond to the hand gesture that the user makes.

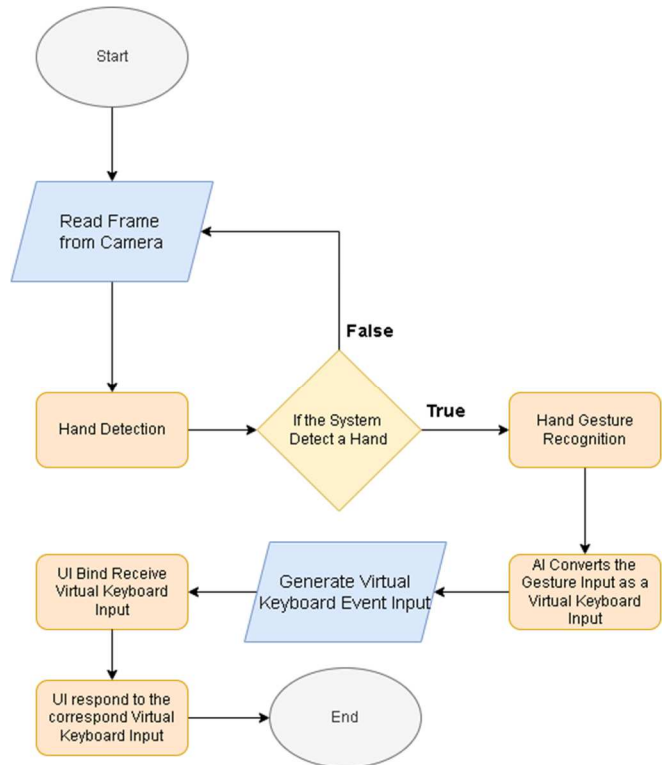








Fig. 6 The Process of Converting Gesture Input into UI Command

TABLE IV
THE RESULTS OF CONVERTING HAND GESTURE INPUT

Hand Gestures	Virtual Key	UI Commands
	NONE	First Initialization before moving to the x and y axis later on using the “Fist” gesture.
	“Right Arrow”, “Left Arrow”, “Down Arrow”, Or “Up Arrow”	Move the selected grid to the left, right, up, or down according to the hand movement.
	“Backspace”	Going back to the previous menu.
	“Enter”	Confirmation or move to the next menu.

	“Space”	Selecting the item.
	“Delete”	Cancel the whole ordering process

6) *Measuring model performance*: When a Raspberry Pi device uses different CNN models for hand gesture recognition, each model will perform differently. This happens because each CNN model has a different architecture that makes each model unique. This is why some models could work well when in a certain situation. Therefore, by using the data collected throughout the training process, it will be possible to compare the five CNN models. Several calculation metrics like mAP, Precision, Recall, and F1-score are used as comparison variables. The comparison variables between each CNN model will determine which is best for hand gesture recognition in a Raspberry Pi device. The samples needed for measuring the CNN model’s performance are split into two categories. The first category is the positive samples, which have the targeted object in them. The second category is the negative samples, which have none of the targeted objects in them. A more detailed explanation of these calculation metrics is explained as follows:

- Precision and Recall

To calculate the value of precision, there are two necessary variables. The first variable is the number of positive samples the model correctly classified and the last one is the total number of samples classified as positive samples (whether the model correctly classified them or not). The range value of precision is from 0 to 1, with 0 as its lowest score and 1 as its highest score. This precision value reflects the model's reliability when classifying the positive samples. The result of precision is obtained by dividing only correctly classified positive samples by the total number of positive samples. Compared to precision, a recall is calculated by dividing the number of positive samples the model correctly classified and the number of total positive samples [27], [28]. Recall completely ignores the negative samples and only focuses on the result of the positive samples. With the range the same as precision, a recall measures how many of the models correctly classifies positive samples. Both precision and recall formulas are illustrated below:

$$\text{Precision} = \frac{TRUE_{positive}}{TRUE_{positive} + FALSE_{positive}} \quad (1)$$

$$\text{Recall} = \frac{TRUE_{positive}}{TRUE_{positive} + FALSE_{negative}} \quad (2)$$

Where:

$TRUE_{positive}$ = total of positive samples that the model correctly classified

$FALSE_{positive}$ = total of negative samples that the model mistakenly classified as positive samples

$FALSE_{negative}$ = total of positive samples that the model could not classify.

- Intersection over Union (IoU)

Two things need to be addressed in Intersection over Union or IoU because the two are defined later in the IoU formula. Those two values are the predicted bounding box and the truth bounding box. The predicted bounding box is the box that the model predicts to have one of the targeted objects or items. Meanwhile, the truth bounding box is the box that the tester initially marked as the targeted object before the measuring process. Finally, the definition of IoU is the ratio between the intersection of the predicted bounding box and the truth bounding box with the combined area or union of the two boxes (see Figure 7).

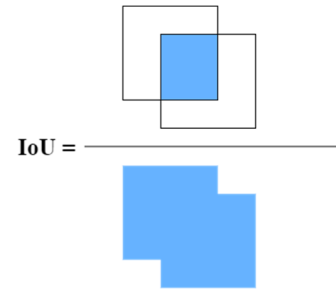


Fig. 7 IoU Formula and Illustration

The more the predicted box overlays the truth box's area, the higher the model's accuracy. In return, the IoU score would be near the value of 1, which is the highest accuracy score [27].

- F1-Score and mAP (mean Average Precision)

F1 Score used the two previous metrics, which are precision and recall, F1-Score is a metric that combines the precision and recall metrics into a single metric. The formula for the F1-score is defined as the average of precision and recall [27], [28]. Besides F1-score that summarizes the two previous metrics, the mean Average Precision (mAP) is the metric that shows the mean value of average precision for the detection process of all the previously determined classes [3]. Average Precision, or AP, is the average of the precision metric across all recall values between 0 and 1 at various IoU thresholds [27]. The mAP model will be one of the core metrics to determine which model has the best overall performance because it considers all previously mentioned metrics. The output of the formula will give an F1-score value ranging from 0 to 1, where 1 is the highest accuracy value.

$$\text{F1 score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \quad (3)$$

III. RESULTS AND DISCUSSION

A. Results

After doing the training proses using the Darknet model, all models resulted in a good loss average result. The training uses PCs with the latest CPU and GPU technology, and the use of PC technology will not affect the aftermath use of the models. This method has a benefit in accelerating the training

duration of the model because the darknet framework supports the GPU Acceleration method for the training phase. Thus, reducing the training time when compared to using the CPU for training. If the training is done with much less advanced technology, it will take more time to finish because the output weight file would result in the same file. The bigger the model architecture, the slower the machine can train. The size of the model also affects the model output size. Fortunately, all tested models are designed for small devices and have a small architecture that could fit Raspberry Pi.

Figure 8 informs that the graph training curve can vary depending on the model architecture. At first, most of the model's loss declined in the first 1000 iterations. However, it didn't happen to YOLO-Fastest-1.1 and YOLO-Fastest-1.1X as they have a similar architecture, with one being smaller than the other. It won't have any effects as long as it declines to a level. After a steep decline at the start, the loss starts to stabilize in a gentle curve. It shows that the model is starting to understand the given dataset. Finally, the graph shows that the loss stabilizes until the end of the iteration. This result means the trained model has learned the given hand gestures dataset without a problem. As stated, each model architecture is unique and has its own beneficial impact in certain cases. Therefore, the experiment can go on using the generated train weights. A more detailed training result can be seen in Table V. Table V informs the average loss of CNN model to determines how it will perform.

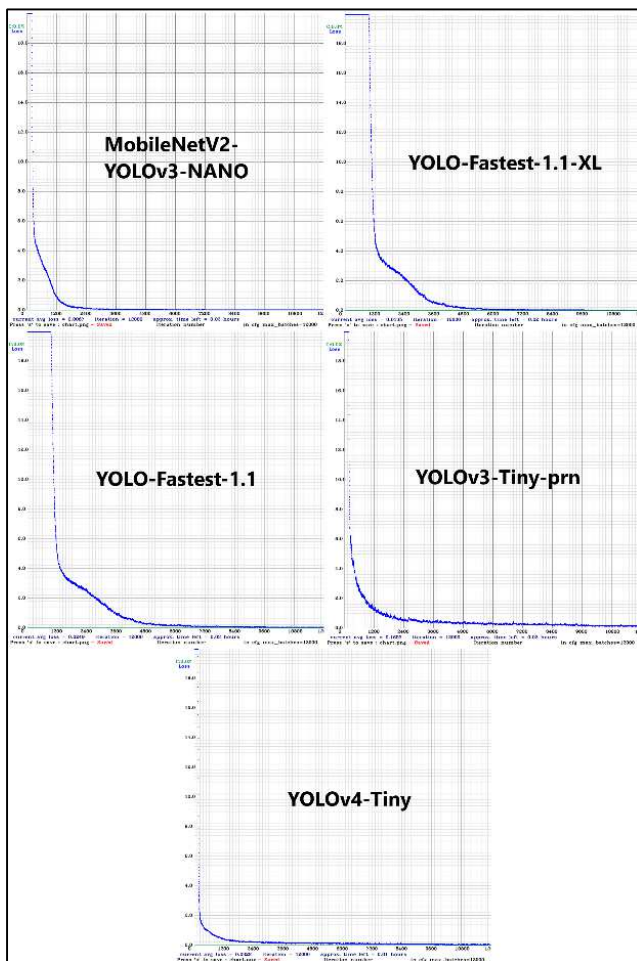


Fig. 8 Training Graph of Each CNN Models

Thus, carrying out it is one of the key parameters that could affect the test. As explained before, the lower the average loss, the better the machine understands the dataset. This way, it could potentially affect the performance of detecting objects. If the machine does not understand, it will not detect the object as expected. Table V shows that all models have an average loss below 0.2. This value is pretty low enough and acceptable for the experiment. YOLOv3-Tiny-PRN has the highest average of 0.1655. Next, YOLOv4-Tiny has the second-highest average at 0.0328. However, the difference between them is more than 0.13, which is a lot. YOLO-Fastest-1.1 comes third with an average loss of 0.0249. Then, YOLO-Fastest-1.1-XL comes after YOLO-Fastest-1.1 with an average loss of 0.0135. Last, MobileNet-YOLOv3-NANO has the lowest average loss with a value of 0.0067. However, without proper testing and analysis, the performance of a model cannot be determined just by using loss value.

TABLE V
TRAINING RESULT OF EACH CNN MODELS USING AVERAGE LOSS PARAMETER

Model	Average Loss (%)
YOLOv3-Tiny-PRN	0.1655
YOLOv4-Tiny	0.0328
YOLO-Fastest-1.1	0.0249
YOLO-Fastest-1.1-XL	0.0135
MobileNet-YOLOv3-NANO	0.0067

Before testing the weights trained in the self-service application, the hand gesture recognition algorithm needs to import the supporting files. The supporting files are the training label, image path, model configuration, and a .data file type called trainer.data. These supporting files are necessary to execute the testing process, which uses the OpenCV library as the inference. OpenCV is an open-source library mainly used for image processing [29]. Then, the self-service application and hand gesture recognition will automatically be initiated simultaneously. Next, an examination of hand gesture recognition is performed. This examination was needed to make sure that the trained hand gesture worked properly. In this case, the machine's frames captured and processed were examined in a separate window. Figures 9 are some of the UI of self-service technology that was previously made, while figure 10 shows captured images from the webcam with detected hand gestures. All of the gestures were shown to be recognized by the machine.

Figure 9 illustrates that there is a total of five menu user interfaces in the self-service technology, but the two user interfaces above can use all or most of the hand gestures. The other three menu user interfaces could only use some hand gestures. The Hi combined with Fist type gestures can be used to move up, down, left, and right around the item grid and to adjust the chosen item quantity in the second step of the process. OK hand gesture is used to move to the next menu, while the three fingers hand gesture is used to go back to the previous menu. The L hand gesture function is to select goods, and gesture C is to cancel the order, and it will automatically go back to the first menu. As previously mentioned, not all hand gestures can be used in all menus since some hand gestures are only necessary for one or two processes.

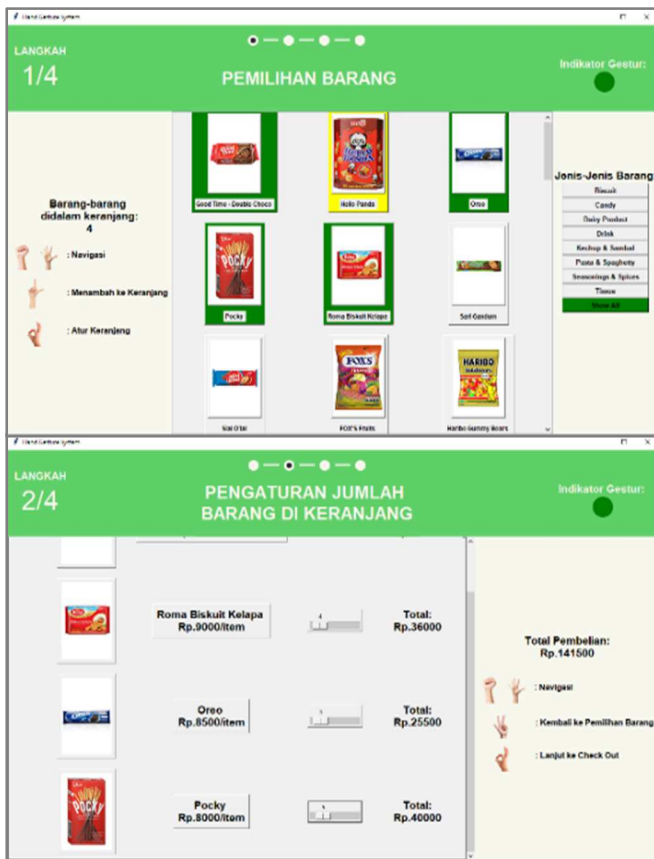


Fig. 9 Two out of Five Interfaces in the Self-Service Technology; Select Item Menu and Quantity Control Menu



Fig. 10 Recognizing an OK, Hi, C, and L Hand Gesture as an example

Figure 10 shows examples of how a CNN model recognizes hand gestures. The system will make a prediction box surrounding the hand gesture with their class type. The number located in the middle of the bounding box is an object ID. The function of an object ID is to prevent the system from recognizing one detected hand gesture as multiple different hand gesture inputs when moving around the frame. For example, a person gestures to input the machine and get the machine's recognition. The system will give the object ID to the corresponding hand. When the hand in the frame moves, it will have the same object ID after some calculations. The figure also shows that each frame of the hand recognition system shows the summary of how many of each type of hand gesture is detected, the total of hand gestures detected, and the current hand gesture present in the frame.

When the application runs on a high-end device, it can flawlessly object detections and calculations with above-acceptable performance. That means a great response feel and fast processing speed. This is required for real-time object detection to make sure everything is processed without delay between interactions. Thus, making a good self-service experience for users. Next, the models get tested in a Raspberry Pi 4B which is a small device. However, smaller size comes at the price of processing performance. When it was tested to run the same hand gesture recognition, the response fell, and the processing speed was not acceptable. So, it requires optimizations to meet acceptable performance. By default, a model has either 416x416 or 320x320 network input size. In the experiment, all of the model's network input sizes were reduced to 224x224. This is to reduce the processing load, which could increase processing time. As stated before, this benefit could come with a cost, as seen in Table VI.

Table VI informs that YOLOv3-Tiny-prn has the highest overall score in all metrics before optimization. Both recall and F1-score of YOLOv3-Tiny-prn have a 0.99 score, while the precision metric has a score of 0.98. YOLOv4-Tiny follows it with the second highest at 0.98 average. MobileNet-YOLOv3-NANO comes third with an average of 0.72. Next, YOLO-Fastest-1.1-XL has an average of 0.64. Last, YOLO-Fastest-1.1 comes with a 0.60 average. After all, models have been optimized; all variables have been reduced significantly. YOLOv3-Tiny-prn has reduced by about 51%, making it move to the third position. YOLOv4-Tiny.

On the other hand, only reduced by about 22%, which makes it the highest average after optimization. Next, MobileNet-YOLOv3-NANO has reduced to about 33%, making it the worst after optimization. YOLO-Fastest-1.1-XL has reduced to only about 11%, which is the third-best. And the last, YOLO-Fastest-1.1 has reduced to about 19%, making it the second-worst performing statistically.

TABLE VI
PRECISION, RECALL, AND F1-SCORE PARAMETERS BEFORE AND AFTER OPTIMIZATION COMPARISON

Model	Precision		Recall		F1-score	
	Before	After	Before	After	Before	After
YOLOv3-Tiny-PRN	0.98	0.65	0.99	0.34	0.99	0.44
YOLOv4-Tiny	0.98	0.83	0.98	0.69	0.98	0.76
YOLO-Fastest-1.1	0.50	0.51	0.75	0.36	0.60	0.42
YOLO-Fastest-1.1-XL	0.49	0.53	0.91	0.61	0.64	0.57
MobileNetV2-YOLOv3-NANO	0.64	0.55	0.81	0.27	0.72	0.36

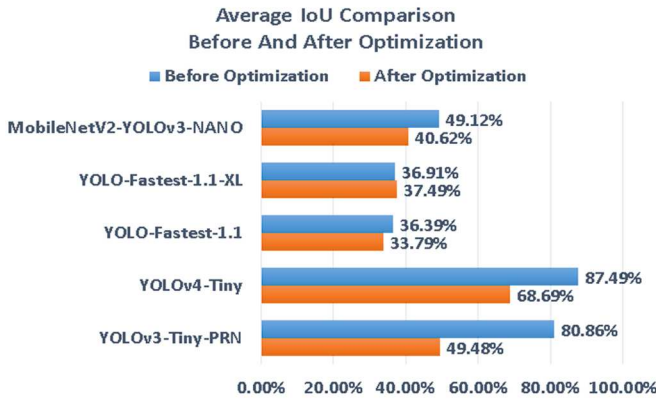


Fig. 11 CNN Models Average IoU Before and After Optimization Comparison

Figure 11 informs that each model's data results vary after the optimization process. As usual, both YOLOv4-Tiny and

YOLOv3-Tiny-PRN have suffered a significant reduction in IoU after optimization compared to the other models. YOLOv4-Tiny has a reduced percentage from 87.49% to 68.69%, which has an 18.8% reduction.

YOLOv3-Tiny-PRN has the biggest reduction in IoU, which has a 31.38% reduction after optimization. Next, MobileNetV2-YOLOv3 has reduced to about 9%, which is not as worse as the previously mentioned models. YOLO-Fastest-1.1 only reduces to about 3%, which is a very small amount compared to the other model. Even though YOLO-Fastest-1.1 has the lowest reduction, it is also the worst-performing model in this chart, with only a 33.79% IoU performance. The last model is the YOLO-Fastest-1.1-XL, the only model that sees an improvement of about 0.58%. All the information about optimization comparison can be seen in Table VI and Table VII.

TABLE VII
TABLE OF COMPARISON BETWEEN INFERENCE TIME, FPS, AND MAP@0.50

Model	Before Optimization			After Optimization		
	Inference Time (ms)	FPS	mAp@0.50	Inference Time (ms)	FPS	mAp@0.50
YOLOv3-Tiny-PRN	245.2819739	3	99.97%	80.59535243	6	43.31%
YOLOv4-Tiny	456.2688647	2	99.83%	137.9290275	4	81.66%
YOLO-Fastest-1.1	51.42538966	6	56.70%	25.54194168	10	33.51%
YOLO-Fastest-1.1-XL	106.0166445	5	67.60%	53.77346182	7	57.70%
MobileNetV2-YOLOv3-NANO	73.94764119	5	71.85%	38.79998658	9	28.17%

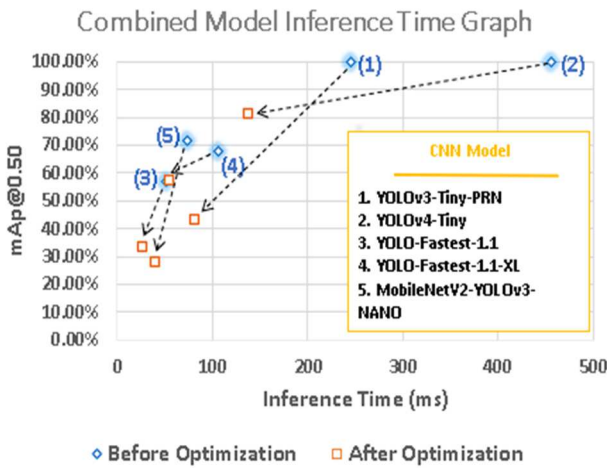


Fig. 12 Combined Model Inference Time Graph Before and After Optimization

Figure 12 illustrates that the model inference time is needed to determine whether the reduced network size also reduces the inference time. Inference time calculates the time between the captured frame and the process until it results in data in terms of object detection [30]. The bigger the inference time, the slower the detection becomes. This also worsens the experience of using this technology. In figure 12, most models show a reduction of about half the original inference time. For YOLOv4-Tiny and YOLOv3-Tiny-PRN, they reduce to about

1/3 the original. However, they still feel too slow and unsuitable for the test device due to the high value of inference time. However, mAP also needs to be considered to know more about compatibility on such devices. The mAP shows about half the amount of reduction for YOLOv3-Tiny-PRN and YOLO-Fastest-1.1. MobileNetV2-YOLOv3-NANO suffered the most reduction to about 28.17% of mAP. Last, YOLO-Fastest-1.1-XL and YOLOv3-Tiny show a slight amount of reduction after optimization. This result shows that YOLO-Fastest-1.1-XL is the most balanced in these metrics after optimization.

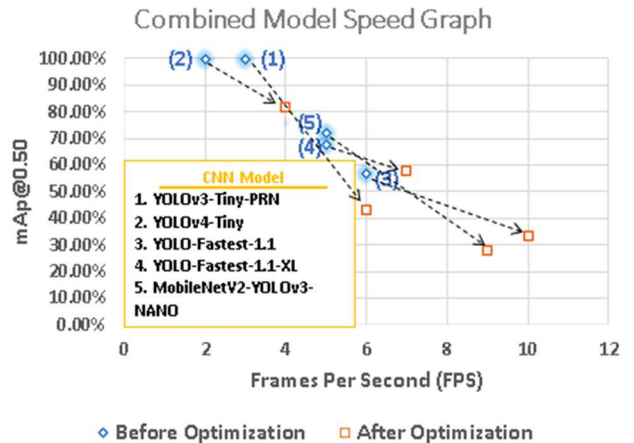


Fig. 13 Combined Model Speed Graph Before and After Optimization

Figure 13 informs that the Frames per Second (FPS) graph is similar to the inference time graph, which shows mAp@0.50 but relative to FPS. More fps means faster processing ability. This matter correlates with inference time. The lesser the inference time, the more FPS it could produce. Figure 13 shows MobileNetV2-YOLOv3-NANO has the highest FPS after optimization at 10 FPS followed by YOLO-Fastest-1.1 at 9 FPS. Both gained 4 more fps which is pretty significant. Next, YOLOv3-Tiny-PRN gained 3 more FPS, improving the average FPS from 3 to 6. YOLO-Fastest-1.1-XL gained 2 more fps from 5 to 7, making it the third-best performer. Lastly, YOLOv4-Tiny also gained 2 FPS, but it is the worst performer at an average of 4 FPS. All models above 6 FPS result in an acceptable experience in using the technology. Like before, mAP needs to be considered to find the most balanced model. In this section, YOLOv3-Tiny-PRN and YOLO-Fastest-1.1-XL are the most balanced.

Figure 14 informs that the average overall latency is equal to the average loop time, which also includes inference time. All models show a reduction in latency, and YOLOv4-Tiny experienced a significant difference, which is more than 1/3 of the original latency. Even with the amount of latency that has been reduced, the YOLOv4-Tiny still has the highest latency compared to other models. YOLOv3-Tiny-PRN's latency was also reduced to about half the original amount, making this CNN model the second-highest latency in the test, even after optimization. The other three model does not lose as much latency as YOLOv3-Tiny-PRN and YOLOv4-Tiny. YOLO-Fastest-1.1-XL has been reduced to almost half of the original value. MobileNetV2-YOLOv3-NANO is the second-lowest latency in the test, with 0.0836 seconds per loop. Last, YOLO-Fastest-1.1 has the lowest latency, with 0.0804 seconds per loop.

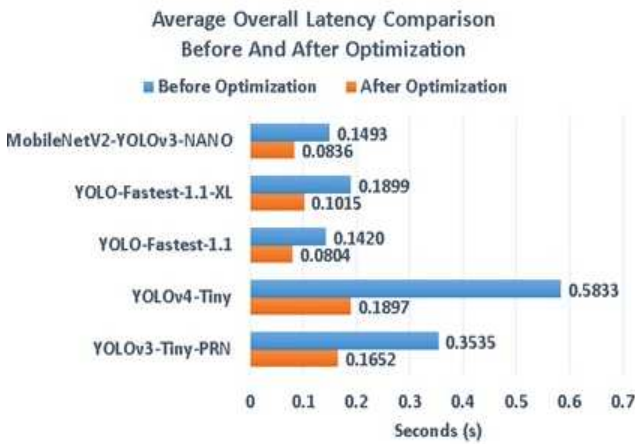


Fig. 14 CNN Models Average Overall Latency Before and After Optimization Comparison

These metrics infer time, FPS, and mAp@0.50 before and after optimization. All models show two similarities. The first similarity is that all models have a reduced inference time and mAp@0.50, with YOLOv3-Tiny-PRN having the biggest reduction in mAp@0.50 and YOLOv4-Tiny having the biggest reduction in inference time. The lowest inference time after optimization is YOLO-Fastest-1.1, and the highest is YOLOv4-tiny. Meanwhile, the highest mAp@0.50 after optimization is YOLOv4-tiny, and the lowest one is MobileNetV2-YOLOv3-NANO. The last similarity is all of

the models have an increased FPS, with YOLO-Fastest-1.1 and MobileNetV2-YOLOv3-NANO having the highest increasing FPS, which is 4. The highest FPS score belongs to YOLO-Fastest-1.1, with a score of 10 FPS. The reduction in inference time and the gain in FPS don't mean better for the tested model. Because other factors also play a role in its overall performance of it. This explanation will be explained further in the discussion.

B. Discussion

In the experiment, there were a total of five models that are being tested for their performance in a self-service technology using a Raspberry Pi device. Each CNN model has its unique architecture, and they all result in different metrics values from each other. These differences will be the key components for comparing the five CNN models and determining which is best suited for self-service technology. In Table VI, all the CNN models suffered a decreasing value in the Precision, Recall, F1-Score, and average IoU (See Figure 11 for average IoU). Only YOLO-Fastest-1.1-XL received an improvement score of average IoU from 36.91% to 37.49%, with an improvement value of 0.58%. The reduction in Precision, Recall, F1-Score, and average IoU metrics affects the accuracy of the CNN model. The changes in how a model detects an object within a camera frame after optimization is statistically shown by these metrics.

TABLE VIII
AVERAGE DIFFERENCE OF EVERY PARAMETER

Parameter	Average Difference
FPS	3
mAP	-30.32%
Inference Time	-119,2601
Precision	-10.40%
Recall	-43.40%
F1-Score	-27.60%

Table VIII illustrates that all metrics experienced a value reduction except for FPS. These metrics need to be looked at to determine the performance. The parameter for FPS, mAP, inference time (ms), Precision, Recall, and F1-Score are resulting in a significant difference after optimization in every model (See Table VIII). Metrics such as FPS and inference time would affect the result performance of a model, especially on a small device. Inference time is the only one that has a performance improvement because lesser inference time means the model becomes lighter to process. A higher FPS means more ability to process frames in a second. The reduction in Precision, Recall, F1-Score, and mAP would reduce the accuracy when detecting the chosen object. Therefore, these statistics show that optimization only benefits the FPS and inference time metrics but affects overall accuracy.

Figure 15 shows how mAP affects the model's ability to recognize objects in real-time without looking at latency stats. The test uses only two hand gestures for the test, which are Hi Gesture and L-Shaped Gesture. Hypothetically, if the mAP of a model is low, the model's ability to recognize the object will also be low. Figure 15 shows two models, which are YOLOv4-Tiny and MobileNetV2-YOLOv3-NANO. The first one has an mAP of 81.66%, and the other has an mAP of 28.17%. The figure tells that YOLOv4-Tiny can detect both

gestures correctly with no False-Positive. For MobileNetV2-YOLOv3-NANO, it shows the result of False-Negative on both gestures. False-negative detection indicates the model's inability to detect the object. Therefore, mAP may affect the performance of detecting objects in real-time. The higher the mAP, the better the machine can detect objects. Finally, other metrics such as FPS and inference time combined with this discovery would tell more about the result of the overall performance of a model.

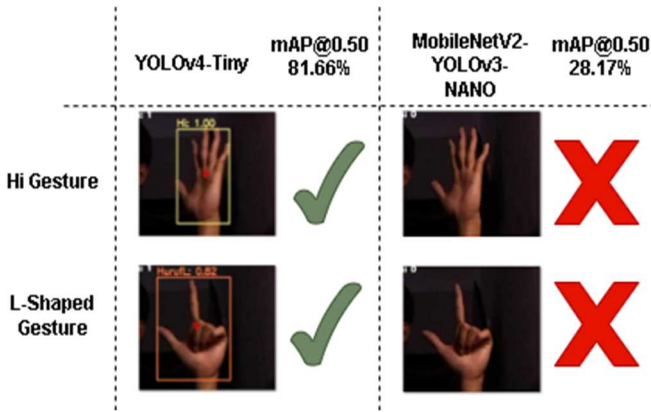


Fig. 15 How mAP Affects the Model Prediction Accuracy when Recognizing the Chosen Object.

As stated before, the lesser the inference time, the lighter the model. This, in turn, increases the FPS. The decrease in inference time also decreases the overall latency. However, not all CNN models will have an acceptable response. Only models with an overall latency of 0.1 seconds or less are acceptable. This means MobileNetV2-YOLOv3-NANO, YOLO-Fastest-1.1-XL, and YOLO-Fastest-1.1 have an acceptable overall latency that is suitable to use on Raspberry Pi devices. But, YOLOv4-Tiny and YOLOv3-Tiny-PRN have an overall latency value of more than 0.1 seconds. Making the two models unsuitable to use in such a device. But when combined with mAP, YOLO-Fastest-XL has the most balanced value between accuracy and performance that is suitable for Raspberry Pi. Anything smaller and slower than Raspberry is recommended to use YOLO-Fastest-1.1. It is not recommended to use MobileNetV2-YOLOv3-NANO due to unable to detect properly trained objects (See Figure 15) which potentially ruins the self-service technology experience.

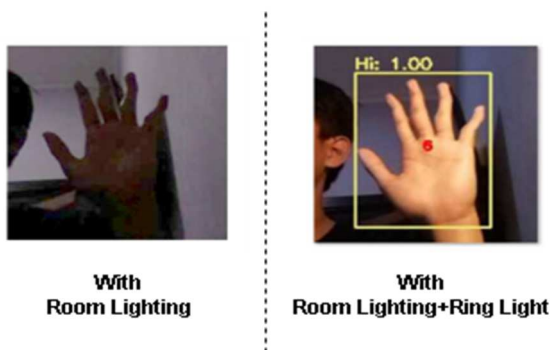


Fig. 16 How Lighting Affects CNN Model (YOLOv4-Tiny) Ability to Detect Chosen Object

Figure 16 illustrates the object detection process, and lighting could affect the technology's performance. The lesser

light in the environment causes a lower accuracy score. So lighting also affects the recognition process of the hand gesture. Just like how human eyes work, most people will have a hard time recognizing something they see. In this case, the camera is the eye for the machine to see the world. This test used YOLOv4-tiny, which has the highest mAP compared to the other model. For the "With Room Lighting Only" case, all lights in the room are on without additional lighting for the camera. The figure shows the machine unable to detect a given hand gesture.

On the other hand, the other case is the same but with the help of a ring light behind the camera. The ring light helps the machine to detect the correct gesture, as in Figure 16. Therefore, this test shows even when a model has high mAP, it is not easy for the machine to detect the gestures under dim light. It is recommended to have sufficient light when using object detection to help the machine recognize objects.

This analysis shows YOLO-Fastest-1.1-XL as the most balanced and YOLO-Fastest-1.1 as the best performer compared to the other CNN models according to the testing data. Based on Table VII, YOLO-Fastest-1.1 provided a better FPS performance than YOLO-Fastest-1.1-XL. This means that YOLO-Fastest-1.1 is a lightweight model suitable for slower devices than Raspberry Pi 4. The only disadvantage is that it would be hard to detect objects due to the low value of mAP. Fortunately, it is still better than MobileNetV2-YOLOv3-NANO in mAP, FPS, and Inference Time.

On the other hand, YOLO-Fastest-1.1-XL has a 24.19 mAP value with similar overall latency compared to YOLO-Fastest-1.1. Therefore, this model is suitable for use in Raspberry Pi 4. YOLOv3-Tiny-PRN and YOLOv4-Tiny, these models are suitable for devices that are faster than Raspberry Pi 4. To support more of these discoveries, a more complex dataset, CNN models that are proven to be a lot more efficient and compatible with small devices, and some tweaks in another area could potentially improve the performance of this kind of research on optimizing object detection performance.

IV. CONCLUSION

This study uses six different chosen hand gesture images as a training dataset to carry out the storage process. The process uses the collected dataset as training for the darknet and imports it to Open CV and other support files to run real-time object detection. All models suffered a reduction in Precision, Recall, and F1-Score. But for IoU, YOLO-Fastest-1.1-XL is the only model that sees an improvement of about 0.58%. Moreover, the optimization result is only beneficial to the FPS and inference time metrics, which improved by an average of 3 FPS and reduced by an average of 119,260 ms than before optimizing all CNN models.

However, room lighting and camera quality affect the performance of real-time object detection on detecting objects. It is recommended to have sufficient lighting when using object detection technology. Two models that show better behavior are YOLO-Fastest-1.1-XL which has the most balanced result, and YOLO-Fastest-1.1, which shows lower accuracy but higher FPS and is suitable for a smaller device. A more complex dataset for training, CNN models that are proven to be a lot more efficient, and some tweaks on some areas might improve the performance even more.

REFERENCES

- [1] H. El-Aawar, "Increasing The Transistor Count by CONSTRUCTING A Two-Layer Crystal Square on A Single Chip," *Int. J. Comput. Sci. Inf. Technol.*, vol. 7, no. 3, 2015, doi: 10.5121/ijcsit.2015.7308.
- [2] R. Singh, "An approach to enhance performance of Computer-Literature Review," *Int. J. Sci. Dev. Res.*, vol. 1, no. 5, 2016, Accessed: Jun. 30, 2022. [Online]. Available: www.ijdsr.org.
- [3] Z. Jiang, L. Zhao, S. Li, Y. Jia, and Z. Liqun, "Real-time object detection method based on improved YOLOv4-tiny," Nov. 2020, doi: 10.48550/arxiv.2011.04244.
- [4] A. H. Rangkuti, V. H. Athala, N. F. Luthfi, S. V. Aditama, and J. M. Kerta, "Reliable of traditional cloth pattern Classification Using Convolutional Neural Network," *2021 2nd Int. Conf. Artif. Intell. Data Sci. AiDAS 2021*, 2021, doi: 10.1109/AiDAS53897.2021.9574402.
- [5] E. Considine and K. Cormican, "Self-service Technology Adoption: An Analysis of Customer to Technology Interactions," *Procedia Comput. Sci.*, vol. 100, pp. 103–109, Jan. 2016, doi: 10.1016/J.PROCS.2016.09.129.
- [6] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," Apr. 2020, doi: 10.48550/arxiv.2004.10934.
- [7] N. F. Thejowahyono, M. V. Setiawan, S. B. Handoyo, and A. H. Rangkuti, "Hand Gesture Recognition as Signal for Help using Deep Neural Network," *Int. J. Emerg. Technol. Adv. Eng.*, vol. 12, no. 2, pp. 37–47, 2022, doi: 10.46338/ijetae0222_05.
- [8] C. Nyaga and R. Wario, "A Review of Sign Language Hand Gesture Recognition Algorithms," *Adv. Intell. Syst. Comput.*, vol. 1213 AISC, pp. 207–216, 2021, doi: 10.1007/978-3-030-51328-3_30.
- [9] V. Moysiadis *et al.*, "An Integrated Real-Time Hand Gesture Recognition Framework for Human–Robot Interaction in Agriculture," *Appl. Sci.*, vol. 12, no. 16, 2022, doi: 10.3390/app12168160.
- [10] M. Peral, A. Sanfeliu, and A. Garrell, "Efficient Hand Gesture Recognition for Human-Robot Interaction," *IEEE Robot. Autom. Lett.*, vol. 7, no. 4, pp. 10272–10279, Oct. 2022, doi: 10.1109/LRA.2022.3193251.
- [11] Q. Gao, J. Liu, and Z. Ju, "Hand gesture recognition using multimodal data fusion and multiscale parallel convolutional neural network for human–robot interaction," *Expert Syst.*, vol. 38, no. 5, Aug. 2021, doi: 10.1111/EXSY.12490.
- [12] N. M. Mahmoud, H. Fouad, and A. M. Soliman, "Smart healthcare solutions using the internet of medical things for hand gesture recognition system," *Complex Intell. Syst.*, vol. 7, no. 3, pp. 1253–1264, Jun. 2021, doi: 10.1007/S40747-020-00194-9.
- [13] E. Spandana, M. Rajasekar, and N. Sandhya, "Care-giver alerting for bedridden patients using hand gesture recognition system," *J. Phys. Conf. Ser.*, vol. 1921, no. 1, 2021, doi: 10.1088/1742-6596/1921/1/012077.
- [14] S. Ameur, A. Ben Khalifa, and M. S. Bouhlel, "Hand-Gesture-Based Touchless Exploration of Medical Images with Leap Motion Controller," *Proc. 17th Int. Multi-Conference Syst. Signals Devices, SSD 2020*, pp. 1116–1121, Jul. 2020, doi: 10.1109/SSD49366.2020.9364244.
- [15] N. Zengeler, T. Kopinski, and U. Handmann, "Hand gesture recognition in automotive human–machine interaction using depth cameras," *Sensors (Switzerland)*, vol. 19, no. 1, Jan. 2019, doi: 10.3390/S19010059.
- [16] H. Feng, G. Mu, S. Zhong, P. Zhang, and T. Yuan, "Benchmark Analysis of YOLO Performance on Edge Intelligence Devices," *Cryptography*, vol. 6, no. 2, pp. 1–16, 2022, doi: 10.3390/cryptography6020016.
- [17] K. Ntzelepi, M. E. Filippakis, M. E. Poulou, and A. Angelakis, "Performance Evaluation of YOLOV4 and YOLOV4-TINY for Real-Time Face-Mask Detection on Mobile Devices," *Int. J. Artif. Intell. Appl.*, vol. 13, no. 3, 2022, doi: 10.5121/ijaa.2022.13303.
- [18] C. Sager, C. Janiesch, and P. Zschech, "A survey of image labelling for computer vision applications," *J. Bus. Anal.*, vol. 4, no. 2, pp. 91–110, 2021, doi: 10.1080/2573234X.2021.1908861/SUPPL_FILE/TJBA_A_1908861_SM4490.DOTX.
- [19] L. Budagyan and R. Abagyan, "Weighted quality estimates in machine learning," *Bioinformatics*, vol. 22, no. 21, pp. 2597–2603, 2006, doi: 10.1093/bioinformatics/btl458.
- [20] A. Anton, N. F. Nissa, A. Janiati, N. Cahya, and P. Astuti, "Application of Deep Learning Using Convolutional Neural Network (CNN) Method For Women's Skin Classification," *Sci. J. Informatics*, vol. 8, no. 1, pp. 144–153, 2021, doi: 10.15294/sji.v8i1.26888.
- [21] J. Redmon, "Darknet: Open Source Neural Networks in C." 2013, Accessed: Jun. 21, 2022. [Online]. Available: <https://pjreddie.com/darknet/>.
- [22] Dog-qiuqiu, "dog-qiuqiu/Yolo-Fastest: yolo-fastest-v1.1.0," *Zenodo*, Jul. 2021, doi: 10.5281/ZENODO.5131532.
- [23] A. H. Rangkuti, V. H. Athala, E. Tanuar, and J. M. KERTA, "Enhancing A Reliable Traditional Clothes Pattern Retrieval : CNN Model and Distance Metrics," vol. 100, no. 10, pp. 3183–3193, 2022, [Online].
- [24] A. H. Rangkuti, V. H. Atthala, E. Tanuar, and J. M. Kerta, "Performance Evaluation of traditional Clothes pattern retrieval with CNN Model and Distance Matrices," pp. 1–9, 2020.
- [25] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 779–788, 2016, doi: 10.1109/CVPR.2016.91.
- [26] S. Makahaube, A. M. Sambul, and S. R. U. A. Sompie, "Implementation of Gesture Recognition Technology for Self-Education Service Platform," *J. Tek. Inform.*, vol. 16, no. 4, pp. 465–472, Oct. 2021, doi: 10.35793/JTI.16.4.2021.34210.
- [27] Esri, "How the Compute Accuracy For Object Detection tool works," *ArcGIS Pro, Esri*, 2020, [Online]. Available: <https://pro.arcgis.com/en/pro-app/latest/tool-reference/image-analyst/how-compute-accuracy-for-object-detection-works.htm>.
- [28] A. F. Gad, "Accuracy, Precision, and Recall in Deep Learning," 2020, [Online]. Available: <https://blog.paperspace.com/deep-learning-metrics-precision-recall-accuracy/>.
- [29] M. Naveenkumar and V. Ayyasamy, "OpenCV for Computer Vision Applications," *Proc. Natl. Conf. Big Data Cloud Comput.*, no. March 2015, pp. 52–56, 2016, [Online]. Available: https://www.researchgate.net/publication/301590571_OpenCV_for_Computer_Vision_Applications.
- [30] I. Martinez-Alpiste, G. Golcarenarenji, Q. Wang, · Jose, and M. Alcaraz-Calero, "Smartphone-based real-time object recognition architecture for portable and constrained systems," *J. Real-Time Image Process.*, vol. 19, pp. 103–115, 2022, doi: 10.1007/s11554-021-01164-1.