Development of Fish Variant Multi Object Detection System for Video based on YOLOv7

Muhammad Daffa Ajiputra  
Computer EngineeringUniversitas Indonesia  
muhammad.daffa99@ui.ac.id

Prima Dewi Purnamasari  
Computer EngineeringUniversitas Indonesia  
prima.dp@ui.ac.id

*Abstract*— Indonesia is one of the world's largest exporters of fish, which exposes Indonesia's fishing sector to many threats. Illegal, unreported, unregulated (IUU) fishing is one of the problems that resulted in a significant impact in a form of a big loss that is created for the Indonesian fisheries sector. To prevent that problem, there are a lot of solutions that have been proposed, one of which is the application of technology such as surveillance cameras, but it still doesn't have a big impact to reduce and eliminate IUU fishing. Therefore, this research is conducted to develop a multi-object detection system for the detection of fish variant based on YOLOv7, an artificial intelligence model that can detect a fish to supervise the number of fish that is caught by the fisherman so IUU fishing can reduce significantly. From the testing, the YOLOv7 model becomes the best YOLOv7 model variant that can be used to detect a fish with the value of mAP that can reach up to 86.1% and the value of inference time up to 14.5 ms that can produce an FPS total up to 69 FPS. The value can be achieved by doing some modifications in data annotation, the training model method, image size, and iteration on training. However, the YOLOv7 model has a very slow inference time up to 797.6 ms when it’s installed in Jetson Nano even though the detection accuracy has the same value.

Key words: deep learning, computer vision, object detection, YOLOv7

# Introduction

Indonesia is one of the world's largest fish exporting countries. With a value of 558 million US dollars [1], Indonesia is rated as the world's seventh largest fish exporting country for filleted and frozen fish in 2021. Tuna is one of Indonesia's high-value fish export commodities. The value of Indonesian tuna exports in 2021 was 325.4 million US dollars, with an export volume of 48.3 Mega Tons for filleted and frozen tuna [2].

On the other hand, due to the high number of Indonesian marine export commodities, the country is not immune to occurrences of illegal, unreported, and unregulated (IUU) fishing. In 2021, the Ministry of Maritime Affairs and Fisheries apprehended 167 Indonesian and foreign ships, including ships from Vietnam and Malaysia, for IUU fishing [3]. It is estimated that illegal, unreported, and unregulated fishing cost Indonesia 74 million US dollars in 2021 [4]. The Ministry of Maritime Affairs and Fisheries has developed multiple strategies to prevent IUU fishing on Indonesian territory. For example, eight Malaysian ships were sunk owing to unlawful fishing in March 2021 [5]. Furthermore, ASEAN countries are collaborating to promote information disclosure in the fisheries industry through ship monitoring systems as well as electronic reporting and surveillance systems.

The application of technology itself has become one of the strategies used to prevent crime in a variety of industries, with computer vision being one example. Computer vision is an artificial intelligence (AI) field that enables computers to see and comprehend crucial information in photos, movies, and other visual inputs [6]. Various approaches to utilizing computer vision in various sectors of life have been developed up to this point, for example object detection.

Object detection is a computer vision algorithm that detects objects in images or videos [7]. The object detection algorithm was discovered in 2001 by the Viola-Jones duo called VJ-Detectors, which were still manufactured manually at the time. When Alexnet was invented in 2012, object detection moved to employ automation approaches. Many deep learning-based object detection algorithms have been found since then, including YOLO (You Only Look Once). Joseph Redmon was the first to discover YOLO in 2016. The YOLO algorithm is still evolving, and there is presently a seventh version of the YOLO algorithm (YOLOv7) which is a development of the fifth version of YOLO.

YOLOv7 is the most recent version of the YOLO algorithm. The primary goal of YOLOv7 development is to reduce the number of parameters and computing processes required to build a machine learning model while maintaining detection speed and accuracy. According to published papers, YOLOv7 outperforms various previously established algorithms, including YOLOR, YOLOX, Scaled-YOLOv4, YOLOv5, DETR, and others, in terms of detection speed and accuracy [8].

Several research have been conducted to detect fish variant utilizing object detection with the YOLO algorithm before YOLOv7 such as YOLOv4 and YOLOv5. For example, in 2022, the YOLOv4-Tiny algorithm was used to detect fish variant in an aquarium using a Raspberry Pi 4 [9]. During the same year, researchers used YOLOv5s, YOLOv5m, and YOLOv5l to detect fish variant beneath water [10]. The majority of these investigations are aimed at detecting fish that are not export commodities, such as tuna. Furthermore, because the viewing angle of the image used as a dataset is extremely close to the fish, there is a risk that the item cannot be categorized if the shape of the object does not match or the distance of the object is too great.

Given this background, this study was carried out to develop a multi-object fish-detecting system in a movie utilizing the YOLOv7 algorithm. This detection system will detect a variety of items, including humans and different sorts of fish, in the video clip captured by the ship's camera. The camera's field of vision will be pointed in one direction, which is where the fish is stored, allowing all operations, including fishing and fish storage, to be monitored. Later, the model can be modified to calculate the number of fish caught by the fishermen on the vessel, so that if there is a disparity between the number of fish detected and the number of fish reported, IUU fishing on the vessel is identifiable.

# Literature Review

## Deep Learning

Deep learning is a machine learning subfield that employs numerous layers of neural networks [11]. Deep learning is an evolution of the standard machine learning method, which still employs a feature extractor to retrieve features from an input. The deep learning algorithm already has a feature extractor in its neural networks layer, allowing the deep learning algorithm to automatically extract features from input. Deep learning's architecture is based on multilayer perceptron. The multilayer perceptron operates by stacking neurons into multiple levels known as hidden layers. The layers in the hidden layer are linked together via weighted connections. A multilayer perceptron's major components are:

1. The input layer, which consists of features owned by input data.

2. Hidden layers, which are layers made up of neurons that extract feature vectors to be processed in the input layer.

3. Weight connections (edges), which is the value contained in each association between nodes in neural networks to reflect their importance in the prediction outcomes.

4. The output layer, which is the outcome of hidden layer processing and might take the shape of an actual value or a collection of probabilities.

Deep learning possesses three stages that must be executed continually in order to produce the best model. The first stage involves calculating the weights sum of each node and activation to provide a forecast, also known as the feedforward process [12]. The weights sum is the sum of the weight vectors in each perceptron. The weights total of each node will be multiplied by a non-linear activation function to obtain the prediction results for each input. The second stage involves calculating the error rate of the neural network prediction results in comparison to the expected output using an error function, which is also known as a loss function. The third stage involves using optimization functions such as gradient descent or other types to iteratively modify neural networks to the most optimal point, which is accomplished by calculating the derivative of the loss function against weights using a chain rule from the output layer to the input layer, also known as backpropagation.

## Object Detection

Object detection is one example of a real-world application of computer vision technology. Object detection is an image classification advancement in which object detection functions to enable interpretation devices used in computer vision systems to determine the location and number of things contained in an image. Object detection, in general, will identify the location and class of objects by dividing the image into smaller portions and giving a class to each broken part, allowing the items present in the image to be named [13]. The object detection framework is made up of four major components:

### Region Proposals

A region proposal is a section provided by the deep learning model in the form of a region of interest (RoIs) that will be further analyzed by the system. A region of interest is a section of an image that the system believes contains an object based on the objectness score. At this stage, the system will create thousands of bounding boxes for neural networks to evaluate and classify. The objectness score of each bounding box is the output of neural network analysis, which determines whether the bounding boxes are in the foreground (object) or background (non-object). If the bounding boxes pass the threshold established by the neural networks, they are categorized as foreground and advanced to the next level.

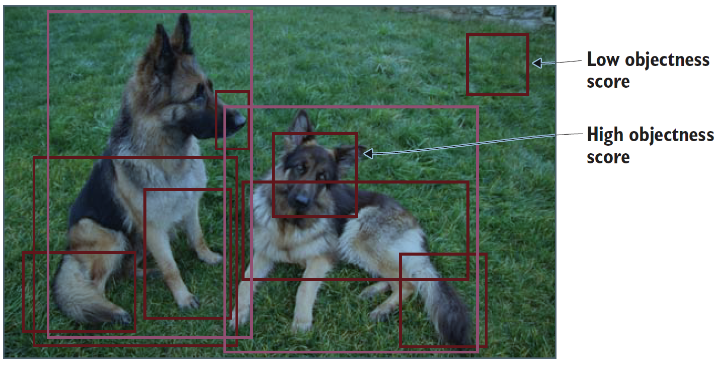


Fig. 1. Example of Region of Interest in a dog image [13]

### Feature Extraction and Network Prediction

This component is the continuation of the region proposal in which the features of the bounding boxes in the foreground region are retrieved and used to determine the class of the detected object in the image. A pre-trained image classification model will extract the characteristics in the foreground region so that the results obtained are equitably generalized. Following feature extraction, the foreground region is examined by neural networks, and two predictions are made for each foreground region, that is:

* Bounding-box prediction, which predicts the location of the foreground region in an image. The forecast will be in the form of a tuple (x,y,w,h), where x and y are the coordinates of the foreground region's center point, and w and h are the region's length and width.
* Class prediction, which predicts the likelihood of each class trained for the object in the image.

### Non-maximum Suppression (NMS)

Non-maximum suppression (NMS) is a component that ensures that each item has just one bounding box. The component works by searching all bounding boxes that point to an object for those with the highest prediction probability and eliminating those with a lower prediction probability. Here is how non-maximum suppression works in determining the highest prediction probability:

* Remove all bounding boxes whose prediction probabilities do not meet the specified confidence threshold.
* Looks through all of the remaining bounding boxes and chooses the ones with the highest probabilities.
* Calculates and compares the intersection regions between the expected bounding boxes and the ground truth bounding boxes. The value of the comparison is known as intersection over union (IoU).
* Remove bounding boxes with IoU values less than the threshold.

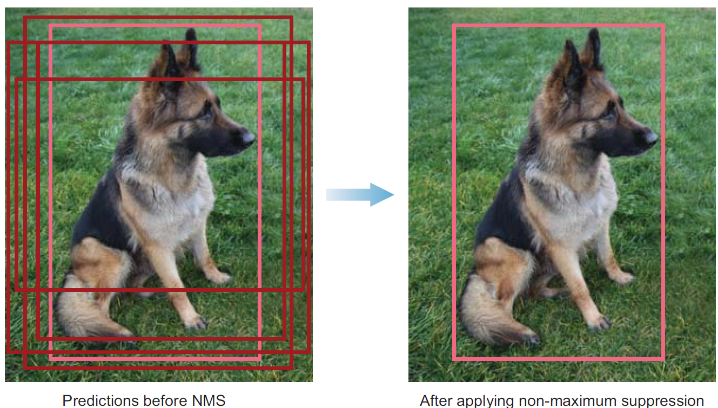


Fig. 2. Prediction results before and after applying non-maximum suppression [13].

### Evaluation Metrics

This component will examine the prediction outcomes from the previous three components. These units are frequently used in evaluating the prediction outcomes of an object detection model:

* *Frames Per Second (FPS)*

Frames Per Second is an evaluation unit that assesses how quickly an object detection model detects objects. The number of frames per second (FPS) can be estimated using the inference time value and the equation:

|  |  |
| --- | --- |
|  | (1) |

* *Intersection over Union (IoU)*

Intersection over Union is an assessment unit that will evaluate the overlap between the expected and ground truth bounding boxes. IoU will assess if the detection result is valid (True Positive) or not (False Positive) using a value range of 0-1, with the bigger number indicating a better detection result. The IoU value of a bounding box can be calculated using the following equation:

|  |  |
| --- | --- |
|  | (2) |

* *Precision-Recall Curve (PR-Curve)*

Precision-recall Curve is a comparison curve for the precision and recall of an object detection model that will be processed for each confidence level. Precision is an evaluation unit that evaluates a model's ability to identify a relevant object in an image. The precision of an object detection model can be calculated using the following equation:

|  |  |
| --- | --- |
|  | (3) |

Then there's recall, which is an evaluation unit that assesses the model's capacity to detect all important things in an image. The recall value can be calculated using the following equation:

|  |  |
| --- | --- |
|  | (4) |

* *Mean Average Precision (mAP)*

We can calculate the average precision (AP) from the precision-recall curve, which is the average precision value for each class in the training model based on the threshold computed using the value of interpolated precision for each recall level. We may calculate the average precision value by averaging interpolated precision over recall for each class. The average precision value simply shows the model's class's average precision. In general, an object detection model contains many object classes, hence that the average precision of all classes must be determined to assess the overall accuracy of the model, which is generally referred to as mean Average Precision (mAP). The following equation gives the mean Average Precision (mAP) value:

|  |  |
| --- | --- |
|  | (5) |

The mAP value is often defined by the IoU threshold used, such as mAP0.5 and mAP0.5:0.95. The mAP0.5 value is the average of each class's AP when the IoU threshold is 0.5. Then, mAP0.5:0.95 is one of the assessment parameters in the COCO 2017 challenge, where the mAP value is the average result of the AP of each class at 10 IoU thresholds ranging from 0.5 to 0.95 with a step size of 0.05.

* *F1-Score*

F1 Score is a metric used to calculate the harmonic mean value of precision and recall. The F1 score has a value range of 0 to 1. The higher the F1 score value, the better the model's precision and recall, as well as its detection accuracy. The following equation can be used to get the F1 score:

|  |  |
| --- | --- |
|  | (6) |

## Convolutional Neural Networks

Convolutional neural networks are neural networks which utilize convolution operations to build their architecture. Convolution is a mathematical linear process that takes two functions and creates a new function. Convolutional neural networks are typically utilized for data that can be applied to a grid cell, such as time series data defined by 1D grid data or images defined as 2D data from a pixel [12]. Convolutional neural network architecture is generally separated into three primary layers, including:

### Convolutional Layers

Convolutional layers are layers that extract information from an image in order to identify objects in the image. Convolutional layers extract features from images by using a window called convolutional filters, which examines all pixels in the image. Each filters have their own weights for each feature in the image and will perform dot product operations between the filter and a specific feature, which will be summed when all dot product operations are completed. These layers will generate a new image known as a convolved image or feature map, the features of which are generated from the dot product operation between the filter and the features of the input image.

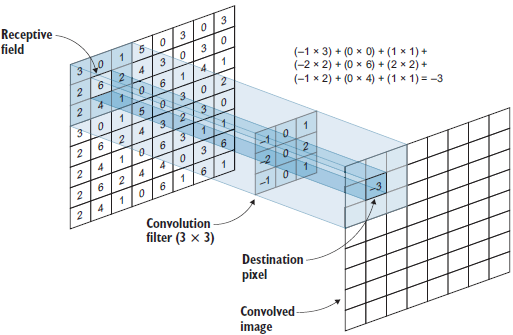


Fig. 3. Convolutional layers feature extraction process [13]

### Pooling Layers

Pooling layers are layers used to reduce the number of parameters so that the computing process is not too difficult. Pooling layers will reduce the feature map using statistical operations such as max or average to reduce the number of parameters that will be transferred to the next layer, where only important parameters will be passed to the next layer. Pooling layers will only change the feature map values without reducing the amount of feature maps created by convolutional layers. There are two types of pooling layers, including:

* *Max Pooling*

Max pooling is a type of pooling layer that examines the feature map created by convolutional layers and takes the highest feature value in the window to create a new feature map that only contains the pooled feature value.

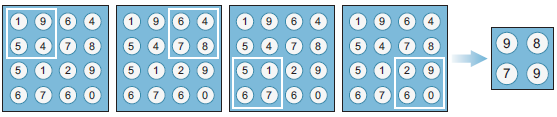


Fig. 4. Example of a max pooling feature map of size 4 X 4 with a 2 x 2 window [13]

* *Average Pooling*

Average Pooling is a type of pooling layer that examines a convolutional layer's feature map and takes the average of the feature values in the window.

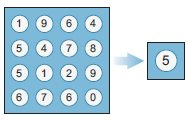


Fig. 5. Example of a global average pooling feature map of size 4 X 4 [13]

### Fully Connected Layers

Fully connected layers are used as classification layers. This layer will receive and handle the results of feature extraction performed on convolutional and pooling layers, allowing the extraction results to be used for classification. The input flattened vector, hidden layer, and output layer are the three main components of the fully connected layers architecture.

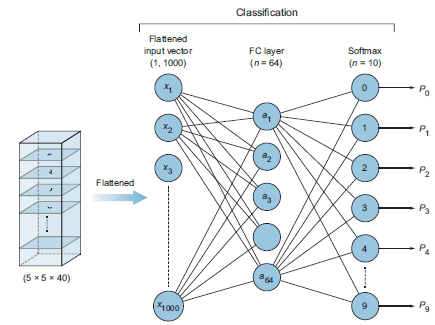


Fig. 6. Fully connected layers architecture [13]

## YOLOv7

YOLOv7 is a model development from the YOLO series that was released in 2022. The development of YOLOv7 focuses on how to optimize a model's training process by strengthening training costs to improve accuracy without raising inference costs by using optimized modules and optimized procedures. YOLOv7 model outperforms YOLOv5, YOLOR, YOLOX, PPYOLOE, and other models in tests. For example, YOLOv7-Tiny-SiLU has a 10.7% higher average precision value than YOLOv5-N, with a faster inference result of 127 FPS [8]. The YOLOv7 architecture is separated into two sections: the backbone and the head. The backbone component of the architecture will execute feature extraction to obtain the item to be identified, while the head component of the architecture will perform detection on the object detection model connected by the neck component [14]. YOLOv7 is divided into two types: YOLOv7-P5 and YOLOv7-P6. The key difference between the two types is the device used to execute the model, with YOLOv7-P5 being more typically utilized on consumer or edge GPU devices and YOLOv7-P6 being more commonly used on GPUs for Cloud Servers. YOLOv7-P5 includes three variants: YOLOv7-Tiny, YOLOv7, and YOLOv7-X.

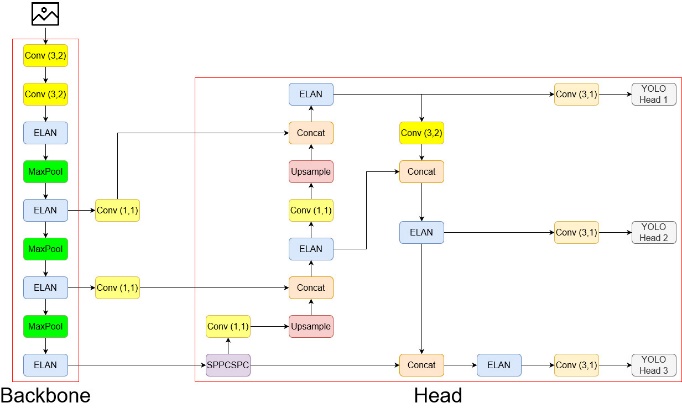


Fig. 7. YOLOv7-Tiny Architecture (Draw based on [8])

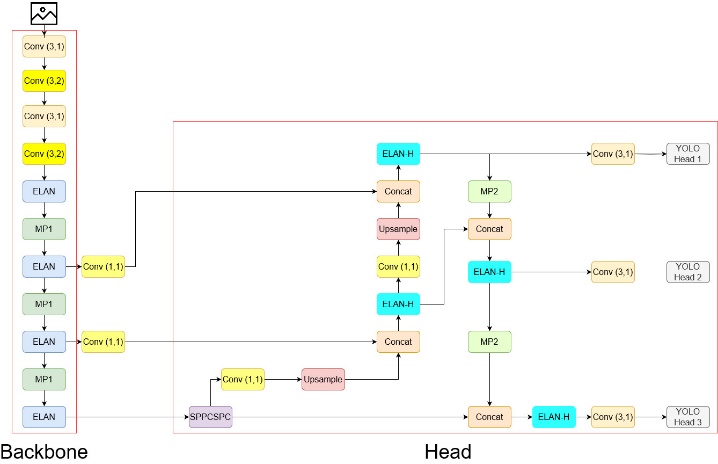


Fig. 8. YOLOv7 Architecture (Draw based on [8])

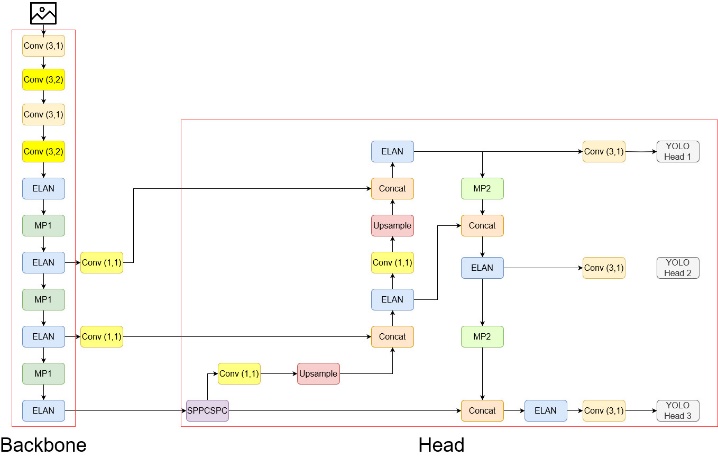


Fig. 9. YOLOv7-X Architecture (Draw based on [8])

## Polygon Annotation

Polygon annotation is a method of labeling data for machine learning model development that requires determining the x and y coordinates of each edge point of the observed item [15]. The final label shape will have the same level of precision as the original object because the x and y coordinates of each object's edge points are determined. As a result, the polygon annotation approach is often used in real life to annotate object detection or object recognition model data, such as in CT scans, monitoring plant development, calculating the cost of fixing car damage, and so on.



Fig. 10. An example of polygon annotation on a damaged car [15]

Polygon annotation provides various advantages, including flexibility when applied to an odd form and excluding pixels that are not part of the object from the label to improve detection results. However, the time required to annotate increases with the complexity of the annotated object, and not all annotators support polygon annotation on objects with holes, such as tires, donuts, and others.

# Research Methodology

## System Requirements

In general, the fish variant detection system includes several hardware and software requirements. In terms of hardware, a system requires two components: a computer to load the fish variant detection model and a camera to serve as a detection tool as well as data collecting for model training and testing. In terms of software, various libraries are required for model execution, including OpenCV, PyTorch, Torchvision, Numpy, Matplotlib, Pillow, PyYAML, Requests, Scipy, Tqdm, Protobuf, and others. Fig. 11 illustrates the system design visually.

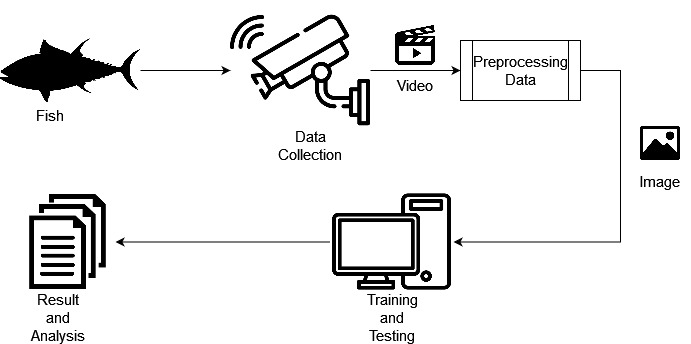


Fig. 11. Design system for detecting fish variant

A computer will be used to run the trained and tested the models. Table 1 shows the computer specifications used for creating the model.

TABLE. 1 COMPUTER SPECIFICATIONS FOR TRAINING MODEL

|  |  |
| --- | --- |
| ***Central Processor Unit*** | Intel i5-13600K @ 3.50 GHz (20 CPUs) |
| ***Graphics Processing Unit*** | NVIDIA GeForce RTX 3070 |
| ***Memory*** | 32 GB DDR5 |
| ***Operating System*** | EndeavourOS Arch Linux |
| ***Python Version*** | 3.8.16 |

## Data Collection

Data for the YOLOv7 model variant's training and testing is collected from camera captures used on the ship. The data will be in the form of a video file saved in the storage memory of the camera used on the ship. If the data from the video clip for each fish variant is still insufficient, the remaining data will be obtained from the Fishnet dataset, which seeks to equalize the data ratio between classes in the dataset. The data is in the form of an image that will be adjusted to the type of fish that does not have enough data, and the data in the Fishnet dataset has been labeled according to the object class, so that the data that will be retrieved is only data that matches the class that the model wants to detect. Fig. 12 shows an example of an image from the Fishnet dataset.

The data will then be integrated with the data from the video recording file. The data will be submitted to the preprocessing stage after it has been combined. The video data will be extracted for each frame to see the objects contained in the frame and labeled with polygon annotation, while the Fishnet dataset data will be relabeled with polygon annotation. In total, 4280 images from 6 classes were used as training and testing datasets for the model. The dataset was divided 7:3 for training and testing the model, with the data for testing the model divided again at a ratio of 1:1 [16]. Table 2 and Table 3 contain information about the image distribution and the amount of data for each class.

A picture containing cockpit, vehicle, plane, indoor

Description automatically generated

Fig. 12. An example of image from Fishnet dataset [17]

TABLE. 2 DISTRIBUTION OF IMAGES ON DATASET

|  |  |
| --- | --- |
| **Data Section** | **Number of Images** |
| *Training* | 2.999 |
| *Validation* | 641 |
| *Testing* | 640 |
| **Total** | 4.280 |

TABLE. 3 THE AMOUNT OF DATA FOR EACH CLASS ON DATASET

|  |  |  |  |
| --- | --- | --- | --- |
| **Class** | **Total Data** | | |
| **Training** | **Validation** | **Testing** |
| *human* | 7423 | 1566 | 1553 |
| *tuna* | 4734 | 1071 | 967 |
| *Skipjack tuna* | 262 | 58 | 42 |
| Tongkol | 447 | 98 | 94 |
| *Squid* | 564 | 93 | 121 |
| *unknown* | 2631 | 560 | 586 |
| **Total** | 16061 | 3446 | 3363 |

## Data Pre-Processing

The obtained data needs to be modified first so that it can be utilized as input data in the training and testing processes during the preprocessing stage. Several operations are carried out during the preprocessing stage. If the data is in the form of a video file, the file will be examined to see if it is corrupted and if there are any objects to be discovered in the file. If it passes this stage, the video file will be extracted frame by frame to discover frames that include the object to be recognized and discard those that do not. The frames will then be combined with data from the Fishnet dataset and tagged using polygon annotation. This annotation type is utilized so that the dataset may be trained using the semantic segmentation approach as one of the model testing scenarios. After labeling, the data will be divided into three sections: training, validation, and testing. Figure 13 shows the preprocessing process visually.

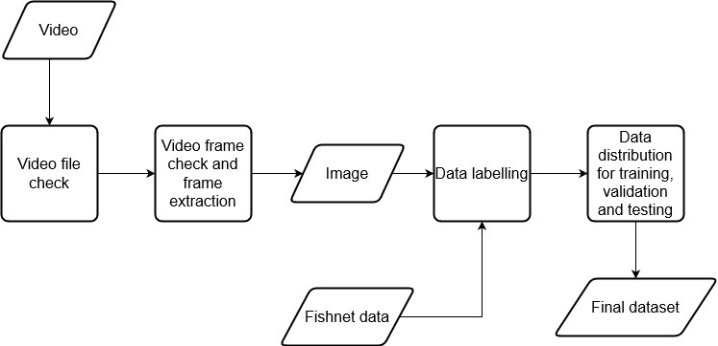


Fig. 13 Pre-processing of data

The initial stage of video file examination is performed manually, however the video frame extraction procedure is automated. After all of the frames have been collected, the data will be converted into an image that must be tagged before it can be trained to construct a machine learning model. Label Studio is used in the data labeling process. Label Studio is one of the open-source programs for annotating data. Label Studio is capable of annotating data with either bounding box-shaped labels or masks. Label Studio may be installed directly using localhost. To install Label Studio, the Anaconda application is used to create and activate a virtual environment on the computer. Fig. 14 shows the commands that can be used to create and activate a virtual environment for Label Studio.

conda create –-name/-n <environment-name>

conda activate <environment-name>

pip -u install label-studio

label-studio

Fig. 14 Anaconda command for launching the Label Studio

A project is required to store the data to be annotated in order to annotate it. Data annotation can be performed once the project has been created. Because the data will be annotated so that it can be trained using semantic segmentation, semantic segmentation with polygons will be used to annotate the data in the form of a polygon. Polygon annotation is a technique for annotating data that involves locating the edge points of the object to be annotated. It tries to create labels that are the same shape as the object. Fig. 15 - 17 show the process of annotating data with Label Studio using polygon annotation.

After all of the photos have been labeled, the images in the project will be exported to obtain the label results from the annotations placed on the image in the required dataset format. YOLO itself has a dataset format in the form of a file with the extension '.txt', and because it uses the polygon annotation method, the label format will be <object-class> <x0> <y0> <x1> <y1> .... <xn> <yn> which represents the coordinates of the edge points on the label, where n will adjust to the number of edge points given on the object. Because the value in the label has been normalized, it is in the range 0 to 1. The data will be divided into three sections, which include training, which will be used to train the model, validation, which will be used to obtain model performance using training data, and testing, which will be used to obtain model performance on data that has never been seen in the model.



Fig. 15 An unlabeled image example



Fig. 16 Image annotation with polygons process



Fig. 17 The results of image annotation with polygons

## Design of the Training Process

After the data has been gathered and preprocessed to meet the requirements, it will be utilized in the model training process. The data will be utilized to build a model that can detect the object to be detected. In general, model training is performed by splitting the data into numerous batches, which then extract the features and calculate the loss value of the model, which indicates the difference between the prediction results and the ground truth. The model will next be verified using the validation section of the dataset to measure detection performance such as detection accuracy and detection speed. Fig. 18 shows the architecture of the fish variant detection system model training process.

The data in the training section of the dataset is used in the model training process of the fish variant detection system. Before training the model, there are numerous parameters known as hyperparameters that may be set to optimize the model. Learning rate, momentum, and augmentation parameters such as translate, degrees, shear, and others are examples of common hyperparameters. After adjusting, the picture will be scaled based on the image size parameters used for model training. After resizing, the data will be separated into batches that will be adjusted to the variant of the model to be trained in order to adjust the training computer's capabilities. This is due to the fact that bigger batch sizes might speed up the training process but need more GPU memory to process the batches simultaneously, and vice versa. As long as it hasn't reached the final batch of training data, the process will continue until it does.

The split batch data will then be trained using a variation of the YOLOv7 model. The model's loss value will then be calculated on that batch of data to compare the detection results to the ground truth. After the model has been trained, it will be validated before being tested with data from the validation section dataset. The data will be utilized to compute the model's detection performance, such as precision, recall, and mAP values. The validation data is also separated into batches based on the batch size used to train the model. The validation process will be repeated until the last set of validation data is received. After calculating the model performance value, the model's weights will be updated and compared. If the weights are the best, the model will be saved. Models trained at multiples of 100 iterations will also be stored. The model training process will terminate when one of two conditions is met: the model training has reached the last epochs or the model training has met the early stopping criteria, which can be triggered when the model weights have not developed by the specified amount of patience.

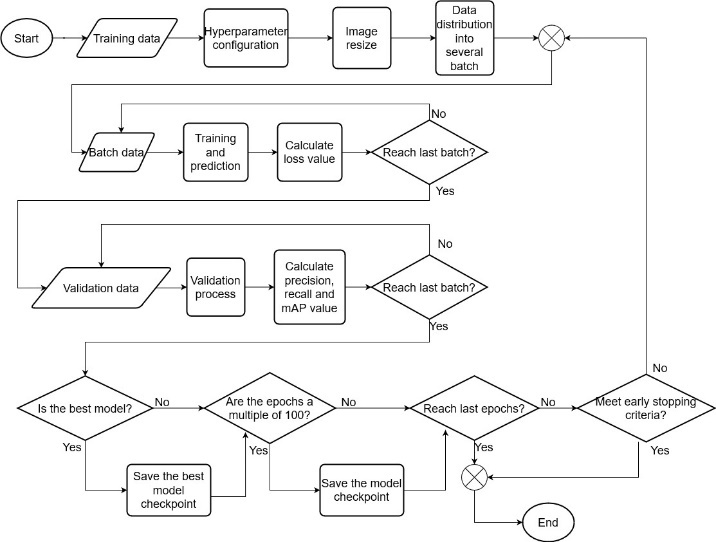


Fig. 18 Model training schema with YOLOv7

The model will be trained using two different label forms, polygon annotation and bounding box forms. This is done to compare detection performance when each label shape is used to train. Because the data has been annotated with polygons, the data label must be transformed into a bounding box label. The bounding box data is composed of four components such as the object's center point on the x and y axes, the object's height, and the object's width. As a result, the label of each picture will be examined to determine the object's maximum and lowest values on the x and y axes. Fig. 19 shows an example of code that may be used to transform polygon annotation data.

new\_labels = []  
for label in labels:  
  label = label.strip().split()  
  object\_class = label[0]  
  vertices = label[1:]  
  
  x\_values = [float(vertices[i]) for i in range(0, len(vertices), 2)]  
  y\_values = [float(vertices[i+1]) for i in range(0, len(vertices), 2)]  
     
  # Calculate the minimum bounding box that encompasses the polygon vertices  
  xmin = min(x\_values)  
  xmax = max(x\_values)  
  ymin = min(y\_values)  
  ymax = max(y\_values)  
  
  x, y, w, h = (xmin + xmax) / 2.0, (ymin + ymax) / 2.0, xmax - xmin, ymax - ymin  
     
  new\_labels.append(f"{object\_class} {x} {y} {w} {h}")  
 save\_dir = os.path.join(data\_dir, "result")  
 with open(os.path.join(save\_dir, img\_file\_id + ".txt"), 'w') as f:  
 f.write('\n'.join(new\_labels))

Fig. 19 Code for converting a Polygon Annotation dataset to a Bounding Box dataset

## Design of the Testing Process

After training, the model will be moved to the model testing process by predicting data on the testing dataset that was not included in the training and validation sections. The model testing process consists of five steps. Data form testing is the initial stage. The form of the data corresponds to the shape of the label assigned to the detected object. This can have an impact on the model's performance since more exact data could lead to better results. The best-performing data form will be selected in the next test, which will evaluate the model training method. This test seeks to determine whether complex methods generate better results than more widely used ones. In the next test, the model training image size test, the best training method will be applied. The purpose of this test is to see whether the image's size used affects the model's performance. The best-performing image size and YOLOv7 model variant will be used in the following test, which will evaluate the number of epochs and patience. The goal of the test is to examine if increasing the number of epochs and using patience as a control variable for early stopping may enhance the model's overall performance. The best model will be tested on the Jetson Nano as an example of a gadget that can be utilized in the real world.

mAP, amount of FPS, and F1-score are some of the key parameters used to compare YOLOv7 model variants. The mAP number reflects how the model recognizes an image or video file, with a larger mAP value indicating better detection accuracy. The number of frames per second then specifies how many frames the model can handle in one second. Despite having a decent mAP value, the model's high FPS makes it suitable for usage in a real-time system. The F1-Score score then describes the coherence of a class's recall and precision values. A model with a high F1-score value has a small probability of making a detection error since the precision and recall values are balanced.

## Design of the Model Implementation on Jetson Nano

After finding the best model based on the test results, the model will be deployed on the Jetson Nano to get an understanding of the model's performance when applied to edge devices, which will be used in real life. The reason for using Jetson Nano as a testing device is that Jetson Nano is an Nvidia product, and Nvidia provides CUDA and CUDNN, which are platforms used to enable GPUs as the main tools for executing and constructing machine learning models. The Jetson Nano is pre-installed with CUDA and CUDNN, which means no specific configuration is required to install the platform.

The YOLOv7 model can be executed on the Jetson Nano by installing the key libraries required to run the YOLOv7 model, such as PyTorch, Torchvision, Matplotlib, Numpy, Pillow, and others. The Jetson Nano has a limited amount of RAM that is insufficient to run the YOLOv7 model, therefore more RAM is required. Because the Jetson Nano runs the Ubuntu operating system, this is possible via Swap memory.

Swap memory is a feature of the Linux operating system that is utilized when the device requires resources to execute programs but the device's RAM is full. Linux will utilize unused storage memory space as extra RAM to allow new processes to run [20]. Swap memory can be created in a variety of methods, including swap partitions and swap files. Swap memory will be created using a Swap file since it will not be used permanently. A Swap file must be created in various phases. Before establishing a Swap file, make sure the device still doesn't have a particular place for Swap memory. After checking that there is no swap memory installed or active on the device, we can see whether there is space on the storage memory that can be utilized as swap memory. We may build a Swap file after ensuring that there is space on the storage memory that can be utilized as Swap memory. The swap file must then be activated as a swap space before it can be utilized as swap memory. Fig. 20 shows the command that may be used to create swap memory.

# Check is there any configured Swap memory in device

sudo swapon --show

free -h

# Check if there is any space for Swap file

df -h

# Create a swap file

sudo fallocate -l size directory

ls -lh directory

# Activate swap file as swap memory

sudo chmod 600 directory

ls -lh directory

sudo mkswap directory

sudo swapon directory

Fig. 20 Command to create a swap file to be used as swap memory

The model will be tested for detection performance in terms of accuracy and detection speed after the swap memory is activated. The model will be installed onto the Jetson Nano in order to do inference on the video data. The model will infer multiple video files. The inferred video files are not included in the video files from which frames are retrieved to be utilized as datasets for creating models. The inference procedure is carried out on each frame of the video clip, resulting in a model detection speed value. Following the completion of the inference procedure, the video file used will be extracted from each frame, and certain frames will be tagged and used as test data. The labeling of the data is performed in order to calculate the accuracy of model detection when loaded on the Jetson Nano. The data in the movie will be labeled in the same form as the dataset used to create the model, specifically with polygon annotation.

# Simulation Results

## System Implementation

A machine learning model development is required to create a fish variant detection system. The YOLOv7 algorithm was used to create the machine learning model on a computer with the specifications stated in Table 1. The device used has CUDA and CUDNN loaded, which are Nvidia libraries that operate to activate the GPU in the development of machine learning models, allowing the model building process to run faster by utilizing the cores on the GPU and CPU at the same time.

The PyTorch framework will be used in the process of developing machine learning models. Before being placed on the system, three variants of the YOLOv7 model will be trained for comparison, which is YOLOv7-Tiny, YOLOv7, and YOLOv7-X. Several test scenarios will be used to determine which model will be placed on the system. The machine learning models will be tested on two devices which is a computer with the specifications stated in Table 1 and a Jetson Nano. Testing on the Jetson Nano is carried out to provide an overview of the model's performance when applied to the device that will be mounted on the ship to perform detection. Table 4 contains information about the Jetson Nano specifications.

TABLE. 4 JETSON NANO SPECIFICATIONS FOR MODEL TESTING

|  |  |
| --- | --- |
| ***Central Processor Unit*** | Quad-Core ARM® Cortex®-A57 MPCore processor (4 CPUs) |
| ***Graphics Processing Unit*** | NVIDIA Maxwell™ architecture |
| ***Memory*** | 4 GB 64-bit LPDDR4 |
| ***Operating System*** | Ubuntu 18.04 |
| ***Python Version*** | 3.6.9 |

## Testing Scenario

In this study, three variations of the YOLOv7 model will be evaluated with various test scenarios to determine which model has the greatest detection quality and performance. The test is performed using the same quantity of data as in Table 2 and 3, so that the three options may be compared with the same test scenario. The dataset attached in the 'testing' section will be used during testing.

In this study, testing is done at the model training and implementation stages on the Jetson Nano, a real-world device. The model training stage will be carried out on each variant of the YOLOv7 model that is compared to evaluate the quality and performance of each variant in detecting. Because of the variable architectural complexity, training on each variation will employ various batch sizes so that model training may be done without exceeding the capabilities of the device used. Several testing components are used during model training, including the form of the dataset, model training methods, and training parameters such as image size and epoch number. Fig. 21 shows a visual illustration of the testing scenario.

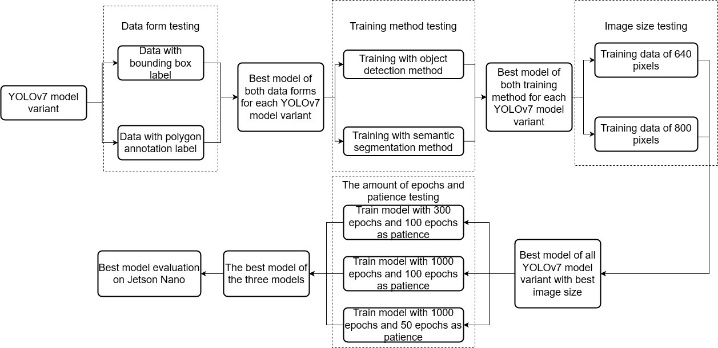


Fig. 21 Details for the test scenario

The quality of the model training results will be determined using mAP and F1-Score. These two criteria are commonly used in determining the quality of object detection models because their calculations can represent the overall accuracy of the model by including not only true positive values, but also false positive and false negative values as values that represent model errors in detection. The detection speed is then evaluated using inference time. Inference time is a unit used to calculate the amount of time it takes the object detection model to complete the detection. After model training, the best model will be tested on the Jetson Nano, which will be used when the model is applied in the real world. This compares whether the model's quality and performance in detecting decrease when the device used has a lower computational process than the device used for model training.

## Testing of Data Form Effect on Accuracy and Total FPS

In this test, every variant of the YOLOv7 model will enter the model training stage with different types of data. The purpose is to determine if the form of the data contained in the dataset has significant impacts on the model's accuracy value. The data form that will be compared is polygon annotation data with bounding box data.

Model training will use 640-pixel images from the training dataset with a learning rate of 1 x 10-2, iterated 300 times with SGD optimizers with momentum because SGD produces more generalized results than other types of optimizers combined with the presence of momentum, which accelerates the convergence process. For each model version, the batch size will be adjusted to meet the capabilities of the training device. The model will then be evaluated using the batch size 1 testing dataset. The same image size as in the model training will be used for testing, with an IoU Threshold of 0.6 and a confidence Threshold of 0.001. The IoU and confidence threshold parameters used are the YOLOv7 model testing configuration's default settings.

Fig. 22 mAP0.5 of data form testing on YOLOv7 model variant graph

Fig. 23 Inference time of data form testing on YOLOv7 model variant graph

Fig. 24 F1-Score of data form testing on YOLOv7 model variant graph

The results in Fig. 22 - 24 show that polygon annotation and bounding box data provide similar results. In terms of accuracy, there is no obvious difference between models trained using polygon annotation and those trained with bounding box data. The difference in mAP and F1-Score values for each variation of the trained model is similarly very small, less than 2%, indicating that these two models will produce similar detection outcomes. Fig. 22 shows that the greater the mAP value obtained, the more complex the architecture used by the model variant, and this also applies to the F1-score value. However, when the test results are compared, there is a difference in detection outcomes between models trained using polygon annotation data and models trained with bounding box data.

In Fig. 25 - 26, for example, it can be seen that the YOLOv7 model trained with bounding box data incorrectly identifies the 'tuna' class as the 'human' class, whereas the model trained with polygon annotation data correctly predicts the 'tuna' but has one false positive value in the 'human' class. Both models make mistakes in object detection, as shown in the figure, but the model trained with polygon annotation data can still identify all objects that should be detected in the image, whereas the model trained with bounding box data cannot.



Fig. 25 Testing image with correct labeled example

a) b)

Fig. 26 YOLOv7 model detection results with a) bounding box and b) polygon annotation data.

In terms of detection speed, the model trained using polygon annotation data outperforms the model trained with bounding box data. The inference time value of the model trained using polygon annotation data can rise by up to 1 ms, resulting in an increase in FPS of up to 16 FPS. As shown in Fig. 23, the slower the detection speed attained, the more complicated the architecture used by the model variant. The reason that may be causing this is related to how the annotation data is created, as illustrated in Fig. 19, where the polygon annotation data is done manually, but the bounding box data annotation is the result of converting the polygon annotation data using a program. The detection speed is reduced because the bounding box labels used are created from the conversion results, and the shape of the labels is not more precise than the polygon annotation.

## Testing of Comparison Between Object Detection and Semantic Segmentation Method on Accuracy and Total FPS

The next test compared the object detection and semantic segmentation methods used in model training. According on the results of previous tests, polygon annotation data offers various benefits over bounding box data. As a result, the purpose of this test is to examine the detection performance of models trained with two different methods using polygon annotation data. Semantic segmentation is an object detection method enhancement where the model detects the object accurately by determining the object's edge points so that the detection result is shaped like the object. As a result, semantic segmentation is widely used in models that require a high level of detection precision, such as in an autonomous car.

The testing process involves retraining the three variations of the YOLOv7 model in the training phase with each model building approach using the polygon annotation dataset. The model training will also use the same settings as the first test, with the exception of the batch size section, which adjusts to the model variations and creation method so as not to exceed the training device's capabilities. The created models will then be evaluated with the testing section's dataset to determine the detection performance of each model using the same testing conditions as the first test.

Based on the model's performance, the results in Fig. 27 - 29 show that object detection improved the value of mAP0.5 by 7 - 15%. Furthermore, the model using the object detection method has a 3 - 11% higher F1-Score value. As demonstrated in Fig. 27 and 29, the higher the mAP and F1-score values obtained by the model, the more complicated the architecture used by the model version. The key factor in the difference is how the datasets used by both models are processed. The datasets used in model training with object detection and semantic segmentation methods are polygon annotation data, but the object detection method will only display the bounding box of the detection results, whereas the semantic segmentation method will display the bounding box and the segmentation results performed by the model on the data, so the processing performed by the model trained with the semantic segmentation method will be more complex.

Fig. 27 mAP0.5 of training method testing on YOLOv7 model variant graph

Fig. 28 Inference time of training method testing on YOLOv7 model variant graph

Fig. 29 F1-Score of training method testing on YOLOv7 model variant graph

a) b)

Fig. 30 YOLOv7-X model detection results with a) object detection and b) semantic segmentation training method

The object detection model also has a quicker inference time value of 1.6 - 2.6 ms, increasing the number of FPS up to 166 FPS when compared to the semantic segmentation model. The detection speed will slow down with the more complicated architecture used by the model variant, as demonstrated in Fig. 28. The factor causing this is also related to the dataset's processing. Because semantic segmentation requires a high level of precision in detection, finding patterns that are similar to objects in the dataset and making sure that the pattern is the object to be detected takes longer than object detection, which only requires adjusting the bounding box of the object to be detected.

## Testing of Image Size Effect on Accuracy and Total FPS

The next test compares the image size used for model training. The purpose of this test is to see if the image size used in model training affects the model created. The model is tested by training it with multiple image sizes, 640 pixels and 800 pixels for each model version. Because the device used for model training would fail to construct a model due to a lack of resources needed to train a model using an image size bigger than 800 pixels, an image size of 800 pixels was chosen. Except for the batch size, which adjusts to the device's capabilities, the dataset and configurations are the same as in the earlier test. Then, testing will be performed using the same parameters as the previous test, with the exception of the image size, which will be adjusted to the size used during model training. The model that performs the best will be used in the next test.

Fig. 31 mAP0.5 of image size testing on YOLOv7 model variant graph

Fig. 32 Inference time of image size testing on YOLOv7 model variant graph

Fig. 33 F1-Score of image size testing on YOLOv7 model variant graph

The results in Fig. 31 - 33 show that each image size has its own advantages. In terms of performance, it can be shown that increasing the image size to 800 pixels in model training increases the model's mAP and F1-Score values, but not dramatically. This happened to all three variations of the model, which observed a 2 - 8% increase in mAP and F1-Score. Fig. 31 and 33 shows that models trained with an image size of 800 pixels had a higher accuracy than models trained with 640 pixels. A bigger image size, when processed, makes more items visible, and the objects are also easier to identify due to the higher number of pixels. This is related to the higher mAP value, which increases the accuracy and recall values of the model with a bigger picture size since the identified items are easier to identify. Furthermore, models with bigger image sizes have lower loss values because more features from the data are successfully extracted and used in the detection, reducing the probability of the model mis-detecting.

a) b)

Fig. 34 YOLOv7 model detection results with image sizes of a) 640 pixels and b) 800 pixels

In terms of detection speed, the model created with an image size of 640 pixels has a quicker inference time value of 0.8 - 5.2 ms, improving the number of FPS to 104.2 FPS when compared to 800 pixels. According to Figure 4.12, the slower the detection speed achieved, the more complicated the architecture used and the bigger the image size used. This is because the pixels processed in the 640 pixels image size are less than the pixels processed in the 800 pixels image size, hence the model's complexity in performing detection is reduced. As a result, lower image size can improve detection speed and FPS.

Because the required model is one with a high level of accuracy, the model's accuracy gets priority over detection speed. As a result, the YOLOv7 model variant with an image size of 800 pixels will be chosen as the model used in the next test because it has an accuracy of 84.3% and a total FPS of 69 FPS, indicating that the YOLOv7 model variant with an image size of 800 pixels performs well in terms of detection accuracy and detection speed. Furthermore, the variation has a file size of 72 MB, which is not excessive for the size of the model and can be kept on a small storage capacity. Furthermore, based on the test results, the variant made no detection mistakes.

## Testing of Increased Epochs to Trigger Early Stopping on Accuracy and Total FPS

The next test was carried out by increasing the number of epochs as the number of iterations in training the model. By increasing the number of epochs, the model may trigger early stopping, which is an approach used to determine the most ideal number of epochs to prevent model overfitting [13]. As a result, the purpose of this test is to see if triggering early stopping improves the accuracy of the model and prevents overfitting. The previous test's best model variation, YOLOv7, will be used in this test, with an image size of 800. The model is trained through 300 epochs with a patience of 100, which is the default patience value for YOLOv7. Later, the model will be trained with the number of epochs increased to 1000, with patience values of 50 and 100 used as parameters for the number of epochs required to stop training the model when no progress is seen. The batch size is also adjusted to match the batch size used by the model in the previous test, which was four.

Fig. 35 mAP0.5 of increased epochs testing on YOLOv7 model graph

Fig. 36 Inference Time of increased epochs testing on YOLOv7 model graph

Fig. 37 F1-Score of increased epochs testing on YOLOv7 model graph

According to the test results in Fig. 35 - 37, increasing the number of epochs has the same impact as increasing the image size. The model's accuracy does not improve significantly, with just a 3% improvement when compared to the model trained with 300 epochs. This might be because the model built can already identify the class to be detected, so increasing accuracy will just add nodes to the neural networks, increasing the likelihood of being recognized as a certain class. The rise in accuracy is followed by an increase in detections, which include both false positives and false negatives.

In terms of detection speed, increasing the number of epochs causes the model to detect at a slower rate of approximately 2%-7%. The reduction in the inference time value can be caused by the complexity of the neural networks built in the model with more nodes, which raises the likelihood of each class and hence increases the time required to categorize the observed objects. The model with 1000 epochs and 50 patience as the best model in terms of accuracy does not see a significant decrease in detection speed. As a result, this model is the best of the test and will be used in the next test.

## Jetson Nano Model Detection Testing

The final test was performed on the Jetson Nano by evaluating the best model based on the results of the previous tests. The purpose of this test is to compare detection results between devices used for model training and devices that may be used in the real world. The test will be carried out using a video that is not part of the test dataset and has a frame rate of 24 FPS and a resolution of 640 pixels. The movie does not use 800 pixels as the optimal image size since the Jetson Nano takes more than 0.3 seconds to detect at non-maximum suppression.

As a result, an image size of 640 pixels will be used to make sure that non-maximum suppression detection does not exceed the time limitation. The video has several things from the detection class. The model will be loaded onto the Jetson Nano and perform inference on the video to recognize objects using IoU threshold parameters of 0.45 and confidence threshold parameters of 0.25, which are part of the YOLOv7 default inference model configuration. The video will then be extracted for each frame, so that certain frames may be used as a new dataset to see if the accuracy of the model on the Jetson Nano and the model training device is the same. On the Jetson Nano, the model's accuracy is tested using the same configurations as in previous testing.

Fig. 38 Inference Time of model detection testing on YOLOv7 model with Jetson Nano graph

According to the test results in Fig. 38, the model's inference time value in detecting on the Jetson Nano has fallen dramatically when compared to the inference time on the training computer. When compared to the training computer, the detection speed on the Jetson Nano can be decreased by 687 - 787 ms. This can happen because of a variety of circumstances, one of which is the difference in computational operations that the Jetson Nano and the training computer can do. In terms of CPU, GPU, and RAM, the Jetson Nano has much lower specifications than the training computer. Because of the decrease in device specifications, the computing process that the Jetson Nano can do cannot equal the computing process of the training computer. Furthermore, the YOLOv7 model version has a more complicated architecture than smaller YOLOv7 variants such as YOLOv7-Tiny, causing a higher number of trainable parameters for the model. As a result of the requirements stated in Table 4.1, the Jetson Nano does not have specifications that can process all parameters quickly, which causes a decrease in the detection speed on the Jetson Nano.

However, when the inference result video is examined further, no video frames are lost despite the drop in detection speed. Fig. 39, for example, shows one of the frames from the hiv00167.mp4 video clip. Even though the detection speed has dropped, the inference model on the video still displays frames with the same detection object. This is possible because inference will be performed on all frames in the video, resulting in a final output with the same time and frame rate even though the detection speed has reduced.



Fig. 39 Hiv00167.mp4 video example of inference result frame on Jetson Nano and training device

The model loaded on the Jetson Nano and the training computer then did not differ in detection accuracy, with the same mAP and F1-Score values despite using devices with significantly different specifications, as shown in Table 5. This is possible because the model used has the same number of trainable parameters, allowing the model to have the same detection capabilities even when installed on devices with significantly different specifications. As a result, using the Jetson Nano device will just reduce detection speed while maintaining detection accuracy.

TABLE. 5 TEST RESULTS FOR MODEL ACCURACY ON THE JETSON NANO AND TRAINING DEVICE

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No | Perangkat | Ukuran Citra | mAP0.5 | mAP0.5:0.95 | *F1-Score* |
| 1 | Jetson Nano | 640 | 0,669 | 0,47 | 0,68 |
| 2 | Komputer Pelatihan | 0,669 | 0,47 | 0,68 |

# Conclusion

Based on the research that is done in this paper, many conclusions can be made regarding the development of a multi-object fish variant detection system using the YOLOv7 object detection algorithm. The development of a multi-object detection system for fish variant detection using the YOLOv7 model was completed successfully using a personally built dataset of 4280 photos reaching 22,870 labels and 6 classes. Then, the polygon annotation data form does not enhance detection accuracy significantly, but it can improve the detection speed of the YOLOv7 model by 0.1 to 1 ms when compared to the bounding box data form. For the training method, object detection can increase the detection accuracy by 45-55% and the detection speed by 1.6 – 2.6 ms compared to semantic segmentation. Then, using bigger image size can increase the detection accuracy by 2 – 8%, but reducing the detection speed by 0.8 – 5.2 ms. Then, Increasing the number of epochs used can improve accuracy by 3% while slowing detection by 0.3 - 1.2 ms. From the test, it concludes that YOLOv7 variant is the best YOLOv7 model variant with detection accuracy up to 86.1% and detection speed up to 14.5 ms. This result can be obtained with some adjustment when training the model by using polygon annotation data form, object detection training method, 800 pixels image size, and 1000 epochs with 50 epochs patience. This model also successfully implemented on Jetson Nano with decreasing detection speed by 687 – 787 ms but with the same detection accuracy.

There are also several suggestions could be given to enhance research results. The number of classes used in the research can be raised to identify more objects along with equalizing the quantity of data throughout classes to get more equal detection results. Furthermore, the variation used can be more suited to the capabilities of the edge devices used, such as yolov7-tiny, or by using better device specifications to operate the YOLOv7 model. The model may also be improved with full semantic segmentation to improve the precision of the discovered fish and raise the detection model's usage.

# References

[1] “Fish; fillets, frozen exports by country |2021.” https://wits.worldbank.org/trade/comtrade/en/country/ALL/year/2021/tradeflow/Exports/partner/WLD/product/030420 (accessed Nov. 20, 2022).

[2] “Tuna export company and exporters in Indonesia - Tridge.” https://www.tridge.com/intelligences/atlantic-bluefin-tuna/ID/export (accessed Nov. 21, 2022).

[3] “KKP Tangkap 167 Kapal Pelaku Illegal Fishing Selama 2021 Halaman all - Kompas.com.” https://money.kompas.com/read/2021/12/13/173905726/kkp-tangkap-167-kapal-pelaku-illegal-fishing-selama-2021?page=all (accessed Nov. 21, 2022).

[4] “IUU Fishing as an Evolving Threat to Southeast Asia’s Maritime Security | Asia Maritime Transparency Initiative.” https://amti.csis.org/iuu-fishing-as-an-evolving-threat-to-southeast-asias-maritime-security/ (accessed Nov. 21, 2022).

[5] “Indonesia sinks 8 Malaysian vessels for illegal fishing.” https://www.aa.com.tr/en/asia-pacific/indonesia-sinks-8-malaysian-vessels-for-illegal-fishing/2181821 (accessed Nov. 21, 2022).

[6] “What is Computer Vision? | IBM.” https://www.ibm.com/id-en/topics/computer-vision (accessed Dec. 05, 2022).

[7] Z. Zou, Z. Shi, Y. Guo, J. Ye, and S. Member, “Object Detection in 20 Years: A Survey,” May 2019, doi: 10.48550/arxiv.1905.05055.

[8] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, “YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors”.

[9] C. J. L. Chan, E. J. A. Reyes, N. B. Linsangan, and R. A. Juanatas, “Real-time Detection of Aquarium Fish Species Using YOLOv4-tiny on Raspberry Pi 4,” in *4th IEEE International Conference on Artificial Intelligence in Engineering and Technology, IICAIET 2022*, Institute of Electrical and Electronics Engineers Inc., 2022. doi: 10.1109/IICAIET55139.2022.9936790.

[10] N. Radha, R. Swathika, and P. S. Shreya, “Automatic Fish Detection in Underwater Videos using Machine Learning,” in *6th International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud), I-SMAC 2022 - Proceedings*, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 587–592. doi: 10.1109/I-SMAC55078.2022.9987363.

[11] V. Lakshmanan, M. Görner, and R. Gillard, *Practical Machine Learning for Computer Vision End-to-End Machine Learning for Images*.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*.

[13] M. Elgendy, O’Reilly for Higher Education (Firm), and an O. M. Company. Safari, *Deep Learning for Vision Systems*.

[14] “Getting Started with YOLO v4 - MATLAB & Simulink.” https://www.mathworks.com/help/vision/ug/getting-started-with-yolo-v4.html (accessed May 25, 2023).

[15] “Polygon Annotation | Humans in the Loop.” https://humansintheloop.org/services/polygon-annotation/ (accessed May 18, 2023).

[16] “Train Test Validation Split: How To & Best Practices [2023].” https://www.v7labs.com/blog/train-validation-test-set (accessed Jun. 10, 2023).

[17] “Description.” https://www.fishnet.ai/description (accessed Dec. 16, 2022).