Enhanced Animal Detection Using YOLOv8 and Data Augmentation Techniques: A Deep Learning Perspective

Rizqi Putri Nourma Budiarti[1*], Fitria Anggraini[2], Ubaidillah Zuhdi[3], Niken Savitri Primasari[4], Dion Satrio[5]

[1,4,5] Digital Business, Universitas Nahdlatul Ulama Surabaya, Surabaya, Indonesia.
[2]Information System, Universitas Nahdlatul Ulama Surabaya, Surabaya, Indonesia.
[3]Management, Universitas Nahdlatul Ulama Surabaya, Surabaya, Indonesia.

*Email: rizqi.putri.nb@unusa.ac.id

Abstract. In various aspects of human life, animals are widely exploited as a labor, consumption, pet, and for research. While animal use contributes favorably to various sectors of life, the number of animal populations allows the farm to expand to a larger scale, increasing the scale of the operation. This has made it increasingly difficult for the monitoring of the farm to do so when animals roam the complex areas, courtyards, open fields, or farmland in search of food. As for the animals that are hard to monitor are cats, goats, chickens, and cows. Thus, the development of animal detection technologies using deep learning to make it easier to monitor animals as they prowl for food. In the study, four types of animal animals, chickens, cats, goats, and cows use deep learning, yolov8 (You Only Look Once version 8) that is known for accuracy in identifying objects. The stages taken on this study include data collection, data annotations, data division, augmentation, data training, and results evaluation. Excellent training results from 125 epoch on the 85% data share scheme training, 15% of validation data, and 5% of data testing with datasets at 11242 pictures achieve an accuracy rate of 99.5%, precision by 79.4%, recall by 79.9%, and f1-score by 76.6%.

Keywords: Immediately Animals Detection, Deep Learning, YOLOv8, Data Augmentation, Computer Vision.

Introduction

In various aspects of human life, animals are widely exploited as Labour, consumption, domesticated, and for research [1]. As mankind's need for animal resources grows, the animal population in Indonesia is also growing. According to [2] Indonesia's population of farm animals in 2022 is estimated to have been 17.75 million cattle, while goats are estimated to have been 18.55 million, and chickens estimated at 3.8 billion. In addition, in Indonesia there are pets, one of which is a 129% increase in population growth from 2017 to 2021, placing an 37% higher ownership rate than only 16% [3].

While animal use makes a positive contribution to various sectors of life, animal populations can also pose some challenges. The more the age grows, enabling the farm to expand to a larger scale, thus increasing the scale of the operation. This caused increased manually monitoring of the farm and increased the risk of loss of farm animals [4]. Difficulty in controlling the animals when they wander complex areas, courtyards, or fields in search of food. As for the animals that are hard to monitor are cats, goats, chickens, and cows [5]. In addition, the animals have the potential to enter and destroy the plants on the farmland when uncured [6].

Detecting objects is an important field in computer vision that USES deep learning and machine learning models to enhance performance in the object's detection process [7]. The deep learning model for this problem consists primarily of two components, at the backbone components that resemble the picture's classification model, and the stake proposals that serve to predict boundary boxes [8]. Deep learning based object detector is divided over two groups, the two-stage detector and the one stage detector [9]. One application of object detection technology is that of animals that have a significant role in solving real-world problems [10].

Deep learning is progressing enormously in computer vision for various tasks, one of which is the detection of objects [11]. Deep learning is able to represent data of pictures, videos, or texts without relying on human rules. Deep learning has three or more artificial neural networks (ANN), thus being able to adapt to multiple data with good performance in resolving complex problems successfully through machine learning [12].

Some scientists have proposed several approaches in which animals should pose in front of cameras to recognize the animals, such as animal facial recognition. However, facial recognition cannot identify animals that do not observe cameras. Animals can appear in any size, position, and color [10]. As on research [4] do the cow detection with the drones using yolov5 and got that the model can detect at an altitude of five meters while silent by 75%. Then on research [13], using the mask r-cnn model to detect and identify animals using the dashboard camera. Results from detecting cows with precision averaged 79.47% and dogs with an average of 81,09% precision.

Thus, the study proposes deep learning using the popular one-stage detector, the model you only look once (YOLO). Yolo is a famous series of object detection for its accuracy and precision. The latest version of yolo, yolov8, is known for its ability to identify objects accurately from inlaid images [14]. Moreover, yolov8 also shows a significant increase in the speed of detecting objects [15]. The purpose of this study is to detect animals, focusing on four categories, such as cows, goats, cats, and chickens using yolov8 which then evaluated yolov8's performance in detecting animals. Thus, it is hoped that research will provide insight into the optimum model of animal detection, as well as knowledge of its performance.

Methods

At this methodology stage, the research flow used in working on the final project will be explained. This research methodology provides guidelines in the form of research flow carried out during the ongoing research.

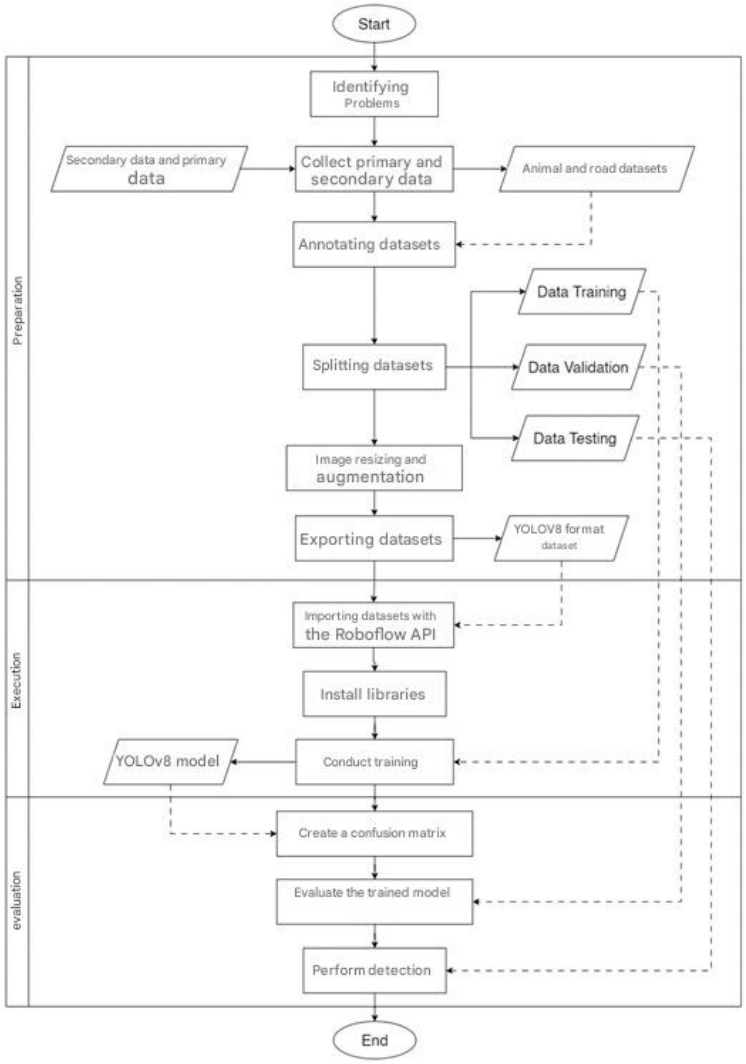


Figure 1. Research methodology

2.1. Preparation

2.1.1. Identifying the problem

Identifying a problem is by searching for and studying a library obtained from previous research, journals and articles on the Internet. The purpose of the study of this literature is to gain insight into the problems of research, the basis of theory, and the methods relevant to the research done.

2.1.2. Data colection

The data for this study included pictures of four categories of animals, including cows, goats, cats, and chickens. It comes from two sources, secondary and primary data assets. Then all of these data are stored in local and Google drives to minimize data loss and access anywhere.

In the study, it will use JPG image format, since it can produce good pictures, especially for photographs with lots of colors and gradients and commonly used in built-in digital camera images carrying android, the website, E-mail. Moreover, JPG format uses a lossy compression on which it omits some information from the original data, but it does not eliminate information significantly in the data overall [16]. Application of lossy images compression helps reduce the bandwidth needed to transmit images in the network and save storage space on devices. It is beneficial to increase network efficiency and use storage space [17].

2.1.3. Annotate the dataset

The dataset annotation process is carried out using Roboflow to identify or label the annotated objects. Dataset annotation is done by placing marking boxes in the image, which are generally known as bounding boxes.

2.1.4. Splitting the dataset

After the process of labeling and annotating data is complete, it divides the data into three parts, training data, validation data, and testing data. This arrangement is important to prevent overfitting and ensure accurate model evaluation. Overfitting occurs when the model is so focused on the training data that it is difficult to recognize patterns in the testing data differently [18]. In this study, six different data distribution schemes for training, validation data, and testing data in sequence are 60%:30%:10%, 65%:20%:15%, 70%:20%:10%, 75%:15%:10%, 80%:10%:10%, and 85%:10%:5%.

Generally, frequent schematics are 80% training, 10% validation, and 10% testing. The bigger the training portion, the better the model recognizes the pattern. This division ensures that there is sufficient data for validation and testing, so that model performance can be properly evaluated on the different data-sharing schemes [19].

2.1.5. Resize the image and augmentation

It is done using roboflow by adjusting data size to 640x640 pixels. It was adapted because of the general use of image resolution 640x640 pixels in the yolo, so the stride used was 80x80 for small objects, 60x60 for medium objects, and 20x20 for large objects [20].

In the augmentation process, it used rotation techniques to augment the training data with varying variations. It enables models to learn from new data sets, which can reduce overfitting of models and also increase accuracy. The rotations to be used include 90°, -90°, 45°, -45°, 15°, and -15° as explained in Table 1 below.

Table 1. Images rotation

Rotation	Description

90°	Rotate the image a quarter turn clockwise.
-90°	Rotate the image a quarter of a turn counterclockwise.
45°	Rotate image one quarter of the 90 brushes counterclockwise.
-45°	Rotate image a quarter of the 90 compounds counterclockwise.
15°	Rotate image one sixth of the 90 brushes counterclockwise.
-15°	Rotate the sixth of the 90 images opposite the clock.

2.1.6. Datasset export

When it has adjusted size and augment, models and source codes are drawn from roboflow by selecting yolov8 and downloading it in *.zip.

2.2. Execute

2.2.1. Importing datasets with the Roboflow API

Datasets that have been pre-processed in Roboflow will be downloaded via Kaggle using the API provided by Roboflow. Then install the Roboflow library to interact with the Roboflow platform.

2.2.2. Install the library

After the yolov8's datasset and model had been uploaded to kaggle, the next step was the install and importation of several such ultralytics to use the yolov8 model, importing the class from roboflow to access data from roboflow.

2.2.3. Go to training

In doing the yolov8 model training using training data. The data consists of previously annotated images with corresponding categories. In this process, the yolov8 architecture is presented in complex code form, but is optimized to process real-time images and detect objects with high levels of precision. Here is the architectural explanation of yolov8 based on the various layers and modules that make up the model.

	from	n	params	module	arguments
0	-1	1	1392	ultralytics.nn.modules.conv.Conv	[3, 48, 3, 2]
1	-1	1	41664	ultralytics.nn.modules.conv.Conv	[48, 96, 3, 2]
2	-1	2	111360	ultralytics.nn.modules.block.C2f	[96, 96, 2, True]
3	-1	1	166272	ultralytics.nn.modules.conv.Conv	[96, 192, 3, 2]
4	-1	4	813312	ultralytics.nn.modules.block.C2f	[192, 192, 4, True]
5	-1	1	664320	ultralytics.nn.modules.conv.Conv	[192, 384, 3, 2]
6	-1	4	3248640	ultralytics.nn.modules.block.C2f	[384, 384, 4, True]
7	-1	1	1991808	ultralytics.nn.modules.conv.Conv	[384, 576, 3, 2]
8	-1	2	3985920	ultralytics.nn.modules.block.C2f	[576, 576, 2, True]
9	-1	1	831168	ultralytics.nn.modules.block.SPPF	[576, 576, 5]
10	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
11	[-1, 6]	1	0	ultralytics.nn.modules.conv.Concat	[1]
12	-1	2	1993728	ultralytics.nn.modules.block.C2f	[960, 384, 2]
13	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
14	[-1, 4]	1	0	ultralytics.nn.modules.conv.Concat	[1]
15	-1	2	517632	ultralytics.nn.modules.block.C2f	[576, 192, 2]
16	-1	1	332160	ultralytics.nn.modules.conv.Conv	[192, 192, 3, 2]
17	[-1, 12]	1	0	ultralytics.nn.modules.conv.Concat	[1]
18	-1	2	1846272	ultralytics.nn.modules.block.C2f	[576, 384, 2]
19	-1	1	1327872	ultralytics.nn.modules.conv.Conv	[384, 384, 3, 2]
20	[-1, 9]	1	0	ultralytics.nn.modules.conv.Concat	[1]
21	-1	2	4207104	ultralytics.nn.modules.block.C2f	[960, 576, 2]
22	[15, 18, 21]	1	3778012	ultralytics.nn.modules.head.Detect	[4, [192, 384, 576]]

Model summary: 295 layers, 25858636 parameters, 25858620 gradients, 79.1 GFLOPs

Figure 2. YOLOv8 architecture

a) Layer 0 – 1: Initial Convolutional Layers

The YOLOv8 model starts with two basic convolutional layers that extract basic features from an image and reduce its size to facilitate processing in the subsequent layers. The first layer (Layer 0) is a convolutional layer that transforms the input image with 3 color channels (RGB) into features with 48 channels. This layer uses a 3x3 kernel and a stride of 2, meaning the input image is reduced by half in each dimension, resulting in a smaller but deeper feature map. In the next layer (Layer 1), the convolution process is repeated with 48 input channels and convolved to 96 output channels using the same kernel and stride.

b) Layer 2, 4, 6, 8, 12, 15, 18, 21: C2f Blocks

The YOLOv8 architecture is built on C2f blocks, which serve as residual blocks. Each C2f block aims to deepen and refine feature representations using fewer parameters and faster computation. For example, the C2f block in Layer 2 takes input from 96 channels and outputs 96 channels processed through two internal convolutional layers. This block is also equipped with a skip connection mechanism that combines the initial input and final output to reinforce important functional signals and minimize the loss of crucial information. The number of internal layers within each C2f block varies between 2 and 4 layers, allowing for more complex features to be extracted at different representation levels.

c) Layer 3, 5, 7, 16, 19: Deeper Layers

These layers (Layer 3, Layer 5, Layer 7, Layer 16, Layer 19) use a 3x3 kernel and convolution with a stride of 2 to further reduce the spatial dimensions of the feature map. Each of these layers increases the number of output channels, deepening the feature representation, and enabling object detection at different scales and resolutions. For instance, Layer 3 takes 96 channels and outputs 192 channels with deeper features. The main function of these layers is to filter and adjust feature information so that the model can observe more details at different image resolution levels.

d) Layer 9: SPPF Block

The SPPF (Spatial Pyramid Pooling Fast) block, included in Layer 9, plays a crucial role in aggregating information from the feature map using different window sizes without changing the feature map's resolution. In this context, the SPPF block combines spatial information from different window sizes (in this case, 5x5) to create an information-rich feature map that enhances the model's performance in detecting objects at various scales. This block allows the model to efficiently gather and integrate information from multiple scales.

e) Layer 10, 13: Upsampling Layers

The upsampling layers (Layer 10 and Layer 13) serve to enlarge the feature map so that information from lower resolution feature maps can be combined with higher resolution feature maps. These layers use proximity upsampling techniques, allowing the model to double the size of the feature map while preserving the details and important information previously reduced through the convolution and downsampling process.

f) Layer 11, 14, 17, 20: Concatenation Layers

The concatenation layers (Layer 11, Layer 14, Layer 17, Layer 20) are responsible for merging feature maps from earlier levels with feature maps from deeper levels. This concatenation process preserves information from various resolution levels and ensures that the model has access to all the necessary details to detect objects at different scales. For example, Layer 11 combines features from the previous layer with features from the sixth layer, allowing the model to integrate information with varying depth and complexity.

g) Layer 22: Detection Layer

The final layer (Layer 22) is the detection layer, which uses features at different resolutions to detect objects of varying sizes. This layer utilizes information from three different resolutions (192, 384, 576) to perform the final detection of objects in the image. It leverages information from multiple scales to enable the model to effectively detect both small and large objects, thereby allowing for highly accurate object detection.

2.3. Evaluation

2.3.1. Creating a Confusion Matrix

A confusion matrix is used to assess the performance of a model. This matrix compares the actual target values with the predictions provided by the machine learning model, offering a tabular summary of the number of correct and incorrect predictions. Its application involves calculating performance metrics such as accuracy, precision, recall, and F1-score to evaluate the performance of the classification model [21].

Table 2. Confusion matrix

		Predicted	
		Negative (N)	Positive (P)
Actual	Negative	True Negative (TN)	False Positive (FP) Type I Error

Positive	False Negative (FN) Type II Error	True Positive (TP)

True Positive (TP): When the actual value is positive and the prediction is also positive.

True Negative (TN): When the actual value is negative and the prediction is also negative.

False Positive (FP): When the actual value is negative, but the prediction is positive. Also known as Type I Error.

False Negative (FN): When the actual value is positive but the prediction is negative. Also known as Type II Error.

2.3.2. Evaluating the trained model

At this stage, the performance of the trained model will be evaluated using the validation data during the model training phase. This data differs from the training data, allowing us to determine whether the model has learned well when given new data.

2.3.3. Performing detection

Next, predictions are run on new images to assess how well the model performs in detecting objects on data it has not been trained and validated on before.

Result and Discussion

The process, mechanism, and results of object detection for four types of animals, including Cows, Goats, Cats, and Chickens, will be explained in detail at this stage. The steps conducted during the research include dataset collection, data preprocessing, data training, model evaluation, and testing using deep learning with YOLOv8 (You Only Look Once version 8). Below is an explanation of the research results related to the detection of four types of animals using YOLOv8.

3.1. Implementation results of modeling using YOLOv8

3.1.1. Dataset

The data used in this study consists of both primary and secondary data, with a total of 2000 images, divided into 500 images per category, including Chickens, Cats, Goats, and Cows. The images of cows and goats are secondary data obtained from the Roboflow platform and Google search, while the images of chickens and cats are primary data collected by taking photos directly in the Surabaya area and its surroundings using a mobile phone camera. The details of the primary and secondary data can be seen in Table 3.

Table 3: Images rotation

Source	Image	Quantity
Roboflow	Goats	286
	Cows	338
Google Search	Goats	214
	Cows	162
Mobile Phone Camera	Chickens	500
	Cats	500
Total		2000

Based on the dataset above, the data can be grouped into four class categories, Chickens, Cats, Goats, and Cows with 500 images for each class.

3.1.2. Preprocessing data

- Data annotations

The data that has previously collected, is identified by labeling, consisting of boundary boxes on each animal object in the picture in accordance with the category. In this study, visual annotations were done by hand through platfrom roboflow by uploading all the images that had been collected into roboflow by creating a new project. There were 2724 divided images of 4 classes, 686 of them for the picture labeled 'Sapi,' 729 for the picture labeled 'Kambing,' 614 pictures labeled 'Kucing,' and 695 pictured as 'Ayam.' Image distribution details can be seen on figure 3.

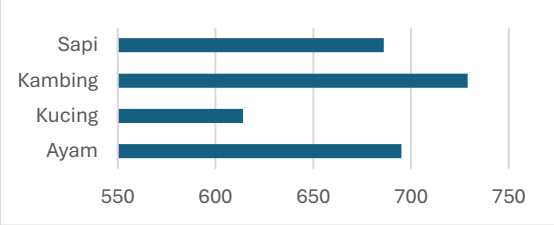


Figure 3. Image distribution

- Resize the images and augmentation data

The data that has previously collected, is identified by labeling, consisting of boundary boxes on each Altering the size of the images is done to ensure that the entire picture in the datasset has a uniform dimension, which is 640x640. This makes it easier for models to process images because they reduce the burden of computing during model training. In this study, an automatic alteration of images was made using roboflow.

Further, augmentation of the images is done by rotating on the images with several angles covering 90 levels, -90 angles, 45 cycles, -45 clams, 15 fiftions, and 15 fiftions. The purpose of image aungmentation is to enrich the variety of pictures and the number of datastices to be trained.

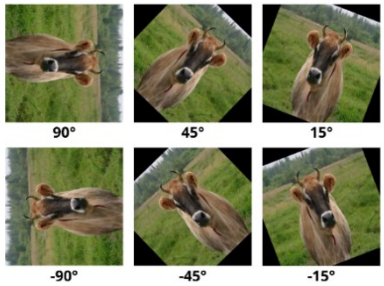


Figure 4. Image rotation

- Splitting the dataset

The splitting of data on different subsets is an important step to ensure that object detection models can be trained, validated and tested in structured and representative ways. The distribution of data is carried out in three different schemes, including training data, validation data, and testing data. Training data is used to train models by identifying patterns, characteristics, and objects that are relevant to be detected. During training, models optimize their weight and energy through several epoch by repeatedly looking at data to enhance model performance. Validation data is used to evaluate model performance during training to avoid overfitting. After each epoch, the model was tested on validation data to give performance descriptions in new data. This outcome is used to perfect hyperparameters and can lead to stopping early training or early stopping if necessary. Testing data is used to evaluate a model's final performance after training is over. This data is used only once and does not participate in the process of training or validation, giving an accurate evaluation of the models already trained.

However, there were extensions to the training data because of a augmentation of the data. Details of the data sharing scheme with the original 2000 images, as well as the addition of new data after augmentation, can be seen on table 4.

Table 4: Data splitting details

Splitting Data	Description	Training Data	Validation Data	Testing Data	Total
60%:30%:10%	Data Asli	1200	600	200	2000
	Hasil Augmentasi	8342	600	200	9142
65%:20%:15%	Data Asli	1300	400	300	2000
	Hasil Augmentasi	9039	400	300	9739
70%:20%:10%	Data Asli	1400	400	200	2000
	Hasil Augmentasi	9733	400	200	10333
	Data Asli	1500	300	200	2000

75%:15%:10%	Hasil Augmentasi	10412	300	200	10912
80%:10%:10%	Data Asli	1600	200	200	2000
	Hasil Augmentasi	11116	200	200	11516
85%:10%:5%	Data Asli	1700	200	100	2000
	Hasil Augmentasi	11811	200	100	12111

3.1.3. Learning and training

Parameters used to evaluate the yolov8 model during training using epoch comparisons and data sharing schemes. Epoch is part of the learning that is carried out by deep learning, where the small amount of epoch influences the duration and quality of training that is done. The process used by epoch is similar to the iteration process, but the epoch is more specific because it involves a full repetition of the data that allows the model to continue studying the patterns in the datasset data. Because of limited memory, the whole datassets cannot be processed into one epoch ata time, so the datassets are divided into several batches. Batch size is the number of samples processed in one mini-batch [22]. Batch size used during training on this research is 64. Using a larger batch size can reduce training time. In addition, GPU use (Graphics Processing Units) can increase the training efficiency.

In this study, the model's performance was evaluated using the Precision, Recall, and mAP (Mean Average Precision) metrics. The Precision metric measures the accuracy of the model in avoiding false positive detections. Meanwhile, the Recall metric assesses the model's effectiveness in detecting objects, focusing on minimizing the omission of any objects (False Negative). The mAP metric is used to understand the model's ability to recognize and locate objects within images by calculating the average precision (AP) for all classes in the dataset and computing it at different Intersection over Union (IoU) thresholds. IoU is used to evaluate the alignment between the predicted bounding box and the actual bounding box (ground truth) in an image [23].

YOLOv8 by default uses mAP50 and mAP50-95. For mAP50, an IoU threshold of 0.5 is used to evaluate whether an object is detected rather than how accurately its location is predicted. On the other hand, mAP50-95 uses IoU thresholds ranging from 0.50 to 0.95, with increments of 0.05, resulting in ten values: 0.50, 0.55, 0.60, 0.65, 0.70, 0.75, 0.80, 0.85, 0.90, and 0.95. mAP50-95 is used to evaluate the model's ability to detect and localize objects with varying levels of precision. Therefore, mAP50-95 may show lower values if the predicted bounding boxes do not align well with the actual bounding boxes.

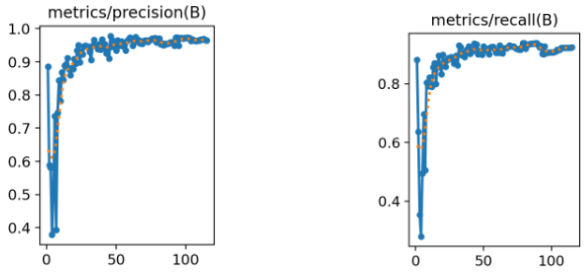
Here are the details of the training results using six different data splitting schemes, which include 60%:30%:10%, 65%:20%:15%, 70%:20%:10%, 75%:15%:10%, 80%:10%:10%, and 85%:10%:5%, with each scheme being trained for 125 epochs.

Table 5: Detailed Training Results with Six Data Splitting Schemes over 125 Epochs

Splitting Data	Epoch	Metrics			
		Precision	*Recall*	mAP50	mAP50-95
60%:30%:10%	125	0.916	0.871	0.932	0.817
65%:20%:15%	125	0.91	0.83	0.91	0.766
70%:20%:10%	125	0.957	0.896	0.958	0.856
75%:15%:10%	125	0.948	0.862	0.929	0.805
80%:10%:10%	125	0.963	0.915	0.966	0.866
85%:10%:5%	125	0.964	0.922	0.966	0.859

Based on the data in the table, experiments were conducted by splitting the training, validation, and test data with various proportions, all with 125 epochs. The data splitting schemes of 80%:10%:10% and 85%:10%:5% showed the best performance with high metrics in Precision, Recall, mAP50, and mAP50-95. In the 80%:10%:10% data split scheme, Precision reached 0.963, and Recall 0.915, with mAP50 at 0.966 and mAP50-95 at 0.866. Meanwhile, in the 85%:10%:5% data split scheme, Precision was slightly higher at 0.964, and Recall was 0.922, with mAP50 remaining at 0.966, but mAP50-95 was slightly lower at 0.859. Overall, the model's performance tended to improve as the proportion of training data increased, with the 85%:10%:5% data split scheme yielding the best overall metrics.

In the 85%:10%:5% data splitting scheme, Figure 5 displays the progress of the Precision and Recall metrics for the YOLOv8 model during the training process. In the Precision graph (a), it can be seen that the Precision value increases rapidly at the beginning of the training and then stabilizes around 0.95. This indicates that the model becomes more accurate in detecting true positive objects as the iterations increase. The Recall graph (b) shows a similar pattern, with a rapid increase at the start of the training and stabilization around 0.85. This indicates that the model becomes more effective at detecting all existing objects without missing many important ones. Overall, these graphs demonstrate that the model achieves good and stable performance, with high Precision and Recall after several iterations, indicating that the model is capable of object detection with good accuracy and sensitivity.



(a) (b)

Figure 5. Precision graph (a) and Recall graph (b) for the 85%:10%:5% data splitting scheme

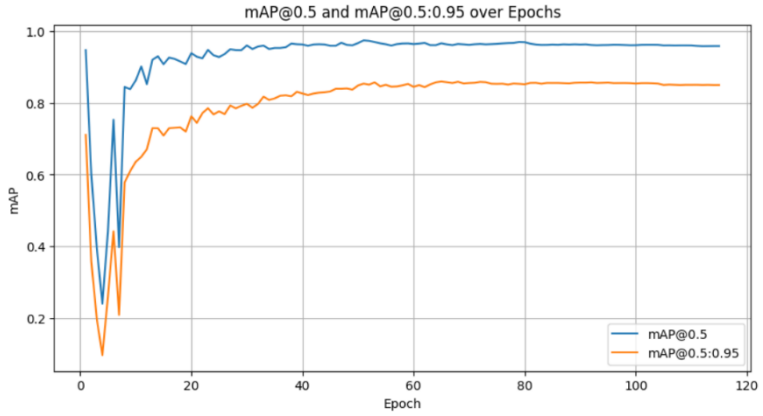
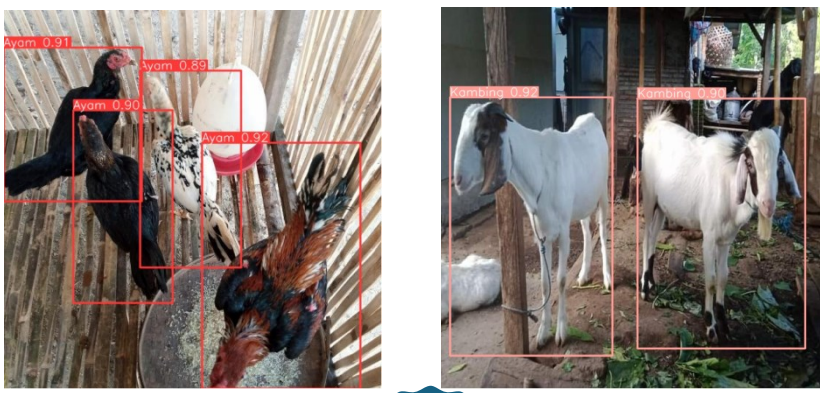


Figure 6. mAP Graph for 85%:10%:5% Data Splitting Scheme

The graph in Figure 6 illustrates the object detection model's performance in terms of mAP50 and mAP50-95 metrics across several epochs, with data split into 85% training, 10% validation, and 5% testing. The x-axis represents the number of training epochs, while the y-axis shows the mAP values, indicating the YOLOv8 model's effectiveness in detecting objects. The blue line represents mAP50 with an IoU threshold of 0.5, meaning predictions are considered correct if there's at least 50% overlap. The graph shows a sharp increase in mAP50 at the start, stabilizing around 0.9 after about 20 epochs, indicating rapid learning for high-precision object detection at this threshold. The orange line represents mAP50-95, a stricter metric that assesses performance across various overlap thresholds. Although mAP50-95 also improves during training, it doesn't reach the same level as mAP50, stabilizing around 0.8 after 20 epochs. This suggests the model performs well but slightly less so under stricter evaluation criteria.

Overall, the graph shows that the model achieves high and stable performance after approximately 20 epochs. The training process seems to have stopped early, likely due to early stopping, which prevents overfitting and saves computational time [24].

Next, a random image sample from the testing data will be displayed. This testing data contains new images or datasets that have never been seen by the model during the training or validation stages. Figure 7 provides 4 random samples with the detected object names and their accuracy results.



(a) (b)

(c) (d)

Figure 7. Prediction Results for Images of Chicken (a), Goat (b), Cat (c), and Cow (d) in the Division of 85%:15%:5%

3.2. Model evaluation for object detection

The model evaluation for object detection using YOLOv8 in this research was conducted with a confusion matrix. This evaluation provides an overview of the model's performance in recognizing and detecting the four types of animals that were previously trained. At this stage, the evaluation was done automatically without manual testing by utilizing a Python library to generate the confusion matrix table, which was then used to manually calculate the accuracy, precision, recall, and F1-score to achieve optimal results. The model used for evaluating the results is the one generated from the previous training process. This model was selected based on the highest results in each data split, specifically in the 85%:10%:5% data split scheme.

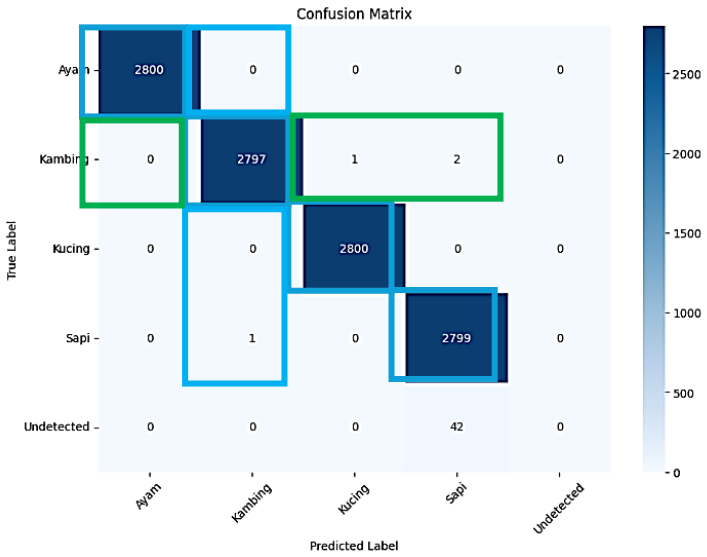


Figure 8: *Confusion matrix 85%:10%:5%*

Figure 8 shows the results of the confusion matrix used to evaluate the performance of the YOLOv8 model for detecting the four animal classes Ayam, kambing, Kucing, and Sapi in the 85%:10%:5% data

split scheme. The confusion matrix has two axes: the y-axis representing the True Label from the test data, and the x-axis representing the Predicted Label generated by the model. It is noted that there is a mapping of confusion matrix values based on different colored lines yellow, blue, and green with distinct meanings. The TP (True Positive) value is defined by the yellow color, the FP (False Positive) value is defined by the blue color, and FN (False Negative) is defined by the green color.

Calculating Accuracy measures how accurately the model can predict, expressed as the ratio of correct predictions to the total predictions.

$$\text{Accuracy} = \frac{\text{TP}}{\sum Data} \times 100\%$$

$$= \frac{11196}{11242} \times 100\%$$

$$= 99.5\%$$

(1)

Calculating Precision is an indicator of the correctness in positive predictions, measuring how many of the positive predictions are actually correct out of the total positive predictions.

$$Precision = \frac{TP}{TP+FP} \times 100\%$$

Table 6: Detailed Training Results with Six Data Splitting Schemes over 125 Epochs

	True Positive	***False Positive***	**Presisi**
Ayam	2800	0	$\frac{2800}{2800} = 1.00$
Kambing	2797	1	$\frac{2797}{2798} = 0.99$
Kucing	2800	1	$\frac{2800}{2800} = 1.00$
Sapi	2799	44	$\frac{2799}{2843} = 0.984$
Undetected	0	0	-
	Total		3.974

$$All\ Precision = \frac{\sum Precision}{Jumlah\ Kelas} \times 100\%$$

$$All\ Precision = \frac{3.977}{5} \times 100\% = 79.5\% \quad (2)$$

Calculating Recall (Sensitivity) is a measure of how well the model can correctly predict actual positive observations, indicating how effectively the model captures true positives.

$$Recall = \frac{TP}{TP+FN} \times 100\%$$

Table 7: Detailed Training Results with Six Data Splitting Schemes over 125 Epochs

	True Positive	*False Positive*	**Presisi**
Ayam	2800	0	$\frac{2800}{2800} = 1.00$
Kambing	2797	3	$\frac{2797}{2800} = 0.998$
Kucing	2800	0	$\frac{2800}{2800} = 1.00$
Sapi	2799	1	$\frac{2799}{2800} = 0.999$
Undetected	0	42	$\frac{0}{42} = 0$
	Total		3.997

$$All\ Recall = \frac{\sum Recall}{Jumlah\ Kelas} \times 100\%$$

$$All\ Recall = \frac{3997}{5} \times 100\% = 79.9\% \quad (3)$$

F1-score is a value ranging from 0 to 1, representing the harmonic mean between precision and recall.

$$F1 - score = 2 \times \frac{(Recall \times Precision)}{(Recall + Precision)} \times 100\%$$

$$F1 - score = 2 \times \frac{(0.799 \times 0.794)}{(0.799 + 0.794)} \times 100\% = 79.6\% \quad (4)$$

After conducting testing with a total of 11,242 images and calculating the values for the Accuracy, Precision, Recall, and F1-Score indicators using the 85%:10%:5% data split scheme and 125 Epochs, it was found that the obtained accuracy is quite high, at 0.995 or 99.5%. Additionally, Precision was 0.799 or 79.9%, Recall reached 0.794 or 79.4%, and the F1-Score was 0.796 or 79.6%. Overall, the model was able to effectively detect the four types of animals it was trained on, showing a balance between Precision and Recall, although the Precision, Recall, and F1-Score values were still relatively low. This occurred because, despite many correct predictions, there were still some errors, particularly in the Undetected class. This class refers to objects that the model was unable to recognize as any of the existing classes (Ayam, Kucing, Kambing, Sapi).

3.3. The detection analysis with the application's web system

3.3.1. Research model integration on the application web system

The study successfully integrated the yolov8 model into a web application to detect four types of animals, chickens, cats, goats, and cows real-time using a laptop camera. Testing indicates that the model trained with 125 epochs and the 85% data scheme for training, 10% for validation, and 5% for testing, produces high accuracy during the training process.

To implement this model on the application web, minimum laptop specs with i3 core Intel processor, 8 GB RAM, minimum SSD storage of 20 GB, and Windows 10 operating systems are required. Virtual GPU (Graphics Processing Units) use is also recommended to accelerate the training process.

After that, the Web Application will be implemented on a previously trained model using the API (Application Programming Interface) from the previously trained model obtained from Roboflow, as illustrated in Figure 9. Deployment of the model was carried out using Python and the Flask framework, with installation and setup of the virtual environment via Anaconda. Flask is used as a microframework to develop a web application that integrates the YOLOv8 model for predictions. Additionally, several folders for HTML, CSS, and JavaScript need to be prepared as the visual elements of the website to be used.

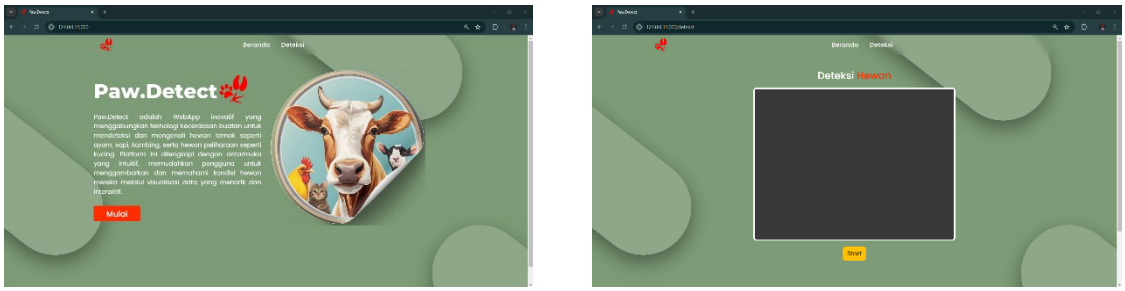
```
[17]  project.version(dataset.version).deploy(model_type="yolov8", model_path="/kaggle/working/runs/detect/train/")

...   View the status of your deployment at: https://app.roboflow.com/datadataset/animalsnew/13
      Share your model with the world at: https://universe.roboflow.com/datadataset/animalsnew/model/13
```

Figure 9. Deployment of Pre-Trained Models

Next, to develop the web application, the repository containing the 'app.py' file is cloned or downloaded, and a new environment is created using the Anaconda prompt with the command 'conda create --name animals_env python=3.10.0'. This is followed by activating the environment using 'conda activate animals_env'. All necessary dependencies are installed by running 'pip install -r requirements.txt'.

The web application is developed using Flask, with 'app.py' as the application's entry point. The trained YOLOv8 model is utilized for prediction through the '/predict' endpoint. The web application is run with the command 'python app.py', and it can be accessed through the provided link, such as http://127.0.0.1:5000/.

In Figure 10, the initial page of the web application is displayed, featuring two main menus: Home and Deteksi. The Deteksi feature is used to perform real-time detection of four animal types Chicken, Cat, Goat, and Cow using a laptop camera. To access this feature, you can click the Start button or directly select Deteksi from the main menu at the top center, which will navigate to the Detection page.

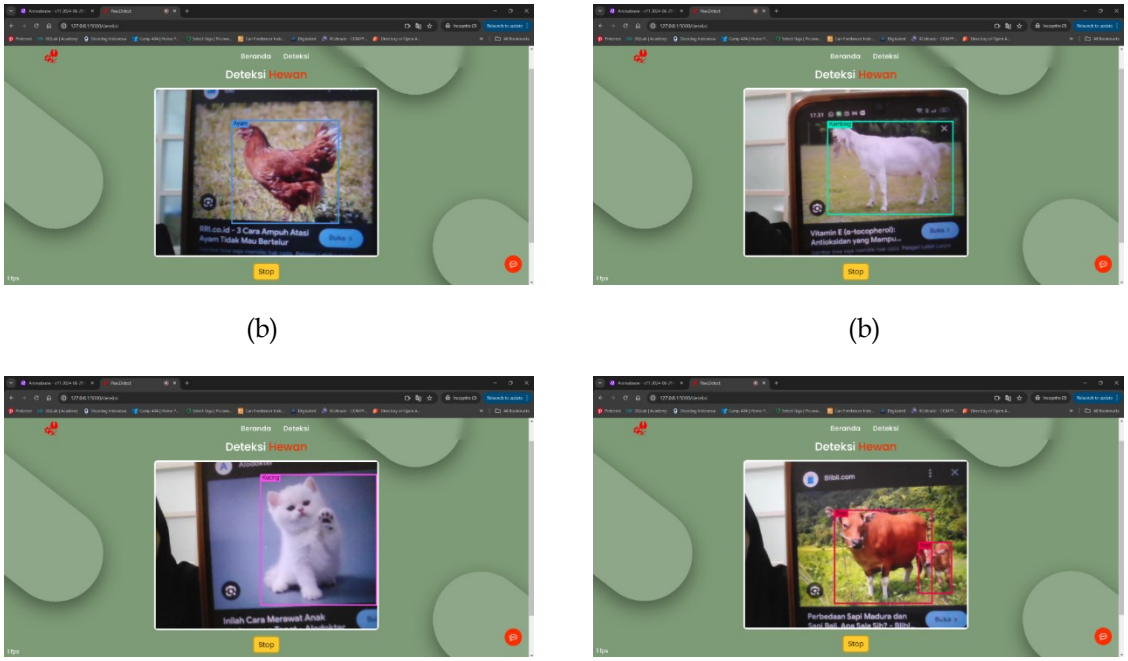


(a) (b)

Figure 10. Display of the initial web page (a) and detection page (b) after deploying

3.3.2. Detection Testing Results with the Web Application System

The testing was conducted using two methods: image-based testing and direct observation of live animals. The image-based testing results can be seen in Figure 11. The system successfully detected chicken, cat, goat, and cow objects with accuracy matching the ground truth, indicating that the model has been well-trained.

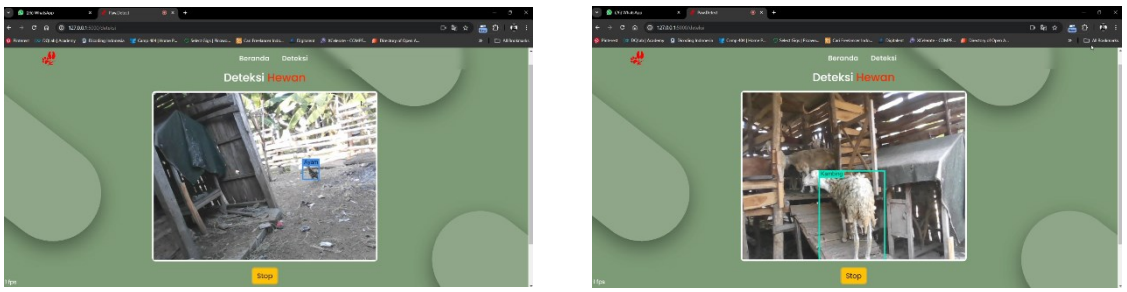


(b) (b)

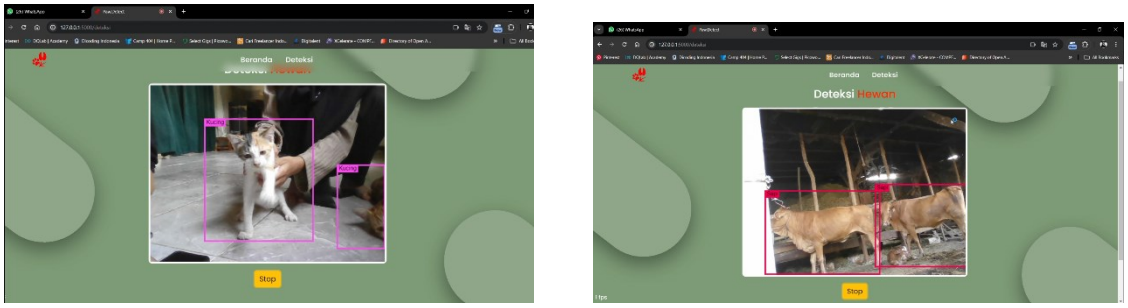
(c) (d)

Figure 11. Detection results using real-time images of Chickens (a), Goats (b), Cats (c), and Cows (d).

The live animal testing results are shown in Figure 12. Detection was performed in real-time using a laptop camera. The results for the cow, goat, and cat show that the system could detect the objects, but the bounding boxes often did not align with the actual position of the objects. This was due to several factors, such as the similarity of the object's color to the background, the large number of animals, and the influence of sunlight. However, in the chicken testing, the system successfully detected the object with high accuracy due to the absence of other disturbances.



(a) (b)



(c) (d)

Figure 12. Detection results of real farmed animals, including Chickens (a), Goats (b), Cats (c), and Cows (d) in real-time.

Conclusion

The conclusions of this study indicate that the YOLOv8 algorithm successfully detected four types of animals (Chicken, Cat, Cow, and Goat) with high accuracy under controlled conditions using static images. The best performance was achieved with a data split scheme of 85% for training, 15% for validation, and 5% for testing, yielding an Accuracy of 0.995, Precision of 0.794, Recall of 0.799, and an F1 Score of 0.796. The web application system, tested in real-time, also successfully detected animals with 100% accuracy. However, the model faced challenges in real-world scenarios involving movement, changing lighting, and multiple objects. To improve detection performance in real environments, it is recommended to increase the training data, perform data augmentation, optimize hyperparameters, and adjust the model to handle undetected objects and diverse environmental conditions. Implementing these recommendations is expected to enhance the system's accuracy and reliability in various conditions.

References

Anand, D., Raj, M. A. M., & Priya, C. S. (2021). Real-time animal detection using YOLO and tracking using DeepSORT. Journal of Physics: Conference Series, 1916, 012107. https://doi.org/10.1088/1742-6596/1916/1/012107

Chai, Y., Du, M., & Yuan, Y. (2022). YOLOv5s-based detection algorithm for beef cattle in a complex natural pasture. Computers and Electronics in Agriculture, 200, 107225. https://doi.org/10.1016/j.compag.2022.107225

Chen, T., Li, Y., & Zhang, J. (2022). Target detection algorithm for goose behavior based on improved YOLOv5. Sustainability, 14(22), 14946. https://doi.org/10.3390/su142214946

Han, L., Zhang, C., Yang, Z., Liu, S., & Li, Y. (2023). Animal detection and recognition using improved YOLOv5 algorithm. Sustainable Computing: Informatics and Systems, 38, 100791. https://doi.org/10.1016/j.suscom.2023.100791

Han, Z., Xie, J., Yu, D., & Yuan, H. (2022). YOLOv4-based multi-object detection algorithm for recognizing farm animals. International Journal of Agricultural and Biological Engineering, 15(1), 181–188. https://doi.org/10.25165/j.ijabe.20221501.6556

Irwansyah, E., & Firmansyah, R. (2021). YOLOv4 algorithm for detection of wildlife species in the forest. International Journal of Artificial Intelligence Research, 5(1), 31–42. https://doi.org/10.29099/ijair.v5i1.290

Kadam, R. A., & Nikam, S. S. (2020). Real-time animal detection using convolutional neural networks. International Journal of Advanced Research in Science, Communication and Technology, 10(2), 220–226. https://doi.org/10.48175/ijarsct-2324

Khadijah, K., & Saepudin, A. (2021). Deteksi hewan liar menggunakan YOLOv3 untuk keamanan kendaraan di jalan. Jurnal CoreIT: Jurnal Hasil Penelitian Ilmu Komputer dan Teknologi Informasi, 7(2), 103. https://doi.org/10.24014/coreit.v7i2.14593

Liang, J., Song, T., He, X., & Wang, Y. (2021). Performance evaluation of YOLO and Faster R-CNN for real-time pig detection. Information Processing in Agriculture, 8(4), 758–767. https://doi.org/10.1016/j.inpa.2020.10.005

Liu, Y., Zheng, Y., Zeng, Y., & Wang, Z. (2023). Animal detection using YOLOv8 with lightweight attention modules. IEEE Access, 11, 64395–64406. https://doi.org/10.1109/ACCESS.2023.3278672

Lu, D., Zhang, H., Liu, B., & Yang, Y. (2023). YOLOv7-based cattle detection in complex environments. IEEE Access, 11, 8703–8712. https://doi.org/10.1109/ACCESS.2023.3241830

Munir, M., Khan, S., Khan, A. A., & Jan, Z. (2021). Real-time animal detection system using YOLOv5. Procedia Computer Science, 184, 781–788. https://doi.org/10.1016/j.procs.2021.03.097

Nuraeni, A., Dewi, R. K., & Suryani, D. (2022). Deteksi hewan ternak menggunakan YOLOv5. Jurnal Riset Informatika, 4(1), 18–24. https://doi.org/10.30656/jri.v4i1.4275

Putri, A. D., & Lestari, R. I. (2023). Deteksi hewan ternak berbasis YOLOv5 dan Google Colab. Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer, 7(3), 321–330.

Rahayu, L., Sarwinda, D., & Wulandari, F. (2023). Implementasi deep learning object detection YOLOv5 untuk identifikasi satwa liar endemik Papua. Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK), 10(4), 821–830. https://doi.org/10.25126/jtiik.2023105643

Raj, D. P., & Bhavya, K. R. (2023). Application of deep learning algorithms for classification and detection of animals using YOLO. International Journal of Research Publication and Reviews, 4(2), 439–443.

Sahab, S., Saputra, R., & Sulistijono. (2022). Animal detection and counting using YOLOv5 and DeepSORT. Jurnal Elektro dan Telekomunikasi Terapan, 9(2), 152–158. https://doi.org/10.25124/jett.v9i2.5781

Suji, K. R. (2023). Deep learning based animal species detection using YOLOv7. International Journal of Advanced Research in Science, Communication and Technology, 17(1), 166–170. https://doi.org/10.48175/IJARSCT-2916

Ultralytics. (2023). YOLOv8 Docs. Retrieved from https://docs.ultralytics.com

Wang, C. Y., Bochkovskiy, A., & Liao, H. Y. M. (2023). You Only Learn One Representation: Unified network for multiple tasks. arXiv preprint. https://arxiv.org/abs/2306.11527

Wulandari, M., Sari, N. P., & Kurniawan, D. E. (2021). Pengembangan sistem deteksi hewan dengan YOLO berbasis Android. Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer, 5(12), 4665–4672.

Xu, L., Yang, W., & Zhang, Q. (2022). A YOLO-based animal detection and tracking method for smart farming. Sensors, 22(11), 4190. https://doi.org/10.3390/s22114190

Yuan, W., Jiang, J., & Huang, Y. (2022). Research on pig behavior detection based on YOLOv5. Sensors, 22(23), 9324. https://doi.org/10.3390/s22239324

Zhang, Y., Fang, S., & Xu, W. (2022). Real-time livestock detection using an improved YOLOv4-tiny algorithm. Sensors, 22(15), 5521. https://doi.org/10.3390/s22155521.