UNIVERSITI TEKNOLOGI MARA

AUTOMATED MONKEY DETECTION SYSTEM

MOHAMAD RAFIQ BIN MOHAMAD SOFI

BACHELOR OF COMPUTER SCIENCE (HONS.)

MARCH 2024

Universiti Teknologi MARA

AUTOMATED MONKEY DETECTION SYSTEM

MOHAMAD RAFIQ BIN MOHAMAD SOFI

Thesis submitted in fulfilment of the requirements for Bachelor of Computer Science (Hons.) College of Computing, Informatics and Mathematics

March 2024

SUPERVISOR APPROVAL

AUTOMATED MONKEY DETECTION SYSTEM

By

MOHAMAD RAFIQ BIN MOHAMAD SOFI

2023385615

This thesis was prepared under the supervision of the project supervisor, Syed Mohd Zahid Syed Zainal Ariffin (Ts Dr). It was submitted to the College of Computing, Informatics and Mathematics and was accepted in partial fulfilment of the requirements for the degree of Bachelor of Computer Science (Hons.).

Approved by

………………………………

Ts. Dr. Syed Mohd Zahid Syed Zainal Ariffin

JULY 8, 2024

STUDENT DECLARATION

I certify that this thesis and the project to which it refers is the product of my own work and that any idea or quotation from the work of other people, published or otherwise are fully acknowledged in accordance with the standard referring practices of the discipline.

………………………………

MOHAMAD RAFIQ BIN MOHAMAD SOFI

2023385615

JULY 8, 2024

ACKNOWLEDGEMENT

Alhamdulillah, praises and thanks to Allah because of His Almighty and His utmost blessings, I was able to finish this research within the time duration given. Firstly, my special thanks goes to my supervisor, Ts Dr Syed Mohd Zahid Syed Zainal Ariffin for assisting me with this momentous task. Special appreciation also goes to my beloved parents for financially assisting my project and giving motivation to complete my proposal. Finally, I would like to give my gratitude to my dearest friend.

ABSTRACT

This research focuses on developing an automated monkey detection system utilizing machine learning models to mitigate human-wildlife conflicts. The study evaluates different deep learning techniques, specifically convolutional neural networks (CNNs) and YOLO algorithms, to enhance detection accuracy and efficiency. The system integrates IoT technologies, including motion sensors and high-resolution cameras, to provide real-time detection and alert mechanisms. Field tests demonstrate the system's potential to reduce property damage and improve safety in areas prone to monkey intrusions. This project addresses the urgent need for advanced technological solutions in wildlife management, promoting human-wildlife coexistence through automated, accurate, and efficient detection methods. The results indicate that the implemented system can significantly enhance wildlife monitoring practices, ensuring both wildlife conservation and the protection of human interests.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AI	ARTIFICIAL INTELLIGENCE
CNN	CONVOLUTIONAL NEURAL NETWORK
IoT	INTERNET OF THINGS
PIR	PASSIVE INFRARED
SVM	SUPPORT VECTOR MACHINE
YOLO	YOU ONLY LOOK ONCE

CHAPTER 1

INTRODUCTION

This chapter provides the background and rationale for the study. It also gives details of the significance of privacy over the Internet, the issues and problems that led to this research.

1.1 Project Background

Wildlife interactions with human populations have long been an issue around the world. In many locations, such interactions pose enormous challenges to people, affecting livelihoods, health, and safety. Wildlife species, such as monkeys, elephants, and wild boars, frequently conflict with humans, resulting in economic losses, injuries, and even deaths. Addressing these difficulties necessitates a thorough understanding of the root causes and the development of appropriate mitigation techniques.

Human-wildlife conflicts are becoming increasingly common as cities expand into the forest and destroy wildlife habitats. As animals seek food and shelter, they often venture into human-inhabited areas, leading to property damage, crop losses, and sometimes direct threats to human safety. Schell et al. (2020) conclude that as the world becomes increasingly urbanized, animals are compelled to adjust quickly. Such adjustments are exacerbating levels of conflict globally, with the recent global COVID-19 pandemic as a significant example.

Wildlife has a significant impact on Malaysia's agricultural economy. Farmers commonly report crop damage from animals like monkeys, wild boars, and elephants. These animals are drawn to the abundant food sources in cultivated lands, resulting in significant economic losses. According to research by the Department of Wildlife and National Parks Peninsular Malaysia, crop damage caused by wildlife has been observed in various states, resulting in annual losses of millions of ringgits (PERHILITAN, 2018). Beyond economic consequences, wildlife can pose significant health and safety risks. Monkeys, for example, have become aggressive when threatened or offered food (Sadili, 2016). They can injure people, especially children and women (New Straits Times, 2022).

Balancing wildlife conservation with human interests is a challenging task. With its vast biodiversity, Malaysia has undertaken considerable efforts in wildlife conservation, such as the National Elephant Conservation Action Plan, or 'NECAP'; However, spreading human activities into wildlife areas has exacerbated conflicts (Lim et al., 2017). Human-wildlife conflicts pose significant concerns that demand meticulous and diverse solutions to tackle. In Malaysia, the growing expansion of human activity into wildlife areas is worsening these conflicts, affecting agriculture, health, and safety. Effective mitigation techniques, including technical advancements such as automated detection systems, are critical for controlling these conflicts and promoting human-wildlife coexistence. One method of detecting animals is template matching, which finds small image regions where the patterns in the actual image and the template image match. Applications such as tracking and identifying wild animals have used this technique. Machine learning methods are also frequently used to identify and categorize animals. Identifying and classifying animal behaviours usually involves extracting features, such as HOOF (Histogram of Oriented Optical Flow) or PHOG (Pyramid Histogram of Oriented Gradients), from images or video frames. Classifiers such as Support Vector Machine (SVM) or KNN (K-Nearest Neighbors) are then used. A potent machine learning technique called deep learning is increasingly used to identify animals in photos and videos. It has demonstrated encouraging results in a variety of computer vision applications. Thus, continued research and collaboration among government agencies, conservationists, and local communities are vital for developing long-term solutions that use artificial intelligence to reduce the manual work of analysis.

1.2 Problem Statement

Human-wildlife interactions in Malaysia present significant challenges, especially in the agricultural sector, where animals like monkeys, elephants, and wild boars cause extensive crop damage, leading to substantial economic losses and safety risks. As urban areas expand into forested regions, wildlife habitats are destroyed, forcing animals into human territories and exacerbating conflicts. The COVID-19 pandemic has further highlighted these issues, as reduced human activity led to increased wildlife incursions into urban areas (Schell et al., 2020). Current manual research methods to mitigate these conflicts are labour-intensive, time-consuming, and prone to errors. Researchers must physically set up and maintain monitoring equipment, such as microphones and cameras, often in difficult and dangerous terrains. They manually collect and analyse data, using spectrographic analysis to identify animal calls and track movements, which is both cumbersome and inaccurate. Despite significant conservation efforts like the National Elephant Conservation Action Plan (NECAP), these manual approaches cannot adequately address the increasing frequency and severity of human-wildlife conflicts (Lim et al., 2017). There is an urgent need for advanced technological solutions to enhance efficiency and accuracy in wildlife monitoring. Artificial intelligence (AI) offers significant potential in this regard, as it can automate detection and tracking, reducing the need for manual intervention and improving the precision of wildlife management. The integration of AI into wildlife management practices is essential for developing long-term strategies for coexistence, ensuring both wildlife conservation and the protection of human interests. Addressing human-wildlife conflicts in Malaysia requires continued research and collaboration among government agencies, conservationists, and local communities to effectively implement AI-driven solutions and promote human-wildlife coexistence (Sadili, 2016; New Straits Times, 2022; PERHILITAN, 2018).

1.3 Project Objective

- To integrate deep learning model for IoT device to detect and recognise monkey.
- To develop the software and hardware components of the automated monkey detection system
- To test the accuracy in detecting and recognising monkey in real-world situations.

1.4 Project Significance

The research into an Automated Monkey Detection System is caused by several problems. First, it seeks to address the lack of an automated observation system, particularly in areas where human habitation intersects with natural habitats. By developing a system that can efficiently detect and alert monkey presence, this project aims to mitigate the risk of property damage and disease transmission from monkeys. According to Ilic et al. (2022), In the global human monkeypox outbreak in 2022, the transmission of monkeypox seems to be highly unusual because for the first time the disease has been reported in countries that have not been affected before and have no epidemiological connection with countries that previously reported human monkeypox (such as the endemic countries in West and Central Africa). This project can benefit local communities living in areas prone to monkey attacks, reducing property damage and improving livelihoods and social well-being in the community.

1.5 Scope and Limitation

Scope outlines what the project aims to achieve and includes its technical, functional, and system aspects. Limitations refer to the constraints or conditions that may restrict or affect the project's capabilities or scope of operation.

1.5.1 Scope

The scope of this project is classified as technical, functional, and system. The technical scope focuses on developing a system capable of detecting monkeys in real time using camera feeds with high accuracy and low false-positive rates. This entails leveraging advanced machine learning models and techniques to ensure precise detection and recognition of monkeys. From a functional perspective, the project aims to utilize these machine-learning models to identify monkeys in live video streams, provided the detection process is efficient and reliable. The system scope covers the broader objective of real-time detection of monkeys, which includes not only accurate identification but also the capability to send immediate alerts to user, ensuring timely responses to potential monkey intrusions.

1.5.1 Limitation

The limitation of this project is that detection only works on monkey. Area of coverage are limited to the camera visibility.

CHAPTER 2

LITERATURE REVIEW

This section provides an overview of existing animal detection systems, emphasizing the role of machine learning and sensor technologies in enhancing the accuracy and efficiency of wildlife monitoring. It discusses traditional methods, their limitations, and the advancements brought by automated systems, particularly in mitigating human-wildlife conflicts.

2.1 Introduction

Animal detection is essential for wildlife conservation, preventing human-wildlife conflicts, and optimizing agricultural practices. Traditional methods, such as manual tracking and camera traps, pose significant challenges due to their labour-intensive nature and limited efficiency. This chapter reviews the literature on animal detection methods, the Internet of Things, and development methodology.

2.2 The Necessity of Animal Detection

The Necessity of Animal Detection is an important task that serves various critical needs. Foremost among these is public safety, as the ability to detect the presence of wildlife, especially in urban or residential areas, can help prevent dangerous encounters between humans and animals. This is crucial for protecting both people and the animals themselves from harm. Beyond public safety, effective animal detection also supports broader efforts in environmental conservation. By monitoring and protecting endangered species and managing human-wildlife conflicts that threaten animal populations and habitats, animal detection technologies play a crucial role in preserving the natural world. In transportation, detecting animals near roads and railways is vital for preventing collisions, which can be devastating for both the animals and vehicle occupants. Animal detection systems that trigger alerts or automated responses can help avoid these dangerous incidents, enhancing overall safety. Animal detection also has essential applications in agricultural settings, where it can help identify and deter pests that threaten crops and livestock. This safeguards food production and protects the livelihoods of farmers. Finally, animal detection technologies enable researchers to study animal behaviour, movement patterns, and population dynamics more accurately and efficiently. This advancement in scientific understanding of the natural world is a crucial outcome of these capabilities. Continued advancements in animal detection are crucial to addressing these diverse needs.

2.3 Animal Detection Methodology

Deep learning is an advanced machine learning method that leverages multi-layered processing to automatically extract features from data, as noted by LeCun (2015). While traditional machine learning methods necessitate specialized knowledge for feature engineering, deep learning distinguishes itself by identifying features autonomously, especially from images. This capability allows it to outperform traditional machine learning approaches in tasks such as speech and image recognition, given a sufficiently large dataset on which to train. As such, the advantages of deep learning in feature extraction and performance in complex recognition tasks make it a preferred choice. Therefore, this analysis will focus solely on comparing different deep learning techniques, given their demonstrated superiority and efficacy.

2.3.1 Convolutional Neural Network

Convolutional neural networks (CNNs) are artificial intelligence systems based on multi-layer neural networks that can identify, recognize, and classify objects and detect and segment objects in images. CNN or ConvNet is a famous discriminative deep learning architecture that could be learned directly from the input object without the obligation for human feature extraction (Koushik, 2016; Bezdan & Džakula, 2019). This network is frequently used in visual identification, medical image analysis, image segmentation, NLP, and many other applications since it is specifically designed to deal with various 2D shapes (Koushik, 2016; Bezdan & Džakula, 2019). It is more effective than a regular network since it can automatically identify key elements from the input without human participation.

2.3.2 Convolutional Neural Network Fundamentals

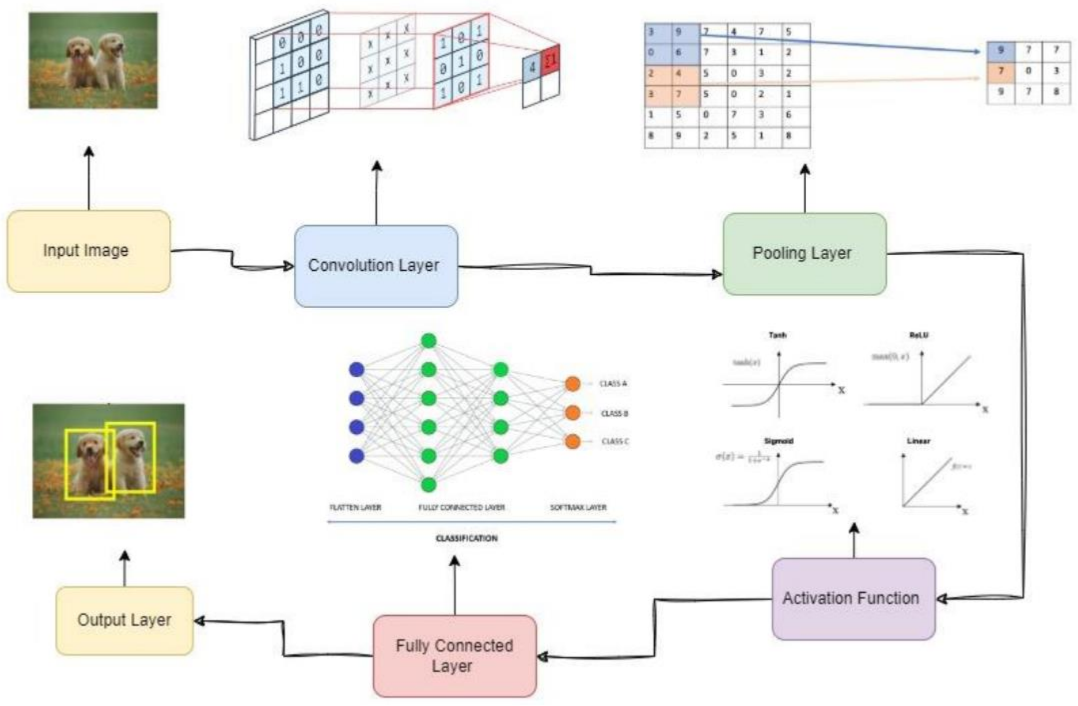


Figure 2.1 CNN Network (*Source*: Taye, 2023)

CNN layers are typically composed of four types: convolutional, pooling, function of activation, and fully connected.

2.3.2.1 Input Image

An image is a matrix of pixel values. Based on the resolution and size of the image, a computer will perceive different matrices, such as a 32 x 32 x 32 matrix (3 refers to RGB values). The matrix's digits each have a value between 0 and 255 that indicates the pixel's current grey level (Zhang, 2018).

2.3.2.2 Convolution Layer

Convolution's primary goal is to extract features from the input image. Convolution learns image attributes using a small square matrix that maintains the spatial relationships between pixels (Zhang, 2018). The input image was "shifted," or convolved, over the kernels to extract its features. The dot products of the kernel matrix and the input image were computed for each convolution operation. The picture below shows a visual depiction of the procedure.

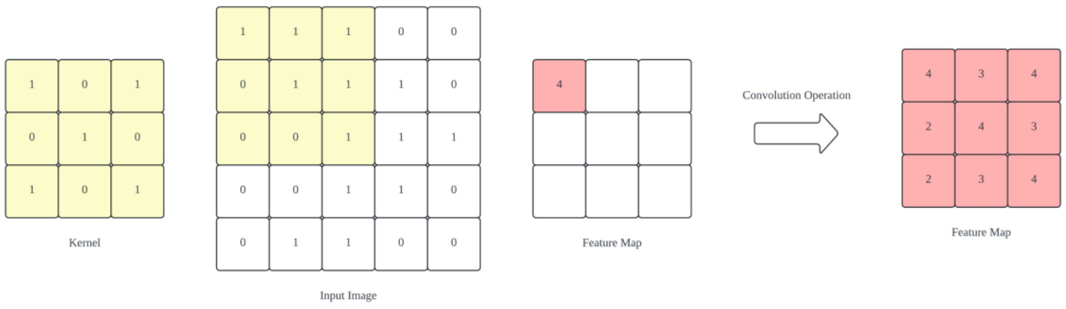


Figure 2.2 Convolution Operation (*Source*: Zhang, 2023)

Figure 2.2 shows the convolution operation. The feature map is the matrix that slides the kernel on the original image and performs the convolution operation.

Every neuron in a feature map has the same weight parameter, called a kernel or filter. For the original input image, the filter functions as a feature detector. For the same image, different filters will yield different feature maps. Changing the filter values makes achieving effects like edge detection, sharpening, and blurring possible. This indicates that different filters pick up on different image elements, like curves and edges. The image's specifications keep getting smaller when the step size is raised. This issue can be resolved by filling the "0" boundaries surrounding the input image. As demonstrated in Figure 2.3, the addition of '0' to the image preserves its original geometry. The size of the image has also been preserved due to the '0' borders. Because the output image and the input image have the same size, this is known as the same padding. The '0' borders have also helped to maintain the image's size.

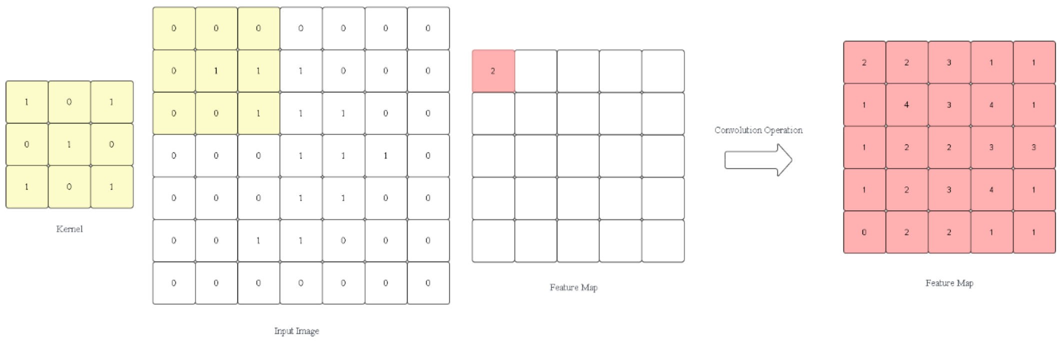


Figure 2.3 Same Padding (*Source*: Zhang, 2023)

2.3.2.3 Pooling Layer

Pooling layers, also called subsampling of the CNN are used after convolutional layers (Zhu et al., 2018). Reducing the dimensionality of the feature layer that emerges from the convolution is the primary goal of a pooling layer. Reducing the model's complexity and quantity of parameters is crucial (O'Shea & Nash, 2015). Several functions have been added to this stage since the pooling layer is used to downscale the dimensionality of the input. The "MAX" function is the most often utilized for scaling input (O'Shea & Nash, 2015, p. 8). Max-pooling is often the "MAX" function utilized in a pooling layer. When using this method, 2 x 2 dimensional kernels are applied to the input layer. When such a pooling layer is used, only the pixel with the highest value out of four will be listed in the output. As a result, the input feature layer's dimension is decreased (Zhu et al., 2018). There are numerous ways to do pooling: max pooling, average pooling, sum pooling, and more. Max Pooling is the most popular type, as seen in Figure 2.4.

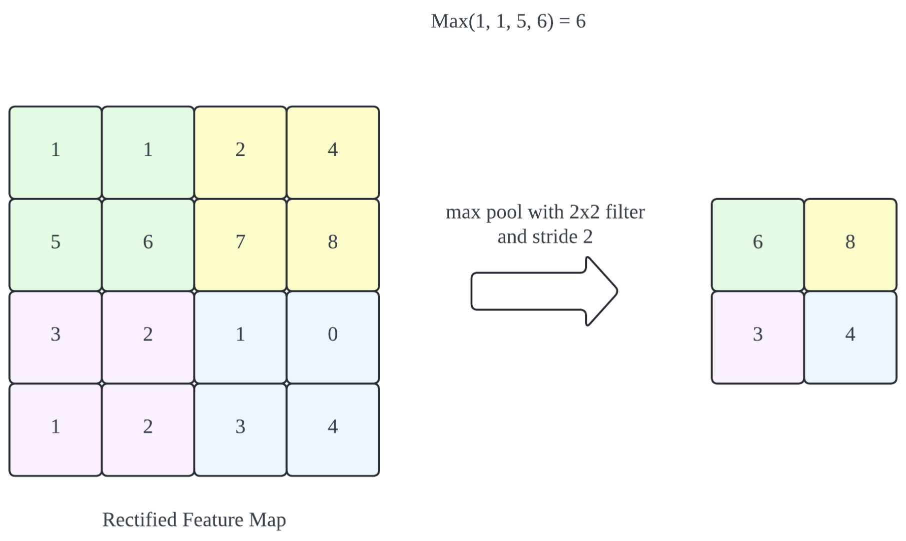


Figure 2.4 Max Padding (*Source*: Zhang, 2023)

2.3.2.4 Nonlinear Activation Function

An appropriate nonlinear activation function can greatly enhance the network's performance. The activation function's purpose is to establish a functional relationship between the input and output, introducing a nonlinear system into the neural network (Gu et al., 2018). Several popular activation functions are displayed in Figure 2.5. Tanh and the sigmoid are referred to be saturating nonlinearities among them. Figure 2.5 and the formula definition demonstrate that the Tanh function saturates at output -1 or 1, and the Sigmoid function saturates at output 0 or 1, depending on how big or tiny the input is. Non-saturating nonlinearities, such as ReLU by Nair & Hinton (2010), Leaky ReLU by Maas et al. (2013), PreLU by He et al. (2015), RReLU Xu et al. (2015), and ELU by Zeiler & Fergus (2013), have been presented as solutions to the issues produced by saturating nonlinearities. Rectified Linear Units (ReLUs) are neurons with non-saturating nonlinearities; they function significantly more slowly than the latter in terms of the amount of time needed for gradient descent training. Compared to comparable networks using tanh as the activation function, this kind of deep convolutional neural network with ReLUs operates several times faster (Nair & Hinton, 2010).

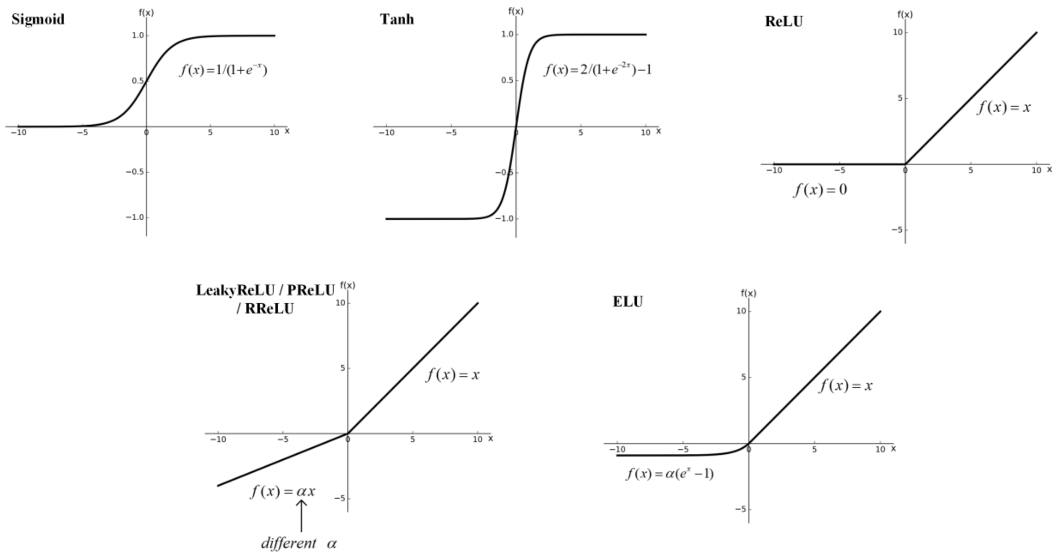


Figure 2.5 Max Padding (*Source*: Chen *et al*., 2021)

2.3.2.5 Fully Connected Layer

Fully connected layers in a Convolutional Neural Network (CNN) are crucial as they act as classifiers. These layers receive the output from the convolutional and pooling layers, which act as feature extractors. The fully connected layer processes these features to determine which category they best fit.

A fully connected layer observes the advanced features extracted by the previous layers and determines their weights to calculate the correct probability for different categories. For example, if a CNN determines the content of a picture as a dog, the feature map with higher values represents advanced features such as claws or four legs. Similarly, the feature map might highlight the wings or beaks of a bird.

The fully connected layer is a Multi-Layer Perceptron that often uses the Softmax activation function as the output layer. The Softmax function converts a vector of real values into a vector of probabilities, each representing the likelihood of a particular category. When the fully connected layer processes the input, it outputs an N-dimensional vector, where N represents the number of categories, and each element represents the probability of a specific category.

The fully connected layers are the final layers within a CNN's structure. They reshape the feature maps from the convolutional and pooling layers into n-dimensional vectors. These vectors allow for deeper and more abstract feature extraction, which is essential for further prediction tasks or classification.

After the last fully connected layer, the activation function typically differs from those used within other layers. For classification tasks involving multiple classes, a function like Softmax is used to normalize the probabilities of the target classes, ensuring that the output is a probability distribution over the predicted categories.

2.3.3 YOLOv5 Algorithm

The initial single-stage object identification technique that Redmon J. devised is called YOLO. The two-stage algorithm's candidate box extraction step is dropped, and the bounding box and classification are combined into a single regression problem. The YOLO algorithm operates as follows: The image is first split up into $S \times S$ meshes. Predicting where the actual box will land in the grid's centre is the task of each grid. From these meshes, a total of $S \times S \times B$ bounding boxes are generated. The target centre point coordinates, the target width and height (x, y, w, h), and the confidence level on the target's containment are the five parameters that each bounding box has. $S \times S$ grids predict the target's category probability within that grid. The category score for every prediction box is then calculated by multiplying the category probability and prediction bounding box confidence. To get the final prediction results, non-maximum suppression (NMS) is used to filter these prediction boxes. The YOLO v5 algorithm ensured speed of operation while achieving a precise accuracy of about 50 mAP in the COCO dataset (Lin et al., 2015).

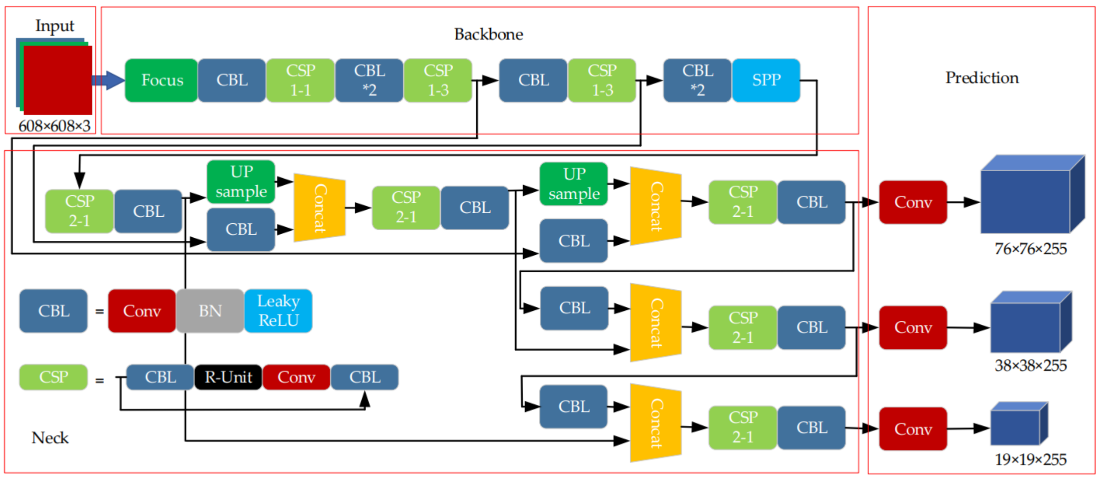


Figure 2.6 YOLOv5 Architecture (*Source*: Zhang *et al*., 2022)

To increase the effectiveness of small target detection, the conventional YOLO v5 randomly scales, crops, arranges and stitches the input using mosaic data augmentation. During dataset training, the input image size is adjusted to a consistent size, specifically 460 × 460 × 30, and then fed into the model for analysis. The original anchor frame for YOLO v5 is (116, 90, 156, 198, 373, 326). The backbone network comprises the Cross-Stage Partial connection structure (CSP) and Focus structure, with the Focus structure slicing the image before it enters the backbone network. The original 608 × 608 × 3 image is diced to form a 304 × 304 × 12 feature map, which is then processed by 32 convolutional kernels through the Focus operation, downsampling the input dimensions while preserving the original image data. The input feature is transitioned into the CSP structure using two 1 × 1 convolutions, enhancing CNN learning capabilities, overcoming computational constraints, and reducing memory expenses. The Neck section, which gathers and transfers picture details to the prediction layer, uses the FPN+PAN structure in YOLO v5. The FPN upsamples and fuses high-level feature data top-to-bottom, while the PAN communicates strong positional attributes bottom-to-top. The prediction layer creates a bounding box based on the image's features, using GIOU_Loss for Boundingbox, which is more efficient than conventional nonmaximum suppression (NMS) in overlapping object detection.

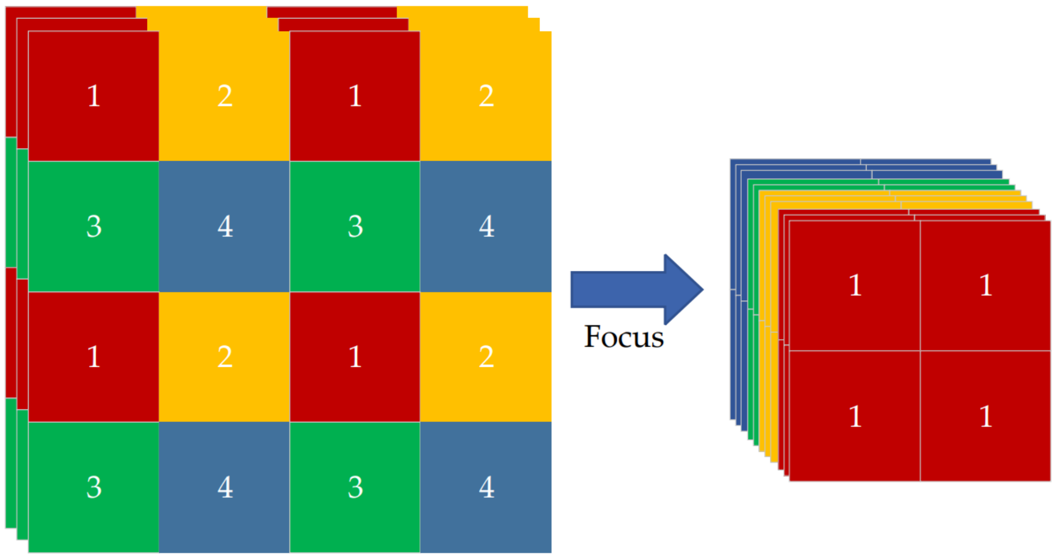


Figure 2.7 Process flow of focus module (*Source*: Zhang *et al.*, 2022)

2.3.4 YOLOv8 Algorithm

A similar framework as YOLOv5 is used by YOLOv8, but with certain changes made to the CSPLayer, which is now known as the C2f module. To improve detection accuracy, the C2f module combines contextual information with high-level features through a two-convolution cross-stage partial bottleneck. Unlike YOLOv5, YOLOv8 is an anchor-free model with decoupled heads that can perform regression, classification, and object detection tasks on its own. Because of this architecture, the accuracy of the model is increased as each branch concentrates on its role. The sigmoid function is used in the YOLOv8 output layer as the activation function for object scores, which represent the likelihood that an object will be inside the bounding box. They express class probabilities—that is, the likelihood that an object will belong to each potential class—using the softmax function.

YOLOv8 uses binary cross-entropy for classification loss and the Complete Intersection over Union (CIoU) and Distribution Focal Loss (DFL) loss functions for bounding box loss. The object-detection performance is improved by these loss functions, particularly for tiny items.

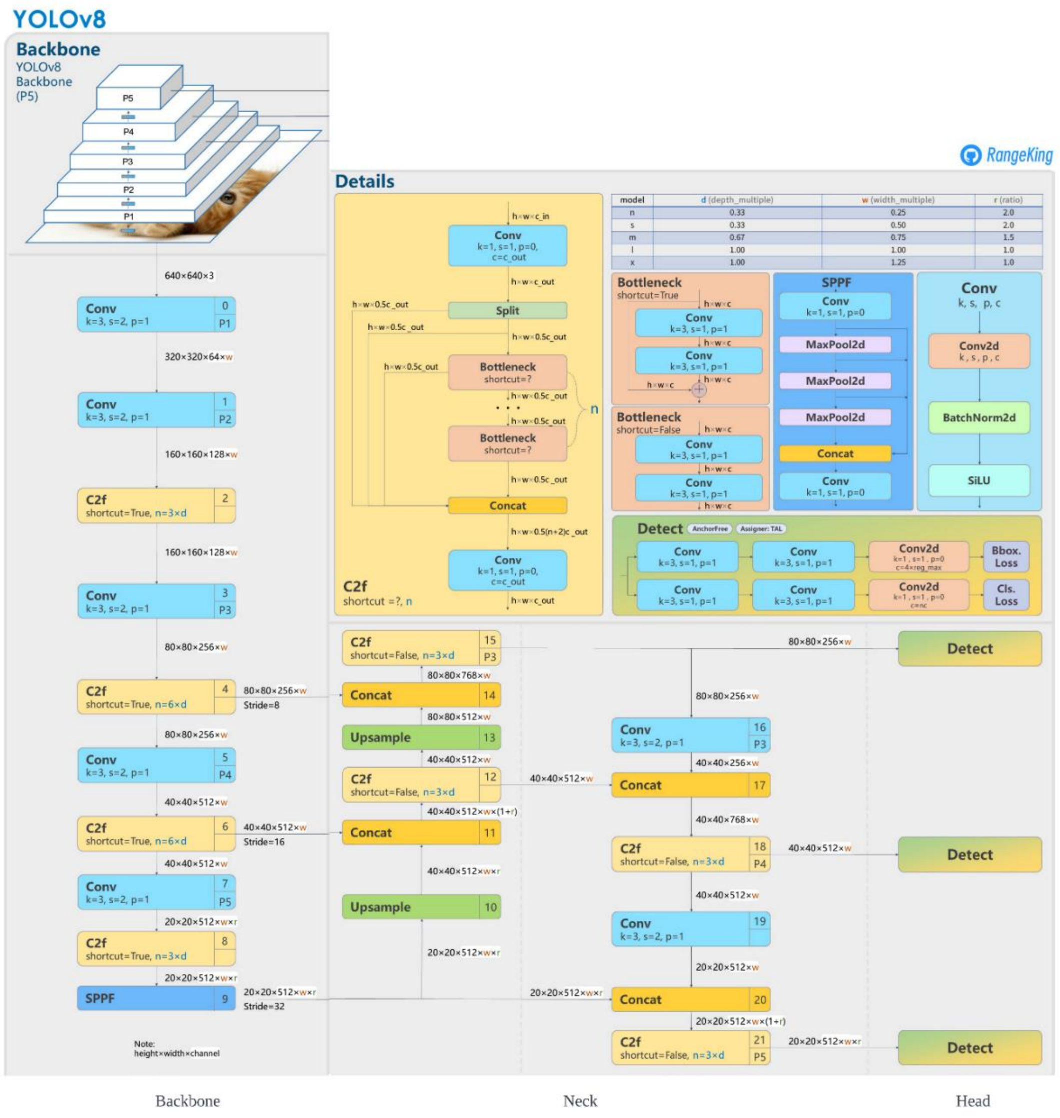


Figure 2.8 YOLOv8 Architecture (*Source*: RangKing, 2023)

2.3.4.1 Backbone

The backbone network of YOLOv8 is a modified version of CSPDarknet53 (Redmon & Farhadi, 2018). The input features are down-sampled five times to yield five different scale features, each designated as P1–P5. Figure 2.8 depicts the backbone network's structure. The C2f module and its structure take the place of the Cross Stage Partial (CSP) module in the original backbone network. To improve the feature extraction network's information flow without sacrificing lightweight, the C2f module uses a gradient shunt connection. To produce the output result, the CBS module, or "Conv" in the graph, first applies a convolution operation to the input data, then batches normalization, and lastly uses SiLU to activate the information stream. In the end, the backbone network pools the input feature maps to a fixed-size map for adaptive size output using the spatial pyramid pooling fast (SPPF) module. By successively connecting the three maximum pooling layers, SPPF has a shorter latency and less computing effort than the spatial pyramid pooling (SPP) structure (He et al., 2014).

2.3.4.2 Neck

Figure 2.8 illustrates how YOLOv8 is structured, with a Path Aggregation Network-Feature Pyramid Network (PAN-FPN) structure at the neck, taking inspiration from PANet (Liu et al., 2018). In contrast to the YOLOv5 and YOLOv7 models' neck structures, the YOLOv8 model preserves original performance while attaining a lightweight design by eliminating the convolution operation in the Path Aggregation Network (PAN) structure following up-sampling. The two distinct feature scales in the PAN structure and Feature Pyramid Network (FPN) structure of the YOLOv8 model are indicated by the numbers 10–13 and 16–19, respectively. To communicate deep semantic information, conventional FPN takes a top-down method. By combining numbers 5–10 and 3–13, the FPN improves the semantic information of the features, but some object localization information will be lost in the process. PAN-FPN integrates PAN with FPN to address this issue. PAN realizes path enhancement in a top-down fashion by combining numbers 10-19 and 13-16, which improves the learning of location information. By building a top-down and bottom-up network structure, PAN-FPN achieves feature variety and completeness by fusing features to realize the complementarity of deep semantic information and shallow positional information.

2.3.4.3 Head

The detecting component of YOLOv8 employs a decoupled head structure, as seen in Figure 2.8. The decoupled head structure employs distinct branches for object classification and predicted bounding box regression, employing distinct loss functions for each task. The binary cross-entropy loss (BCE Loss) is employed for the classification problem. The distribution focal loss (DFL) by Li et al. (2020) and CIoU by Zheng et al. (2020) are used to predict bounding boxes. This detection framework has the potential to enhance detection precision and expedite model convergence. YOLOv8 is a detection model that does not rely on predefined anchor boxes to identify objects. Instead, it accurately defines positive and negative samples. In addition, the model utilises the Task-Aligned Assigner Feng et al. (2021) to allocate samples dynamically, hence enhancing the model's detection accuracy and robustness.

2.3.4.4 YOLOv8 Scaled Model Comparison

To enable flexible deployment on hardware devices across broad application scenarios, the YOLOv8 model has been modified to produce five unique scaled models by manipulating two parameters: width and depth. The models are designated as YOLOv8n, YOLOv8s, YOLOv8m, YOLOv8l, and YOLOv8x. The parameters and resource consumption of the five models rise sequentially, resulting in an improvement in the detection performance. Figure 8 displays the dimensions, and the maximum number of channels associated with these five models. Table 1 illustrates the performance of each scaled model based on the COCO val2017 dataset. The mAPval numbers represent the performance of a single model at a single scale on the COCO val2017 dataset. Meanwhile, the average speed was calculated by utilising an Amazon EC2 P4d instance using COCO validation pictures.

Table 2.1 Comparison of Scaled YOLOv8 Model

Model	Size (pixels)	mAP^{val} 50-95	Speed CPU ONNX	Speed A100 TensorRT	params (M)	FLOPs (B)

			(ms)	(ms)		
YOLOv8n	640	18.4	142.4	1.21	3.5	10.5
YOLOv8s	640	27.7	183.1	1.40	11.4	29.7
YOLOv8m	640	33.6	408.5	2.26	26.2	80.6
YOLOv8l	640	34.9	596.9	2.43	44.1	167.4
YOLOv8x	640	36.3	860.6	3.56	68.7	260.6

Source: Jocher *et al.*, 2023

2.3.5 Similar System

This research project will analyse two existing systems that use YOLO for animal image recognition.

2.3.5.1 Fruit Ripeness Identification using YOLOv8 Model

Fruit Ripeness Identification using YOLOv8 Model is an article under Multimedia Tools and Applications Journal, volume 83, pages 28039–28056, (2024). The purpose of this article is to identify fruit ripeness.

The dataset utilised for this research comprises images depicting a variety of fruits at several stages of ripeness. The specific particulars of the dataset, such as the quantity of photos and the distribution between the training and testing sets, are not disclosed in the extracted parts. Nevertheless, the primary emphasis is on determining the ripeness of apples and pears.

Table 2.2 Training parameters for YOLOv8

Parameters	Value
optimizer	Stochastic Gradient Descent (SGD)
batch	2
mask_ratio	4
box	7.5
cls	0.5
weight_decay	0.0005

Source: Bingjie *et al.*, 2023

Table 2.3 Result of Scaled YOLOv8 Model

Weight	Epoch	Class	AP0.5	AP@0.5:0.95	Average inference time(ms)
YOLOv8n	100	Ripe apple	0.995	0.991	2.9
		Overripe apple	0.995	0.990	
		Ripe pear	0.995	0.995	
		Overripe pear	0.995	0.994	
YOLOv8m		Ripe apple	0.994	0.992	6.6
		Overripe apple	0.995	0.992	
		Ripe pear	0.995	0.994	
		Overripe pear	0.995	0.994	
YOLOv8x		Ripe apple	0.995	0.993	17.2
		Overripe apple	0.995	0.991	
		Ripe pear	0.995	0.994	
		Overripe pear	0.995	0.993	

Source: Bingjie *et al.*, 2023

Based on Table 2.3, it is evident that the YOLOv8x model outperforms others in terms of accuracy, earning the greatest AP@0.5:0.95 values for all classes. More precisely, it achieves a score of 0.993 for ripe apples and overripe pears and a score of 0.994 for ripe pears. YOLOv8x demonstrates exceptional precision in identifying the ripeness stages of fruits, making it the optimal selection for situations demanding the highest level of accuracy. Conversely, the YOLOv8n model performs exceptionally well in terms of inference time, with an impressive average processing speed of only 2.9 ms. YOLOv8n has superior speed compared to previous models, rendering it very well-suited for real-time applications that require rapid processing. Although YOLOv8n is significantly less precise than YOLOv8x, it still achieves very high AP@0.5:0.95 values. Specifically, it achieves an AP@0.5:0.95 of 0.991 for ripe apples, 0.990 for overripe apples, 0.995 for ripe pears, and 0.994 for overripe pears. Ultimately, YOLOv8x is the most suitable model for applications that require the utmost precision, while YOLOv8n is the best option for situations prioritising speed.

2.3.5.2 Intelligent Detection Method for Wildlife Based on Deep Learning

Intelligent Detection Method for Wildlife Based on Deep Learning is an article under Sensor Journal, volume 23, issues 24. The purpose of this article is to find the best detection method for wildlife based on deep learning.

The self-built wildlife dataset includes images of three types of animals: Budorcas taxicolor (takin), Rhinopithecus (monkey), and Panthera uncia (leopard). It consists of 5724 instances, categorized into small (107), medium (1145), and large (4472) instances. The dataset contains 3796 images, divided into 2654 for training, 762 for validation, and 380 for testing. These images are from wildlife videos taken in national parks, public datasets, and online documentaries. Various wild environments characterize the dataset and includes images that are blurred, poorly illuminated, and blocked, providing a challenging context for wildlife detection models.

Table 2.4 Training parameters for self-build wildlife dataset

Parameters	Value
optimizer	Adam
lr0 (Initial learning rate)	0.001
amp (Automatic Mixed Precision)	True
close_mosaic	30
mixup	0.5
mosaic	0.5
save_period	5

Source: Li *et al.*, 2023

Table 2.5 Experimental Results for YOLOv5 and YOLOv8 on Self-Built Dataset

Model	mAP	AP-L@0.5

YOLOv5	0.885	0.97
YOLOv8	0.929	0.967

Source: Li *et al.*, 2023

Based on Table 5, YOLOv8 has the highest level of accuracy across all the models. The YOLOv8 model achieves a mean average precision (mAP) of 0.929, surpassing the mAP of YOLOv5, which is 0.885. The greater mean Average Precision (mAP) of YOLOv8 demonstrates its superior performance in accurately recognising and classifying items throughout the whole dataset. Although YOLOv5 has a slightly higher AP-L@0.5 value of 0.97 compared to YOLOv8's 0.967, this discrepancy is negligible and does not substantially affect the overall performance. The mean Average Precision (mAP) is a crucial indicator that evaluates the model's accuracy more thoroughly, considering all classes and object sizes. Thus, although YOLOv5 has a modest edge in recognising large objects, YOLOv8's superior overall mean average precision (mAP) validates it as the more precise model.

2.3.5.3 Summary

The YOLOv8 model demonstrates superior accuracy, with a mean average precision (mAP) of 0.929, surpassing the mAP of 0.885 achieved by YOLOv5. The greater mean Average Precision (mAP) of YOLOv8 indicates its improved ability to detect and identify objects reliably throughout the collection. While YOLOv5 has a slightly higher AP-L@0.5 value of 0.97 compared to YOLOv8's 0.967, this discrepancy is negligible and does not substantially impact the overall performance. YOLOv8x is the most precise model in terms of AP@0.5:0.95 values for all classes. It scores 0.993 for ripe apples and overripe pears and 0.994 for ripe pears. Despite being somewhat less accurate than YOLOv8x, YOLOv8n stands out regarding inference time, boasting an average processing speed of 2.9 ms. YOLOv8n makes it well-suited for real-time applications. Therefore, YOLOv8x is the suggested choice for applications that demand the utmost accuracy, whilst YOLOv8n is more suitable for scenarios prioritising speed.

2.4 Internet of Things

According to Madakam et al., 2015, the Internet of Things is best defined as "An open and comprehensive network of intelligent objects that have the capacity to auto-organise, share information, data and resources, reacting and acting in face of situations and changes in the environment". The Internet of Things is reaching a state of maturity and is currently the most widely discussed and highly anticipated topic in information technology. In recent years, the Internet of Things (IoT) concept has gained significant interest. It refers to a global network of interconnected physical objects that allows for constant interaction and communication, not limited to individuals. The Internet of Things is a worldwide network that enables communication between human-to-human, human-to-things and things-to-things. It accomplishes this by assigning a unique identity to every object in the world (Aggarwal & Das, 2012). The term "Internet of Things" (IoT) refers to a global network in which nearly any object or device can be interconnected and communicate with each other in a more intelligent manner than ever before. The Internet of Things (IoT) refers to the connection of sensors and actuators that are integrated into physical objects, such as highways and pacemakers. These devices are interconnected over wired and wireless networks, typically utilising the same Internet Protocol (IP) that connects to the Internet.

2.4.1 Motion Detection

Motion detection refers to identifying moving things inside a given environment through diverse sensor technologies. Sensors have become essential and irreplaceable equipment in our daily lives. The market for sensors has experienced a significant surge, given their widespread application across several domains. Increased demand has led to notable progress in sensor technology, improving reliability and cost-effectiveness (Lobachev & Cretu, 2016). Various sensors gauge multiple physical properties, including temperature, pressure, strain, ultrasonic, microwave, and passive infrared (PIR). Ultrasonic, microwave, and PIR sensors are commonly employed for motion detection. PIR sensors are favoured over the other two options because they specifically detect signals produced by an individual, hence optimising power usage (Liu et al., 2015). PIR sensors are passive devices that detect thermal radiation emitted by a moving object within their detection range. These sensors provide rapid reaction, affordability, minimal power usage, dependable performance, and high precision. PIR sensors are favoured in numerous applications due to their superior benefits over other types of motion sensors. They are utilised for intrusion detection, automated instrument control, and artificial intelligence.

2.4.2 Distance Measurement

Distance measurement is the process of determining the separation between two points or objects in space. Several methods and techniques are used to measure distance, each with its own advantages and limitations. One standard method is ultrasonic sensors, as described by Gorman & Knipe (2021). Based on Figure 2.8, ultrasonic machines determine distance by measuring the time a pulsed sound wave travels to a reflective interface and back. The distance is calculated using the formula distance = (speed x time)/2, where the speed is the assumed propagation speed of sound waves through human tissue (1540 m/sec).

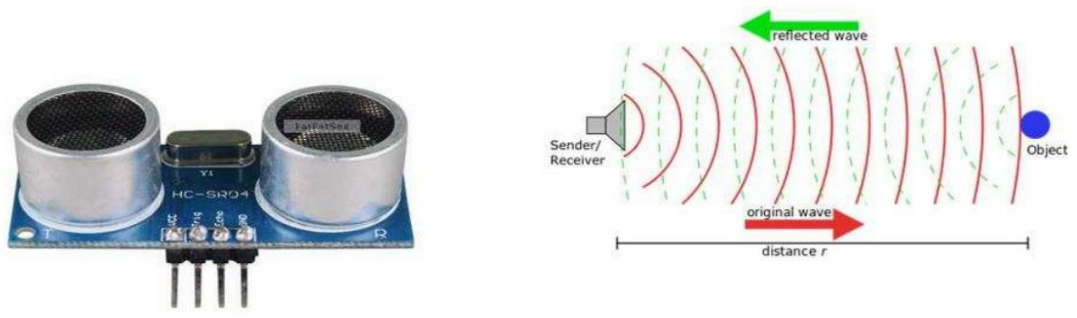


Figure 2.9 Ultrasonic sensor working process (*Source*: Latha *et al*., 2023)

2.4.3 Summary

The Internet of Things (IoT) is a network of intelligent objects that can self-organize, share information, and respond to environmental changes (Madakam et al., 2015). It involves interconnected physical objects with unique identities, enabling communication between human-to-human, human-to-things, and things-to-things (Aggarwal & Das, 2012). IoT integrates sensors and actuators into everyday objects, connecting them via wired and wireless networks using Internet Protocol (IP). Motion detection, an essential aspect of IoT, uses ultrasonic, microwave, and passive infrared (PIR) sensors to identify moving objects. PIR sensors are preferred for their low power usage, high precision, affordability, and dependable performance, making them ideal for intrusion detection and automated control (Liu et al., 2015). Distance measurement in IoT is often achieved using ultrasonic sensors, which calculate the distance by timing the travel of a sound wave to a reflective surface and back, using the formula distance = (speed x time)/2 (Knipe & O'Gorman, 2021).

2.5 Development Methodology

Software development methodology is a structured approach to planning, organizing, and designing an information system project. It typically consists of four steps within the Software Development Life Cycle: planning, analysis, design, and implementation. Pavaloaia (2012) states that the domain of a software project determines the selection of a software development methodology. This literature analysis will specifically examine the prevailing approaches that are frequently employed. The approaches that will be explored are waterfall, iterative, and iterative waterfall.

2.5.1 Waterfall Methodology

In 1970, Winston Royce, a pioneer in software development, devised a waterfall model. Although it is one of the oldest models in the Software Development Life Cycle (SDLC), it has not been regularly used in recent times. The process follows a linear sequential progression, with advancements being produced and flowing down through the stages of development. At this location, all requests are gathered and then proceed to the subsequent phase of the project. Each stage in the process relies on the information gathered in the previous stages, as it prohibits the progression to the next phase unless the preceding phase is finished. The waterfall approach lacks the flexibility to backtrack to a previous stage or accommodate modifications. The waterfall model is suitable for small projects due to its limited adjustment capacity once each stage is finalised. In the waterfall approach, issues are not addressed until the development phase progresses to the maintenance stage. The stages of a waterfall model consist of requirements analysis, system design, implementation, testing, deployment, and maintenance.

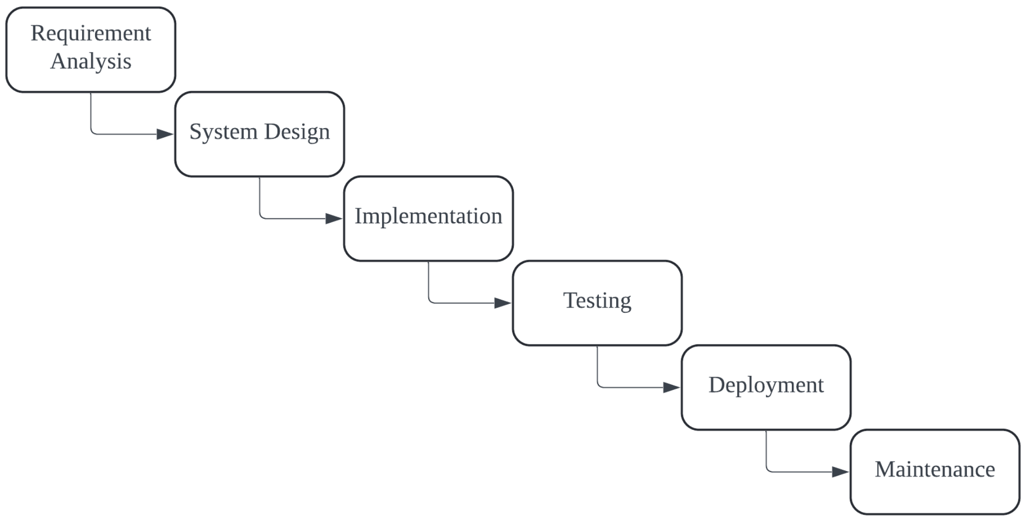


Figure 2.10 Waterfall Methodology (*Source*: Gupta, 2022)

2.5.2 Iterative Methodology

Iterative development is an approach that involves allocating time specifically for revising and enhancing different components of a system (Halvorsen, 2020). In this scenario, the feature code is planned, created, and tested in multiple cycles. Before starting, a comprehensive set of specifications is unnecessary, as progress can be initiated by identifying and implementing specific software components. This technique is then repeated to generate another software version at the end of each iteration cycle. The iterative development paradigm in Figure 11 divides the progress of a large application into smaller components. It is considered the act of decomposing the software development of a large program into smaller components.

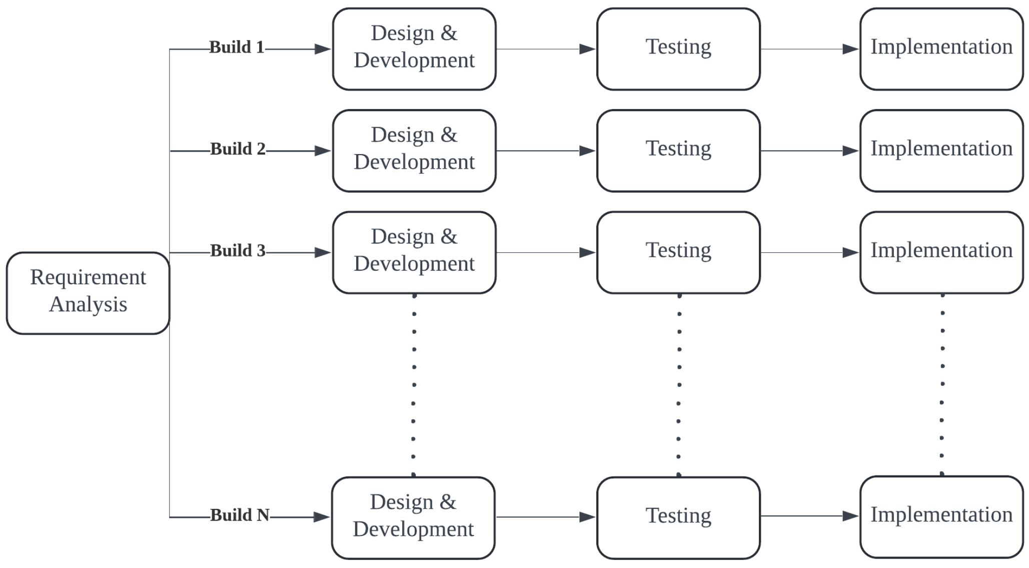


Figure 2.11 Iterative Methodology (*Source*: Okesola, 2020)

2.5.3 Iterative Waterfall Methodology

The problems with the classical waterfall model created a demand for a new model. The iterative model came into existence to cope with the problems of the original waterfall model. The iterative waterfall model is an enhanced version of the classical waterfall model, which could provide faster results, require less time, and have good flexibility. In an iterative model, the project is divided into small parts, allowing the developer team to go easily and quickly toward their goal and obtain valuable user feedback. No feedback path is provided for the requirement definition phase, so if any change is required, the iterative model does not have scope for modification or corrections.

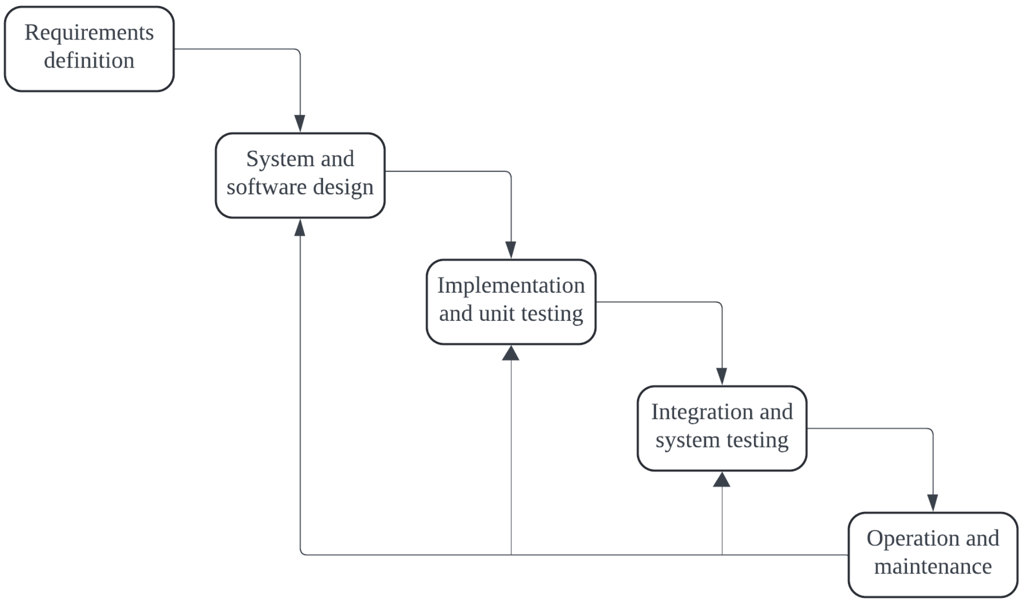

Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

Figure 2.12 Iterative Waterfall Methodology (*Source*: Rather & Bhatnagar, 2015)

2.5.4 Summary

The iterative waterfall methodology combines the methodical, step-by-step approach of the classical waterfall model with the flexible, feedback-oriented cycles of iterative development. This mix provides a robust framework that improves the flexibility and efficiency of managing software projects. The iterative waterfall model facilitates incremental progress by dividing the development process into smaller, manageable chunks. This approach allows teams to adjust and enhance components based on user feedback and testing at each stage. This strategy minimizes the likelihood of significant obstacles and enhances the overall progress schedule by enabling modifications without disrupting the project process. The approach efficiently combines the waterfall model's predictability with the iterative cycle's adaptability, making it well-suited for projects that need a combination of structure and flexibility.

CHAPTER 3

METHODOLOGY

This section explores the integration of IoT technologies in wildlife monitoring, highlighting their capabilities for real-time data collection and analysis. It covers the use of IoT-enabled devices, such as motion sensors and cameras, and discusses how these technologies improve the responsiveness and effectiveness of wildlife management strategies.

3.1 Introduction

Iterative waterfall development-based methodology was used. Major considerations that had been considered when selecting the development methodology were flexibility and simplicity. Traditionally, iterative methodology involves feasibility study, requirements analysis, design, coding, testing, and maintenance. However, for this project, the methodology was modified to requirements analysis, design, implementation, and testing as seen by Figure 3.1. By using the Iterative Waterfall Methodology, changes and improvements can be made at each stage of the development process, instead of waiting until the end of the project. In this manner, any mistake can be rectified in each phase of the project. It is important to note that once requirements analysis has been completed, it cannot be changed, ensuring a clear and stable foundation for the subsequent stages.

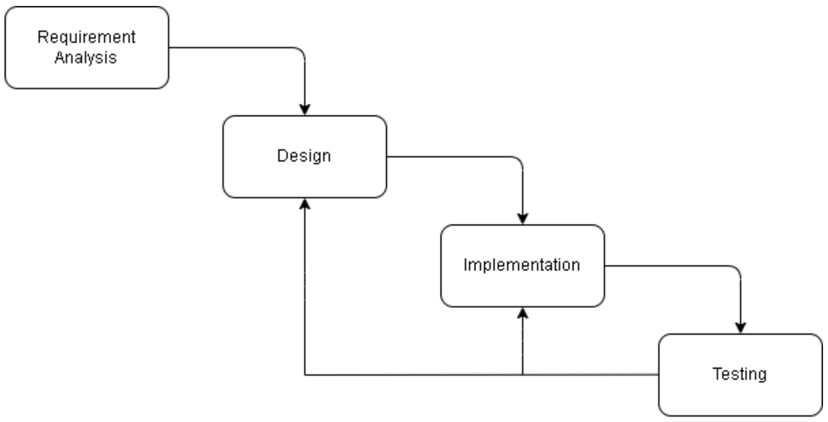


Figure 3.1 Iterative Waterfall Methodology

3.2 Project Framework

The iterative waterfall technique integrates all phases of software development in a linear process while still allowing for flexibility. This section will comprehensively explain the many stages involved in the process. Figure 3.2 displays the deliverables associated with each phase of the Iterative Waterfall methodology.

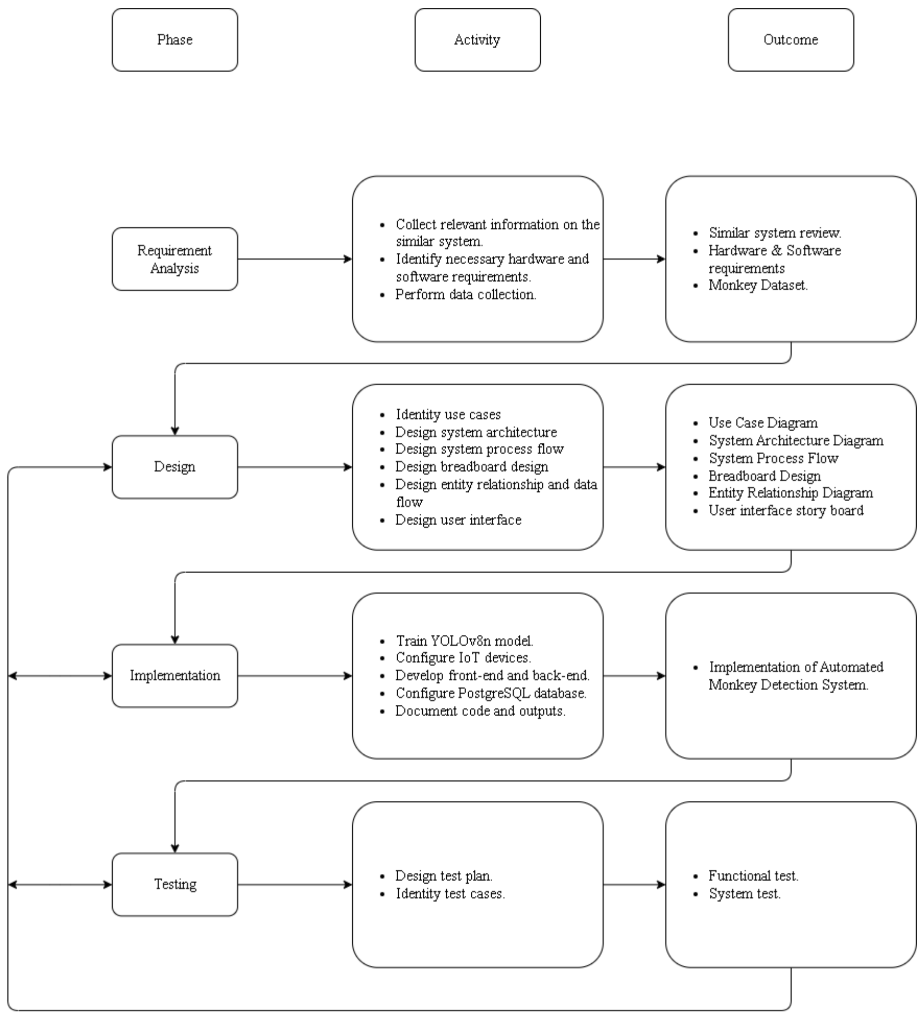
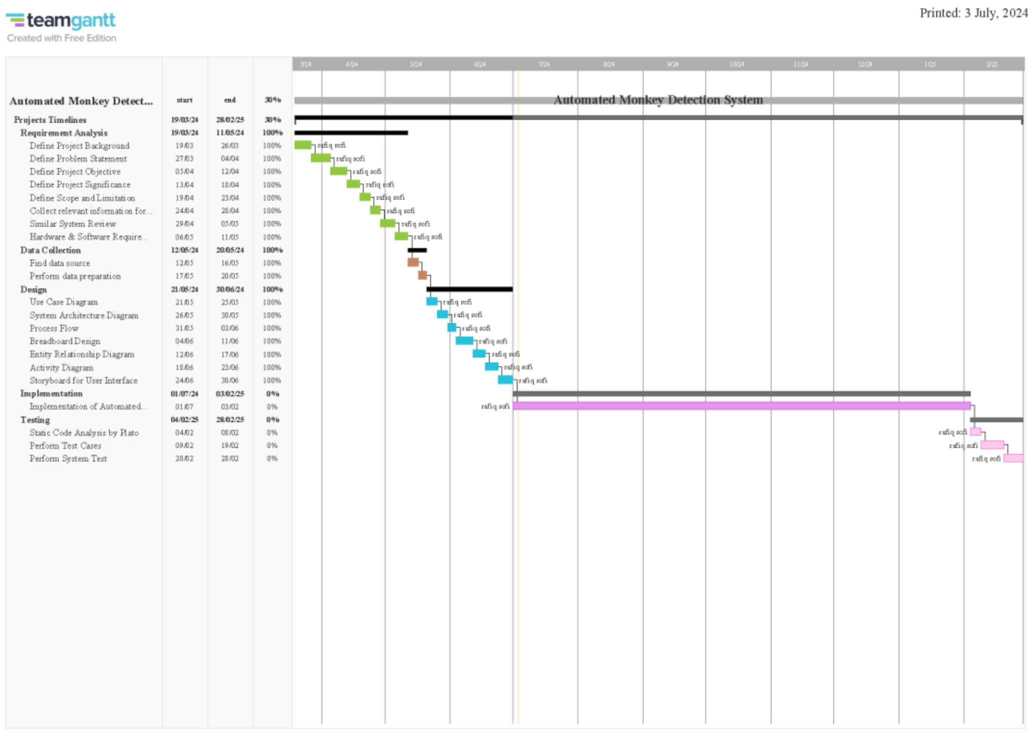

Phase
Activity
Outcome
Requirement Analysis
Collect relevant information on the similar system.
Identify necessary hardware and software requirements.
Perform data collection.
Similar system review.
Hardware & Software requirements
Monkey Dataset.
Design
Identity use cases
Design system architecture
Design system process flow
Design breadboard design
Design entity relationship and data flow
Design user interface
Use Case Diagram
System Architecture Diagram
System Process Flow
Breadboard Design
Entity Relationship Diagram
User interface story board
Implementation
Train YOLOv8n model.
Configure IoT devices.
Develop front-end and back-end.
Configure PostgreSQL database.
Document code and outputs.
Implementation of Automated Monkey Detection System.
Testing
Design test plan.
Identity test cases.
Functional test.
System test.

Figure 3.2 Project Framework Based on Iterative Waterfall


Printed: 3 July, 2024

Automated Monkey Detect...	start	end	30%
Projects Timelines	**19/03/24**	**28/02/25**	**30%**
Requirement Analysis	**19/03/24**	**11/05/24**	**100%**
Define Project Background	19/03	26/03	100%
Define Problem Statement	27/03	04/04	100%
Define Project Objective	05/04	12/04	100%
Define Project Significance	13/04	18/04	100%
Define Scope and Limitation	19/04	23/04	100%
Collect relevant information for...	24/04	28/04	100%
Similar System Review	29/04	05/05	100%
Hardware & Software Require...	06/05	11/05	100%
Data Collection	**12/05/24**	**20/05/24**	**100%**
Find data source	12/05	16/05	100%
Perform data preparation	17/05	20/05	100%
Design	**21/05/24**	**30/06/24**	**100%**
Use Case Diagram	21/05	25/05	100%
System Architecture Diagram	26/05	30/05	100%
Process Flow	31/05	03/06	100%
Breadboard Design	04/06	11/06	100%
Entity Relationship Diagram	12/06	17/06	100%
Activity Diagram	18/06	23/06	100%
Storyboard for User Interface	24/06	30/06	100%
Implementation	**01/07/24**	**03/02/25**	**0%**
Implementation of Automated...	01/07	03/02	0%
Testing	**04/02/25**	**28/02/25**	**0%**
Static Code Analysis by Plato	04/02	08/02	0%
Perform Test Cases	09/02	19/02	0%
Perform System Test	20/02	28/02	0%

Figure 3.3 Gantt Chart

According to the modified Iterative Waterfall Model, the project is structured into five main phases. The first phase, spanning from March 19, 2024, to May 11, 2024, focused on the requirement analysis. This phase included defining the project background, problem statement, project objective, project significance, scope, and limitations, collecting relevant information for similar systems, and hardware and software requirements based on the Iterative Waterfall Model; this phase cannot be reiterated. The second phase, data collection, occurred from May 12, 2024, to May 20, 2024. During this period, tasks included finding data sources and preparing data. The third phase, design, occurred from May 15, 2024, to May 29, 2024. During this period, tasks included designing a Use Case Diagram, designing a System Architecture Diagram, designing a Process Flow, designing a Breadboard Design, designing an Entity Relationship Diagram (ERD) and Activity Diagram, and making a Storyboard for the user interface. The fourth phase, implementation, will be done on July 1, 2024, until February 3, 2025. During this period, task implementation of an automated monkey detection system will be done. The final phase, testing, will be done on February 4, 2025, until February 28, 2025. During this phase, static code analysis by Plato, performing test cases, and system tests will be performed.

3.3 Requirements Analysis

3.3.1 Similar System Review

Intelligent Detection Method for Wildlife Based on Deep Learning is an article under Sensor Journal, volume 23, issues 24. The purpose of this article is to find the best detection method for wildlife based on deep learning.

Table 3.1 Overview of Intelligent Detection Method for Wildlife Based on Deep Learning

Aspect	Description		
Purpose	Develop a deep learning-based model for wildlife detection to aid in wildlife monitoring and conservation efforts by providing accurate data on wildlife presence and behavior.		
Dataset	Self-built wildlife dataset includes images of three types of animals: Budorcas taxicolor (takin), Rhinopithecus (monkey), and Panthera uncia (leopard). It consists of 5724 instances, categorized into small (107), medium (1145), and large (4472) instances. The dataset contains 3796 images, divided into 2654 for training, 762 for validation, and 380 for testing.		
Training Results	Model	mAP	AP-L@0.5
	YOLOv5	0.885	0.97
	YOLOv8	0.929	0.967

Source: Li *et al.*, 2023

Fruit Ripeness Identification using YOLOv8 Model is an article under Multimedia Tools and Applications Journal, volume 83, pages 28039–28056, (2024). The purpose of this article is to identify fruit ripeness.

Table 3.2 Overview of Fruit Ripeness Identification using YOLOv8 Model

Aspect	Description					
Purpose	To identify the ripeness of fruits using the YOLOv8 model, providing an efficient and accurate method for fruit classification based on ripeness levels.					
Dataset	The dataset is divided into training and validation sets in a 9:1 ratio. Includes images of ripe and overripe apples and pears, with different ripeness levels for classification.					
Training Results	Weight	Epoch	Class	AP0.5	AP@0.5:0.95	Average inference time(ms)
	YOLOv8n	100	Ripe apple	0.995	0.991	2.9
			Overripe apple	0.995	0.990	
			Ripe pear	0.995	0.995	
			Overripe pear	0.995	0.994	
	YOLOv8m		Ripe apple	0.994	0.992	6.6
			Overripe apple	0.995	0.992	
			Ripe pear	0.995	0.994	
			Overripe pear	0.995	0.994	
	YOLOv8x		Ripe apple	0.995	0.993	17.2
			Overripe apple	0.995	0.991	
			Ripe pear	0.995	0.994	
			Overripe pear	0.995	0.993	

Source: Bingjie *et al.*, 2023

3.3.2 Platform

The primary platform for this project is the Raspberry Pi, specifically the Raspberry Pi 4B and the Raspberry Pi Zero W for sensor and camera feed management. These platforms are chosen for their affordability, versatility, and widespread use in IoT applications.

Table 3.3 Overview of platform

Platform Name	Operating System	Version	Descriptions
Raspberry Pi 4 Model B	Raspberry Pi OS (64-bit)	Kernel version: 6.6 Debian version: 12 (bookworm)	Main processing
Raspberry Pi Zero W	Raspberry Pi OS (64-bit)	Kernel version: 6.6 Debian version: 12 (bookworm)	sensor and camera feed management

3.3.3 Tools

Key tools include the Yolov8 by Ultralytics, OIDv4 Toolkit by EscVM, OIDv4 Toolkit forked by the AIGuysCode, Node.js, Nextjs by Vercel, Tailwind CSS, and PostgreSQL.

Table 3.4 Overview of tools

Tool Name	Version	Descriptions
Yolov8 by Ultralytics	8.2.38	To train, validate, and to predict data.
OIDv4 ToolKit by EscVM	-	To download data from Google Open Image Dataset v4.
OIDv4 ToolKit forked by theAIGuysCode	-	To do annotation conversion.
Node.js	20.15.0 LTS	Node.js allows for server-side JavaScript execution.
Nextjs by Vercel	-	Back-end framework for user interface.
Tailwind CSS	-	Front-end framework for user interface.
PostgreSQL	16.3	Open-source relational database.
Visual Studio Code	1.90.2	Code editor

3.3.4 Programming Language

Table 3.5 Overview of programming language

Programming Language	Version	Descriptions

Python	3.12.3 (64-bit)	used for the backend and processing tasks
JavaScript	-	used for developing the React-based front-end.
SQL	-	used for managing the database

3.3.5 Data Source

The Google Open Images Dataset v4 is the main data source for this project. This dataset consists of many photos annotated for various things, including animals. The dataset's vast size, diversity, and thorough annotations make it ideal for training object detection models like YOLOv8-m. Monkey photos and annotations were downloaded using the OIDv4 ToolKit; however, the annotations are unsuitable for the Yolo8-m, which uses the Yolo Darknet Format (e.g., <class_label <x_center> <y_center> <width> <height>). To make the dataset useable Convert_annotations.py from OIDv4_ToolKit-Forked-AIGuy is used to transform it.

3.3.6 Data Size and Subjects

For this project, a subset of the Google Open Images Dataset v4 focusing on images containing monkeys was extracted. The number of images extracted totalled to 2717 images. Each image also included it accompanied annotations for objects within the image.

3.3.7 Data Preparation

To begin the process, data is downloaded using the OIDv4_ToolKit-Forked-AIGuy with the parameter "–classes Monkey –type_csv all". This command downloads all available monkey images along with their corresponding annotations from the Google Image Dataset v4, and subsequently splits the data into training, testing, and validation sets. However, the downloaded annotations are not in a format that Yolov8-m can accept. To address this, the conversion.py script from OIDv4_ToolKit-Forked-AIGuy is utilized. This script converts the annotations from the format <name_of_the_class> <left> <top> <right> <bottom> to the Yolo Darknet Format, which is <class_label> <x_center> <y_center> <width> <height>.

3.3.8 Data Splitting

To avoid overfitting, a situation where a model performs remarkably well on training data but fails to generalize to new, unseen examples, the dataset will be split into three subsets: training, testing, and validation. The training dataset, comprising 78.87% of the total dataset, includes 2143 images with corresponding annotations. This subset is used to train the YOLOv8-m model, enabling it to learn the features and patterns associated with monkeys in various contexts.

The testing dataset, making up 15.53% of the total dataset, includes 422 images with annotations and is reserved for testing. This subset evaluates the model's performance, ensuring it accurately detects monkeys in new, unseen images. Lastly, the validation dataset, consisting of 5.59% of the total dataset, includes 152 images with annotations.

3.4 System Design

The system design phase included creating detailed use case diagram, architectural diagrams, database design, process flows, user interface mock-ups, and breadboard diagram. These designs guided the implementation phase.

3.4.1 Use Case Diagram

The use case diagram offers stakeholders a comprehensive overview of the system at a higher level of conceptualization, aiding their understanding. This graphic will illustrate the use cases of each actor and their corresponding activities within the system. Figure 3.4 depicts the use case diagram for the system.

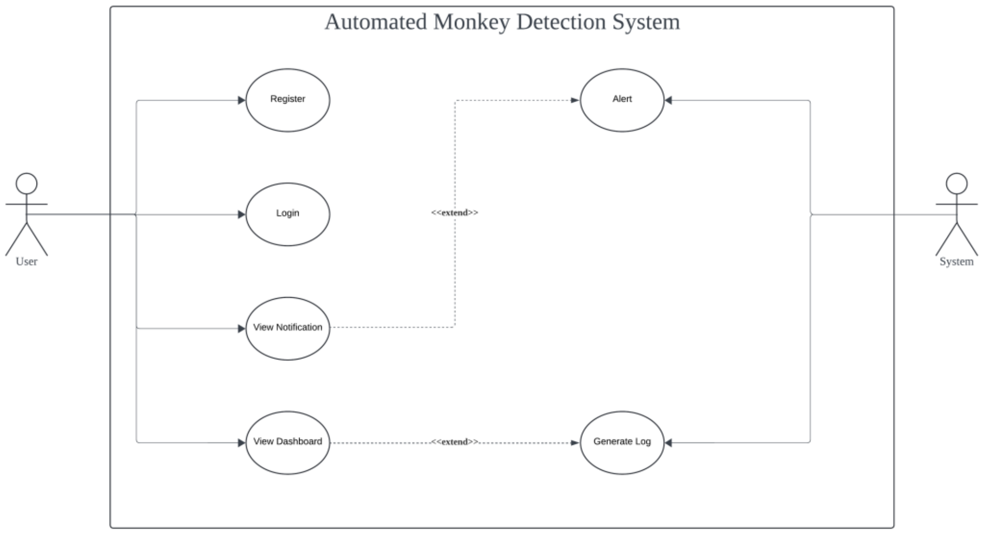


Figure 3.4 Use Case Diagram

The following use case description will describe the details of each use case and its interaction with the actors.

Table 3.6 Use Case Description of Register

Use Case Title:	Register
Actors:	User
Normal flow of events: 1. The user enters Automated Monkey Detection System's register page. 2. Register their credentials of their email, username, and password. 3. Verify credentials. Upon successful registration, the system stores their registration credentials.	

Table 3.7 Use Case Description of Login

Use Case Title:	Login
Actors:	User
Normal flow of events: 1. The user enters Automated Monkey Detection System's login page. 2. Enter their username, and password. 3. Verify credentials. Upon successful login, the user is then redirected to dashboard page.	

Table 3.8 Use Case Description of View Notification

Use Case Title:	View Notification
Actors:	User
Normal flow of events: 1. The user navigates the view notifications section from the navigation bar. 2. The user views the notifications table for any related alerts from the detection logs.	

Table 3.9 Use Case Description of View Dashboard

Use Case Title:	View Dashboard
Actors:	User
Normal flow of events: 1. The user navigates the dashboard section from the navigation bar. 2. The system displays the dashboard, which include. Various data, graph, and live feed from the camera.	

Table 3.10 Use Case Description of View Dashboard

Use Case Title:	Generate Log
Actors:	System
Normal flow of events: 1. The user can choose specific data from the tables 2. The user can export the generated report for further processing outside of the system.	

Table 3.11 Use Case Description of Alert

Use Case Title:	Alert
Actors:	System
Normal flow of events: 1. The system will send alert to related user through SMS, email, or telegram bot.	

3.4.2 System Architecture

The system architecture outlines the overall structure of the monkey detection system, including the hardware and software components and their interactions. This architecture is crucial for understanding how data flows through the system and how different components communicate with each other.

As shown in Figure 3.5, the monkey detection system is built around a Raspberry Pi 4B, which serves as the main processing unit responsible for running the YOLOv8-m model and processing the camera feed. Complementing this, a Raspberry Pi Zero powers the camera feed and various sensors, including PIR sensors for motion detection and ultrasonic sensors for measuring proximity to detect the presence of monkeys. A camera is employed to capture real-time video feeds that are analysed by the YOLOv8-m model to identify monkeys. Upon detection, a truck horn is activated to alert the user. The system also incorporates a database to store detection logs, camera feed data, and user interaction history. The user interface (UI), developed using React, displays the camera feed, recent detections, graphs, and other relevant information, ensuring a user-friendly experience.

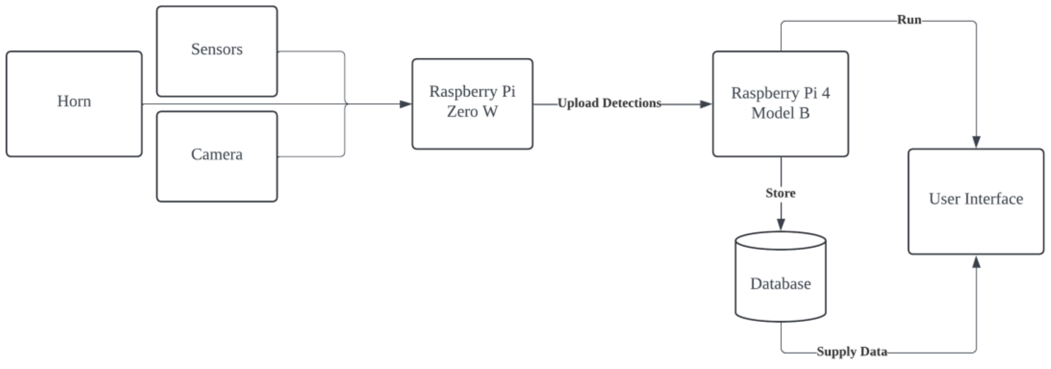


Figure 3.5 System Architecture Diagram

3.4.2 Process Flow

Based on Figure 3.6. The backend system is designed to monitor and respond to motion detection events. Initially, it continuously checks for motion through a condition motionDetected? If no motion is detected, the system loops back to re-evaluate the condition. Upon detecting motion, the system initiates a series of actions: it takes a picture and begins measuring distance using an ultrasonic sensor. The collected data, including the image and distance measurements, are then sent to a Raspberry Pi 4 Model B. This Raspberry Pi stores the data in a PostgreSQL database and triggers alerts to the user through SMS, email, or a Telegram bot. This integrated system ensures that all relevant data is captured and securely stored, and that the user is promptly notified of any detected motion.

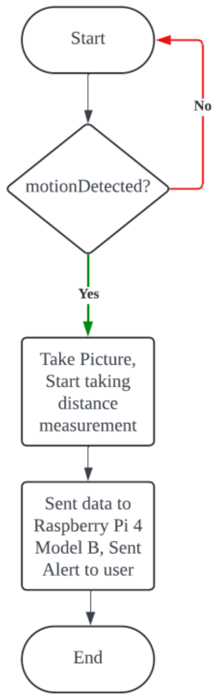

Start
No
motionDetected?
Yes
Take Picture, Start taking distance measurement
Sent data to Raspberry Pi 4 Model B, Sent Alert to user
End

Figure 3.6 Flowchart of back-end process

Based on Figure 3.7. The frontend flow begins with the Login process, where users are prompted to enter their credentials. Upon attempting to log in, the system evaluates the condition isUser?. If the user does not exist (condition is "No"), they are redirected to the SignUp page to create a new account. After successfully signing up, the user is directed back to the Login process. If the user does exist (condition is "Yes"), the system proceeds to the following: isAuthenticated?. If the user is not authenticated (condition is "No"), they are redirected to the Login process to re-enter their credentials. If the user is authenticated (condition is "Yes"), they are redirected to the Dashboard.

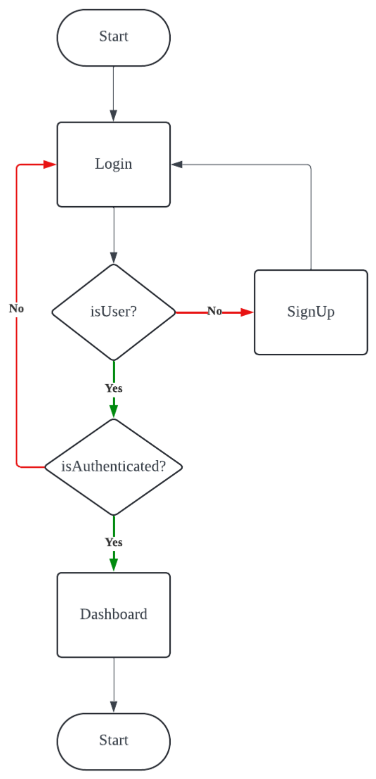


Figure 3.7 Flowchart of front-end process

3.4.3 Breadboard Design

The breadboard design outlines the physical layout of the hardware components and their connections. It is important for prototyping and testing the hardware setup before final deployment.

Based on figure 3.8. The circuit integrates various components to interface with a Raspberry Pi Zero W, including a PIR sensor for motion detection, an HC-SR04 Ultrasonic Sensor for distance measurement, a relay module for controlling power to an external device (Horn), and a Raspberry Pi Camera Module 3 120° for visual data capture. The external device circuit is powered by a PSU (Power Supply Unit). The Raspberry Pi Zero W serves as the central processing unit for managing sensor inputs, controlling outputs based on the programmed logic, and sending data to Raspberry Pi 4 Model B for further processing. Meanwhile, the Raspberry Pi 4 Model B is just for processing data sent from Raspberry Pi Zero W.

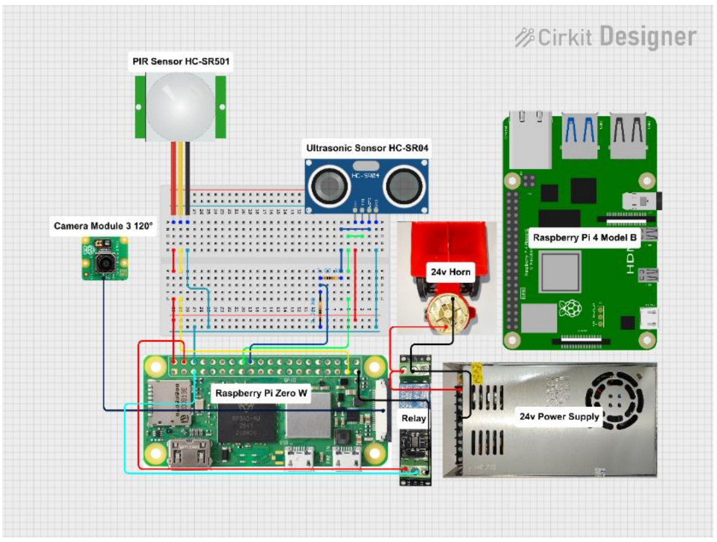


Figure 3.8 Breadboard Design

3.4.4 Database Design

3.4.4.1 Entity Relationship Diagram

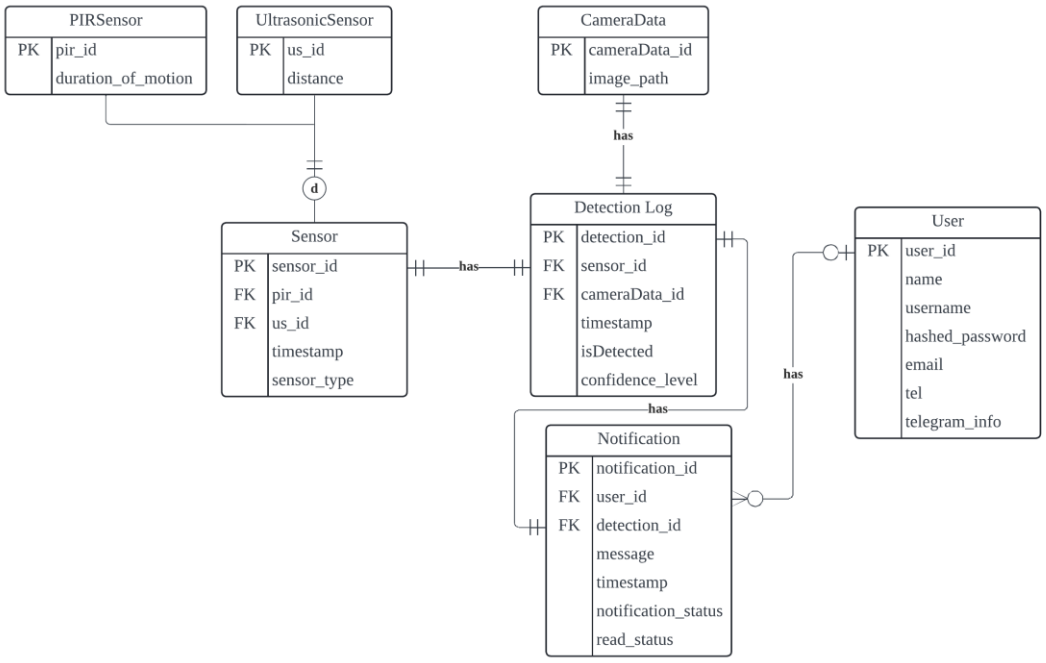


Figure 3.9 Entity Relationship Diagram

The ERD of the system consists of seven tables which are PIRSensor, UltrasonicSensor, CameraData, Sensor, Detection Log, Notification, and User.

Table 3.12 Overview of Entity Relationship Diagram's entity table

Tables	Description
User	Store users' information.
Notification	Store users' notifications information.
Detection Log	Store detection logs.
Sensor	Store the information of all sensors.
PIRSensor	Store the information of pir sensor.
UltrasonicSensor	Store the information of ultrasonic sensor.
CameraData	Store the information of camera.

3.4.4.2 Data Dictionary

Table 3.13 Overview of User's entity Data Dictionary

Column	Data Type	Description	Primary Key	Foreign Key	Referenced table
user_id	Integer	User unique identifier	Yes	No	-
name		User's name			
username	Varchar	User's username	No	No	-
hashed_password	Varchar	User's hashed password using bcrypt	No	No	-
email	Varchar	User's email address	No	No	-
tel	Varchar	User's telephone number used for SMS alert	No	No	-
telegram_info	Jsonb	User's telegram information used for telegram notification	No	No	-

Table 3.14 Overview of Notification's entity Data Dictionary

Column	Data Type	Description	Primary Key	Foreign Key	Referenced table
notification_id	Integer	Notification unique identifier	Yes	No	-
user_id	Integer	User unique identifier	No	Yes	User
detection_id	Integer	Detection unique identifier	No	Yes	Detection Log
message	Text	Notification's message	No	No	-
timestamp	timestamp	Data and time when notification is sent	No	No	-
notification_statu s	Boolean	Status of notification. It's sent or failed.	No	No	-
read_status	Boolean	Notification read status	No	No	-

Table 3.15 Overview of Detection Log's entity Data Dictionary

Column	Data Type	Description	Primary Key	Foreign Key	Referenced table
detection_id	Integer	Detection unique identifier	Yes	No	-
sensor_id	Integer	Sensor unique identifier	No	Yes	Sensor
cameraData_id	Integer	Camera unique identifier	No	Yes	CameraData

timestamp	timestamp	Data and time when detection is logged	No	No	-
isDetected	Boolean	True when the model detected a monkey	No	No	-
confidence_level	Float	The model's detection confidence level	No	No	-

Table 3.16 Overview of Sensor's entity Data Dictionary

Column	Data Type	Description	Primary Key	Foreign Key	Referenced table
sensor_id	Integer	Sensor unique identifier	Yes	No	-
pir_id	Integer	Pir unique identifier	No	Yes	PIRSensor
us_id	Integer	Ultrasonic sensor unique identifier	No	Yes	UltrasonicSensor
timestamp	timestamp	Data and time when sensor is triggered	No	No	-
sensor_type	Varchar	Type of sensor, it can be pir or ultrasonic	No	No	-

Table 3.17 Overview of PIRSensor's entity Data Dictionary

Column	Data Type	Description	Primary Key	Foreign Key	Referenced table

pir_id	Integer	PIR unique identifier	Yes	No	-
duration_of_motion	Float	The duration of motion in seconds detected by the PIR	No	No	-

Table 3.18 Overview of UltrasonicSensor's entity Data Dictionary

Column	Data Type	Description	Primary Key	Foreign Key	Referenced table
us_id	Integer	Ultrasonic unique identifier	Yes	No	-
distance	Float	Distance in cm measured by the sensor	No	No	-

Table 3.19 Overview of CameraData's entity Data Dictionary

Column	Data Type	Description	Primary Key	Foreign Key	Referenced table
cameraData_id	Integer	camera unique identifier	Yes	No	-
image_path	Varchar	Relative path to the image store in the server internal storage	No	No	-

3.4.5 User Interface Design

3.4.5.1 Sitemap

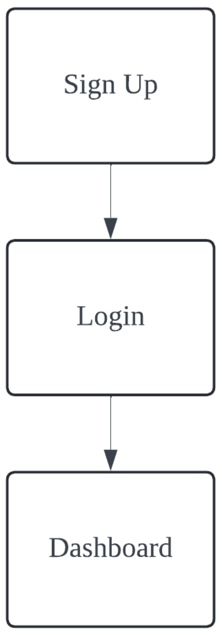


Figure 3.10 Sitemap for User Interface

The first page that the user will encounter is the sign-up page. From the sign-up page, user can navigate to login. After successful login user can then navigate to dashboard.

3.4.5.2 Sign Up Page

Figure 3.11, feature the sign-up page which displays email address box, username box, password box, and re-enter password box. Users need to fill all the boxes correctly to sign up.

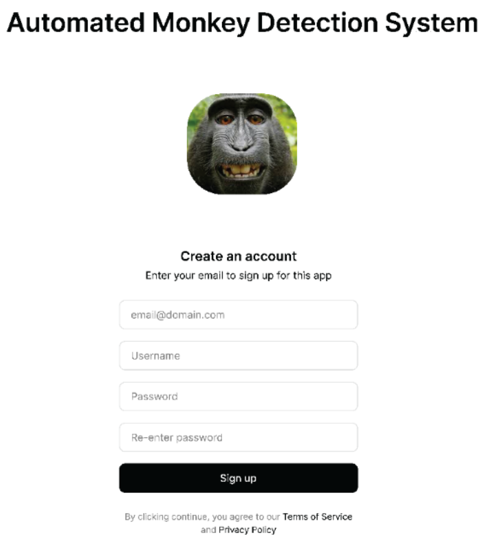

Automated Monkey Detection System
Create an account
Enter your email to sign up for this app
email@domain.com
Username
Password
Re-enter password
Sign up
By clicking continue, you agree to our Terms of Service and Privacy Policy

Figure 3.11 Screen capture of the sign-up page

3.4.5.3 Login Page

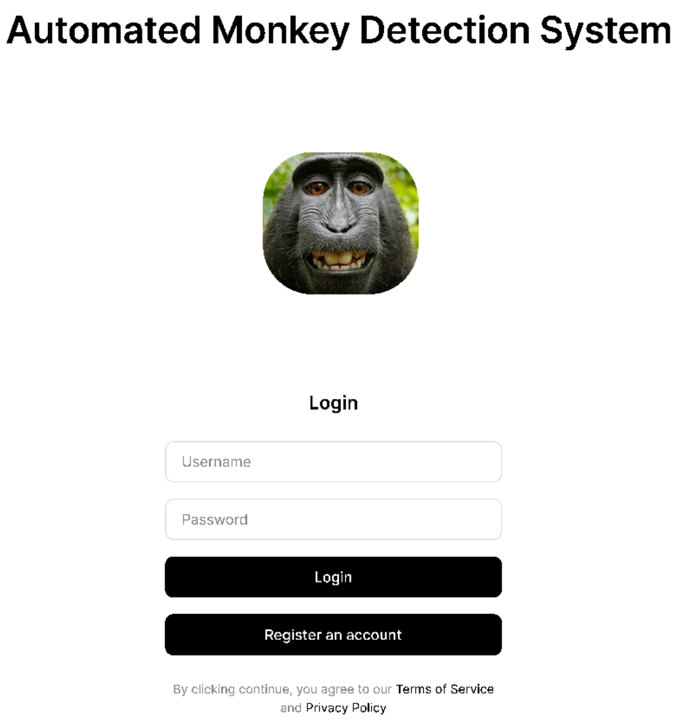
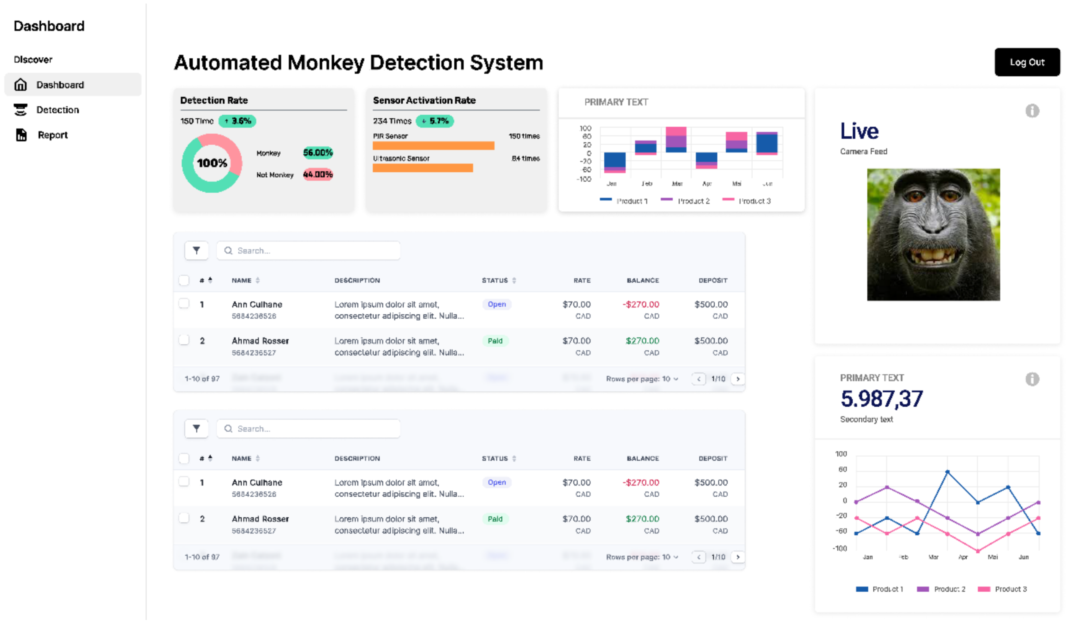

Automated Monkey Detection System
Login
Username
Password
Login
Register an account
By clicking continue, you agree to our Terms of Service and Privacy Policy

Figure 3.12 Screen capture of the login page

Figure 3.12, feature the login page which display username and password box that users need to fill in correctly to log in.

3.4.5.4 Dashboard Page

Figure 3.13, feature the dashboard page which will display various graph and table for system status, recent detections, and live camera feed.


Dashboard

Discover

- Dashboard
- Detection
- Report

Automated Monkey Detection System

Log Out

Detection Rate

150 Time ↑ 3.6%

100%

Monkey	56.00%
Not Monkey	44.00%

Sensor Activation Rate

234 Times ↓ 5.7%

PIR Sensor	150 times
Ultrasonic Sensor	84 times

Search...

#	NAME	DESCRIPTION	STATUS	RATE	BALANCE	DEPOSIT
1	Ann Culhane 5684236526	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla...	Open	$70.00 CAD	-$270.00 CAD	$500.00 CAD
2	Ahmad Rosser 5684236527	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla...	Paid	$70.00 CAD	$270.00 CAD	$500.00 CAD

1-10 of 97 Rows per page: 10 1/10

Search...

#	NAME	DESCRIPTION	STATUS	RATE	BALANCE	DEPOSIT
1	Ann Culhane 5684236526	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla...	Open	$70.00 CAD	-$270.00 CAD	$500.00 CAD
2	Ahmad Rosser 5684236527	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla...	Paid	$70.00 CAD	$270.00 CAD	$500.00 CAD

1-10 of 97 Rows per page: 10 1/10

Live

Camera Feed

PRIMARY TEXT

5.987,37

Secondary text

Figure 3.13 Screen capture of the dashboard page

3.5 Implementation

A proper implementation phase is planned to achieve the objectives of this project, which is to develop an Automated Monkey Detection System. This phase encompasses the project's outcome expectations and defines the criteria for the system's development. The phase will outline the project's front-end and back-end development implementation, aligning with the platform's usage, tools, and programming language. Visual Studio Code will be utilized as the primary IDE, and the Next.js framework will be used for the front-end integration.

For the integration of IoT, a Raspberry Pi Zero W will be employed to operate a PIR sensor for motion detection, an ultrasonic sensor for distance measurement, a horn for early alerts, and a 120-degree camera for capturing images of monkeys. Python will control all the sensors, manage the live feed, and handle all IoT-related tasks. YOLOv8n will train the model using a Google Open Image Dataset v4 dataset to detect monkeys. If a monkey is detected, the Raspberry Pi Zero W will send the logs to a Raspberry Pi 4 Model B for inference with the trained model. Upon confirming a monkey detection, an SMS, email, or Telegram bot-message will be sent to the user, and the horn will loudly alert the user of the surroundings.

The Dashboard page that displays recent detections, details report and a live feed from the camera.

3.6 Testing

This section covers functional testing, and system testing, each of which serves a specific purpose in validating different aspects of the system.

3.6.1 Test Planning

Section 3.6.1 outlines test planning for front-end and back-end components, covering features such as account registration, login validation, and report exporting using Google Chrome. Testing uses use case techniques and includes functional, and system tests.

3.6.1.1 Test Components

Components list covered in front end includes:

- Login.js
- Signup.js
- Dashboard.js

Components list covered in back end includes:

- Main.py

3.6.1.2 Features to be tested

Features to be tested for the front end includes:

- Register account
- Form Validation
- Login
- Filtering Table for Specific Date of Detection
- Export Excel from Dashboard Page
- Logout

3.6.1.3 Strategy

The test will be conducted for two framework that is front end and back end. For the front end, the test will be carried with Google Chrome version 126.0.6478.127 (Official Build) (64-bit). Use case testing techniques will be use.

3.6.1.4 Result Classification

When testing is finished and the desired result is obtained, the item is considered a pass. If there was a difference between the expected and the actual result, or if the standards weren't met, an item is deemed to have failed.

3.6.1.5 Test Preparation

Test plan creation, unit testing and system testing are the main objectives for this test session. Google Chrome version 126.0.6478.127 (Official Build) (64-bit) needs to be installed on the local computer before the unit and integration tests are run.

3.6.2 Test Cases

Front-end test cases verify that user interface elements function correctly and display properly. Back-end test cases ensure server-side processes handle requests and data accurately, ensuring seamless application operation.

3.6.2.1 Front-end Unit Test

Table 3.20 Front-end test case for Register Account

Test Form Number	1				
Test Designed By	Mohamad Rafiq bin Mohamad Sofi				
Test Perform By					
Test Designed Date	22/6/2024				
Test Perform Date					
Test Case					
Name	Register Account				
Pre-Condition	User is in sign up page				
Dependencies	PostgreSQL, Signup.js				
Test Activities					
Summary	Test Step	Test Data	Expected Result	Actual Result	Status
Register Account	1. Enter email 2. Enter username 3. Enter Password 4. Re-enter password	1.rafiqbinsofi@gmail.com 2.rafiqsofi 3.123 4.123	"Registered successfully" message will pop-up and user will be directed to login page.		

Table 3.21 Front-end test case for Form Validation

Test Form Number	2
Test Designed By	Mohamad Rafiq bin Mohamad Sofi
Test Perform By	

Test Designed Date	22/6/2024				
Test Perform Date					
Test Case					
Name	Form Validation				
Pre-Condition	User is in login page				
Dependencies	PostgreSQL, Login.js				
Test Activities					
Summary	Test Step	Test Data	Expected Result	Actual Result	Status
Form Validation	1. Enter username 2. Enter Password by purposefully make it wrong	1.rafiqsofi 2.1234	"Username or password are wrong!" message will pop-up and password field empty out.		

Table 3.22 Front-end test case for Login

Test Form Number	3				
Test Designed By	Mohamad Rafiq bin Mohamad Sofi				
Test Perform By					
Test Designed Date	22/6/2024				
Test Perform Date					
Test Case					
Name	Login				
Pre-Condition	User is in login page				
Dependencies	PostgreSQL, Login.js				
Test Activities					
Summary	Test Step	Test Data	Expected Result	Actual Result	Status

Login	1. Enter username 2. Enter Password	1.rafiqsofi 2.123	“Successfully Log in!” message will pop-up and user will be directed to dashboard page		

Table 3.23 Front-end test case for Filtering Table for Specific Date of Detection

Test Form Number	4				
Test Designed By	Mohamad Rafiq bin Mohamad Sofi				
Test Perform By					
Test Designed Date	22/6/2024				
Test Perform Date					
Test Case					
Name	Filtering Table for Specific Date of Detection				
Pre-Condition	User is in dashboard page				
Dependencies	PostgreSQL, Dashboard.js				
Test Activities					
Summary	Test Step	Test Data	Expected Result	Actual Result	Status
Filtering Table for Specific Date of Detection	1. Locate Search on table detection. 2. Enter date with format (dd/mm/yyyy) into Search.	1. 08/07/2024	Table will only display selected date.		

Table 3.24 Front-end test case for Exporting Excel from Report Page

Test Form Number	5
Test Designed By	Mohamad Rafiq bin Mohamad Sofi

Test Perform By					
Test Designed Date	22/6/2024				
Test Perform Date					
Test Case					
Name	Exporting Excel from Dashboard Page				
Pre-Condition	User is in dashboard page				
Dependencies	PostgreSQL, Dashboard.js				
Test Activities					
Summary	Test Step	Test Data	Expected Result	Actual Result	Status
Exporting Excel from Dashboard page	1. Locate Excel button on top of detection table. 2. Click the button. 3. Wait for save as file prompt to appear. 4. Save file as test.xlsx	-	User will be able to download excel file.		

Table 3.25 Front-end test case for Logout

Test Form Number	6
Test Designed By	Mohamad Rafiq bin Mohamad Sofi
Test Perform By	
Test Designed Date	22/6/2024
Test Perform Date	
Test Case	
Name	Logout
Pre-Condition	User is in any page
Dependencies	PostgreSQL, Report.js

Test Activities					
Summary	Test Step	Test Data	Expected Result	Actual Result	Status
Logout	1. Locate Log Out button top right of the screen. 2. Click the button.	-	“Successfully Log Out!” message will pop-up and user will be directed to login page		

3.6.2.2 Back-end Unit Test

Table 3.26 Back-end test case for Capturing Motion

Test Form Number	1			
Test Designed By	Mohamad Rafiq bin Mohamad Sofi			
Test Perform By				
Test Designed Date	22/6/2024			
Test Perform Date				
Test Case				
Name	Capturing Motion			
Pre-Condition	Technician is using unittest and using python version 3.12.3			
Dependencies	Main.py, unittest (Python Unit Testing Framework)			
Test Activities				
Summary	Function	Expected Result	Actual Result	Status

Capturing Motion	@patch('pir_sensor.GPIO')
	def test_motion_detected(self, mock_gpio):
	mock_gpio.input.return_value = mock_gpio.HIGH
	sensor = PIRSensor(pin=4)
	self.assertTrue(sensor.is_motion_detected())
	mock_gpio.setmode.assert_called_once_with(mock_gpio.BCM)
		1. test_motion_detected should return True when mock_gpio.input.return_value is set to HIGH.
		2. test_no_motion_detected should return False when		

	mock_gpio.setup.assert_called_once_with(4, mock_gpio.IN)
	mock_gpio.input.assert_called_once_with(4)
	
	@patch('pir_sensor.GPIO')
	def test_no_motion_detected(self, mock_gpio):
	mock_gpio.input.return_value = mock_gpio.LOW
	sensor = PIRSensor(pin=4)
	self.assertFalse(sensor.is_motion_detected())
	mock_gpio.setmode.assert_called_once_with(mock_gpio.BCM)
	mock_gpio.setup.assert_called_once_with(4, mock_gpio.IN)
	mock_gpio.input.assert_called_once_with(4)
		mock_gpio.input.return_value is set to LOW.		

Table 3.27 Back-end test case for Measuring Distance

Test Form Number	2			
Test Designed By	Mohamad Rafiq bin Mohamad Sofi			
Test Perform By				
Test Designed Date	22/6/2024			
Test Perform Date				
Test Case				
Name	Measuring Distance			
Pre-Condition	Technician is using unittest and using python version 3.12.3			
Dependencies	Main.py, unittest (Python Unit Testing Framework)			
Test Activities				
Summary	Function	Expected Result	Actual Result	Status

Measuring Distance	@patch('hc_sr04_sensor.GPIO')
	@patch('hc_sr04_sensor.time')
	def test_measure_distance(self, mock_time, mock_gpio):
	sensor = HCSR04Sensor(trigger_pin=18, echo_pin=24)
	
	mock_gpio.input.side_effect = [0, 1, 1, 0]
	mock_time.time.side_effect = [1, 1, 1.0001, 1.0001]
	
	distance = sensor.measure_distance()
		1.test_measure_distance should return approximately 1.715 cm when the mocked time values are used.		

	self.assertAlmostEqual(distance, 1.715, places=3)
	
	mock_gpio.setmode.assert_called_once_with(mock_gpio.BCM)
	mock_gpio.setup.assert_any_call(18, mock_gpio.OUT)
	mock_gpio.setup.assert_any_call(24, mock_gpio.IN)
	mock_gpio.output.assert_any_call(18, True)
	mock_gpio.output.assert_any_call(18, False)			

Table 3.28 Back-end test case for Horn Activation

Test Form Number	3
Test Designed By	Mohamad Rafiq bin Mohamad Sofi
Test Perform By	
Test Designed Date	22/6/2024
Test Perform Date	
Test Case	
Name	Horn Activation
Pre-Condition	Technician is using unittest and using python version 3.12.3
Dependencies	Main.py, unittest (Python Unit Testing Framework)
Test Activities	

Summary	Function	Expected Result	Actual Result	Status

Horn Activation	@patch('relay_horn.GPIO')
	def test_activate_horn(self, mock_gpio):
	horn = RelayHorn(relay_pin=17)
	
	horn.activate_horn()
	
	mock_gpio.setmode.assert_called_once_with(mock_gpio.BCM)
	mock_gpio.setup.assert_called_once_with(17, mock_gpio.OUT)
	mock_gpio.output.assert_called_once_with(17, mock_gpio.HIGH)
		1.test_activate_horn should set the GPIO output to HIGH on the relay pin.
		2.test_deactivate_horn should set the GPIO output to LOW on the relay pin.		

	@patch('relay_horn.GPIO')
	def test_deactivate_horn(self, mock_gpio):
	horn = RelayHorn(relay_pin=17)
	
	horn.deactivate_horn()
	
	mock_gpio.setmode.assert_called_once_with(mock_gpio.BCM)
	mock_gpio.setup.assert_called_once_with(17, mock_gpio.OUT)
	mock_gpio.output.assert_called_once_with(17, mock_gpio.LOW)			

Table 3.29 Back-end test case for Camera Feed

Test Form Number	4			
Test Designed By	Mohamad Rafiq bin Mohamad Sofi			
Test Perform By				
Test Designed Date	22/6/2024			
Test Perform Date				
Test Case				
Name	Camera Feed			
Pre-Condition	Technician is using unittest and using python version 3.12.3			
Dependencies	Main.py, unittest (Python Unit Testing Framework)			
Test Activities				
Summary	Function	Expected Result	Actual Result	Status

Camera Feed	@patch('camera_feed.PiCamera')
	def test_start_preview(self, mock_camera):
	camera_feed = CameraFeed()
	
	camera_feed.start_preview()
	
	mock_camera().start_preview.assert_called_once()
		1.test_start_preview should call start_preview on the camera instance.
		2.test_capture_image should call capture with the output file name.		

	@patch('camera_feed.PiCamera')
	def test_capture_image(self, mock_camera):
	camera_feed = CameraFeed()
	output = "test_image.jpg"
	
	camera_feed.capture_image(output)
	
	mock_camera().capture.assert_called_once_with(output)
	
	@patch('camera_feed.PiCamera')
	def test_stop_preview(self, mock_camera):
	camera_feed = CameraFeed()
	
	camera_feed.stop_preview()
	
	mock_camera().stop_preview.assert_called_once()
	mock_camera().close.assert_called_once()
		3.test_stop_preview should call stop_preview and close on the camera instance.		

Table 3.30 Back-end test case for Detection Logging

Test Form Number	5
Test Designed By	Mohamad Rafiq bin Mohamad Sofi
Test Perform By	
Test Designed Date	22/6/2024
Test Perform Date	
Test Case	
Name	Detection Logging
Pre-Condition	Technician is using unittest and using python version 3.12.3
Dependencies	Main.py, unittest (Python Unit Testing Framework)
Test Activities	

Summary	Function	Expected Result	Actual Result	Status
Detection Logging	@patch('detection_logging.psycopg2.connect') def test_log_pir_data(self, mock_connect):	1.test_log_pir_data should insert PIR		

	mock_conn = MagicMock()
	mock_cursor = MagicMock()
	mock_connect.return_value = mock_conn
	mock_conn.cursor.return_value = mock_cursor
	mock_cursor.fetchone.return_value = [1]
	
	logger = DetectionLogger(dbname='test_db', user='test_user', password='test_password')
	
	duration_of_motion = 5
	timestamp = '2024-06-27 12:00:00'
	sensor_id = logger.log_pir_data(duration_of_motion, timestamp)
	self.assertEqual(sensor_id, 1)
	insert_query_pir = 'INSERT INTO PIRSensor (duration_of_motion) VALUES (%s) RETURNING pir_id'
	insert_query_sensor = 'INSERT INTO Sensor (pir_id, timestamp, sensor_type) VALUES (%s, %s, %s) RETURNING sensor_id'
	mock_cursor.execute.assert_any_call(insert_query_pir, (duration_of_motion,))
	mock_cursor.execute.assert_any_call(insert_query_sensor, (1, timestamp, 'PIR'))
	mock_conn.commit.assert_called()
	
	@patch('detection_logging.psycopg2.connect')
	def test_log_us_data(self, mock_connect):
	mock_conn = MagicMock()
	mock_cursor = MagicMock()
	mock_connect.return_value = mock_conn
	mock_conn.cursor.return_value = mock_cursor
	mock_cursor.fetchone.return_value = [1]
	
	logger = DetectionLogger(dbname='test_db', user='test_user', password='test_password')
	
	distance = 100
	timestamp = '2024-06-27 12:00:00'
	sensor_id = logger.log_us_data(distance, timestamp)
	self.assertEqual(sensor_id, 1)
		sensor data and return the sensor ID.
		2.test_log_us_data should insert Ultrasonic sensor data and return the sensor ID.
		3.test_log_camera_data should insert camera data and return the camera data ID.
		4.test_log_detection should insert detection data and return the detection ID.
		5.test_close should close the database connection and cursor.		

	insert_query_us = 'INSERT INTO UltrasonicSensor (distance) VALUES (%s) RETURNING us_id'
	insert_query_sensor = 'INSERT INTO Sensor (us_id, timestamp, sensor_type) VALUES (%s, %s, %s) RETURNING sensor_id'
	mock_cursor.execute.assert_any_call(insert_query_us, (distance,))
	mock_cursor.execute.assert_any_call(insert_query_sensor, (1, timestamp, 'US'))
	mock_conn.commit.assert_called()
	
	@patch('detection_logging.psycopg2.connect')
	def test_log_camera_data(self, mock_connect):
	mock_conn = MagicMock()
	mock_cursor = MagicMock()
	mock_connect.return_value = mock_conn
	mock_conn.cursor.return_value = mock_cursor
	mock_cursor.fetchone.return_value = [1]
	
	logger = DetectionLogger(dbname='test_db', user='test_user', password='test_password')
	
	image_path = '/path/to/image.jpg'
	cameraData_id = logger.log_camera_data(image_path)
	self.assertEqual(cameraData_id, 1)
	insert_query = 'INSERT INTO CameraData (image_path) VALUES (%s) RETURNING cameraData_id'
	mock_cursor.execute.assert_called_once_with(insert_query, (image_path,))
	mock_conn.commit.assert_called()
	
	@patch('detection_logging.psycopg2.connect')
	def test_log_detection(self, mock_connect):
	mock_conn = MagicMock()
	mock_cursor = MagicMock()
	mock_connect.return_value = mock_conn
	mock_conn.cursor.return_value = mock_cursor
	mock_cursor.fetchone.return_value = [1]			

	logger = DetectionLogger(dbname='test_db', user='test_user', password='test_password')
	
	sensor_id = 1
	cameraData_id = 1
	timestamp = '2024-06-27 12:00:00'
	isDetected = True
	confidence_level = 0.95
	detection_id = logger.log_detection(sensor_id, cameraData_id, timestamp, isDetected, confidence_level)
	self.assertEqual(detection_id, 1)
	insert_query = '''INSERT INTO DetectionLog (sensor_id, cameraData_id, timestamp, isDetected, confidence_level) VALUES (%s, %s, %s, %s, %s) RETURNING detection_id'''
	mock_cursor.execute.assert_called_once_with(insert_query, (sensor_id, cameraData_id, timestamp, isDetected, confidence_level))
	mock_conn.commit.assert_called()
	
	@patch('detection_logging.psycopg2.connect')
	def test_close(self, mock_connect):
	mock_conn = MagicMock()
	mock_cursor = MagicMock()
	mock_connect.return_value = mock_conn
	mock_conn.cursor.return_value = mock_cursor
	
	logger = DetectionLogger(dbname='test_db', user='test_user', password='test_password')
	logger.close()
	mock_cursor.close.assert_called_once()
	mock_conn.close.assert_called_once()			

3.6.3 System Test

To validate the end-to-end functionality of the system in a real-world scenario and ensures the entire system works as intended and meets the user's needs, validating the complete workflow from motion detection to alert (Notifications).

Table 3.31 Overview of System Test

Name	Pass/Fail
Motion Detection	
Distance Measurement	
Horn	
Camera Feed	
Monkey Detection	
User Interface	
Alert (Notifications)	

References

AFP. (2022, July 25). Marauding monkeys injure 42 in Japanese city. *New Straits Times*. https://www.nst.com.my/world/world/2022/07/816334/marauding-monkeys-injure-42-japanese-city

Aggarwal, R., & Das, M. L. (2012). RFID security in the context of "internet of things." *Proceedings of the First International Conference on Security of Internet of Things*, 51–56. https://doi.org/10.1145/2490428.2490435

Bezdan T., B. D. N. (2019). Convolutional Neural Network Layers and Architectures. *Sinteza 2019 - International Scientific Conference on Information Technology and Data Related Research*, 445–451. https://doi.org/10.15308/Sinteza-2019-445-451

Chen, L., Li, S., Bai, Q., Yang, J., Jiang, S., & Miao, Y. (2021). Review of image classification algorithms based on convolutional neural networks. In *Remote Sensing* (Vol. 13, Issue 22). MDPI. https://doi.org/10.3390/rs13224712

Feng, C., Zhong, Y., Gao, Y., Scott, M. R., & Huang, W. (2021). *TOOD Task-aligned One-stage Object Detection*. httpsarxiv.orgabs2108.07755

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., & Chen, T. (2018). Recent advances in convolutional neural networks. *Pattern Recognition*, *77*, 354–377. https://doi.org/https://doi.org/10.1016/j.patcog.2017.10.013

Gupta, S., Banga, J., Dabas, S., & Bhatia, Dr. M. K. (2022). A Comprehensive Study of Software Development Life Cycle Models. *International Journal for Research in Applied Science and Engineering Technology*, *10*(12), 354–358. https://doi.org/10.22214/ijraset.2022.47868

Halvorsen, H.-P. (2020). *Software Development A Practical Approach!* https://www.halvorsen.blog

He, K., Zhang, X., Ren, S., & Sun, J. (2014). Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. In *Lecture Notes in Computer Science* (pp. 346–361). Springer International Publishing. https://doi.org/10.1007/978-3-319-10578-9_23

He, K., Zhang, X., Ren, S., & Sun, J. (2016, June). Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Ilic, I., Zivanovic Macuzic, I., & Ilic, M. (2022). Global Outbreak of Human Monkeypox in 2022: Update of Epidemiology. *Tropical Medicine and Infectious Disease*, *7*(10). https://doi.org/10.3390/tropicalmed7100264

Jocher, G., Chaurasia, A., & Qiu, J. (2023). *Ultralytics YOLO*. https://github.com/ultralytics/ultralytics

Knipe, H., & O’Gorman, P. (2016). Distance measurement. In *Radiopaedia.org*. Radiopaedia.org. https://doi.org/10.53347/rID-46916

Koushik, J. (2016). *Understanding Convolutional Neural Networks*. https://arxiv.org/abs/1605.09081

Latha, N. A., Murthy, B. R., & Kumar, K. B. (2016). Distance sensing with ultrasonic sensor and Arduino. *International Journal of Advance Research, Ideas and Innovations in Technology*, *2*(5), 1–5.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature, 521*(7553), 436–444. https://doi.org/10.1038/nature14539

Li, S., Zhang, H., & Xu, F. (2023). Intelligent Detection Method for Wildlife Based on Deep Learning. *Sensors*, *23*(24). https://doi.org/10.3390/s23249669

Li, X., Wang, W., Wu, L., Chen, S., Hu, X., Li, J., Tang, J., & Yang, J. (2020). Generalized Focal Loss: Learning Qualified and Distributed Bounding Boxes for Dense Object Detection. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, & H. Lin (Eds.), *Advances in Neural Information Processing Systems* (Vol. 33, pp. 21002–21012). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2020/file/f0bda020d2470f2e74990a07a607ebd9-Paper.pdf

Lim, T., Loke, V., Solana Mena, A., Pura, P., Angah, R., Tan, A., & Campos-Arceiz, A. (2017). Mapping the Distribution of People, Elephants, and Human-Elephant Conflict in Temengor Forest Complex, Peninsular Malaysia. *Malayan Nature Journal*.

Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., & Dollár, P. (2015). *Microsoft COCO: Common Objects in Context*. https://arxiv.org/abs/1405.0312

Liu, P., Nguang, S.-K., & Partridge, A. (2016). Occupancy Inference Using Pyroelectric Infrared Sensors Through Hidden Markov Models. *IEEE Sensors Journal*, *16*(4), 1062–1068. https://doi.org/10.1109/JSEN.2015.2496154

Liu, S., Qi, L., Qin, H., Shi, J., & Jia, J. (2018). *Path Aggregation Network for Instance Segmentation*. https://arxiv.org/abs/1803.01534

Lobachev, I., & Cretu, E. (2016). Smart sensor network for smart buildings. *2016 IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 1–7. https://doi.org/10.1109/IEMCON.2016.7746273

Maas, A. L., Hannun, A. Y., Ng, A. Y., & others. (2013). Rectifier nonlinearities improve neural network acoustic models. *Proc. Icml*, *30*(1), 3.

Madakam, S., Ramaswamy, R., & Tripathi, S. (2015). Internet of Things (IoT): A Literature Review. *Journal of Computer and Communications, 3*, 164–173. https://doi.org/10.4236/jcc.2015.35021

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 807–814.

Okesola, O. J., Adebiyi, A. A., Owoade, A. A., Adeaga, O., Adeyemi, O., & Odun-Ayo, I. (2020). *Software Requirement in Iterative SDLC Model* (pp. 26–34). https://doi.org/10.1007/978-3-030-51965-0_2

O'Shea, K., & Nash, R. (2015). An Introduction to Convolutional Neural Networks. *CoRR, abs/1511.08458*. http://arxiv.org/abs/1511.08458

Pavaloaia, V. (2012). Methodological Approaches to Computer Modelling Possibilities in Financial Analysis. *Annals of the Alexandru Ioan Cuza University - Economics, 59*. https://doi.org/10.2478/v10316-012-0026-5

Rather, M. A., & Bhatnagar, M. V. (2015). A comparative study of software development life cycle models. *International Journal of Application or Innovation in Engineering & Management (IJAIEM), 4*(10), 23–29.

Redmon, J., & Farhadi, A. (2018). *YOLOv3: An Incremental Improvement*. https://arxiv.org/abs/1804.02767

Rose Sadili, D. A. (2016). *UNIVERSITI PUTRA MALAYSIA HUMAN-MACAQUE CONFLICT BETWEEN TOURISTS AND LONGTAILED MACAQUES IN KANCHING RECREATIONAL FOREST, RAWANG, SELANGOR, MALAYSIA.*

Schell, C. J., Stanton, L. A., Young, J. K., Angeloni, L. M., Lambert, J. E., Breck, S. W., & Murray, M. H. (2021). The evolutionary consequences of human–

wildlife conflict in cities. *Evolutionary Applications*, *14*(1), 178–197. https://doi.org/10.1111/eva.13131

Taye, M. M. (2023). Theoretical Understanding of Convolutional Neural Network: Concepts, Architectures, Applications, Future Directions. In *Computation* (Vol. 11, Issue 3). MDPI. https://doi.org/10.3390/computation11030052

Xiao, B., Nguyen, M., & Yan, W. Q. (2024). Fruit ripeness identification using YOLOv8 model. *Multimedia Tools and Applications*, *83*(9), 28039–28056. https://doi.org/10.1007/s11042-023-16570-9

Xu, B., Wang, N., Chen, T., & Li, M. (2015). *Empirical Evaluation of Rectified Activations in Convolutional Network*. https://arxiv.org/abs/1505.00853

Yusop, H., Osman, N. A. W., Magintan, D., Julaihi, A. M., Ilias, R., Ramli, N. E., Rosli, N., Ithnin, H., Hamsi, N., Isa, S. N. A., Halim, H. R., Idris, M., Unau, J., Yahaya, N., Noh, N., Jamaludin, E. H., Mustafa, N., Mutalib, S. M., Paimin, R., … Mahayudin, S. A. (2018). *Perhilitan_LT2018*. https://www.wildlife.gov.my/images/document/penerbitan/laporantahunan/LT2018.pdf

Zeiler, M. D., & Fergus, R. (2013). *Stochastic Pooling for Regularization of Deep Convolutional Neural Networks*. https://arxiv.org/abs/1301.3557

Zhang, Q. (2018). *Convolutional Neural Networks*.

Zhang, Y., Guo, Z., Wu, J., Tian, Y., Tang, H., & Guo, X. (2022). Real-Time Vehicle Detection Based on Improved YOLO v5. *Sustainability*, *14*(19). https://doi.org/10.3390/su141912274

Zheng, Z., Wang, P., Liu, W., Li, J., Ye, R., & Ren, D. (2020). Distance-IoU loss: Faster and better learning for bounding box regression. *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*(07), 12993–13000.