Yun Sang Wong

Intelligent Object Recognition for Temperature-Based Fan Control

Metropolia University of Applied Sciences

Bachelor of Engineering

Electronics

Bachelor’s Thesis

28 November 2024

Abstract

Author:	Yun Sang Wong
Title:	Intelligent Object Recognition for Temperature-Based Fan Control
Number of Pages:	60 pages + 3 appendices
Date:	28 November 2024
Degree:	Bachelor of Engineering
Degree Programme:	Degree Program in Electronics
Professional Major:	Electronics
Supervisors:	Tuan Nguyen, Senior Lecturer
	Anssi Ikonen, Senior Lecturer

This thesis presents the development and implementation of intelligent object recognition on a temperature-based fan control system, aimed at advancing automation through more thoughtful and adaptive solutions. The proposed system is designed to recognize cats and humans using a machine learning model deployed on an edge device, contributing to smart home technology, human and pet care products, and environmental control systems focusing on sustainability and energy efficiency.

The project involved data collection, labelling, model training, and deployment on Raspberry Pi 3, along with the design of a temperature-based fan control system, a sensor platform printed circuit board design, 3D enclosure design. The image data for the model training was captured using a smartphone camera with labelling performed in Labellmg. Model training was conducted on Google Colab, using a pre-trained SSD-MobileNet-v2 (FP32) TensorFlow model. The trained model achieved a mean average precision (mAP) score of 80.83% across various Intersections over Union (IoU) thresholds and 97.55% at a 0.5 IoU threshold. The model was then deployed on Raspberry Pi 3. The temperature-based fan control system, which includes a DHT11 sensor, a DC fan, and an LCD, successfully adjusted fan speed based on user input and temperature data.

As a result, the integrated system successfully detected objects which were cats and humans, performing tasks based on the detected objects and temperature status. The system’s performance met expectations.

Keywords: machine learning, deep learning, automation, temperature-based fan control, object recognition

---

Contents

List of Abbreviations

AI:	Artificial Intelligence.
AP:	Average Precision.
ANN:	Artificial Neural Network.
BGR:	Blue, Green, Red.
BPNs:	Backpropagation Networks.
CNN:	Convolutional Neural Network.
CPU:	Central Processing Units.
CSI:	Camera Serial Interface.
CSV:	Comma-Separated Values.
DL:	Deep Learning.
FC Layer:	Fully Connected Layer.
FN:	False Negatives.
FP:	False Positives.
FPN:	Feature Pyramid Network.
FPS:	Frame Per Second.
GAP:	Global Average Pooling.
GMP:	Global Max Pooling.

GPIO Pins: General-Purpose Input/Output Pins.

GPUs: Graphical Processing Units.

IoU: Intersection over Union.

LCD: Liquid-crystal display.

mAP: Mean Average Precision.

ML: Machine Learning.

MLFFNs: Multi-Layer Feed Forward Networks.

NLP: Natural Language Processing.

OpenCV: Open Source Computer Library.

PWM: Pulse Width Modulation.

RAM: Random Access Memory.

ReLU: Rectified Linear Unit.

RGB: Red, Green, Blue.

SSD: Singel Shot MultiBox Detector.

Tach: Tachometer.

TN: True Positives.

TP: True Negatives.

TPUs: Tensor Processing Units.

VOC: Visual Object Classes.

1 Introduction

This chapter provides an overview of the project, detailing the background, research question, and project objectives.

1.1 Background

Artificial Intelligence (AI) has become a transformative force in recent years. It has the potential to bring substantial changes across all areas of life by providing new solutions. The usage of AI is not only in the new technologies, integrating AI into existing technologies is another significant matter. [1.]

On the other hand, as the population ages the labour shortage becomes a bigger issue in many countries such as Japan, South Korea, and Germany. But, at the same time, automation is rapidly developing to address these challenges in those countries. As many countries' populations are still aging quickly, automation is becoming increasingly important in many aspects. [2.]

A new concept of automation is born by the integration of AI technology: AI automation. While traditional automation allows machines to do repetitive tasks following the instructions made by humans, AI automation enables systems to achieve the tasks by making predictions based on experience and new data. Integrating AI into automation systems helps enhance the performance of the systems by improving them through learning and adapting. [3.]

Nowadays, automation is everywhere enhancing convenience in our daily lives, from automatic doors in shopping malls, to sensor-driven flushing systems and lighting systems in restrooms. However, because traditional automation relies on the instructions made by humans, it can sometimes make mistakes when the conditions are achieved in other ways. For instance, automatic doors may open simply when someone stands nearby, the toilet flushing system activates while people are using it and the lights may turn off when people are still present,

people must keep moving their bodies to keep the lights on. To avoid those issues and reduce the consumption of energy, integrating AI systems may be the solution to solve the problems in traditional automation.

In contrast to the labour population and the fertility rate, the pet population is still rising in many countries [4]. Figure 1 below represents the growth of the global pet industry. The bar graph in the image shows data on the pet industry. According to the data, the pet industry has been growing rapidly since 2022 to the future prediction of 2030. As the pet industry increases quickly in the future based on the prediction, describing that the needs of pets and pet owners also increase.

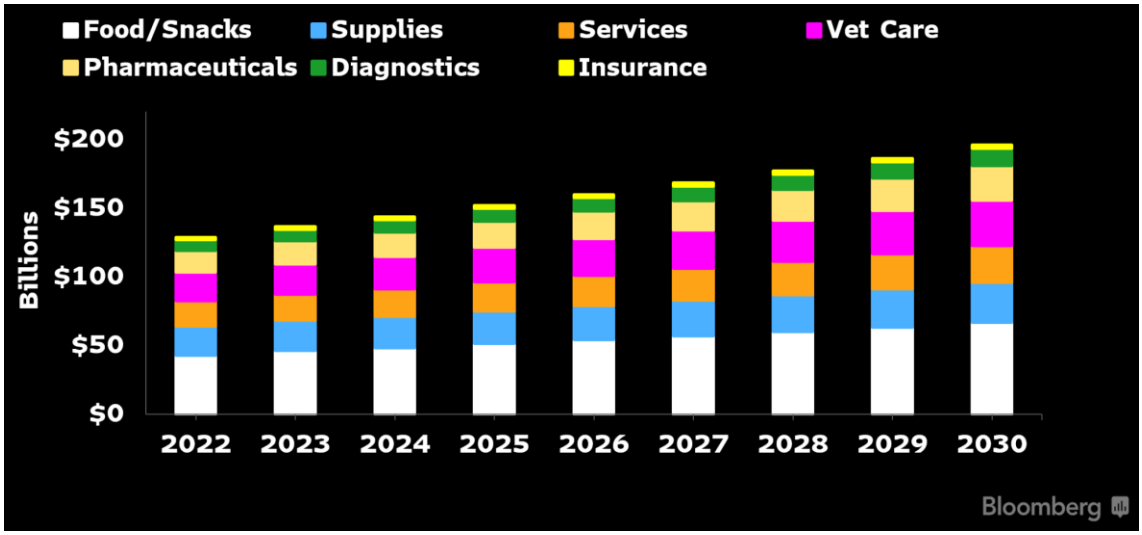

Figure 1. US Pet Industry May Approach $200 Billion by 2030 [5].

This project explored the potential of combining two ‘smart’ technologies: AI and automation to meet the needs of pets and their owners in response to the growing pet population. The goal was to enhance home automation technologies and discover new possibilities and insights.

1.2 State the Research Questions

This thesis seeks to answer the following research questions:

- How accurately can a customized model of object recognition system identify humans and cats?

- How well does the object recognition system work on Raspberry Pi 3?

- How effectively does the integrated system of object recognition and temperature-based fan control function on Raspberry Pi 3?

1.3 Project Objectives

A general automatic temperature controller primarily focuses on temperature as the sole factor [6]. It is an effective method for maintaining the temperature in a certain range. However, similar to other automatic products as mentioned in the background section, a standard automatic fan control system cannot detect whether the users or the objects are present.

This project expects that integrating an object recognition system can improve the temperature control system, allowing it to consider additional factors, such as the presence of pets, leading to more efficient home automation technology.

The object recognition system is trained by the Machine Learning (ML) method, to achieve the tasks of identifying cats and humans accurately and it is integrated into a temperature-based Fan control system to optimize the environmental conditions. When a human is identified, the system displays the current temperature on a Liquid-Crystal Display (LCD). When a cat is identified and the temperature exceeds a certain temperature set by the user, a fan is automatically activated to cool the cat. An air conditioner is an ideal device for the temperature control system, but in this project, a DC fan will be used as a substitute to simulate its functionality, therefore, the temperature control system is replaced by a temperature-based fan control. The main contribution of the thesis work is described below.

- Training a model for cat and human recognition based on image data.
- Deploying and testing the trained model on Raspberry Pi 3.
- Implementing a temperature-based fan control system with temperature sensing, fan control, and LCD monitoring.
- Designing and implementing a sensor platform circuit board and a 3D enclosure model, integrating components into the platform, and housing them within the enclosure.
- Integrating and testing the object recognition system together with the temperature-based fan control system in real cases.

2 Literature Review

This chapter discusses the theories that are essential for understanding the process of this project, including the definition of AI and its subsets, machine learning and Deep Learning (DL), with a particular focus on the Convolutional Neural Networks (CNNs), the primary method used in this project. Additionally, the principles and the functions of the object recognition system and temperature-based fan control system will also be covered.

2.1 Overview of Artificial Intelligence

AI is the technology that allows computers and machines to do tasks in ways like humans such as learning, problem-solving, decision-making, prediction, creativity, and autonomy [7].

ML is a part of AI that includes Deep Learning (DL) and involves using data and algorithms to allow AI to learn and get better at tasks, which means more accuracy such as human learning. Figure 2 below shows the relationship between AI, ML, and DL. [8.]

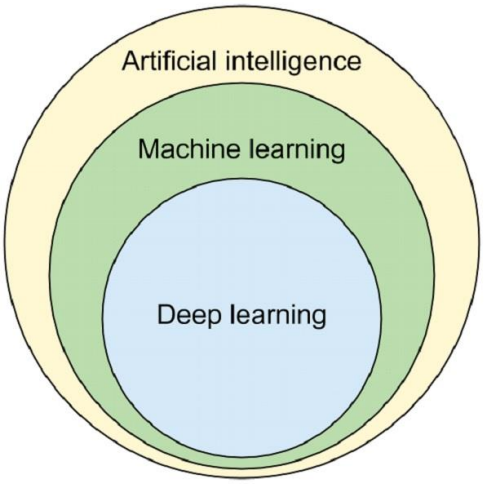

Figure 2. Hierarchy of AI, ML, and DL [9].

A well-trained model can identify patterns within raw data and make accurate predictions based on those insights. It can be applied in image recognition, speech recognition, Natural Language Processing (NLP), recommendation systems, fraud detection, and many other fields.

Deep learning is a subset of Machine learning that uses Artificial Neural Networks (ANNs) to learn from input data, simulating the structure and function of the human brain. [10.]

2.2 Artificial Neural Networks (ANNs)

Artificial Neural Networks (ANNs) are the foundational structures of deep learning consisting of layers and interconnected nodes, known as neurons, which give machines the ability to learn. In this chapter, the following section will explore the details of the structure and the components of ANNs and Multi-Layer Feed Forward Networks (MLFFNs).

2.2.1 Structure of ANNs

Figure 3 below illustrates the structure of ANNs. The small circles represent artificial neurons, also known as units. A neural network can consist of from a few to millions of units, which are organized into layers. Each of the layers is

connected by those units. The units in the input layer, called input units, are responsible for receiving information and data from the external environment. The number of neurons in the input layer usually equals the number of features in the input data. If the input data is an image, the pixels represent the features, meaning the number of neurons in the input layer equals the number of pixels in the image. [12.]

The output layer consists of output units. They produce the possible outcomes and results after learning, and eventually, the output of the results is fed into a logistic function that calculates its probability score. [12.]

The hidden layer is placed between the input and output layers, the input data from the input layer is then fed into this layer. Depending on the input data and the model, the hidden layer part can range from a single layer to multiple layers, they are together responsible for the process of learning. Each of the hidden layers consists of hidden units, which are usually more than the number of pixels in the image, and they can have different numbers of hidden units. The number of connections between units is called weights. It represents the strength of the connections between units. [12.]

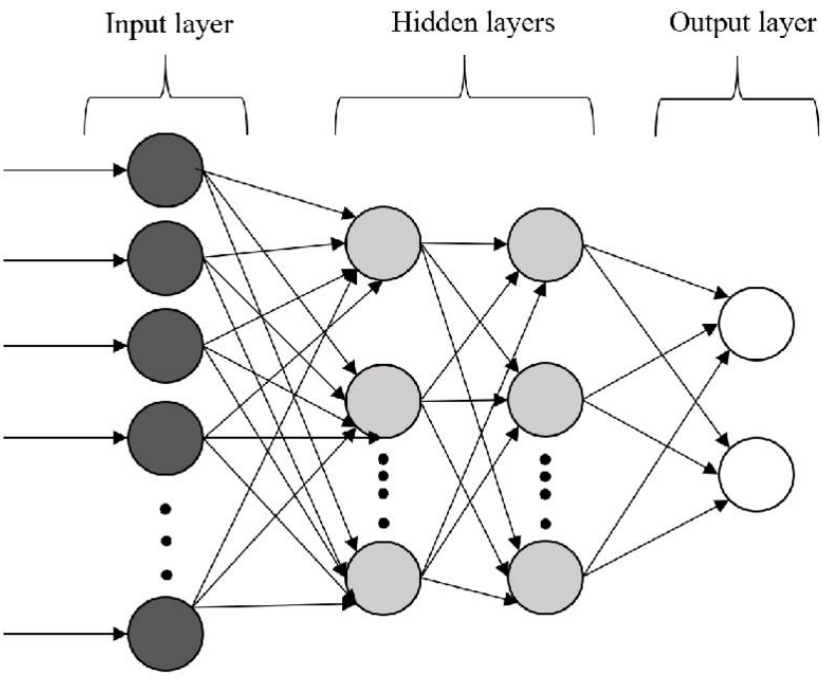

Figure 3. Neural network architecture [11].

2.2.2 Example of General Feed Forward Operation

Figure 4 shows an example of a fully connected neural network with 2 inputs, 2 hidden units, and 1 output. This section will discuss the details of the general feed-forward operation based on this image.

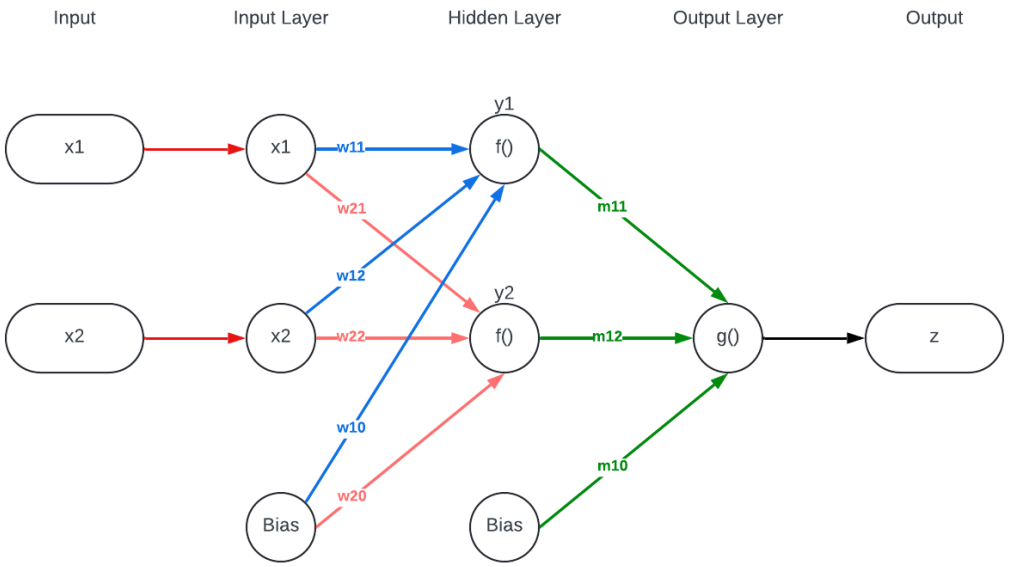

Figure 4. 2-2-1 Fully Connected Neural Networks [13].

Equation (1) below is the generalized equation of the output from hidden layer units. Additionally, 'i' in the equation refers to the number of units in the input layer, 'j' refers to the number of units in the hidden layer, and 'k' refers to the number of units in the output layer. [14.]

$$y_j = f\left(\sum_{i=1}^{n} w_{ji} * x_i + w_{j0}\right) \quad (1)$$

Based on the image above, x_1 and x_2 are the inputs of features fed into the input layer. Bias is a constant input in the input layer as well, and it is always equal to 1. The input units then connect to hidden units through the weights w_{11}, w_{21}, w_{12}, w_{22}, w_{10}, and w_{20}, meaning each input is multiplied by corresponding weights and fed into the hidden layer units. In the hidden layer, the hidden units apply an activation function f () to calculate the outputs from hidden layers y_1 and y_2. Equations (2) and (3) represent the calculations for y_1 and y_2. [14.]

$$y_1 = f(w_{11} * x_1 + w_{12} * x_2 + w_{10}) \quad (2)$$

$$y_2 = f(w_{21} * x_1 + w_{22} * x_2 + w_{20}) \quad (3)$$

Equation (4) below is the generalized equation of outputs from the output layer. The outputs from hidden layers y_1 and y_2 are then passed to the output layer by multiplying by corresponding weights, here m_{11}, m_{12}, m_{21}, and m_{22} represent the weights between the hidden layer and output layer. The output units in the output layer apply an activation function g () to calculate the outputs from the output layer z_k, where k is equal to 1 here. Equation (5) represents the calculations for z. [14.]

$$z_k = g\left(\sum_{j=1}^{n} m_{kj} * y_j + m_{k0}\right) \quad (4)$$

$$z = g(m_{11}y_1 + m_{12}y_2 + m_{10}) => \quad (5)$$

$$g(m11f(w11 * x1 + w12 * x2 + w10) + m12f(w21 * x1 + w22 * x2 + w20) + m10)$$

z is not only the output from the output layer but also the final output of the network, which represents the predicted values or classifications after processing the inputs through the network. The equations can be represented in matrix form as well. [14.]

2.2.3 Error Calculation and Weight Updates in Backpropagation Networks (BPNs)

After the feed-forward operation, commonly known as forward propagation, there is another important step in the ANNs, called backpropagation. If the neural network has only forward propagation without feedback, it might not be able to learn, meaning that it cannot correct the fault of the previous step by itself. Therefore, the operation of backpropagation is crucial, it improves the accuracy and decreases the error of the network, similar to how humans review and correct their mistakes while learning. [14.]

Before the process of the BPNs, there are two parameters are required, target value and learning rate. The target value ‘t’ is the expected value from the output of the model, it represents the aims of the model from prediction. The learning

rate ‘α’ is a hyperparameter that determines the adjustment of the weights during each step of the gradient descent. It decides the size of the updates made to the model according to the error calculation by the loss function and it has effects on the speed of the learning process of the model. [14.]

In this project, the cosine decay learning rate with the warm-up phase is applied, meaning that in the warm-up steps, the learning rate increases linearly from the warm-up learning rate to the base learning rate. After the warm-up steps, the learning rate decreases gradually to 0 according to the cosine function until the end of training. [14.]

First, the error between the output layer and hidden layer ‘e_k’ will be calculated. ‘k’ refers to the number of units in the output layer as mentioned in the before section. Equation (6) is the generalization of the error equation. [14.]

$$e_k = (t_k - z_k) * g'\left(\sum_{j=1}^{n} m_{ki} * y_i + m_{k0}\right) \quad (6)$$

The calculation is based on the same image mentioned above. Since ‘k’ equals 1, meaning that there is only an error between the output layer and the hidden layer, e, respectively. Equation (7) represents the calculation of error between the output layer and hidden layer according to the output z. [14.]

$$e = (t - z) * g'(m_{11}y_1 + m_{12}y_2 + m_{10}) \quad (7)$$

Next, the error between the hidden layer and the input layer will be calculated and represented as ‘E_j’ in the generalization of the error equation (8) below, where ‘j’ refers to the number of units in the output layer as mentioned in the before section [14].

$$E_j = e * m_{kj} * f'(y_j) \quad (8)$$

Based on the generalized equation and the image above, the calculations representing the errors of hidden units can be formulated as follows equation (9) and equation (10) [14].

$$E_1 = e * m_{11} f'(y_1) \quad (9)$$

$$E_2 = e * m_{12} f'(y_2) \quad (10)$$

This stage represents the final step in the BPNs. Following the error calculations, the next step is to update the weights. The calculations for updating the weights between the hidden layer and the output layer are as follows equations (11), (12), and (13). [14.]

$$m_{11}(new) = m_{11} + \alpha * e * y_1 \quad (11)$$

$$m_{12}(new) = m_{12} + \alpha * e * y_2 \quad (12)$$

$$m_{10}(new) = m_{10} + \alpha * e * 1 \quad (13)$$

The following equations (14), (15), and (16) are for updating the weights between the input layer and the hidden unit y_1 [14].

$$w_{11}(new) = w_{11} + \alpha * E_1 * x_1 \quad (14)$$

$$w_{12}(new) = w_{12} + \alpha * E_1 * x_2 \quad (15)$$

$$w_{10}(new) = w_{10} + \alpha * E_1 * 1 \quad (16)$$

The following equations (17), (18), and (19) are for updating the weights between the input layer and the hidden unit y_2 [14].

$$w_{21}(new) = w_{21} + \alpha * E_2 * x_1 \quad (17)$$

$$w_{22}(new) = w_{22} + \alpha * E_2 * x_2 \quad (18)$$

$$w_{20}(new) = w_{20} + \alpha * E_2 * 1 \quad (19)$$

2.2.4 Activation Functions in Neural Networks

In the topic of feed-forward operation and BPNs, activation functions denoted as ‘f ()’ and ‘g ()’, have appeared several times. The activation function is usually applied in the hidden layers and the output layer in ANNs as a mathematical function to avoid only linear relationships between inputs and outputs. It acts as a switch that determines whether to activate the units or not depending on its features. A suitable activation function allows the model to learn complex data effectively with non-linearity. This section discusses the activation functions, Rectified Linear Unit (ReLU) activation function, ReLU6, and Sigmoid activation function, which are applied to the project. [15.]

ReLU activation function is non-linear, or it can be considered that it is a piecewise linear function. The ReLU activation function is defined as equation (20) below. [15.]

$$f(x) = \max(0, x) \quad (20)$$

The slope change from 0 to a linear positive slope at a threshold, greater than 0, as Figure 5 displays below, meaning that if the inputs to the function are negative, the outputs will remain 0. Conversely, if the inputs are positive, the output of the function will increase linearly with a gradient of 1, passing positive inputs through unchanged. The outputs range from 0 to infinity. Because the ReLU activation function outputs 0 when inputs are negative, this action encourages sparse activations, meaning that many units remain inactive, relatively reducing the number of computations, leading to a reduction of memory used and a faster training process. [15.]

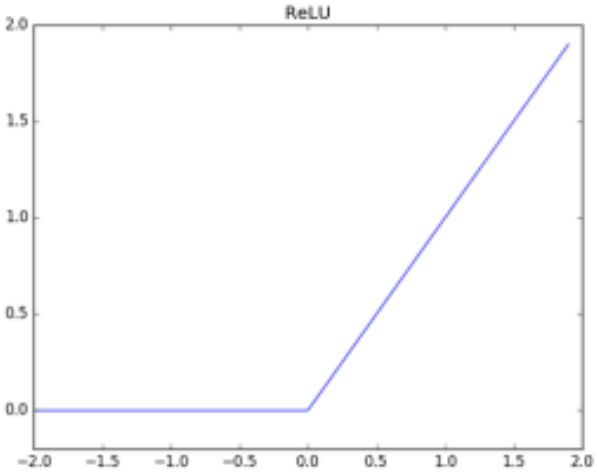

Figure 5. Graph of ReLU activation function [16].

Due to the relative ease of model training and its ability to achieve good performance, the ReLU activation function is widely used in hidden layers for many types of ANNs [18].

Although the ReLU activation function is an effective and helpful tool for deep learning, it still has some limitations. For instance, “Dying ReLU”. When the weighted sum of units is negative in the training progress, the outputs are 0 and the units assigned to it are inactive and stop learning. This can result in a lack of diversity in the trained model. [18.]

The reLU6 activation function is a modification of the ReLU activation function with an upper limit of 6, which is not in the standard ReLU activation function. It is defined as equation (21) below. [20.]

$$f(x) = \min\left(\max(0, x), 6\right) \tag{21}$$

The sigmoid activation function is a commonly used non-linear function as well. It is defined as equation (22) below. [15.]

$$f(x) = \frac{1}{(1+e^{-x})} \tag{22}$$

The outputs from the Sigmoid activation function are plotted in an ‘S’ shaped curve, starting from negative inputs to positive inputs as Figure 6 displays below. Since this function has a range of outputs between 0 and 1, it often applies in the output layer for binary classification problems. For instance, if an output from the function is greater than 0.5, the result can be predicted as 1, otherwise, it is predicted as 0. [15.]

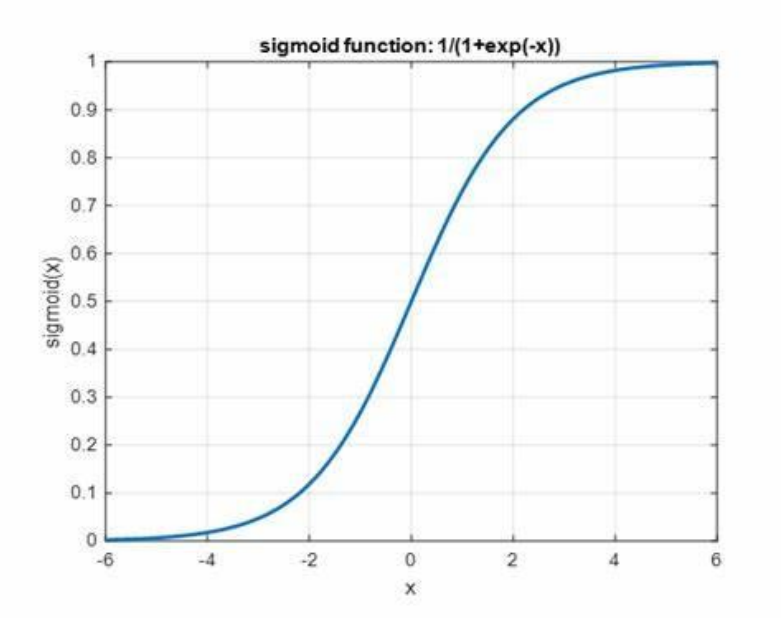

Figure 6. The shape of the sigmoid function [20].

While the sigmoid function is a powerful tool for model training, it has some disadvantages as well. “Vanishing gradient” is one of the problems that may be encountered during the model training progress with sigmoid function. The reason this happens is that it has a small range from 0 to 1, meaning that the gradient is close to 0, causing the updating operation of weights to be small or even stopping during the backpropagation. Eventually, it may slow down or stop the learning progress. [15.]

2.3 Convolutional Neural Network (CNN)

CNN is a type of ANN, commonly utilized for the tasks involving computer vision, such as object recognition and image classification using three-dimensional data

to achieve the model training. Additionally, Graphical Processing Units (GPUs) are needed for the model training. This section discusses the process of CNN and the MobileNetV2 architecture which are used in the project [21].

CNN is typically composed of 3 main types of layers, convolutional layer, pooling layer, and Fully Connected (FC) layer. Figure 7 below displays the simple CNN architecture [21].

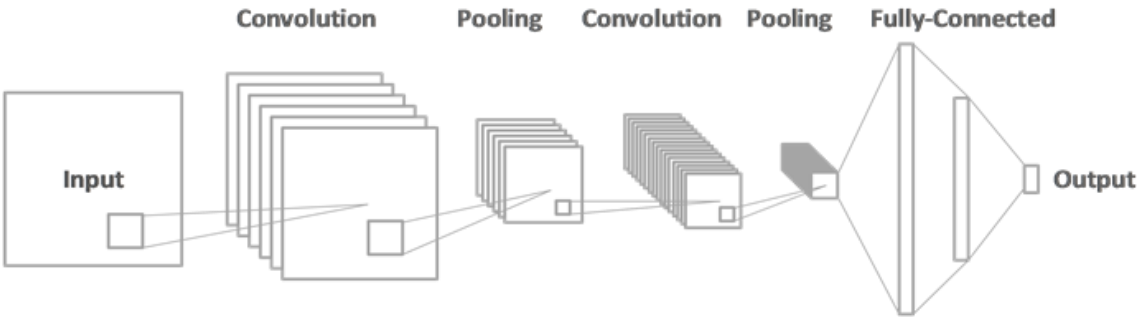

Figure 7. Flow diagram for robust feature extraction [22].

Similar to other types of ANN, the input layer is where the data is fed into the model. The next step after the input layer is the convolutional layer. Most of the computations in the training process are done in this layer, therefore, the convolutional layer is the core building block of a CNN. In this layer, the process of convolution is operated. Figure 8 below displays the process of convolution. First, the specified two-dimensional weights called filters, also known as kernel or feature detector, convolve across pixels from the input. A filter is usually built with a matrix where the number of rows and columns is specified by its designer, and the values in the matrix are set to random numbers. For instance, if a 3x3 matrix filter is set, it will move across each 3x3 block of pixels to check the feature from the input image and calculate the dot product between the filter and the input pixels. After the calculation, the dot product is transferred to an output array. This process is repeated until all pixels in the input image are checked. Eventually, the array of dot products forms a feature map, also known as a convoluted feature, and it is the output from this layer. Usually, there are several same-size filters applied for different specific patterns in the image such as the edges, corners,

textures, colour gradients, simple shapes, etc. These filters help CNN to recognize the features of the input image. [21.]

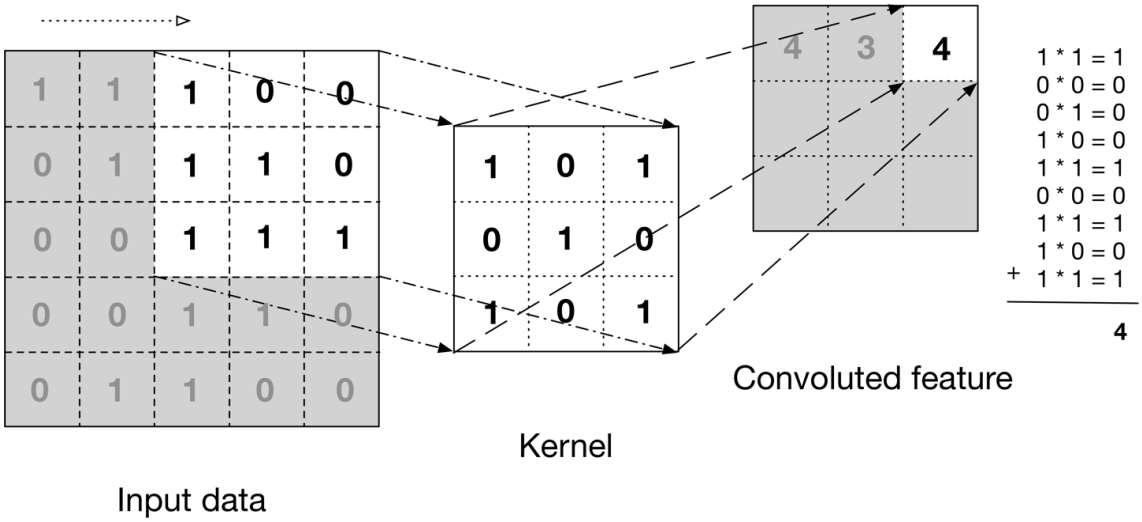

Figure 8. Example of a convolution operation [23].

The convolutional layer can be followed by additional convolutional layers. In this case, the feature map of the initial convolution layer is the input for the next convolution layer, and the following convolution layer will form a feature map of the input from the initial convolution layer and so on [21].

The size of the feature map depends on the size of the filter, stride, padding, and the size of the input image. Stride is the number of pixels the filter covers, also called step size. Filter covers more pixels of input with a higher stride and reduces the size of the feature map. Similarly, the larger size of the filter itself covers more input data at a time. It results in reducing the size of the output, which is a feature map. Padding is also one of the factors that affect the dimensions of the feature map. Padding is a process of adding extra border pixels to the input image. In the process of convolution, the output feature map is generally smaller in size compared to the input. Therefore, without padding, several convolution layers later, the output may be much smaller than the input, the spatial information may be lost, and the border pixels may be ignored during the convolution process. By applying padding, usually with zeros to add extra border pixels, the spatial size

can be controlled to prevent the issues mentioned above. In this case, the more padding added, the larger dimensions of the feature map are outputted. [21.]

Generally, each convolution operation follows a ReLU activation function, as mentioned in the previous section. This activation function allows the model to learn more complex patterns in the data by introducing non-linearity into the network, improving computational efficiency by increasing sparsity and avoiding the vanishing gradient problem. [21.]

The pooling layer usually follows the convolution later. Its purpose is to obtain the most significant features from the feature map by a two-dimensional filter, to reduce the dimension of the feature maps, improving the speed of the training process. The common types of pooling operation include max pooling, average pooling, and global pooling. [21.]

The max pooling extracts the maximum element within the area covered by the filter in the feature map. Figure 9 below is an example of the operation of the max pool with a 2x2 filter. In this example, 21 is the maximum element in the green zone, similarly, 12, 18, and 10 are the maximum elements respectively in the zone they belong to. The operation would be repeated until all data in the feature map has been covered. The max pooling operation eventually generates an updated version of the feature map containing the most significant features. In this case, a 4x4 matrix feature map became a 2x2 matrix feature map after the max pooling operation. Since the maximum pooling preserves the most significant features and ignores the less relative information, it is often used in tasks for object recognition. [21.]

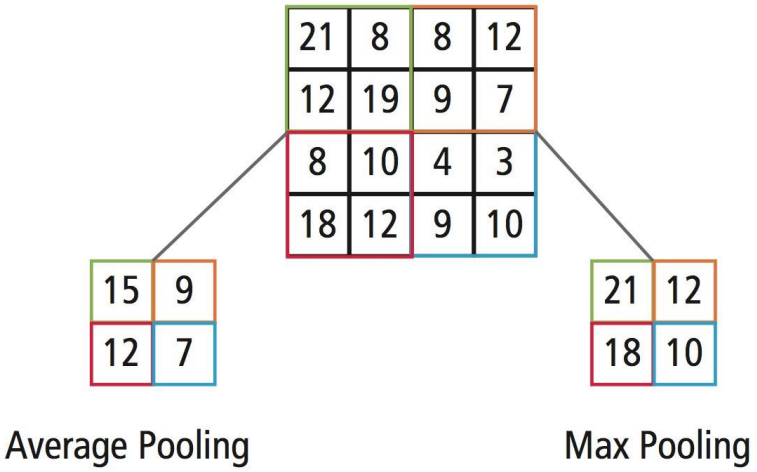

Figure 9. Example of a pooling operation with a stride length of 2 [23].

Average pooling is a similar operation to max pooling. The difference between them is that instead of extracting the maximum element from the region covered by the filter, it calculates the mean value of the elements and generates an updated feature map composed of mean values from each region covered by the filter. See the example in Figure 9 above. The new feature map conducted by average pooling contains the average values in the region they belong. From left to right, up to down, they are respectively 15, 9, 12, and 7. Compared to the output from max pooling, the output from average pooling contains more information, but less significant features. It is often utilized in the tasks for image segmentation and object detection. [21.]

Global pooling has two different types, global max pooling and average pooling. They are almost using the same method besides the size of the filter. Instead of utilizing a smaller size, 2x2 or 3x3 in the average pooling and max pooling, global pooling applies a same size filter of the input. By the same example above, since the input is a 4x4 matrix feature map, therefore the filter would be a 4x4 matrix filter as well. The output from Global Max Pooling (GMP) would be the maximum element of the input, which is 21 in this example, and the output from Global Average Pooling (GAP) would be the mean value of the input, which is 10.625. Global pooling significantly reduces the number of parameters in the network, but also increases the risks of losing important information. [24.]

The final layer in the CNN would be the fully connected layer. Before the output from the pooling layer moves to the fully connected layer, it would be flattened during the flattening operation in the fully connected input layer, meaning that the matrix would be transformed into a one-dimensional format vector. After flattening, the vector would be fed into the fully connected layer and usually ReLU activation function is applied in this layer. After the feature analysis, the model would do the first prediction of the label. The final prediction would be done in the fully connected output layer, generally, softmax or sigmoid would be applied to this layer. While CNN has high accuracy and the ability to learn complex features, it requires large amounts of input data and large memory for the edge devices. [21.]

2.4 Pre-trained Model: SSD MobileNetV2 FPNLite 320x320

A pre-trained TensorFlow model is necessary for the model training process, and SSD MobileNet V2 FPNLite 320x320 has been chosen for the project. The inputs are resized to 320x320 pixels before it is fed into the model. In this section, the other details of the chosen model will be discussed.

2.4.1 MobileNetV2

MobileNetV2 is a lightweight architecture of CNN designed for mobile and embedded devices, developed by Google [25]. The reason it is ideal for mobile and embedded devices is because the model is fast and small in size. Depth-wise separable convolutions and inverted residuals with linear bottlenecks are the key components determining the smaller size of the model.

Depth-wise separable convolution is a technique used to reduce the number of multiplications and increase the efficacy of the model. It splits a general standard convolution operation into two steps: depth-wise convolution and pointwise convolution. In depth-wise convolution, each input channel is convolved with a filter, therefore, the number of channels remains the same as the input, and all channels are stacked together as one stacking channel. The pointwise

convolution is the next step after the depth-wise convolution, which is also known as 1x1 convolution, meaning that a 1x1 filter is applied to each stacking channel. Eventually, the number of multiplications is reduced by this operation. [26.]

Inverted residuals are designed to improve the efficiency of networks. It is a reverse structure of the traditional residuals. The architecture of traditional residuals follows the wide-to-narrow-to-wide structure, meaning that the input consists of numerous channels. Reduce the number of channels in the 1x1 convolution with the ReLU activation function by reducing the number of filters. The operation of reducing the number of channels increases the efficiency of the 3x3 convolution with the ReLU activation function due to less input. Since the skip connection requires the same number of channels, after 3x3 convolution, another 1x1 convolution with ReLU activation function is applied to increase the number of channels. Skip connection, also known as shortcut connection, is the connection between the input and the output of the residual networks. This connection allows the input directly transferred to the output to avoid the vanishing gradient problem. [26.]

The architecture of inverted residuals with linear bottleneck follows the narrow-to-wide-to-narrow. It starts with an input that has fewer channels. In the next step, the number of channels is expended using 1x1 convolution to ensure there are enough features for the next step, a 3x3 depth-wise convolution with ReLU6 activation function. A 1x1 convolution with ReLU6 activation function is applied to increase the number of channels to match the input for the skip connection. [26.]

2.4.2 Feature Pyramid Network (FPN)

Feature Pyramid Network (FPN) is a tool called feature extractor designed for detecting objects in various scales in images with the pyramid method for higher accuracy. [27.]

The architecture of FPN is displayed in the image below, Figure 10, it involves two pathways, a bottom-up pathway with feature maps on the left side within the

image and a top-down way with reconstructed layers on the right side. The bottom-up pathway is composed of traditional convolutional and pooling operations for creating features of various scales. As the layer of bottom-up increases, the resolution decreases. On the contrary, since the deeper layers of a convolutional network and pooling, the more complex pattern of features can be captured, and the semantic value increases. Since the bottom layers lack enough semantic value, it is not used for the prediction, only the upper layers with high enough semantic value are used for detection. However, the upper layers have lower resolution, which makes it difficult to detect a small object. The top-down pathway can solve this problem by enlarging the scale of the high-level semantic value from deeper layers of the network to combine with lower-level features which are having higher resolution. To avoid losing the locations of objects after the operation, the connections between the feature map and corresponding reconstructed layers are needed. [27.]

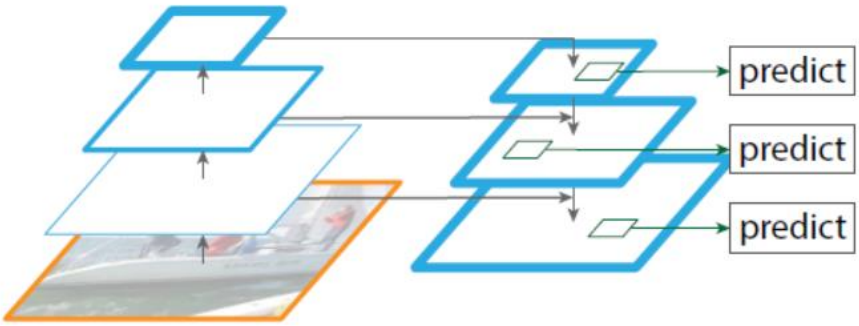

Figure 10. Feature pyramid network [28].

FPNLite is a lightweight variant of the FPN, it is designed for mobile or embedded devices by reducing the number of convolutional layers and filter sizes.

2.4.3 Singel Shot MultiBox Detector (SSD)

The architecture of SSD is composed of three main parts, backbone, also known as base work, extra feature layers, and detection layers as displayed in Figure 11.

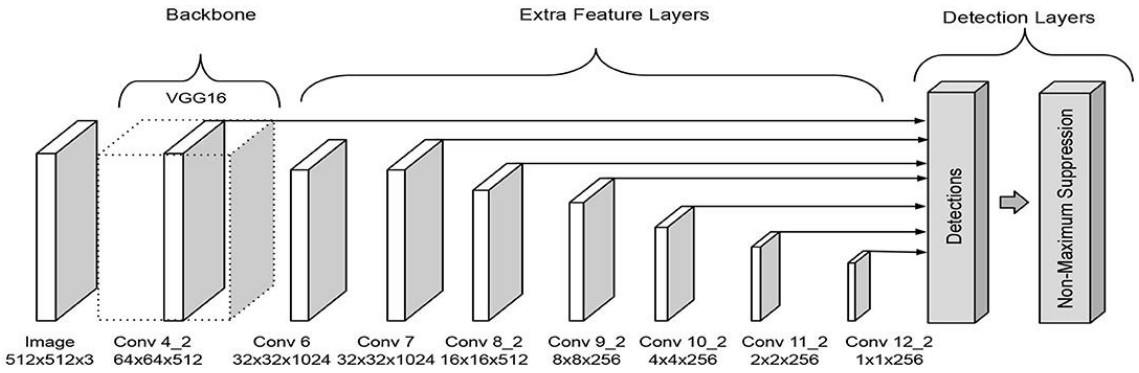

Figure 11. SSD architecture [29].

In the image above the backbone consists of the VGG-16 model network, which is a CNN architecture used for feature extraction. In this project, VGG-16 is replaced by MobileNetV2 to increase the efficiency. FPN is placed in the extra feature layers to allow the model to detect different scales of objects. Detection layers are the place for the last prediction in the SSD. [30.]

The bounding boxes, also known as default boxes or anchor boxes, are generated by the layers from the backbone and extra feature layers. The model makes two types of prediction for each bounding box: location offsets and confidence scores. Location offsets adjust the bounding boxes' location to align with the objects precisely. The main parameters of the location offsets are the center coordinates (cx, xy), width (w), and height (h) of the bounding box. Confidence score 'c' is the number of classes, representing the confidence of the prediction for each object inside the bounding box. [31.]

A loss function is a mathematical function that tells how well the predictions is, based on the ground truth. Ground truth is the true value of the locations of the bounding boxes and the labels of classes decided by the designer. Usually, the smaller the loss, the better predictions the model makes.

In this project, the weighted sigmoid focal loss function is used as the classification loss function and Weighted Smooth L1 Loss is used as the localization loss function. Additionally, an L2 regularization is applied as well.

The weighted sigmoid focal loss function is designed to solve the problem in one-stage object detection models: the class imbalance between foreground and background classes. Foreground classes refer to the classes of the objects of interest, and the background classes refer to non-object classes. Equation (23) below displays focal loss calculation, where weighting factor α is a prefixed value within 0 to 1 to keep the balance between classes. The focusing parameter γ is a prefixed value as well, it controls the focusing effect. For instance, if γ is greater than 0, it can give more weight to hard examples (low p_t) and less weight to easy examples (high p_t). p_t is the model's predicted probability of true class where p is the output from the sigmoid function, see equation (22) above. [32.]

$$L_{classification} = FL(p_t) = -\alpha_t(1-p_t)^{\gamma}\log(p_t) \quad (23)$$

$$\alpha_t = \begin{cases} \alpha & if\ y = 1 \\ 1-\alpha & otherwise \end{cases}$$

$$p_t = \begin{cases} p & if\ y = 1 \\ 1-p & otherwise \end{cases}$$

Equation (24) below displays the calculation of the smooth L1 loss function, where x is the actual coordinate value and y is the coordinate value of prediction. This function helps the model learn how to place the bounding boxes on the objects accurately. It represents the difference between the actual location and the predicted location. [33.]

$$L_{localization} = SmoothL1(x,y) = \begin{cases} 0.5 * \beta * (x-y)^2 & if|x-y| < \beta \\ |x-y| - 0.5 * \beta & otherwise \end{cases} \quad (24)$$

The combination loss is the weighted sum of the classification loss and the localization loss. See equation (25) below, in which W refers to weights. [33.]

$$Loss = L_{classification} * W_{classification} + L_{localization} * W_{localization} \quad (25)$$

L2 Regularization, also known as ridge regression, is a technique with statistical method, used to avoid overfitting in the training process. See the L2 regularization

equation (26) below, where λ is a model hyperparameter, which can adjust the complexity of the model to find the balance between the bias and variance. N is the number of the weights and w is the weight. [34.]

$$L2\ regularization = \lambda \sum_{i=1}^{N} w_i^2 \quad (26)$$

Thus, the total loss, which is the sum of loss and regularization, is defined by the equation presented below, see equation (27) [34].

$$Loss_{tot} = Loss + L2regularization \quad (27)$$

2.5 Mean Average Precision (mAP)

Mean Average Precision (mAP) is a metric used for evaluating the performance of models in localization and classification for object detection. Localization means that the model finds the location of the objects by drawing a bounding around them and classification means that the model identifies what the object is inside the bounding box. In this project, cats and humans are the objects the system needs to identify. [35.]

For the calculation of mAP, the confusion matrix consists of 4 attributes that are needed. They respectively are true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN). When the presence of a label is detected correctly by the model and it aligns with the ground truth, it is TP. When the model identifies the absence of a label correctly, which aligns with the ground truth, it is TN. When the model incorrectly predicts the presence of a label that is not aligned with the ground truth, it is FP. When the model fails to detect a label that is present in the ground truth, it is FN. [35.]

The first calculation is the percentage of the correct predictions when the object is present, known as precision, which is represented by equation (28) below [35].

$$Precision = \frac{TP}{TP+FP} \quad (28)$$

Similarly, how many of the actual instances present in the ground truth are detected correctly by the model, known as recall, is represented by equation (29) below [35].

$$Recall = \frac{TP}{TP+FN} \tag{29}$$

The next calculation is for the Intersection over Union (IoU), which represents the overlapping area between the predicted bounding box and the truth bounding box. It is represented by equation (30) below. [35.]

$$IoU = \frac{overlapping\ area}{area\ of\ union} \tag{30}$$

Next is the example of the usage of the IoU value. If the IoU threshold is set as 0.5, and by the calculation, the IoU value is 0.6, which is greater than the value of the IoU threshold 0.5, then the prediction is classified as TP. On the contrary, if the value of IoU is 0.4, which is less than the IoU threshold value of 0.5, it is classified as FP. [35.]

Average Precision (AP) evaluates the performance in detecting and classifying objects based on the average precision-recall curve, which is the curve of the precision and the recall. It is the area under the precision-recall curve. The curve is not simply plotted by the precision values, it is plotted by the interpolated across 11 recall values, also known as the 11-point interpolation method. The interpolated precision means that takes the highest precision of each recall level in the descending order. For instance, Figure 12 below is an example of a precision-recall curve. The orange curve represents the values of the original precision according to the recall. The green curve represents the interpolated precision according to the recall. From recall interval 0 to 0.4, the highest precision is 1, therefore, the curve remains at 1. According to the second highest precision, the next interval is from 0.5 to 0.8, the highest precision is 0.57, therefore, this curve remains at 0.57. Similarly, in the interval 0.9 to 1, the curve remains at 0.5, which is the highest precision when recall is 1. Equation (31) below is the calculation of AP. [35.]

$$AP = \frac{1}{11} * Sum(11point\ interpolated\ precision) \quad (31)$$

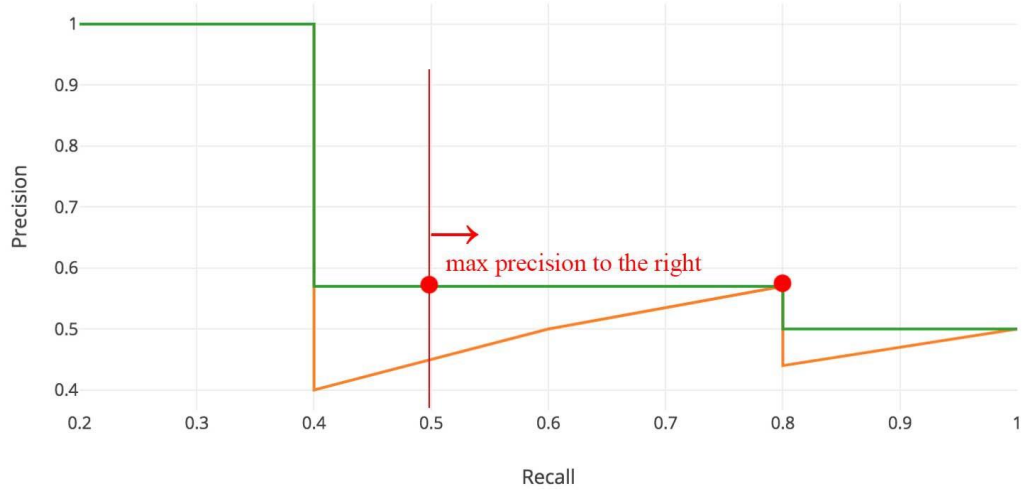

Figure 12. Recall the precision curve [36].

Finally, the mAP is calculated by the average of the AP values across all classes. Equation (32) below represents the calculation of the mAP, where N is the total number of classes. [37.]

$$mAP = \frac{1}{N} * (\sum_{i=1}^{N} AP_i) \quad (32)$$

The mAP is calculated starting from a 0.50 IoU threshold, with increments of 0.05 up to 0.95. Usually, the higher the score of the total average mAP, the higher the accuracy of the model is, and it should be higher than 50% to achieve the tasks of object detection with high accuracy. [37.]

3 System Architecture and Workflow

This chapter presents the system's architecture design and development methodologies for the object recognition and temperature-based fan control system in Raspberry Pi 3.

3.1 System Architecture

The system consists of 4 main components: object recognition, temperature sensing, fan control, and LCD for monitoring. The architecture of the system is displayed in Figure 13 below. The object recognition tasks are conducted by a TensorFlow Lite model on the Raspberry Pi 3, with inputs from the camera connected via the Camera Serial Interface (CSI). The temperature sensing task is performed by a DHT11 sensor connected to the Raspberry Pi through the General-Purpose Input/Output (GPIO) pins. The fan control task is performed by a 12V DC fan connected to the Raspberry Pi via GPIO pins that support Pulse Width Modulation (PWM). The monitoring task is performed by a 16x2 black-on-yellow LCD with an I2C connection with Raspberry Pi.

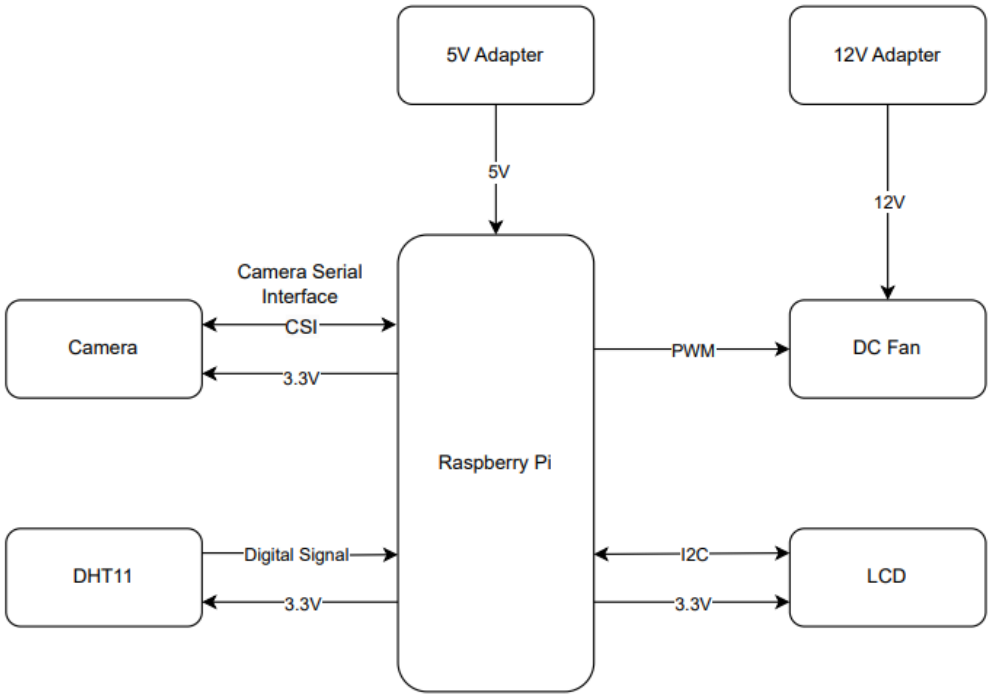

Figure 13. System architecture.

3.2 System Workflow

The system's operation depends on several conditions. See the process flowchart displayed in Figure 14 below. The first condition is whether the objects,

here cats and humans, are in the sight of the camera. If there is no object detected, the system should remain inactive and the counter for ‘no object’ should add 1. If the system is active, it will become inactive after the no-object counter count to 6. When an object or multiple objects are detected, the no-object counter resets to 0, and the system can perform three different main actions depending on the situation. The three situations respectively are: cat, human, cat, and human. When cats are detected by the camera, the system checks the ambient temperature. If the temperature is also higher than the specific temperature degrees, which is the lower threshold here, the fan turns on. The specific temperature is determined by the user. For the explanation here, 20 Celsius degree is used as the lower threshold and 35 Celsius as the upper threshold. When the ambient temperature is lower than 20 Celsius degrees, the fan turns off. When the ambient temperature is higher than the upper threshold of 35 Celsius degrees, the fan remains at maximum speed, which is the same as the ambient temperature at 35 Celsius degrees. Whether the fan is on or not, an awake cat face is displayed on the LCD. When humans are detected, the information of current temperature and fan speed are displayed. Since the fan is off in this situation, the fan speed should be 0%. When both cats and humans are detected, if the temperature is higher than the lower threshold in degrees Celsius, the fan turns on, otherwise the fan remains off. Additionally, when the ambient temperature is higher than 20 Celsius degrees but lower than 35 Celsius degrees, the higher the Celsius degrees the faster the fan speed archives. When the ambient temperature is over 35 Celsius degrees, the fan speed always remains the same as when it is at 35 Celsius degrees. The information, which is the current temperature and fan speed, is displayed on LCD. The information dominates the LCD by using a counter called Msg. The cat's awake face is displayed only when this counter equals to or above 5. The Msg counter resets to 0 when humans are detected, otherwise, it adds 1. Another counter is used to count the appearance frequency of humans. If humans appear in a row 6 times or more, then the system considers that it is the situation in which a human is detected. Otherwise, the counter is reset to 0, and the fan keeps running.

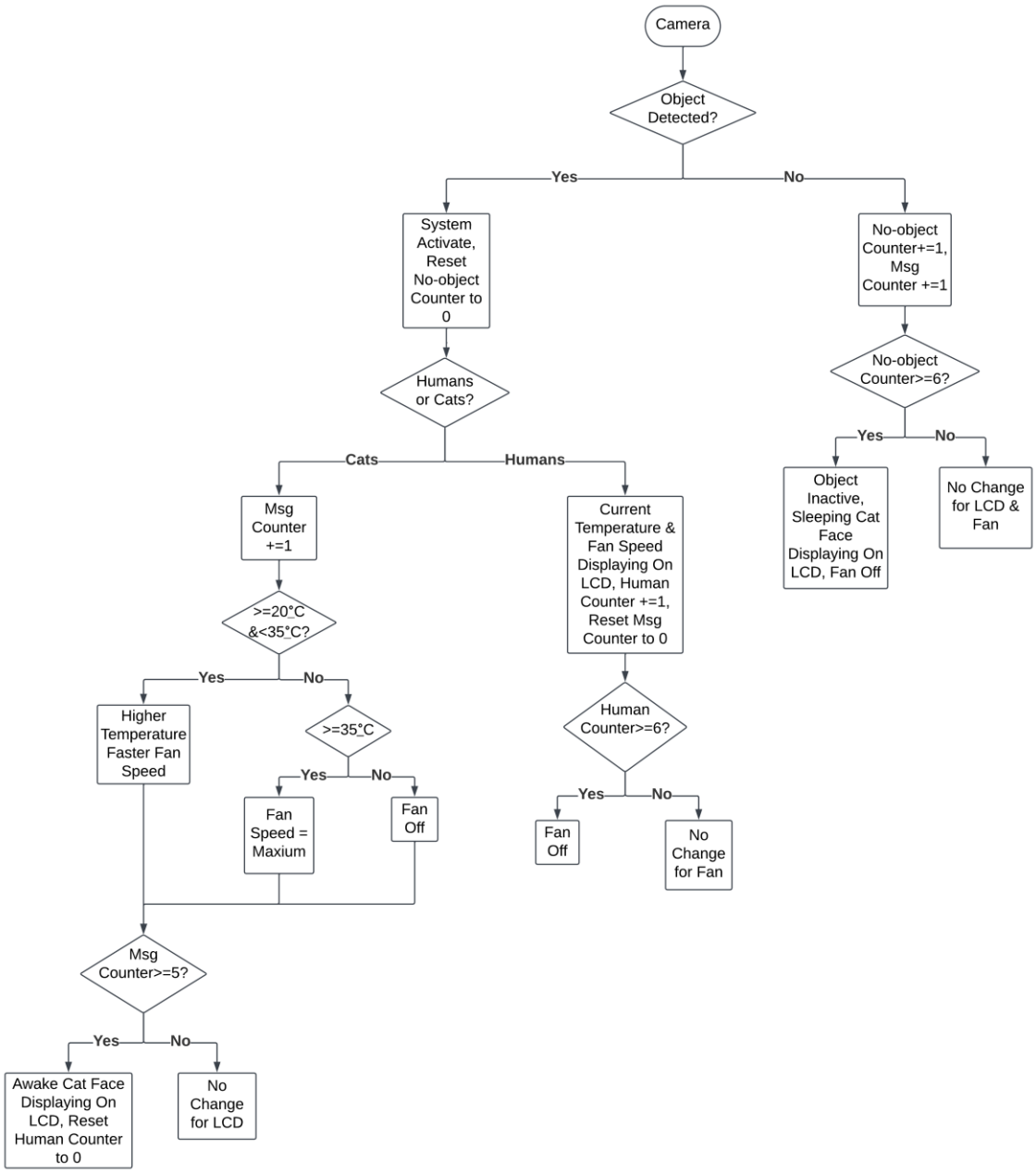

Figure 14. System process flowchart.

4 System Implementation

This chapter covers the steps involved in executing the project based on the theories which have been discussed in Chapter 2.

4.1 Object Recognition System Implementation

The object recognition system was implemented first with each step discussed in the subsections below.

4.1.1 Data Collection and Image Annotation

Input data is important for machine learning, as it determines what the model can learn. As the model is designed to recognize two objects: cats and humans, the first thing needed to do was to collect sufficient images of cats and humans as input dataset. It was ideal to have various human' and cat' images, but the experimental subjects for this project were only me and my cat. Therefore, I collected 236 pictures of myself, 226 pictures of my cat, and 8 group pictures of both. To enhance the training process, pictures of us were in various poses, angles, distances, and brightness along with random objects in different backgrounds.

The next step is image annotation, which is the process of labelling using annotation text. To annotate the image dataset, an open-source graphical image annotation tool called labelImg was used. LabelImg allows users to draw a box around the object and write the name of the class to which the object belongs.

Figure 15 below is one of the images used for the labelling process in labelImg. A bounding box was drawn around the cat. The less background area inside the bounding box, the better the material for model training. The cat within the bounding box was labelled as a 'cat'.

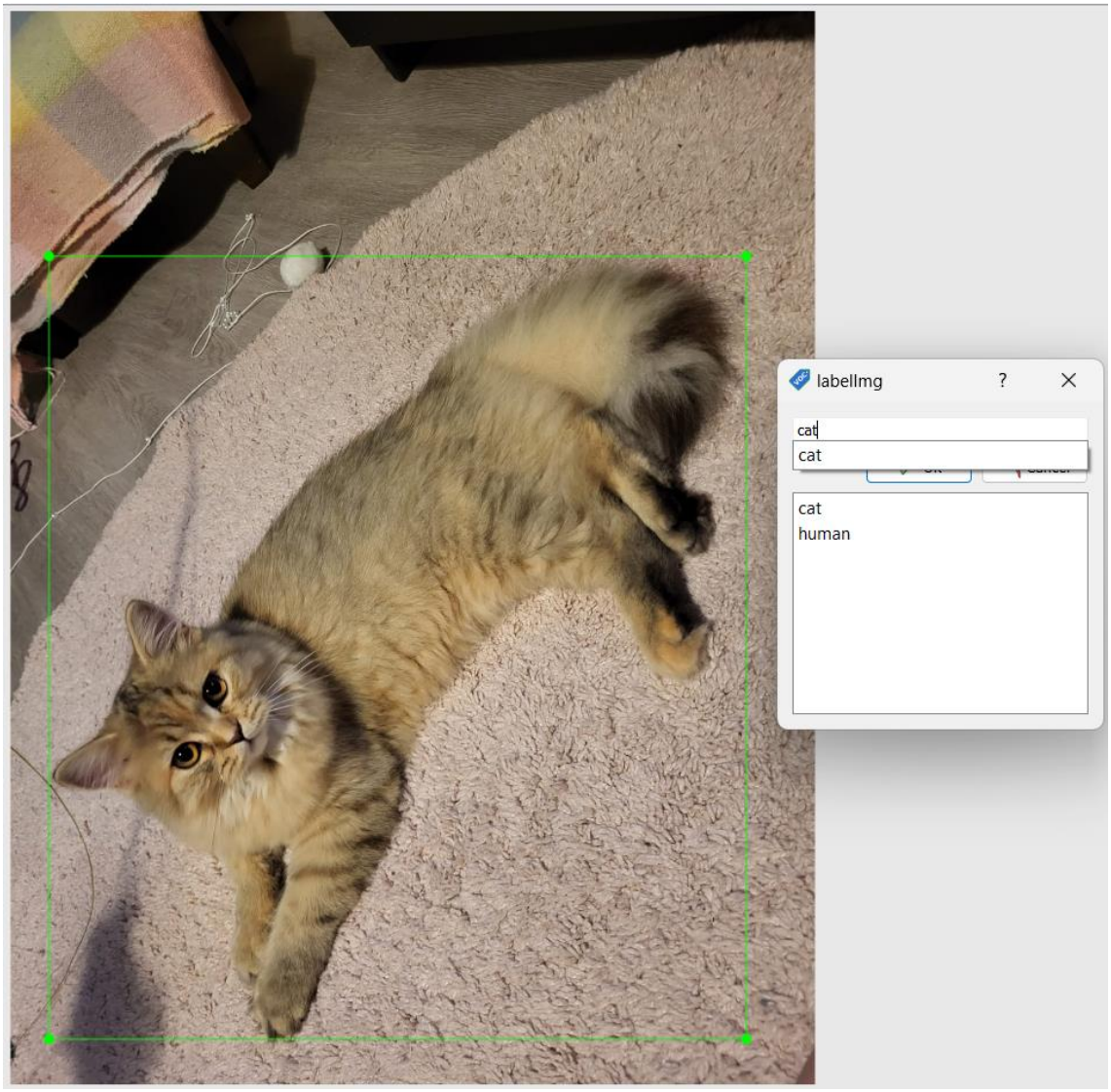

Figure 15. An example of labelling process.

After the operation of labelling, an XML file was generated, which contained annotation data such as the bounding box coordinates, classes, and other information for the image in Pascal Visual Object Classes (VOC) format. VOC format is a widely used format for image datasets for object detection.

Figure 16 below displays the contents of the XML file for Figure 15. This XML file contained the filename, the folder name, and the location of the chosen image. The origin of the image was not specified while this photo was taken by a smartphone. The size subsection represents the image was 1200 pixels wide, 1600 pixels tall, and has a depth of 3, indicating it is a colour image. For black-

white images, it is 1. Since pixel-wise segmentation masks did not apply to the image, it was not segmented, therefore the value in the subsection was 0. The next section tells the name of the class that the image belongs to. The pose of the object was not specified, and it was not truncated, which means the object was fully visible, it was not extended beyond the bounding box. The difficult tag tells if the image is difficult to detect. 0 means it is not difficult, else it is 1. Bndbox tag represents the pixel coordinates of the bounding box. Tag xmin refers to the left boundary of the bounding box, tag xmax refers to the right bounding box, tag ymin refers to the top boundary of the bounding box and tag ymax refers to the bottom boundary of the bounding box.

```
<annotation>
    <folder>Belle2</folder>
    <filename>cat_13.jpg</filename>
    <path>C:\Users\ASUS\Pics\Belle2\cat_
13.jpg</path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>1200</width>
        <height>1600</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>cat</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>57</xmin>
            <ymin>365</ymin>
            <xmax>1097</xmax>
            <ymax>1532</ymax>
        </bndbox>
    </object>
</annotation>
```

Figure 16. An example content of XML file.

4.1.2 Model Training and Conversion

In this project, model training was trained on Google Colab, which is a cloud-based platform with GPUs and Tensor Processing Units (TPUs) that allows users to write and run Python code in the web browser. An open-source machine

learning software library called TensorFlow was used to build and train the deep learning model for this project.

Before the training, there were several dependencies had to be installed for TensorFlow. The images of humans and cats were uploaded into Google Colab. The images were split into three different folders: ‘train’, ‘validation’, and ‘test’. The images in the ‘train’ folder were the inputs in the training process. A batch size of images from the ‘train’ folder is input into the neural network at each training step. The loss is calculated through the training algorithm. Based on the loss, the network’s weights are updated through backpropagation to improve accuracy. The images in the ‘validation’ folder were not used for training the model. Instead, it was used to monitor the training progress and adjust the hyperparameter, such as the learning rate. The images in the ‘test’ folder would not appear in the training process. They were used for the final testing to see the accuracy of the trained model. The images were randomly distributed to those three folders. 376 images in the ‘train’ folder, 47 images in the ‘validation’ folder, and 47 images in the ‘test’ folder.

The next step was to create a label map file, which is a text file containing a list of classes with corresponding integer IDs used in model training. When the model detects objects, the integer IDs are assigned based on this label mapping. In this project, the list of classes included cats with ID of ‘1’ and humans with ID of ‘2’. Since the XML files with Pascal VOC format cannot be used directly, they had to be converted to TFRecord files eventually. TFRecord is a binary format for storing data with higher efficiency during the smaller size. But before that, XML files had to be converted to Comma-Separated Values (CSV) files in CSV format first, which was represented as ‘file name, width, height, class, xmin, ymin, xmax, yman’. Then the CSV files were converted to TFRecord files.

A pre-trained TensorFlow model was required for the training process. According to the research conducted by EJ TECHNOLOGY [38], the SSD-MobileNet-v2 (FP32) model performs with excellent accuracy even with a relatively small training dataset, making it a good fit for this project, despite it being slower than

the regular SSD-MobileNet-v2 model. The pre-trained model and the configuration file should be downloaded to Google Colab. Additionally, the code of the SSD-MobileNet-V2(FP32) pre-trained model is provided in Appendix 1. [38.]

Despite the configuration file from open sources defining most of the custom dataset for the model, there were still two parameters that needed to be set before the training: the number of training steps and the size of the batch. A step is a gradient update. Due to the usage limits in Google Colab, the number of steps was set to 20,000. The size of the batch represents the number of images used per training step. Generally, 16 is used for SSD models, therefore, the batch size was set to 16. If a size of 16 leads to memory error, the problem can be solved by reducing the size of the batch and raising the number of steps. [30.]

Model training was ready to start in the TensorFlow object detection API. Without any interference, usually the training process stops after all steps are completed, but if the model is good enough, the training process can be stopped earlier manually. To avoid the training time reaching the limitations, the training process was stopped manually when the step was around 18,000.

To observe the result of the training, a monitor called TensorBoard was required for the project. The graphs of the loss and the learning rate were displayed on the TensorBoard during the training. Figure 17 displays the graphs of losses during training. Total loss is the sum of classification loss, localization loss, and regularization, which have been mentioned in the theoretical Section 2.4.3. In general, the lower the loss, the better performance the model makes.

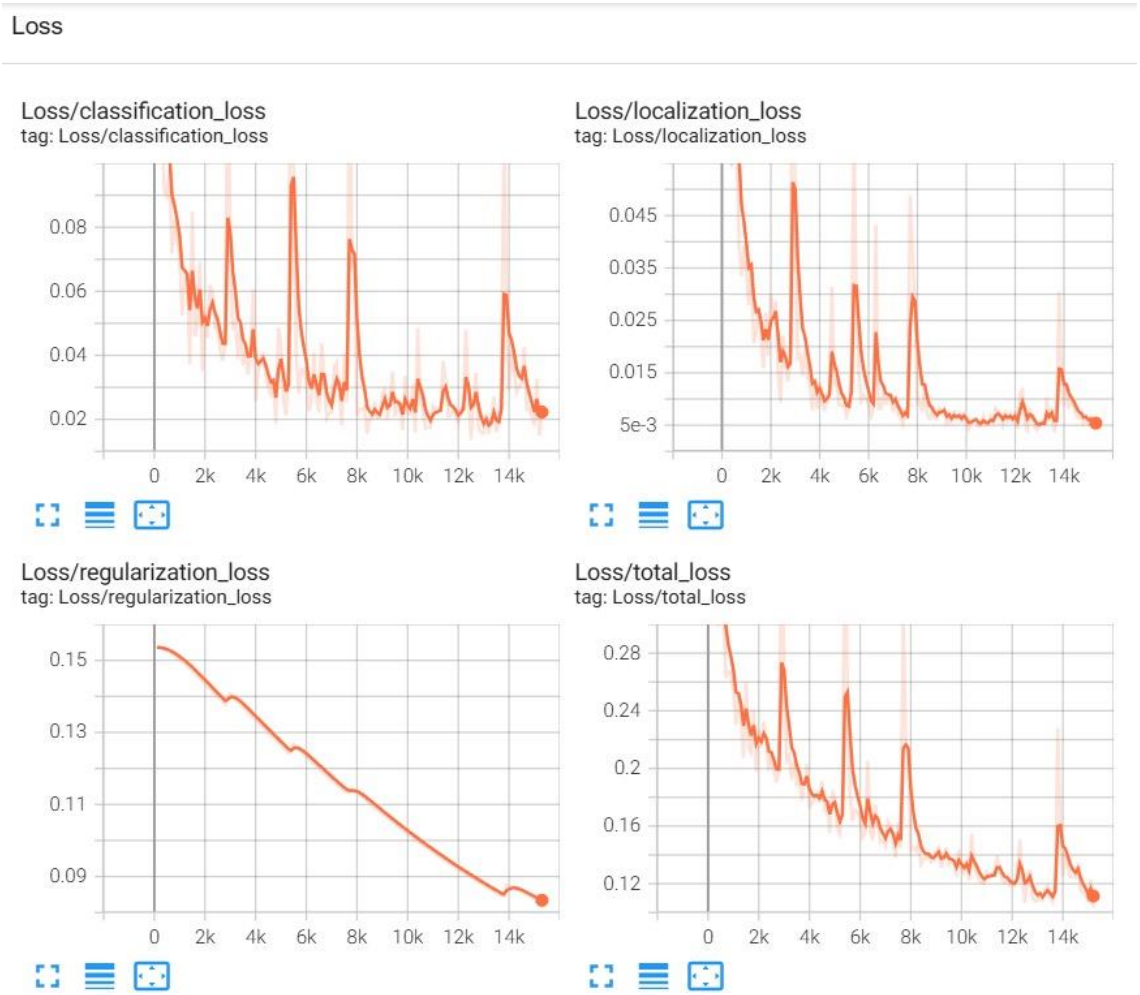

Figure 17. Graphs of loss displayed on TensorBoard during training.

Figure 18 below is the graph of the learning rate during training. As mentioned in the theoretical Section 2.2.3 above, this is a cosine decay learning rate. The learning rate first increases linearly in the warm-up phase, then decreases gradually to approach 0. The learning rate decides the speed of the learning process of the model.

Figure 18. Graph of learning rate displayed on TensorBoard during training.

The next step was to export the file that contains the information on the architecture and weights of the model, also known as the model graph, to a format of TensorFlow Lite-compatible format and convert the exported graph file into the TensorFlow Lite model file. The TensorFlow Lite model file is the file that is allowed to be deployed into edge devices through the TFLiteConverter module. After the conversion of the model, it was ready for the next step. Additionally, the size of the trained model was 10.9 MB, and the training time was about 3 hours.

4.1.3 Model Deployment

After the training process, the trained model needed to be downloaded and deployed into a device. For this project, Raspberry Pi 3 with the Raspberry Pi operating system was the selected edge device for the trained model.

Since the TensorFlow and other dependencies in Raspberry Pi may conflict with other versions existing in the Raspberry Pi, it was necessary to create a virtual environment for version management to avoid the conflict issue.

To run the trained model in Raspberry Pi successfully, both TensorFlow and Open Source Computer Vision Library (OpenCV) had been installed beforehand.

OpenCV mainly is used in video capture from a webcam, and frame preprocessing including tasks such as converting the captured frame from BGR format to RGB format and resizing the frame for the requirements, Drawing the bounding boxes and displaying labels, confidence scores, and frame rate which is in Frames Per Second (FPS) and other necessary adjustments.

After the installation, Raspberry Pi was ready for the model. The TensorFlow-lite model file and the label map text file were transferred from the computer to the Raspberry Pi 3. A Raspberry Pi camera was required as the eye for the system, it was connected to the Raspberry Pi 3 through Camera Serial Interface (CSI). The object recognition display was shown on the computer screen connected with Raspberry Pi 3. After the preparation mentioned before, the model was ready to use in Raspberry Pi 3 for the task of object detection.

4.2 Temperature-Based Fan Control System Implementation

This section provides a detailed discussion of the implementation of the temperature-based fan control system.

4.2.1 Temperature Sensing Module

The main temperature sensing component is a DHT11 sensor, which is a temperature and humidity sensor. It has low power consumption, and it is easy to use in a Python environment which meets the conditions of this project.

There are 4 pins on the sensor, the VCC pin was connected to the 3.3V output on the Raspberry Pi. The GND pin was connected to the ground pin on the Raspberry Pi. The NC pin was not connected. The D2 pin was connected to a GPIO pin on the Raspberry Pi. After the configuration of the development environment and the installation of the necessary libraries, the DHT11 sensor was ready. Since the temperature is the only required sensing value for this system from the DHT11 sensor, therefore, the sensing value of humidity was ignored.

4.2.2 DC Fan Cooling Module

A 12V 4-wire DC fan with a required current of 0.45A was used in the cooling module. The 4 pins respectively are ground, power supply, sense/tach, and control/PWM, tach refers to tachometer here. The ground pin was connected to the Raspberry Pi sharing the same ground with other components. The pin for the power supply was connected to a 12V adapter with a maximum supply current of 0.65A. The sense/tach pin generally is used for providing a feedback signal that indicates the fan's rotational speed. Since the fan speed was set by the user here, to simplify the circuit and the code, this pin was not connected. The control/PWM pin was connected to a GPIO pin which supports the PWM on Raspberry Pi.

The frequency of PWM was set to 1000 Hz, which meant there were 1000 on/off cycles per second. A range of temperature from 20 Celsius degrees to 35 degrees was set. According to the design plan for the functionality, the fan speed increases as the temperature rises in the range. When the temperature is lower than 20 Celsius degrees, the fan is off. When the temperature is higher than 35 Celsius degrees, the fan speed remains the same as at 35 Celsius degrees, which is the maximum fan speed for this project. The fan speed is calculated by equation (33) below, which is based on the linear interpolation method, where temp represents the current temperature detected by the DHT11 sensor, low represents the lower bound of the range in temperature, it is set to 20 Celsius degrees, high represents the upper bound of the range in temperature, which is 35 Celsius degrees here. xFactor is a flag, it can toggle on or off for the fan depending on whether a human or cat is detected. When a human is detected, xFactor is set to 0, therefore, the fan remains off even if the temperature is above 20 Celsius degrees. When a cat is detected, xFactor is set to 1, allowing the fan to turn on and run the speed.

$$Fan\ Speed = xFactor * (temp - low) * \frac{100}{high - low} \quad (33)$$

4.2.3 Monitoring Module

The main component in the monitoring module is a 16x2 black-on-yellow LCD. It is connected to the Raspberry Pi via an I2C interface. The operating voltage was 3.3V supplied by Raspberry Pi. The LCD has 2 build-in pull-up 10kΩ resistors, one of them is located between VCC and SCL pins, another one is located between VCC and SDA pins [39]. Similar to the DHT11 sensor, after the configuration of the development environment, the monitoring module was ready. The main task for the project was to display the value of temperature, fan speed, and digital cat faces according to the mode of the system.

4.3 Sensor Platform Circuit Board and 3D Enclosure Design

This section explains the sensor platform circuit board for a temperature-based fan control system to simplify the connections between components and Raspberry Pi. Additionally, a 3D model design for the enclosure of the object recognition system is displayed.

Figure 19 below displays the schematic of the sensor platform circuit. J1 is a 2x8 16-pin female header. It is used for the connection between the sensor platform circuit board and the Raspberry Pi 3 via GPIO pins, therefore, it was placed on the bottom layer of the circuit. J2 is another header, its front 4 pins are used to connect an LCD via an I2C interface. Pin 5 has no connection, and the last 3 pins are used to connect an external DHT11 sensor. The circuit board itself has an internal DHT11 sensor IC1 as well. The external DHT11 sensor will be used when the Raspberry Pi 3 gets hot, since the heat from Raspberry Pi 3 may affect the accuracy of the sensing result. External DHT11 sensor and internal DHT11 sensor can be toggled by manually switching the slide switch S1. R1 is a 5KOhm resistor required for the internal DHT11 sensor. SW_Um1 is a 2-pin terminal block connector. It is used to connect a 12V adapter to the circuit board. S2 is a switch that allows users to turn on or off the 12V power supply. J3 is a 4-pin terminal block connector. It works as a DC fan interface.

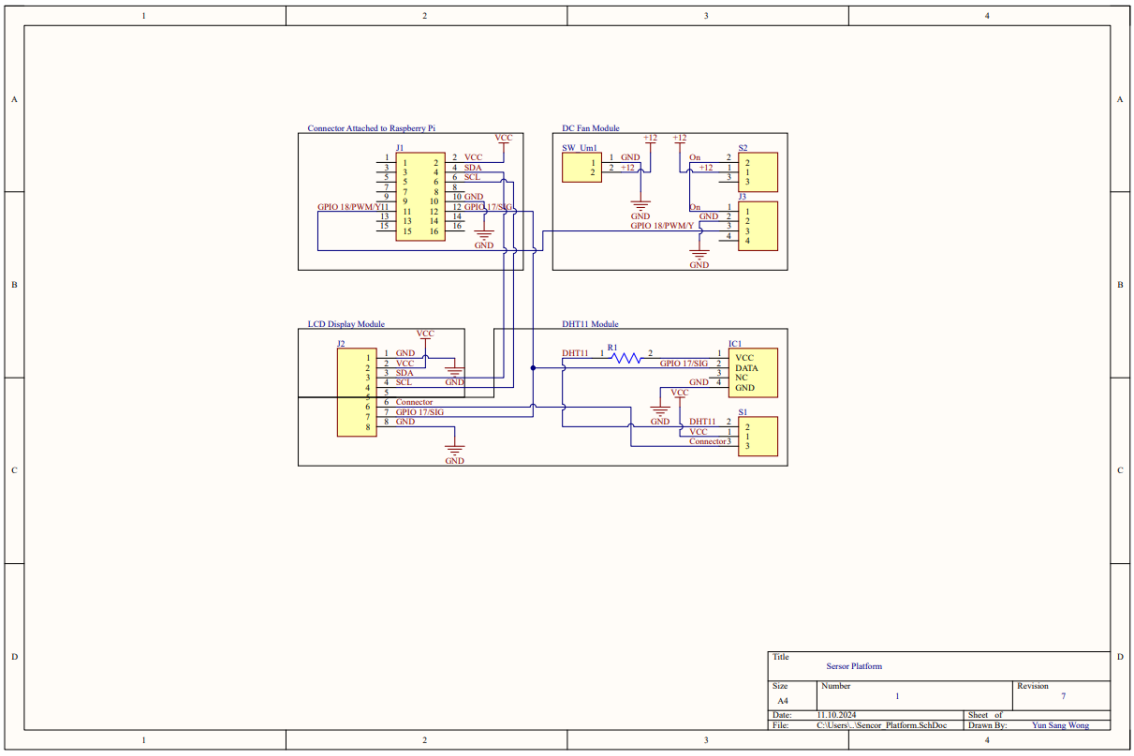

Figure 19. Schematic of the sensor platform circuit.

Figure 20 below is a 2-layer PCB design for the sensor platform circuit. A ground plane was applied and located on the top layer. As mentioned before, only J1 was placed on the bottom. Therefore, its designator was invisible in the overall view option. Other components were all located on the top layer.

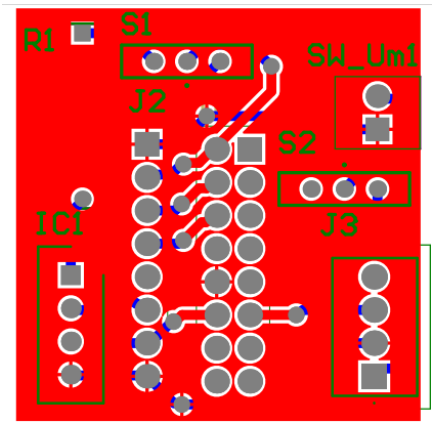

Figure 20. PCB design for the sensor platform circuit.

Since the device is used for pets, an enclosure is necessary for safety. Additionally, an enclosure can also enhance the device's overall elegance. Figure

21 displays a 3D design enclosure. To avoid overheating issues, numerous holes were applied around the enclosure in the design. The fan was designed to be placed at the bottom of the enclosure. The camera was designed to be placed above the fan. The LCD would be above the camera and the Raspberry Pi 3 would be placed behind the fan. To allow the Raspberry Pi 3 and the fan to connect to the power supply, monitor, keyboard, and mouse, two rectangular openings were made on the back and left side of the enclosure. Additionally, the physical implementation of the sensor platform circuit board and the enclosure is provided in Appendix 3.

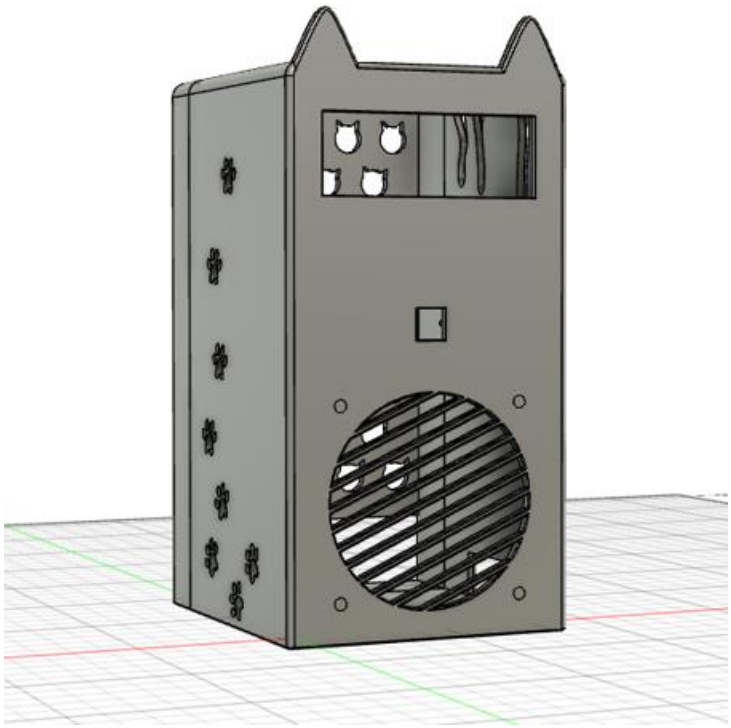

Figure 21. A 3D design enclosure.

5 Experimental Results

5.1 Object Recognition System Results

This session describes the mAP results, test on image data, and real-time performance on Raspberry Pi 3 and computers with Windows systems.

5.1.1 Mean Average Precision (mAP) Results

This test was conducted by calculating the mAP of the model over 47 images in the ‘test’ folder. Table 1 below are the results of the Average Precision (AP) and mAP in different IoU thresholds and the average of them for cats and humans. The AP for the human class remains high across most IoU thresholds, while the AP for the cat class decreases as the IoU threshold increases. As a result, the average mAP was 75.09% for cats and 86.57% for humans, indicating that the model detects humans more accurately than cats. This discrepancy can probably be attributed to the larger number of training images used for humans. The overall was the average value of the mean value of cat and human together. It scored an average of 80.83%, which was above 50%, it is considered quite accurate. Especially when the IoU threshold at 0.5, the model scored 97.55%, which is considered a high accuracy based on the test made by Pascal VOC 2007 [40].

Table 1 Result of the mAP at various IoUs.

	IoU Threshold	Cat AP(%)	Human AP(%)	mAP(%)
	0.50	99.87	95.24	97.55
	0.55	99.87	95.24	97.55
	0.60	99.87	95.24	97.55
	0.65	92.98	95.24	94.11
	0.70	87.65	95.24	91.45
	0.75	83.32	95.24	89.28
	0.80	79.55	95.24	87.39
	0.85	51.22	95.24	73.23
	0.90	42.40	95.24	68.82
	0.95	14.20	8.57	11.38
Average		75.09	86.57	80.83

5.1.2 Model Testing on Image Data

There were 47 images in the ‘test’ folder, which had not been used during the model training. That means the model had never seen those images. Therefore, they were used for the image testing.

Figure 22 below is one of the results from model testing on image data. Slightly blurry and partially visible images were intentionally used in the test to simulate real conditions with moving objects. The location of the bounding boxes was quite accurate. It found the locations of the objects in the image, even though they were only partially visible. The model also successfully identified the objects in the image with a confidence score of 99% for a human and 97% for a cat. The results on other images were also as good as this result.

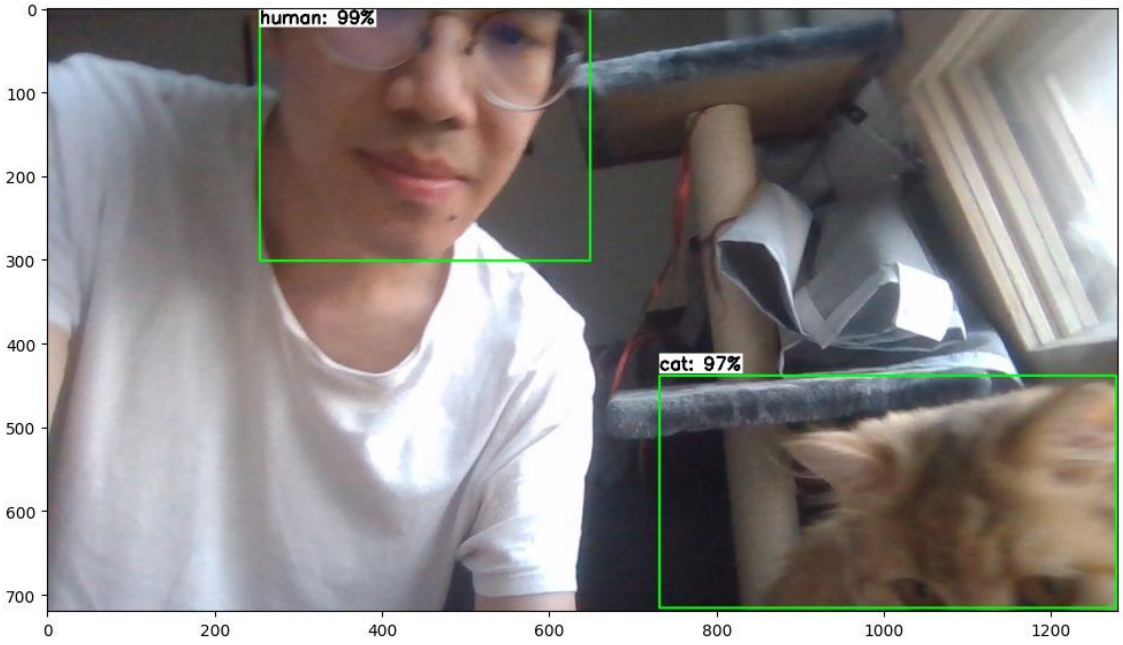

Figure 22. One of the results from model testing on image data.

5.1.3 Real-Time Model Testing

This test was conducted on the Raspberry Pi 3 with a Raspberry Pi camera using the trained model. The object recognition display was shown on a computer monitor connected with Raspberry Pi 3.

Figure 23 below displays the performance. The programme with the trained model could detect humans and cats successfully with high confidence scores and output the labels ‘cat’ and ‘human’ when the cat and human were detected accordingly even though we were backlit. However, the FPS was respectively

low, which reduced the responsiveness of the real-time detection. Reducing image resolution for the object detection interface could increase the FPS slightly. However, the main problem caused by it was probably because the Raspberry Pi 3 itself had a relatively low-powered Central Processing Unit (CPU) and insufficient Random Access Memory (RAM). Overall, the performance of the object recognition was still satisfactory and met the requirements.

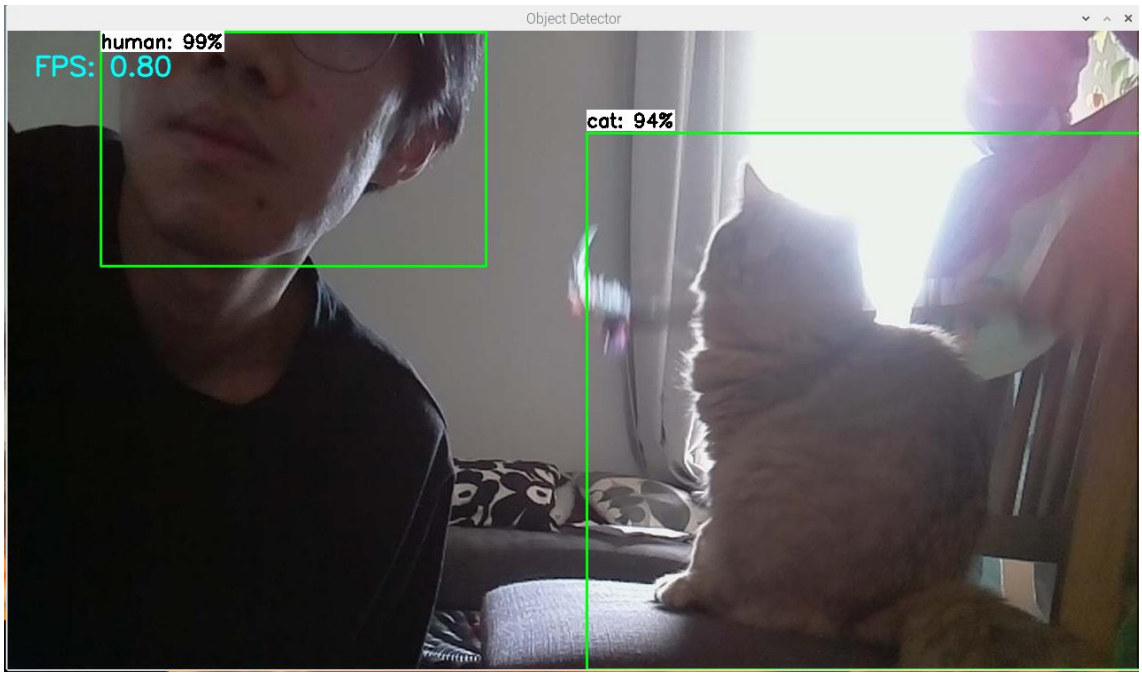

Figure 23. Performance of object recognition system on Raspberry Pi 3.

To have a comparison with the performance of Raspberry Pi 3, the model had been deployed on Windows as well. Figure 24 below is the performance of the object recognition system on Windows. Same as before on Raspberry Pi 3, the model could rapidly recognize humans and cats. The FPS on Windows was around 13 to 14, which was faster than on Raspberry Pi 3.

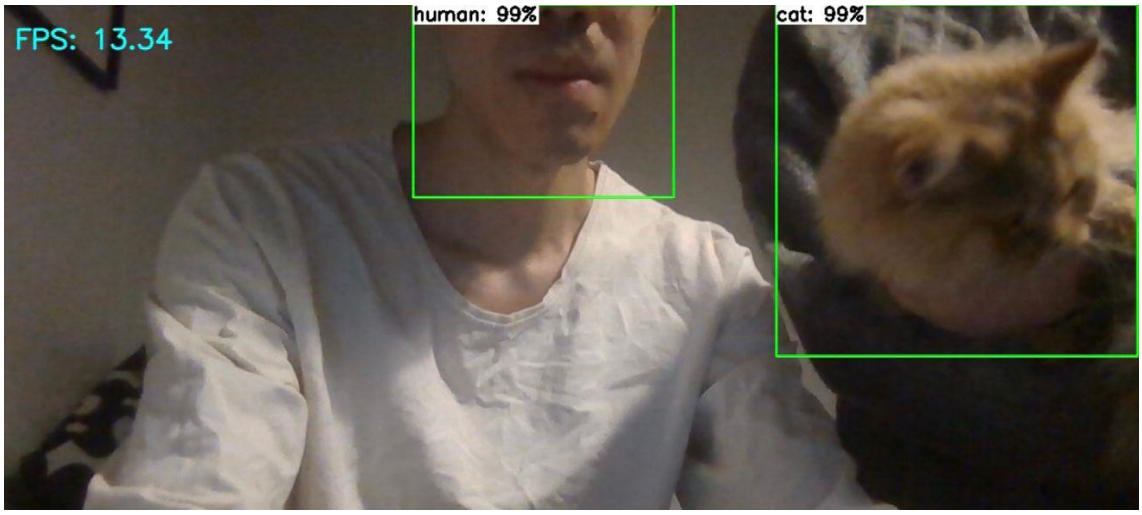

Figure 24. Performance of object recognition system on Windows.

5.2 Temperature-Based Fan Control System Testing and Results

Since there was not any input for the temperature-based fan control system from the object recognition system, the inputs for the fan control system were set by the user to simulate the output from the object recognition system, which were string text: ‘cat’, ‘human’ and ‘none’. The situation of ‘cat and human’ is achieved by inputting ‘cat’ and ‘human’ alternately. The indoor temperature was 24 Celsius degrees, which was above the lower threshold and lower than the upper threshold, meaning that it is within the range of 20 to 35 Celsius degrees. To have other temperature statuses, below the lower threshold and above the upper threshold, the range was adjusted accordingly. Table 2 below are the results of the testing. The various temperature statuses were the variables used for testing.

As shown in the table, when the current temperature was below the lower threshold, which was adjusted to 26 Celsius degrees, the fan speed remained at 0%. The contents displayed on the LCD were depending on the input. When the inputs were ‘human’ the input was alternated between ‘human’ and ‘cat’, and the LCD displayed the information on the current temperature and fan speed. When ‘cat’ was the input of the system, the face of the awake cat was displayed. When the input was ‘none’ in a row of 6 times or more than 6 times, the sleeping cat face was displayed. The performance in this status aligned with the design plan.

The next temperature status was that the current temperature was within the temperature range of 20 to 35 Celsius degrees. In this status, when ‘human’ was the input in a row at least 6 times, the fan speed was 0% and the information on the current temperature and the fan speed was displayed on the LCD. When ‘cat’ was the input for the system, the fan was switched on with a speed of 27%, and the awake cat face was displayed on the LCD after a counter called ‘Msg’ was counted 5 times. When both ‘human’ and ‘cat’ were the alternating input of the system, the fan was running at a speed of 27%, and the information on the current temperature and fan speed was displayed on the LCD. When the input was ‘none’ in a row of 6 times or more, the fan was turned off and the sleeping cat face appeared on the LCD again. The performance of this status was as planned as well.

The last temperature status was that the current temperature was above the upper threshold of 22 Celsius degrees. In this status, the performance was almost the same as the status in which the current temperature was within the range, the only difference was the fan speed. In this status with input conditions of ‘cat’ and alternating with ‘human’ and ‘cat’, the fan remained at its maximum speed of 100%. The performance of this status was also as expected.

Table 2. Results from object recognition system testing.

Current Temperature(°C)	Temperature Range(°C)	Temperature Status	User Input	Fan Speed(%)	LCD Display
24	26-35	Below Lower Threshold	Human	0	Info
24	26-35	Below Lower Threshold	Cat	0	Awake Cat Face
24	26-35	Below Lower Threshold	Human&Cat	0	Info
24	26-35	Below Lower Threshold	None	0	Sleeping Cat Face
24	20-35	Within Range	Human	0	Info
24	20-35	Within Range	Cat	27	Awake Cat Face
24	20-35	Within Range	Human&Cat	27	Info
24	20-35	Within Range	None	0	Sleeping Cat Face
24	20-22	Above Upper Threshold	Human	0	Info
24	20-22	Above Upper Threshold	Cat	100	Awake Cat Face
24	20-22	Above Upper Threshold	Human&Cat	100	Info
24	20-22	Above Threshold	None	0	Sleeping Cat Face

5.3 Integrated System Testing and Results

This testing was conducted by integrating an object recognition system and the temperature-based fan control system. When ‘human’ or ‘cat’ was detected, the string ‘human’ or ‘cat’ was successfully output a string text accordingly. Figure 25 below illustrates the object detection display when system detects a human. According to the image, the system successfully located a partially visible human face by a bounding box with a high confidence score of 99%. The FPS was 0.28, which is slow.

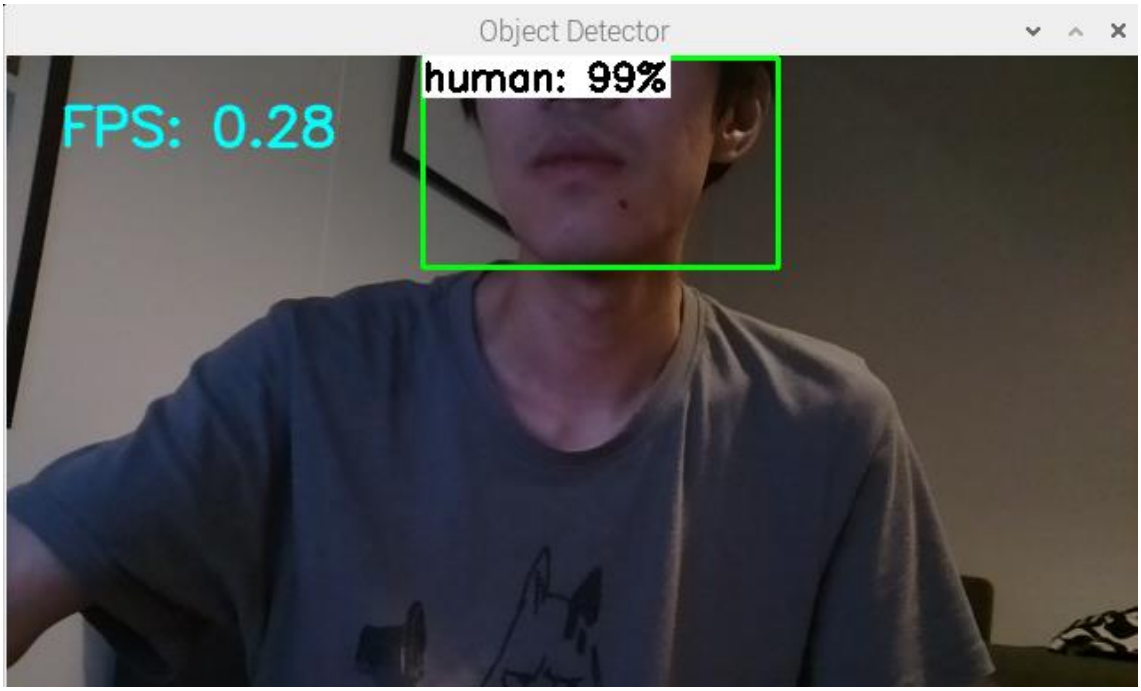

Figure 25. Object recognition display interface when a human is detected.

Figure 26 illustrates the object detection display when the system detects a human and a cat together. According to the image, when both a human and a cat were in the camera's view, they were successfully detected together on the object detection display with alternating output of 'human' and 'cat'. The system successfully located both objects with high confidence scores: 96% for the human and 98% for the cat. The FPS was 0.11, which is even lower than when only a human was detected.

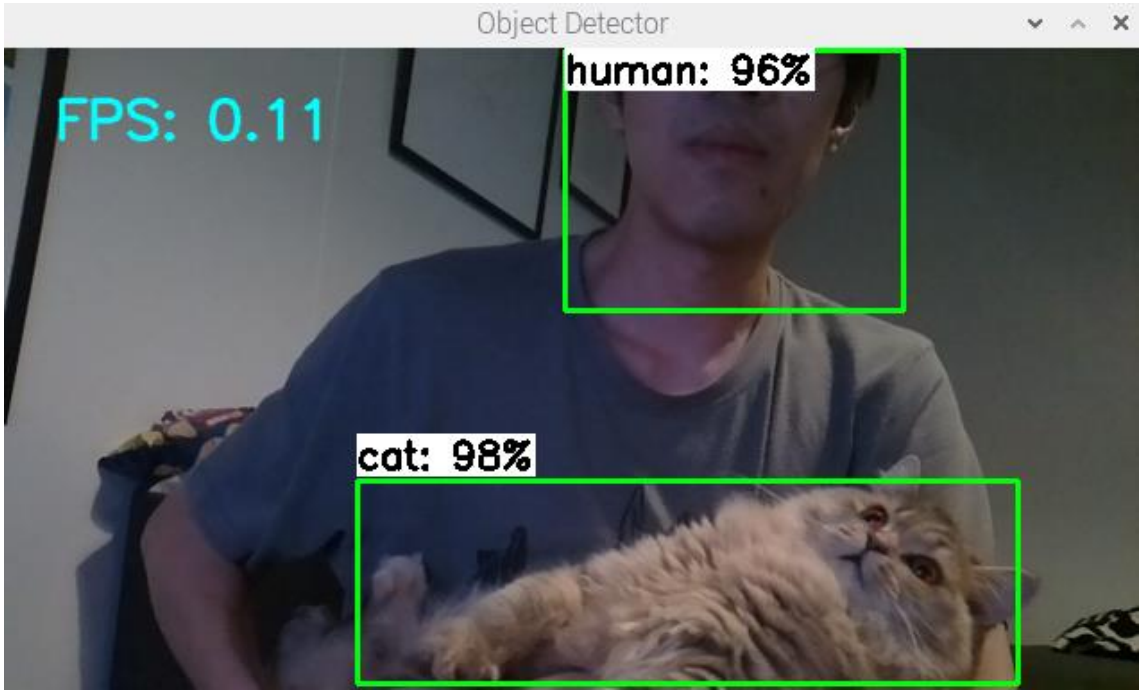

Figure 26. Object recognition display interface when a human and a cat are detected together.

The string outputs from the object recognition system were fed into the temperature-based fan control system as its inputs, which then executed actions according to those inputs. Table 3 below displays the results of the testing which are almost the same as the results of the testing for the temperature-based fan control system. The difference between them is that instead of using inputs from the user, the inputs come from the object recognition system, and there is no 'none' input. This condition indicates that no object has been detected. Since the outcome from this part was the same as the outcome from the test of the temperature-based fan control system, look at the analysis in Section 5.2 for the details. Additionally, the actual images of the contents displayed on the LCD are provided in Appendix 2.

Table 3. Results from integrated system testing.

Current Temperature(°C)	Temperature Range(°C)	Temperature Status	Object Detected	Fan Speed(%)	LCD Display
24	26-35	Below Lower Threshold	Human	0	Info
24	26-35	Below Lower Threshold	Cat	0	Awake Cat Face
24	26-35	Below Lower Threshold	Human&Cat	0	Info
24	26-35	Below Lower Threshold	None	0	Sleeping Cat Face
24	20-35	Within Range	Human	0	Info
24	20-35	Within Range	Cat	27	Awake Cat Face
24	20-35	Within Range	Human&Cat	27	Info
24	20-35	Within Range	None	0	Sleeping Cat Face
24	20-22	Above Upper Threshold	Human	0	Info
24	20-22	Above Upper Threshold	Cat	100	Awake Cat Face
24	20-22	Above Upper Threshold	Human&Cat	100	Info
24	20-22	Above Threshold	None	0	Sleeping Cat Face

While the performance of the integrated system met expectations, there were delays of several seconds due to the low FPS. To improve the FPS, several changes were made in the code. Since the delay slowed down the detection rate, some not necessarily delay functions inside the loop were removed. The FPS increased to around 0.5 for most of the time.

The tasks of temperature sensor polling, fan control, and LCD update were considered the main reasons for slowing down the FPS. This was evident when comparing object recognition testing, the FPS on a single system was faster than the FPS on this integrated system. To ensure the guess, a counter for the temperature sensor, fan control, and LCD were set to reduce the frequency. When the counter was in place, FPS increased from around 0.2 to approximately 1. However, once the value of the counter reached its threshold, the FPS decreased. Especially the sensor polling was affected the most, the FPS

decreased from around 1 back to around 0.2 again. The results confirmed the guess.

To have a smooth system performance, the counter for sensor pooling was set to 60 which was about 1 minute delay. While this slightly affected reaction time, the impact was minimal. To minimize the impact of counters on the system's responsiveness for other tasks, conditions were set to reduce the frequency of fan control, and LCD updates without significantly affecting system performance. The first condition was to compare the current temperature to the previous temperature. The fan control will not progress unless the absolute value of the difference between the current temperature and the previous temperature is greater than 1. For the LCD, a flag was implemented: The flag is set to 0 for no object is detected and to 1 for cats are detected. The LCD only updates if there is a change in the flag. For the case of humans, instead of using a flag, the changes in temperature or fan speed were set as the condition. This operation prevents repetition when the content remains unchanged.

Figure 27 below represents the final performance of the object recognition system on Raspberry Pi 3 after the modification of the code. With the same functionality outcomes as before, the FPS improved to around 1 most of the time, representing a significant improvement compared to previous performance.

Figure 27. Performance of object recognition system on Raspberry Pi 3 after modification.

5.4 Discussion

This section evaluates the performance of the trained model and the systems, addressing the challenges encountered in this project and exploring potential improvements.

5.4.1 Summary of Performance

The object recognition model, based on the SSD MobileNetV2 architecture, successfully located and recognized objects, cats, and humans in all the images inside the test folder using bounding boxes with labels and high confidence scores for the early test after training, and achieved an average mean Average Precision (mAP) of 80.83% at various IoU and 97.55% when IoU threshold at 0.5 on the test dataset. This indicates that the model has been well-trained with high accuracy in identification and classification for tasks of object recognition.

The results of object recognition system testing demonstrated that the model was successfully deployed on the Raspberry Pi 3, achieving high accuracy in object

identification and classification tasks under various challenging conditions, including low light, motion, and backlighting. Although the Frames Per Second (FPS) was respectively low, around 0.8 to 0.9, the system still performed as expected with high accuracy.

The performance of temperature-based fan control system testing presented that the system successfully managed tasks such as fan control and LCD adjustments based on varying conditions including detected objects and real-time temperature status. When the object recognition system identified the objects, the temperature-based fan control system adjusted the fan speed accordingly to maintain a comfortable environment for the cat. The LCD displayed necessary information for humans. This dynamic functionality fully met the requirements for the project.

The results of integrated system testing demonstrated that the integration of both systems was successful. The integrated system could detect objects with high accuracy, comparable to the standalone object detection system. Once the objects were detected, the integrated system responded quickly with actions based on specified conditions. However, the FPS in integrated systems was much slower than the FPS in the standalone object detection system. The object detection display was significantly delayed. The slow FPS may decrease the accuracy of object recognition, leading to incorrect actions and difficulties in tracking fast movements, especially for pets like cats, which move quickly. However, after the improvement in the code, the FPS increased from around 0.2 to around 1 for most of the time, and the integrated system performed as expected. Overall, the integrated system demonstrated strong performance, effectively achieving its objectives in object detection and temperature-based fan control.

5.4.2 Challenges

Starting from theoretical studies, and data collection until the testing of the integrated system, this project had a long-term process, and there are several challenges across various phases.

Despite the abundance of machine learning sources for object recognition, understanding numerous new technologies in a limited time was challenging. Some technologies lacked detailed instruction of the background theories, which complicated the learning process.

The data collection phase was hard since cats are living animals, and they have quick movement, it was not easy to collect a large number of images. After the data collection phase, the labelling phase presented another difficulty. To obtain a well-trained model with high accuracy in deep learning, numerous images were required, which required each object within these images must be labelled individually by drawing bounding boxes manually. In the model training phase, Google Colab has limitations in training time and storage for regular users without telling the details of them. It was difficult to know the exact limitation of the training time without encountering it during training. Once the limits were reached, the training process had to be restarted. The version compatibility of software presented significant challenges as well because some newer versions of the software were not compatible with Raspberry Pi systems.

5.4.3 Potential Improvements

While the system demonstrates strong performance with high accuracy in classification and identification tasks, it still makes mistakes sometimes, such as mistaking a yellow sweater or a plush doll for a cat. To solve this problem, several steps can be improved.

In the step of data collection, more images of objects can provide more learning materials for the training process. It can potentially improve the accuracy of object

recognition tasks. Adding more classes for other objects that are similar to cats and humans may be another solution.

On the step of model training, more steps of training can also potentially increase the accuracy of the model. The more steps required more time for training.

Another issue was that the FPS was slow because the Raspberry Pi 3 itself had a relatively low-powered CPU and insufficient RAM. To solve this problem, a newer version with a high-powered CPU and sufficient RAM or adding a Google Coral USB accelerator is the potential solution.

For temperature-based fan control tasks, a small DC fan is insufficient to make a significant change in indoor temperatures. An air conditioner with both cooling and warming functions is the ideal solution for this project.

6 Conclusion

This project aimed to develop an integrated system for object recognition and temperature-based fan control, focusing on training an efficient model for object recognition, implementing a responsive temperature-based fan control system, and evaluating their combined performance. By deploying Artificial Intelligence (AI) technologies, the project intended to enhance automation technologies.

The customized model for object recognition was successfully trained using Google Colab with a pre-trained TensorFlow model called SSD-Mobile-v2(FP32). The trained object recognition model achieved a mean average precision (mAP) score of 80.83% across various Intersections over Union (IoU) thresholds and 97.55% at a 0.5 IoU threshold. Indicating the high accuracy in object recognition.

The trained object recognition model had been deployed into Raspberry Pi 3 and it successfully detected humans and cats with high confidence scores and bounding boxes surrounding each object under various challenging conditions,

including low light, motion, and backlighting. The FPS was around 0.9 which was slightly slow, but the slow FPS did not affect the performance in object recognition.

The temperature-based fan control system included a DHT11 sensor for temperature sensing, a DC fan to provide comfort temperature for cats, and an LCD to display information such as current temperature and fan speed for humans. The temperature-based fan control system completed the tasks based on the user input and the temperature status.

The integration of an object recognition system and temperature-based fan control system was also successful. The integrated system, utilizing the sensor platform PCB board and 3D enclosure, completed the tasks with a satisfactory FPS based on the detected objects and the temperature status. The successful results indicated that integrating to temperature control system with an air conditioner.

Since automation appears everywhere in daily life, integrating it with object recognition makes systems smarter and more efficient. Such a system can perform various tasks based on the objects it detects. This thesis project demonstrated the feasibility of integrating a machine learning-based object recognition system into a basic temperature-based fan control system through a low-cost solution. The satisfactory performance demonstrated by the project indicated that integrating object recognition into a temperature control system with an air conditioner is achievable, also providing insight into customized object recognition systems on edge devices for other automation technologies. This approach makes intelligent object recognition more accessible and practical applications in smart home automation. It contributes to the field of smart home technology, innovative products for humans, pets, and other objects, and environmental control systems while considering sustainability through energy efficiency.

References

1 Mustafa Suleyman. How the AI Revolution Will Reshape the World [Internet]. TIME; [updated 2023 Sep 1; cited 2024 Nov 10]. Available from: https://time.com/6310115/ai-revolution-reshape-the-world/.

2 As a population gets older, automation accelerates [Internet]. ScienceDaily; 2021 [cited 2024 Nov 10]. Available from: https://www.sciencedaily.com/releases/2021/09/210916114542.htm#google_vignette.

3 Jelena Smiljkovic. What is AI Automation? Benefits, Use Cases, and How It Works [Internet]. automatio; 2024 [cited 2024 Nov 10]. Available from: https://automatio.ai/blog/ai-automation/.

4 Shelly Volsche. New research suggests cat and dog 'moms' and 'dads' really are parenting their pets – here's the evolutionary explanation why [Internet]. THE CONVERSATION; 2021 [cited 2024 Nov 10]. Available from: https://theconversation.com/new-research-suggests-cat-and-dog-moms-and-dads-really-are-parenting-their-pets-heres-the-evolutionary-explanation-why-169600.

5 Global Pet Industry To Grow To $500 Billion By 2030, Bloomberg Intelligence Report Finds [Image on the internet]. Bloomberg; 2023 [cited 2024 Nov 10]. Available from: https://www.bloomberg.com/company/press/global-pet-industry-to-grow-to-500-billion-by-2030-bloomberg-intelligence-finds/.

6 Temperature Controllers [Internet]. OMRON; [cited 2024 Nov 10]. Available from: https://www.ia.omron.com/support/guide/53/introduction.html.

7 What is artificial intelligence (AI)? [Internet]. IBM; [cited 2024 Nov 3]. Available from: https://www.ibm.com/topics/artificial-intelligence.

8 What is machine learning (ML)? [Internet]. IBM; [cited 2024 Nov 3]. Available from: https://www.ibm.com/topics/machine-learning.

9 Yakoove. Fig-X: All ML as a Subfield of AI [Image on the internet]. Wikimedia Commons; 2020 [updated 2022 August 19; cited 2024 Nov 3]. Available from: https://commons.wikimedia.org/wiki/File:Fig-X_All_ML_as_a_subfield_of_AI.jpg.

10 Introduction to Deep Learning [Internet]. GeeksforGeeks; [updated 2024 May 26; cited 2024 Nov 3]. Available from: https://www.geeksforgeeks.org/introduction-deep-learning/.

11 Aya Selmoune. Illustration of neural network architecture [Image on the internet]. ResearchGate; 2022 [cited 2024 Nov 3]. Available from:

https://www.researchgate.net/figure/Illustration-of-neural-network-architecture_fig2_363717459.

12 Artificial Neural Networks and its Application [Internet]. GeeksforGeeks; [cited 2024 Nov 3]. Available from: https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/.

13 HK Lam. Multilayer Neural Networks – Part 2: Feedforward Neural Networks Example [Video]. YouTube; 2021 [cited 2024 Nov 3]. Available from: https://www.youtube.com/watch?v=QYMsvtMMS-g&t=2s.

14 Understanding Multi-Layer Feed Forward Networks [Internet]. GeeksforGeeks; [updated 2021 Oct 16; cited 2024 Nov 3]. Available from: https://www.geeksforgeeks.org/understanding-multi-layer-feed-forward-networks/.

15 Moez All. Introduction to Activation Functions in Neural Networks [Internet]. DataCamp; [updated 2024 Sep 12; cited 2024 Nov 4]. Available from: https://www.datacamp.com/tutorial/introduction-to-activation-functions-in-neural-networks?dc_referrer=https%3A%2F%2Fwww.bing.com%2F.

16 Sefik Likin Serengil. ReLU [Image on the internet]. Sefik Likin Serengil's blog; 2018 [cited 2024 Nov 4]. Available from: https://sefiks.com/2018/05/31/an-overview-to-gradient-vanishing-problem/.

17 Jason Brownlee. How to Choose an Activation Function for Deep Learning [Internet]. MACHINE LEARNING MASTERY; 2021 [cited 2024 Nov 4]. Available from: https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/.

18 Piyush Kashyap. Understanding the Dying ReLU Problem and its Variants in Neural Networks [Internet]. Medium; [cited 2024 Nov 4]. Available from: https://medium.com/@piyushkashyap045/understanding-the-dying-relu-problem-and-its-variants-in-neural-networks-e94200fbc117#:~:text=The%20Dying%20ReLU%20problem%20occurs%20when%20neurons%20consistently,die%2C%20the%20less%20the%20network%20can%20effectively%20learn.

19 Devin Schumacher. ReLU6 [Internet]. SERP AI; [cited 2024 Nov 4]. Available from: https://serp.ai/relu6/#:~:text=ReLU6%20limits%20the%20output%20of%20the%20ReLU%20function,defined%20as%20f%20%28x%29%20%3D%20min%20%28max%20%280%2Cx%29%2C6%29.

20 Logistic Regression [Image on the internet]. Angelo's Math Notes; 2020 [cited 2024 Nov 4]. Available from: https://angeloyeo.github.io/2020/09/23/logistic_regression_en.html.

21 What are convolutional neural networks? [Internet]. IBM; [cited 2024 Nov 4]. Available from: https://www.ibm.com/topics/convolutional-neural-networks#:~:text=Convolutional%20neural%20networks%20use%20three-dimensional%20data%20for%20image.

22 Hachim Ei Khiyari, Harry Wechsler. Age Invariant Face Recognition Using Convolutional Neural Networks and Set Distances [Image on the internet]. Scientific Research; 2017 [cited 2024 Nov 4]. Available from: https://www.scirp.org/journal/PaperInformation?PaperID=77647.

23 David Batista. Convolutional Neural Networks for Text Classification [Image on the internet]. David Batista's blog; 2018 [cited 2024 Nov 4]. Available from: https://www.davidsbatista.net/blog/2018/03/31/SentenceClassificationConvNets/.

24 Jason Brownlee. A Gentle Introduction to Pooling Layers for Convolutional Neural Networks [Internet]. MACHINE LEARNING MASTERY; 2019 [cited 2024 Nov 4]. Available from: https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/.

25 What Is Mobilenet V2? [Internet]. GeeksforGeeks; [Updated 2024 Jul 5; cited 2024 Nov 13]. Available from: https://www.geeksforgeeks.org/what-is-mobilenet-v2/.

26 Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks [Internet]. Cornell University; 2019 [cited 2024 Nov 4]. Available from: https://arxiv.org/pdf/1801.04381.

27 Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, Serge Belongie. Feature Pyramid Networks for Object Detection [Internet]. Cornell University; 2017 [cited 2024 Nov 4]. Available from: https://arxiv.org/pdf/1612.03144.

28 [CV] Detection [Image on the internet]. LucasX; 2019 [cited 2024 Nov 13]. Available from: https://lucasxlu.github.io/blog/2019/07/20/cv-detection/.

29 Federico Furlán, Elsa Rubio, Humberto Sossa, Víctor Ponce. CNN Based Detectors on Planetary Environments: A Performance Evaluation [Image on the internet]. fronties; [cited 2024 Nov 4]. Available from: https://www.frontiersin.org/journals/neurorobotics/articles/10.3389/fnbot.2020.590371/full#B13.

30 Neha Vishwakarma. Real-Time Object Detection with SSDs: Single Shot MultiBox Detectors [Internet]. Analytics Vidhya; [updeted 2023 Nov 17; cited 2024 Nov 4]. Available from: https://www.analyticsvidhya.com/blog/2023/11/real-time-object-detection-with-ssds-single-shot-multibox-detectors/.

31 Eddie Forson. Understanding SSD MultiBox — Real-Time Object Detection In Deep Learning [Internet]. Medium; 2017 [cited 2024 Nov 4]. Available from: https://towardsdatascience.com/understanding-ssd-multibox-real-time-object-detection-in-deep-learning-495ef744fab.

32 Lei Mao. Use Focal Loss To Train Model Using Imbalanced Dataset [Internet]. GitHub; 2019 [cited 2024 Nov 4]. Available from: https://leimao.github.io/blog/Focal-Loss-Explained/.

33 Pratyaksha Jha. A Brief Overview of Loss Functions in Pytorch [Internet]. Medium; 2019 [cited 2024 Nov 4]. Available from: https://medium.com/udacity-pytorch-challengers/a-brief-overview-of-loss-functions-in-pytorch-c0ddb78068f7.

34 Raimi Karim. Intuitions on L1 and L2 Regularisation [Internet]. Medium; 2018 [cited 2024 Nov 4]. Available from: https://towardsdatascience.com/intuitions-on-l1-and-l2-regularisation-235f2db4c261.

35 Kukil. Mean Average Precision (mAP) in Object Detection [Internet]. LearnOpenCV; 2022 [cited 2024 Nov 4]. Available from: https://learnopencv.com/mean-average-precision-map-object-detection-model-evaluation-metric/.

36 Nguyen Van-Khoa. Large-Scale Object Detection for Social Media User's Visual Privacy Protection [Image on the internet]. ResearchGate; 2020 [cited 2024 Nov 4]. Available from: https://www.researchgate.net/figure/Recall-Precision-Curve_fig5_347530551.

37 Jalaj Agrawal. Mean Average Precision (mAP) Explained in Object Detection [Internet]. Medium; 2022 [cited 2024 Nov 4]. Available from: https://medium.com/@jalajagr/mean-average-precision-map-explained-in-object-detection-fb61adf67ef4.

38 Evan Juras. TensorFlow Lite Object Detection Model Performance Comparison [Internet]. EJ TECHNOLOGY; 2022 [cited 2024 Nov 4]. Available from: https://www.ejtech.io/learn/tflite-object-detection-model-comparison.

39 Grove – 16x2 LCD [Internet]. seeed studio; [cited 2024 Nov 4]. Available from: https://wiki.seeedstudio.com/Grove-16x2_LCD_Series/.

40 Object Detection on PASCAL VOC 2007 [Internet]. Meta AI; [cited 2024 Nov 10]. Available from: https://paperswithcode.com/sota/object-detection-on-pascal-voc-2007.

Pre-trained Model

Listing 1. ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.config[https://github.com/tensorflow/models/blob/master/research/object_detection/configs/tf2/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.config]

```
model {
  ssd {
    inplace_batchnorm_update: true
    freeze_batchnorm: false
    num_classes: 90
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
    matcher {
      argmax_matcher {
        matched_threshold: 0.5
        unmatched_threshold: 0.5
        ignore_thresholds: false
        negatives_lower_than_unmatched: true
        force_match_for_each_row: true
        use_matmul_gather: true
      }
    }
    similarity_calculator {
      iou_similarity {
      }
```

```
}
encode_background_as_zeros: true
anchor_generator {
  multiscale_anchor_generator {
    min_level: 3
    max_level: 7
    anchor_scale: 4.0
    aspect_ratios: [1.0, 2.0, 0.5]
    scales_per_octave: 2
  }
}
image_resizer {
  fixed_shape_resizer {
    height: 320
    width: 320
  }
}
box_predictor {
  weight_shared_convolutional_box_predictor {
    depth: 128
    class_prediction_bias_init: -4.6
    conv_hyperparams {
      activation: RELU_6,
      regularizer {
        l2_regularizer {
          weight: 0.00004
        }
      }
      initializer {
        random_normal_initializer {
          stddev: 0.01
          mean: 0.0
        }
      }
```

```
      batch_norm {
        scale: true,
        decay: 0.997,
        epsilon: 0.001,
      }
    }
    num_layers_before_predictor: 4
    share_prediction_tower: true
    use_depthwise: true
    kernel_size: 3
  }
}
feature_extractor {
  type: 'ssd_mobilenet_v2_fpn_keras'
  use_depthwise: true
  fpn {
    min_level: 3
    max_level: 7
    additional_layer_depth: 128
  }
  min_depth: 16
  depth_multiplier: 1.0
  conv_hyperparams {
    activation: RELU_6,
    regularizer {
      l2_regularizer {
        weight: 0.00004
      }
    }
    initializer {
      random_normal_initializer {
        stddev: 0.01
        mean: 0.0
      }
```

```
    }
    batch_norm {
      scale: true,
      decay: 0.997,
      epsilon: 0.001,
    }
  }
  override_base_feature_extractor_hyperparams: true
}
loss {
  classification_loss {
    weighted_sigmoid_focal {
      alpha: 0.25
      gamma: 2.0
    }
  }
  localization_loss {
    weighted_smooth_l1 {
    }
  }
  classification_weight: 1.0
  localization_weight: 1.0
}
normalize_loss_by_num_matches: true
normalize_loc_loss_by_codesize: true
post_processing {
  batch_non_max_suppression {
    score_threshold: 1e-8
    iou_threshold: 0.6
    max_detections_per_class: 100
    max_total_detections: 100
  }
  score_converter: SIGMOID
}
```

```
  }
}

train_config: {
  fine_tune_checkpoint_version: V2
  fine_tune_checkpoint:
"PATH_TO_BE_CONFIGURED/mobilenet_v2.ckpt-1"
  fine_tune_checkpoint_type: "classification"
  batch_size: 128
  sync_replicas: true
  startup_delay_steps: 0
  replicas_to_aggregate: 8
  num_steps: 50000
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    random_crop_image {
      min_object_covered: 0.0
      min_aspect_ratio: 0.75
      max_aspect_ratio: 3.0
      min_area: 0.75
      max_area: 1.0
      overlap_thresh: 0.0
    }
  }
  optimizer {
    momentum_optimizer: {
      learning_rate: {
        cosine_decay_learning_rate {
          learning_rate_base: .08
          total_steps: 50000
          warmup_learning_rate: .026666
```

```
          warmup_steps: 1000
        }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
}

train_input_reader: {
  label_map_path: "PATH_TO_BE_CONFIGURED/label_map.txt"
  tf_record_input_reader {
    input_path:  "PATH_TO_BE_CONFIGURED/train2017-?????-of-
00256.tfrecord"
  }
}

eval_config: {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
}

eval_input_reader: {
  label_map_path: "PATH_TO_BE_CONFIGURED/label_map.txt"
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path:    "PATH_TO_BE_CONFIGURED/val2017-?????-of-
00032.tfrecord"
  }
}
```

Contents on LCD Display

Figure 1. Sleeping cat face on LCD.

Figure 2. Awake cat face on LCD.

Figure 3. Information for humans when a cat is absent, or the sensing temperature is below the lower threshold.

Figure 4. Information for humans when a cat is present and sensing temperature is within the set thresholds range.

Physical Implementation

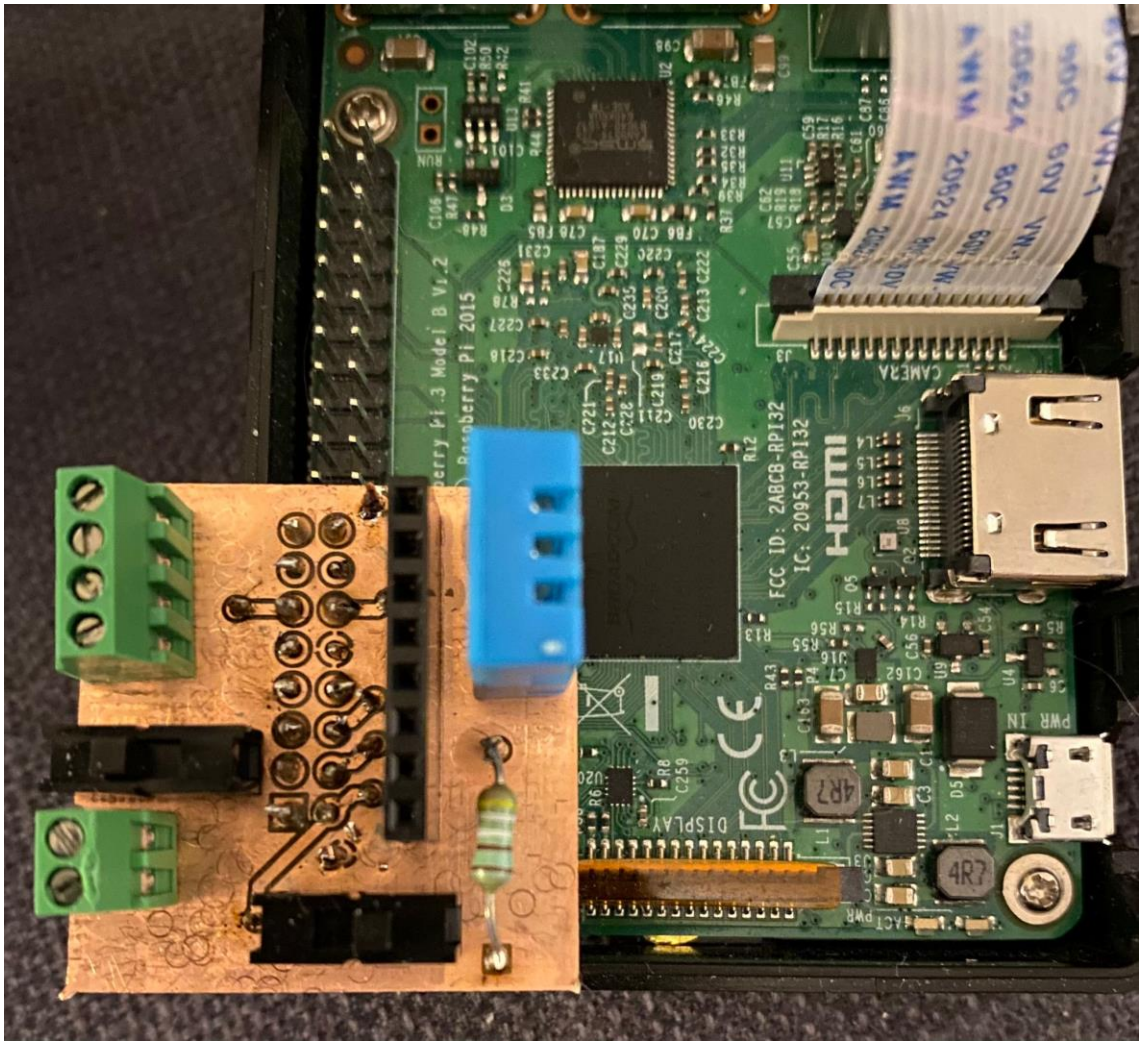

Figure 1. Physical implementation of sensor platform attached to the Raspberry Pi 3.

Figure 2. Physical implementation of the enclosure.