Real-time Object Detection Using IoT Devices: Detecting Rodents and Birds for Conservation

A thesis presented in partial
fulfilment of the requirements

for the degree of

Master of Science in Computer Sciences

at Massey University, Auckland, New Zealand

Kyuwon Shim

2021

Contents

List of Figures

List of Tables

Abstract

In New Zealand forests rodents currently threatens various species of native birds. Unfortunately, the traditional traps and poisoning are not sufficient to eliminate an increasing number of rodents in our forests. In this work we developed an auxiliary control tool that combines the latest computer technology, such as object recognition algorithms, with Internet of Things(IoT) devices. A drawback of most object recognition algorithms is that it requires a tremendous amount of computation, so these techniques are not necessarily adequate to be applied to IoT devices.

This thesis proposes to create a real-time object detection algorithm using IoT devices for two classes of objects, rodents and birds. Two methods were implemented and tested. The first method is a shape-based algorithm using Fourier Descriptors (FD). In order to find the ideal classification algorithm for the FD method, we compared the accuracies of four different classification algorithms: Neural Network, Random Forest, AdaBoost, and Support Vector Machine(SVM).

The second method is based on YOLO (You Only Look Once). As YOLO requires a huge amount of images to train well, we created a semi-automated labelling system in our detection algorithm to boost the number of images for training. The labelling system produces the training requirements for both Yolo and FD and promises much faster processing time than manual labelling work.

The labelling system succeeded in obtaining the labelled data of 2172 rodents and 3494 birds. Using a desktop machine and the FD method, and the average accuracy rate was 80% when using the Random Forest classifier. Using a desktop machine and the YOLO method, the average accuracy rate was 97%. The frame rates were 19 fps for the FD, 3.7 fps for the YOLO on CPUs and 90.9 fps for YOLO running on a GPU.

Using a Raspberry Pi 3 and a video dataset, the FD method resulted in an 83% general accuracy and a frame rate of 6.33 fps, which is 30% more accurate and 21 times faster than YOLO. The experiments showed the feasibility of using the FD method in Raspberry Pis for real-time detection, with performance advantages compared to the YOLO architecture.

Acknowledgements

I would like to express my special thanks to my supervisors Dr. Andre Barczak and Dr. Napoleon Reyes for their significant supports in accomplishing this thesis. Also, a special thanks to Dr. Nasim Ahmed for guiding and encouraging the writing of the thesis. I would also like to thank the online communities for sharing the method of using a wide range of applications and tools that saves tremendous time in research, especially implementing OpenCV and YOLO version 5 in the Raspberry Pi 3 machine. The tools and applications include OpenCV library, Google Colab, Gnuplot, and LaTeX.

Chapter 1

Introduction

1.1 The Basic Idea

Some native bird species in New Zealand are on the verge of extinction because of the expansion of rodent breeding in the wild field, especially Norway rats(Rattus norvegicus) and ship rats (R. rattus) [17]. In the early 1960s, the ship rat invaded South Island, and they devastated habitats of more than six native bird species [18]. Vertebrate pesticides are widely utilized to protect native birds' nest from predators in New Zealand's forests. These pesticides are more effective than trapping, but it has a high risk of death to the native birds and other mammals that are not considered pests [17].
Regardless of the type of trap, they can be effective in capturing rodents, but the growth rate of the rodent population can overcome trapping very rapidly. There are limitations to use of traditional trapping because of the difficulties of the terrain. That is why poisoning by helicopter is so widespread in New Zealand. As an additional tool for monitoring pests in our forests, we come up with the idea of a detector of rodents and birds using a portable computer (Raspberry Pi) that can communicate via the Internet. This detector can monitor and potentially activate a physical trap using object recognition technology in computer vision. Thus, multiple rodents can be detected without any manual reset. Our objective is to develop a real-time detection algorithm that recognizes rodents and birds in a low specification machine. There are various algorithms for object detection, but those usually require higher specification machines. Therefore, designing a new detection algorithm for Raspberry Pi, which is an example of low specification machines, might be a helpful tool to add the the current toolbox used in our forests.

1.2 Research Objectives

1. Find the ideal image datasets for this project and investigate forms of automatically label images.

2. Investigate the requirements of labelling in Fourier Descriptors (FD) and YOLO (You Only Look Once) methods.

3. Create a system that can do quickly labelling for training data.

4. Analyse the accuracy and speed of classifiers trained by FD and YOLO.

5. Implementing our detection code in Raspberry Pi.

6. Make a comparison of the accuracy and the performance of FD and YOLO in Raspberry Pis.

1.3 Significance of the Research

Most YOLO detection research involves the detection of new types of object and measurement of speed and accuracy. In this research, YOLO will be used to compare to our detection algorithm (FD). This research also contributes to the creation of a semi-automatic labelling system, a new object detection algorithm working in real-time, and assessing the performance and speed of our algorithm (FD) and YOLO in a Raspberry Pi. We proposed a new algorithm using Fourier Descriptors, named FD, that can run on low specification machines, such as a Raspberry Pi, in order to design different forms of traps by detecting predators in the natural habitat.

Briefly describing the results, in a desktop machine the FD achieved an accuracy of 80% using a static dataset, and the YOLO achieved 97% accuracy in the same dataset. The frame rates were 19 fps for the FD, 3.7 fps for the YOLO on CPUs and 90.9 fps for YOLO running on a GPU.

the FD algorithm showed a rate of 6.33 fps and 83% accuracy in Raspberry Pi, which is 21 times faster and 30% accurate than Yolo version 5.

1.4 Scope and limitations

1.4.1 Scope

The proposed algorithm FD and YOLO (You Only Look Once) has been trained by a collection of more than five thousand labelled images and tested around a thousand images. Also, videos to test the algorithms in real-time were created. Both algorithms in FD and YOLO are implemented in Raspberry Pi3 for comparing the accuracy and speed.

1.4.2 limitations

The Limitations are as follows:

1. Real-time object detection in both FD and YOLO were tested in videos generated inside the computer laboratory.
2. In the real-time videos, the sample objects in real-time detection used the silhouettes of rodents and birds.
3. The camera needs to be static.
4. The object positions should be dynamic (moving targets).
5. Our algorithm (FD) can detect only one object at a time.

1.5 Structure of the Thesis

The thesis is structured as follows. Chapter 2 discusses the related work, including similar approaches and the theoretical discussion about various algorithms (e.g., filters). In chapter 3, we show how the proposed algorithm works. In chapter 4 the semi-automatic labelling system is shown, and the YOLO approach is discussed. The results are presented in chapter 5, and a discussion of the results follows. Finally, chapter 6 concludes the thesis. Additional information about the videos is discussed in the appendix.

Chapter 2

Literature Review

2.1 Related Works

Recently, there are many IoT (Internet of Things) works that directly use computer vision for some type of wild life monitoring. Rendall et al.[19] used a camera set on a pole to monitor different kinds of rodents. Ross et al.[20] built a trap system with a computer vision application called RatSpy, a visual and low-power bait monitoring system. Krynitsky et al.[21] developed the RAT (Rodent Arena Tracker) system using a machine vision camera, which is the OpenMV Cam M7 (Arduino machine vision camera), open-source, and costs about $120. This system can analyse the psychological state of rodents using transmitted data of their movement from RAT in real-time [21]. Gerós et al.[22] researched rodents' behavioural activity using low-cost depth cameras. They invented auto-classification, spending time in under 30 minutes with annotated videos, and emphasised that a system is an easy-to-use tool, low-cost to install, and able to use in dark conditions [22]. These researchers focus on easy-to-install and low-cost machines.

Also, there is an attempt to apply a deep neural network in a low specification machine. Rosebrock [23] implemented the YOLO (You Look Only Once) technique in a Raspberry Pi with Movidius NCS, which is the vision processing unit technology and can process deep neural networks (DNNs). The fastest architecture in YOLO (Tiny-YOLO) is capable of obtaining around 4.28 fps (frames per second) in a Raspberry Pi with Movidius NCS[23]. This speed is moderate, and this contribution might be sufficient to implement object detection in various fields of research.

Furthermore, there are some efforts to develop a rodent trapping system. The Cacophony Project is a not-for-profit organization and aims to eradicate pests by applying new technologies [1]. This organization makes some products related to their monitoring, including various monitoring devices, open-source software, and a thermal camera using Raspberry Pi, to expand their project. Their project is designed to protect the bird's habitat from predators using their invented device named the Cacophonator (figure 2.2a),

which is the Raspberry Pi machine equipped with a thermal camera, sensors, and speakers [2]. This device automatically identifies the object and proposes to eliminate invasive predators by using the device as trap activator in New Zealand's native habitat [2]. In addition, this organization built up an application programming interface (API) server to communicate the Cacophonator with the monitoring device in figure (2.2b) and web server in figure (2.1). The preparation of this project collects the appearing predators of birds in the targeted area using the Cacophonator, and creates a classifier using collected data, and then installs an appropriate lure inside the trap to attract the predators [1]. The trap generates the command from the Cacophonator whether the detected object is a predator or not [1]. Using this method, Hampshire and Sapphire [1] classified the various detected objects as possums, rodents, mustelids, cats, hedgehogs, and birds, then obtained the general trap interaction rate as 32.4%, excluding unknown objects. This report was recently published on 27th May 2021 [1], and this trap system is still in progress.

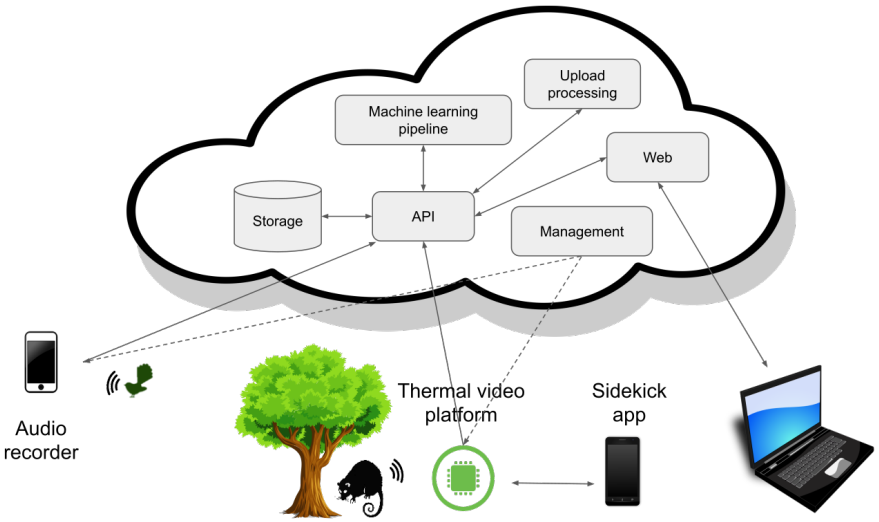


Figure 2.1: The concept of communication between the Cacophonator and others [1].

(a) The Cacophonator. (b) The bird monitor device.

Figure 2.2: The devices for monitoring predators [2].

2.2 Raspberry Pi

The Raspberry Pi is a portable Linux base machine that supports various sensor tools at a relatively economical price, and it allows users to communicate with this machine over the internet from a long distance. There are some advantages of the Raspberry Pi. The first is that this machine has good portability. The CPU, named BCM2835, does not consume much battery power [24], and this cheap battery requirement makes it possible to use a power bank with the Raspberry Pi. Therefore, the Raspberry Pi can be installed anywhere with a 5 volts power supply. The second is that the Raspberry Pi is manufactured at an outstandingly low price. Richardson et al. [25] emphasize that the low cost of Raspberry Pi is the main reason used in various fields, and the manufacturer supplies this machine without a profit margin: the manufacturer offers this machine from US$25 to US$35, and the resellers arranged the machine prices between US$35 and US$40. The last advantage is that it is a Linux based machine. The operating system of the Raspberry Pi is Raspbian which is based on Debian and is Linux compatible [26]. Raspbian has great compatibility with Ubuntu, which is also based on Debian and the most popular Linux operating system. This feature allows that programs in different environments and high specification machines as laptop and desktop can be ported to the Raspberry Pi. The Raspberry pi machine is shown in figure 2.3.

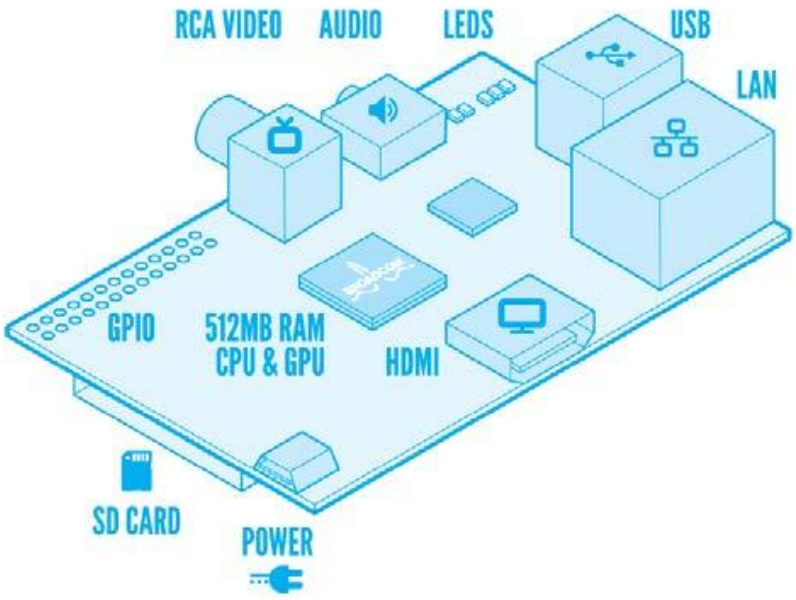


Figure 2.3: The components of the Raspberry Pi [3].

2.2.1 Data Transfer in Raspberry Pi

Raspberry Pi 3 requires the micro SD card to boot because it does not have the internal storage or built-in operating system [27]. The capacity of the micro SD card is limited, so a system of transferring the captured data into other big storage machines is required. The Raspberry Pi 3 supports a wireless internet connection, and it might be the solution for sending data from the Raspberry Pi to the main machine. The captured data can be transferred through SSH, which is a highly effective and multi-purpose protocol commonly applied in transport layer mechanisms and interactive shells [28]. However, connecting wireless internet in long-distance is one issue because the machine is mostly placed in distant areas. Ismail et al. [29] successfully extended the network in a maximum of 80 meters with a speed of 1.5 Mbps from 100 Mbps LAN using Raspberry Pi 3B+ and its external antenna in a free space environment. Therefore, sending the data through the internet to other machines can help to deal with the lack of memory capacity in the Raspberry Pi, and the external antenna provides more options for the location of installation.

2.2.2 Applying Machine Learning techniques in Raspberry Pi

Despite these advantages, this machine may not be suitable for applying the new technology that has been introduced nowadays due to its low specifications. New machine learning technologies in computer vision have come up with algorithms using a graphics processing unit(GPU), but a Raspberry Pi machine requires an external GPU device for processing a GPU program, and this external device is more expensive than the Raspberry machine itself. Having external GPUs, the battery power consumption is increased dramatically. Therefore, using GPU programming in Raspberry Pi 3 is not efficient in terms of cost and battery consumption.

Also, there is much effort allocating the machine learning program in low specification machines. For instance, YOLO-LITE [30] is the fastest YOLO and it is designed for non-GPU machines with a focus on increasing the speed by reducing layer sizes [31]. Sismananda et al. [31] measured the processing time in the Raspberry Pi using YOLO-LITE, and it showed approximately 2.5 frames per second and around five times faster than version 3, but the YOLO-LITE has 30% less accurate than version 3. It has great success in the point of speed, but this is not feasible in real-time detection due to the lack of accuracy.

2.3 Object Detection with YOLO

2.3.1 Related Techniques: Convolutional Neural Network

Some techniques motivated the creation of You Only Look Once (YOLO), and the most relevant one is Convolutional Neural Networks (CNN). CNN shows great contributions over the past decade related to pattern recognition, such as voice recognition and image processing [4]. Furthermore, Albawi et al. [4] mention that the outstanding achievement of CNN is able to deal with a complex tasks, which requirements are impossible to meet with classical Artificial Neural Networks (ANNs) [32]. Each object has a unique characteristic, and the CNN captures and trains the characteristic patterns of the object. In order to capture the pattern and estimate the location of an object, some filter operations are applied (figure 2.4). In addition, the stride technique is for decreasing parameters and reducing operational time, and pooling is purposed to reduce the complexity for the following layers. Detecting the location of objects from an image in YOLO is based on CNN.

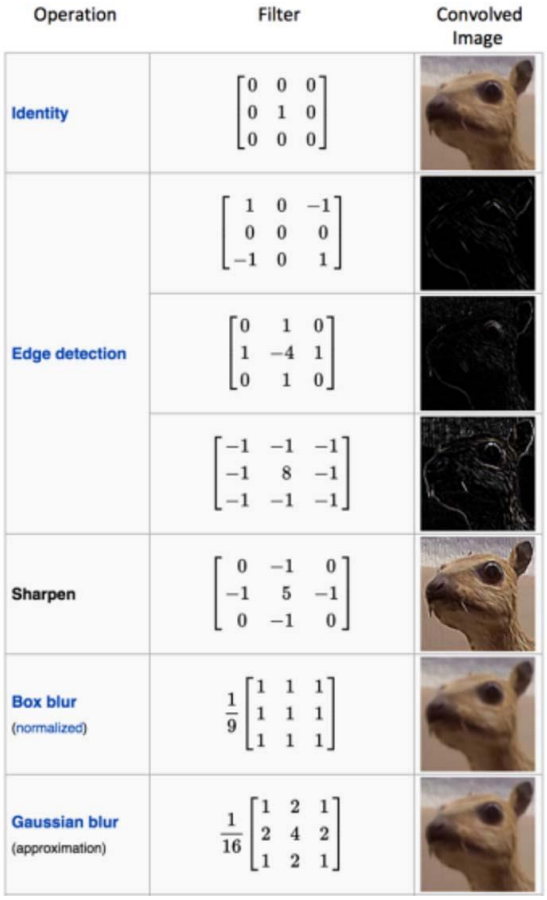


Figure 2.4: The filters in CNN [4].

2.3.2 You Only Look Once (YOLO)

Object detection is widely used in the area of augmented reality, robot navigation, automatic driving, medical diagnosis, etc. Those methods are based on Deep Neural Networks (DNNs), which ensure prominent performance but require sufficient memory and high computational ability [33]. Therefore, this method can only be deployed in constrained environments. In 2015, YOLO was the state-of-the-art in DNN-based object detection and introduced to deal with both accuracy and speed without a high specification machine [9]. YOLO algorithm consists of fully connected CNNs [34]. Redmon et al. [9] mention that YOLO is remarkably fast; the performance shows 150 frames per second on a Titan XGPU with tiny-YOLO-V3, which is the lightest version in YOLO.

2.3.3 Segmentation

YOLO segmentation method is related to Region based convolutional neural networks (R-CNNs) [35], which obtains a prominent accuracy in object detection using deep ConvNet, unlike the sliding window method in figure 2.6. However, there are some drawbacks such as the training is so expensive in space and time because a multi-stage pipeline is applied, and object detection is slow due to heavy cost calculation [36]. Generally, R-CNN extracts various features and feed into CNNs [30] in figure 2.5. YOLO adopts to transform the later process of figure 2.5 of "After initial Segmentation" using unified detection, anchor boxes, and non-max suppression. This transformation in YOLO results in fewer than half the amount of background errors than Fast R-CNN.

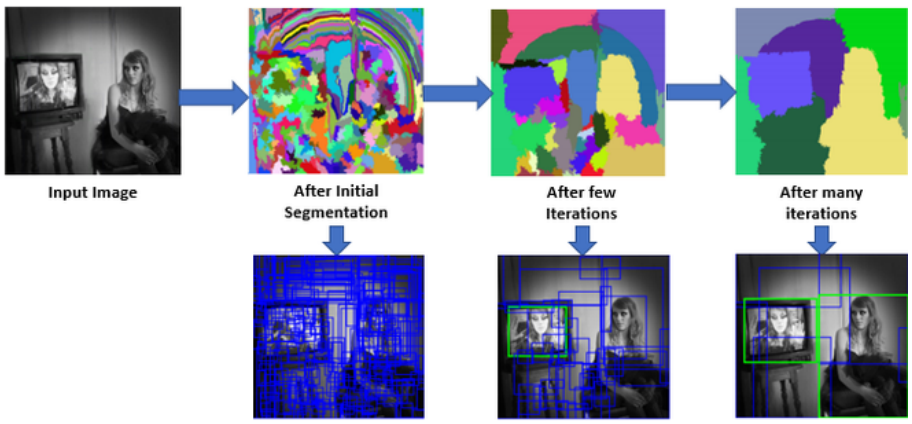


Figure 2.5: The process of segmentation in R-CNN [5].

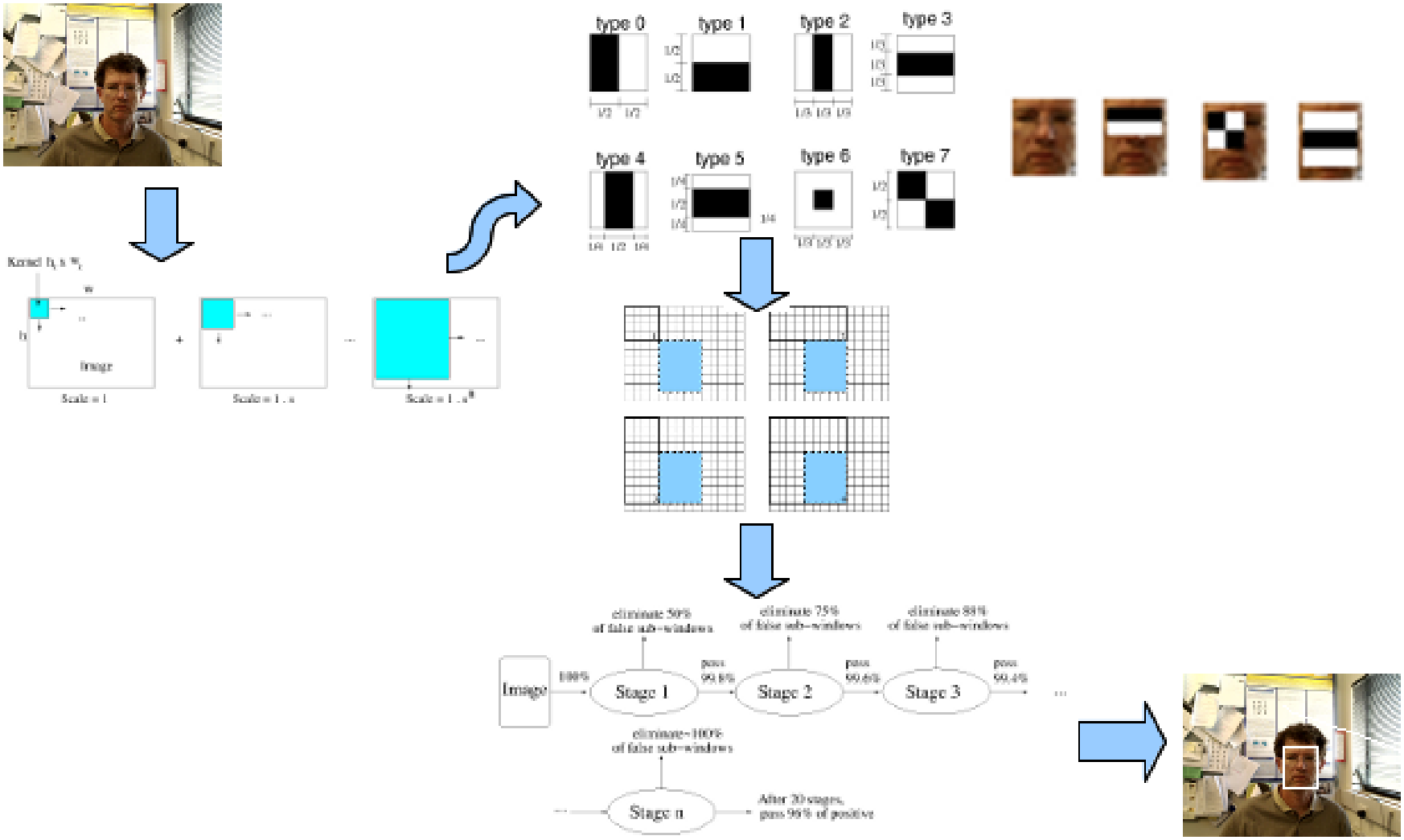


Figure 2.6: Object detection based on sliding window([6], [7]).

2.3.4 Unified Detection

The input image is divided into an X by X grid in YOLO. In case that the centre of an object is placed in a particular grid cell, this grid cell takes responsibility for detecting this object [9]. Moreover, each grid cell predicts B bounding boxes' location and confidence scores related to B bounding boxes' accuracy, confidence scores defined as $Pr(Class_i) \times IOU\binom{truth}{pred}$ in equation 2.2 and is zero when objects do not exist [9]. In addition to the bounding box, there are five components to predict targets: x coordinate, y coordinate, w (width), h (height), and confidence. The x and y coordinates describe the generalised centre point of the boundary box, and the w and h are the generalized width and height of the boundary box in figure 2.7. Moreover, one grid cell predicts only one class, and each grid cell highlights the boundary box, which has relatively high confidence score, such as the example that can be seen in figure 2.8.

$$T = S \times S \times (B \times 5 + C) \tag{2.1}$$

$$Pr(Class|Object) \times Pr(Object) \times IOU\binom{truth}{pred} = Pr(Class_i) \times IOU\binom{truth}{pred} \tag{2.2}$$

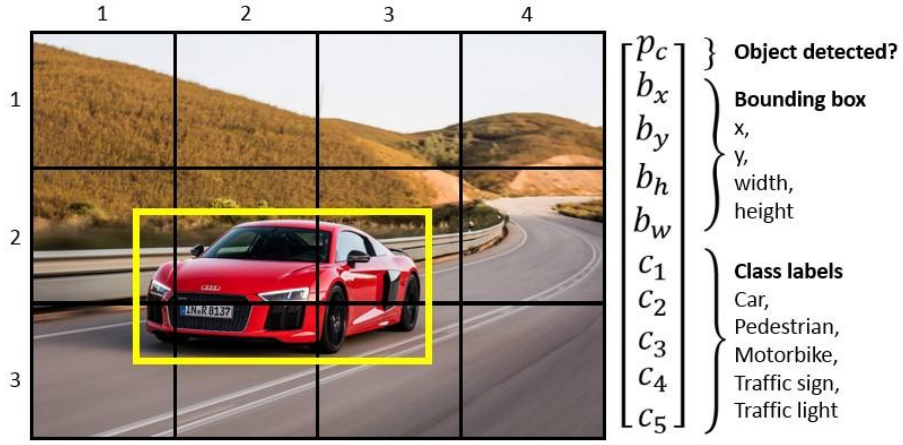


Figure 2.7: Bounding box [8].

2.3.5 Network Design

To assess YOLO on PASCAL VOC [37], which is a prepared dataset for object detection, the YOLO inventor set the values as: S to 7, B to 2, C to 20 [9]. Therefore, the final output tensor is 7 by 7 by 30. In terms of network architecture, the GoogLeNet model for image classification inspired YOLO architecture with two fully connected layers and 24

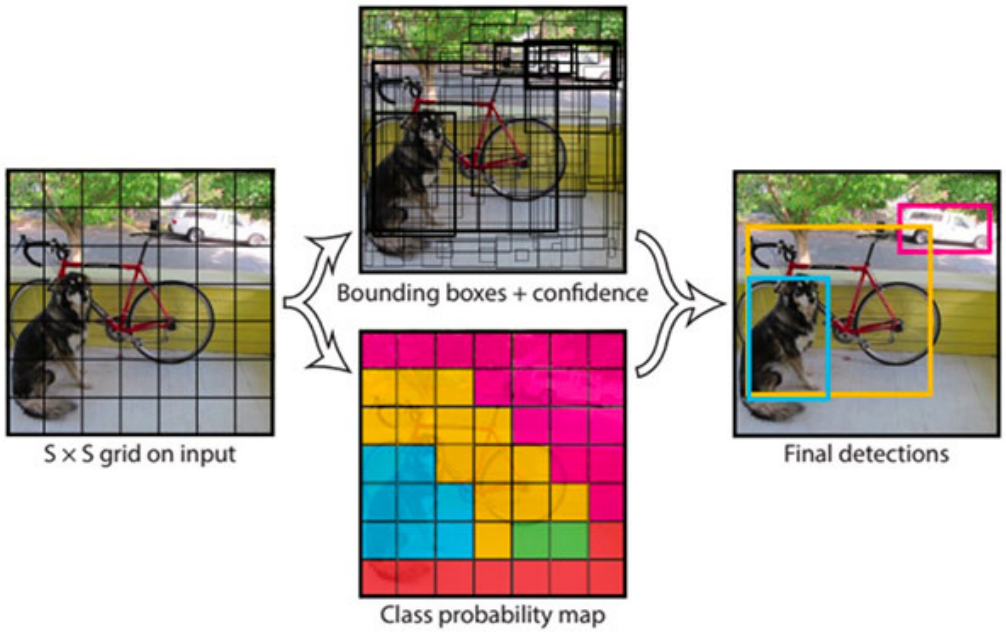


Figure 2.8: YOLO example image [9].

convolutional layers [38]. The role of 2 fully connected layers predicts the boundary box's coordinates and probabilities, and 24 layers are used to extract the features from the input image [9]. Inception modules of YOLO followed by GoogLeNet, but the concept of one by one reduction layers, which is the next step of 3 by 3 convolutional layers, is motivated by Lin et al. [39] seen in figure 2.9. There are some different architectures of YOLO, which are YOLO-tiny, YOLO-320, YOLO-416, and YOLO-608. The fastest version consists of fewer layers for extracting features from images than the standard version, such as six convolutional layers for features instead of 24. Owing to reducing layers, the fastest version of YOLO shows accelerated speed, but a lack of accuracy results due to the relatively fewer steps for extracting features [9].

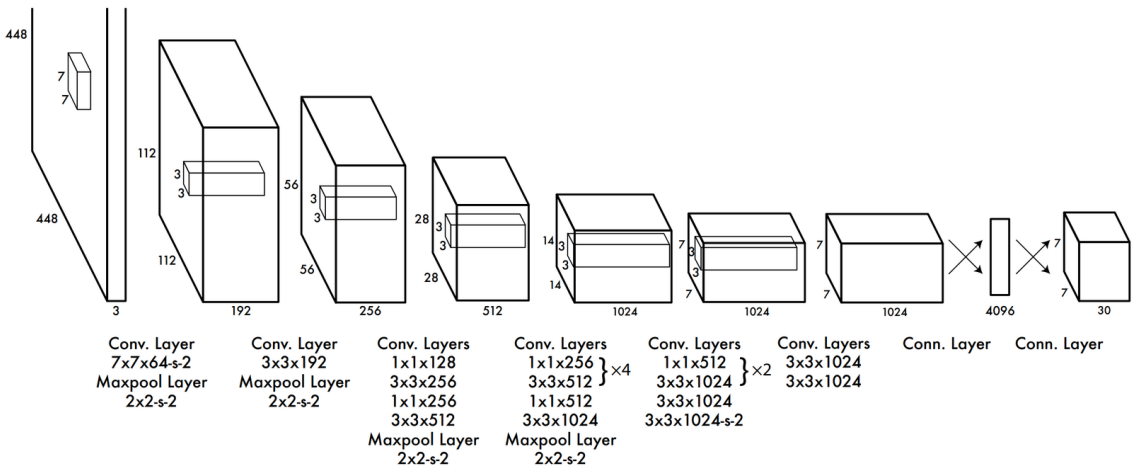


Figure 2.9: YOLO Architecture [9].

2.3.6 Anchor Box

The unique characteristic of the YOLO is that the one cell is able to predict only one class. However, when the centre points of multiple objects are located in one cell, YOLO chooses only one object. In order to deal with this problem, YOLO adopts anchor boxes, which can detect multiple overlapped objects in one grid cell. One example is someone standing beside a car (figure 2.10). The centre points of two different objects are placed in the third column and second row of the grid cell in figure 2.7. Thus, two different dimension anchor boxes detect a car and a person.

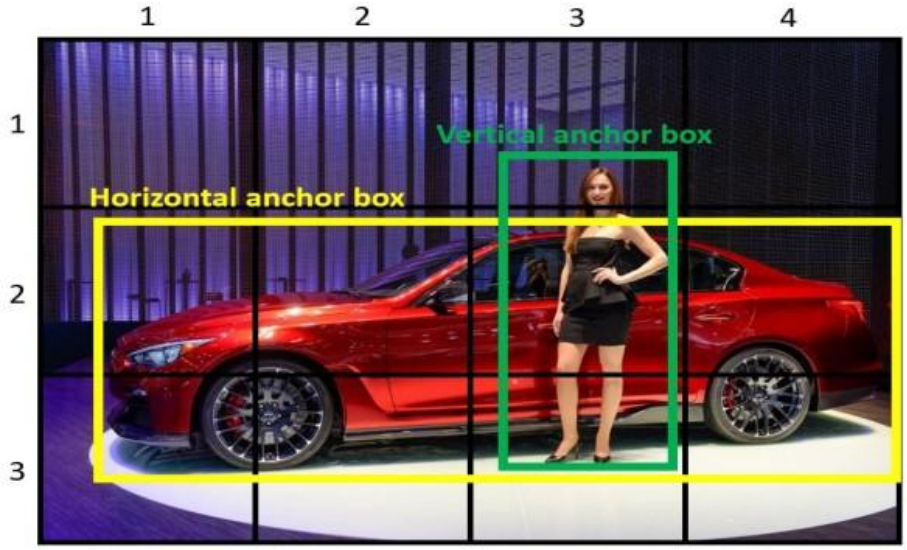


Figure 2.10: Anchor box example [8].

2.3.7 Non-Max Suppression

Sometimes the algorithm detects various bounding boxes of different sizes and locations for a single object. YOLO invented a technique named Non-Max Suppression to choose a bounding box between multiple ones using probability and intersection over union (IOU). IOU is the overlapped area between the predicted bounding box and the ground truth (figure 2.11a). The method is to select the bounding box with the highest probabilities between all bounding boxes, which are the value of IOU is over 0.5 (or it can be changed to any threshold value) about the target object. For instance, in figure 2.11b it shows the probabilities of the bounding boxes as 0.9 (Green), 0.7 (Red), and 0.6(Yellow). Choosing the highest bounding box, the green box in figure 2.11b, and then eliminating all boxes with IOU is bigger than 0.5, which are red and yellow boxes. The final result is shown in figure 2.11c.

(a) Intersection over union example. (b) Before Non-Max suppression. (c) Non-max suppression result.

Figure 2.11: IOU and Non-max.

2.4 Object Detection using Fourier Descriptors

Fourier Descriptors (FD), which is inspired by the early research of Cosgriff [40], enables researchers to apply the theory of Fourier Descriptors to shape description in computer vision [41]. The idea of Fourier Descriptors is to transform the outline of an object into a set of numbers that represent the whole shape and is obtained from the Fourier coefficients [41]. The main benefit of Fourier Descriptors is that it is able to compute invariant features from features are not invariant [7]. The meaning of invariant features is that they are not changed when transforming the image by rotation and scaling [42]. So that Fourier Descriptors can efficiently deal with the scaling issue of the object's profile and accurately reconstruct by using its Fourier coefficients [41]. Fourier Descriptors can be extracted from the coordinates of the boundary points (x,y), such as $(x_0, y_0), .., (x_{k-1}, y_{k-1})$ using the following equations:

$$s(k) = x(k) + iy(k) \tag{2.3}$$

The modified form is:

$$a(u) = \sum_{k=0}^{K-1} s(k) e^{-i2\pi uk/K} \tag{2.4}$$

In addition, the inverse transform can retrieve the initial positions of the feature [7]:

$$s(k) = \frac{1}{K} \sum_{u=0}^{K-1} a(u) e^{i2\pi uk/K} \tag{2.5}$$

Partially reconstructing the profile is able to use a finite number of coefficients, but

there are some possibilities of losing some appearance of the feature. The first few coefficients would be able to reconstruct the profile with some accuracy.

2.4.1 Implementing Fourier Descriptors

The first step for implementing Fourier Descriptors is to divide the real from imaginary elements of the combined number in equations 2.3, 2.4, and 2.5

$$a_{real}(u) = \sum_{k=0}^{K-1} x(k)cos(\theta) - y(k)sin(-\theta) \tag{2.6}$$

$$a_{imag}(u) = \sum_{k=0}^{K-1} x(k)sin(-\theta) + y(k)cos(\theta) \tag{2.7}$$

where: $\theta = (2\pi uk)/K$ and K is the total number of x and y coordination in the contour. To reconstruct the profile, the following equations can be used:

$$x(k) = \sum_{u=0}^{K-1} a_{real}(u)cos(\theta) - a_{imag}(u)sin(\theta) \tag{2.8}$$

$$y(k) = \sum_{u=0}^{K-1} a_{imag}(u)cos(\theta) + a_{real}(u)sin(\theta) \tag{2.9}$$

However, the equations 2.6, 2.7, 2.8, and 2.9 are not helpful in the cases when the image is rotated, translated, and scaled. Fortunately, the following equations are invariant features from Fourier Descriptors [41]:

$$a_x(k) = \frac{1}{K} \sum_{u=0}^{K-1} x(k)cos(\theta); \tag{2.10}$$

$$b_x(k) = \frac{1}{K} \sum_{u=0}^{K-1} x(k)sin(\theta); \tag{2.11}$$

$$a_y(k) = \frac{1}{K} \sum_{u=0}^{K-1} y(k)cos(\theta); \tag{2.12}$$

$$b_y(k) = \frac{1}{K} \sum_{u=0}^{K-1} y(k)sin(\theta); \tag{2.13}$$

The features set arranged like the equation 2.14 (k starts from 1).

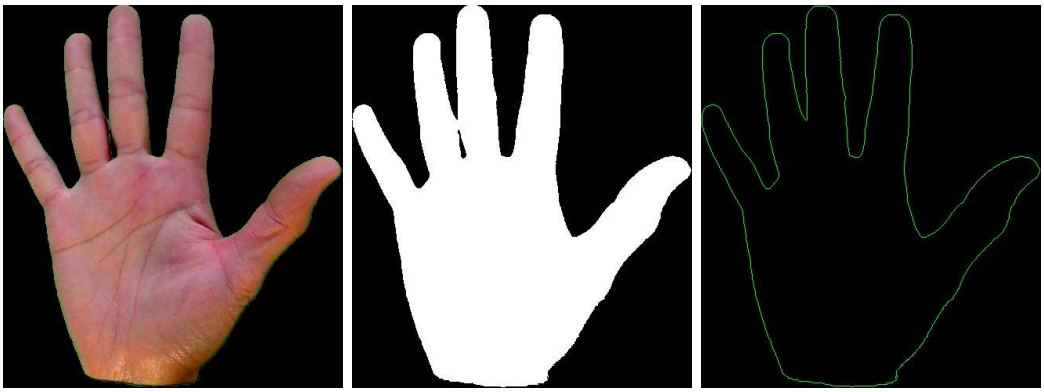
$$CE(k) = \sqrt{\frac{a_x^2(k) + a_y^2(k)}{a_x^2(1) + a_y^2(1)}} + \sqrt{\frac{b_x^2(k) + b_y^2(k)}{b_x^2(1) + b_y^2(1)}} \tag{2.14}$$

The above CE form is called Elliptic Fourier Descriptors [41], and the function is

primarily invariant to affine transformations such as translation, rotation, and scaling. The listing 2.1 shows the implementation of the equation 2.14. The figure 2.12 is an example result of Elliptic Fourier Descriptors.

Listing 2.1: Elliptic Fourier Descriptors [7].

```
1  void EllipticFourierDescriptors(vector<Point>& contour,
       vector<float> CE){
2   vector<float> ax, ay, bx,by;
3   int m=contour.size();
4   int n=20;//number of CEs we are interested in computing
5   float t=(2*PI)/m;
6   for(int k=0;k<n;k++){
7     ax.push_back(0.0);
8     ay.push_back(0.0);
9     bx.push_back(0.0);
10    by.push_back(0.0);
11    for (int i=0;i<m;i++){
12      ax[k]=ax[k]+contour[i].x*cos((k+1)*t*(i));
13      bx[k]=bx[k]+contour[i].x*sin((k+1)*t*(i));
14      ay[k]=ay[k]+contour[i].y*cos((k+1)*t*(i));
15      by[k]=by[k]+contour[i].y*sin((k+1)*t*(i));
16    }
17    ax[k]=(ax[k])/m;
18    bx[k]=(bx[k])/m;
19    ay[k]=(ay[k])/m;
20    by[k]=(by[k])/m;
21  }
22  for(int k=0;k<n;k++){
23    CE.push_back( sqrt((ax[k]*ax[k]+ay[k]*ay[k])/(ax[0]*ax[0]+
        ay[0]*ay[0]))+sqrt((bx[k]*bx[k]+by[k]*by[k])/(bx[0]*bx
        [0]+by[0]*by[0])) );
24  }
25  for(int count=0;count<n && count<CE.size();count++){
26    printf("%d CE %f ax %f ay %f bx %f by%f \n",count
        ,CE[count],ax[count],ay[count],bx[count],by[count]);
27  }
28 }
```



(a) The input hand image. (b) Segmentation of (a) (c) Contour of (b)

Figure 2.12: The first five FD coefficients for the contour in (c) are: 0.741, 0.315, 0.411, 0.174, 0.324 [7]

2.5 Image Segmentation: Image Subtraction

Image subtraction is a whole operation by itself, as single pixels are subtracted from a different image [43]. The primary purpose of image subtraction is to recognize the difference between two images. This technique is commonly used for extracting an object by removing the background from consecutive frames created by static cameras. When the subtraction is performed, there are two pre-processes before applying subtraction. The first is that an input image such as RGB is converted to a grey-scale image. Using subtraction of two sequential colour images could not recognise the general changes between two images because the value of RGB is sensitively changed. On the other hand, the subtraction of two sequential images in grey-scale highlights the changes between images. The last is that size and number of channels of sequential frames must be the same. The two conditions above are pre-requisites in subtracting two images. If the two conditions are not satisfied, a segmentation fault may occur.

The process of image subtraction is shown below in figure 2.13. Subtracting two sequential frames does not extract the target object perfectly from the scene due to a sudden environmental changes. Therefore, an additional filtering process is required for perfect image segmentation using sequential images.

2.6 Filters: Additional Tools for Image Segmentation

The technique of image subtraction is not sufficient to deal with image segmentation perfectly. Some supporting filters are required to succeed in the processing of image segmentation. We will discuss the Median and Kuwahara filters with a Summed-area table in this section.

Figure 2.13: Image subtraction process [10].

2.6.1 Median Filter

Tukey [44] suggests one-dimensional waveforms, which is the root concept of median filter, and then this technique expands to two-dimensional form in order to apply for image noise smoothing. The median filter eliminates the mess of an image (see an example in figure 2.14) and is commonly used as pre-processing before applying the detection technique. For instance, according to Narendra and Patrenahalli [45], a median filter is implemented before processing an edge operator because this operator struggles to detect the line where remaining salt-and-pepper noise exist, and the median filter would transform this noise into a smooth area.

The intensity of the median filter can be adjusted by changing the size of the sliding window, which is usually a square M×M [45]. A smaller size erases less noise, and a larger size erases more noise. However, if the window size is relatively large, the performance is significantly decreased. There is a trade off between the size of the sliding window, the ability to suppress noise and the speed.

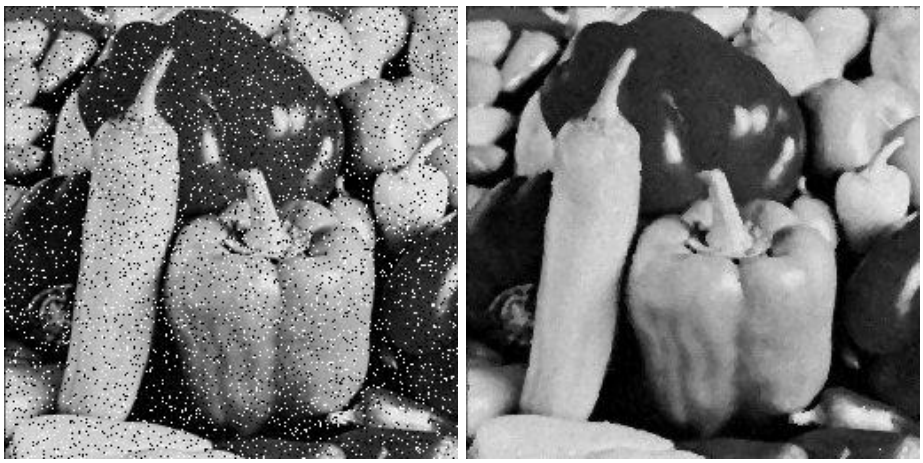


Figure 2.14: Median filter example [7].

2.6.1.1 Median Filter Algorithm

The window size of a median filter must be only odd to avoid the ambiguity of selecting the central pixel value. After determining the window size, the values inside the window are sorted in ascending order, then the final value is the middle one from the sorted values (figure 2.15). This process is applied in all pixels except for the edge of an image (grey values in "Output" of figure 2.15). Therefore, the size of the edge, where the median filter is not applied, depends on the size of the window in equation 2.15.

Figure 2.15 shows a simple example of a median filter applied to a generic image. The window size in figure 2.15 is set to 3×3. In that case, the values inside the window, which are the values of the black and grey background colour of "Input" in figure 2.15, are read one by one, then sorted from the lowest number. The 5th value, which is the central value of "Sorted" in figure 2.15 is the final value for this window, which is the second row and column value from "Output" in figure 2.15. The original corners from "Output" in figure 2.15 are preserved.

$$The\_Size\_of\_Edge = \frac{window\_size + 1}{2} - 1 \tag{2.15}$$

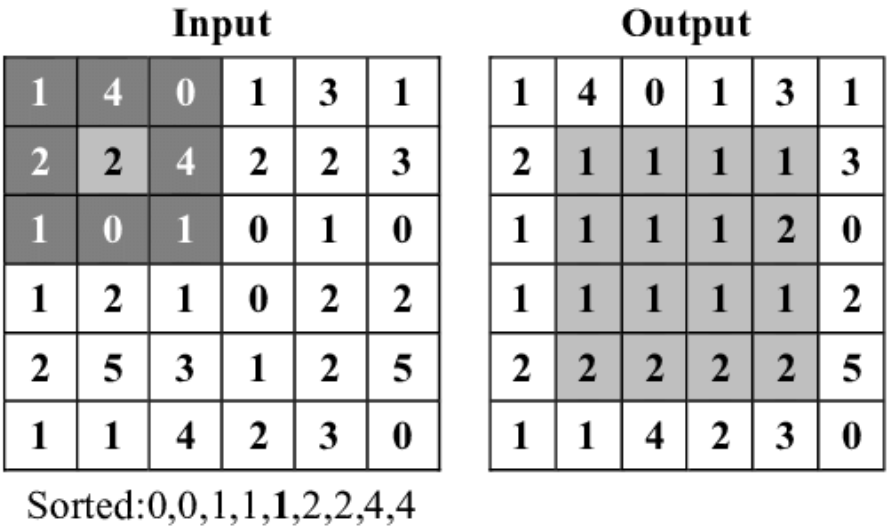


Figure 2.15: Median Filter Window[11].

2.6.2 Kuwahara filter

The Kuwahara filter is a non-linear function and was invented by Kuwahara et al. [46] to improve the medical model, especially in dynamic heart muscle [46]. This filter can reduce noise by improving the strength of the edge and creating stylized abstractions from images by transforming the input image into a smooth image [47], as it is shown in

figure 2.16. In order to determine the value of the pixel, the kernel, which is divided into four different areas (figure 2.16). The centre pixel of the kernel is determined by mean of the region with the smallest variance, which is computed between four regions. The final value, the green value of the figure 2.16, is determined by the smallest variance of one the areas between 1 and 4 (figure 2.16) using equation 2.18. Therefore, choosing the area with the smallest variance makes the areas of objects more homogeneous and makes the object contour more clear.

In addition, the bigger size of windows creates a more abstract image (figure 2.17). However, applying this filter with the big size of the window needs a tremendous amount of calculation and access to the pixels. For instance, a 5 by 5 window size in figure 2.16 needs to access 36 pixels to determine only one pixel, whereas 11 by 11 window requires to access 100 pixels in order to deal with only one pixel. Therefore, the Kuwahara filter requires more computations than a Median filter. The most time-consuming work in the Kuwahara filter is to calculate the sum of pixels in areas of the kernel. The runtime in the Kuwahara filter can be decreased by applying Summed-area tables [48], especially when increasing the size of the kernel of the filter. The section 2.6.3 will discuss Summed-area tables.

$$N = \left(\frac{Kernel\_Windows\_Size + 1}{2} \right)^2 \tag{2.16}$$

$$\mu_a = \frac{1}{N} \sum_{(x,y) \in a} (i(x,y)) \tag{2.17}$$

$$\sigma_a^2 = \frac{1}{N} \sum_{(x,y) \in a} (i(x,y) - \mu_a)^2 \tag{2.18}$$

where:

a is $\{0, 1, 2, 3\}$,

i is the input image, and i(x,y) is the pixel value at the coordinates (x,y)

μ_a is the average value of the pixels in area a

σ_a is the variance of the pixel values in area a

2.6.3 Integral Images (or Summed-area Tables)

Summed-area tables (SATs) were suggested by Crow [48] in computer graphics and is one of the main contributions for generating real-time face detection by Viola and Jones [6]. Each cell in the SAT organizes the sum of all elements from above to the left of the location of the cell in the input image, and it is also called a cumulative table [12] or integral images [6]. The main strength of the summed-area table is to calculate the summed value from any scale of the area with a runtime that is independent of the size

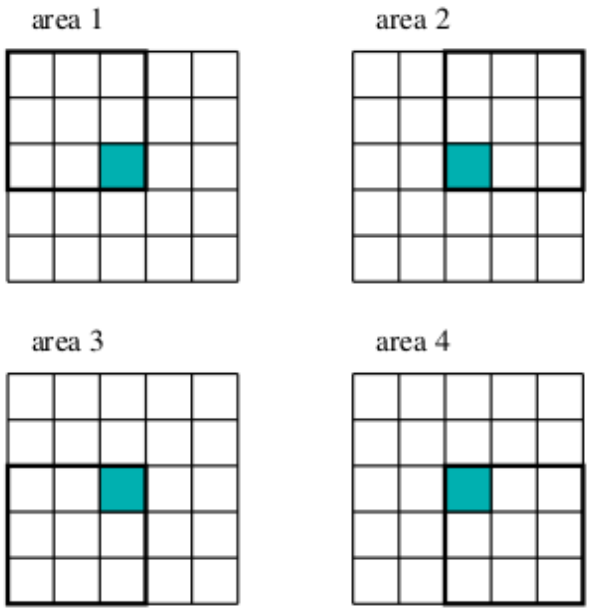


Figure 2.16: Kuwahara filter with 5x5 kernel [7].

of the area.

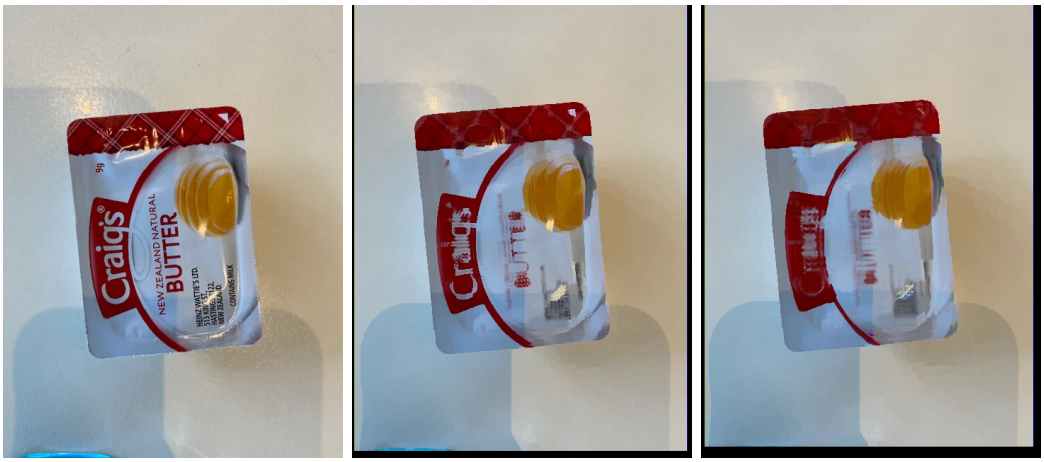
2.6.3.1 The Method of Summed-area Tables

There are three primary steps for creating and applying a summed-area table. The first step is converting the input image to a grey-scale one, which has the level intensity of each pixel from 0 to 255 [49]. The second step is creating a new two-dimensional array, which contains the summed information using the equation 2.19 [7]. For example, the summed value of the blue cells in the left of the figure 2.18 is corresponding to the value of the fourth row and column, which is the blue cell in the right of figure 2.18. The last step is to calculate the summed value using all edge values of the rectangle area. For instance, the sum of green values in the left figure (2.19), which is 27, is the target of the calculation. Using a summed-area table, the summed value of "B" and "C" in the right figure 2.19 is 31 subtracted from the added value of "A" and "D" in the right figure 2.19 is 58, so the result is 27. Because the access is point by point, this method does not affect the performance in larger kernels when computing a sum of an area. Therefore, implementing the Kuwahara filter using Summed-area Tables has a great benefit for performance.

$$S(x,y) = \sum_{i=1}^{x}\sum_{j=1}^{y} T(i,j) \tag{2.19}$$

$$Sum(R) = A(R) - B(R) - C(R) + D(R) \tag{2.20}$$

$$A(R) = S(R_x max, R_y max) \tag{2.21}$$



(a) Input. (b) 11×11 Kernel. (c) 17×17 Kernel.

Figure 2.17: Example of Kuwahara filter.

input table

1	3	0	2	1	2
3	2	4	3	6	0
0	5	1	1	5	3
2	2	3	3	7	2
4	2	8	4	2	5

summed-area table

1	4	4	6	7	9
4	9	13	18	25	27
4	14	19	25	37	42
6	18	26	35	54	61
10	24	40	53	74	86

(a) Input Table. (b) Summed Area Table.

Figure 2.18: The First Example of The Summed-Table [12].

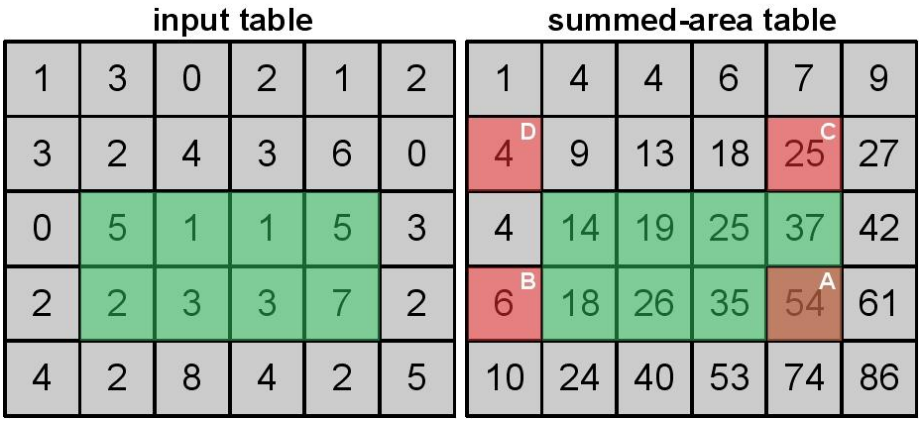
$$B(R) = S(R_xmin - 1, R_ymax) \tag{2.22}$$

$$C(R) = S(R_xmax, R_ymin - 1) \tag{2.23}$$

$$D(R) = S(R_xmin - 1, R_ymin - 1) \tag{2.24}$$

2.6.3.2 Limitation of the implementation of a Summed-area Table

The implementation of summed-area tables should be aware of the size of the variable used in the matrix. The edge value of the right-bottom of the integral image is the summed value of all pixels from an image, and the maximum of this value is the equation 2.25 [7]. Therefore, checking the maximum value of the summed-area table is essential to avoid overflows in the matrix. The high resolution of an image may cause overflow in the matrix, and in such cases it is recommended that the size of an image is reduced.



(a) Computing the green area: 5+1+1+5+2+3+3+7=27

(b) This is equivalent to: 54-25-6+4=27

Figure 2.19: Summed-area table example [12].

$$Maximum = Width * Height * 255 \tag{2.25}$$

2.7 Object Labelling: Blob Finder

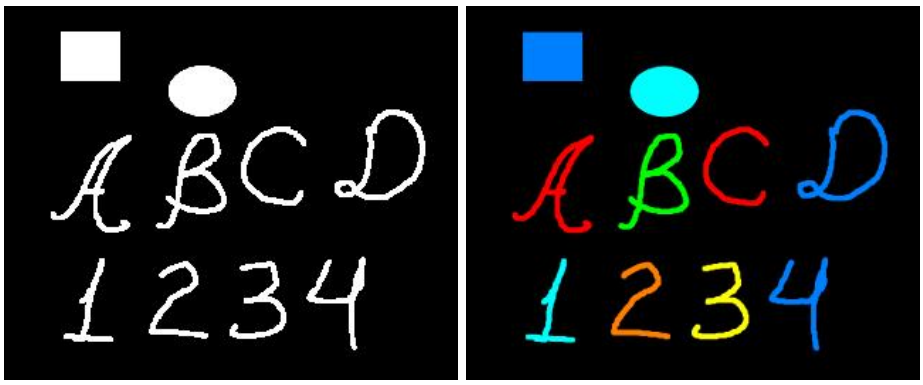
An automated system for segmenting an image remains a challenging work in the computer vision area, and this work is closely related to the result of some applications, including image annotation, object recognition, image indexing, and image coding [50]. Image segmentation is sometimes achieved using a blob detection. In order to begin the segmentation of the region in an image, the input image change to a binary image (zero for a black pixel or non-zero for a white one). The zero pixels are set as the background, and the non-zero pixels are the object. Grouping the non-zero pixels is the goal of the blob finder.

The approach of counting and finding the non-zero pixels' location in a computer is different than human eyes. The programming approach is to access and scan every pixel using a loop, and after finishing the loop, it returns the results of the analysis. So, it is important that the data structure of this algorithm is considered because the grouping process can be complex. For example, merging and splitting sets may need to use and reuse indexes.

The method of joining pixels into a group is to check the adjacency of the pixel. If any neighbour non-zero pixels exist from the target pixel, this target joins the neighbour group. If the pixel is initially not connected to any other pixel, the data structure creates a new group. There are two types of adjacency; four and eight. Diagonal pixels of the target can be the neighbour in 8-adjacency, but the simplified version as 4-adjacency has better performance than 8-adjacency. The figures 2.20a and 2.20b are examples of the

blob finder at work.

In the view of programming in 4-adjacency, the algorithm determines the direction of the scan from left to right and top to down. During the scanning and grouping process, the program only needs to check two neighbours, which are the left and top pixels, because other directions will be checked by the right and bottom pixels. The pseudo-code and process is illustrated in the listing 1 and the figure 2.21 respectively.



(a) Input Image (b) Result of Blob

Figure 2.20: Blob Processing [7].

Algorithm 1 ***Object Labelling using 4-adjacency*** [7].

Require: a **binary** image $i(x, y)$
declare a vector of sets $SET[\,]$
declare integers $counter = -1$, $s1$, $s2$
declare $A[i.width][i.height]$ {a 2D vector or mat initialised to -1}
for $y = 1$ to $i.height$ **do**
  for $x = 1$ to $i.width$ **do**
    if ($i(x, y) \neq 0$) **then**
      if ($i(x-1, y) \neq 0$ OR $i(x, y-1) \neq 0$) **then**
        $s1 = A[x-1][y]$
        $s2 = A[x][y-1]$
        if ($s1 \neq -1$) **then**
          $i(x, y) \rightarrow SET[s1]$ {insert point $i(x, y)$ into $SET[s1]$}
          $A[x][y] = s1$
        end if
        if ($s2 \neq -1$) **then**
          $i(x, y) \rightarrow SET[s2]$
          $A[x][y] = s2$
        end if
        if (($s1 \neq s2$) AND ($s1 \neq -1$) AND ($s2 \neq -1$)) **then**
          $SET[s1] \cup SET[s2]$ {Union, keep set $SET[s1]$ and empty the other}
          Reset all points of $A(x, y)$ belonging to $SET[s2]$ to $s1$
          empty $SET[s2]$
        end if
      else
        $counter = counter + 1$
        Create new set $SET[counter]$
        $i(x, y) \rightarrow SET[counter]$
        $A[x][y] = counter$
      end if
    end if
  end for
end for

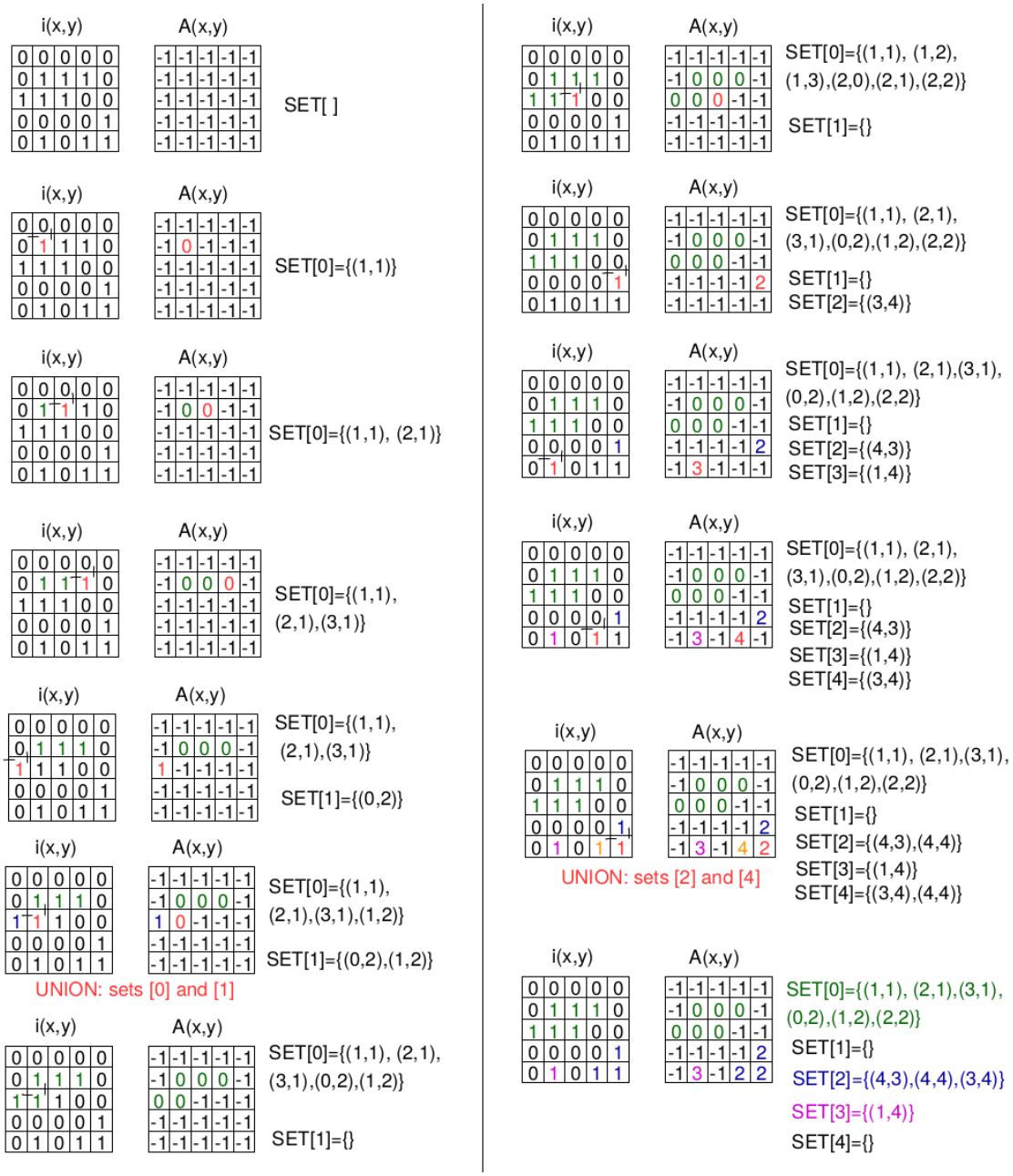


Figure 2.21: Blob finder example [7].

2.8 Machine Learning

Machine learning is the area about computer algorithms that develops knowledge automatically by experience using data [51]. According to Arthur Samuel [52], machine learning algorithms create a model by training data to predict the output through the analysis of the input value without being explicitly programmed. Sometimes, humans struggle to predict the statistical answer with organized data, but machine learning algorithms are designed to solve these problems quickly. There are several efforts to apply the vast data sets in machine learning by many programmers and mathematicians [13]. Machine learning has various algorithms to predict answers given features [13] (see example in figure 2.22). Between the figure 2.22, we briefly explore Random Forest, AdaBoost, Support Vector Machine (SVM), and Neural Network.

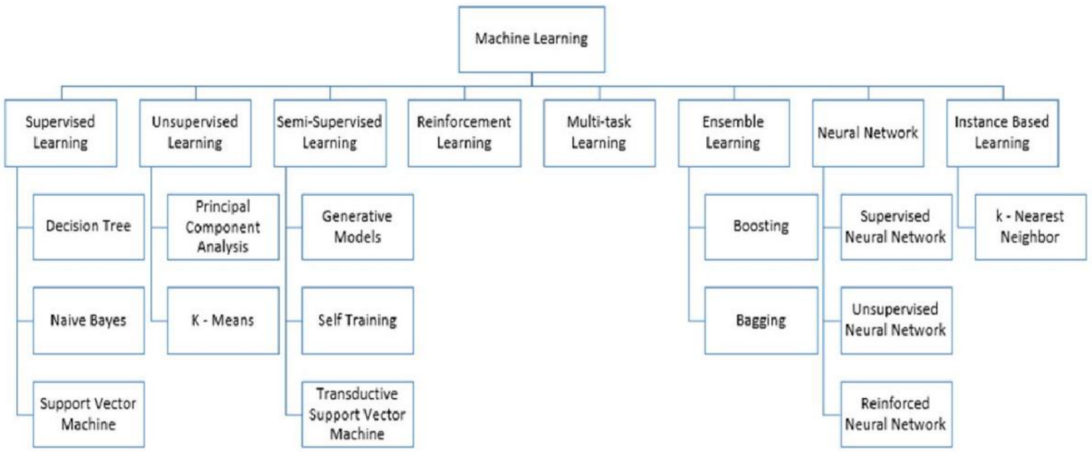


Figure 2.22: The various types of machine learning [13]

2.8.1 Algorithms

2.8.1.1 Random Forest

The random forest classifier (RF) consists of independently trained tree classifiers. Those trees are generated by a process called *bootstrap*, which re-samples the training set randomly, with replacement. The majority vote selects the final class [53]. The tree classifier is a decision tree which is the form of a binary tree with various nodes. The nodes are organised by choice or event, and the edges of the binary tree represent states or decisions [13]. The user declares the specific number of randomly selected variables to split at each node [54]. Splitting each node can use either the Gini index or entropy [55]. Briefly, random forest determines the answer based on the choice of the most voted by each decision trees. The figure 2.23 shows the process of random forest algorithm.

Figure 2.23: Example of Random Forest [14]

2.8.1.2 AdaBoost

Boosting refers to the ensemble learning algorithms that combines multiple weak classifiers into a single robust classifier. AdaBoost is one of the first boosting algorithms and it was introduced by Freund and Schapire [56]. AdaBoost uses a forest of stumps rather than a full-sized decision tree in figure 2.23. A single stump is able to use a single feature per decision, with a confidence slightly better than 50%, so stumps are called "weak learners". Each stump has a different weight. Larger stumps have more influence in the final classification than a small one. In addition, the stumps are connected to each other, so the error from the previous stump can influence the answers of the next stump. To train a stump, the distribution of the training samples uses weights. After a stump is trained, the weights of the training sample are updated, changing the distribution at each round.

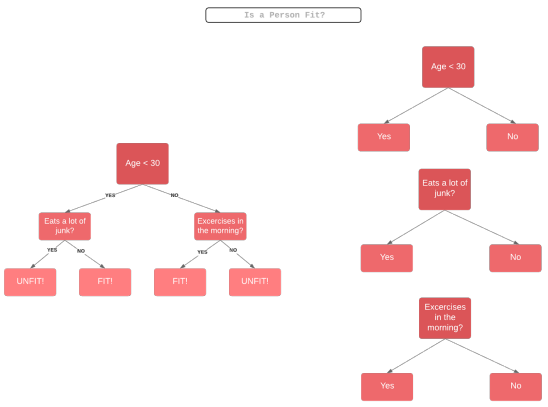


Figure 2.24: Examples of a decision tree and stumps [15].

2.8.1.3 Support Vector Machine

Support Vector Machines (SVM) is a supervised learning model related to learning algorithms for regression analysis and pattern classification [57]. The method of SVM is to create a hyperplane, which is the lane for dividing each class. In addition, SVM can perform both a linear and non-linear classification. The kernel in SVM is able to generate mapping inputs into high-dimensional feature spaces sequentially by drawing hyperplanes in each class [13]. This hyperplane is drawn by the centre point of the nearest two different classes in order to minimize the classification error. An example can be seen in figure 2.25.

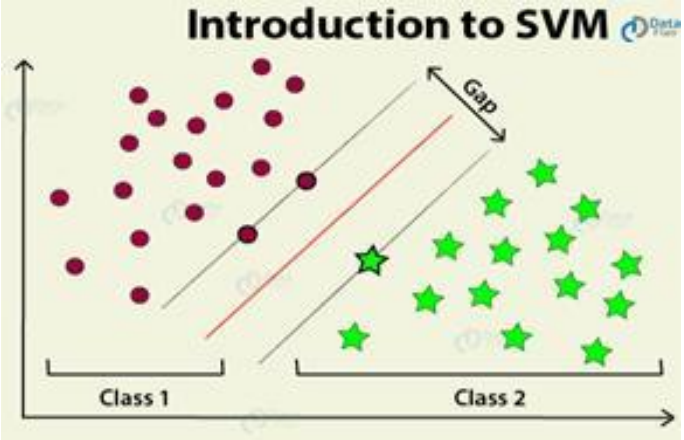


Figure 2.25: The example SVM [13].

2.8.1.4 Neural Network

A neural network is a kind of algorithm that attempts to identify the relationships in a group of data through a method that imitates a human brain [13] and is specialised to make decisions in a human-like ways and recognize patterns. A neural network consists of nodes or artificial neurons [58] and follows the system of neurons. An artificial neuron is a compound of input units, hidden units, and output units (figure 2.26). Input units perform to receive the several forms of the input units, and the hidden units are the primary brain process to analyse the pattern and recognize it, then finally output units return the answer. The connections between units are called weights, and these weights are fully connected between all nodes in the figure 2.26. The primary benefit of a neural network is producing the ideal output by adapting to change inputs without reconstructing the output criteria, and this concept is the root in artificial intelligence [13].

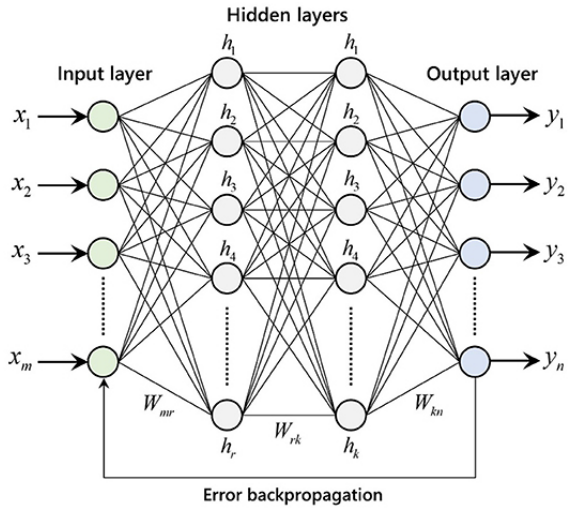


Figure 2.26: The example neural network [16].

2.8.2 Metrics for Machine Learning

2.8.2.1 Confusion Matrix and Accuracy

Confusion Matrix, commonly known as an error matrix, is the visualised data storage that can simply calculate the accuracy of a land-cover map [59]. This confusion matrix is also in the form of a square matrix, which is arranged in the same number of columns and rows, organized with two dimensions as actual and predicted. The main purpose of employing this matrix in the field of machine learning is to estimate the accuracy from prediction. Accuracy is the number of correct prediction $(TP + TN)$ divided by the total number of predictions $(TP + TN + FP + FN)$, as per equation 2.26. In addition, the method of calculating the accuracy using confusion matrix table is that whole-cell values, which are the values of grey and red cells in figure 2.27 divided by the values of red cells in figure 2.27. In machine learning, this matrix is a tool to arrange the next training plan and tune parameters for learning as evaluating the significantly incorrect predicted classes with the value of true negative and false negative. Therefore, the confusion matrix is essential for analysing the results of a model.

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.26}$$

- True Positive (TP)
  - The actual value is 0, and return the predicted value as 0 correctly.
- True Negative (TN)
  - The actual value is 1, but return the predicted value as 1 correctly.

- False Positive (FP)
  - The actual value is 1, but return the predicted value as 0 incorrectly.
- False Negative (FN)
  - The actual value is 0, but return the predicted value as 1 incorrectly.

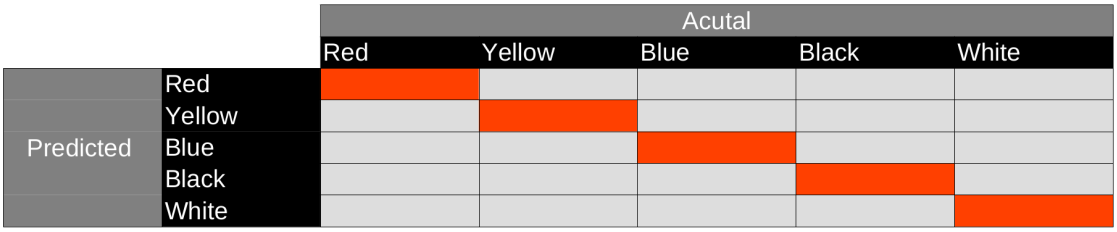


Figure 2.27: Example of multi-class confusion matrix.

2.8.2.2 F1-Score, Precision, and Recall

The accuracy above is the general evaluation of the detection of an algorithm. However, it does not constitute the individual assessment in the accuracy. Therefore, F1-score is able to measure the quality of one algorithm and the balance between recall and precision (equation 2.29). Precision is the amount of the correctly detected target class between all predicted 'True' labels [60] (equation 2.27). For instance, there are two classes in a rodent and a bird, and the target class is a rodent. The precision of a rodent refers to the confidence rate of detecting a rodent. In addition, recall represents the percentage of the correctly predicted target class between the total amount of actual 'True' labels [60] (equation 2.28). If the recall value shows a high score, it means the algorithm has a robust ability to detect the target class [60]. F1-score is a compounded value of precision and recall, and this value is suitable to evaluate the model because often precision and recall are contradictory [60].

$$Precision = \frac{TP}{TP + FP} \tag{2.27}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.28}$$

$$F1 = 2 \cdot \frac{Precision * Recall}{Precision + Recal} \tag{2.29}$$

2.8.2.3 Cross Validation

From the section 2.8.2.1 and 2.8.2.2, we looked at the method for calculating the accuracy and f1-score using a confusion matrix. Cross-validation helps to have more objectivity in the measurement of the accuracy, which is calculated using a confusion matrix. Cross-validation allows for the computation of an average measurement of fitness in prediction in order to obtain a precise statistical result of the model prediction performance [61]. The first process before applying cross-validation with the provided data is to determine the value of the block for splitting all data into test and training data. For example, suppose the block value is 5. In that case, 20% of the data set is for test, and the rest of the data is for training (figure 2.28). These split training set and test set are not correlated, so using this method gives a better estimate of the accuracy of the classifier. Moreover, the accuracy is measured by computing the average value of accuracies from the iterations in figure 2.28.

K=5	All Data				
Index	0	1	2	3	4
Iteration 0	Test	Train			
Iteration 1	Train	Test	Train		
Iteration 2	Train		Test	Train	
Iteration 3	Train			Test	Train
Iteration 4	Train				Test

Figure 2.28: Cross validation.

2.9 Conclusion of the Literature Review

In chapter 2, we discussed the related works, the well-known real-time detection algorithm (YOLO), and literature relevant to our new detection algorithm. We notice that there are currently some efforts to apply object detection technologies in low specification machines such as Raspberry Pi, but it is hard to balance speed and accuracy in real-time detection with a low specification machine. On the other hand, if the detection algorithm is transformed to allow for increased speed, for instance by simplifying the architecture, the result usually shows a lack of accuracy. Therefore, the primary objective of this thesis is to design a new detection algorithm that has a well-balanced accuracy and speed characteristic, while using a Raspberry Pi. We focused in finding other researches that aim to increase the speed of the algorithm without a reduction of accuracy.

Chapter 3

A New Detection Algorithm Based on FDs

The idea of detecting objects comes with the extraction of the object by subtracting an image from its background in grey-scale. The sequential images, which show the dynamic movement of the target objects with a relevant static change of background, are suitable for this subtraction. For instance, a fixed-angle camera captured sequential images with a bird flying. The bird's location is shifted compared with frame by frame, whereas there are no dramatic changes with other elements. This flying bird is an example of a case for extracting an object by subtraction. Therefore, at least two sequential images are an essential requirement for our detection algorithm. Any recorded video with a fixed angle camera located in natural habitats can be used.

3.1 Feature Extraction Algorithm

3.1.1 Predicting Target Location: Region of Interest(ROI)

The diagram below in figure 3.1 is the summarised process of predicting the target location based on the image subtraction algorithm. The method of estimating the target location is to extract the object by obliterating the content of the background. After subtracting two images in grey-scale, the result shows the difference between the two images. After the subtracting process, the most significant group from the subtracted image assumes the target location in the next step ("Determine the size of window and location" in figure 3.1). Then, a blob finder is applied to find the position of the most significant chunk of no-zero pixels. Finally, the ROI is decided.

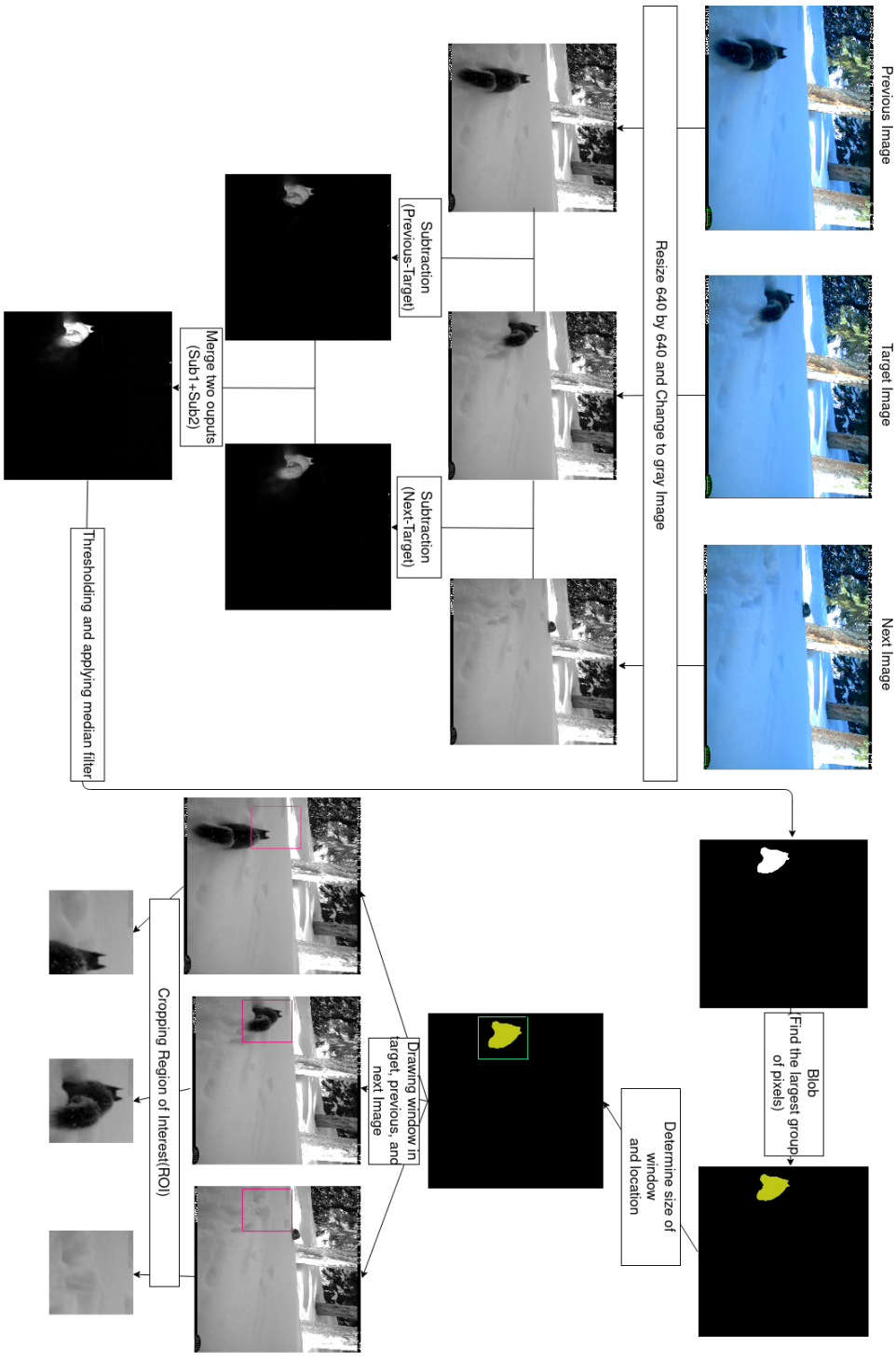


Figure 3.1: The process predicting target location.

3.1.1.1 The Order of the Subtraction

The subtract process begins after converting the RGB scale to grey-scale. On a grey-scale, the relatively bright pixel has a value close to 255, and the moderately dark one has a value close to 0. When investigating the rodent and the bird images, the images have in common that the tone of targets on grey-scale is darker than the target's background (figure 3.2). That is, each pixel value of the target is mostly lower than its background. Suppose that following equation 3.1 is used, some pixels on the target location will be negative. The negative value is regarded as zero because the range of a pixel is from 0 to 255. The result of the equation 3.1 is shown in figure 3.3 (c), that it extracts the target from the previous image. For this reason, the non-target image is placed in front of the minus operator (equations 3.2, 3.3, and 3.4), with the listing 3.1. Therefore, The result of the subtraction is related to its order (figures 3.3 c and d).

$$result0 = target\_frame - previous\_frame \tag{3.1}$$

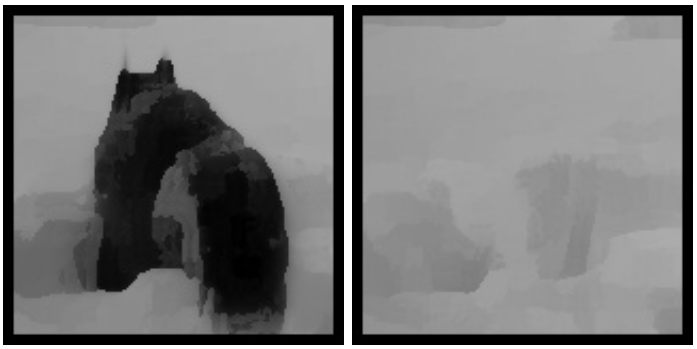
$$result1 = previous\_frame - target\_frame \tag{3.2}$$

$$result2 = next\_frame - target\_frame \tag{3.3}$$

$$result3 = (next\_frame - target\_frame) + (previous\_frame - target\_frame) \tag{3.4}$$

```
1 Mat result, target_frame, previous_frame;
2 for(int i=0; i < target_frame.cols;i++) {
3         for(int j=0;j<target_frame.rows;j++) {
4                 Mpixel(result,i,j) =(int) Mpixel(frame,i,j)-(int
                      )Mpixel(target_frame,i,j)
5         }
6 }
```

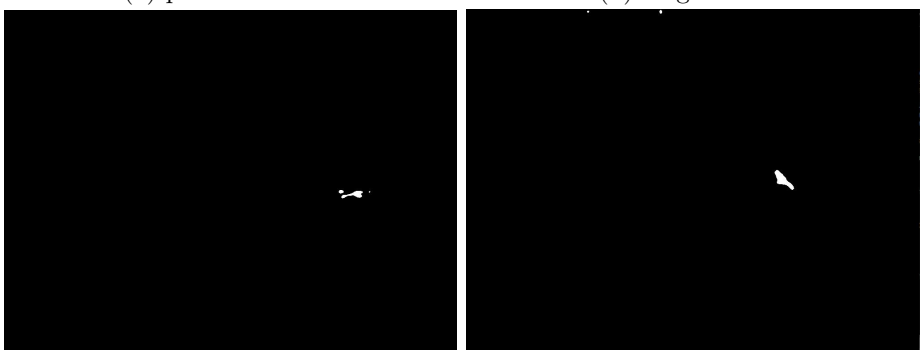
Listing 3.1: Subtraction Operation



(a) Target image. (b) Target background image.

Figure 3.2: Cropped target and background images.

(a) previous_frame (b) target_frame



(c) target_frame - previous_frame (d) previous_frame - target_frame

Figure 3.3: Example of a subtraction

3.1.1.2 Filters: Median filter

The median filter is a time-consuming process, but it is essential to reduce process time of the blob finder. There is a high correlation between processing time and the number of blobs (or groups), so the responsibility of a median filter is to lessen the task of the blob finder by eliminating miscellaneous noise. Although the median filter is relatively slow in performance, without this technique the blob finder spends more time. Moreover, the median filter is a promoting technique to obtain a more clear outline shape of the target. However, sometimes significant features are removed by the median filter. For instance, the outline of the birds' legs is erased because the legs are too thin and regarded as noise in the median filter process. Some bird features without their leg can lead to ambiguity as to whether the object's outline shape is a bird or a rodent, so the median filter's intensity should be dealt with carefully.

3.1.1.3 Filters: Blob Finder

Applying the blob finder technique is to know which of the pixel group is the biggest and where this group is in place. Figure 3.4 illustrates several groups of the pixels with a different colour, and the group of yellow colour is the biggest. This filter returns the vector array, including the group's size and position, and those values are the primary key to finding the location and group. Even after a median filter process, sometimes various noises still remain because of sudden dynamic changes in its scenery, such as wind. In this scenario, more computation is required in the blob finder process, and it cannot extract the object because there is a high possibility of scenery being regarded as an object. In order to solve this issue, we transformed the blob finder algorithm that returns an error code when the groups of pixels exceed 100.

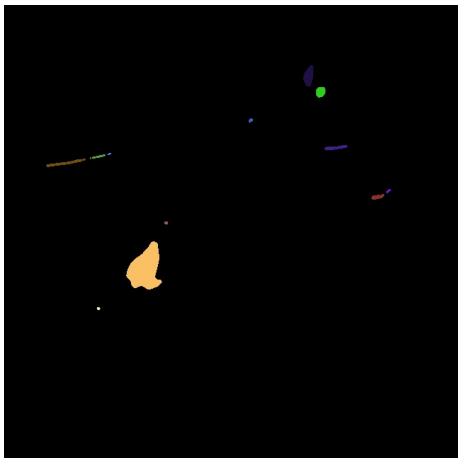


Figure 3.4: Blob Image

3.1.1.4 Determining Window Size and The Center Point of ROI

The blob finder technique enables users to know the location of the largest group of pixels. The method of determining a window's size is based on scanning the pixels at the edges of the window. For in-depth clarification, the figure 3.5 illustrates the determination process of window size. The first is that the programmer sets up the default window size, such as 20 by 20, then the scanning process operates to the edge of this window, whether non-zero pixels exist or not (figure 3.5 (a)). If finding the non-zero pixel at the edge of the temporary window during the scanning process, increase the size of the square (figure 3.5 (b)). Once reach the condition in which all the edges have a value of zero (figure 3.5 (c)), expanding the size one more time, then return the window size and centre point (figure 3.5 (d)). The reason for expanding one more size is in the case that the thresholding process cuts off parts of the target.

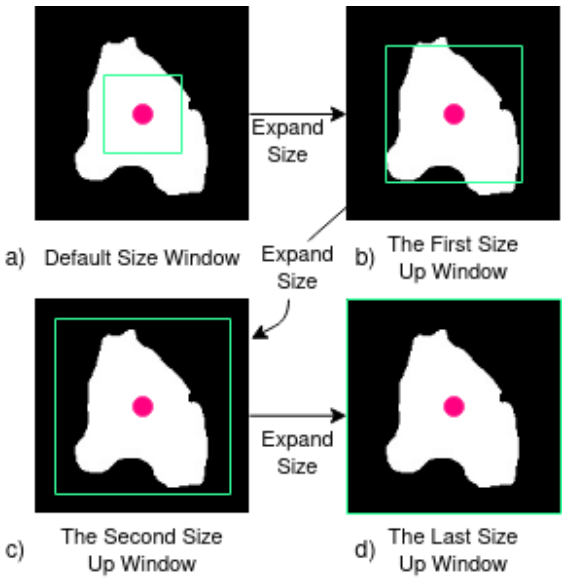


Figure 3.5: The Process Expand Window Size

3.1.1.5 Failing Cases of Finding Region of Interest (ROI)

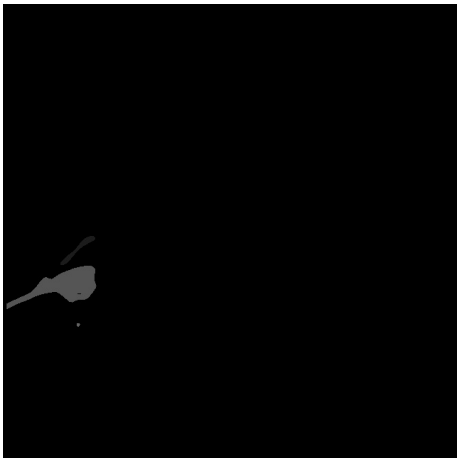
The algorithm for finding the approximate location of the target is to find where is the biggest difference between the target and the previous frame. However, sometimes the movement of the target is too static, for example when an animal is looking around or feeding. Such a small difference of images may cause a failure of the algorithm. There are two examples. One case is where the proportion of the dynamic movement of the target is smaller than the background change. The wrong place can be regarded as the object location. In this case, the algorithm allows to keep processing until the end, and the other processes will filter it.

The other case is that where the approximate location of an object has been detected

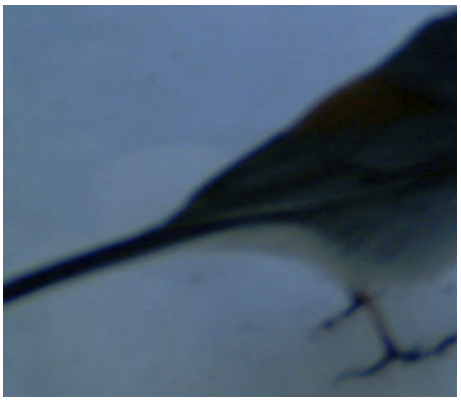
correctly from ROI finding process, but the output of the ROI finding process captured only the parts of the object features such as head and tail (figure 3.6 (c) and (d)). This case occurs when the movement of the target object is not always dynamic. Sometimes the animal stops moving and looks around (figures 3.6 (a) and (b)). To turn this failure into success case, after finishing the algorithm of determining window size, it increases the window size one more time as shown in figure 3.6 (e). The algorithm of determining window size and the centre point of ROI is illustrated in the next section.

(a) Case2: Previous Frame

(b) Case2: Next Frame



(c) Case2: subtracted image



(d) Case 2: without increase size one more time

(e) Case 2: increase size one more time

Figure 3.6: Comparison of Subtraction

3.1.1.6 The Output Files from The finding ROI

The goal of this section is to determine the window's cropping pinpoint and size about the target and non-target images; in short, the process finds the region of interest. The following process, which transforms the contour image into 20 floating-point values, enables to proceed without finding the ROI. However, it does not provide a delicate object outline than an additional processed one with cropped images, even if it ensures a better performance. For this reason, some further processes will be moderately repetitive. The three images below in figure 3.7 show an example of outputs of cropped images, the pinpoint and size are determined by figure 3.7 (b).

(a) Output: Previous Frame (b) Output: Target Frame (c) Output: Next Frame

Figure 3.7: The final output of the first subtraction.

3.1.2 From the Outline into Data using Fourier Descriptors

In the previous section, the process results in a cropped target and non-target images (see the first three images of figure 3.8). The input resources of this section are the output of section 3.1.1.6. The form of the output in this section is the collection of information, including class number, a target image name, and 20 floating point values for the Fourier Descriptors (FD). The 20 values are used by the machine learning algorithms for predicting the detected object class.

3.1.2.1 The Kuwahara Filter

The reason for applying the Kuwahara filter in input images is that it improves the extracting of the outline of an object. However, the biggest drawback is the reduced speed due to requiring numerous calculations, even using the summed-area table (section 3.1.1.2). The speed for applying this filter is strongly correlated with the image's resolution; it is faster with a lower resolution. The cropped image has a relatively low frame size; it does not affect the total process time in this case.

We organised the experiment of processing time in the Kuwahara filter with cropped images. Figure 3.9 illustrates the measured time, in frames per second, for modifying a

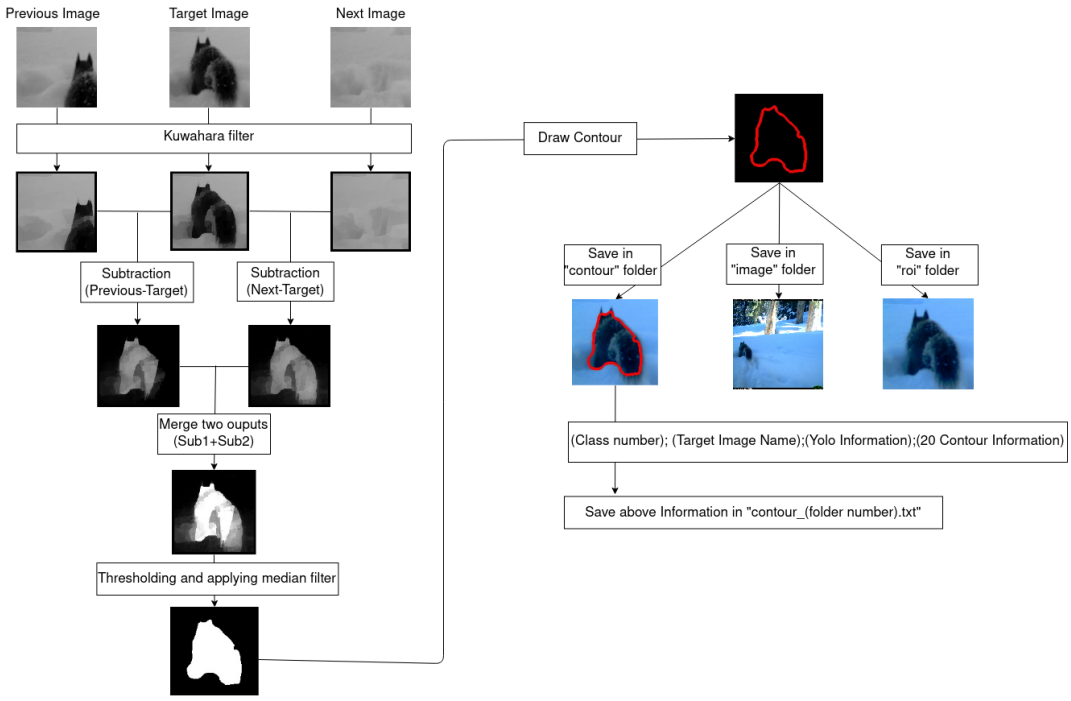


Figure 3.8: The Process of Transforming Data

single cropped image to the Kuwahara filtered one with 200 random images, which the range of size is between 20×20 and 200×200. Most images achieve less than 1200 frames per second, whereas the others show significantly higher speeds. The reason is that those are all small frame sizes (20*20). The green line, f(x), shows the average processing speed of all data as 508.977 frames per second. Therefore, applying the Kuwahara filter in cropped images has less influence on the overall performance. Moreover, this filter can also produce more clear contours.

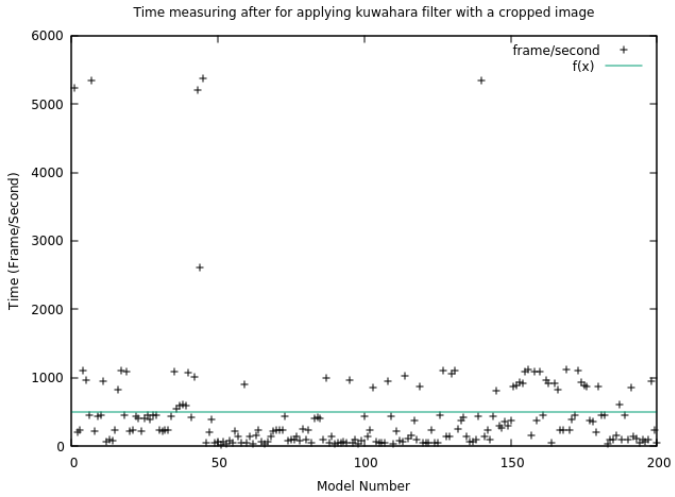


Figure 3.9: Kuwahara performance with different window sizes.

3.1.2.2 The Final Output of The Algorithm

The final output has the additional files shows in the listing 3.2. The values from CE0 to CE19 are used to predict the class of an object, and the others are saved as the resource of the labelling system for training. The detail of the training file structure and further process will be discussed section 4.4.

```
1 <class_number>;<file_name>;<x>,<y>,<width>,<height>;<CE0
    >,..,<CE19>
```

Listing 3.2: The Final Output of Detection

3.2 Labelling for Training and Detection in Real-Time

In the section 3.1, we discussed the main algorithm for feature extraction from sequential images. This algorithm can be adjusted for the parameters, depending on the purpose, either speed or accuracy. For example, in order to boost the speed of the processes, the parameters can be tuned by compressing the window size (section 5.6) and by reducing the strength of the filters, including Median and Kuwahara filters. However, this may result in losing the details of the object's contour, and it may cause the reduction of accuracy. Therefore, tuning the parameters is very important, and this will shown in the experiments in sections (5.3) and (5.6).

Feature extraction is used in a semi-automated labelling system in chapter 4. In this labelling system, the parameters of this algorithm are set as the high-strength values in filters because this labelling system focuses on obtaining the specific details of the object's contour without considering runtime. In contrast, feature extraction for real-time detection should consider the balance between accuracy and runtime. Sections (5.3) and (5.6) will discuss how to find the ideal parameters considering both the accuracy and runtime.

Moreover, we added a feature to the real-time detection algorithm that can be turned on or off depending situation to increase frame rates.

3.3 Real-time Detection

The parameters for the feature extraction in real-time detection are similar to the one used in a semi-automated labelling system. There are two significant differences between the feature extraction used in labelling system and the one used in real-time detection: light architecture for appropriate performance in relation to accuracy and the event trigger.

3.3.1 Light Architecture

In order to create a light architecture, it is essential to analyse what the time-consuming work is. We pointed out the slow factor for performance as the Median filter, Blob finder, and Kuwahara filter from section 3.1. Both the median and Kuwahara filters can be adjusted for fast computation, and the blob finder performance relies on the result of the median filter. Nevertheless, switching to a faster median filter could lead to more processing time in the blob finder because grouping process in the blob finder spends more time with noisier images. Besides, the Kuwahara filter might influence classification. Therefore, tuning the parameters of the median and Kuwahara filters should be handled carefully.

In real-time detection, we adjusted the parameters considering the balance of the median and Kuwahara filters to run faster.

The other way to reduce workload in detection is to scale down the input images. The benefit of this approach is that it does not require modifying any code inside the algorithm, and it simply changes the input image size. The processing time for the three time-consuming filters, which are median, Kuwahara filters and blob finder, also depends on the frame size. However, it decreases the accuracy rate from a particular frame size because the extracted object may be too small to recognize. Therefore, we will experiment with tuning the frame size to understand the relationship between processing time and accuracy in Chapter 5.

3.3.2 Event Trigger

Event trigger is the feature of turning on and off classification. There are two purposes of event trigger: reducing unnecessary calculation for increasing performance and classifying the suitable object. About the first purpose, most of the time in real-time detection the algorithm waits for the target object to be in range of the camera. If the program tries to detect something during this waiting time, segmentation fault may occur because a very large number of blobs can lead to overflowing the data structure holding information about each group. Therefore, real-time detection has the additional feature of pausing classification until a moving target shows up.

For the second purpose, the classifier is always showing the result as one of the trained classes. For instance, if a withered leaf trembles in the wind, the program tries to classify this leaf as whether the rodent or bird. There is no option for neither the rodent nor the bird. The classification during waiting time is the factor decreasing accuracy and delaying other processes. For those reasons, turning on and off the classification process is an essential requirement for our approach for real-time classification using FD.

3.3.2.1 Trigger for Turning On

When activating the classification in real-time, subtracting the current from the previous frame is in process continuously. The algorithm measures the proportion of non-zero pixels using the equation 3.5. If the proportion returns a relatively large value, it has shown a sudden significant change between the previous and next frame. This sudden change can define the appearance of new objects in the captured scene. The value of the equation 3.5 is subjective, and it requires adjusting the appropriate value turning on/off the classification algorithm. The current value for trigger set up 0.01. In other words, the classification algorithm will be activated if the amount of non-zero pixels in the subtracted window is larger than 1%.

$$proportion\_total\_non\_zero\_pixels = total\_non\_zero\_pixels/(frame\_width * frame\_width) \tag{3.5}$$

3.3.2.2 Trigger for Turning Off

When real-time classification is being activated, the target does not always stay from the camera's range, and it can disappear suddenly. Therefore, a feature for turning off the classification algorithm is also required. This feature is the exact opposite turning on method, which stops the classification algorithm when the change between the sequential images is less than 1%. However, sometimes, the trigger for turning off activates wrongly because the animal movement can stop. For instance, if a bird is sitting on the ground for feeding, the motion of a bird should be minimal. In this case, the trigger for turning off is activated. Therefore, the condition of the trigger for turning off the classification adjusts to a smaller value of 0.5%.

3.3.3 Summary

This chapter explored the main algorithm of finding an object and the different usages of this algorithm for a semi-automated labelling system and real-time detection. The summarized steps of the main algorithm are illustrated below.

The steps of main algorithm for finding an object:

1. Region of Interest(ROI).
   (a) Subtracting the sequential images.
   (b) Thresholding the subtracted image.
   (c) Median filter.

(d) Blob finder.

(e) Determining the location and size of ROI.

(f) Cropping ROI in the sequential images.

2. Transform object outline into data.

(a) Kuwahara filter in the sequential ROI images

(b) Subtracting the sequential ROI images.

(c) Median filter.

(d) Blob finder.

(e) Draw contour.

(f) Extract 20 floating point values from the contour.

Chapter 4

Semi-automated Labelling System and YOLO

4.1 Semi-Automated Labelling System

According to Santos Ferreira[62], supervised deep neural networks have outstanding achievements in many areas, especially in image recognition, but manual data labelling takes tremendous time for completion. Therefore, we developed a semi-automatic labelling system to prepare the necessary information from the dataset relatively quickly before training. We assume that the required training images per class are at least 1,000 images, but it takes a tremendous amount of time to do these tasks manually. This preparation system transforms original images into trainable images with labelled data, but the output may not be perfect in every case. For that reason, the user must still deal with filtering the appropriate samples from the labelled images for training.

Overall, the labelling system has three stages. The first is that the selected algorithm produces label information of Yolo and FDs for training automatically. The second stage is that the user selects the appropriate labelled data from the images produced by the first stage. The final stage is the final confirmation of the selected data from the second stage, avoiding any mistakes in the second stage. These three stages are used for each cycle in the cycle boxes of figure (4.1).

Figure (4.1) illustrates the process of obtaining the final labelled data using the semi-automated labelling system. There are three divisions of cycling, which depend on the different techniques and datasets. Those three cycles are closely related; the second cycle produces labelled data using the created Yolo weight from the first cycle, and the third one also uses the second Yolo weight in the figure (4.1). Therefore, the number of labelled data has been improved over the cycles. After the third cycle, the system is able to produce more labelled data if the user finds other suitable video files for labelling. The detail of each cycle will be discussed after sections 4.2 and 4.3.

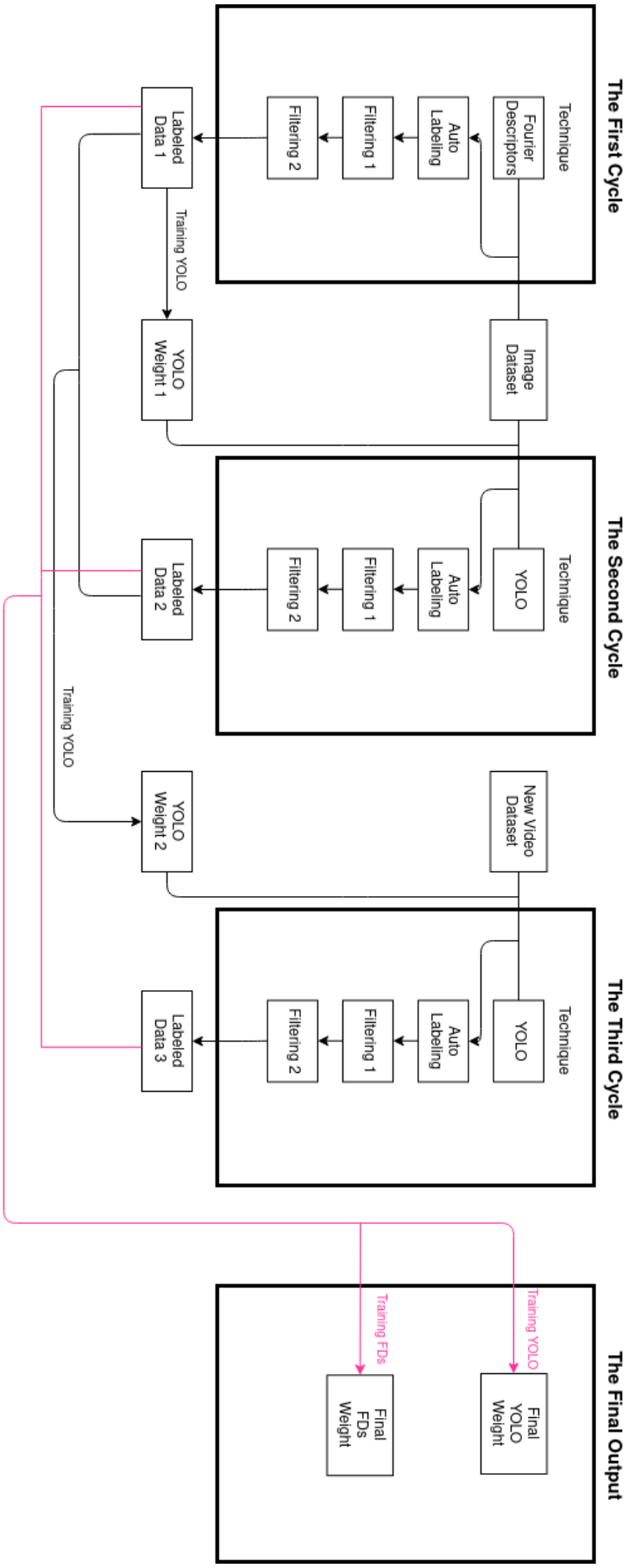


Figure 4.1: The Semi-automated Labeling System Process Diagram

4.2 Analysis of The Dataset

4.2.1 Dataset Type

The choice of each image in the dataset is dealt with care because the accuracy of the algorithm highly depends on them. There are four considerations to determine the dataset. The first is to select the dataset that contains similar species with a wide variety in the targeted area. The detection algorithm is based on the silhouette of the target. The shape of some random species does not match with the target area, so the detection algorithm might struggle to distinguish the target correctly.

The second consideration is that the dataset should include various shapes of the posture of an object. If the dataset of a bird only has the posture of sitting down, this trained dataset will not be able to detect the flying bird.

The third consideration is that a captured image in the dataset must be sequential, and the recommended gap between sequential images is less than one second. If the gap between images is more than a second, the possibility of background changes might be increased by various factors, such as gust, rain, and sudden light changes. The sudden background changes present noise in an image subtraction, and there is more workload for dealing with this noise.

The last consideration is that it secures as many images as possible. We set the goal of preparing data at least a thousand images per class. Choosing the dataset, which has less than a thousand images, is insufficient.

A dataset that satisfies those four conditions, is the one collected by Tabak et al.[63], with 74,335 bird images and 68,298 rodent images.

4.2.2 The Process to Read the Sequential Image by Name Tag

The requirement of our algorithm is at least one sequential image of the target image. Mostly, the creator of the dataset organized the file name in a particular rule. The selected dataset from us has the rule of the file name that two unique names and four-digit numbers follow the dataset: "XXXX" at the end of name, such as below name tag, associated with the saving order. When a new unique name has changed, the four-digit numbers reset to zero, so it is unnecessary to deal with the unique names carefully. In terms of implementation of the program in C++, the "glob" function, which loads the directory files, shows the files in alphabetical and ascending number order. This loading sequence using "glob" provides to read the file in sequential image order ideally. The next image of the target one has the numeric value with added one from the target number, and on the contrary, the previous one is the minus one value from the target number. The diagram (4.2) is the example of reading the previous and the following file name. Sometimes, there is the case of failure to read both the previous and next files from the

dataset, which means neither the previous nor next image of the target image, it regards as an exceptional case. In this case, the program skips the further process and reads other images. In the diagram (4.2), it is the case of "Yes" from "Fail to read both previous and next image".

$$Year\_UnitName\_Author\_imgxxxx.jpg$$

where:
Year= The year this picture was taken,
UnitName= The area where this picture was taken with number,
Author= The person who took this picture with number,
imgxxxx= Unique image name

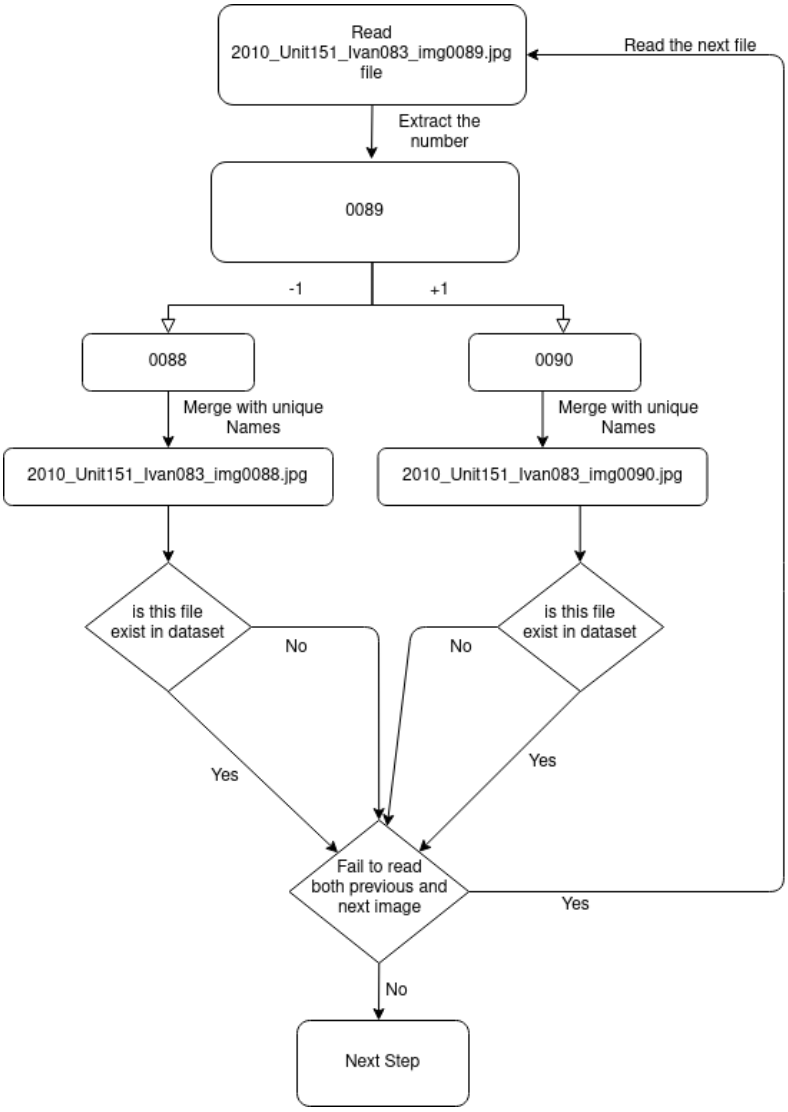


Figure 4.2: The Example Diagram Read The Previous and Next Image File from Dataset

4.2.3 Subtraction Cases in The Dataset

In the dataset, sequential images are organized as the *previous, target, and next* images. In order to generate the labeling system, both the *previous and next* photos are not always need. The requirement of the subtraction process is at least one sequential figure, whether it is the previous or next image. The purpose of combined two subtracted images is to increase the number of success cases when preparing the training image. However, the *previous and next* images are not always provided from the original dataset, so there are 4 cases when reading the previous and the following files, (see below).

1. Only the previous file exists.
2. Only the next file exists.
3. Both the previous and next files exist.
4. Neither the previous nor the next file exists.

The fourth case is not able to be processed because there is no sequential image of the target image. Moreover, the figure 4.3 is the representative case of turning a failure to a success case. Figure (b) of 4.3 successfully extracts the shape of the rodent body except for the tail. In contrast, only the tail is extracted in figure (a) of 4.3. The combined result satisfies the lack of each component. In another example, one of the sequential images is dynamic, and the other is static. For instance, the rodent approaches for feeding. The first action of jumping to food will provide a distinct outline shape in subtraction due to dynamic moves. Eventually, if the final result merges with those two cases, it will succeed. Shortly, both the previous and next photos are not always required for subtraction in the labeling system, but the algorithm combined two subtracted images produces more successful cases for training. However, detection for real-time cannot use this algorithm because real-time detection only provides the *current* and *previous* frame.

(a) sub1=previous-target (b) sub2=next-target (c) merged=sub1+sub2

Figure 4.3: Subtracted image: the case of both thre previous and next files exist

4.3 The Requirements of Yolo Custom Training

Our goal of the labeling system is to organize labeled files in both Fourier Descriptors and Yolo from the dataset. Knowing the requirements for FDs and Yolo for training is essential. About FDs, we discussed the requirements for training in the section (3.1.2.2) in chapter 3, but we have not explained about Yolo. In this section, we will discuss the necessary steps for labeling training images in Yolo.

4.3.1 How to Create Custom Weight in YOLO

The benefit of YOLO custom training is that the materials to be prepared are simple, and there are several supported programs for organizing the materials. The requirements of train custom data in Yolo are the image file that the target object is included and the corresponding text file, including class number, the object's center coordinates of x and y, height, and width. Besides, the text file's name must be the same as the image file except for the extension in lines 1 and 2 of listing 4.1. In addition, the five values from the text file contain information about the objects' location and are generalized in the equation of (4.1 to 4.4) to find this boundary box accurately on any size image (figure 4.4). Shortly, the role of supported programs for organizing training data in YOLO is to create the text file, which includes the boundary box's information. Still, the drawback of those programs is that it takes a long time to select the location of the box, owing to be done manually.

$$x = x\_coordinate/width\_image \tag{4.1}$$

$$y = y\_coordinat/height\_image \tag{4.2}$$

$$width = width\_of\_rectangle/width\_image \tag{4.3}$$

$$height = height\_of\_rectangle/height\_image \tag{4.4}$$

(a) Original Image (b) With The Boundary Box

Figure 4.4: Checking the Location of Rectangle Box with The Value of Listing 4.1

```
//Image Name: 2010_Unit152_Ivan089_img0094.jpg
//Text  Name: 2010_Unit152_Ivan089_img0094.txt
//Format: <class> <x_center> <y_center> <width> <height>
      0 0.612500 0.715625 0.312500 0.312500
```

Listing 4.1: Rectangle Box Information

4.3.2 YAML File and Weight

Creating the custom weight requires images, the label of the images containing the target location information, and a link file call as YAML containing the information about the path of images, the total number of classes, and each class name. The detail of YAML is illustrated in listing (4.2). About the weight files, the "last.pt" and "best.pt" files are created while training with prepared materials in Yolo. The file "best.pt" is the highest mAP(mean average precision) score weight during training, and "last.pt" one is the last epoch of training. Therefore, the ideal final weight file is "best.pt".

```
# train and val datasets (image directory or *.txt file with
    image paths)
train: (path of train images)
val: (path of validation images)
test: (path of test images)

# number of classes
nc: 2

# class names
names: ['Rodent','Bird']
```

Listing 4.2: YAML file

4.4 The First Cycle: Using FDs

We discussed the target dataset and the requirements of Yolo and FDs for training. In this section, it closely explains the details of the labeling system. There are three primary internal processes of the labeling system. The first is generating detection, which is the same process of labeling used in chapter 3. The second is that the user selects the right labeled images for training from the results made by the first task. The third one is the final decision of labeling for training. It is the last confirmation of the labeled images and data.

4.4.1 Automatic Data Labeling Using Fourier Descriptors

Automatic data labeling can apply to both FDs and Yolo. In order to use the Yolo technique in automatic data labeling, the trained weight is required, but the trained weight of the FDs is not compulsory in this process. Therefore, the created algorithm for labeling in chapter 3 generates the locational labels in Yolo and 20 CE values in the FDs. This process runs completely automated, but it takes a long time as an average of three hours, dealing with one class, which is about 65,000 images. However, faults can occur during the process because of the long runtime. This issue will discuss in section (4.4.1.1) of this chapter.

4.4.1.1 Using a Bash Script

Automatic Data Labeling generates about 70,000 images in each species. The entry design of the program was generated until reading the whole image in a row from the dataset folder, and the resulting file is not saved until the process was done because of a segmentation fault. The critical trouble is that some minor issues in the program turn into a serious problem when dealing with those many files in a long time, such as long lifetime variable, small leaking memory. Therefore, in order to make the C++ program runtime shorter and reliable for the whole dataset, the program is divided into a C++ and a bash script. The bash script is responsible for running the C++ program repeatedly until it reaches the condition in which all files are read from the source dataset folder. Because of this bash script, this program must work in a Linux environment.

4.4.1.2 Dividing The Program's Role into C++ and Bash Script

One of the benefits of a bash script is to run a program continuously with input which the programmer sets. Using the advantage of combining the bash script and the C++ program, we change the terminating condition of the C++ program, which terminates when 100 successful outputs are produced. Moreover, the C++ program saves the processed index number when finished. The bash script reads the saved index number and informs the start point of reading. This saved index number is the trigger to finish the bash script loop compared with the total number of images in the dataset. Figure (4.5) below and the listing (4.3) describe the details of the process.

```
1         #parameters
2 cls=0
3 mode=0
4 path_of_file=<path_of_the_dataset>
5 idx_file_name=<path_of_index_file>
6 idx=output_the_first_contour_yolo/birds/index.txt
```

```
7
8  emph# echo $idx
9  ./main -mode $mode --cls $cls --idx $idx --data $path_of_file
10
11 emph#read index
12 value=`cat <path_of_index_file>`
13
14 emph#read total number of item in directory
15 total=`ls $path_of_file -l | wc -l`
16
17 echo $total
18
19 total=`expr $total - 5`
20 emph#loop work until index number over total number of item in
      directory
21 while [ $value -le $total ]
22 do
23
24         ./main -mode $mode --cls $cls --idx $idx --data
              $path_of_file
25         value=`cat <path_of_index_file>`
26 done
27 echo "Terminate bash programming"
```

Listing 4.3: Bash Script in filtering

4.4.1.3 The Output Files

The output file of the first filtering in the diagram (4.6) is the result of the section 3.1. In the detail of the organized folder, the folders contain three different types of folders in the figure (4.6), which are "Image", "ROI", and "Contour". The first folder is called "Image" and contains images compressed to 640 by 640 from the original image. These compressed images are used for creating custom training weight in the Yolo technique. The second folder name is ROI which consists of images of the target cropped by the original, and these images intend to analyze object appearance. The last folder, which is the name contour, contains the images, with an added red outline drawn on the cropped image. This red outline on the cropped image serves as a subjective criterion at the next filtering step for determining whether the image is suitable for training the Fourier Descriptors technique. Besides, there is an additional file in the "Contour" folder, which is "contour_(folder_name).txt" in the figure (4.6). This file consists of information about

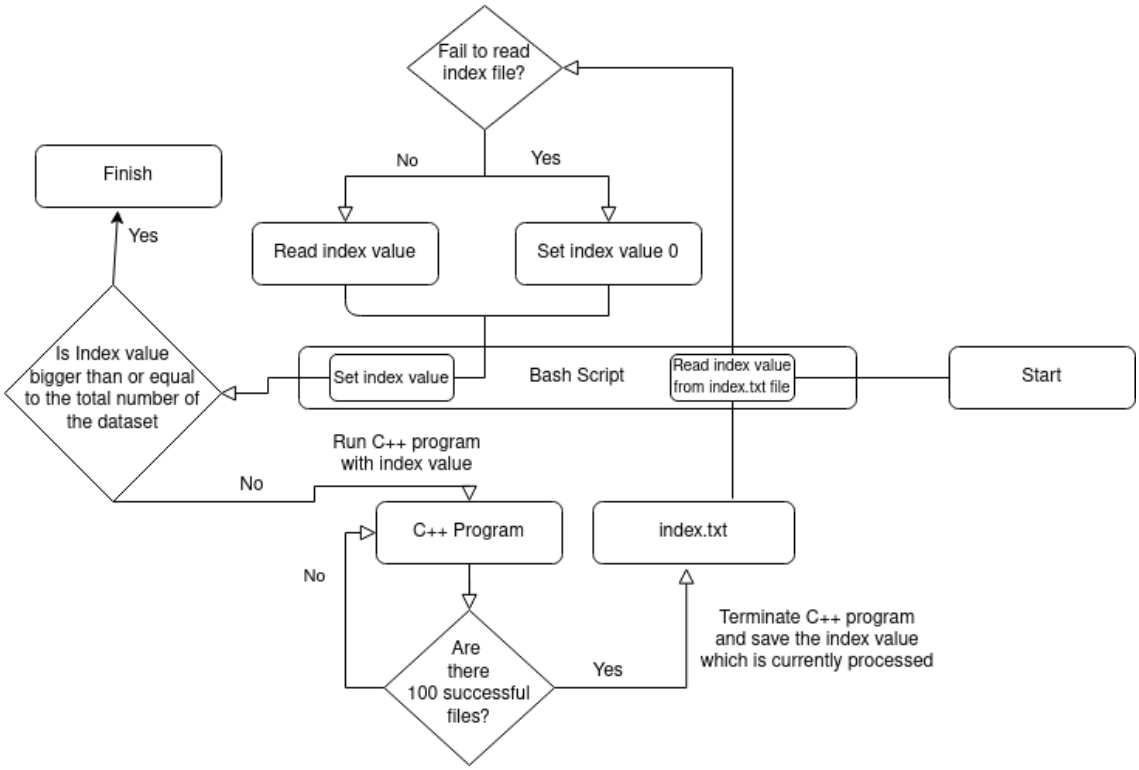


Figure 4.5: The Diagram of Bash and C++

the class number, filename, the requirements of the YOLO custom training, and the Fourier Descriptors training components, 20 floating point numbers about the red contour, respectively.

4.4.2 Filtering 1: Selecting Suitable Images

The primary goal of filtering 1 is to filter out the unusable images produced by the previous process. Even if we choose the dataset with some considerations, the result of automatic data labeling is not always suitable for training due to some failures of the algorithm for the labeling, including dramatic weather changes and images taken at night. In order to deal with some of these failures, the user can decide the correctly labeled image manually. Therefore, the filtering system by the user is essentially required to complete the transformation of the labeled dataset from the collection of images.

The method of filtering 1 is that the user manually chooses the suitable images because the criteria of appropriate images and labels for training are justified by the user's opinion. Besides, this process requires a great deal of time and has been carefully managed because it is closely relevant to the accuracy of objects. Briefly, there are two primary processes, which are merging multiple images as grid view and saving image information, which the user selects.

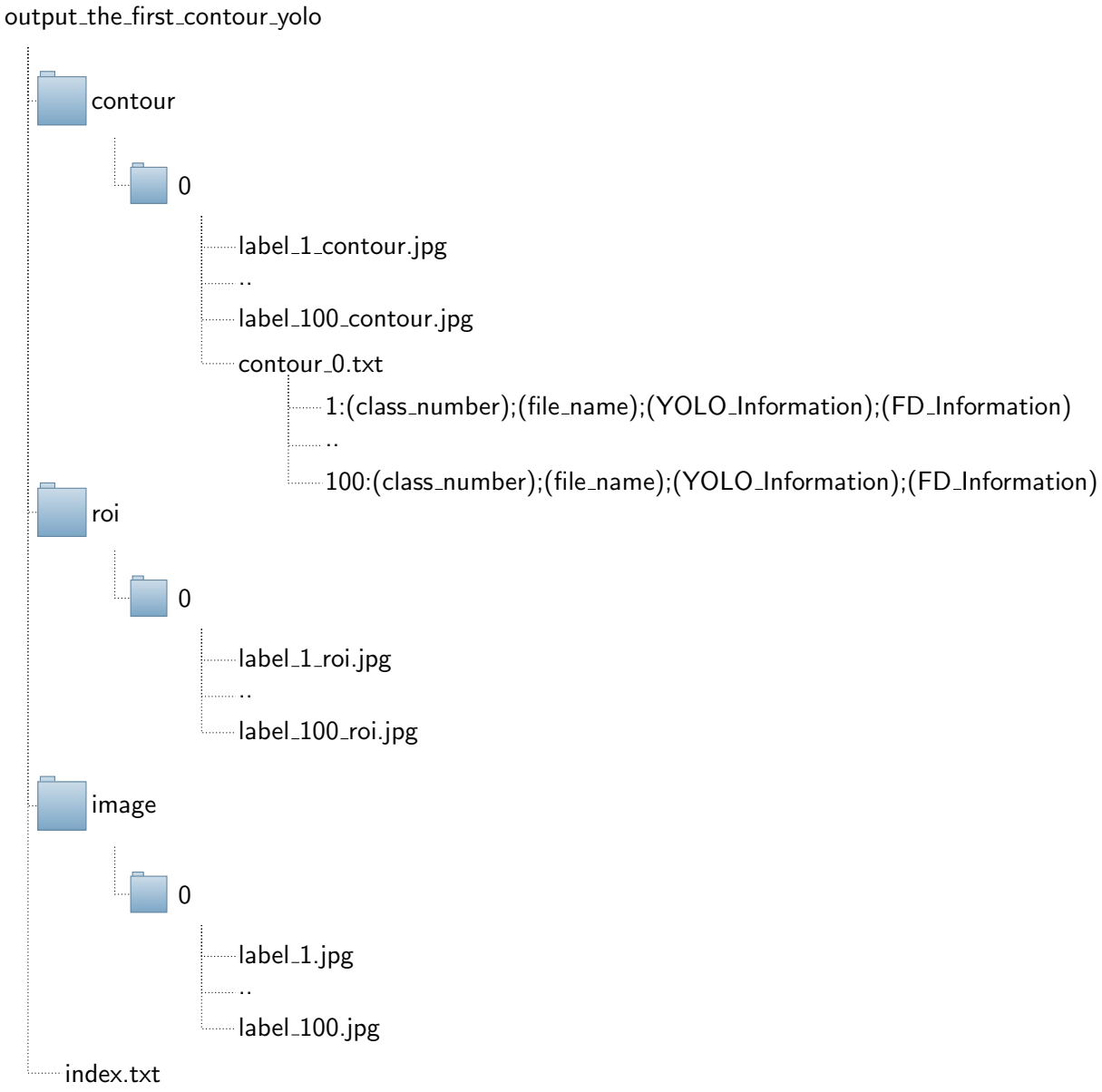


Figure 4.6: Contents of Output and Hierarchy

4.4.2.1 The Beginning Version of Filtering 1

The beginning version of filtering 1 deals with images one by one. Pressing the "t" button if the user regards the sample as correctly labeled, the "f" button is used to skip and move on to the following image. The user might pick the suitable images one by one, but processing approximately 65,000 images takes an enormous time for filtering 1. There is no benefit in creating this system one by one because of the long time required. For this reason, the system has changed as showing and processing multiple images using a grid view of the images.

4.4.2.2 Merging Multiple Images as Grid View

Handling multiple images at once demands more intricate processes than one by one. This intricate process is based on the "contour_(folder_number).txt" file. The file "contour_(folder_number).txt" (which is the "contour_0.txt" file in figure (4.7) is the representative data file of its folder number. For example, the "contour_0.txt" file in figure (4.7) contains the class, Yolo labels, and 20 FDs of 100 images from 0 folders. This file is connected with every 100 images from all 0 folders of "contour", "image", and "roi". Shortly, this is the navigating file for loading and saving the different types of images and data.

About the primary processes of creating the grid view in filtering 1, there are two significant steps in processing multiple images: tying data and images using indexes and showing these linked images after creating grid view.

The first step is to link data from the "contour_(folder_number).txt" file and images from each folder, including "contour", "image", "roi", using the file name, that is the red and green line the figure (4.7). The file name is the index key that connecting the images with the data, and the name format with the image consists of a name with underscore and type such as "name_(type).jpg"; "name_contour.jpg" in the "contour" folder and "name_roi.jpg" in the "roi" one. For instance, the contour case transforms the image name "2010_Unit151_Ivan083_img0091.jpg" into "2010_Unit151_Ivan083_img0091_contour.jpg", and the ROI case transforms into "2010_Unit151_Ivan083_img0091_roi.jpg". Except for the type name behind the underscore, all names in the three folders are the same. Therefore, it is able to link and load three different types of images using the name provided from the "contour_(folder_number).txt" file.

The following step is to create a grid view with the loaded images. The size of images must be equivalent to combine multiple images into one in the OpenCV library, so all images from the "contour" folder unify to the appropriate size as 20 by 20 using compression. In the next stage, after the compressing process is done, the program writes the temporary index number onto all cropped images in figure (4.8), and all images are merged into one window (figure (4.9)). About the grid view design, it intends to promptly

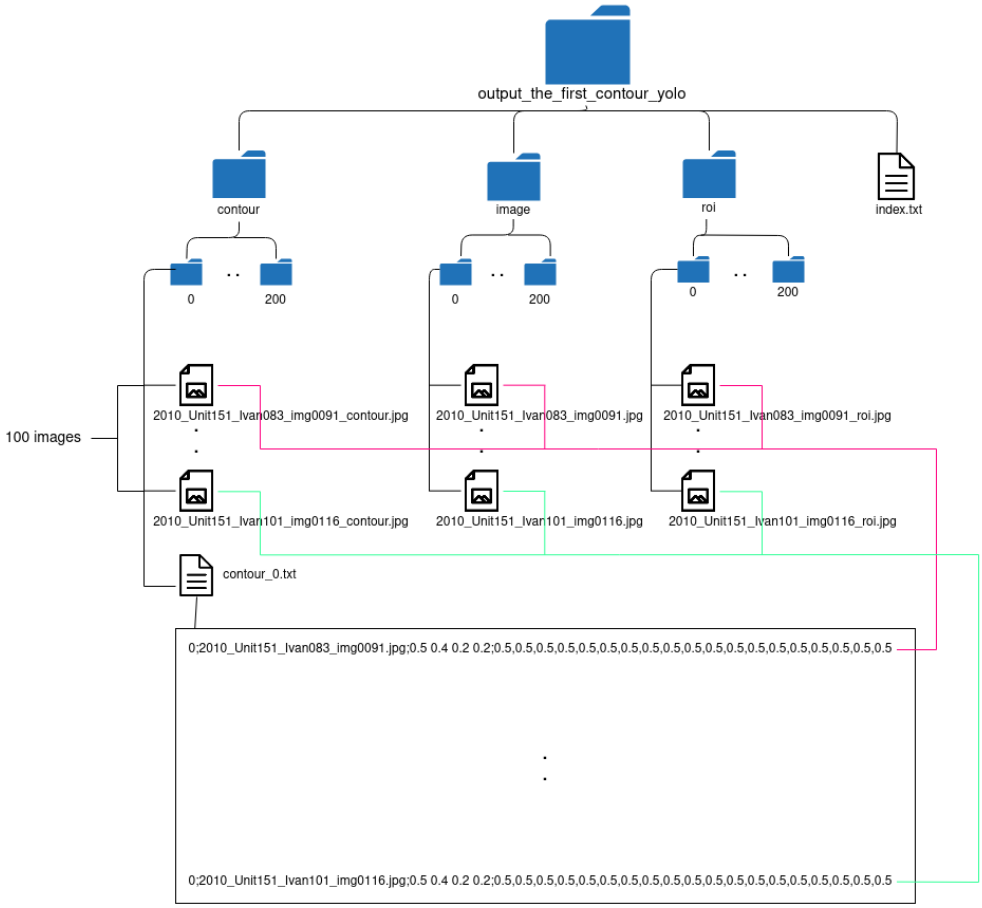
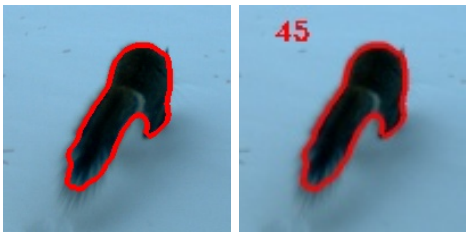


Figure 4.7: File Connection

recognize the index value and the outline of the target. The number of images shown in the grid view of the figure (4.9) sets as 100 due to the capable size of the monitor 2560 by 1440(16:9). The number of grid view is a changeable setting depending on the resolution of the monitor.



(a) Before Writing Index Number (b) After Writing Index Number

Figure 4.8: Writing an Index Number onto an Image

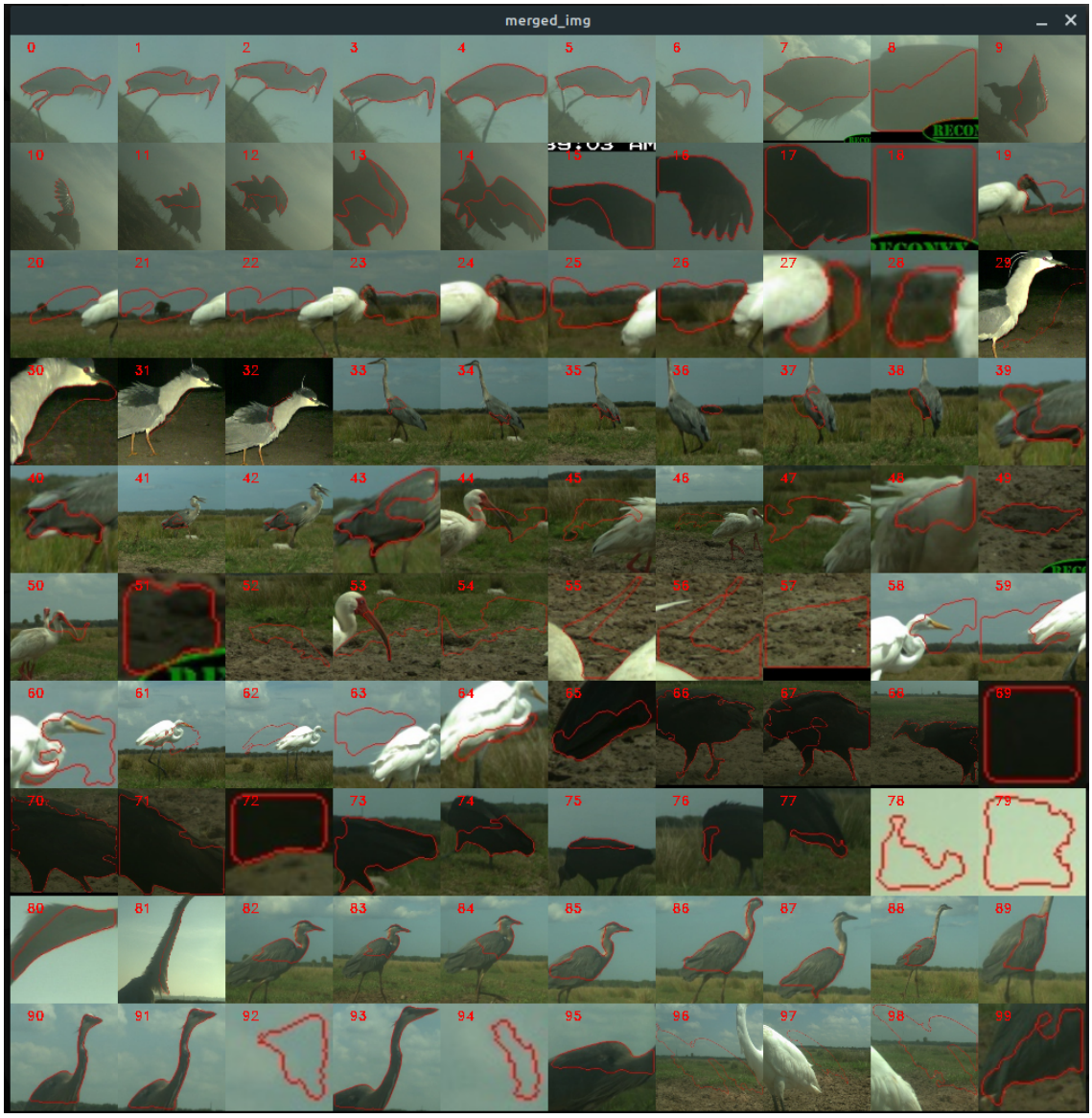


Figure 4.9: Grid View Layout

4.4.2.3 The Method of Selecting Trainable Images

We discussed how to make the grid view window and linking data and images in the previous section. This section will explain the detail of the image selection. When generating the filtering 1, it shows the window of multiple images and the text on the command "Choose what you want:" in the figure (4.10). The index number of images displays on the upper-left corner of each image. The user chooses the appropriate images satisfied in the labels of both FDs and Yolo for training by typing the index number on the command in ascending order, such as the figure of 4.10. Some cases labeled the correct location of the object, but it fails to draw the perfect silhouette of the object, including 0, 1, 2, 3, 50 images from the figure (4.10). This case is correctly labeled in Yolo, but it is insufficient for FDs. We aim to make the comparison of the performance and accuracy between Yolo and FDs, so the labeled dataset must fit the condition of both Yolo and FDs techniques.

Figure 4.10: Select The Suitable Images

4.4.2.4 Dividing The Yolo and Fourier Descriptors Information

After selection work is finished, this program automatedly saves the selected index files. In this section, we discuss the division of files for the subsequent filtering with these selected files. The form of output in automatic data labeling is the merged data (line 2 at listing 4.4), and this data will divide into the "Yolo"(line 6 at listing 4.4) and the "Contour" (line 8 at listing 4.4) forms in the result of filtering 1. The reason for this division is to transform into a suitable form of labeling for training in each technique. Between data, the class number and file name are the essential requirements for both the Yolo and FD training. The green information saved in Yolo, and the red one save in FDs in the figure (4.11). The filename in FD is a resource to load the image from the dataset, and it is a role as an index for Filtering 1 and 2. Moreover, the filename in Yolo is a fundamental element to link between image file and label text file by setting up the same name with image except for the extension. About the division method, the semi-colons are the key delimiter for splitting up FD and Yolo information at line 2 in listing (4.4). All FD information is saved in the single file name as "contour.txt" at the bottom of the figure (4.11). On the other hand, each Yolo information is saved in the created text file, which is the same label as the image name "(filename).txt" inside the "yolo" directory, there is an example in the figure (4.11) of the "output_the_second_contour_yolo" folder hierarchy.

```
1        //the file format from the "contour_(folder_number).txt"
             file
2        <class_number>;<file_name>;<x>,<y>,<width>,<height>;<CE0
             >,..,<CE19>
3
4        //the file format from the "contour_(folder_number).txt"
5        //Yolo information
6        <class_number> <x> <y> <width> <height>
7
8        //Contour information
9        <class_number>,<file_name>,<CE0>,..,<CE19>
```

Listing 4.4: yolo and contour information

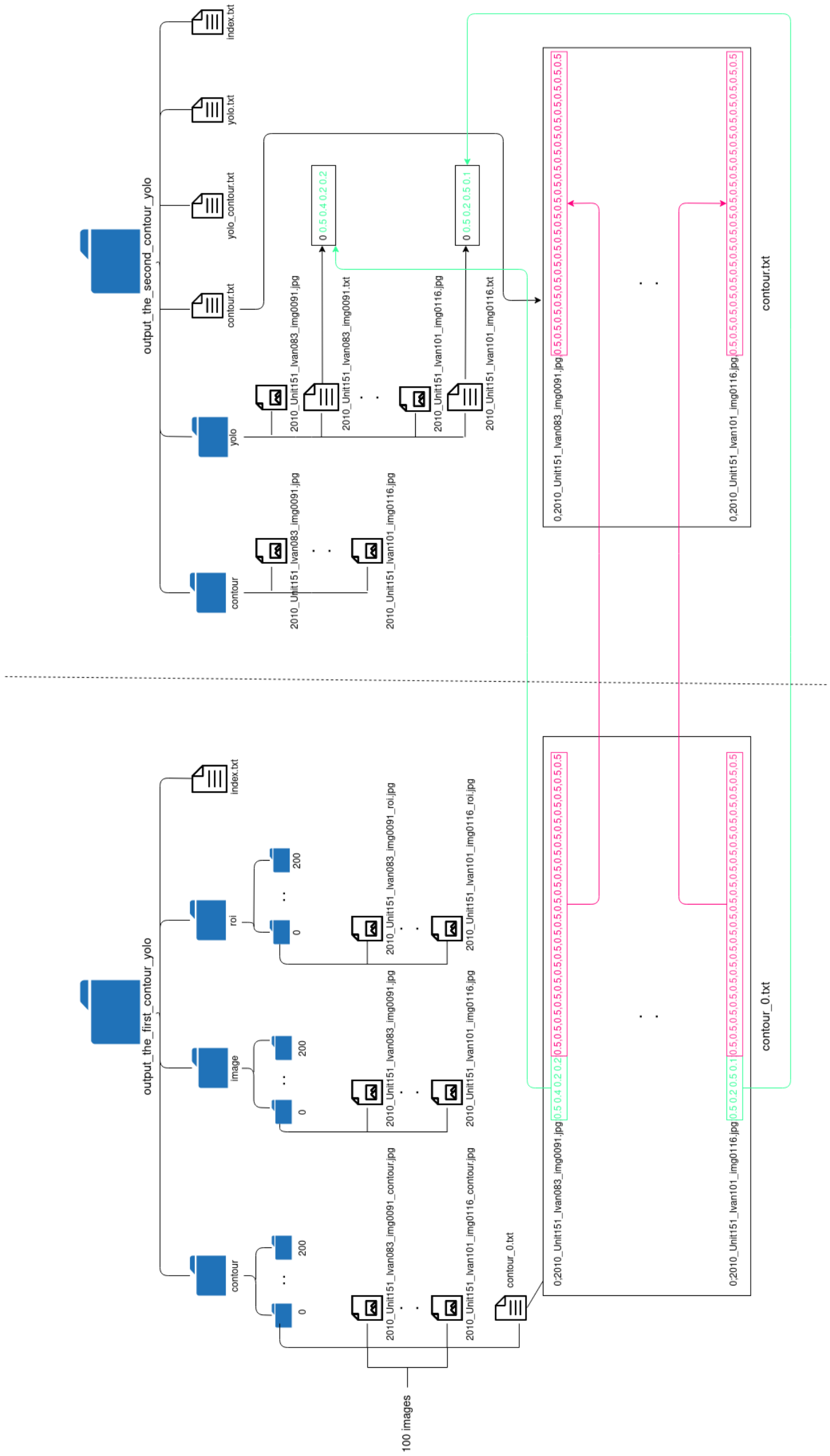


Figure 4.11: The Division of FD and Yolo Information

4.4.2.5 The Output of Filtering 1

As shown below in the figure (4.12), various outputs were created in filtering 1. The text file `contour.txt` is the only requirement for FDs training, and the labeled files in the `yolo` folder are for the Yolo custom training. Although all the data has been arranged to train each technique, we assume to need an extra filtering process to find some mistaken labels from filtering 1. Therefore, filtering 1 has produced some additional files for the subsequent filtering. The text and image for the next filtering are that the `yolo_contour.txt` file is text file linked images and data, and images from the `contour` folder are a cropped object with red line silhouette, which used for creating the grid view.

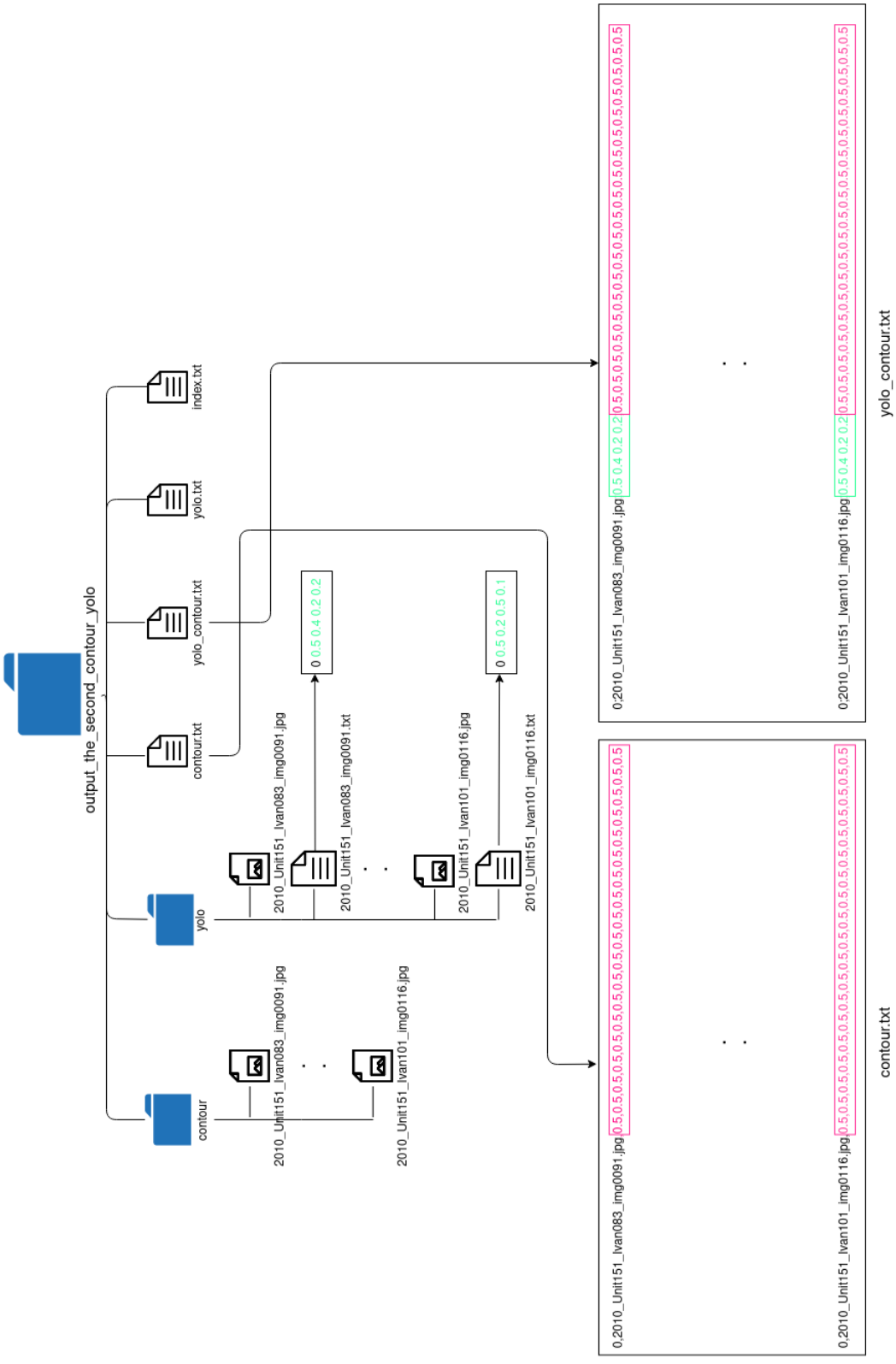


Figure 4.12: Output of Filtering 1

4.4.3 The Filtering 2: Final Confirmation

Filtering 2 has many similarities to filtering 1. The main difference is choosing the indexes that want to delete from the output of filtering 1. Whereas the method of selection in filtering 1 is typing the indexes that wish to save. Most outputs from filtering 1 are correctly labeled for training, so writing down index numbers, which the user wishes to save, in command requires a great deal of time-consuming. That is why changes the selection style to type the indexes that do not include the dataset for training. The form of outputs in filtering 2 is the same as in the previous filtering, and the great advantage of filtering 2 is able to filter continuously using its output. In other words, the output of filtering 1 can be an input resource for filtering 2. This system enables filtering again after analyzing the factor of decreasing accuracy. Finally, this system is establishing concrete labeled data for training with the repetition of the filtering.

4.5 The Second Cycle: Using Yolo

From the first cycle, we transform the image files into labeled data for training. However, a thousand images from around 70,000 in each class are correctly labeled for training. This amount of data is close to our minimum goal for training. It does not show the satisfying accuracy in the FDs technique, which is around 70%, which is illustrated in chapter 5. Therefore, we invented other approaches for creating more labeled data to increase accuracy.

4.5.1 The Fusion of YOLO

The second cycle's idea is to produce the additional labeled data applying the custom weight of YOLO, which is created using the labeled data from the first cycle. Our custom Yolo technique can generate object detection of the rodent and bird's location, and it can create the text file about the class number and locational information of the rodent and bird as changing Yolov5 code. We adopt version five between various versions of Yolo because it is the latest one. The additional reason is that, compared with version 2 written by C language, the YOLOv5 code is intuitive to understand process flows, so it is feasible to rearrange the code to return the result of detection as a text file, including class number, class probabilities, and boundary box information. This text file is a substitution for the process of predicting the target location of FDs in chapter 3. The process details are illustrated in the figure (4.13). The fusion of YOLO detection in the second cycle expects to handle some failed cases of the failure to find the correct location of the target from the first cycle.

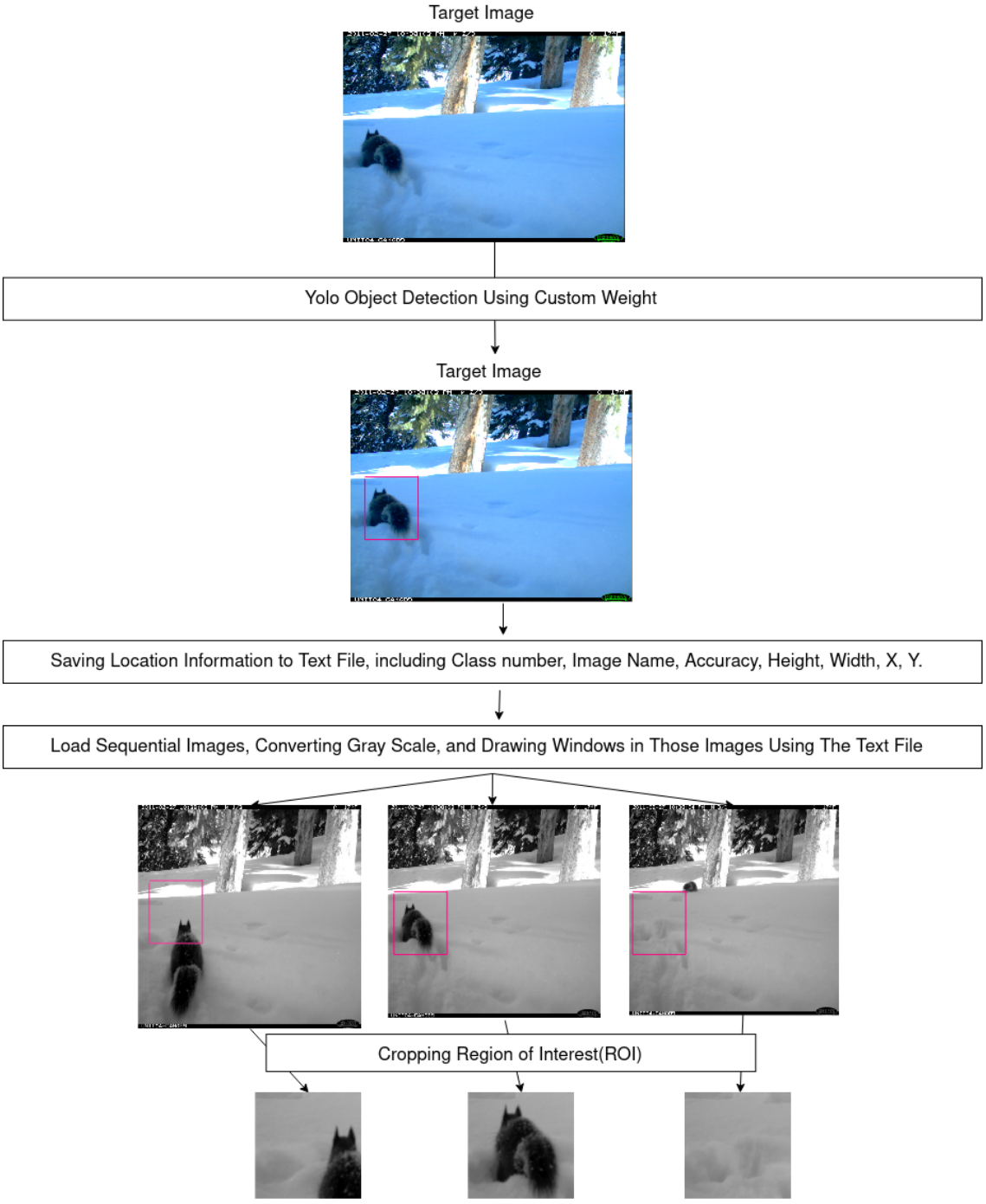


Figure 4.13: The Determination of The Target Location in The Second Cycle

4.5.2 The Text File

The main difference between the first and second cycles is to be supported by the Yolo technique about knowing the Region of Interest(RoI) in the section 3.1.1. We mention that the text file is the substitution of locational information of the bird and rodent. Due to this text file, our algorithm can skip the process of the section 3.1.1. This file includes the class number, class probabilities, x, y, width, and height. YOLO detection can capture multiple targets from a single image and the information of various targets saved in a single text file. However, the design of the labeling system aims to detect only one target from a single frame. Therefore, The criterion for deciding a target is required when detected multiple objects. The criterion is a combination of the correct class number and high-class probabilities. Besides, the text file's name is the same as the based image except for the file extension, and this is the clue to load the relevant image such as the *next, previous,* and *target* image from the dataset. Figure (4.14) is the image combined with text information. The value of 0.82 is the class probability, the boundary box created by text file information, and the label is "Rodent" with the class number zero.

Figure 4.14: Image Combined With Text Information

4.5.3 Second Cycle Process

This section explains the second cycle process using the algorithm diagram 4.15. The process starts with weight creation using the labeled data by the first cycle in the "train" process of the figure (4.15). This created weight is the primary criterion, whether the image is a rodent or a bird.
The detection in Yolo can generate using the source custom weight and the source dataset, which are "Custom weight File" and the files inside the "Initial Dataset" folder in the figure (4.15). In addition to the source dataset, this is the same dataset as the first cycle, and the already labeled images from the first cycle are excluded by reading the labeled images in the "Scan whether the image exists in this folder" process of the figure (4.15). After finishing the operation of Yolo detection, the program returns some text files, which are inside the "Output Texts" folder in the figure (4.15). This text file includes the information of the class number, accuracy, width, height, x and y coordinates, and its name written by the same image name, which is the significant element for loading the corresponding image.

The next step is the loading process of the previous or next sequential image. The sequential images are relevant to the number at the end of the image name; the target number minus one is the previous image, and added one is the following image. In C++, the program can examine whether the previous and next image exists or not as attempting to load those files.

After loading one or two sequential images in the "Load Image" process of the figure (4.15), the program translates the text file information in order to determine the location of the cropping area from images. After the cropping process is finished, the rest of the process is the same as filtering 1 and 2 in the first cycle.

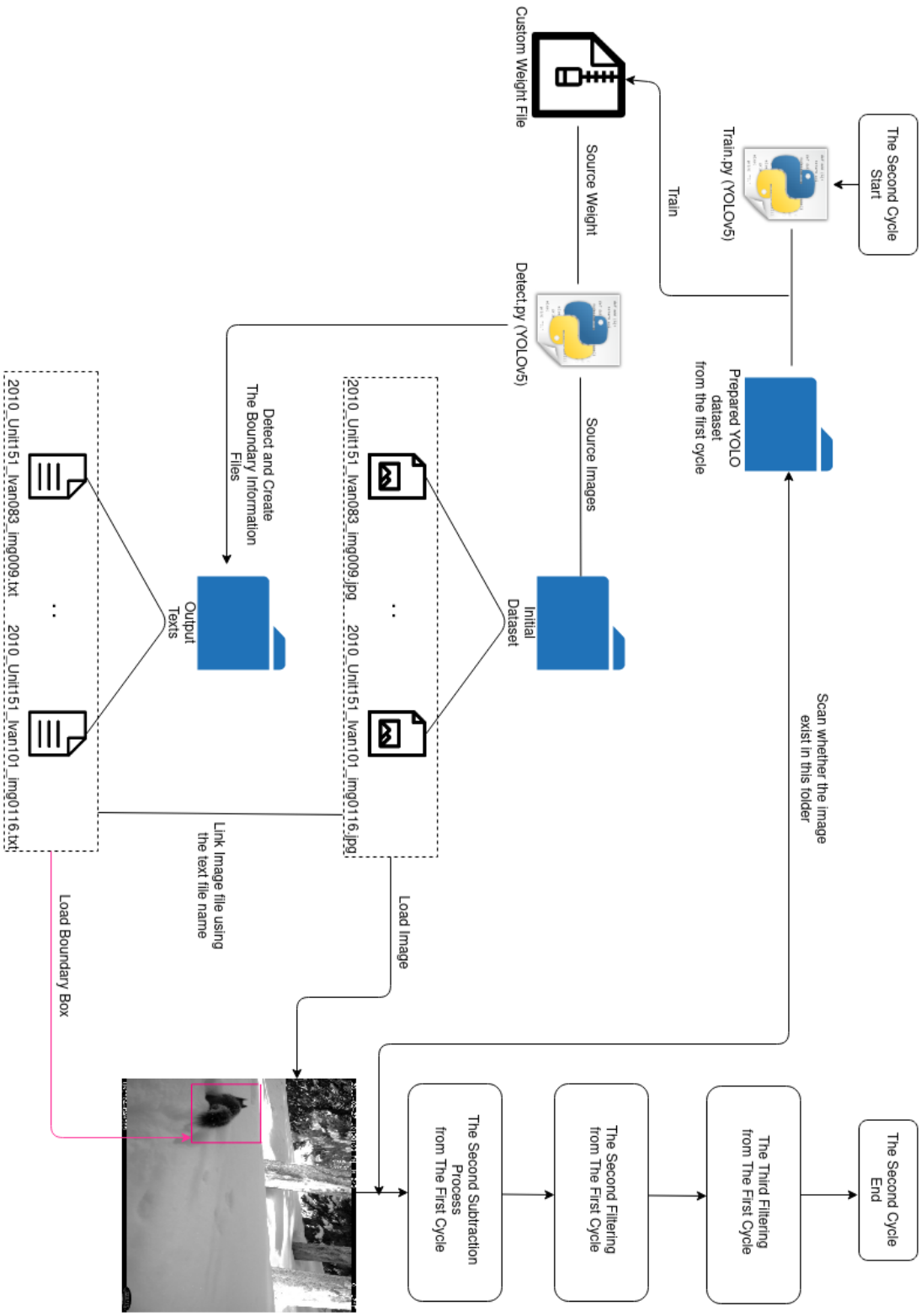


Figure 4.15: The Second Cycle Diagram

4.6 The Third Cycle: Using Yolo with Video file

The second cycle produced 896 rodents and 903 birds. Although the accuracy has improved from 73 to 76 percent in the best training technique, it still does not reach the goal accuracy. Other approaches should be attempted to reach the goal of 80% accuracy. It does not come up with other alternative ideas as transforming the algorithm in order to obtain more trainable data using the dataset, which is the source in the first and second cycles. Therefore, we explore the new dataset for collecting more trainable data.

After dealing with the first and second cycles, there is a more profound understanding of the process in our detection, such as the worst and best cases in our detection and labeling system. We found that two video files, which are [64] and [65], were captured at a fixed angle of the camera, and the best case of our detection can be the dataset. The method is that the Yolo technique generates the object detection using the video file and saves every single frame as an image file. When the object is detected, the program returns its text information, which is the same file in the second cycle. In short, the role of the Yolo technique is that it converts a single video file into a group of image files and returns the result of detection information as text files. Therefore, Any fixed angle videos which are relevant to rodents and birds can be transformed into training images.

4.6.1 The Criterion of Best Case Video as The Dataset

Choosing the dataset video should be dealt with care. There are four conditions for choosing the dataset video. Firstly, the most significant thing about choosing the video is that the angle of the camera must be fixed. If the angle changes dynamically, it is impracticable applying the background subtraction.

The second condition is to show the obvious difference in color between the object and background. After segmentation in our algorithm, the Blob is to find the area showing the significant difference between a sequential image. If the color of the target is similar to its surrounding background, then the subtracted result might not show the location of the object.

The third condition is that the object's color should be darker than its surrounding background. In chapter 3, we mentioned this issue because of the subtraction order.

The last condition is that the movement of the background in the video should be as little as possible. For instance, when leaves and branches are waving in wind gusts, the subtraction part produces more noise, and this makes some trouble discovering the correct

object location and extra calculations in the Blob finder process.

In view of these conditions, the observation camera for feeding the rodents and birds on the snow ground in clear weather is ideal for additional labeling.

4.6.2 The Method of Subtraction

In the first and second cycles, the target image is subtracted by the previous or next sequential images. However, the difference of the sequential images, which are transformed by the video, is that the movement of the target is extremely small because the frame rate of the video is high, approximately over 100 frames per second. For that reason, the result of the subtraction reveals the partial silhouette of the object, which is the second case of "The Fail Cases of Finding Region of Interest(ROI)" in chapter 3. Therefore, subtracting the target image from the previous or next one is unsuitable for the third cycle.

Coming up with this problem, the duration of the video makes short such as 10 seconds, the non-target image set up as the first frame, and subtracts this image with the target image. Finally, the result of detection provides a clear shape of the target. Figure (4.16) shows the process of the third cycle, and it has a similar process to the second cycle, including automatic labeling data, filtering 1, and filtering 2.

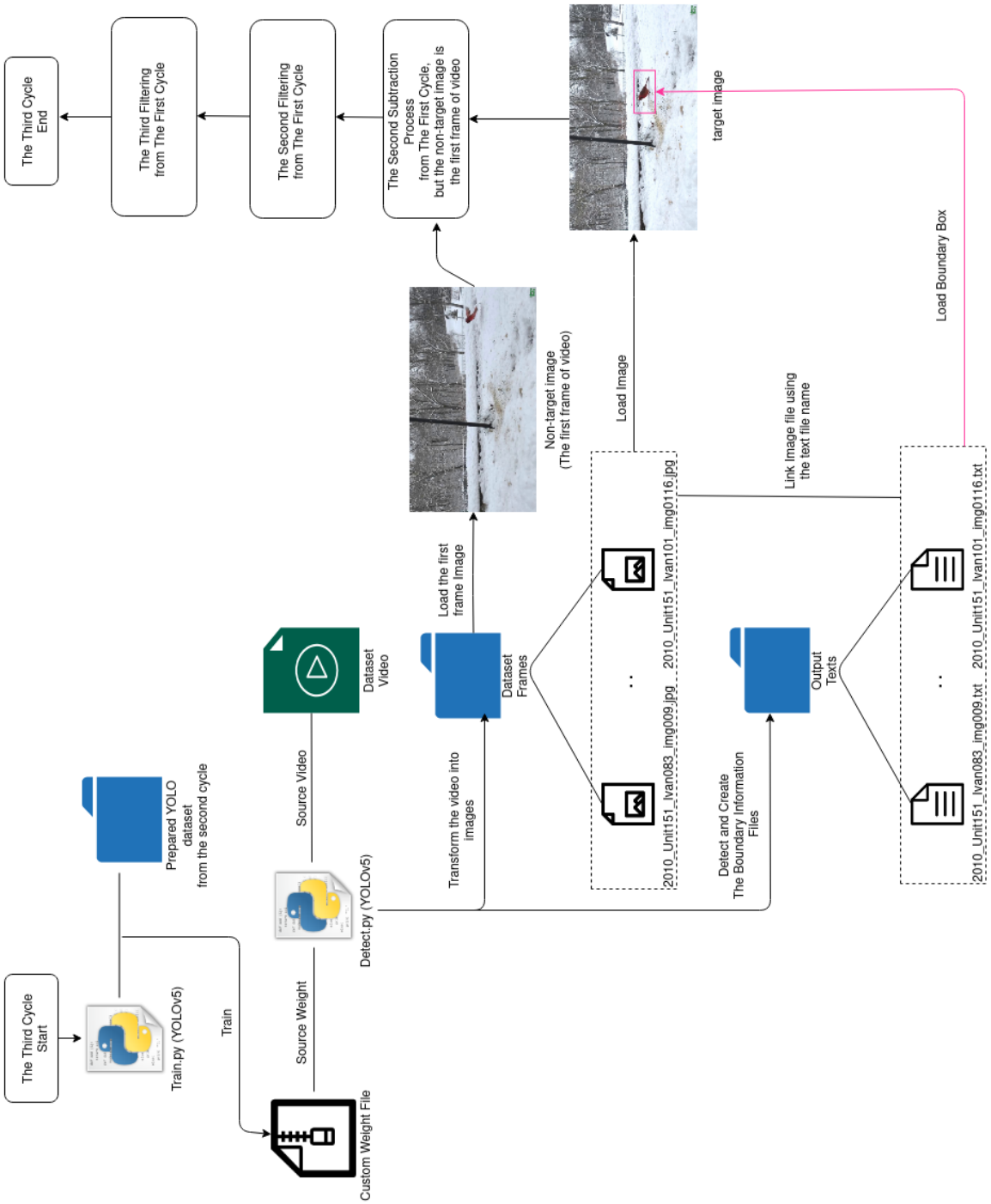


Figure 4.16: The Process of The Third Cycle

4.7 Summary

We explored the process of transforming image data into labeled data by the first, second, and third cycles using a semi-automated labeling system. This system produces the labeled data of 2172 rodents and 3494 birds. The summarized steps of the semi-automated labeling system are described below.

The steps of a semi-automated labeling system:

1. The First Cycle.
   (a) Applying the detection algorithm, which is in chapter3.
   (b) Filtering 1.
   (c) Filtering 2.
   (d) Saving the labeled data both Yolo and FDs.
   (e) Create Yolo and FDs weights.
2. The Second Cycle.
   (a) Detecting an object of the dataset images using Yolo with weights from the first cycle.
   (b) Filtering 1.
   (c) Filtering 2.
   (d) Saving the labeled data both Yolo and FDs.
   (e) Create Yolo and FDs weights.
3. The Third Cycle.
   (a) Detecting an object of the new dataset video using Yolo with weights from the second cycle.
   (b) Filtering 1.
   (c) Filtering 2.
   (d) Saving the labeled data both Yolo and FDs.
   (e) Create the final Yolo and FDs weights.
4. Merging the all labeled data from each cycle in both Yolo and FDs.

Chapter 5

Results and Discussion

We discussed the methodology of the new algorithm using FD and a semi-automated labelling system from chapters 3 and 4. This chapter will analyse labelled data obtained by the labelling system, training and test results of FD and Yolo, and real-time detection in Raspberry Pi.

5.1 Experimental Environment

Most research has been done using a single computer, which is illustrated in table 5.1. All experiments except for measuring speed in Raspberry Pi and training Yolo were generated in this environment. In the part of training in Yolo, we adopted the Colab environment, which is a platform provided by Google, because it does not requires installing additional libraries for training using GPU, and the provided hardware is better than our machine. The detail of the Colab will be discussed in the Yolo experimental section. Moreover, the final goal is to measure the performance of the Raspberry Pi.

Table 5.1: Software and Hardware Specifications

Hardware		**Software**					
CPU	**GPU**	**OS**	**Python**	**PyTorch**	**OpenCV (C++)**	**OpenCV (Python)**	**YOLO v5**
Intel Xeon CPU E5-2620 v3	GeForce GTX TITAN X [GM200]	Linux Mint 20	3.8.2	1.7.0	4.5	4.4	Ultralytics

5.2 The Labelled Data

5.2.1 The Result of Automatic Data Labelling System

We transformed the dataset into the labelled data for training in FD and Yolo using the labelling system. The table (5.2) illustrates the number and percentage of labelled data of rodents and birds from each cycle. It allows comparisons between the labelled number of rodents versus birds from each dataset and assumes the efficiency of the labelling system.

About the dataset, both the first and second cycles choose the same one. However, showing slightly less dataset number in the second cycle is that the second dataset excludes the number of labelled images from the first cycle. Moreover, the third dataset was extracted from the two different videos about rodents and birds. The number of dataset images, which is the value of the fourth and fifth columns in the third cycle of the table (5.2), is measured as the frame numbers of video, and the duration of videos is 40 and 130 seconds, as rodent and bird respectively.

In the success rate, the labelled rates for the first and second cycles on the table (5.2) show a similar trend. But in the third cycle, the success rates show an overwhelmingly high percentage than the first and second one because we adopt videos which consist of the best scenario and environment to extract the usable images.

In addition to the trend of the table (5.2), the number of rodent images shows a lower trend than birds between three cycles, especially in the third cycle. Also, the reason for showing an enormous gap between the number of rodents and birds in the third cycle is that the duration of bird video is exactly four times longer than the rodent one. Finally, according to the labelled rate of rodents and birds in table (5.2), the semi-automated labelling system used in chapter 4 has a slightly better ability of labelling birds than rodents.

Cycle	**Usable Rodent Images**	**Usable Bird Images**	**Original Dataset Rodent Images**	**Original Dataset Bird Images**	**The Percentage of Usable Rodent Images**	**The Percentage of Usable Bird Images**
1	1009	1228	74335	68298	1.3%	1.7%
2	896	903	73326	67070	1.2%	1.3%
3	267	1363	1133	4532	23.6%	30.1%

Table 5.2: Obtained Data

5.2.2 Distribution of Labelled Data for Measuring Accuracy

Table 5.3 compares the growth of the number of rodent and bird images over the three cycles and shows the division of test and train set. The amount of test dataset is divided into approximately 20% for the test set and 80% for the training one. This divided data is used in cross-validation, which is an average measurement of fitness in prediction in order to obtain a precise statistical result of model prediction performance [61] in our experiments.

The figure 5.1 describes an example of cross-validation with 20% for test and 80% for training. To explain figure 5.1, the green data blocks of each iteration in figure 5.1 are used in training a classifier, and the blocks of red in each iteration of figure 5.1 are the section for measuring the accuracy. All accuracy values obtained from iteration 0 to 4 in the red blocks of figure 5.1 are used for calculating the mean and standard deviation in order better estimate the test accuracy. Therefore, despite the complexity of the process, we applied cross-validation.

Table 5.3: Total Number of Samples

Cycle	Rodent	Bird	Test	Train
1	1009	1228	459	1836
2	1905	2131	807	3228
3	2172	3494	1133	4532

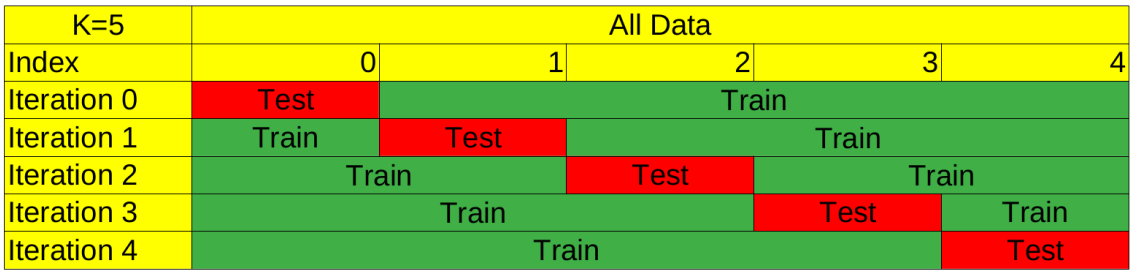


Figure 5.1: Cross Validation Example.

The following steps are used in the cross-validation approach:

1. Determine block size for dividing into training and test sets, K value in figure 5.1.
2. Create the classifier weight using training data, not including the test set.
3. Measure the values for the test dataset using the prepared weights.
4. Create a confusion matrix, comparing labels from a test with the predicted value.
5. Repeat this process, 5 times in the case of figure 5.1.
6. Analyse whether the dataset is reliable or not using mean square error, which is calculated by each confusion matrix in figure 5.1.

5.3 Fourier Descriptors Experiments

5.3.1 Finding The Most Accurate Algorithm

In the previous section, we discussed the labelled data information and the method of the accuracy calculation. There are various algorithms for training the labelled data. We adopted representative algorithms, namely; Neural Network(NN), Adaboost(ADA), Random Forest(RF), and Support Vector Machine(SVM). Tables (5.4, 5.5, and 5.6) illustrate the training runtime, training and test accuracies in each cross-validation cycle.

We set the second priority of technique determination as to the training runtime, which is the second column in tables 5.4, 5.5, and 5.6, because it shows a relatively small amount of time. Moreover, accuracy is the highest priority because our final goal is to measure the accuracy in real-time using a Raspberry Pi, and the training time is not relevant to the real-time detection. The training runtime is only necessary when the results have the same accuracy between the algorithms.

Figure 5.3 shows training runtime in increasing trend when the size of the dataset is bigger. The SVM shows the tremendously fast training time, such as 1.2 seconds in the third cycle, between the four algorithms. Whereas NN is dramatically slower compared with others as 331 seconds, and it spends nearly 350 times more than SVM (figure 5.3).

Table 5.4: Comparison of Accuracy of The First Cycle

Type	Time	Decision	Accuracies					Mean ± Stand. dev.
NN	106s	Test	61.9%	66.4%	61.9%	62.5%	63.2%	63 ± 2%
		Train	99.8%	97.7%	98.6%	99.7%	99.5%	99 ± 1%
ADA	3s	Test	68.6%	72.5%	73.2%	67.5%	72.3%	71 ± 2%
		Train	100%	100%	100%	100%	100%	100 ± 0%
RF	14s	Test	75.6%	73.4%	70.4%	69.9%	74.9%	73 ± 2%
		Train	99.9%	100%	99.9%	99.9%	100%	100 ± 0%
SVM	0.3s	Test	72.1%	72.3%	68.0%	65.1%	71.9%	70 ± 3%
		Train	70.5%	69.0%	71.0%	71.9%	70.2%	71 ± 1%

Table 5.5: Comparison of Accuracy of The Second Cycle

Type	Time	Decision	Accuracies					Mean ± Stand. dev.
NN	136s	Test	66.9%	69.4%	69.8%	67.5%	71.1%	69 ± 2%
		Train	96.8%	94.9%	94.5%	98.2%	95.3%	96 ± 1%
ADA	6s	Test	71.7%	71.3%	72.0%	70.5%	74.6%	72 ± 1%
		Train	100%	100%	100%	100%	100%	100 ± 0%
RF	27s	Test	75.3%	77.0%	74.3%	75.5%	76.2%	76 ± 1%
		Train	100%	100%	100%	100%	100%	100 ± 0%
SVM	0.5s	Test	68.2%	70.6%	68.9%	69.4%	68.5%	69 ± 1%
		Train	70.2%	69.4%	69.7%	69.9%	69.7%	70 ± 0%

Table 5.6: Comparison of Accuracy of The Third Cycle

Type	Time	Decision	Accuracies					Mean ± Stand. dev.
NN	331s	Test	74.7%	74.6%	74.7%	73.5%	74.7%	74 ± 0%
		Train	95.9%	96.6%	97.3%	98.7%	98.5%	97 ± 1%
ADA	9s	Test	78.1%	77.9%	76.7%	78.6%	77.7%	78 ± 1%
		Train	100%	100%	100%	100%	100%	100 ± 0%
RF	45s	Test	79.4%	78.6%	79.3%	79.5%	80.8%	80 ± 1%
		Train	100%	100%	100%	100%	100%	100 ± 0%
SVM	1.2s	Test	74.9%	73.3%	73.0%	74.8%	74.3%	74 ± 1%
		Train	74.0%	74.5%	74.7%	74.4%	74.3%	74 ± 0%

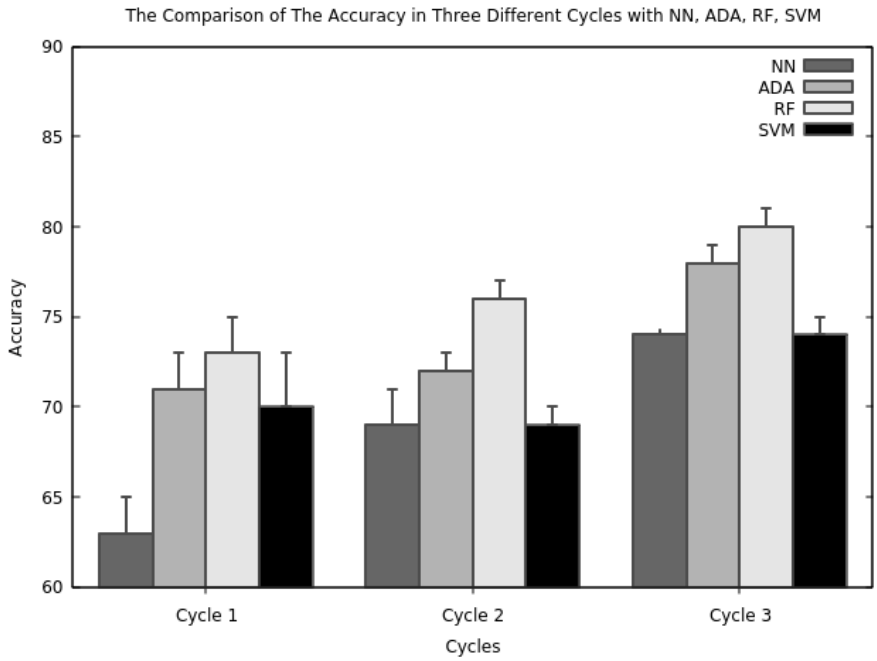


Figure 5.2: Accuracy Graph

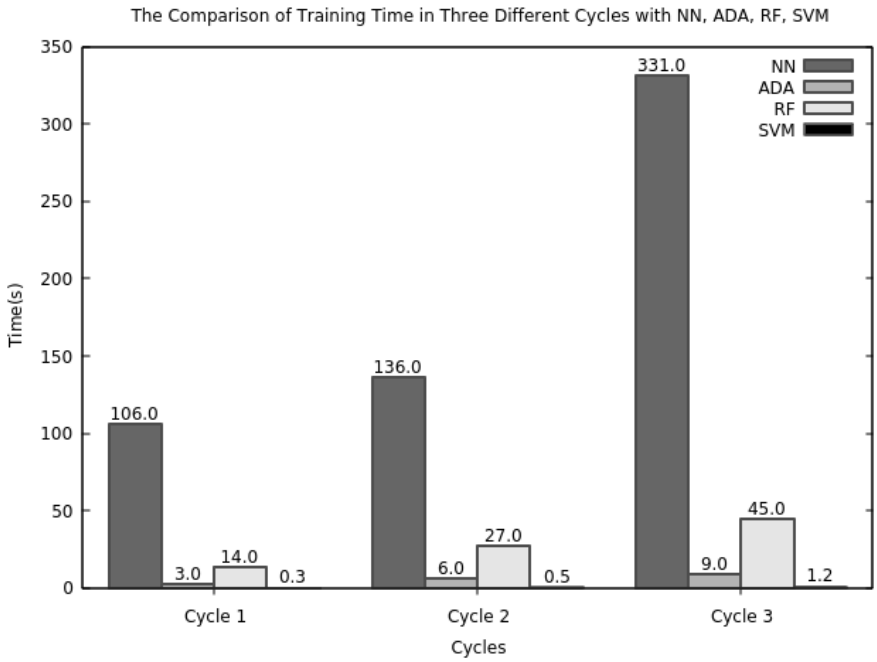


Figure 5.3: Runtime for Training

5.3.1.1 The Mean Accuracies of The Training and Test

Figure 5.2 illustrates the comparison of the mean accuracy obtained from the five cross-validated values using NN, ADA, RF, and SVM algorithms in each cycle. The trend of the mean accuracy gradually has been increased between the first cycle to the third one, except for SVM. So, the increasing number of labelled images influences the growth of the accuracy of the test, except for SVM. Cycle 3 in figure 5.2 is the final labelled data merged with the first, the second, and the third outputs. Therefore, the mean accuracy in cycle 3 is the criteria for choosing the best accuracy between algorithms. In conclusion, RF in figure 5.2 shows the best accuracy of 80%, which is 2% higher than the second-highest (AdaBoost), and spends a relatively short period of time for training.

In addition, the mean average of training is the value to evaluate the reliability of trained classifiers because the same trained images calculate its mean average value. The ideal mean average value of training should be above 95%. Therefore, although SVM has shown the fastest speed for training with reasonable accuracy, we excluded adopting this technique for further experiments, which are to a lack of reliability, shown as 71%, 70%, and 74% in each cycle, respectively.

5.3.1.2 Summary

Finally, we compare the algorithms with the time for training and the mean accuracies of the test and training. Random forest is the appropriate technique to use with our labelled dataset and Fourier Descriptors because it shows the highest mean accuracy of test 80%, reliable the mean accuracy of training as 100%, and sufficient training time as 45 seconds.

5.3.2 Experiment Related to Different Number of CEs Values

We determined the ideal algorithm when using FD by the mean accuracy in the previous section. In the previous section, the mean accuracy was measured using 20 CE values. Each CE is a floating-point representing one Fourier Descriptor. Thus, the few numbers of CEs might not be enough to describe the detail of an object's contour, and the many numbers of CEs might be meaningless. The number of input parameters affects slightly the reduction of the performance. Therefore, it is necessary to experiment with determining the ideal number of CE values. The experiment in this section organizes to analyse the relationship between the accuracy and the number of CEs used by a classifier. We speculate that increasing the number of CE values may result in the growth of accuracy.

5.3.2.1 The Accuracy

Table 5.7 and figures 5.4 and 5.5 illustrate the comparison of accuracy and training time with the different numbers of CEs using NN, ADA, RF, and SVM. Each value in table 5.7 are the mean and standard variation. The time in table 5.7 refers to the training runtime.

In figure 5.5, the accuracy of each technique shows the trend of the logarithmic growth when increasing the number of CEs. The accuracy gradually increases between 1 to 10 CEs. Although RF, which is the red bar column and line in figure 5.5, is the lowest accuracy using 1 CE value such as 59%, it develops the highest accuracy when CE values are 20 as 80%. Adaboost, which is the grey bars in figure 5.5, shows the second-highest accuracy from 3 to 20 CEs and a similar trend with random forest. On the other hand, according to figure 5.5 between 4 to 20 CEs, the number of CEs has no influence on the accuracy for NN and SVM. Therefore, the growth of accuracy correlates with the number of CEs until the 10 CEs, and Random Forest produces the best accuracy in our dataset between algorithms in 20 CEs, but it has similar accuracy to AdaBoost.

	Average Accuracy(Mean Square Errors)						
Type	**1**	**2**	**3**	**4**	**10**	**15**	**20**
NN	**62 ± 1%**	**70 ± 1%**	73 ± 1%	75 ± 2%	74 ± 1%	74 ± 1%	74 ± 0%
ADA	60 ± 1%	68 ± 1%	73 ± 1%	75 ± 0%	78 ± 1%	78 ± 1%	78 ± 1%
RF	59 ± 1%	**70 ± 1%**	**75 ± 1%**	**77 ± 1%**	**79 ± 1%**	**79 ± 1%**	**80 ± 1%**
SVM	62 ± 0%	69 ± 1%	73 ± 1%	73 ± 1%	74 ± 1%	74 ± 1%	74 ± 1%
	Time(seconds)						
Type	**1**	**2**	**3**	**4**	**10**	**15**	**20**
NN	32	52	240	207	440	238	331
ADA	0.5	1	1.8	2	4	6	9
RF	19	14	22	21	32	44	45
SVM	**0.05**	**0.4**	**0.6**	**0.8**	**1**	**1.1**	**1.2**

Table 5.7: The Measurement of Accuracy and Time with Different CE numbers

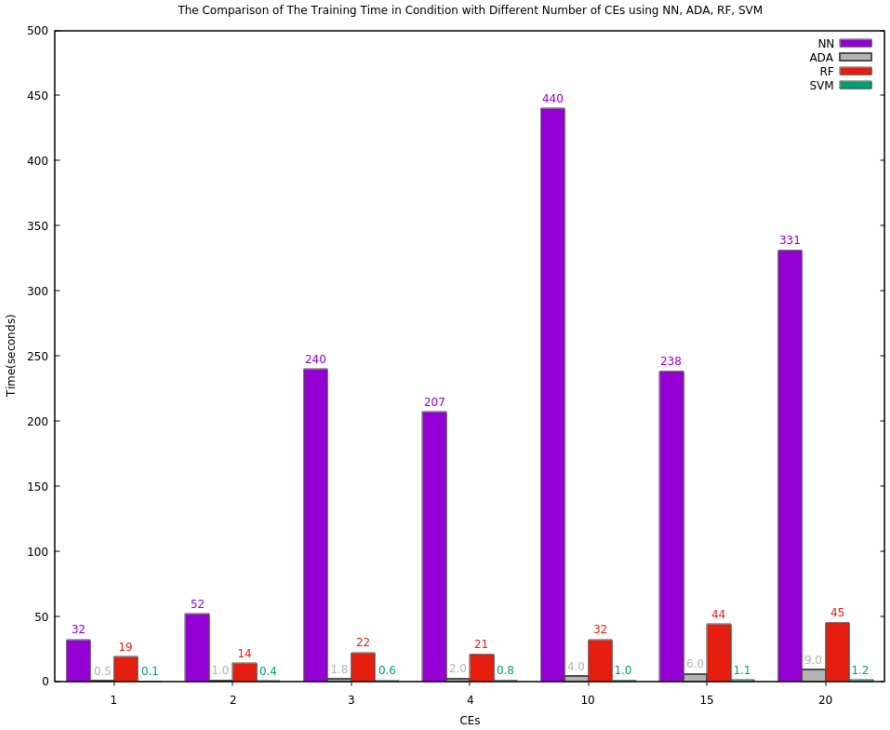


Figure 5.4: Training Runtime in Different CEs

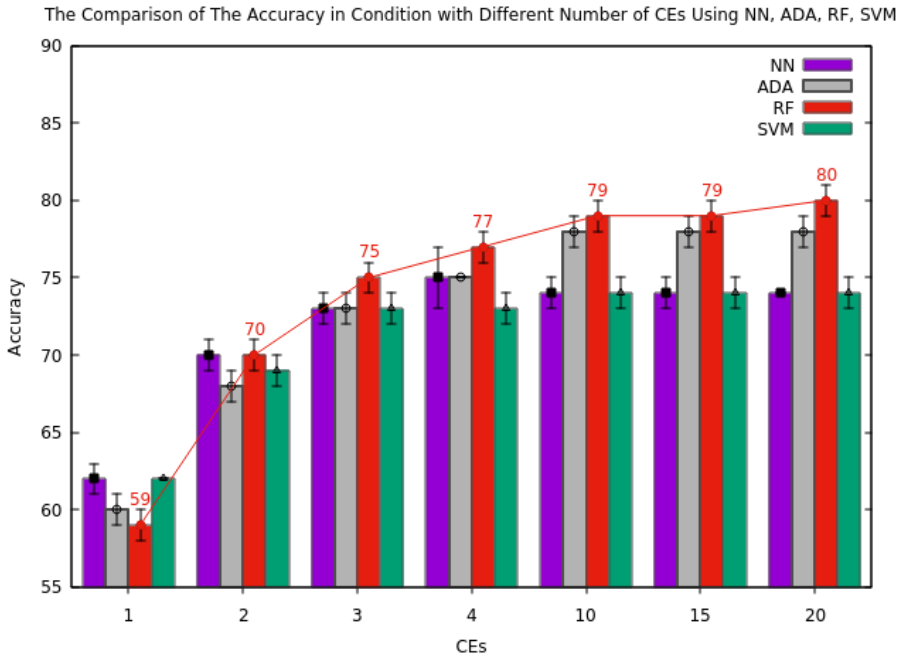


Figure 5.5: Accuracies in Different CEs

5.3.2.2 The Training Runtime

In figure 5.4, it illustrates training runtime for some techniques, including ADA, RF, and SVM, with a different number of CE values. Mostly, the runtime is closely related to the growth of the number of CEs, whereas NN reveals fluctuations without regularity in the occurrence pattern. The priorities in the comparison of different CEs have same criteria as the previous section, so the primary objective in figure 5.4 compares training time between RF and ADA because both techniques show similarly high accuracy in figure 5.5. The training time of AdaBoost is always much lower than RF, but the gap between them is closing. In the 20 CEs, RF training runtime spends five times more than ADA.

5.3.2.3 Summary

We explored the relationship between the different numbers of CEs using NN, ADA, RF, and SVM. Generally, the number of the training input value, which are CEs, is related to the accuracy and training runtime. 20 is the ideal number of CEs because it shows the highest accuracy in Random Forest, even 1% growing up from 10 CEs.

5.4 Measuring F1-score

In the previous section, we measured time and accuracy compared to different CE values in order to adopt adequate training techniques, and Random Forest has shown the best accuracy among Neural Network, Ada Boost, and Supported Vector Machine. However, this research aims to protect bird habitats from rodents, so analysing rodent detection rates is also carefully considered. Even though the general accuracy of identifying a bird or rodent shows relatively high accuracy, the result with the low precision of rodent detection is unsuitable. Therefore, an additional experiment about analysing the accuracy of rodents is required.

In order to evaluate the rodent detection, we have to prepare the calculated data about the precision, recall, and f1 score of the rodents. From the perspective of rodents, precision refers to the correctly detected rate of rodents between all predicted proportions of rodents, and recall is the correctly detected amount of rodents between all actual labels of rodents, and F1 score is the comprehensive score about the quality of rodent detection algorithm [60].

5.4.1 Precision and Recall

Tables 5.8, 5.9) and 5.4.1 reveal average precision, recall, and f1-score about the rodent class calculated by five cross-validated values in each cycle among four machine learning techniques: Neural Network, Ada Boost, Random Forest, and Supported Vector Machine. Precision, recall, and f1-score values in tables 5.8, 5.9, and 5.4.1 are calculated by the equation 5.1. Moreover, figures 5.6, 5.7, 5.8 show the precision-recall curve for the rodent class using the data of precision and recall values from tables 5.8, 5.9, and 5.4.1.

Generally, there is not significant precision gap between in the algorithms in figure 5.6, but the SVM algorithm shows much below precision rate than others in figures 5.7 and 5.8. In addition, the model RF (Random Forest) achieves the highest AP (Average Precision) score as 64%, 73%, and 69% from each cycle, and the SVM is the lowest precision algorithm of each cycle.

About the point of recall, the recall rate in all algorithms is gradually increased from the cycle 1 to 3 and this rate is also higher than precision in figures 5.6, 5.7, 5.8 in general. Analysing the equation 5.1, FP value (the actual is a bird, but it is wrongly predicted as a rodent) is higher than FN (the actual is a rodent, but it is wrongly predicted as a bird). In other words, our algorithm more struggles for predicting a bird correctly than a rodent.

$$P = \frac{TP}{TP + FP}$$

$$R = \frac{TP}{TP + FN} \tag{5.1}$$

$$F_1 = 2\frac{P * R}{P + R}$$

where:

TP = True Positive (Actual label Rodent and Predicted to Rodent)
FP = False Positive (Actual label Bird and Predicted to Rodent)
FN = False Negative (Actual label Rodent and Predicted to Bird)
TN = True Negative (Actual label Bird and Predicted to Bird)
P = Precision
R = Recall
F_1 = F1 Score

Table 5.8: Recall and Precision Table in The First Cycle

Type	**Precision**					**Recall**					**F1-Score**				
	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
NN	0.53	0.62	0.57	0.58	0.56	0.57	0.62	0.57	0.57	0.59	0.55	0.62	0.57	0.58	0.57
Boost	0.63	0.65	0.65	0.56	0.58	0.70	0.70	0.66	0.61	0.70	0.66	0.68	0.66	0.59	0.63
RF	0.68	0.66	0.66	0.58	0.65	0.75	0.72	0.68	0.66	0.75	0.71	0.69	0.67	0.62	0.69
SVM	0.57	0.60	0.53	0.45	0.57	0.74	0.72	0.67	0.65	0.73	0.64	0.66	0.59	0.53	0.64
	Average Precision					**Average Recall**					**Average F1-Score**				
NN	57.2%±3%					58.4%±2%					57.8%±2%				
Boost	61.4%±4%					67.4%±4%					64.4%±3%				
RF	**64.6%±4%**					**71.2%±4%**					**67.6%±3%**				
SVM	54.4%±6%					70.2%±4%					61.2%±5%				

Table 5.9: Recall and Precision Table in The Second Cycle

Type	**Precision**					**Recall**					**F1-Score**				
	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
NN	0.67	0.67	0.70	0.64	0.71	0.64	0.68	0.67	0.66	0.69	0.66	0.67	0.69	0.65	0.70
Boost	0.68	0.67	0.69	0.64	0.67	0.75	0.73	0.71	0.72	0.71	0.71	0.70	0.70	0.68	0.69
RF	0.72	0.75	0.72	0.70	0.76	0.76	0.77	0.74	0.74	0.75	0.74	0.76	0.73	0.72	0.75
SVM	0.61	0.59	0.59	0.60	0.59	0.72	0.71	0.72	0.71	0.72	0.66	0.65	0.65	0.65	0.65
	Average Precision					**Average Recall**					**Average F1-Score**				
NN	67.8%±3%					66.8%±2%					67.4%±2%				
Boost	67%±2%					72.4%±1.6%					69.6%±1%				
RF	**73%±2.4%**					**76%±1.3%**					**74%±1.6%**				
SVM	59.6%±0.9%					71.6%±0.5%					65.2%±0.4%				

Table 5.10: Recall and Precision Table in The Third Cycle

Type	**Precision**					**Recall**					**F1-Score**				
	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
NN	0.64	0.64	0.65	0.64	0.65	0.68	0.68	0.68	0.66	0.68	0.66	0.66	0.66	0.65	0.66
Boost	0.67	0.66	0.66	0.65	0.63	0.74	0.74	0.71	0.76	0.75	0.70	0.70	0.70	0.70	0.68
RF	0.69	0.69	0.70	0.68	0.71	0.75	0.73	0.75	0.76	0.77	0.72	0.71	0.72	0.72	0.74
SVM	0.50	0.50	0.50	0.51	0.50	0.74	0.75	0.75	0.74	0.74	0.59	0.60	0.60	0.60	0.60
	Average Precision					**Average Recall**					**Average F1-Score**				
NN	64.4%±0.5%					67.6%±0.8%					65.8%±0.4%				
Boost	65.4%±1.5%					74%±1.8%					69.6%±0.9%				
RF	**69.4%±1.1%**					**75%±1.4%**					**72.2%±1.1%**				
SVM	50.2%±0.4%					74.4%±0.5%					59.8%±0.4%				

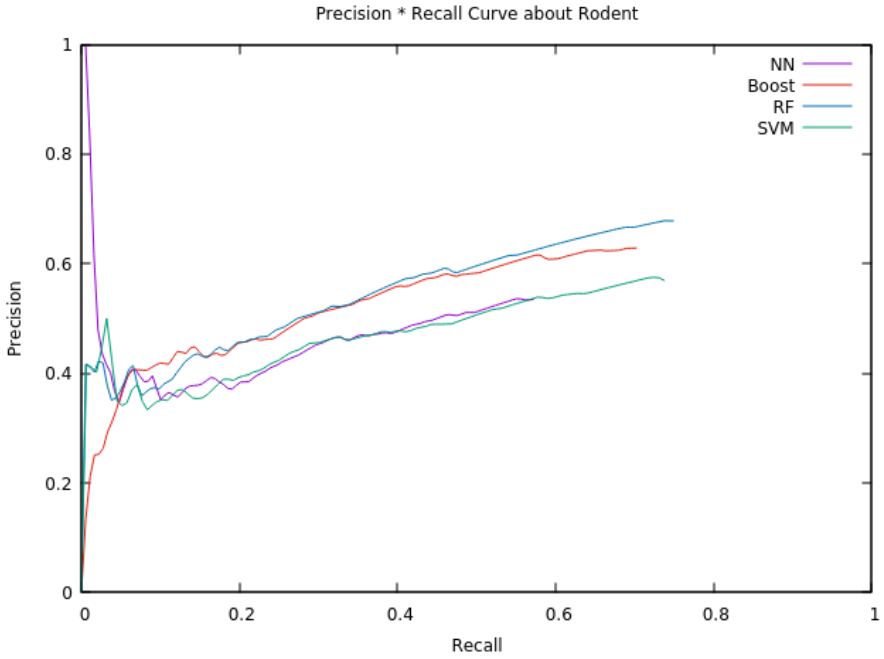


Figure 5.6: The Precision-Recall Graph in The First Cycle

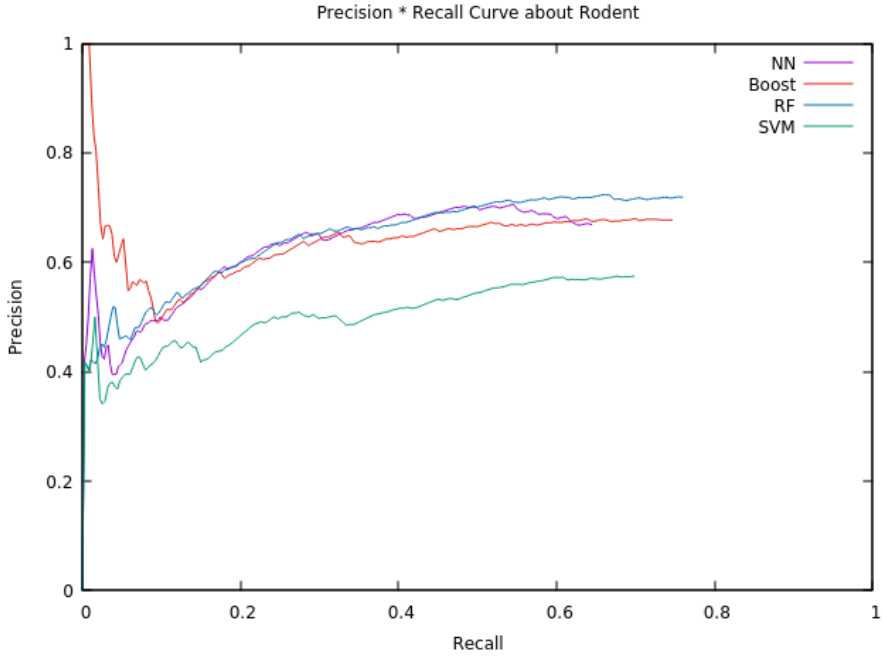


Figure 5.7: The Precision-Recall Graph in the Second Cycle

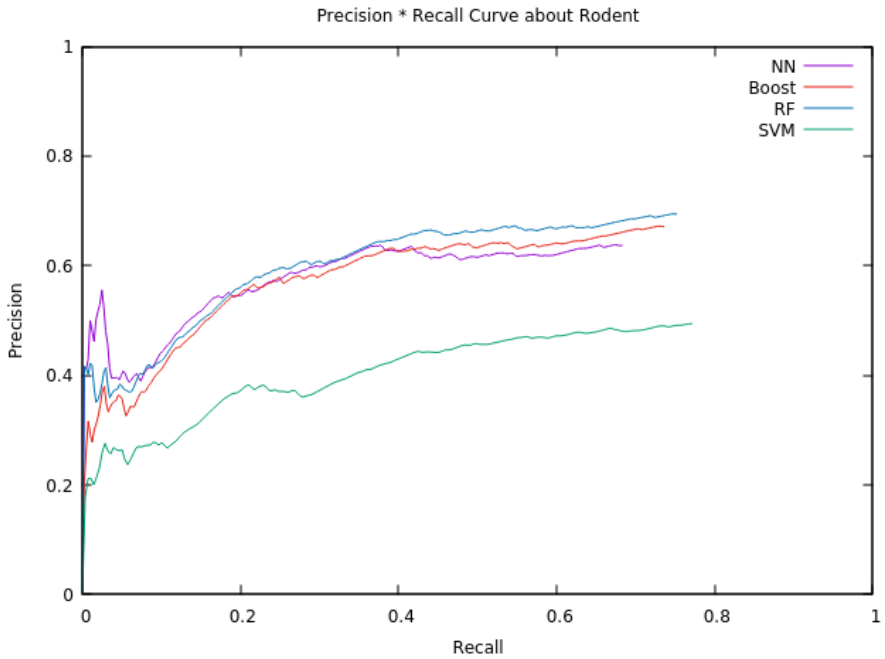


Figure 5.8: The Precision-Recall Graph in The Third Cycle

5.4.2 F1-Score

The column graph in figure 5.9 compares the f1-score of the rodent in three different cycles using four other techniques. The f1-score is the criteria for evaluating rodent detection. The highest score between the techniques in each cycle is random forest as 67%, 74%, and 72%, respectively. To analyse the reason for showing the decreasing trend of f1-score between the second and the third cycle, we organized table 5.11 and figure 5.10 about the ratio of the label number of the rodent and bird dataset for training in each cycle. According to figure 5.10, the first and second cycles show a relatively balanced ratio, and the f1-score in all techniques has increased between those cycles in figure 5.9. On the other hand, the added rate of the rodent in the third cycle from figure 5.10 as 0.164 is conspicuously low compared to birds (0.836), and the f1-score shows the slightly decreasing trend between the second and third cycles in figure 5.9. Therefore, adding the unbalanced number of data might be a factor reducing the f1-score in the third cycle.

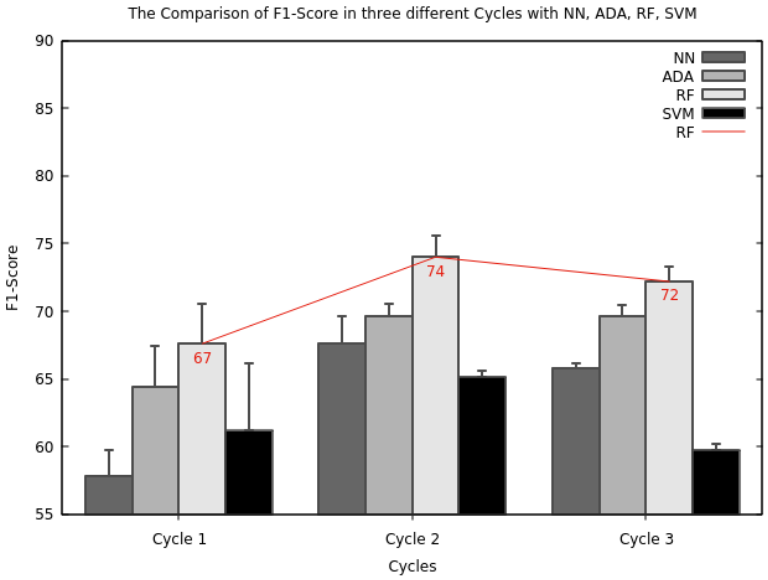


Figure 5.9: Comparison with F1-Score

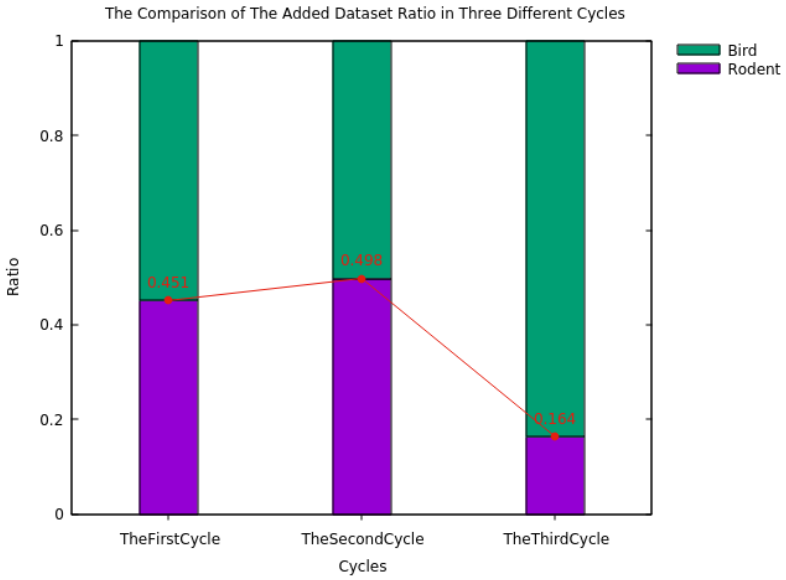


Figure 5.10: The Comparison of Labeled Ratio with Each Cycle

Table 5.11: The Ratio of added data

Cycle	Rodent	Bird	Ratio(Rodent)	Ratio(Bird)
1	1009	1228	0.451	0.549
2	896	903	0.498	0.502
3	267	1363	0.164	0.836

5.4.3 Summary

We analysed the precision, recall, and F1 values for evaluating rodent detection. Adding the unbalanced number of the labelled data, which is the target class number is much less than non-target class, for training might be a reason for reducing the f1-score. Moreover, Random Forest is the highest F1-score at 72% and shows an appropriate rate for detecting a rodent.

5.5 YOLO Experiments

We discussed the Fourier Descriptors in the previous section. This section will examine the change of the average accuracy and training time in each cycle using Yolo version 5 with the fastest architecture. Moreover, we will test the comparison of the average training time and the frame rate in both GPU and CPU using different architectures of Yolo version 5, which are small, medium, large, and extra-large. We can evaluate the accuracy of our labelled data, training time, and the ideal architecture in Yolo version 5.

5.5.1 YOLO Training Environment

Yolo experiment is trained in the google "Colab", which allows users to generate arbitrary python code using google's hardware (table 5.12). The reason for using google "Colab" is that it promises a shorter training time due to relatively better specifications than the local machine.
In addition, we apply cross-validation in order to obtain more precise accuracy and compare the accuracy of FD techniques and Yolo in the following experiment. The training using the fastest architecture in Yolov5 demands relatively longer times than other techniques in FD. Therefore, the k-fold value in the cross-validation was reduced to 4. The proportion of the training dataset modify from 80% to 75%, and the test one is from 20 %to 25%, shown in the table (5.13). The following parameters are used for training in general.

The Parameter of Training:

- *!python train.py –img 640 –batch 16 –epochs 100 –data custom_yaml.yaml –weights yolov5s.pt –nosave –cache*

Table 5.12: Google Colab Specifications

Hardware		**Software**						
CPU	**GPU**	**Platform**	**CUDA**	**Python**	**PyTorch**	**OpenCV (C++)**	**OpenCV (Python)**	**YOLO v5**
Intel(R) Xeon(R) CPU @ 2.20GHz	Nvidia Tesla T4	Google Colab	11.2	3.8.2	1.7.0	4.5	4.4	Ultralytics

Table 5.13: The Distribution of Labeled Data in Test and Training Set

Cycle	Rodent	Bird	Test	Train	The Added Number of Labeled Data
1	1009	1228	574	1722	2237
2	1905	2131	1009	3027	1799
3	2172	3494	1360	4080	1630

5.5.2 The Accuracy and Training Time

This section is organized to observe the change of accuracy and training runtime in each cycle. The training time and accuracies are generated in the Google Colab environment using GPU.

Table (5.14) illustrates the average runtime for training and the average accuracy value, which is calculated by the four different fold accuracies, with three different cycles trained by the light version of architecture (yolov5s.pt) in Yolov5. The training time is measured as an hour, so the 0.5 hour is 30 minutes. Figures (5.11 and 5.12) are provided based the table (5.14) for clearly identify variations in the average accuracy and the training time between cycles, respectively. We will discuss the accuracy graph in figure (5.11) firstly.

Figure (5.11) shows logarithmic growth of the average accuracy. A significant increase in the proportion of the accuracy can be seen between the first and the second cycle from 76% to 96%, and then the third cycle shows slight growth from 96% to 97%. According to the trend of the accuracy graph in the figure (5.11), the increasing number of the labelled data is closely related to the growth of the accuracy. We can conclude that a sufficient number of the labelled data related to the accuracy of the Yolo is around 2000 in each class.

K=4		**All Data**				
Cycle	**Average Training Time (hours)**	**0**	**1**	**2**	**3**	**Average Accruacy**
1	0.581hrs	76.5%	75.2%	72.5%	78.5%	75.6±2.5%
2	0.761hrs	99%	94%	97%	95%	96.25±2.2%
3	1.416hrs	98%	97%	97%	96%	97±0.8%

Table 5.14: Yolov5s Training Time and The Average Accuracy

In contrast, the training time in the figure (5.12) shows the trend in exponential growth over the cycles. The time grew approximately 30% between the first and second cycle and roughly 86% between the second and third cycle. According to table 5.13, the added number of images has been reduced, such as 1799 in the second and 1630 in the third cycle, but the growth rate has been increased. Therefore, the increasing number of the labelled data is also relevant to the growth of the training time, and the training time is growing exponentially.

5.5.3 Summary

We examined to observe the change of accuracy and training time over the cycles using Yolo version 5 with the lightest architecture in the Google Colab environment. Both accuracy and training time has a prominent relationship with the increasing number of labelled data. The accuracy shows the logarithmic growth over the cycles, whereas the training

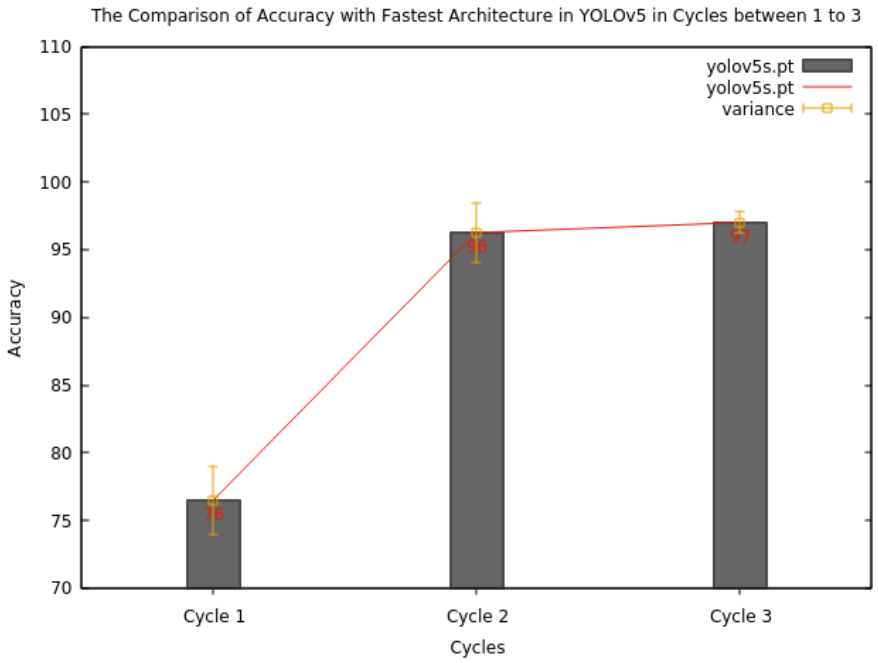


Figure 5.11: Accuracies in Each Cycle

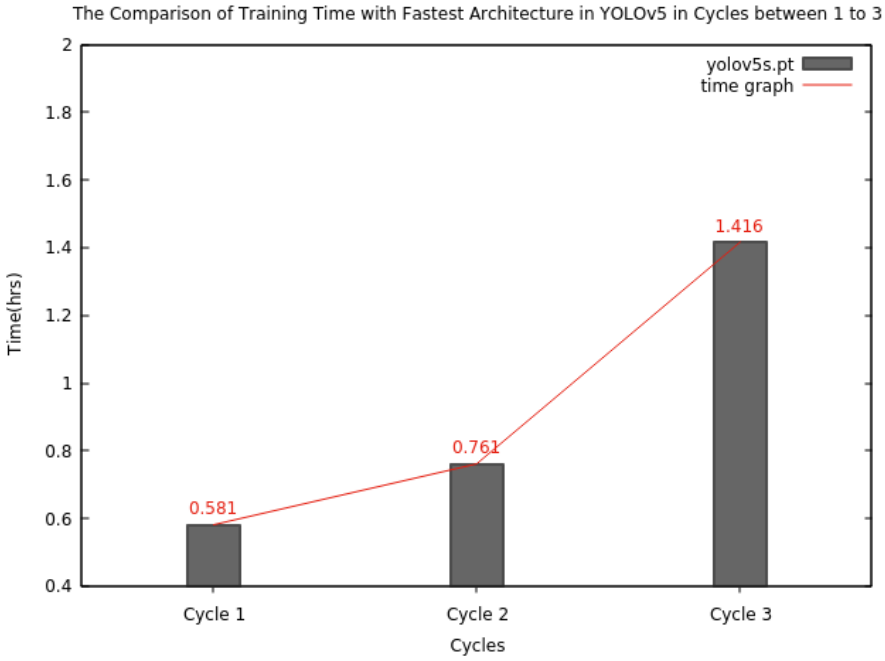


Figure 5.12: Training Runtime in Each Cycle

time is increasing exponentially. The final accuracy and training runtime, which used 1360 test and 4080 training set in the third cycle, are 97% and 1.416 hours, respectively.

5.5.4 The Comparison of Different YOLO Architectures

In the previous section, we analysed the difference in accuracy and training time in each cycle using the lightest architecture from Yolo version 5. The reason for the choice of the lightest model in the previous experiment was that we presumed this model is more suitable for low specification machines. Other architectures might provide a higher accuracy score with adequate performance compared to the lightest one. Therefore, comparative analysis between different architectures, which are small, medium, large, and extra-large, is required in order to determine the fittest architecture with a low specification machine and our dataset.

Table 5.15 illustrates the amount of training time in the dataset of the third cycle, the processing time for a single frame in a second using CPU and GPU, and the comparison of accuracy in small, medium, large, and extra-large architectures. The training is generated in the condition with GPU, 100 epochs, 16 batches, and image size compressed 640 by 640. From "small" to "extra-large", those categories are in ascending order of the complexity and size of architecture, so the "small" is the lightest architecture, and "extra-large" is the heaviest one.

5.5.4.1 The Training Time

In terms of training time, figure 5.13 shows a continued increase exponentially from 1.416 hours in small to 7.665 hours in "extra-large". Comparing to the previous architecture in ascending order, the training time of the next one is slightly less than twice. For instance, the "medium" as 2.636 approximately spends 1.9 times more than the "small" as 1.416 in training. In the end, the gap between the lightest and heaviest is 5.4 times. Therefore, when the complexity and size of architecture have been increased, the training time rises exponentially, and the "small" architecture has a significant benefit of training time comparing the "extra-large" one.

Architecture	**Average Training Time (hours)**	**GPU Detection Speed (Frame Per Seconds)**	**CPU Detection Speed (Frame Per Seconds)**	**Average Accuracy**
Small	1.416hrs	90.90fps	3.70fps	97%±0.8%
Medium	2.636hrs	58.82fps	1.50fps	97%±1%
Large	4.128hrs	38.46fps	0.75fps	96%±0.2%
Extra Large	7.665hrs	25.00fps	0.42fps	97%±0.4%

Table 5.15: The Comparison of Yolo Performance with Four Different Architectures

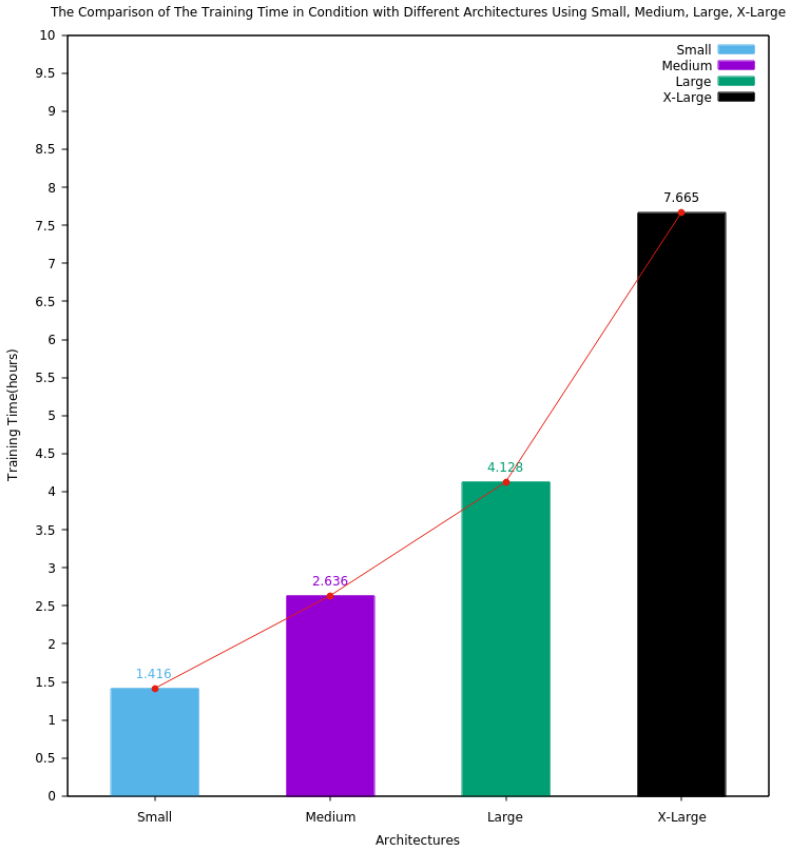


Figure 5.13: The Comparison of Training Time with Different Architectures

5.5.4.2 The Frame Rate between CPU and GPU

Figure (5.14) reveals the data comparison of the third and fourth columns of the table 5.15 about the detection performance of GPU and CPU measured by frame per second. Overall, all architectures show faster processing time in GPU than CPU. The frame rate of CPU detection started at just over 3.70 fps in the category of "small", and then it decreases steadily to reach the bottom of almost 0.42 fps in "extra-large", which is nearly nine times slower than the "small" one. Similarly, the case of GPU dramatically decreases the frame rate from 90.90 fps in "small" to 25 fps in "extra-large". In comparing the fastest CPU to the slowest GPU, GPU in the heaviest architecture "extra-large" as 25 fps shows a significantly expeditious frame rate than CPU in the lightest architecture "small" as 3.70 fps. Namely, the frame rate of detection in GPU is significantly faster than CPU. The CPU in the small architecture of Yolo version 5 is almost 30 times slower than its GPU, and even the lightest architecture of CPU is nearly seven times slower than the heaviest one of GPU.

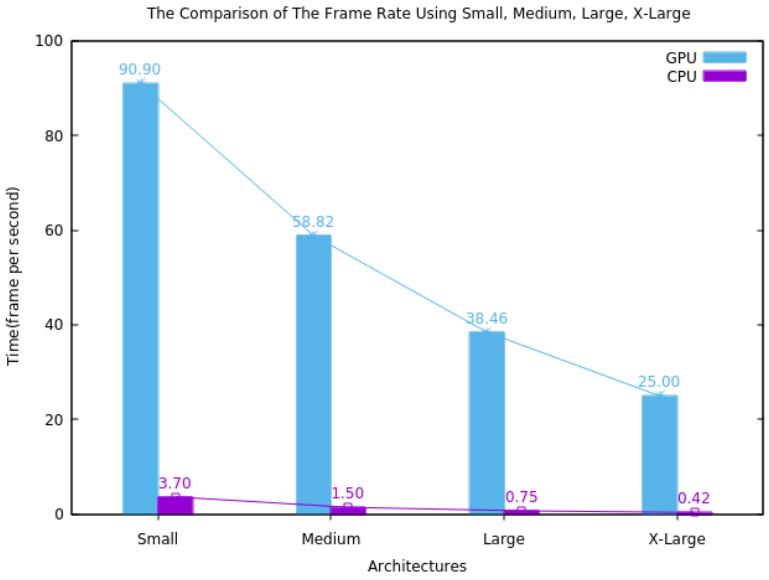


Figure 5.14: The Comparison of Detection Frame Rate with Different Architectures

5.5.4.3 Accuracy

The last comparison is the accuracy between architectures. The accuracy is the highest priority choosing architecture but based on figure 5.15, it does not notice the difference in accuracy among them; all are around 97%. There is a lack of relation between the accuracy and architecture in our dataset. In an explanation of figure 5.15, the accuracy of "extra-large" is the highest in consideration of the variation in the fifth column of table 5.15. Although accuracy is the highest priority, the "small" might be the best architecture to reflect on the results of detection and training time.

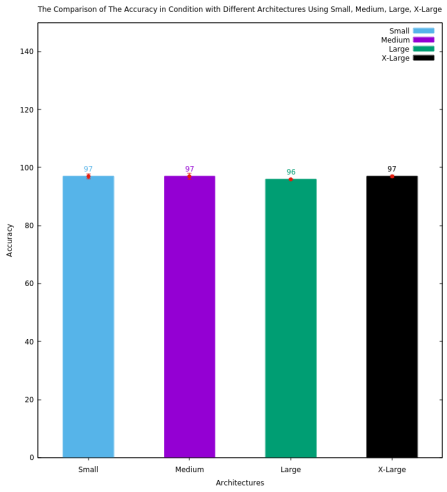


Figure 5.15: The Comparison of Accuracy with Different Architectures

5.5.4.4 Summary

To sum up, GPU and CPU detection performance, training time, and accuracy are examined to decide the best architecture between "small", "medium", "large", and "extra-large" for a low specification machine using our dataset. We conclude that Yolo version 5 has a prominent benefit using GPU from the comparison of detection speed between GPU and CPU. In the point of a low specification machine, the "small" is the appropriate architecture because it shows the fastest training as 1.416 hours, and the highest frame rate of CPU detection as 3.7 fps, which the second-highest accuracy between all architectures, but it is a minor difference from others.

5.6 Real-time Detection: Fourier Descriptors

We measured the performance and accuracy in FD and Yolo techniques using the labelled test and training dataset. This section will examine the performance and accuracy in real-time with the chosen detection algorithms, which are Random Forest in FD and the small architecture in Yolo, using the final labelled data from the third cycle.

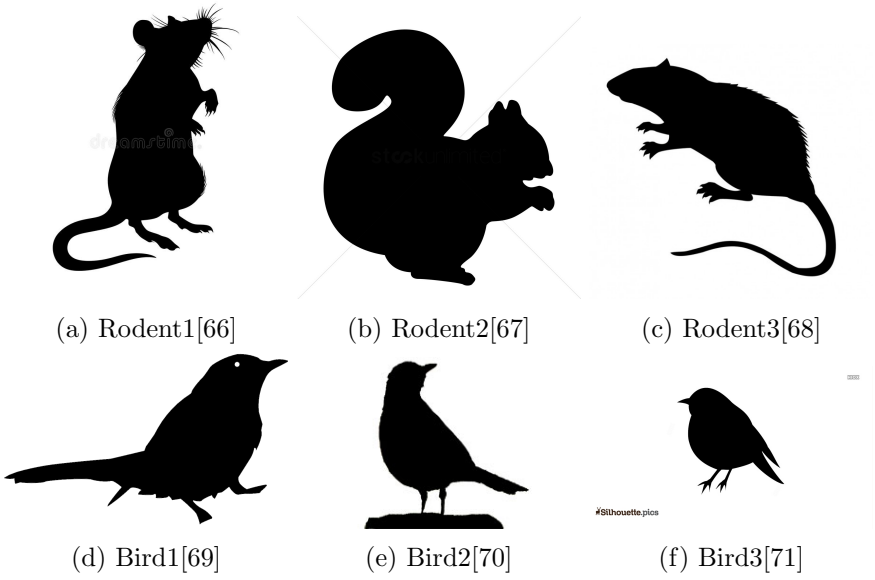
5.6.1 Image Sample

The method of the experiment is to calculate the accuracy and performance using the 6 sample videos about the silhouette image of rodents and birds, which are figures 5.16a, 5.16b, 5.16c, 5.16d, 5.16e, and 5.16f) The scale of the six samples is various because we presume that The target size affects the performance because of the Blob computation, which finds the location of the target and the most time-consuming work. The bird2 in figure 5.16e is the smallest sample, and the bird1 in figure 5.16d is the biggest one. Moreover, we set up some conditions for the video below.

1. The video is recorded in the best state for the detection algorithm in the FD, such as white background with the black object.
2. Captured video frame size is 800 width and 600 height.
3. The duration of the video is around 10 seconds, which is between 250 to 300 frames.
4. We try to imitate the practical motion by moving the printed target dynamically.

5.6.2 The Accuracy for The Different Window Size

The result of the speed and accuracy in our detection depends on the size of the window. The window is subsequently the transformed image applied to some filters and algorithms to obtain the target contour from the image. We assume that as the window size is large,



(a) Rodent1[66] (b) Rodent2[67] (c) Rodent3[68]

(d) Bird1[69] (e) Bird2[70] (f) Bird3[71]

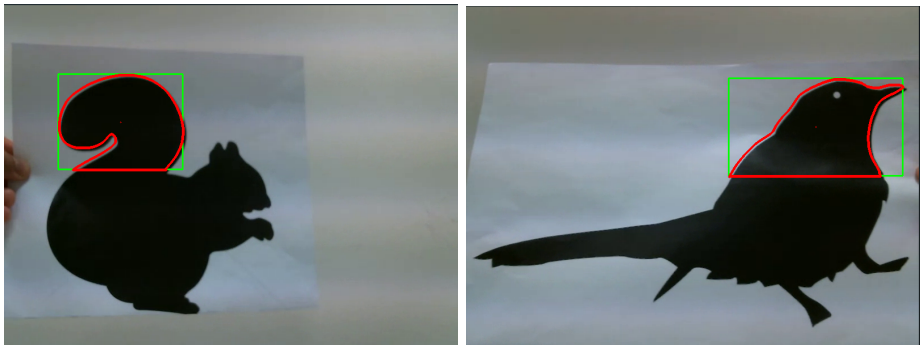
Figure 5.16: Sample Shapes

the accuracy might improve. However, the computational time also increases related to the growth of the window size. So expanding the window size may not be suitable for our purpose. One of the partial goals of our detection is to be faster than the speed of Yolo detection in the CPU with similar accuracy. Therefore, this section will investigate the variation of accuracy and performance related to window size in FD using real-time detection.

5.6.2.1 The Partial Detection in The Accuracy

Figure 5.18a illustrates the precision of different sizes of windows from the captured size, which is 800 width and 600 height, to the one-tenth size of the captured one using each three different silhouette samples of a bird and a rodent, based on the table (5.18). Overall, the accuracy of the rest of the samples fluctuates depending on frame size, whereas the precision of *Rodent2* declines when reduced the size of the window. The expectation of the accuracy in the captured size of the window is slightly better than its half size, but *Rodent2* shows the opposite result that precision increased approximately more than twice from 43% to 98% according to table 5.16. The reason that the accuracy is significantly decreased in the original size compared to its half size is the failure to capture the full profile of the target image. The detection program continuously captured the tail of the *Rodent2* (figure 5.17a). The same issue of *Rodent2* occurs to *Bird2* (figure 5.17b). However, the accuracy of its original window of the *Bird2* is approximately 7% better than the half one in table 5.16 because the profile of the bird's head has a unique feature than its full profile. We regard the captured partial profile as failing to detect the target properly because of a lack of profile information and detecting the wrong location of the

target. Therefore, the original captured size of the window is not the proper size in the Fourier Descriptors because it frequently detects the only partial contour of the target, and it is insufficient information to discriminate the target.



(a) The partial detection: Rodent2 (b) The partial detection: Bird1

Figure 5.17: The Fail Cases

	The Accuracy for Different Window Sizes (Original Size * x)					
X/Sample	**1**	**1/2**	**1/4**	**1/6**	**1/8**	**1/10**
Rodent1	70.84%	49.81%	67.5%	42.06%	43.17%	66.05%
Rodent2	43.41%	98.44%	98.4%	83.72%	15.89%	10.46%
Rodent3	91.9%	75.3%	76.9%	24.69%	33.19%	69.23%
Bird1	90.2%	83.67%	86.9%	86.12%	84.08%	85.30%
Bird2	91.91%	93.75%	93.7%	86.76%	65.44%	15.80%
Bird3	82.78%	88.52%	88.1%	86.88%	85.65%	84.01%

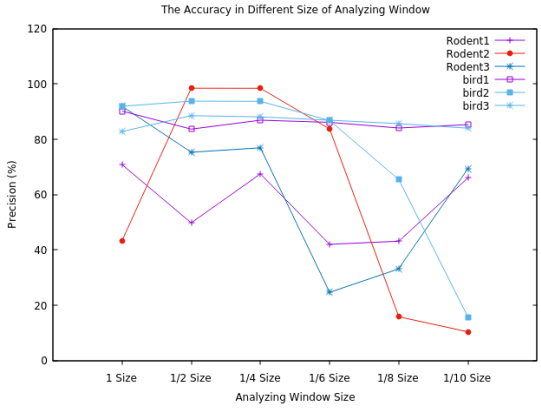
Table 5.16: The Accuracies in Different Window Sizes Using FD in Desktop

	The Frame Rate for Different Window Sizes (Original Size * x)					
X/Sample	**1**	**1/2**	**1/4**	**1/6**	**1/8**	**1/10**
Rodent1	1.39 fps	14.35 fps	19.87 fps	35.1 fps	41.19 fps	43.18 fps
Rodent2	2.19 fps	17.16 fps	21.79 fps	35.98 fps	39.54 fps	44.49 fps
Rodent3	2.03 fps	21.24 fps	25.70 fps	37.29 fps	42.01 fps	44.73 fps
Bird1	2.91 fps	11.99 fps	17.98 fps	31.81 fps	37.40 fps	43.09 fps
Bird2	5.89 fps	26.52 fps	28.55 fps	40.52 fps	43.07 fps	45.00 fps
Bird3	1.58 fps	23.72 fps	27.84 fps	38.24 fps	42.25 fps	43.93 fps

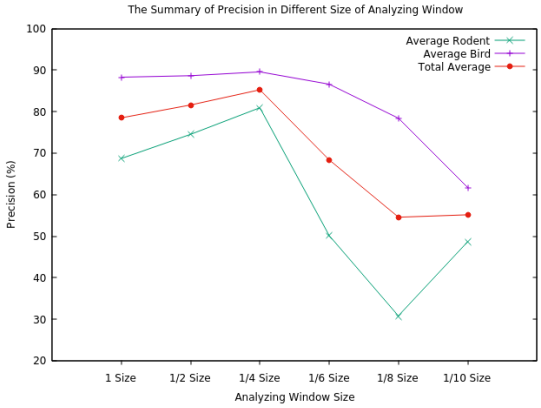
Table 5.17: The Frame Rate of Different Window Sizes Using FD in Desktop

	Different Window Sizes (Original Size * x)					
X/Average	**1**	**1/2**	**1/4**	**1/6**	**1/8**	**1/10**
The Average of Rodent Accuracy	68.71%	74.51%	80.93%	50.15%	30.75%	48.58%
The Average of Bird Accuracy	88.29%	88.64%	89.56%	86.58%	78.39%	61.70%
The Overall Accuracy	78.50%	81.58%	85.25%	68.37%	54.57%	55.14%
Average Frame Rate	2.66 fps	19.16 fps	23.62 fps	36.49 fps	40.91 fps	44.07 fps

Table 5.18: The Summary of Different Window Sizes Using FD in Desktop

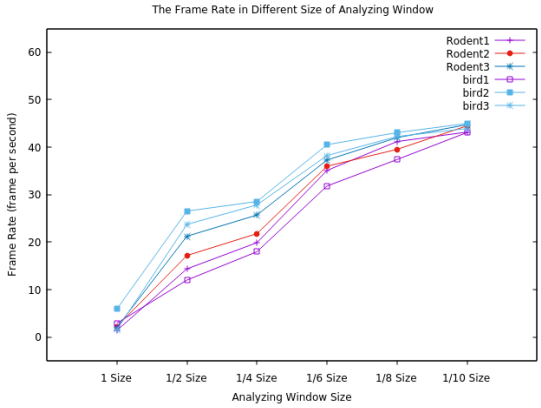


(a) The Collection of The Accuracy for Different Window Sizes Using FD

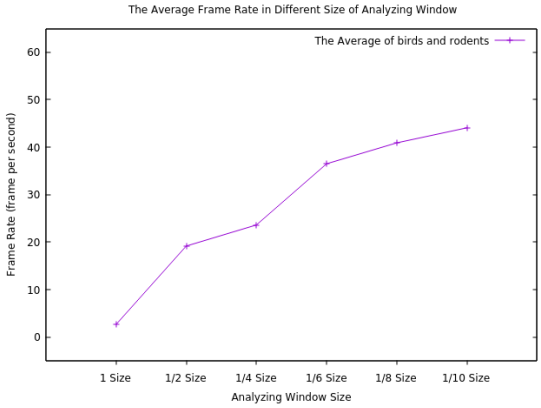


(b) The Summary of The Accuracy for Different Window Sizes Using FD

Figure 5.18: The Accuracy on The Desktop in FD



(a) The Collection of The Frame Rate for Different Window Sizes Using FD



(b) The Average of The Frame Rate for Different Window Sizes Using FD

Figure 5.19: The Frame Rate on The Desktop in FD

5.6.2.2 The Trend of Accuracy

Another interesting finding is shown in figure (5.18a). After the precision reached below 60% in all samples, the accuracy value fluctuates dramatically between size reduction. This fluctuation implies that the detection algorithm randomly chooses the value between 0 for the rodent and 1 for the bird using the shape of the target. In addition, if the class value frequently changes in a particular shape of the target during the detection, the algorithm does not have the ability to distinguish the shape of the target. Therefore, we can conclude that returning inconsistent value generally starts once the accuracy reaches below 60%, and the program loses the ability to recognize the target from this point.

5.6.3 The Frame Rate

Figure 5.19a illustrates the frame rate, which is in frames per second, in the different window sizes in all samples, and figure 5.19b is general frame rate in all classes based on figure 5.19a.

Overall, the frame rate has been increased when the size of the window is decreased in figure 5.19b. The half-size of the window shows the most significant growth rate between samples, and then this growth rate has steadily reduced from the half-size to the one-tenth size of the original one in figure 5.19b. In addition, in the "Image Sample" section, we mentioned the biggest scale of the sample is *Bird1*, and the smallest one is Bird2. Figure 5.19a generally shows that *Bird1* is the worst frame rate and Bird 2 is the best one between samples. From this trend, the size of the detected target substantially influences the frame rate. The influence of the target size has been gradually reduced when the size of the window shrinks because of reducing the gap of the frame rate between samples in the figure 5.19a. The best frame rate is the one-tenth size of the original one as 44.07 fps in table 5.18, but the quarter and one-sixth size are ideal sizes concerning its accuracy in figure 5.18b. Therefore, the frame rate is closely related to the analysing window size, which is the growth of the frame rate when the analysing size has reduced, and the detected target size is also relevant to the frame rate, but this influence decrease when the window size is reduced. Moreover, both the quarter and one-sixth size of the windows are ideal for different purposes of real-time detection. The quarter is a suitable window size if the user considers the accuracy more important, or the one-sixth size, which is the best one for the frame rate.

5.6.4 The Comparison of The Frame Rate between Desktop and Raspberry PI

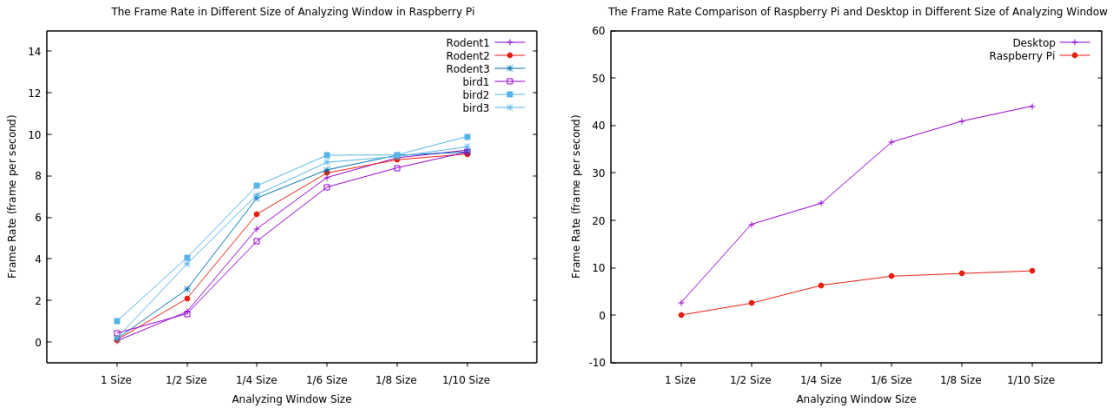
The previous section concludes that the quarter size is ideal if weighed on the accuracy and the one-sixth size is the one if considering that the frame rate is more important. We

can assume that the speed of the smaller window size is the faster speed than the bigger window in Raspberry Pi from the previous experiment, but it is hard to estimate the change of the frame rate between different window sizes in Raspberry Pi. The accuracy must be the same between Desktop and Raspberry Pi because both use the exact same code. Therefore, we measure the operation time in Raspberry Pi using the method and code of the previous section and compare the difference of the performance from Desktop to Raspberry Pi.

	The Frame Rate for Different Window Sizes (Original Size * x)					
X/Sample	**1**	**1/2**	**1/4**	**1/6**	**1/8**	**1/10**
Rodent1	0.05 fps	1.46 fps	5.46 fps	7.93 fps	8.87 fps	9.24 fps
Rodent2	0.10 fps	2.09 fps	6.16 fps	8.14 fps	8.78 fps	9.06 fps
Rodent3	0.15 fps	2.53 fps	6.94 fps	8.29 fps	8.99 fps	9.31 fps
Bird1	0.43 fps	1.36 fps	4.86 fps	7.46 fps	8.39 fps	9.15 fps
Bird2	1.00 fps	4.06 fps	7.53 fps	8.99 fps	9.01 fps	9.90 fps
Bird3	0.19 fps	3.75 fps	7.09 fps	8.66 fps	8.93 fps	9.40 fps

Table 5.19: The Comparison of The Frame Rate with Different Window Size in Raspberry Pi

	The Frame Rate for Different Window Sizes (Original Size * x)					
X/Machine	**1**	**1/2**	**1/4**	**1/6**	**1/8**	**1/10**
Desktop	2.66 fps	19.16 fps	23.62 fps	36.49 fps	40.91 fps	44.07 fps
Raspberry Pi	0.05 fps	2.54 fps	6.33 fps	8.24 fps	8.82 fps	9.34 fps



(a) The Frame Rate of All Samples in Raspberry Pi

(b) The Average Frame Rate Compared to Desktop

Figure 5.20: The Frame Rate on Raspberry Pi in FD

5.6.5 Summary

Figure 5.20b illustrates the comparison of the average frame rate between Desktop and Raspberry Pi. Generally, the frame rate in Raspberry Pi shows steady growth when the size of the window is reduced, whereas the desktop shows a significant increase in between the size of the window. The main comparison is between the quarter and one-sixth sizes because the quarter-size window shows better accuracy than the one-sixth window, whereas the one-sixth window results in a better frame rate than The quarter-size one. In comparing the growth frame rate in the desktop between the quarter and one-sixth sizes, the one-sixth size has improved 12.87 fps, whereas, in Raspberry Pi, only 1.91 fps is faster. The growth frame rate between the quarter and one-sixth sizes in Raspberry Pi is very low, but the quarter one is 16.88% more accurate than the one-sixth one (table 5.21). Therefore, we can conclude that the quarter size is a suitable window size for the Fourier Descriptors method.

5.7 Real-time Detection: The Comparison of The FD and YOLO

Previously, the section examined and found the most appropriate size of the window. We finally determined that the one-quarter size of the window and Random Forest in Fourier Descriptors is the best algorithm. This established technique will be compared to Yolo with the fastest architecture about the performance and accuracy. The experiment in this section will be carried out with the exact same target images and methods as the previous section. Moreover, we also experiment with the accuracy and frame rate changes in the best and random environment between FD and Yolo.

5.7.1 General Trend

Figure 5.21 illustrates the comparison of the average accuracy using the techniques of the lightest architecture of Yolo version five and Random Forest in Fourier Descriptors with each three different silhouette samples of birds and rodents. Generally, in figure 5.21, the bird category shows a similar accuracy as approximately 89% in each technique, whereas the accuracy of the rodent in FD is significantly higher than Yolo's one, almost four times higher. The common thing between Yolo and FD is that both can easily recognise birds rather than rodents.

	FD (Random Forest)		Yolo (Small)	
Samples	**Precision**	**Frame Rate**	**Precision**	**Frame Rate**
Rodent1	67.5%	19.9 fps	9.2%	6.4 fps
Rodent2	98.4%	21.8 fps	13.2%	6.4 fps
Rodent3	76.9%	25.7 fps	45.1%	6.3 fps
Bird1	86.9%	17.9 fps	67.6%	6.3 fps
Bird2	93.7%	28.6 fps	99.6%	6.6 fps
Bird3	88.1%	27.8 fps	100%	6.4 fps

Table 5.20: The Comparison of Performance and Frame Rate in FD and Yolo

	Average Precision	Average Rodent Precision	Average Bird Precision	Average Frame Rate
FD	85.25%	80.93%	89.56%	23.62 fps
Yolo	55.78%	22.5%	89.06%	6.40 fps

Table 5.21: The Summary of Performance and Frame Rate

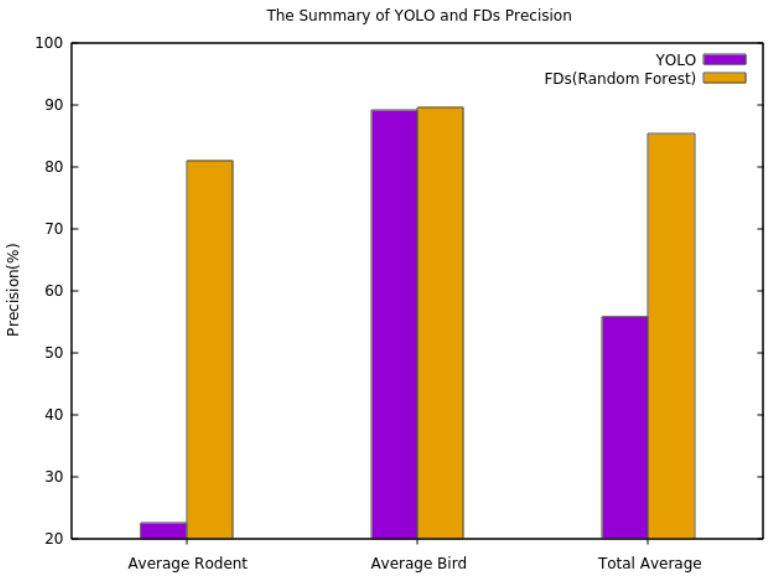


Figure 5.21: The Comparison of The Accuracy in FD and Yolo

5.7.2 Analysing The Sample of The Lowest Accuracy in Yolo

From table 5.21, which is the summary of the performance and frame rate, the gap between the bird and rodent about the average accuracy in Yolo is very wide; the gap is almost 67%. We select the representative sample as *Rodent1*, which has the worst accuracy, to find the reason for this issue.

There are three main reasons. The first reason is that the training dataset does not involve any similar shape to the sample of *Rodent1*, which is the first sample in table 5.20. The training dataset does not include similar species of *Rodent1*, even though the rodent's standing posture is unique. On the contrary, showing the high accuracy in the *Bird2* and *Bird3* samples, which measured the accuracies 99.6% and 100%, is particularly in the common shape of the bird and is included in a large amount of the training dataset. For instance, the first and second cycles organize the similar silhouette of *Bird2* mainly, and the target bird shape of the third cycle is also similar to *Bird3*. Therefore, training similar forms of the target influences the accuracy.

The second reason that Yolo detection frequently fails to detect the target in *Rodent1* compared to other samples. In table 5.22, 89 frames recognize none of the targets from the total 282 frames. That is, the program fails to detect the target in 32% of the total length of the video, excluding the time of appearing and disappearing the target. This failure is also related to the first reason; it is the lack of pre-trained information.

The last reason is a false detection. Yolo detects the *Rodent1* as the bird in 168 frames from the 282 frames in table 5.22. The accuracy of the *Rodent1* in FD is also relatively low as 67.5% in table 5.20, and the FD technique has a false detection in 91 frames. Therefore, the *Rodent1* might have an ambiguous shape for detecting in both Yolo and FD.

Rodent1 (282 frames)	**Rodent**	**Bird**	**Not-detected**
Correct Detect	6	0	19
Wrong Detect	0	168	89

Table 5.22: The detection result of the *Rodent1* in the best environment

To sum up, the average accuracy of the bird in the figure (5.21 shows almost the same as 89%. However, the average accuracy of rodents in FD as 80% is almost 68% higher than Yolo, which is about 22%. Between the three reasons, The first reason is the primary to show critically low accuracy in the rodent category in Yolo because the second and third reasons occur by the first reason. We can conclude that the Yolo technique substantially struggles more to detect the unfamiliar silhouette of a target than FD.

5.7.3 Frame Rate

About the frame rate, in table 5.20, Yolo shows a consistent frame rate between six samples, whereas the FD varies. Yolo is objective in any size of the target, while the

frame rate is closely related to the target size. For instance, the *Bird2* is the smallest target size comparing other samples, and it shows the fastest performance at 28.55 fps in table 5.20, it almost 9 fps faster than the slowest one, which is the most significant target size and frame rate is 19.87 in the *Bird2* of table 5.20. Furthermore, *Bird2*, the lowest frame rate in FD, is faster than any Yolo sample. Therefore, although FD shows the inconsistency of the frame rate, the average speed is approximately four times faster than the Yolo.

5.7.4 Summary

We compared the general accuracy and frame rate between FD and Yolo. The general accuracy in FD is approximately 30% higher than Yolo, which are 85.25% in FD and 55.78%. Showing much less accuracy in Yolo is that it has a weak ability to detect an unfamiliar shape of a target than FD. In addition, the FD offers a four times faster frame rate than Yolo, 23.62 fps in FD and 6.40 fps in Yolo. The characteristic of the frame rate in FD is inconsistent depending on the detected target size, whereas the Yolo is consistent.

The experiment in this section was conducted with the most optimised environment in FD. For a more precise and objective analysis of the comparison between FD and Yolo, we prepared the additional experiment in a different environment. The advanced examination will proceed in the next section.

5.8 The Comparison of Different Environment

We assume that both FD and Yolo might show a considerable variation of accuracy and detection speed in a different environment. In order to fulfil the experiment, we organize the additional recorded video to examine the difference in accuracy and detection speed between the two dissimilar backgrounds using the same sample images in each technique. The first subjective video is the one used in the previous experiment, which is the best case with the plain background environment. Another video is the random environment, which imitates nature habitat using some papers with various colors, which is the figure (5.22).

5.8.1 The Accuracy

Overall, the result in table 5.25 shows a similar to the previous experiment; the average detection rate in a rodent is less than a bird, and a rodent accuracy in Yolo is significantly less than a bird. Figure 5.25 illustrates the comparison of the precision in the different environments, called the best and random environment, using FD and YOLO. The random environment shows slightly less accuracy than the best one in both FD and Yolo, which are 3% in FD and 4% less in Yolo. In FD, the disturbance caused by the background

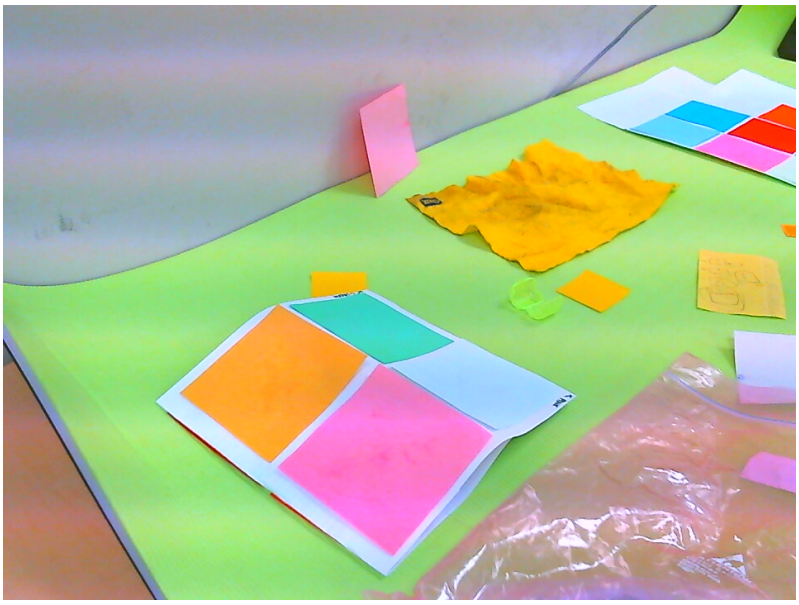


Figure 5.22: The Random Environment

noise results in false detection, seen in figures 5.24a and 5.24b. About Yolo, the rate of detecting the none of target increases in the random environment comparing with the best one. For instance, the false none-detected frame in the rodent1 of the random environment is approximately 72% in table 5.23, the none-detected frame when the object exists as 127 is divided by the total frame 176. The rate of **"no-detection"** when the target exists in the random background rises more than twice than the best one in figure 5.23.

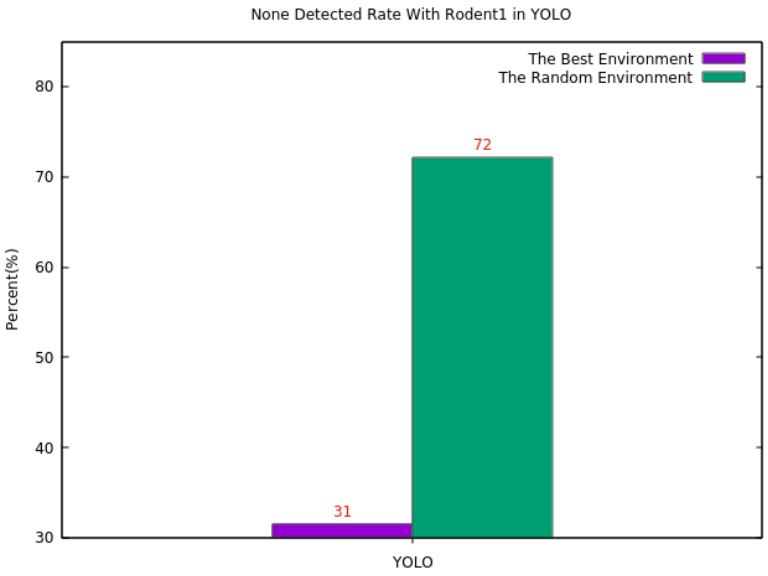


Figure 5.23: The **"Nothing"** Detected Rate between The Random and Best Background

Rodent1 (176 frames)	Rodent	Bird	Not-detected
Correct Detect	1	0	34
Wrong Detect	0	14	127

Table 5.23: The Detection Result of The Rodent1 In The Random Environment

(a) Fail Case 1 (b) Fail Case 2

Figure 5.24: Fail Cases in The Random Environment

	FD (Random Forest)		Yolo (Small)	
Samples	**Precision**	**Frame Rate**	**Precision**	**Frame Rate**
Rodent1	78.7%	18.7 fps	19.8%	5.8 fps
Rodent2	83.7%	18.4 fps	20.2%	5.9 fps
Rodent3	75.0%	19.5 fps	31.1%	5.6 fps
Bird1	88.6%	16.4 fps	53.9%	5.8 fps
Bird2	80.8%	23.9 fps	99.4%	5.9 fps
Bird3	86.6%	21.6 fps	85.6%	5.8 fps

Table 5.24: The Accuracy and Frame Rate Using FD and Yolo in The Random Environment

	Average Precision	Average Rodent Precision	Average Bird Precision	Average Frame Rate
FD	82.23%	79.1%	85.3%	19.75 fps
Yolo	51.7%	23.7%	79.6%	5.8 fps

Table 5.25: The Summary of Performance and Frame Rate in The Random Environment

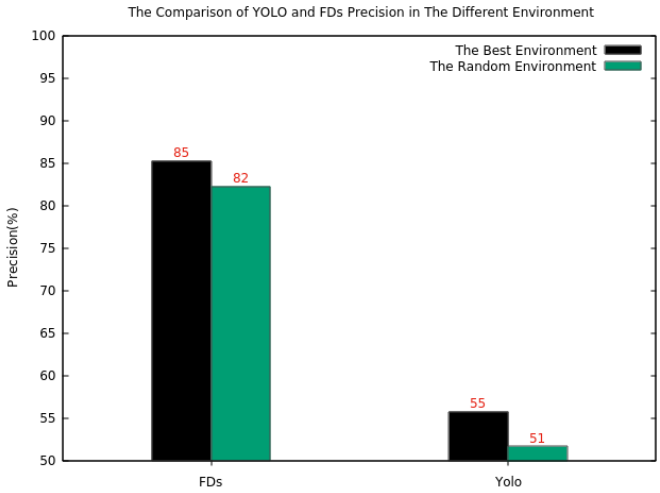


Figure 5.25: The Comparison of The Accuracy Using FD And Yolo In The Different Environment

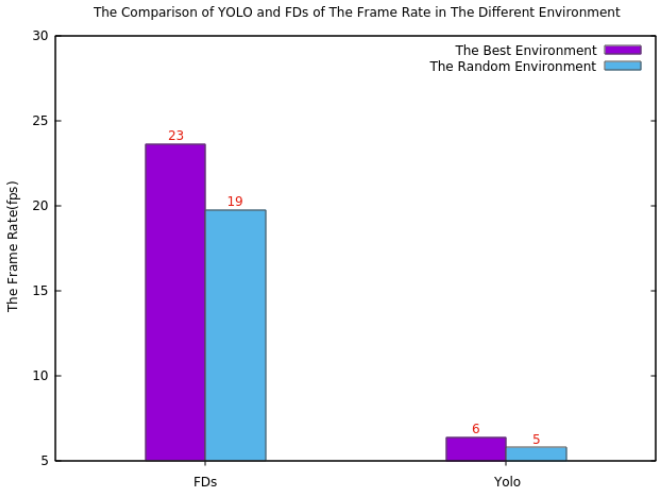


Figure 5.26: The Comparison of The Speed Using FD And Yolo In The Different Environment

5.8.2 The Frame Rate

Moreover, figure 5.26, which is the comparison of the frame rate in a different environment using FD and Yolo, also shows a similar trend of the accuracy comparison, in which the random environment is 4 fps less than the best one in FD and 1 fps less in Yolo. The frame rate in Yolo rarely affects the frame rate in different environments, whereas the background influences the FD.

5.8.3 Summary

We analysed the influence of accuracy and frame rate in different environments in FD and Yolo. The accuracy in both FD and Yolo decreased approximately 4% in the random environment because the FD technique has the disturbance of obtaining a clear outline of a target by the environment, and Yolo increases the failure to detect an object. Whereas the FD technique generally has a 4 fps decrease, the Yolo has a minor reduction in speed (1 fps reduction in the random environment).

5.9 Additional Experiment: The Frame Rate in Raspberry Pi Using Different Architectures

In this section, we examine the influence of frame rate in Raspberry Pi using different architectures. Figure 5.27 and shows the speed rate with four different architectures in Raspberry Pi using CPU. Generally, the more complex architecture is, the slower speed rate it achieves; the next complex architecture is approximately two times slower than the previous one. The fastest one, which is small and 0.310 fps, almost eight times faster than the slowest one, which is the extra-large and 0.046. In other words, the fastest one spends around 3 seconds dealing with the one frame, whereas the slowest one requires 24 seconds processing the one frame image in Raspberry Pi. Even the small architecture does not have an appropriate frame rate for real-time detection because it shows less than one frame per second. Therefore, any architectures in Yolo version five using CPU are not feasible to generate real-time detection in the Raspberry Pi machine.

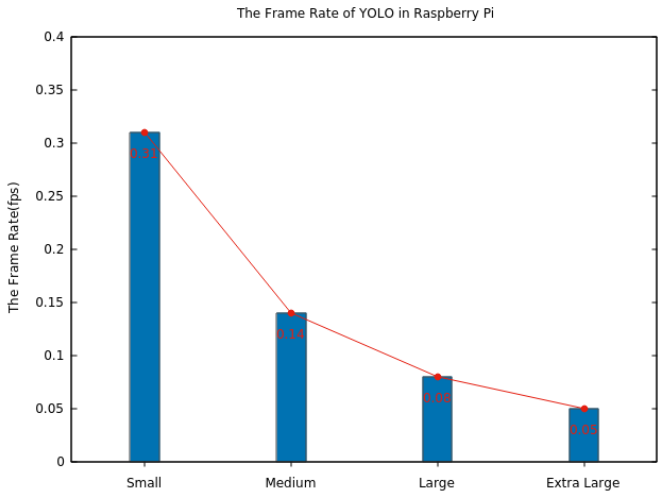


Figure 5.27: The Final Frame Rate Comparison in Raspberry Pi

	The Frame Rate in Different Size of Analyzing Window (Original Size * x)			
Architect / Sample	**Small**	**Medium**	**Large**	**Extra Large**
Rodent1	0.310 fps	0.147 fps	0.077 fps	0.046 fps

Table 5.26: The Final Frame Rate Comparison in Raspberry Pi

5.10 Chapter Conclusions

1. In the algorithm of FD, Random Forest shows the highest mean accuracy between Neural Network, AdaBoost, and Support Vector Machine as 80% with an adequate training time of 45 seconds.

2. 20 input CE values are the ideal number in the Random Forest of FD because it showed the highest mean accuracy.

3. Adding the unbalanced number of the labelled data might decreases F1-score in a specific class.

4. Our trained weight shows a better precision rate for the bird in both Yolo and FD.

5. Yolo shows an extremely fast detection speed in GPU as 90.90 fps than CPU as 3.7 fps in the "small" architecture of Yolo from the desktop.

6. The "small" is the appropriate architecture in Yolo by showing the fastest training as 1.416 hours, the accuracy as 97%, and the frame rate of CPU detection as 3.7 fps on a desktop.

7. The accuracy and frame rate of both FD and Yolo are reduced in the random environment, comparing with the best environment. The accuracy of both techniques decreased 4%, and the frame rate declined 1 fps in Yolo and 4 fps in FD.

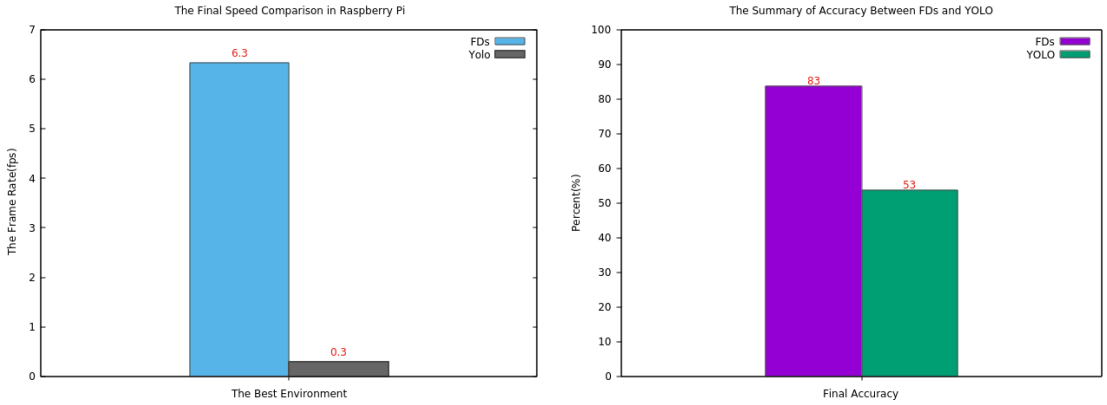
8. About real-time detection of FD, the quarter size of analyzing window is the ideal size both accuracy as 83% and frame rate as 6.3% in Raspberry Pi.

9. About real-time detection of Yolo, the "small" architecture shows 53 % accuracy and 0.3 fps in Raspberry Pi.

10. The heaviest architecture, which is extra-large Yolo, shows the frame rate of 0.046 in Raspberry Pi; it spends around 20 seconds dealing with one frame.

We generated some experiments to find the ideal algorithm in FD and architecture in Yolo using 5666 labelled data and to compare the accuracy and frame rate in real-time about FD and Yolo using different environments. The Random Forest is the ideal algorithm, and the quarter size of the window is the best one considering both accuracy and performance

in our algorithm, which used Fourier Descriptors. Moreover, using our labelled data, the small architecture is suitable for accuracy and performance in Yolo version 5. Figures 5.28a and 5.28b describe the summary of our experiment; it shows the average frame rate and accuracy in Raspberry Pi between FD and Yolo. In conclusion, the Fourier Descriptors is 21 times faster and 30% more accurate than Yolo in the Raspberry machine using the training weight trained by our labelled data.



(a) The Final Frame Rate Comparison in Raspberry Pi

(b) The Final Accuracy Comparison in Raspberry Pi

Figure 5.28: Final Results

Chapter 6

Conclusions

6.1 Conclusion

In this thesis, we have proposed implementing a new detection algorithm in Raspberry Pi using Fourier Descriptors to recognize whether an object is a rodent or a bird in real-time within a fixed camera position in order to protect bird habitat from rodents. We also created a semi-automated labelling system using a collection of sequential images and video files. This system promises a faster labelling process than a manual one and it succeeded in obtaining 2172 rodents and 3494 birds labels. In Fourier Descriptors (FD), when increasing the number of CEs, the accuracy shows a growing trend. The Random Forest in our detection algorithm using Fourier Descriptors is the most accurate classifier compared to Neural Network, ADA Boost, Support Vector Machine. The average accuracy rate is 80%, and the f1-score for rodents is 72%. The average accuracy rate in You Only Look Once (YOLO) version 5 is 97%, and there is not much influence of accuracy in different architectures (from small to extra large) using our labelled data. The detection frame rate of the CPU in the small architecture of YOLO is 25 times slower than its GPU and 60 times slower in the extra-large architecture. Our detection algorithm in Raspberry pi 3 with a different dataset shows a frame rate of 6.33 fps, which is 21 times faster than YOLO version 5 with small architecture and an accuracy of 83%, which is 30% more accurate in real-time detection than YOLO.

6.2 Future Work

The main directions for future work are to install Raspberry Pi in the natural habitat of New Zealand to test real-time detection and collect data on rodents and birds. Also, our algorithm struggles with detection in particular backgrounds, so our program requires practical analysis in the actual field in order to improve the algorithm based on this research. Moreover, additional tools and methods for the communication between Rasp-

berry Pi and the main machine are needed (e.g. LoRaWAN), so an efficient device can be used to help protecting native birds in their natural habitat.

Bibliography

[1] S. Hampshire, "Invasive species monitoring and predator control via innovative ai technology." https://cacophony.org.nz/sites/default/files/SaphyHampshireCacophonyProject2021-Edited.pdf, 2021. Online; accessed 18 June 2021.

[2] T. D. Hunt, "The cacophony project: Moores law for new zealands birds." http://researcharchive.wintec.ac.nz/6842/4/The%20Cacophony%20Project%20Tim%20Hunt%20v3.pdf, 2019. Online; accessed 18 June 2021.

[3] C. Bell, *Beginning sensor networks with Arduino and Raspberry Pi*. Apress, 2014.

[4] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network", in *2017 International Conference on Engineering and Technology (ICET)*, pp. 1–6, Ieee, 2017.

[5] Pawan Saxena, "Selective Search for Object Detection — R-CNN." https://www.geeksforgeeks.org/selective-search-for-object-detection-r-cnn/, 2020. Online; accessed 9 July 2021.

[6] P. Viola and M. Jones, "Robust real-time face detection", *International Journal of Computer Vision*, vol. 57, pp. 137–154, May 2004.

[7] A. L. C. Barczak, *Computer Vision - 159731 Study Guide*. Massey University, 9 ed., 2019.

[8] A. orovi, V. Ili, S. uri, M. Marijan, and B. Pavkovi, "The real-time detection of traffic participants using yolo algorithm", in *2018 26th Telecommunications Forum (TELFOR)*, pp. 1–4, IEEE, 2018.

[9] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[10] S. H. Shaikh, K. Saeed, and N. Chaki, "Moving object detection using background subtraction", in *Moving object detection using background subtraction*, pp. 15–23, Springer, 2014.

[11] R. Raju and A. Bargiela, "New image processing pipelines for membrane detection", *Journal of the Institute of Industrial Applications Engineers Vol*, vol. 3, no. 1, pp. 15–23, 2015.

[12] M. P. B. Slomp, "Real-time photographic local tone reproduction using summed-area tables." `https://www.lume.ufrgs.br/handle/10183/34766`, 2008. Online; accessed 18 June 2021.

[13] B. Mahesh, "Machine learning algorithms-a review", *International Journal of Science and Research (IJSR).[Internet]*, vol. 9, pp. 381–386, 2020.

[14] S. Misra, H. Li, and J. He, "Noninvasive fracture characterization based on the classification of sonic wave travel times", in *Machine Learning for Subsurface Characterization*, pp. 243–287, Gulf Professional Publishing, 2020.

[15] Vihar Kurama, "Fully grown decision tree (left) vs three decision stumps (right)." `https://blog.paperspace.com/adaboost-optimizer/`, 2020. Online; accessed 9 July 2021.

[16] P. L. Fernández-Cabán, F. J. Masters, and B. M. Phillips, "Predicting roof pressures on a low-rise structure from freestream turbulence using artificial neural networks", *Frontiers in Built Environment*, vol. 4, p. 68, 2018.

[17] C. T. Eason, E. C. Murphy, G. R. Wright, and E. B. Spurr, "Assessment of risks of brodifacoum to non-target birds and mammals in new zealand", *Ecotoxicology*, vol. 11, no. 1, pp. 35–48, 2002.

[18] B. D. Bell, "The big south cape islands rat irruption", *The ecology and control of rodents in New Zealand nature reserves*, vol. 4, pp. 33–45, 1978.

[19] R. AR, S. DR, C. R, and W. J, "Camera trapping: A contemporary approach to monitoring invasive rodents in high conservation priority ecosystems", *PLoS ONE*, vol. 9, no. 3, 2014.

[20] R. Ross, L. Parsons, B. Thai, and M. Hall, R.and Kaushik, "An iot smart rodent bait station system utilizing computer vision", *Sensors*, vol. 20, no. 17, 2020.

[21] J. Krynitsky, A. A. Legaria, J. J. Pai, M. Garmendia-Cedillos, G. Salem, T. Pohida, and A. V. Kravitz, "Rodent arena tracker (rat): a machine vision rodent tracking camera and closed loop control system", *Eneuro*, vol. 7, no. 3, 2020.

[22] A. Gerós, A. Magalhães, and P. Aguiar, "Improved 3d tracking and automated classification of rodents behavioral activity using depth-sensing cameras", *Behavior research methods*, pp. 1–12, 2020.

[23] Rosebrock, "Yolo and tiny-yolo object detection on the raspberry pi and movidius ncs."

[24] M. Schmidt, *Raspberry Pi: a quick-start guide*. Pragmatic Bookshelf, 2014.

[25] M. Richardson and S. Wallace, *Getting started with raspberry PI. "* O'Reilly Media, Inc.", 2012.

[26] A. N. Ansari, M. Sedky, N. Sharma, and A. Tyagi, "An internet of things approach for motion detection using raspberry pi", in *Proceedings of 2015 International Conference on Intelligent Computing and Internet of Things*, pp. 131–134, IEEE, 2015.

[27] S. E. Princy and K. G. J. Nigel, "Implementation of cloud server for real time data storage using raspberry pi", in *2015 Online International Conference on Green Engineering and Technologies (IC-GET)*, pp. 1–4, IEEE, 2015.

[28] C. Rapier and B. Bennett, "High speed bulk data transfer using the ssh protocol", in *Proceedings of the 15th ACM Mardi Gras conference: From lightweight mash-ups to lambda grids: Understanding the spectrum of distributed computing requirements, applications, tools, infrastructures, interoperability, and the incremental adoption of key capabilities*, pp. 1–7, 2008.

[29] N. Ismail, N. Abd Rashid, N. Zakaria, Z. I. Khan, and A. Mahmud, "Low cost extended wireless network using raspberry pi 3b+", in *2020 IEEE Symposium on Industrial Electronics & Applications (ISIEA)*, pp. 1–4, IEEE, 2020.

[30] R. Huang, J. Pedoeem, and C. Chen, "Yolo-lite: a real-time object detection algorithm optimized for non-gpu computers", in *2018 IEEE International Conference on Big Data (Big Data)*, pp. 2503–2510, IEEE, 2018.

[31] P. Sismananda, M. Abdurohman, and A. G. Putrada, "Performance comparison of yolo-lite and yolov3 using raspberry pi and motioneyeos", in *2020 8th International Conference on Information and Communication Technology (ICoICT)*, pp. 1–7, IEEE, 2020.

[32] S. Agatonovic-Kustrin and R. Beresford, "Basic concepts of artificial neural network (ann) modeling and its application in pharmaceutical research", *Journal of pharmaceutical and biomedical analysis*, vol. 22, no. 5, pp. 717–727, 2000.

[33] W. Fang, L. Wang, and P. Ren, "Tinier-yolo: A real-time object detection method for constrained environments", *IEEE Access*, vol. 8, pp. 1935–1944, 2019.

[34] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net", *arXiv preprint arXiv:1412.6806*, 2014.

[35] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.

[36] R. Girshick, "Fast r-cnn", in *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448, 2015.

[37] M. Everingham, S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective", *International journal of computer vision*, vol. 111, no. 1, pp. 98–136, 2015.

[38] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[39] M. Lin, Q. Chen, and S. Yan, "Network in network", *arXiv preprint arXiv:1312.4400*, 2013.

[40] R. Cosgriff, "Identification of shape, rep. 820-11, astia ad 254792, ohio state univ", *Research Foundation, Columbus, Ohio USA*, 1960.

[41] M. Nixon and A. Aguado, *Feature extraction and image processing for computer vision*. Academic press, 2019.

[42] H. Schulz-Mirbach, "Invariant features for gray scale images", in *Mustererkennung 1995*, pp. 1–14, Springer, 1995.

[43] A. Kaehler and G. Bradski, *Learning OpenCV 3: computer vision in C++ with the OpenCV library.* " O'Reilly Media, Inc.", 2016.

[44] J. W. Tukey *et al.*, *Exploratory data analysis*, vol. 2. Reading, Mass., 1977.

[45] P. M. Narendra, "A separable median filter for image noise smoothing", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 1, pp. 20–29, 1981.

[46] M. Kuwahara, K. Hachimura, S. Eiho, and M. Kinoshita, "Processing of riangiocardiographic images", in *Digital processing of biomedical images*, pp. 187–202, Springer, 1976.

[47] J. E. Kyprianidis, "Image and video abstraction by multi-scale anisotropic kuwahara filtering", in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering*, pp. 55–64, 2011.

[48] F. C. Crow, "Summed-area tables for texture mapping", *Computer Graphics*, vol. 18 (3), pp. 207–212, July 1984.

[49] T. Kumar and K. Verma, "A theory based on conversion of rgb image to gray image", *International Journal of Computer Applications*, vol. 7, no. 2, pp. 7–10, 2010.

[50] T. Athanasiadis, P. Mylonas, Y. Avrithis, and S. Kollias, "Semantic image segmentation and object labeling", *IEEE transactions on circuits and systems for video technology*, vol. 17, no. 3, pp. 298–312, 2007.

[51] T. M. Mitchell *et al.*, *Machine learning*. McGraw-hill New York, 1997.

[52] J. R. Koza, F. H. Bennett, D. Andre, and M. A. Keane, "Automated design of both the topology and sizing of analog electrical circuits using genetic programming", in *Artificial Intelligence in Design96*, pp. 151–170, Springer, 1996.

[53] L. Breiman, "Random forests", *UC Berkeley TR567*, 1999.

[54] T. Shi and S. Horvath, "Unsupervised learning with random forest predictors", *Journal of Computational and Graphical Statistics*, vol. 15, no. 1, pp. 118–138, 2006.

[55] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and regression trees*. Routledge, 2017.

[56] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting", *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[57] V. Vapnik, *The nature of statistical learning theory*. Springer science & business media, 2013.

[58] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities", *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[59] S. V. Stehman, "Selecting and interpreting measures of thematic classification accuracy", *Remote Sensing of Environment*, vol. 62, no. 1, pp. 77–89, 1997.

[60] H. Li, L. Deng, C. Yang, J. Liu, and Z. Gu, "Enhanced yolo v3 tiny network for real-time ship detection from visual image", *IEEE Access*, vol. 9, pp. 16692–16706, 2021.

[61] G. Seni and J. F. Elder, "Ensemble methods in data mining: improving accuracy through combining predictions", *Synthesis lectures on data mining and knowledge discovery*, vol. 2, no. 1, pp. 1–126, 2010.

[62] A. dos Santos Ferreira, D. M. Freitas, G. G. da Silva, H. Pistori, and M. T. Folhes, "Unsupervised deep learning and semi-automatic data labeling in weed discrimination", *Computers and Electronics in Agriculture*, vol. 165, p. 104963, 2019.

[63] M. A. Tabak, M. S. Norouzzadeh, D. W. Wolfson, S. J. Sweeney, K. C. VerCauteren, N. P. Snow, J. M. Halseth, P. A. Di Salvo, J. S. Lewis, M. D. White, *et al.*, "Machine learning to classify animal species in camera trap images: Applications in ecology", *Methods in Ecology and Evolution*, vol. 10, no. 4, pp. 585–590, 2019.

[64] seahue, "Birds eating on snowy ground 2/16/2015." `https://www.youtube.com/watch?v=ksUPcqqMri0&t=804s`, 2015. Online; accessed 29 June 2020.

[65] H. Nature, "Snow Squirrels - 10 Hours - January 28, 2021." `https://www.youtube.com/watch?v=0tgQSBsIGzs&t=11474s`, 2021. Online; accessed 29 June 2020.

[66] K. Golovina, "Rat, isolated monochrome image, silhouette." `https://www.dreamstime.com/rat-isolated-monochrome-image-silhouette-symbol-year-according-to-chinese-calendar-image162685702`, 2020. Online; accessed 18 June 2020.

[67] F. I. Tank, "Silhouette Of Squirrel." `http://getdrawings.com/squirrel-vector#squirrel-vector-9.jpg`, 2019. Online; accessed 18 June 2020.

[68] Nerodear, "Rat Silhouette dxf File." `https://3axis.co/rat-silhouette-dxf-file/lopnk67w/`, 2019. Online; accessed 18 June 2020.

[69] James, "Bird Silhouette dxf File." `https://3axis.co/bird-silhouette-dxf-file/qomze873/`, 2019. Online; accessed 18 June 2020.

[70] K. Designs, "Cartoon Robin Bird Silhouette." `https://www.silhouette.pics/146401/cartoon-robin-bird-silhouette.php`, 2018. Online; accessed 18 June 2020.

[71] Heyyou, "Bird Id Skills Size Amp Shape All About Birds." `http://getdrawings.com/get-silhouette#tree-shapes-silhouette-2.jpg`, 2015. Online; accessed 18 June 2020.

Appendix A

Program and Videos

1.1 Program and Instruction Videos

The git repository in this appendix is used to produce training data and is created by us. The requirements for compiling this code are **OpenCV 4.3.0 in C++** and Ubuntu 18.04. In addition, the two instruction videos in this appendix are prepared in order to show how the program works without compiling.

Our program git:

`https://github.com/ShimQ88/master_project`

Transformed YOLOv5 git:

`https://github.com/ShimQ88/master_project_yolo`

A Semi-automated Labeling System:

`https://youtu.be/08csQzL1UZQ`

Training Labeled Data and Real-Time Detection in FDs:

`https://youtu.be/LMtrj1llQX4`

1.2 The Result Videos in Real-Time Detection

The below appendix presents the sample videos applied in real-time detection of FDs and Yolo techniques. The videos are analyzing resources in chapter 5, and there is an additional comment about calculated accuracy and the frame rate. Total 24 video links are included in this appendix (6 target images using FDs and Yolo in two different environments).

1. **The Best Environment**
   (a) Fourier Descriptors
      i. Rodent1: https://www.youtube.com/watch?v=AGmrn_u8o8w
      ii. Rodent2: https://www.youtube.com/watch?v=DCL40NNW-6Y
      iii. Rodent3: https://www.youtube.com/watch?v=cNe6ZmmwKg0
      iv. Bird1: https://www.youtube.com/watch?v=Af-OLRdbzys
      v. Bird2: https://www.youtube.com/watch?v=XFfisVuKw90
      vi. Bird3: https://www.youtube.com/watch?v=dDiboXMwdMM
   (b) YOLO
      i. Rodent1: https://www.youtube.com/watch?v=GzatDKyPH7s
      ii. Rodent2: https://www.youtube.com/watch?v=EJlKJetTjCo
      iii. Rodent3: https://www.youtube.com/watch?v=DtduquYg05w
      iv. Bird1: https://www.youtube.com/watch?v=ByOlenQtfrU
      v. Bird2: https://www.youtube.com/watch?v=QT4DaLYP3FE
      vi. Bird3: https://www.youtube.com/watch?v=ub7cEpng4Xk
2. **The Random Environment**
   (a) Fourier Descriptors
      i. Rodent1: https://www.youtube.com/watch?v=e5950EixP0I
      ii. Rodent2: https://www.youtube.com/watch?v=lKFwTa9ouws
      iii. Rodent3: https://www.youtube.com/watch?v=ThG6wgI6_QM
      iv. Bird1: https://www.youtube.com/watch?v=vd-v5Fvy164
      v. Bird2: https://www.youtube.com/watch?v=ojJsboPxfUk
      vi. Bird3: https://www.youtube.com/watch?v=fpAMOrPa5jY
   (b) YOLO
      i. Rodent1: https://www.youtube.com/watch?v=nVrjrTaYYoI
      ii. Rodent2: https://www.youtube.com/watch?v=Mw3uhIoJTj4
      iii. Rodent3: https://www.youtube.com/watch?v=m_DcC_tTFAk
      iv. Bird1: https://www.youtube.com/watch?v=difctqPKVGw
      v. Bird2: https://www.youtube.com/watch?v=QZIzwL_rxP0
      vi. Bird3: https://www.youtube.com/watch?v=RxTytUIApkk