DEPARTMENT OF COMPUTER SCIENCE

Deep Learning for Penguin Detection and Predation Behavior in Underwater Videos

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Bachelor of Science in the Faculty of Engineering.

Thursday 4th May, 2023

Abstract

This project delivered a highly reliable penguin detector for underwater video and made the first steps towards intricate behaviour detection.

The African penguin is an endangered specie, and ecologists are trying to protect them by collecting their underwater videos to learn about their habits. However, due to the huge amounts of video data, observing them manually takes a lot of time and effort. This project aims at using deep learning techniques to detect penguins and their feeding behaviour in the underwater video, which helps ecologists to process the video data.

This project implements a penguin detector based on YOLOv5 for the identification of African penguins in underwater video. On the test set of this project, it achieves an accuracy of 99% for penguin detection. In addition, a dual-stream network was built to detect the feeding behaviour of penguins in this project. The YOLOv5-based penguin detector and the ResNet-18-based fish detector are used as spatial streams to supply the spatial information of the video in the dual-stream network. The dense optical flow is computed as the temporal flow in the dual network to capture the motion information in the video. The final entire model achieved 74% accuracy in detecting penguin predation events on the test set of this project.

Overall, this project combines deep learning techniques with computer vision techniques to provide penguin detection models and predation event detection models for ecologists working with underwater videos of African penguins. In future research, the performance of the model can be further improved by adding more data and selecting more diverse features during the model training process.

The following is the summary of achievements in this project:

- I have spent 100 hours to take the image from the provided underwater video and label the penguin and fish to create the object detection dataset.
- I have trained the YOLOv5 object detector on the labelled dataset as the penguin detector. It achieved 99% accuracy in penguin detection and 70% accuracy in fish detection. In addition, I wrote about 200 lines of code to implement the penguin detector for video detection.
- I wrote a total of 600 lines of Python code with PyTorch deep learning framework to build the ResNet-18 model as the fish detector. It achieved an average detection accuracy of 91% on the images of the test set.
- I spent 50 hours to crop the original video and label the time of the predation event.
- I wrote a total of 500 lines of Python code to build the LSTM model as a behaviour detector. It achieved an average accuracy of 75%.

Contents

List of Figures

List of Tables

Code Listings

Ethics Statement

This project did not require ethical review, as determined by my supervisor, Dr Tilo Burghardt.

Supporting Technologies

Throughout the project, there are a number of third-party software and facilities that were used to implement this project. They are listed below:

- I used Roboflow [44] to create the object detection dataset.
- I used PyTorch [42] to build the deep learning model.
- I used Comet [6] to implement real-time monitoring of the metrics of the model during training.
- I used OpenCV [39] to implement the operations on images and the computation of optical flow.
- I used Git [13] to manage my project code.
- I used the supercomputer Blue Crystal Phase 4 at the University of Bristol [4] to train the model.

Notation and Acronyms

ANN	:	Artificial Intelligence Neural Networks
CNN	:	Convolutional neural networks
CV	:	Computer Vision
DL	:	Deep Learning
I3D	:	Inflated 3D Convolutional Neural Networks
LSTM	:	Long and short-term memory neural networks
ML	:	Machine Learning
RNN	:	Recurrent Neural Networks

Chapter 1

Introduction

As a result of the accelerated development of technology, artificial intelligence technology has become widespread in many fields. In particular, the advent of deep learning technology has enabled computers to autonomously extract features and perform tasks such as classification, recognition and prediction by learning from data, resulting in significant advancements in fields such as image processing and behavioural detection. Consequently, deep learning is also widely employed in wildlife conservation and ecosystem management. This thesis will use deep learning and computer vision techniques to detect penguin and their predation events in underwater video. In this chapter, I will describe the motivation of the project and some relevant research and objectives that will be achieved at the end of this project.

1.1 Motivation

The African Penguin, also known as the Spheniscus Demersus, is endemic to southern Africa and lives mainly along the coastlines of southwest Africa [8]. Sea fish, such as sardines and anchovies, are their main source of food. Unfortunately, this species was first listed as Endangered in 2010 [7]. According to a study by Richard et al [50], between 1989 and 2019, the population of penguins in Africa declined from approximately 51,500 pairs to 17,700, a total decrease of approximately 65%. In addition, their study [50] also points out that its annual population change rate has remained at -4% since 1979 and has reached -5.6% in the last 20 years. Therefore, the protection and management of the ecosystem of the African penguin are crucial to the survival of the species.

In 2019, Crawford et al. stated that the main cause of the population decline in African penguins comes from the scarcity of food resources in their habitat [9]. In order to better protect the African penguin, which has become an endangered species, intensive research into the underwater behaviour of African penguins can provide information and references for their conservation and management. However, underwater foraging behaviour is difficult to observe and study directly by humans, because African penguins mainly conduct their foraging activities in the ocean, which demand a significant investment of time and resources for human observation and analysis in underwater environments, and present inherent safety hazards.

In recent years, with the rapidly development of camera technology, more and more researchers have strated using underwater cameras to automatically collect data to monitor the behaviour of marine animals. This approach has reduced the workload on researchers and increased the accuracy and reliability of the data [35]. However, the huge amount of video data collected requires a lot of time and effort for manual analysis and classification. Additionally, when multiple researchers are analysing the data, different standard for judging predation events in the video may exist, which might lead to the result being unreliable. Thus, finding an efficient and accurate method to study the underwater behaviour of African penguins has become important.

1.2 Dataset

Computer scientist Dr Tilo Burghardt from the University of Bristol and ecologist Dr Richard Shelley from the University of Exeter provided the video dataset of penguin dives in this study. The video dataset was captured by using an underwater camera. Specifically, an animal-borne video recorder (AVR) was attached to the backs of the penguins so that footage of their actions underwater could be recorded [35].

Figure 1.1 shows the specific equipment used and its installation on the penguin. The majority of the footage I used was captured in 2017, with a total of 63 videos being used.

Figure 1.1: **Data collection equipment**: The picture of AVR on the back of the penguin from [35]

1.3 Related Work

As stated in section 1.1, because of the large volume of data in the penguin underwater video and the human uncertainty of observers, finding an effective and accurate method to assist researchers in analysing this data has become an important part of the study of underwater behaviour for African penguins.

The emergence of deep learning techniques has provided new ideas and methods for camera data processing and analysis. As computer vision and deep learning techniques continue to develop, the use of these techniques in wildlife conservation research is gradually increasing. A study in 2022 by Tuia et al. incorporated machine learning and computer vision into the workflow of animal conservation research and combined them with knowledge of animal conservation, leading to the conclusion that machine learning and computer vision techniques have great potential for the analysis of large-scale ecological data [53].

In 2018, Norouzzadeh et al. applied machine learning in a camera-trap images dataset [37]. They used a deep neural network model that enables the automatic identification, counting and description of wild animals in camera-captured images. They experimented with a total of four tasks and achieved the best accuracy of 96.8% in detecting the existence of animals in images, 94.9% in identifying animal species, 62.8% in counting the number of animals and 76.2% in animal behaviour recognition. They also compared the accuracy rate for the above tasks performed by humans and concluded that deep learning models and humans can process the data with the same level of accuracy, thus proving deep learning techniques can be useful for ecologists in image data processing.

Particularly, in the study of marine biology, machine learning and computer vision are also applied in many situations. In 2020, Lopez-Vazquez et al. applied machine learning techniques to the detection and classification of underwater animals [32]. They compared four classical machine learning algorithms (SVM, K-NN, Decision Tree and Random Forest) and two neural network models (CNN and DNN). The results showed that from the classical machine learning algorithms, SVM had the highest accuracy rate with 73.9%. While for two types of neural network models, the best results came from the DNN with 87.6%. Furthermore, by comparing the experiments, they concluded that the deep neural network model overall outperformed the traditional machine learning algorithm. In addition, in 2022, Sakeh et al. conducted a survey on the application of computer vision and deep learning for fish classification. They summarised the deep learning and computer vision research conducted over the last 20 years on underwater fish classification and analysed the results produced by each year [48]. The survey reflect that the number of studies using a conjunction of deep learning and computer vision has increased significantly since 2016, and the obtained accuracy rates have improved compared to previous computer vision studies. Thus, Sakeh et al. believe that deep learning techniques are an economical and effective solution for processing underwater video data and, combined with computer vision techniques, can transform an expensive task into a simple problem that can be handled by a computer [48].

In summary, based on the existing research described above, deep learning and computer vision techniques are effective ways to solve the problems that were described in Section 1.1.

1.4 Aims and Objectives

This project aims to use deep learning techniques and computer vision techniques to detect the occurrence and time points of predation events in a video. In particular, a dual-stream network architecture will be used to process the temporal and spatial information in the video. The entire dual stream model consists of three models, each with the following objectives and final overall goals:

- Create two distinct image datasets: one with object labels based on bounding boxes, and the other with binary classification image datasets based on fish-related images from the video dataset.
- Label the time point when the predation event occurred in the provided video dataset.
- Implement the object detection model on a single frame based on the YOLOv5 model.
- Implement an image binary classification model to determine whether contains a fish on a single frame based on the ResNet neural network model.
- Use the dense optical flow algorithm to calculate optical flow in the video.
- Implement an LSTM model to detect the predation event in the video based on the output of two previous models and the optical flow of the video.

1.5 Challenges

Insufficient Training Samples

In the provided video data, only 20 videos contain penguin predation events out of 63 videos. In a total of 20 videos out of 300 minutes, there were only 188 predation events. This limited amount of data brings challenges to developing accurate deep-learning models and also limits the ability to evaluate their performance. In addition, the significant imbalance between the number of predation and non-predation event samples could lead to the model being biased towards predicting non-predation events, potentially affecting its overall effectiveness.

Chapter 2

Technical Background

This section presents the content of the research and methodology involved in this project, including the description of the task, basic required theoretical knowledge, specific concepts and training tools, and detailed explanations of the performance metrics used to evaluate the model results.

2.1 Computer Vision

Computer vision is a field of artificial intelligence (AI) that is concerned with extracting and analysing information from visual data of that scene, such as pictures or videos[45]. It is an interdisciplinary field including digital image processing, machine learning and artificial intelligence that attempts to provide computer systems with the ability to self-learn the visual data. And also make it be able to understand the data and distinguish differences in and between images. There are many popular computer vision applications that relate to trying to identify things in images, such as object detection, object recognition and so on [2]. Figure 2.1 shows some example applications. This project can be classified as a computer vision task because it involves analysing visual data of penguins in water to develop a system that can identify and track the movements of penguins.

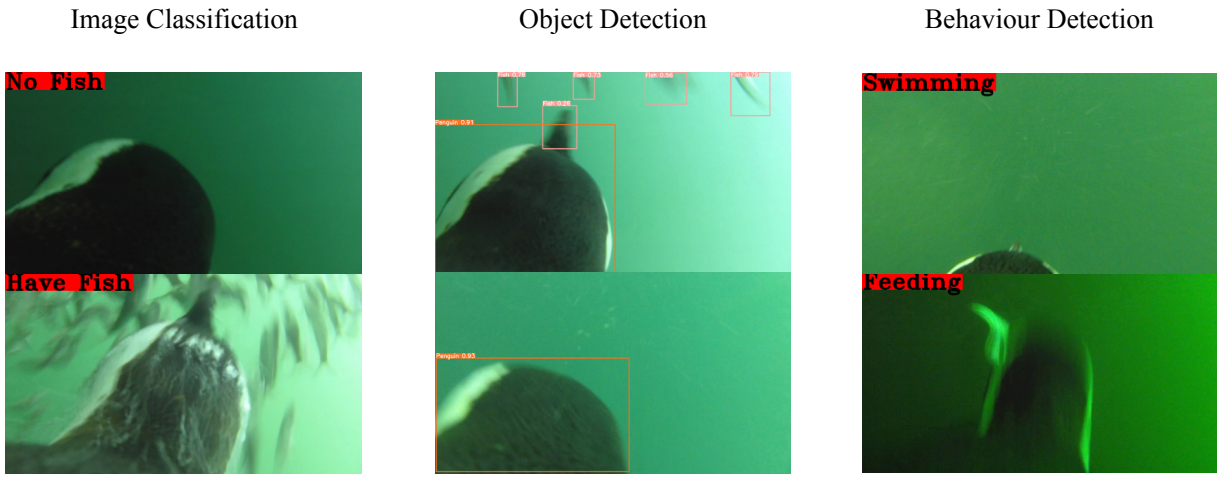


Figure 2.1: **Task Example**: This figure shows examples of object detection tasks, image classification tasks and behaviour detection tasks.

2.1.1 Image Classification

In the penguin diving video, determining the existence of a fish in each frame of the video is an image classification task. Image classification is a task in the field of computer vision research that involves classifying an input image into different categories [15]. In general, this task involves using a computer to analyse the features in the input image and compare these features with the features of images from known categories to determine which category the input image should be classified into. This process

involves building and training a classifier that takes a set of images from known categories as training data so that the classifier can learn and summarise the features of each category. After that, the classifier can classify a new input image as one of these known categories.

2.1.2 Object Detection

Identifying and locating the class and position of African penguins in a video is a object detection task in the field of computer vision. The task can be divided into two main parts, one is target classification and the other is target localisation [59]. Target classification is the classification of targets in an image into different categories. Target localisation is to locate the position of a target, usually using bounding boxes to indicate the location and size of the target.

2.1.3 Temporal Behaviour Detection

Understanding and recognising the behaviour of penguins by observing consecutive frames in a video is a temporal behaviour detection task. Traditional behavioural detection involves classifying a given video clip, generally using data where the actions are first segmented and each clip has a well-defined behavioural label. Compared to traditional behavioural detection tasks, temporal behaviour detection not only detects the occurrence of an action in a video but also needs to detect when this action happened in the videl. This task is often applied to long unsegmented videos and the target behaviour is generally only a small part of the video, which can also be understood as the detection of a specified behaviour in the video.

2.2 Machine Learning

Machine Learning is an essential branch of the field of artificial intelligence in computer science, which specialises in the study of computers that simulate or implement human learning behaviours [33]. This means that a computer learns from a large amount of available data through statistical algorithms or models that can extract features from the data and then use the generated experience to make decisions and predict the target by itself. With the development of machine learning, more and more machine learning models have been developed, such as SVM and so on. Because of the ability that machine learning gives computers to learn, it is used in many fields.

2.2.1 Supervised Learning

When training a penguin detection model or a penguin behaviour recognition model, the data used is labelled, which can be seen as a form of supervised learning. Supervised learning is where each sample being used for training has a corresponding expected value. Mapping from the input feature vector to the target value is achieved by building a model. Models with supervised learning rely on specific labels or targets, which means that the model is trained to minimise the error signal. The supervised learning process is shown in Figure 2.2.

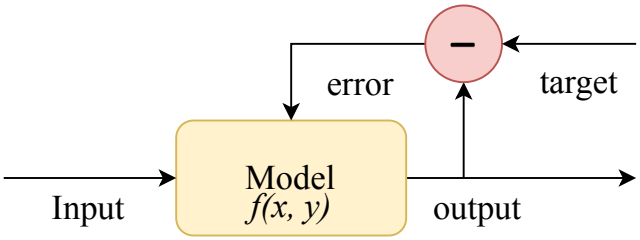


Figure 2.2: **Supervised learning model**: When using supervised learning, the model also needs to be given a correct label when given input.

2.2.2 Deep Learning

Deep learning is a method in machine learning that uses artificial neural networks (ANNs) [49] as the architecture for learning the features and laws of data. Deep means that the neural network structure

used is multi-layered, with each layer containing multiple neurons. Learning means that data is fed into the constructed network and patterns are learned by automatically adjusting the parameters of each layer and neuron in the layer to minimise the error between the output and the target result.

2.3 Neural Network

The neural network is a complex network structure that is formed by several interconnected neurons. It is based on modelling the organisation of the human brain to create an operational mechanism and uses mathematical models to simulate neuronal activity.

2.3.1 Artificial Intelligence Neural Network

The artificial intelligence neural network, also named ANN, was first proposed by Minsley and Papert in 1943 and was also known as a multi-layer feed-forward neuronal network [34]. The ANN typically includes three types of layers, input layer, hidden layer and output layer. The left-hand side of Figure 2.3 shows a simple architecture of an ANN. It can be seen that the data enters the model from the input layer and make a output from the output layer of the model after a series of calculations in the hidden layer and output layer. The ANN model can have multiple hidden layers and there are multiple neurons inside each layer. Additionally, the right-hand side of Figure 2.3 shows the computational activity of a single neuron. As shown, a neuron can have multiple inputs and compute an output. There is a weighted connection between each input and neuron, which can amplify or suppress the input. During training, these weights are consequently updated to achieve a better mapping of the approximation function mapping. In the computation of a neuron, the neuron summarises the product of all the inputs and their corresponding weights and adds a constant bias to them. At this point, the calculation is linear. After the linear calculation, the result will pass an activation function to do the non-linear calculation, which adds non-linear factors to allow the model to learn and understand more complex non-linear data.

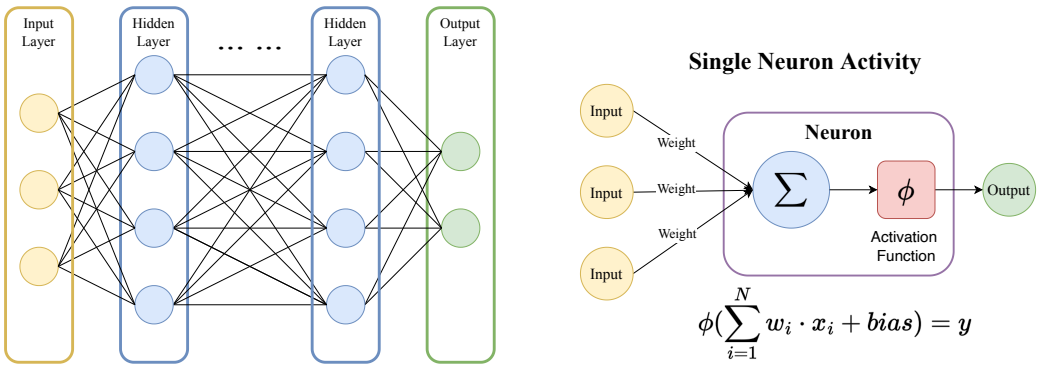
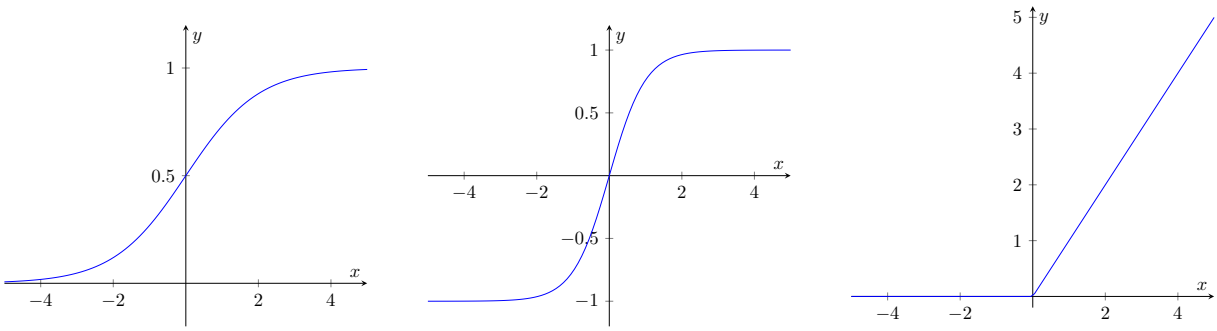


Figure 2.3: **Artificial intelligence neural network**: The structure of ANN (left) and the flow chart for the single neuron activity (right)



(a) sigmoid (b) tanh (c) ReLU

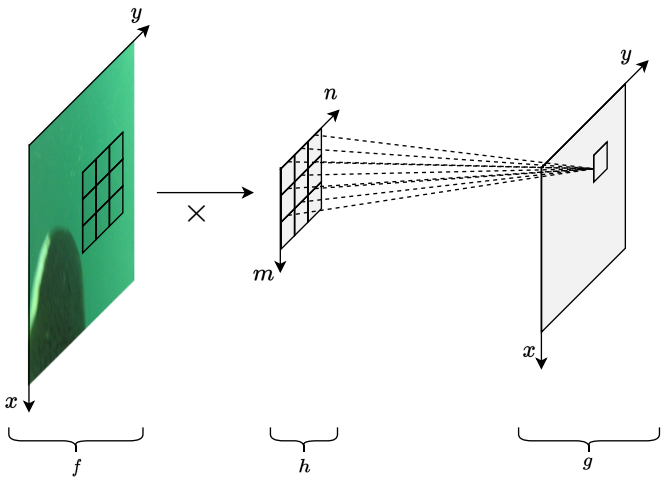
Figure 2.4: **Activation function**: The figures show three common activation functions. sigmoid takes values from 0 to 1, tanh takes values from -1 to 1 and ReLU takes values from 0 to positive infinity.

2.3.2 Activation Function

The function of activation is applied to the neurons in a neural network, mapping the inputs of the neurons to the outputs [5]. The non-linear activation functions allow the network to fit more complex data. Leshno et al. proved that the use of a non-polynomial activation function in a multi-layer feed-forward neural network can make the network approximate any function [27]. Figure 2.4 shows the function graphs of the three commonly used non-linear activation functions and their corresponding mapping ranges.

2.4 Convolution Neural Network (CNN)

For image data, basic ANN might not be applicable because it is structured data and has local features. For example, in an image of a penguin diving, there is usually a white area on both sides of the head of the penguin and it is difficult for an ANN to capture such local features when the image is flattened into a vector. However, the convolution operation in the convolution neural network can solve this drawback of the basic ANN. The convolution neural network is a special type of ANN whose main characteristic is that it contains a convolutional layer in the network and outputs a feature map containing the features in the data. Figure 2.5 shows a process of basic convolution operation in the convolutional layer to output a feature map [1]. There is a convolution kernel that scans the input image from left to right and top to bottom. At each scan, each element of the convolution kernel is multiplied by the pixels of the image it covered and the result is added together to obtain a scalar. The result of each convolution operation is stored in the feature map at the corresponding position to the input image.



$$g(x,y) = \sum_m \sum_n f(x-m, y-n)h(m,n)$$

Figure 2.5: **Convolution operation**: This figure show the convolution operation in CNN. The re is a convolution kernel that scans the whole image and does the operation with the overlapping pixels to achieve feature extraction.

Following feature extraction in the convolutional layer, the feature map is transformed into a one-dimensional vector. This vector is then forwarded to a fully connected layer, where it is used to generate the final outcome. The number of outputs produced by the last fully connected layer corresponds to the number of classes required for the objective classification task, with each value representing the likelihood of belonging to a specific class. For example, in the fish detector of this project, the output is the probability of the image having fish and not having fish.

The convolutional layer of the CNN, typically contains multiple convolutional kernels and each kernel can extract a specific set of features. Similar to ANNs, the weights of these convolutional kernels and neurons are adjusted during training, which aims to make the CNN use the appropriate convolutional kernels for a specific task and various types of features. Thus, CNN avoids the complex process of manually designing and selecting convolutional kernels. Because of these mentioned key features of the CNN, it motivated this project to select it to implement the global fish detector.

2.4.1 ResNet

The residual neural network (ResNet) is a deep convolution neural network, which was proposed by Kaiming He et al. in 2015 and has been widely used for image classification tasks [17]. It involves adding residual connections to a convolutional neural network, which allows the model to reach deeper layers. In its internal structure, it is mainly built by residual blocks. The operation flow of the residual block is that the input data is summed with the output after two or three convolutional operations, and then passed through the activation function to produce the output. Figure 2.6 shows two types of residual block in ResNet.

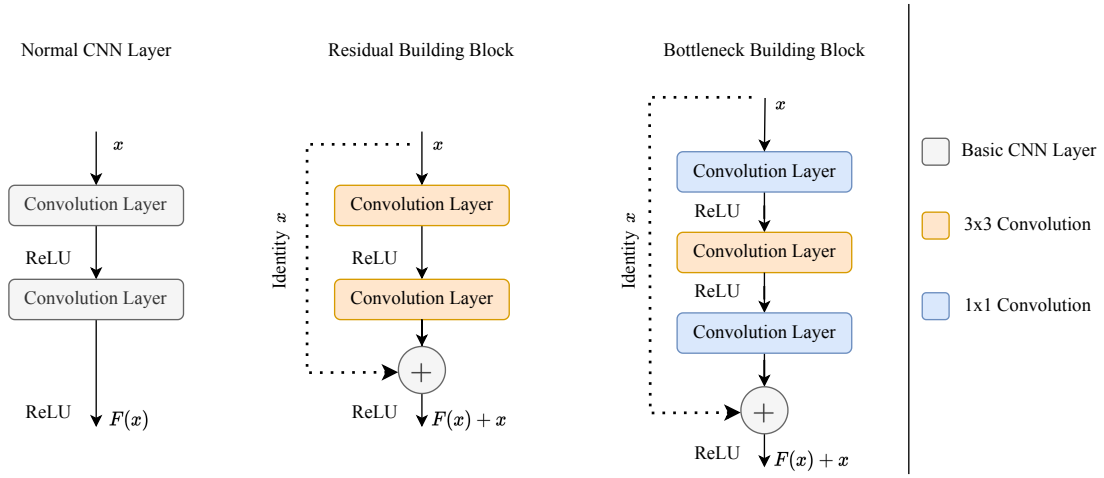


Figure 2.6: This image shows the difference between a normal convolutional layer and a residual block. In the residual block, there is a residual connection to add the input data and the output data together. In the bottleneck structure, it uses two 1×1 convolution kernels to do the dimensionality reduction and dimensionality increase. This picture is inspired from [17].

The residual block with two convolutional operations is usually used in shallow ResNet, such as ResNet-18. While the bottleneck structure three-layer residual block is mainly used in the deeper ResNet, such as ResNet-50. In the bottleneck structure, the first convolutional layer of the bottleneck structure is doing a 1×1 convolution operation for data downscale. Afterwards, a convolution layer with a 3×3 convolution kernel is used for feature extraction. Finally, the output is upscaled by a 1×1 convolution operation. The bottleneck structure reduces the computational volume of the deeper model and ensures the ability of the model to extract features. Because the ResNet has small number of parameters, small computational volume and deeper model layers, this project uses it to build a global fish detector.

2.5 Recurrent Neural Network (RNN)

Recurrent neural networks (RNNs) are a special type of artificial intelligence neural network that is designed to process sequential data. In 1986, Jordan introduced the concept of recurrent and created the Jordan network [24]. Later in 1990, Elman changed the Jordan network and used the backpropagation algorithm (BP) to update the weights for training [11], which formed the current recurrent neural network. The left-hand side of Figure 2.7 illustrates the architecture of a simple RNN neural network. Compared to the classical ANN described in section 2.3.1, the nodes of its hidden layer are also connected to each other, but it does not have to be fully connected. In addition, the right-hand side of the figure shows the internal computational flow of the unfolding of the RNN, which illustrates that the input to a neuron contains the input from the input layer and the output of the neuron from the previous moment.

These features give the RNN the ability to memorise the previous information and apply it to the computation of the current output. In this project, the video can be interpreted as a sequence of time-series data, making it a viable option to employ an RNN for analysis.

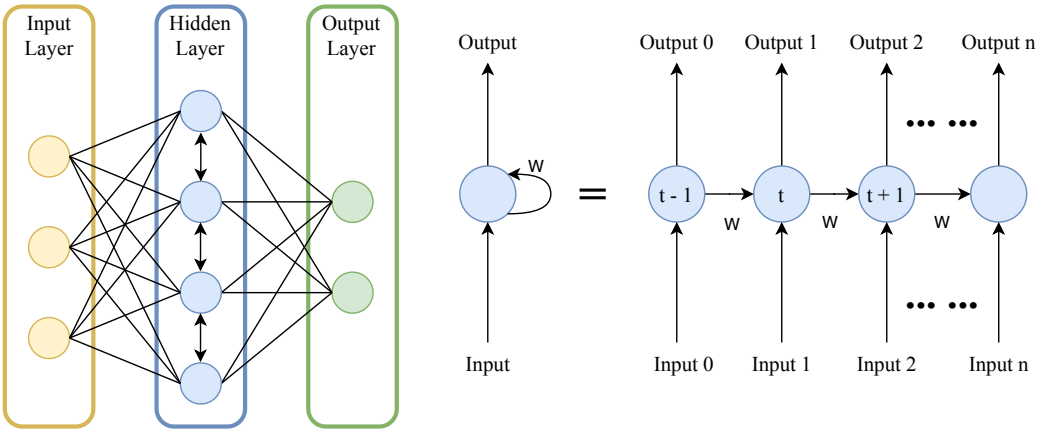


Figure 2.7: **Recurent neural network**: The simple structure of RNN (left) and unfolded unit of RNN (right). This figure inspired from [16].

2.5.1 Long Short-Term Memory (LSTM)

The long short-term memory neural network is an type of the recurrent neural network, which was proposed by Hochreiter and Schmidhuber in 1997 [20]. The neurons in LSTM neural networks are replaced by memory cells that can selectively store and read information, which gives the network ability to remember for a long time and effectively avoid the problems of gradient explosion and gradient disappearance. Therefore, it has been shown to handle time series problems [20]. Figure 2.8 show the internal structure of the LSTM memory cell, which includes three gates and a cell state [28]:

- **Cell State:** Cell state constitutes the memory portion of the LSTM. As depicted in the diagram, information can be transferred from the previous time state to the current time memory state cell computation.
- **Forget Gate:** The forget gate is crucial in determining which information from the last time state cell state should be retained or discarded. It will receive information from the hidden state of the previous time state and the information in the current time state, which will be passed into the *sigmoid* function simultaneously. The output of the forget gate is a number from 0 to 1. After that, it is multiplied by the cell state from the previous time state. Near to 0 indicated it should be forgetton, and nearer to 1 indicates it should be retained.
- **Input Gate:** The input gate regulates the flow of new information into the cell state of a model. First, the information from the previous time state and the current time state input are transmitted to the *sigmoid* function, which determines what data should be updated. The same input is then sent to the *tanh* function, which generates the cell state of the current time state. Through multiplying the outcomes of both functions, the output of the *sigmoid* function determines which data from the *tanh* output should be used to updated the cell state. If this value is near 0, it should be dropped. Conversely, if it is near to 1, it should be updated.
- **Output Gate:** Determine the hidden state value for the current time step. The output gate is utilized. Initially the *sigmoid* function receives the previously hidden state and the current time state input. The updated cell state from the input gate is fed into the *tanh* function. Subsequently, the results from the two functions are multiplied to determine the hidden state of the current time state to be output, and the result is transferred into the next time state LSTM memory cell with the updated cell state.

In summary, because of the ability of neural networks for long and short-term memory, it is possible to achieve temporal behaviour detection in video. Particularly in the underwater video, M.G.B. Palconit et al. applied LSTM to the detection of fish movement tracks and obtained the lowest error in comparative experiments [40]. Therefore, in this project, LSTM can be used as an effective method to detect the hunting action of African penguins.

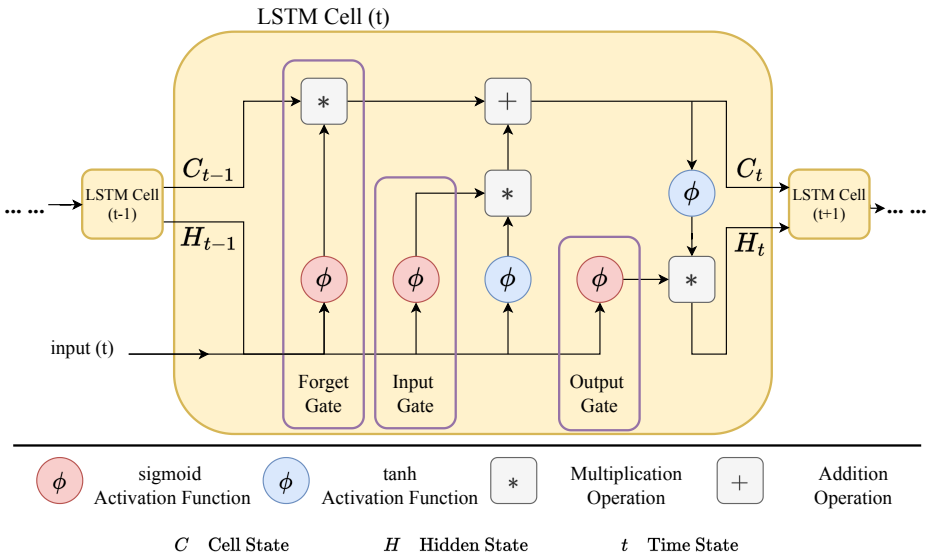


Figure 2.8: **LSTM memory cell**: The structure of a single LSTM memory cell and the three gates in the memory cell. This figure is inspired from [47]

2.6 YOLO Algorithm

Finding the location of penguins and fish in an image is the task of this project, which is an object detection task. You only look once (YOLO) is a real-time objective detection algorithm, which is based on CNN neural network, that can detect and classify penguins and fish at the same time in an image. As the name implies, YOLO only requires passing the image forward once in the network and produces two outputs simultaneously [43]. One is the position of objects at each location on the input image, and the other is the category in which each object is located. Thus, it can be used as an efficient method to find the location of objects in an image and determine whether they are fish or penguins.

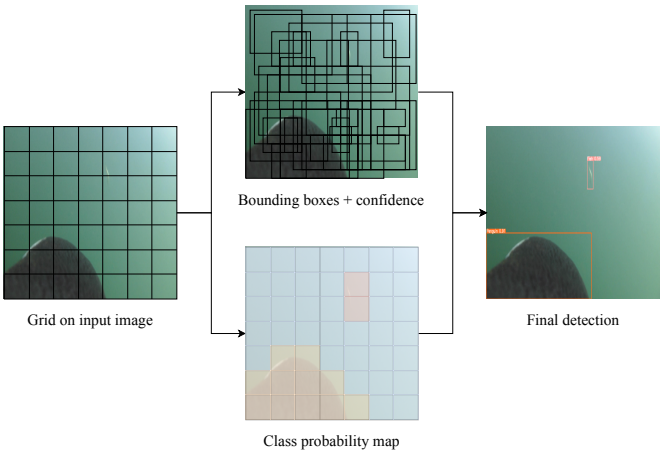


Figure 2.9: **The YOLO algorithm**: This is a simple calculation of the YOLO algorithm. It divides the input image into a grid and predicted several bounding boxes for each grid. After that, it predicts the confidence and class probability for each bounding box. Finally, the predicted results are combined to give the final detection. This figure is inspired by [43].

The whole algorithm is implemented as follows [43]. The input image is first divided into a grid. Then, for each grid cell in the image, a bounding box prediction is made and the confidence score of each bounding box is calculated. The bounding box prediction contains the centre coordinate of the box and the width and height of the box. The confidence score reflects how confident the model believes the bounding box contains the object. Also, each grid cell has a category probability and these probabilities are conditional

on the grid cell containing an object. Finally, these predictions are then combined to produce the final detection. This process is illustrated in Figure 2.9.

Since the presentation of the YOLO algorithm in 2016 [43], there have been several variants of the implementation. According to a study of the development of YOLO algorithms in the past, YOLOv5 is becoming more popular than other versions of YOLO [23]. It was developed by **Ultralytics** company and has five different depths of the YOLO models [54], which allows researchers to control the models more flexibly. In addition, based on these related applications [55, 22, 57], the YOLOv5 also has good performance for underwater image detection. Therefore, this project will use the YOLOv5 as the main object detection model to achieve the detection of penguins and fish in penguin diving videos.

2.7 Hyperparameters

Hyperparameters of a model are parameters that need to be set before the model training process. It is used to control the training process and performance of the model. They are not learned by the model and instead need to be manually selected. Therefore, better initial values of them will produce better convergence and better results from the model. The following model hyperparameters are relevant to the experiments conducted in this project.

2.7.1 Loss Function

The concept of loss in machine learning refers to a numeric representation of the difference between the predicted and actual output of a model, which can serve as a measure of its performance. The task of computing the loss is assigned to the loss function. There are many different loss functions, and even given the same values they are calculated differently. Thus, choosing a suitable loss function is crucial for model training and evaluation. The loss functions that will be used in this project are as follow:

- **Cross-Entropy Loss:** The cross-entropy loss function is commonly used in classification problems [60] and measures the difference between the predicted and actual distributions of a model. For an input sample x, its cross-entropy loss function is the sum of the negative logarithms of the differences between the actual labels of all categories and the probabilities predicted by the model.

$$L_{CE} = -\sum_{i=1}^{C} y_i \log p_i \tag{2.1}$$

  where C is the total number of class; y_i is the actual label of sample x in class i, the range is 0 to 1; p_i is the model predicted probability of the sample x that belongs to the class i.

- **Focal Loss:** Focal loss is a loss function to solve the hard classify sample in unbalanced datasets [30]. In the traditional cross-entropy loss function, the weight of the loss function is the same for all samples [41]. However, the focal loss function makes the model focus more on hard-classified samples by adjusting the weights of the samples, which can improve the classification accuracy of the model on unbalanced datasets.

$$FL(p_t) = -\alpha_t(1-p_t)^{\gamma}\log(p_t) \tag{2.2}$$

  where p_t presents the probability of the predicted class is correct; α_t is the coefficient of weights; γ is the hyperparameter that adjusts the weights of the hard and easy samples. When the prediction probability p_t is smaller (i.e. the more uncertain the prediction of the model for the sample), the $(1-p_t)^{\gamma}$ will become larger, which will increase the weight of the hard classify samples in the loss function. Conversely, when the prediction probability p_t is larger (i.e. the more certain the prediction of the model for the sample), the $(1-p_t)^{\gamma}$ will become smaller, which will decrease the weight of the easily classified samples in the loss function. In computer vision, the focus loss function has been used in many tasks. In 2017, Lin et al. implemented a dense detection model and found that the model with the focal loss function was more accurate in unbalanced datasets [30]. In the dataset of the penguin and fish detectors for this project, it is difficult to ensure that the numbers of fish and penguins are the same, so to avoid this imbalance in sample size, the Focal loss calculation will be applied in penguin detector.

2.7.2 Optimiser

During the model training phase of machine learning, the optimiser is an algorithm used to adjust the parameters of a model to minimise loss. When updating the weights, the neural network utilises the back-propagation method [18]. The neural network begins by computing the gradient of the loss at the present weight. The gradient is computed using the chain rule by beginning at the output layer of the network and iterating backwards, layer by layer, until the starting layer is reached. After that, the network then updates the weights in the opposite direction of the calculated gradient by using the optimiser. The following are associated optimisers with the project:

- **Stochastic gradient descent (SGD):** SGD is a commonly used optimiser that focuses only on the current model weights to calculate the gradient. When the gradient of the loss function is derived, the magnitude of the decrease in each parameter is calculated based on the gradient and the learning rate. Afterwards, the current parameters are subtracted by the size of the gradient decline to determine the new parameter values, as depicted in Equation 2.3.

$$W_{t+1} = W_t - \alpha \cdot \nabla L(W_t) \tag{2.3}$$

  where W is the parameter of the model; α is the learning rate for the optimiser; $\nabla L(W_t)$ is the gradient of the loss function; t is time step.

- **Stochastic gradient descent with momentum:** In comparison to the base SGD, the SGD with momentum has a momentum term added. The momentum term will take into account the previous gradient direction and the current gradient direction, and accelerate the gradient descent along the direction of descent [19]. Thus, this allows the model to jump out of the local optimum and accelerate the convergence of the gradient descent. The formula for the SGD optimisation algorithm with the addition of the momentum term is shown in Equation 2.4.

$$\begin{aligned} v_{t+1} &= \beta v_t + \alpha \cdot \nabla L(W_t) \\ W_{t+1} &= W_t - v_{t+1} \end{aligned} \tag{2.4}$$

  where W is the parameter of the model; α is the learning rate for the optimiser; $\nabla L(W_t)$ is the gradient of the loss function; t is time step; v is momentum; β is the momentum coefficient.

- **Adaptive Moment Estimation (Adam):** Adam is an optimisation algorithm that optimises the parameters of a model by calculating the exponential moving average of the mean and squared value of the gradient for each parameter [26]. The exponential moving average of the mean of the gradient represents the average trend of the gradient. And the exponential moving average of the squared value of the gradient represents the magnitude of the variation of the gradient. Adam uses this information to adaptively adjust the learning rate of each parameter in the model. Because of this feature, the model can converge faster and more efficiently in computation. In addition, the complex process of manually selecting the learning rate can be avoided.

2.7.3 Learning Rate

The learning rate plays a crucial role in regulating the magnitude of each parameter update while training a neural network [21]. For instance, in Equation 2.3 of SGD, α of stochastic gradient descent, where the gradient is multiplied by the value of α, is the learning rate. The equation indicates that setting a high value for the learning rate could result in faster gradient descent. Conversely, if the learning rate is too low, the gradient descent will proceed slowly, leading to a longer convergence time.

2.8 Model Optimisation Techniques

During the training of deep learning models, model optimisation techniques optimise model performance. Listed below are some of the model optimisation techniques attempted in this project.

2.8.1 Dropout Layer

The dropout layer is a type of neural network layer that can reduce the overfitting of the model [52]. Overfitting happens when a machine learning model performs better on training data than on test data

[10]. The reason for this phenomenon is that the model has overlearned the features of the training data. During the training of the model, the Dropout layer randomly drops some neurons [58] based on a dropout rate that is a type of hyperparameter to control the probability of each neuron being dropped. Figure 2.10 shows an example of the connections after dropping some neurons. This method permits training the model with a certain amount of randomness, which allows the model to learn more robust features.

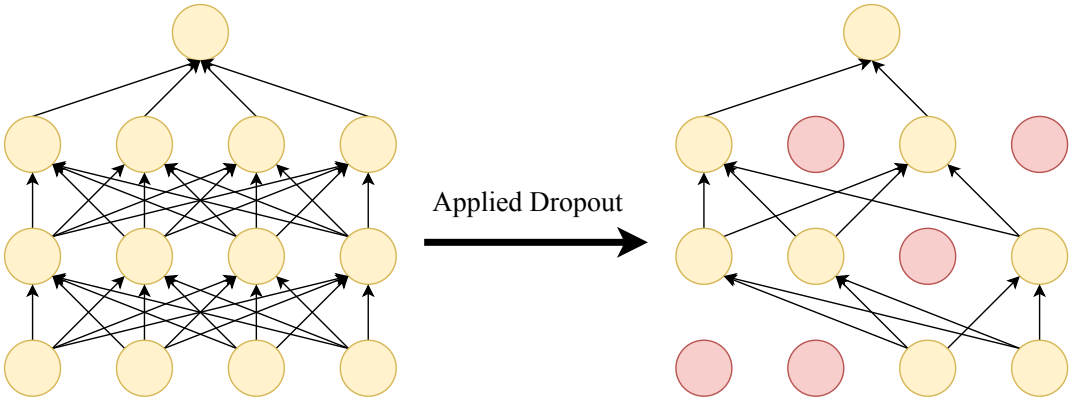


Figure 2.10: **Dropout**: The comparison of the connection of neurons in the model before and after using Dropout. The dropout layer randomly dropout some connections of neurons. This figure is inspired from [58].

2.8.2 L2 Regularisation

L2 regularisation is also a way to reduce the phenomenon of model overfitting, which is achieved by adding a regular term to the loss function calculation. The following Equation 2.5 illustrates this process:

$$L(W) = L(W) + \lambda \sum w^2 \tag{2.5}$$

where the first part of the equation, $L(W)$, is a loss function that is calculated by predicted value and actual value. The second part, $\lambda \sum w^2$, corresponds to the L2 regularisation, which includes a hyperparameter λ to control the strength of the regularisation and a sum of squares of the weights. When the weights are large, the sum of squares will be large, which leads to a larger regularisation term and its loss will be larger. Therefore, in order to minimise the loss, the model prefers to choose small weights, which will reduce the complexity of the model and the risk of overfitting situation [36, 58].

2.8.3 Data Augmentation

Data augmentation is an approach for improving the performance of a model developed using machine learning, when a series of transformations are applied to the original training dataset to attempt to add some new samples. For image-type datasets, operations such as flipping, adding noise and rotating are effective methods of data augmentation. By augmenting the training dataset, the model can learn more samples to improve the generalisation ability of the model [51, 58].

2.9 Optical Flow

Extracting information about the movement of the penguin in the video can be obtained by observing the changes in two consecutive frames. Optical flow is a method used to describe the movement of a penguin (either the movement of the penguin itself or the movement of the camera) between two consecutive frames in a dive video. Basically, it is a two-dimensional vector field and each vector in it represents the displacement of the pixel point from the previous frame to the next frame.

2.9.1 Dense Optical Flow

There are two type of optical flow calculation, one is dense optical flow and the other one is sparse optical flow. In this project, the penguin predation events detector wants to know all the motion information

in the video, which can be done by using dense optical flow. Compared to sparse optical flow, dense optical flow calculation calculates the optical flow for every pixel in the image, which will produce more accurate information about the optical flow of the image [25]. This method allow the model to capture the movement of the penguin in the video more accurately. However, it takes a long time to compute because of the large amount of computation.

2.10 Metrics

For an understanding of the performance of the trained model in identifying penguins and detecting penguin predation events in videos, it is necessary to evaluate the model using specific metrics. The following metrics are related to this project and will be used in the different components of the whole architecture:

- **Confusion Matrix** [29]: The three subtasks in this project can be considered as a classification task, so the confusion matrix can be applied as an evaluation method in this project. Table 2.1 shows an example of a binary classification confusion matrix. in addition, for each class, the correct prediction is referred to as a positive and the incorrect prediction as a negative. The confusion matrix contains four elements about the prediction, TP, FN, FP and FN.
  - **TP**: True positive, i.e. samples that are actual class c are predicted to be class c.
  - **FN**: False negative, i.e. samples that are actual class c are predicted to be not class c.
  - **FP**: False positive, i.e. samples that are not actual class c are predicted to be class c.
  - **TN**: True negative, i.e. samples that are not actual class c are predicted to be not class c.

Actual/Predicted	Positive	Negative	Total
Positive	TP	FN	TP+FN
Negative	FP	TN	FP+TN
Total	TP+FP	FN+TN	TP+FP+FN+TN

Table 2.1: **Confusion Matrix**: This is an example of a binary classification confusion matrix. Each row and column of the confusion matrix represents the actual labels and the predicted labels by the model, and the elements of the matrix represent the classification results.

- **Accuracy**: The accuracy rate is the proportion of the number of samples correctly predicted by the model to the total number of samples. The formula is calculated as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.6}$$

- **Precision**: The precision rate is the proportion of samples with true positive instances to the number of samples with positive predictions. The formula is calculated as:

$$Precision = \frac{TP}{TP + FP} \tag{2.7}$$

- **Recall**: The recall is the proportion of the number of samples correctly identified by the model as positive examples to the number of all positive samples. The formula is calculated as:

$$Recall = \frac{TP}{TP + FN} \tag{2.8}$$

2.11 PyTorch

PyTorch is a Python library optimised for deep learning, which supports running on CPUs and GPUs[42]. In Pytorch, models can be built directly by calling the API. In addition, some of the libraries already provide some built models, such as **torchvision**, which contains the commonly used models in computer vision and also contains some pre-trained weights. Furthermore, PyTorch provides a GPU acceleration library specifically for neural networks with support for NVIDIA CUDA. In this project, the GPU-accelerated library improves the speed of model training and evaluation when faced with large amounts of image processing. Overall, the use of PyTorch greatly facilitates the training of models and can be combined with other commonly used Python libraries.

2.12 OpenCV

In order to detect the temporal behaviour of penguins in the video, it is necessary to process the image data in the video and analyse each frame of the video. OpenCV [39] is an effective tool for this purpose. It is an open source computer vision library that provides a rich set of image processing methods and computer vision algorithms and supports the Python language. Therefore, it can be applied to many types of computer vision tasks. Particularly in this project, video processing, such as reading video files, extracting video frames, optical flow algorithms, etc., will be achieved by using functions from the OpenCV library.

2.13 MegaDetector

MegaDetector is an object detection model based on deep learning technology developed by a team of Microsoft researchers, especially for use in the field of wildlife conservation. It aims to identify animals, people and vehicles in camera trap images. It was trained using camera trap image data from the Ecological Institute of British Columbia Canada, and achieved 82% accuracy in accuracy in the detection of animal categories [12].

2.14 Penguin Only Detector

Penguin only detector has been developed by Mingyu Yang at the University of Bristol in 2022 [57]. The detector is specifically designed to detect and identify the presence of penguins in the penguin dive video, which is based on the YOLOv5 model. In the research by Mingyu, there is a comparison of MegaDetector v4 and v5 based on the dataset described in Section 1.2 and the conclusion was made that MegaDetector v5 had a better performance. Furthermore, Mingyu has fine-tuned a penguin detector based on the YOLOv5 model by using the same video dataset. The detection results are shown in 4.4.

2.15 Inflated 3D Convolution Neural Network

The I3D model is a two-stream inflated 3D convolutional neural network based on a 2D convolutional neural network designed to handle video classification problems [3]. It uses the two-stream network approach, which includes two convolutional neural networks to be a spatial network and a temporal network. In order to extract spatial information from the RGB images in a video, a spatial network is employed. Subsequently, a temporal network is used to extract motion information from the optical flow of the video. After that, the classification of the video is achieved by merging the features extracted by the two networks. Carreira et al. proposed I3D in 2017 and demonstrated its effectiveness on the Kinetics dataset of human action videos [3]. This dataset comprises 400 classes of human behavior, each with 400 video clips. Consequently, the I3D model will be evaluated and compared to the final behavioral detection model of this research as a state-of-the-art model with proven success.

Chapter 3

Project Execution

This chapter will demonstrate the entire project procedure. During the process, the composition of the project dataset and the implementation and methodologies utilised for each component of the model will be described.

3.1 Platform

In this project, I develop each deep learning model using the PyTorch framework and the CUDA library that can make the model trained on GPU. Additionally, all the experiments were run on the high-performance computer that is Blue Crystal Phase 4, with Intel Xeon Processor E5-2680 v4 CPUs and NVIDIA Tesla V100 GPUs, at the University of Bristol [4].

3.2 Dataset

The video dataset is supplied by Dr Richard Sherley from the University of Exeter, which is introduced in Background 1.2. Based on that, I create three different datasets for this project to train the specific model, which are the local frame detection dataset, whole frame detection dataset and multi-frame detection dataset. The following section describes the contents of each dataset and information about the data.

3.2.1 Local Frame Detection Dataset

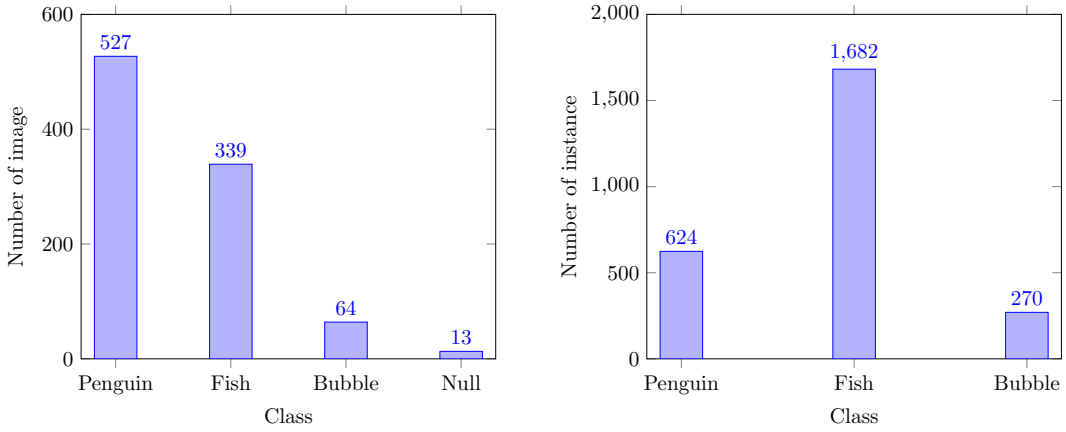
The local frame detection dataset consists of 602 images, which are selected from the supplied video dataset. All the images were scaled from 1920×1080 to 640×640 in order to reduce the training time. These images were divided into two folders, one for the training set and the other for the test set. The training set contains 418 images, and the test set contains 184 images. In this dataset, each object in the image was labelled by a bounding box and each image has a matching YOLO format *.txt file, Listing 3.1 is an example. Each line in the file indicates the category of the object in the image that is labelled by the bounding box and the size and location of the corresponding bounding box.

```
2 0.41328125 0.81328125 0.4609375 0.3734375
1 0.33359375 0.45625 0.04609375 0.21953125
1 0.58828125 0.30859375 0.03125 0.1734375
```

Code Listing 3.1: **Matched *txt file**: This txt file holds information on the category and position of the objects in the image corresponding to the file name. The numbers in each line represent a class of the object, the normalised x-coordinate and y-coordinate of the centre point of the bounding box, and the normalised width and height of the bounding box.

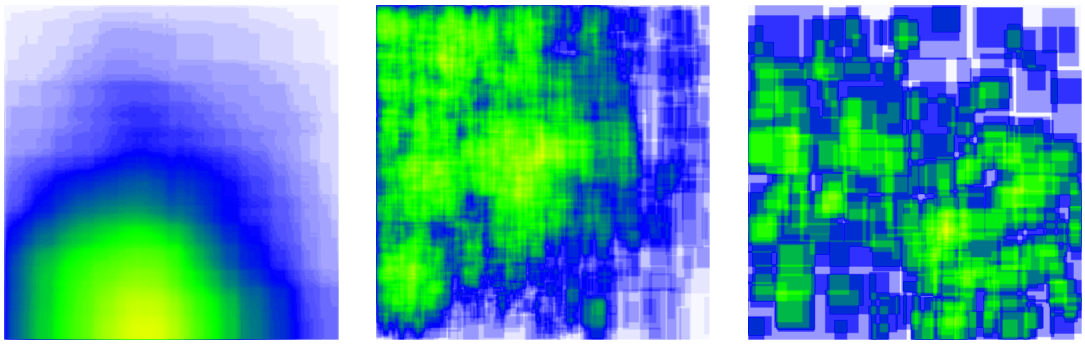
Subsequently, Figure 3.1 illustrates the number of instances and distribution of each category throughout the dataset. The graph clearly demonstrates a more significant number of fish instances compared to other categories, despite not having the biggest amount of fish-related photographs. Furthermore, Figure 3.2 displays the location heat map for each item category in the annotated dataset. Brighter colours indicate higher concentrations on the heat map, which depicts the relationship between the number of

instances for each category and their relative places in the picture in the dataset. Analysis of the heat map shows that most of the penguin instances are located in the lower middle of the image, while the fish instances are mainly concentrated in the left half. Besides, when combined with the penguin heat map, water bubbles can be seen to be located mainly around the location of the penguin instances.



(a) Number of image distribution (b) Number of instance distribution

Figure 3.1: **Distribution of local frame detection dataset**: These histograms show the number of samples of penguins, fish and bubbles in the local detection dataset, and the number of images containing them. There is a class imbalance in the local frame detection dataset.



(a) Penguin (b) Fish (c) Bubble

Figure 3.2: **Heatmap of annotation**: These heatmaps show the distribution of penguins, fish and bubbles labelled on the image in the dataset. The brighter areas are annotated more often.

3.2.2 Whole Frame Detection Dataset

The whole frame detection dataset contains 797 images, which are selected from the supplied video dataset, and was resized from 1920×1080 to 640×640. The dataset, the structure shown in Figure 3.3, is split into two sets: a training dataset with 558 images and a test dataset with 239 images. The images are categorized into two classes: "Have Fish" and "No Fish". There are 370 images labelled as "Have Fish" and 427 images labelled as "No Fish".

3.2.3 Multi-frame Detection Dataset

The multi-frame detection dataset contains the videos of a penguin dive and a JSON file. The videos of the penguin dives are all from the provided video dataset that is described in Section 1.2. However, in the original video dataset, some videos contain clips where the penguin is not diving and the image is background. Also, the original video is longer and the penguins are kept swimming for most of the time. Therefore, for training the model better, all long videos were cut into short videos that do not contain penguins not underwater. Afterwards, each short video was annotated with the predation events time

```
train
├── hasFish
│   └── *****.jpg
└── noFish
    └── *****.jpg
```

```
test
├── hasFish
│   └── *****.jpg
└── noFish
    └── *****.jpg
```

Figure 3.3: **Structure of whole frame detection dataset**: This dataset split the training and test data into two separation folders. In each folder, there are two folders representing two classes in the dataset.

point in the video and saved in the JSON file, the Listing 3.2 is an example of the annotation in the JSON file. Finally, there were 85 short videos and 188 predation events in this dataset. Figure 3.4 shows the distribution of the number of predation events contained in each video.

```
{
    "videoName": "20170929_A_6_00-12-53_00-13-18.mp4",
    "timeSegmentAnnotations": [
        {
            "displayName": "feeding_event",
            "startTime": "3.670s",
            "endTime": "3.670s"
        }
    ]
}
```

Code Listing 3.2: **The annotation of predation event in the video**: In the single annotation, videoName is the name of the penguin diving video, and the timeSgementAnnotations is a list that saves the time point of each predation event.

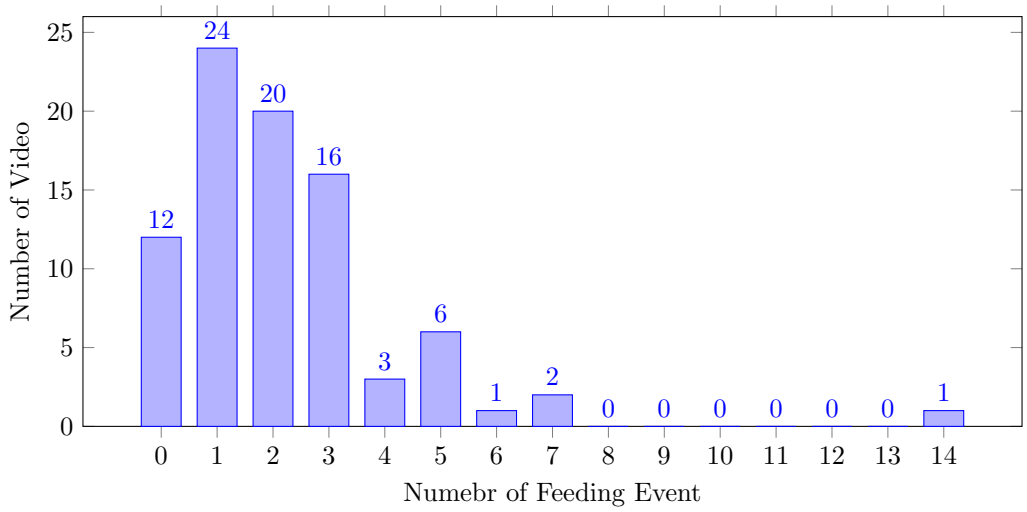


Figure 3.4: **Predation events distribution**: A bar chart showing the distribution of the number of occurrences of penguin predation in the cropped video in the multi-frame detection dataset.

3.3 Model Structure

In this project, I use a dual-stream network structure to detect predation events in penguin dive videos. Figure 3.5 demonstrates the flow chart of the dual stream network structure. In this Figure, the two streams process the spatial and temporal information in the video respectively. The spatial streams focus on acquiring spatial information about objects in the video, and it concatenates two models. One is based on the YOLOv5 model implemented for local detection of the image, which will generate the

position and class of the object in each frame of the video. The other is a global detection of the image based on the ResNet18 model, which will determine whether each frame of the video contains fish or not. The temporal stream processes the temporal information in the video. It will calculate the optical flow information between every two frames of the input video. At the end of the model structure, there is an LSTM model. It will receive the output produced by both streams at the same time and the penguin predation event will be detected. The implementation of each model is described in the following sections.

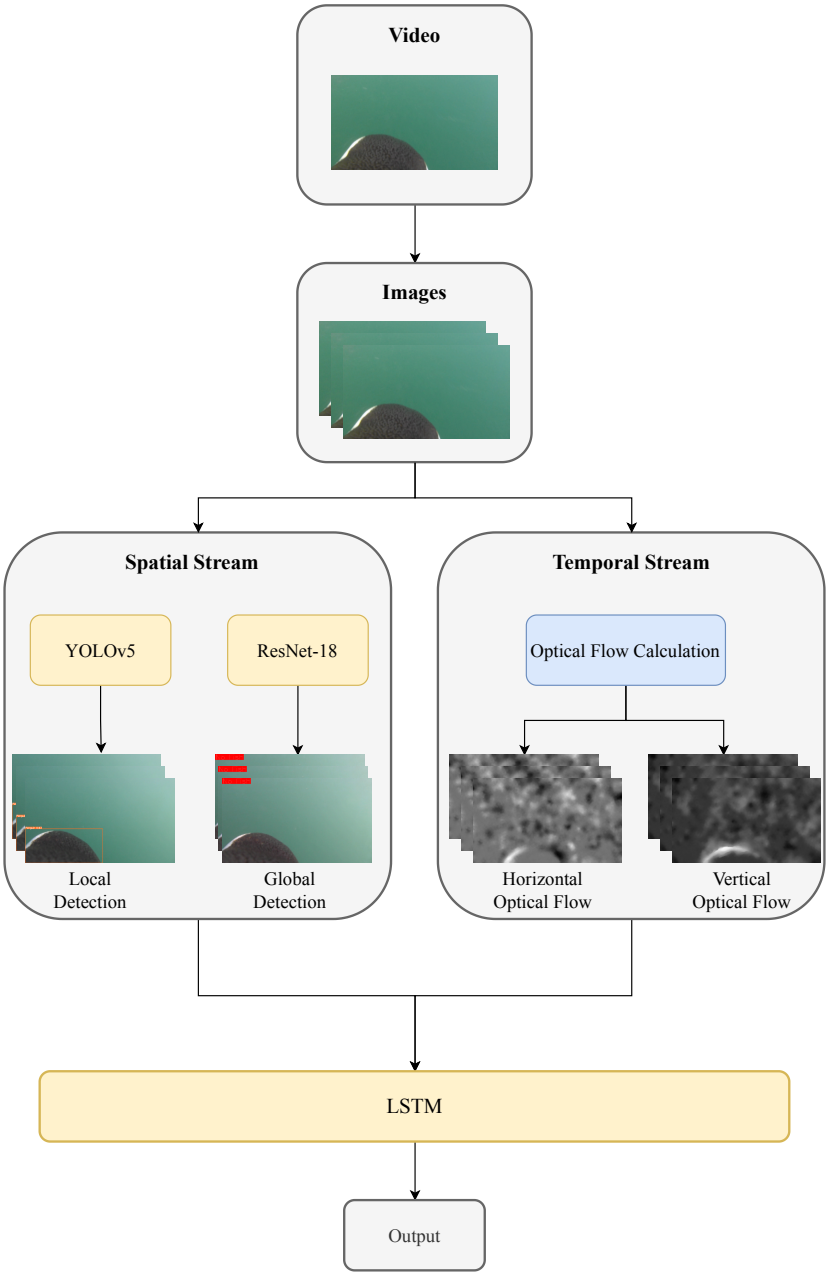


Figure 3.5: **Dual stream Network**: This is a diagram of the dual-stream network used in this project. On the left is the spatial stream, which shows the model and results used to obtain the location information of the fish and penguin in the video. On the right is the temporal stream, which is used to calculate the temporal optical flow information in the video. Both outputs of the two streams are integrated to determine whether the penguins are predating or not.

3.4 Spatial Stream

This section will describe the process of implementing spatial stream in the used dual stream network structure. It focuses on two types of object detection, that are local and global detection, and describes the implementation of these two types of object detection tasks.

3.4.1 Local Detection with YOLOv5

The local detection model is implemented based on the YOLO algorithm, which is a real-time object detection technique based on deep learning, as described in Section 2.6. Figure 3.6 illustrates the full process of implementing a local detecter and this section will describe each part of the process.

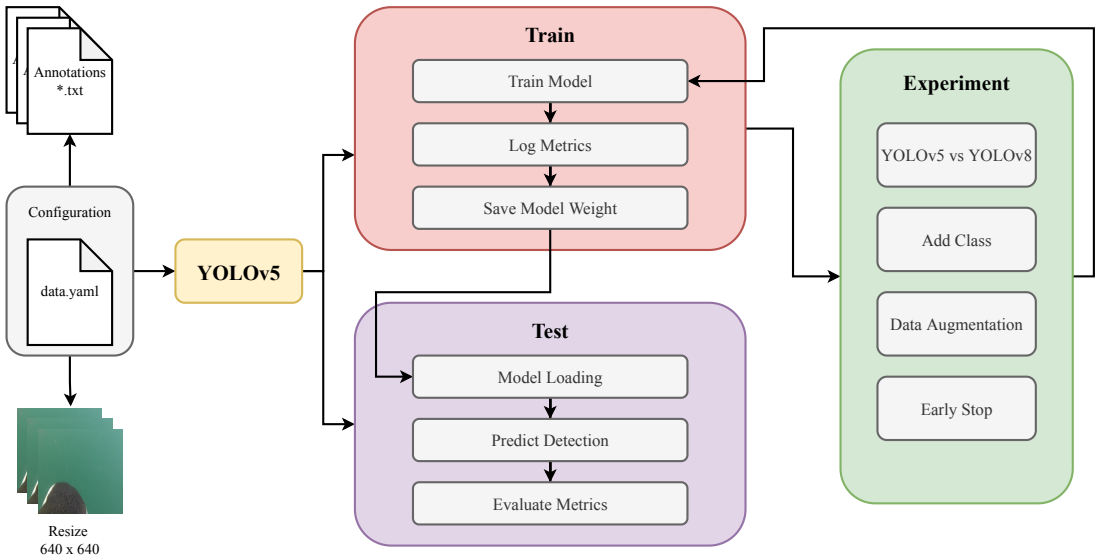


Figure 3.6: **Local Detection Implementation**: This flow chart shows the whole process of the local detector implemented in this project based on YOLOv5, and lists the relevant experiments.

Loading Data

In the process of training YOLOv5, the local detection dataset, described in Section 3.2.1, was used. The model requires a YAML file to import the data from the training and validation datasets in the dataset. The YAML file holds the root path of the entire dataset and the path of the training and test sets. It also holds the names of the category and the corresponding numerical labels in the dataset. The used YAML file in this project is shown in Listing 3.3.

```
path: ../video-dataset/coco128
train: images/train
val: images/test
test: images/test

# Classes
names:
  0: Bubble
  1: Fish
  2: Penguin
```

Code Listing 3.3: **data.yaml**: This is the required format of the data import configuration file during the YOLOv5 loading data in this project.

Pre-train

YOLOv5 officially provides two ways to train the model. One option is to randomly initialise the model weights for data learning at the start of the model training. The other option is to use a pre-trained

model weight as the starting point to learn the data. In this project, I used the official YOLOv5 pre-training model weight as a starting point for the training model. The official YOLOv5 pre-training model was trained on the COCO (Common Objects on Context) dataset [54]. The COCO dataset is a large-scale target detection dataset containing 328K images and 250K annotations for 91 object types [31]. Therefore, the pre-trained model of YOLOv5 is able to extract common features and help speed up the convergence of the model training.

Hyperparameters

As mentioned in Section 2.7, the hyperparameters of the model are crucial for the training of the mode. The YOLOv5 model has about 30 hyperparameters and its officials provide a hyperparameter file [54], shown in Appendix A, that they have optimised and validated on the COCO dataset. Therefore, the hyperparameter file provided by YOLOv5 was used in the training of the local detector for this project to achieve better results.

Train

The official training script provided by YOLOv5 [54] is responsible for the entire training process for this part. This script enables the setting of individual parameters for training by parsing the arguments on the command line, as shown in Listing 3.4. The YOLOv5 model is initialised by passing the path of a pre-written data configuration file and a pre-downloaded officially provided model pre-training weights file and parameter configuration file to the command line. Besides, parameters of the training process can be defined on the command line by passing in arguments, such as epoch, batch size and so on.

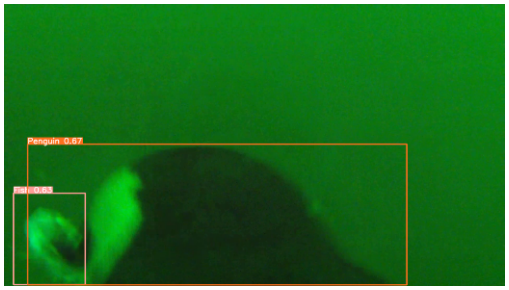
```
python yolov5/train.py --data data.yaml --weights yolov5s.pt --cfg yolov5s.yaml --
    save-period=200 --batch-size 6 --epochs 1000
```

Code Listing 3.4: **Run YOLO training script**: This command line shows how to run the training script for YOLOv5 model with specific arguments

A number of metrics are calculated during the training process, which includes class average accuracy, recall and loss. They are all recorded using Comet because it has integrated YOLOv5 into its library [6]. In the training script, the Comet library is imported and the API key is set in the Comet configuration file. During the training process, Comet will automatically record the metrics on the training dataset and validation dataset for each epoch and upload them to the dashboard of Comet in real time.

Model Output

Once the model has been trained, the fine-tuned model can be implemented to detect the location of penguins and fish in the penguin diving image. The model produces two outputs, an image with the calibrated bounding box and a corresponding *.txt file containing information on the detected object class and the position of the bounding box. Figure 3.7 shows an example of the output of the local detector.




```
1 0.0903646 0.832407 0.143229 0.324074
2 0.42526 0.745833 0.756771 0.499074
```

Output Image

Output *.txt File

Figure 3.7: **The output of fine-tuned YOLOv5**: This is an example of the results from YOLOv5 model. On the left is an image after the detection, with a bounding box for each detected object, and on the right is a *.txt file with the location and category information corresponding to each bounding box.

3.4.2 Global Detection With ResNet

In penguin dive videos, learning about the location of fish is essential for determining the predation events of penguins. However, it is not enough to rely on local detection to obtain information about the location of fish. The reason for that is fish are often seen in groups in the video, and it is difficult for a local detector to locate fish in a large and dense group of fish. Therefore, I trained a fish binary classification model as a global detector, which was used to determine whether fish were present in the image. It is built based on the ResNet neural network, which is a deep convolutional neural network as described in the 2.4.1 section. This section will demonstrate the implementation of the ResNet-based global frame fish detector, as shown in Figure 3.8.

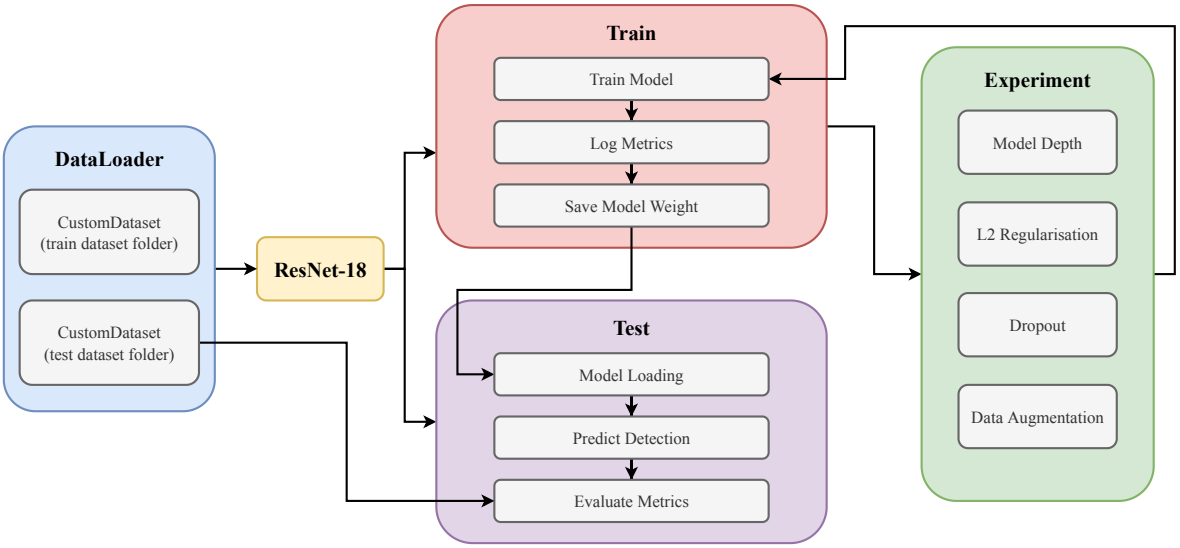


Figure 3.8: **Global Detection Implementation**: This flow chart shows the whole process of the global fish detector implemented in this project based on ResNet-18, and lists the relevant experiments.

Loading Data

The whole frame detection dataset will be used when training the global fish detector. As described in Section 3.3, each class has its own folder. Therefore, I created a **CustomDataset** object for encapsulating the dataset. The **CustomDataset** is the implementation shown in the Listing 3.5. This dataset object contains a function that reads images from the data folder, labels them based on their located data folder, and saves the data in the attributes of the object. Next, I use the **DataLoader** function provided by PyTorch to pass in a **CustomDataset** object that has already been initialised. During the training process, **DataLoader** can split the dataset into batches and feed them into the model batch by batch and loaded image data and labels directly from the initialised **CustomDataset** object.

```
class CustomDataset(Dataset):
    def __init__(self, path):
        self.images = []
        self.labels = []
        self.path = path
        self.transforms = transforms.Compose([
            transforms.ToTensor(),
            transforms.Resize((640, 640))
        ])

    def __len__(self):
        return len(self.images)

    def __getitem__(self, index):
        path = self.images[index]
```

```
        image = Image.open(path)
        label = self.labels[index]
        if self.transforms:
            image = self.transforms(image)
        return image, label

    def initDataset(self):
        for file in os.listdir(self.path + '/hasFish'):
            self.images.append(self.path + '/hasFish/' + file)
            self.labels.append(1)
        for file in os.listdir(self.path + '/notHasFish'):
            self.images.append(self.path + '/notHasFish/' + file)
            self.labels.append(0)
        return
```

Code Listing 3.5: **Custom Dataset Class**: This is a custom dataset class for loading image the dataset under the given path. The class stores the image files under the specified path in **self.images** and the corresponding labels in **self.labels**, with 0 for images with fish and 1 for images without fish. Additionally, the **transforms.Compose()** function defines a set of image transform operations, including converting the image to a tensor and resizing it into 640×640.

Pre-train

In this project, a global detector for fish will be implemented based on an 18-layer variant of the ResNet neural network architecture, called ResNet-18. I load this model architecture and the pre-train model weight from the **torchvision.model** library from PyTorch. The model weights provided by PyTorch are pre-trained on the ImageNet dataset which is a publicly available image classification dataset containing 1.4 million images in approximately 1000 categories [46]. I used this model weight as a starting point to fine-tune the ResNet-18 model on the global fish detection task.

Hyperparameter

The main hyperparameters involved in this model are the loss function, optimiser, L2 regularisation and dropout rate. The model uses cross-entropy as the loss function and an SGD optimiser with a learning rate of 0.0001 and a momentum of 0.9. Furthermore, in order to allow the model to learn more robust features, the dropout layer with a 0.5 dropout rate was added before the output fully connected layer of the model. The model was prevented from overfitting by setting the weight decay to 0.0001 in the SGD optimiser to use L2 regularisation.

Train

The model will be trained according to a given number of training epochs. First, during the training phase, the model will extract features from the input image and predict the labels. Then, the loss function will be used to calculate the loss between the predicted labels and the true labels. The optimiser will update the weights of the model in the opposite direction based on the calculated loss. When the training phase is complete, the model will be tested on the test dataset, but the model weights will not be updated. In the two phases above, the model and data are sent to the GPU by using CUDA to enable accelerated computation. Over the whole training process, the loss of the models and their accuracy on the training and test datasets for each epoch is saved to plot the learning curve, which shows how the models performed during training. In addition, the model weight that has the best performance on the test dataset is saved for later evaluation.

Model Result

The trained model is able to take an image of a penguin diving and generate an output list of length two. This output list contains the probabilities of the existence and non-existence of fish in the image. These probabilities are used to determine whether there is a fish in the image. If the image is determined to have fish in it, the image is annotated. Figure 3.9 shows an example of the output of this model.

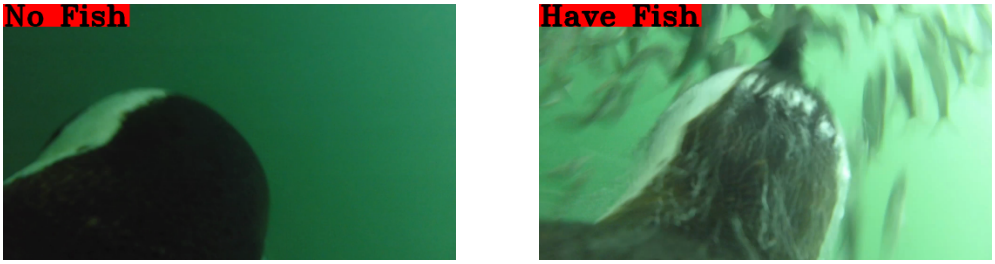


Figure 3.9: **The output of fine-tuned ResNet-18**: This is an example of the output of a global fish detector that will determine whether the image has fish or no fish

3.4.3 Video Detection

With the two detectors above, the location of the penguin in the image and the presence of fish in the image can be obtained. They are both based on image training. Therefore, to achieve the detection in video, the video is converted into a sequence of images. I used the OpenCV library, which was described in Section 2.12, to implement this part. First, use the **VideoCapture** class in OpenCV and pass the video path into it. Then, I use its built-in reading function that can convert the video into a sequence of frames and loop through each frame to get the corresponding image. Once all the images have been acquired, they can be passed into the model one by one for inspection and the labelled images can be saved. Figure 3.10 shows this procedure. For visualisation of the overall result, the labelled images can be combined into a video with a specified frame rate using the write function in the **VideoWriter** class of OpenCV, in this project, the videos used are all at FPS 30.

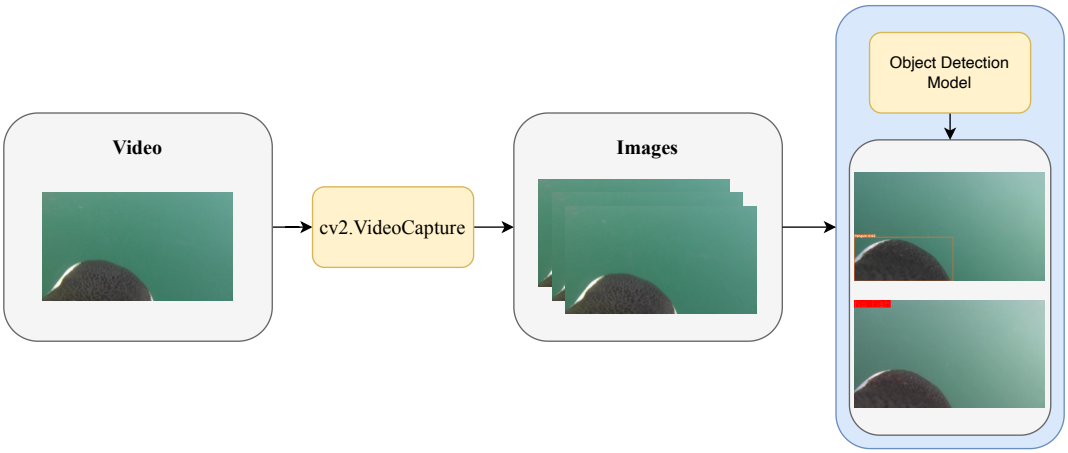


Figure 3.10: **Video detection impelmentation**: The flow chart for the process of video detection implementation.

3.5 Temporal Stream

This section describes the process of realising the temporal stream in the dual-stream network structure used. It focuses on how to implement the calculation of dense optical flow between all consecutive frames in a penguin dive video.

3.5.1 Dense Optical Flow

For the purposes of capturing the movement of the penguins in the video, I used dense optical flow, introduced in section 2.9, to describe the temporal movement information of each pixel in the video. The entire process of calculating the dense optical flow is illustrated in Listing 3.6 in the format of Python code. The procedure can be divided into two sections, the first is the extraction of frames from the video and the second one is the calculation of pixel changes between two consecutive frames. I used the OpenCV library to implement the whole process, which is mentioned in section 2.12.

When capturing frames from video, I use the **videoCapture** class of OpenCV to automatically fetch the data for each frame in the video by passing the path of the video. After that, the **imwrite** function was used to save each frame as an image.

After all the image frames have been captured, the optical flow image between each two frames can be calculated. The two neighbouring frames are read and greyscaled, which used the **imread** and **cvtColor** functions from the OpenCV library respectively. The reason for greyscaling is that the original RGB is affected by lighting, but the greyscale image is a single-channel image, which only contains brightness information. Thus, it is possible to reduce the effects caused by lighting in the optical flow calculation. Afterwards, the dense optical flow between the two frames is calculated by using the **optflow.createOptFlowDualTVL1.calc** method from the OpenCV library, which returns a two-dimensional array, where each element represents the movement vector from a pixel point in the previous frame to the corresponding pixel point in the next frame. The horizontal and vertical components of the movement vector for each pixel point can be extracted from the resulting two-dimensional array and normalised to a range of pixel values (from 0 to 255) by using **cv2.normalise** function. Once the calculation is completed, it is saved using **imwrite** function in OpenCV, which gives the respective optical flow images between the two continuous frames in the horizontal and vertical directions. Figure 3.11 shows an example of produced result optical flow image.

In optical flow calculations, the motion that occurs on an image can often be decomposed into motion in the horizontal and vertical directions. For example, in an image of the feeding movements of a penguin from bottom to top in a video, the motion can be decomposed into an upward vertical motion and almost no horizontal motion. Thus, by decomposing the optical flow into horizontal and vertical components, it is possible that the images of the optical flow they produce are relatively independent and do not affect each other and also to better extract the motion of the object in the video.

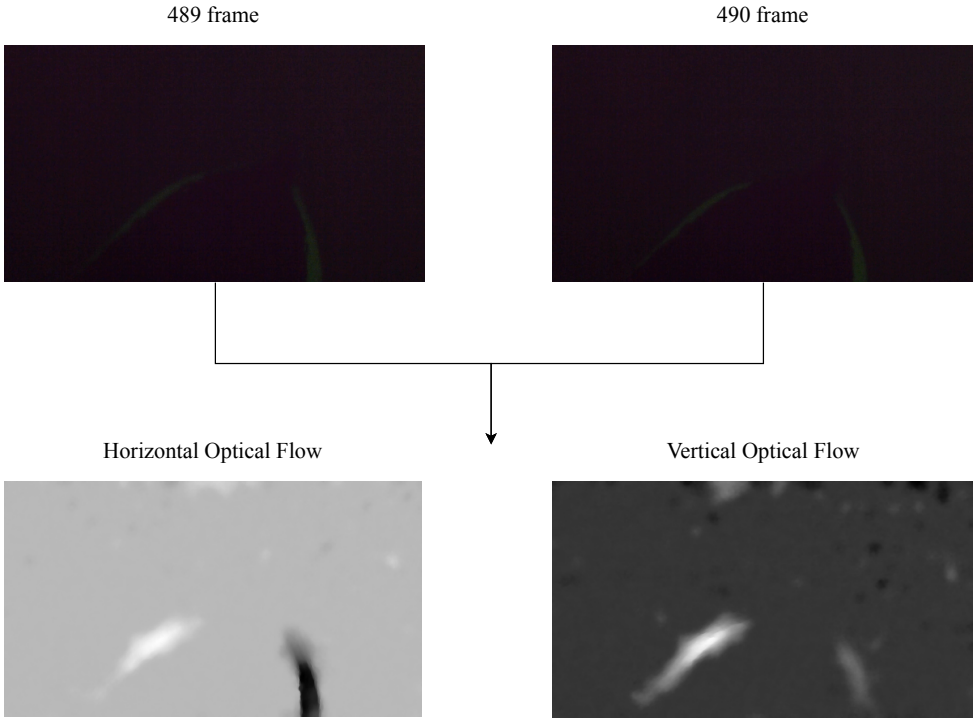


Figure 3.11: **Example of optical flow image**: This shows an example of dense optical flow calculation in two consecutive frames to produce horizontal and vertical optical flow images.

```
# Capture frame from the video
cap = cv2.VideoCapture(video_path) # Loading video
frame_count = 0
while True:
    ret, frame = cap.read()
    if ret:
```

```
        img_path = os.path.join(ori_img, f'{frame_count:04d}.jpg')
        cv2.imwrite(img_path, frame) # save each frame
        frame_count += 1
    else:
        break
cap.release()
# Loading the video frame
frames_path = ori_img
frames = os.listdir(frames_path + '/') # get the
frames.sort() # sort by the frame number

# Calculation of dense optical flow
for f in tqdm.tqdm(range(0, len(frames))):
    # Loading two neighbour frame
    frame_1_path = frames_path + "/" + frames[f]
    frame_2_path = frames_path + "/" + frames[f+1]
    frame_1 = cv2.imread(frame_1_path)
    frame_2 = cv2.imread(frame_2_path)
    # Grayscale the image
    prev_frame = cv2.cvtColor(frame_1, cv2.COLOR_BGR2GRAY)
    next_frame = cv2.cvtColor(frame_2, cv2.COLOR_BGR2GRAY)
    # Compute and normalise TV-L1 optical flow
    dtvl1 = cv2.optflow.createOptFlow_DualTVL1()
    flow = dtvl1.calc(prev_frame, next_frame, None)
    horz = cv2.normalize(flow[..., 0], None, 0, 255, cv2.NORM_MINMAX)
    vert = cv2.normalize(flow[..., 1], None, 0, 255, cv2.NORM_MINMAX)
    horz = horz.astype("uint8")
    vert = vert.astype("uint8")
    # Save optical flow images
    horizontal_frame_path = video_folder_path + '/horizontal_flow'
    vertical_frame_path = video_folder_path + '/vertical_flow'
    cv2.imwrite(f"{horizontal_frame_path}/frame_{f}.jpg", horz)
    cv2.imwrite(f"{vertical_frame_path}/frame_{f}.jpg", vert)
    # Pad final image, because the final image do not have optical flow
    if(f+1 >= len(frames)-1):
        cv2.imwrite(f"{horizontal_frame_path}/frame_{f + 1}.jpg", horz)
        cv2.imwrite(f"{vertical_frame_path}/frame_{f + 1}.jpg", vert)
        break
```

Code Listing 3.6: **Calculation of Optical flow**: This code shows the whole process of calculating the dense optical flow in a video. It can be roughly divided into two parts. The first part is to capture the frames in the video and the second part is to calculate the optical flow between the two frames on the captured frames. This code is inspired by [38].

3.6 LSTM Behaviour Detector

The behaviour detector is the last part of the overall project implementation diagram in Figure 3.5, which will accept information on the spatial and temporal streams to output whether the behaviour of penguin is predation or not. Since determing whether the current frame is a predation event requires observing the movement of the penguins before and after it. Therefore, I used an LSTM model to handle this time series task, which is described in Section 2.5.1. The structure of the entire LSTM behaviour detector implementation is shown in Figure 3.12.

3.6.1 Loading Data

The multi-frame detection dataset described in Section 3.2.3 will be used when training the LSTM behaviour detection model. The entire dataset is encapsulated as a object and using **DataLoader** function of PyTorch to import the data into the model at training process. Figure 3.12 contains the

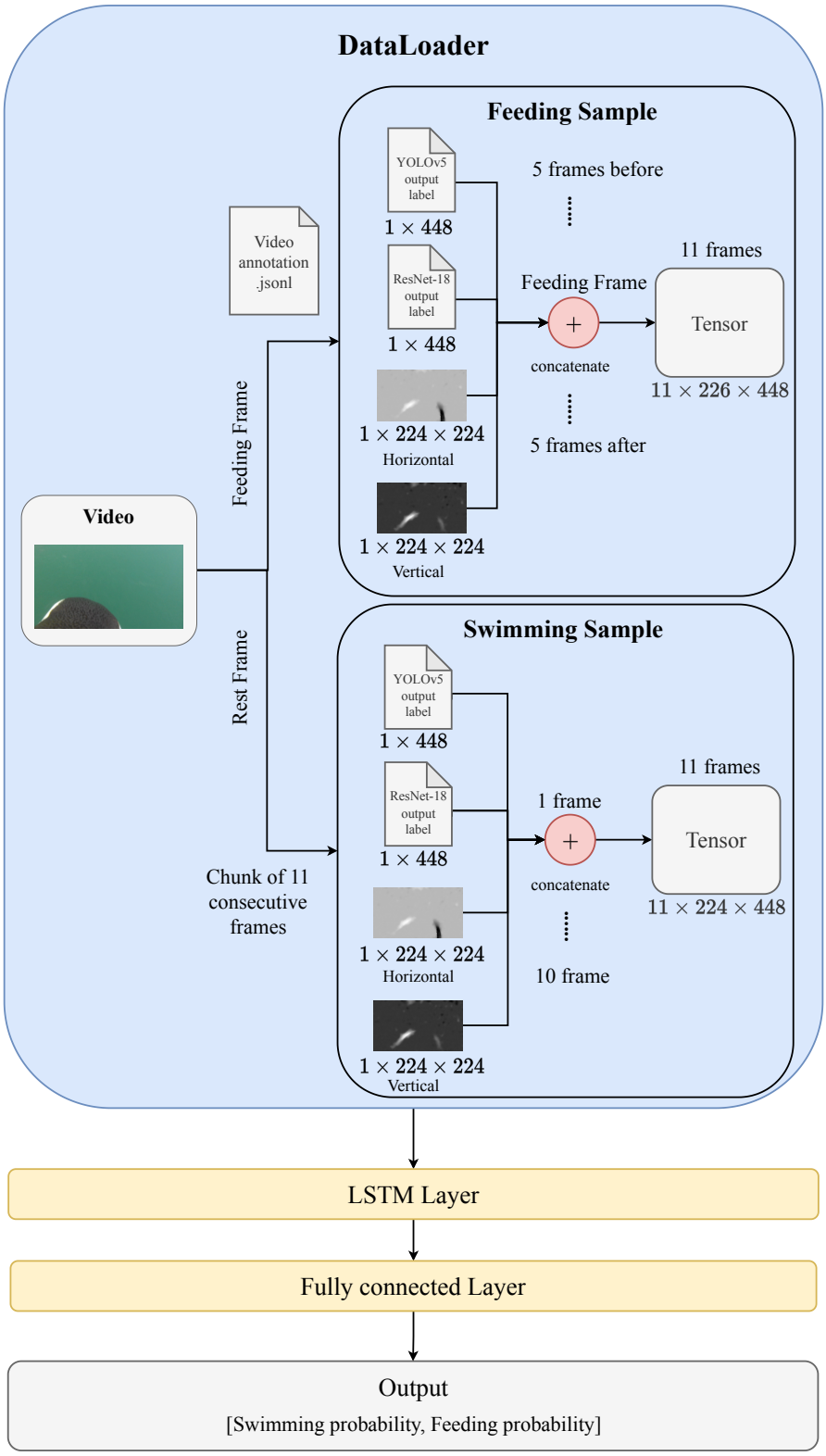


Figure 3.12: **The implementation of LSTM behaviour detector**: There are three component in the whole implementation. The first one is DataLoader, which can take the feeding sample based on the video annotation JSON file and the swimming sample from the rest of the video. Each sample contains the output from the spatial stream and temporal stream to make an 11-frame time series. After that, the sample will be input to the LSTM layer and Linear layer to output a 1×2 sequence that contains the probability of feeding and swimming.

process of extracting samples from the video. Firstly, the predation event frames in the video are extracted from the video based on the video annotation file in the multi-frame detection dataset. After that, the horizontal and vertical optical flow images of the predation event frame from the temporal streams and the spatial detection results of the frame from the YOLOv5 and ResNet detector model are transformed into a tensor. These tesor are concatenated together to form the data of the predation event frame, whose dimention is $1 \times 226 \times 448$. The same operation is performed on the five frames before and after this predation event frame to form a complete sequence data of predation events. Then, these 11 tensor are concatenated together based on their position in the video to result an $11 \times 226 \times 448$ tensor, which form a predation sample. For the non-predation sample acquisition, the frames already inccluded in the predation sample are removed from the frames of the videl, the the remaining frames are formed into a chunk with every 11 consecutive frames and same operation for predation event sample is performed to obtain an $11 \times 226 \times 448$ tensor. Finally, by applying the above operation to all the annotated videos, the training set and test set for this model can be generated.

3.6.2 Hyperparameter

In this model, I use cross entropy as the loss function to calculate the loss during the training process and use the SGD optimiser to update the weights of the model based on the loss, these two hyperparameters are introduced in Section 2.7. For the hyperparameters inside the LSTM model, the input size is set to 101248, which is equal to the amount of data in one sample, the hidden size is set to 256 and the number of layers is 2. For the last fully connected linear layer, the output size is set to 2, which will represent the probability of two categories in the dataset.

3.6.3 Train

As described in Section 3.2.3, only 188 predation events were annotated in the entire video dataset, which would imply a significant imbalance in the number of predation and non-predation event samples. This situation may lead to the model focusing excessively on the non-predation event samples and consequently reducing the accuracy of the model in classifying the predation event samples. Therefore, during the training process, the non-predation event sample will be random sampling. In a training epoch, the model will use all the predation event samples and the equivalent number of non-predation samples to learn and calculate the loss and update the weights of the model. After that, before the next training epoch starts, the predation event sample is kept unchanged. But the used non-predation event sample in the previous epoch is randomly replaced with other non-predation event samples with the same number of predation event samples for the next round. Repeat this operation until all epochs are completed. The advantage of this method is that the model always uses a balanced amount of data during training and does not focus heavily on the non-predation event samples. Moreover, during each round of model training, the resulting model weights are tested on a test set and the best model weights are saved.

3.6.4 Model Result

The behaviour detector can be imported as a video and cut the video into chunks of length 11. Each chunk will first pass through the spatial and temporal streams to obtain the optical flow image and spatial information of the chunk. The information obtained is input to a trained behaviour detection model for detection. The model outputs a list of length 2, which contains the probability of the penguin swimming and feeding in the chunk. Figure 3 shows the results of the model for a section of the chunk.

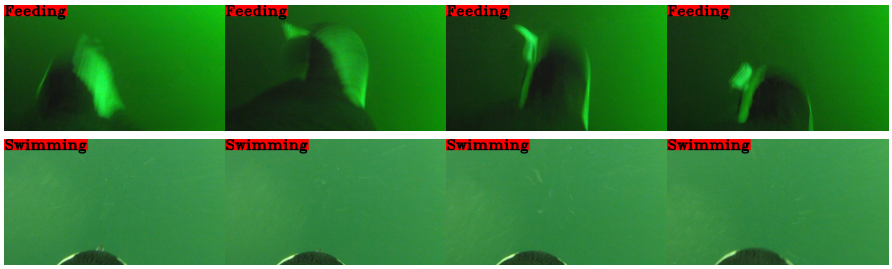


Figure 3.13: **Example result from LSTM detector**: This figure shows an example of the model correctly detecting feeding and non-feeding events.

Chapter 4

Results & Evaluation

This chapter shows the results of local detection models, global fish detection models and feeding behaviour detection models that can be applied in penguin diving videos. It contains a number of implementations under multiple model configurations and explores and evaluates the motivation and impact on the results developed for each configuration. In addition, this chapter also summarises the process of discovering the best model for each of the models.

4.1 YOLOv5 Local Detector

As described in Section 3.4.1, local detection is used to detect fish and penguins that can be localised and classified in the video. In Figure 3.6, the implementation of the model undergoes a number of experiments to find the best detection results. The following sections describe each of these experiments and their effect on the model results.

4.1.1 Baseline Model

This section will illustrate the process of finding a baseline model and the related experiments. In the local detector implementation, I mainly compared two versions of YOLO models to select the model with the best performance as the baseline model.

YOLOv5 vs. YOLOv8

The first experiment was to find a suitable version of YOLO. For this project, two versions of YOLO were considered, YOLOv5 and YOLOv8. Both versions were developed by Ultralytics company. According to the official description of the YOLOv8, it was based on their previously developed YOLOv5 and adds new features such as image segmentation and poses estimation. Therefore, in order to choose a better model as a baseline, I trained the same depth of YOLOv5 and YOLOv8 on the local detection dataset. The dataset was described in Section 3.2.1 and the result of both trained models on the test dataset are shown in Table 4.1. The comparison reflects that YOLOv8 has a slightly smaller parameter size than YOLOv5, but the accuracy and average accuracy in each category of penguin and fish is slightly lower than YOLOv5. The reason for this is that YOLOv8 uses smaller parameters, which might affect the model to learn the data. Therefore, the local detection model used YOLOv5 as the baseline model.

	Penguin	Fish	Class Average	Parameters
YOLOv5s	98.1%	63.1%	80.6%	14MB
YOLOv8s	96.4%	62.7%	79.5%	11.2MB

Table 4.1: **Comprision of YOLOv5s and YOLOv8s**: This table contains the precision rates of each object type on the validation set and the model parameters size for YOLOv5 and YOLOv8.

Baseline result

Figure 4.3a shows the confusion matrix for the YOLOv5 baseline model. As can be seen from the figure, the penguins can essentially be classified correctly. However, for fish detection, the performance is not good. Although the proportion of fish being correctly classified is not bad, the background is often classified as fish. By visualising the results of the model in Figure 4.1, it can be seen that the water bubbles in the background are often labelled as fish. The reason for this is that the water bubble has visually similar characteristics to the fish in the underwater video, such as shape and colour et al.. Therefore, there needs a way to enable the model to better distinguish between fish and background water bubbles.

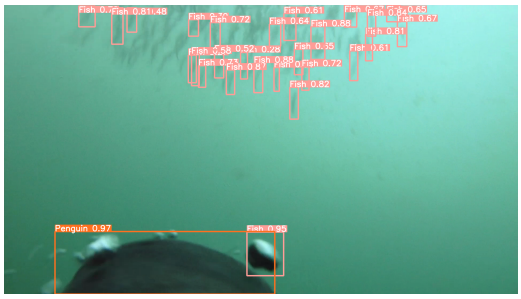


Figure 4.1: **The example of fish not being detected correctly**: This image shows the model detects the water bubble beside the penguin as fish.

4.1.2 Add Bubble Class

For solving the problem of confusing fish with water bubbles, adding the bubble class to the dataset allows the model to capture some features that distinguish water bubbles and fish. Table 4.2 show the effect on the results of the model with the addition of the water bubble class to the dataset.

The table shows an increase in the accuracy for fish, but a decrease in accuracy for penguin species and average class. In the results of the visualisation model in Figure 4.2, it was found that the model has the ability to distinguish a portion of the water bubble from the fish. However, this brought an insignificant increase because of the insufficient sample of bubbles in the dataset, which is shown in Figure 3.1 in Section 3.2.1. And the average class accuracy was reduced by the lower accuracy of the bubbles.

	Penguin	Fish	Bubble	Class Average
Penguin + Fish	98.1%	63.1%	0%	80.6%
Penguin + Fish + Bubble	97.4%	63.4%	52.2%	71%

Table 4.2: **Two classes versus three**: This table shows the result of the model only learning the fish and penguin features and the impact of adding the bubble class during training. An improvement in the accuracy of fish detection can be found with the addition of the bubble class.

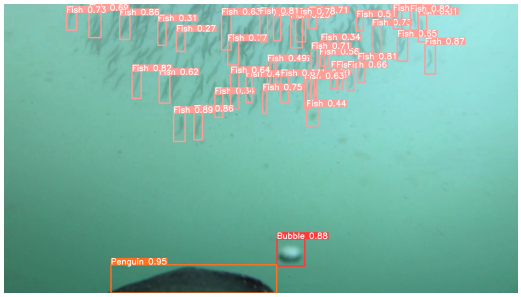


Figure 4.2: **The example of bubble detetion**: This image shows the model does not detect water bubbles as fish and missing detect some fish.

4.1.3 Data Augmentation

Data augmentation is a method to address the insufficient sample of data and add a new sample to make the model learn more features during the training, which is described in Section 2.8.3. In the local detection dataset, I used horizontal flipping, small angle rotation and the Gaussian noise to extend the number of samples. The reason for using these data augmentation techniques is to ensure that the augmented data is matched to the situation in the real dataset. I did two experiments on data augmentation. One was to apply data augmentation on all images of the entire dataset. The other one was to perform data augmentation only on images containing water bubbles. The results of these two experiments are presented in Table 4.3.

	Penguin	Fish	Bubble	Class Average
No Augmentation	97.4%	63.4%	52.2%	71%
Augmentation For All Class	96.1%	56.8%	45.9%	66.3%
Augmentation Only For Bubble	98.7%	65.6%	51.2%	71.8%

Table 4.3: **The effect of different data augmentations**: This table shows that the data augmentation only for the water bubble class can improve the overall accuracy of the detection model.e

The comparison shows that data augmentation for all images reduces the performance of the model and has an impact on the accuracy of all classes. Although data augmentation of all images doses increases the amount of data that can be learned by the model, because of the imbalance between the number of sample classes in the original dataset, augmenting all of the image data does not improve the situation and might amplify this situation. In contrast, applying data augmentation only to images that contain water bubbles allows the number of bubble samples to be increased. This would improve the situation where the original dataset had fewer samples of water bubbles than the other categories, as mentioned in Section 3.2.1, which could reduce the impact of the imbalance in sample size between categories on model performance and allow the model to capture more features of the water bubble category. From the results shown in Table 4.3, the experiments with data augmentation only on the bubble sample images showed better results and improved accuracy on fish.

4.1.4 Early Stop

For preventing overfitting of the model, I try to set up the early stop hyperparameter for the model training. By monitoring the performance of the model on the test dataset, when the performance starts to decline, the model stop training and saves the weight of the best-performing model. This improves the generalisation ability of the model and avoids learning more details about the training data. It was found that the performance of the model could be improved by using this approach and the results are shown in Table 4.4.

	Penguin	Fish	Bubble	Class Average
Run All Epoch	98.7%	65.6%	51.2%	71.8%
Early Stop	99%	70.2%	74.2%	81.1%

Table 4.4: **The effect of early stop**: This table shows the improvement of performance for the model with early stop training in the training process.

4.1.5 Final Model

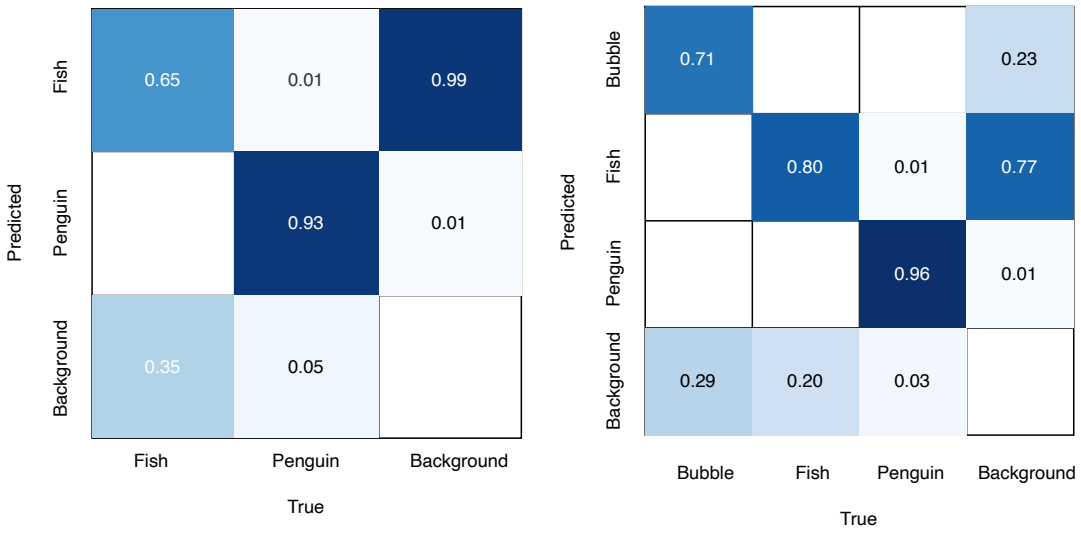
After the previous optimisation approach, the final model of the local detector has been generated. The overall performance of the final model on the test dataset of the local frame detection dataset is shown in Table 4.5 and Figure 4.3b. The table shows that the final model detects relatively better on all classes, with an average precision of 81.1%. In particular, the performance in the penguin class is excellent, with a detection precision of 99% and a recall rate of 95.2%. By comparing the confusion matrix of the final model and baseline model, the model has better performance of the fish detection after optimisation. The proportion of fish correctly identified increased from the previous 0.65 to 0.8 and reduced the proportion

of background identified as fish from 0.99 to 0.77. Thus, the final model is effective in fish detection, with 70% accuracy, but could be further improved.

As described in Section 2.9, each bounding box in YOLOv5 has a confidence score to indicate whether the model believes that the bounding box contains a target object. Therefore, I used the mAP50 and mAP50_95 metrics to evaluate the model performance under different confidence score thresholds. mAP50 means the confidence score threshold is under 0.5 and mAP50_95 means the score is between 0.5 and 0.95. By observing Table 4.5, for each class, the overall model performs better at lower confidence thresholds.

	Precision	Recall	mAP50	mAP50-95
ALL	81.1%	77.3%	81.1%	52.5%
Penguin	99%	95.2%	98%	83.8%
Fish	70.2%	70%	73.3%	32.2%
Bubble	74.2%	66.8%	72%	41.8%

Table 4.5: **Metrics of final YOLOv5 local detector**: This table shows the performance of the local detector model under different metrics. The calculation method for each metric is shown in Section 2.10



(a) Baseline Model Confusion Matrices

(b) Final Model Confusion Matrices

Figure 4.3: **The comparision of confusion matrix for YOLOv5 detector**:This confusion matrices shows the results of the YOLOv5 baseline model and final model. Each row is a predicted label and each column is a true label. Each cell represents the percentage of matches between the actual and predicted categories. By comparing these two confusion matrices, the final model has improved in the fish detection and penguin detection. And the false detecion between fish and background is reduced.

4.1.6 Qualitative Analysis

Section 2.14 describes a study from Mingyu, who implemented an underwater video penguin detector based on YOLOv5. Figure 4.4 shows comparison between the final model described above and the fine-tune model from Mingyu. By comparing the detection result in the same frame. the model from Mingyu detects other objects in the video also as penguins. However, the penguin detector in this project makes the model to better distinguish between penguins and other objects in the video by adding fish and bubble classes during model training. This reduces some false positives detection and imporves the accuracy of penguin detection in underwater videos.

The best model from Mingyu Final model

Figure 4.4: **Comparison of results with Mingyu**: The left side of this image shows the detection results based on the research conducted by Mingyu, and the right side shows the detection results from the final model of this project. It can be seen that both models have the ability to detect the location of the penguins, but the model from this project can better distinguish between the penguins and other objects in the video.

4.2 ResNet-18 Global Detector

The global detection model is used to determine the existence of fish in a video frame and it is a binary model. The detailed implementation of this model is shown in Section 3.4.2. In the implementation process, Figure 3.8 shows some relevant experiments that aimed to find the optimal model. The results of each experiment and their impact on the model results are evaluated in the following sections.

4.2.1 Baseline Model

The following sections will describe some experiments that were conducted to find the baseline model for global fish detectors. The experiments are about the choice of the model architecture, optimiser and learning rate.

ResNet-18 vs. ResNet-50

ResNet is a special type of deep convolutional neural network, which uses residual learning. In this experiment, ResNet18 and ResNet50 will be used as experimental subjects. The reason for choosing these two architectures is that they have different network depths and the ResNet50 uses a bottleneck-structured building block, which is described in Section 2.4.1. The experimental results shown in Table 4.6 indicate that ResNet-50 did not obtain better results than ResNet-18 by increasing the number of layers and using deeper residual block, with 86.9% and 89% average accuracy respectively. Theoretically, ResNet-50 has deeper depth and larger parameters, which allows the model to learn more informative and complex features. However, because of the limited sample volume of the project dataset introduced in Section 3.2.2, the ResNet-50 are unable to take advantage of the depth to learn more effective features. Therefore, the model architecture of ResNet-18 was used as the baseline model for the global fish detector.

	Has Fish Accuracy	No Fish Accuracy	Average Accuracy
ResNet-18	95.1%	82.9%	89%
ResNet-50	82.8%	91.1%	86.9%

Table 4.6: **The comparison of ResNet-18 and ResNet-50**: ResNet has a higher accuracy on having fish class and average accuracy than ResNet-50.

Transfer Learning

The transfer learning machine learning methods using pre-trained models are considered for the training of models for global fish detectors. As the pre-trained model has already learned the general features of the image on a large dataset, transfer learning can be used for faster convergence and better generalisation on new datasets than training a model from scratch. The experiments were conducted using the pre-trained model weights provided by PyTorch. There are two implementations for transfer learning in this experiment. Feature-based transfer learning involves freezing all pre-trained model weights when training the models on a new dataset and only updating the weights of the final fully connected output layer. In contrast, parameter-based transfer learning uses the pre-trained model weights as a starting point to fine-tune the weights of the model when training on a new dataset. The experimental results in Table 4.7 show that parameter-based transfer learning has higher accuracy for each class and higher average accuracy than feature-based transfer learning. The reason for this is that feature-based transfer learning can only apply the learned features from the pre-trained model to the new dataset. But, these features cannot be further fine-tuned to generate more applicable features for the new dataset. Therefore, the parameter-based transfer learning approach will be used in the baseline model.

	Has Fish Accuracy	No Fish Accuracy	Average Accuracy
No pre-train	82.05%	85.25%	83.68%
Fine-tune	95.1%	82.9%	89%
Feature extraction	88.5%	80.5%	84.5%

Table 4.7: **The comparison of different types of transfer learning**: The results of both models using pre-training have higher average accuracy than the models without pre-training. And the model trained by the fine-tuning whole model with the pre-train method gives the best results.

Optimisers

The optimiser generally implies how the model updates the internal weights based on the loss, so it is crucial to choose a suitable optimiser that can be applied to the global fish detector. The three optimisers presented in Section 2.7.2 were compared in this experiment and their training results are shown in Table 4.8. The result shows that the model using the basic SGD optimiser has the poorest performance than other optimisers, with an average classification of 89%. Whereas, the model using the SGD with momentum performed has the best performance, with an average classification of 90.6%. As described in Section 2.7.2, the momentum can accelerate the gradient descent along the direction of the gradient descent, which can reduce oscillations in the SGD optimiser update weight process. Because of

this characteristic, SGD with momentum reduces the risk of being stuck in the local optimal solution and allows faster convergence to the global optimal solution. In comparison, Adam optimiser has a relatively worse performance, with an average accuracy of 90.2%. Based on the research from Wilson et al. [56], their experiments show that SGD is better than Adam, especially for CNN. Therefore, based on the above experiment, SGD with momentum is more appropriate for the image classification task of this project.

	Has Fish Accuracy	No Fish Accuracy	Average Accuracy
SGD	95.1%	82.9%	89%
SGD + Momentum with 0.9	90.2%	91.1%	90.6%
Adam	88.5%	91.9%	90.2%

Table 4.8: **The comparison of different optimisers**: The model using SGD optimiser had the highest accuracy on the category with fish. And the optimiser with SGD with momentum was the best in terms of average accuracy.

Learning Rate

The learning rate is an important hyperparameter in training an image classification model, which controls the size of the steps during the optimiser updating the model weight in each round of training. In determining the baseline model, there is an experiment with different learning rates to find an optimal learning rate. Table 4.9 shows the performance of the model at different learning rates, which shows that the model does not develop a regular variation according to the decreasing learning rate. For comparison accuracy, the accuracy of both fish and non-fish classification is closer when using a learning rate of 0.0001, and the average accuracy is at the highest value. The model with a learning rate of 0.01 has a similar performance on average accuracy, but there is a risk of overfitting because of the large step size for updating the weights. Therefore a learning rate of 0.0001 is the optimal choice for the global fish detector of this project.

	Has Fish Accuracy	No Fish Accuracy	Average Accuracy
with 0.01	87.7%	93.5%	90.6%
with 0.001	90.9%	87.8%	89.4%
with 0.0001	90.2%	91.1%	90.65%

Table 4.9: **The comparison of different learning rates**: The models when the learning rate was 0.0001 had a better overall performance.

ResNet Fish detector baseline model	
Architecture	ResNet-18
Pre-trained	Yes
Transfer learning	Fine-tune whole model
Optimiser	SGD + Momentum
Loss	Cross Entropy
Learning Rate	0.0001
Baseline model result on test set	
Has Fish Accuracy	No Fish Accuracy	Average Accuracy
90.2%	91.1%	90.6%

Table 4.10: **The hyperparameter and result of the baseline model**: This table shows the finalised hyperparameter configuration of the baseline model after a series of experiments. The baseline model had an average accuracy of 90.6%.

Baseline Model Result

After the above experiments to find the baseline model, the information on the identified baseline model and the results on the test set are presented in Table 4.10. The final baseline model will use the pre-training weights for transfer learning and fine-tuning the whole model parameters. Losses will be calculated using cross-entropy during training and the weights will be updated using an SGD optimiser with momentum and a learning rate of 0.0001. By observing the performance of the baseline model on the test set, it has achieved an average accuracy of 90.6%, an accuracy of 90.2% for the having fish class and 91.1% for the no fish class.

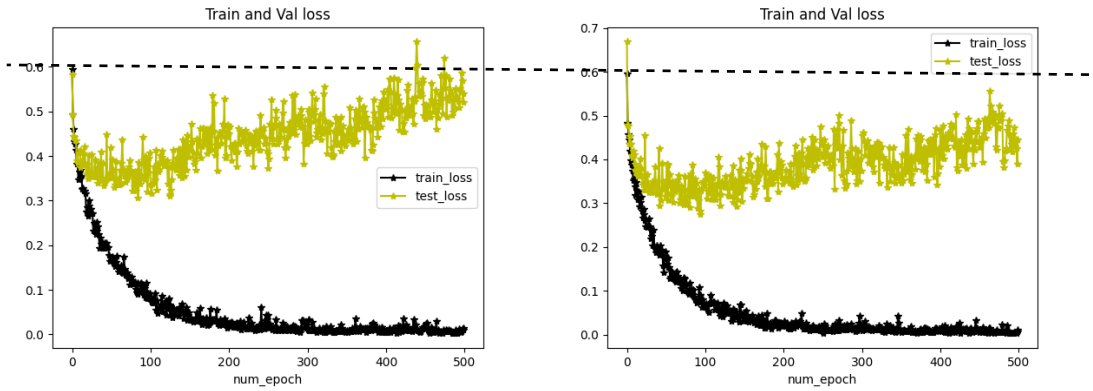
4.2.2 L2 Regularisation

The accuracy of the baseline model is not bad. However, by observing its loss curve on the training and test sets in Figure 4.5, it can be seen that there is overfitting, which means that it performs better on the training set than on the test set, and the gap between the two performances is gradually getting larger. L2 regularisation, introduced in Section 2.8.2, is a method to reduce overfitting. It controls the strength of the regularisation by using a regularisation factor. Experimental results for different regularisation factors are shown in Table 4.11.

	Have Fish Accuracy	No Fish Accuracy	Average Accuracy
Baseline	90.2%	91.1%	90.65%
L2 with 0.01	88.2%	93.3%	90.75%
L2 with 0.001	88.5%	92.7%	90.6%
L2 with 0.0001	91.8%	90.2%	91%

Table 4.11: **Comparison of different regularisation parameters**: Using L2 regularisation can increase the accuracy of model. The best results are obtained when the regularisation parameter is 0.0001.

As can be seen in the table, L2 regularisation can improve the average classification accuracy of the model. The most significant improvement in average accuracy is seen when the L2 regularisation factor is 0.0001, and the improvement in classification accuracy is obvious for having fish. By comparing the loss curve of the baseline model with the loss curve of the model using the L2 regularisation with a regularisation factor of 0.0001 in Figure 4.5, it can be proved that L2 successfully slowed down the growth of the loss curve on the validation set and the trend of the curve became flat.



Baseline Mode Loss Curve

Baseline + L2 Loss Curve

Figure 4.5: **Loss curves of the model with L2 regularisation**: By comparing the loss curves of the baseline model and the model with L2, the loss on the test set was successfully reduced.

4.2.3 Dropout Layer

For the purposes of solving the problem of overfitting and improving the generalization of the model, I tried to add Dropout layers into the model. As described in Section 2, Dropout prevents the model to rely

excessively on specific neurons by dropping a random portion of neurons and their connections during training. This approach helps the model to learn less specific features of the training set and improves the ability of the model to generalise to new data. Table 4.12 shows the results of three experiments with different probabilities of random dropout in the Dropout layer.

	Have Fish Accuracy	No Fish Accuracy	Average Accuracy
Only L2	91.8%	90.2%	91%
L2 with Dropout 0.2	88.5%	92.7%	90.6%
L2 with Dropout 0.5	88.5%	94.3%	91.4%
L2 with Dropout 0.8	93.5%	85.2%	89.4%

Table 4.12: **Experimental results with different dropout probabilities**: the model has the best results with a dropout rate of 0.5, but it reduces the accuracy of the model on having fish class.

In the tables, it is observed that using a dropout rate of 0.2 and 0.8 reduces the average accuracy of the model, whereas using a dropout rate of 0.5 combined with L2 regularisation improves the accuracy of the model relative to using only L2. However, there is a decrease in the classification accuracy for having fish. As determining the existence of fish in an image is an important condition for subsequent behavioural detection models to determine penguin predation events, applying a dropout rate of 0.5 might add more noise to the subsequent model. Hence, adding a Dropout layer is not effective optimisation in this project.

4.2.4 Data Augmentation

Adding more samples to the training dataset allows the model to learn more features. Data augmentation, described in Section 2.8.3, is a method of adding new samples to a dataset by transforming the original images to generate new samples that match the real situation in the dataset. This is an effective method to expand the dataset. In this section of the experiment, I compared two different combinations of data augmentation techniques, the results of them are shown in Table 4.13.

	Have Fish Accuracy	No Fish Accuracy	Average Accuracy
No Augmentation	91.8%	90.2%	91%
Noise + Flip	91.31%	78.69%	85.36%
Noise + Flip + Rotate	93.16%	82.97%	87.87%

Table 4.13: **Different combinations of data augmentation**: The table reflects that using more data augmentation techniques can lead to better accuracy for the model. However, on the dataset of this project, data augmentation did not bring an improvement to the model.

Compare two combination of data augmentation, the results show that using noise and horizontal flipping with small angle rotation gives better training results than only using noise and horizontal flipping. The reason for this is that adding the small angle rotation in data augmentation process can further increase the diversity of the data, which can prevents the model to rely on the feature at specific location and angle. However, the advantages from data augmentation did not perform in this project. Compared to the models trained without data augmentation, although the models trained with data augmentation showed a significant improvement in accuracy on the having fish class, the accuracy of the average classification was lower, especially in the no fish category. This means that many images without fish were misclassified as having fish, which could potentially add noise to the subsequent predation detection model. In order to prevent this, data augmentation will not be applied on the training process of global fish detection model.

4.2.5 Final model

After the optimisation process described above, the final global fish detector was produced. In Table 4.14, the best results of all optimisation processes are shown and illustrate which ones will be used. Although the combination of the Dropout layer increased the average accuracy of the model, it significantly reduced the accuracy of the model in the category with having fish. In addition, when data augmentation was

used, the average accuracy was significantly reduced but the accuracy in the having fish category was increased. Therefore, in order to maximise the accuracy of each of the two classes and reduce the impact of misclassification on the subsequent model, the final model is composed of the baseline model with L2 regularisation added.

Applied	Optimisation	Have Fish Accuracy	No Fish Accuracy	Average Accuracy
	Baseline	90.2%	91.1%	90.65%
	Baseline + L2 0.0001 (Final Model)	91.8%	90.2%	91%
x	Baseline + L2 0.0001 + Dropout 0.5	88.5%	94.3%	91.4%
x	Baseline + L2 0.0001 + Noise + Flip + Rotate	93.16%	82.97%	87.87%

Table 4.14: **Summary of the final model**: This table shows those optimisations that were not applied in the final model.

4.3 LSTM Behaviour Detector

The LSTM behavioural detection model is used to determine whether the penguin is swimming or predating in the underwater video. It accepts optical flow images from the temporal stream and spatial information from the spatial stream, as described in Section 3.6. In the following section, the effect of the hyperparameters in the LSTM on the model is analysed and compared the final result with the I3D model that is described in Section 2.15.

4.3.1 Selection of LSTM Hyperparameters

In the LSTM model, I have considered two main hyperparameters in LSTM model, which are the number of layers of the LSTM and the hidden side of the LSTM. The number of layers controls the complexity of the model, and the hidden size controls the output dimension size of the memory cells in the LSTM, which has an impact on the expressiveness and learning ability of the model. Therefore, the following experiments will be conducted to find the best of these two parameters.

	Feeding Accuracy	Swimming Accuracy	Average Accuracy
One LSTM layer with 512 hidden size	76%	56%	66%
Two LSTM layers with 512 hidden size	78%	78%	78%
Two LSTM layers with 256 hidden size	74%	92%	83%
Two LSTM layers with 128 hidden size	86%	64%	75%

Table 4.15: **Different combination of LSTM hyperparameters**: The experiment of different combinations reflects that using two-layer LSTM and a 256 hidden output size can achieve best result.

Table 4.15 shows that the use of the different LSTM layers and hidden layer sizes have an impact on the recognition accuracy of different penguin behaviours. For feeding and swimming behaviours, the accuracy of the model using a two-layer LSTM is higher than using a one-layer LSTM. The size of the hidden layer does not have the same effect on the two behaviours. When the output size of the hidden layer is 128, the accuracy of predation events is the highest. And when the hidden layer output size was 256, the highest accuracy was achieved in determining penguin swimming and the highest average accuracy was also obtained. The reason for this is that the behavioural patterns of penguins during predation are relatively simple and do not require much detail or features to be captured. In contrast, the behaviour of penguins when swimming might have more variation, such as turning, accelerating, and so on. Therefore, it requires more information to make an accurate classification. In summary, I chose the two LSTM layers and a hidden output size of 256 that had the highest average accuracy.

4.3.2 Qualitative Analysis

Based on the experiments in Section 4.3.1, the final LSTM model had an accuracy of 74% on predation events, and 92% on non-predation events. By visualising the prediction of the model, we could better understand the decisions of the model. Figure 4.6 shows an example of the model that was correctly

classified and two examples that were not correctly classified. The example that was not correctly classified shows that the entire model does not learn enough specific features for predation events. For example, when the penguin turns in a situation where there are fish, the model considers this to be a predation event. In addition, when the penguins dive rapidly, the model similarly identified it as a predation event. These misclassified examples show that it is difficult for the model to capture the specific feature of the feeding behaviour because of the high variability of non-feeding events. Therefore, the accuracy of this model is not enough for direct use in actual ecological studies. The analysis in this section gives direction for further improvement of the model, which will be described in Section 5.1.

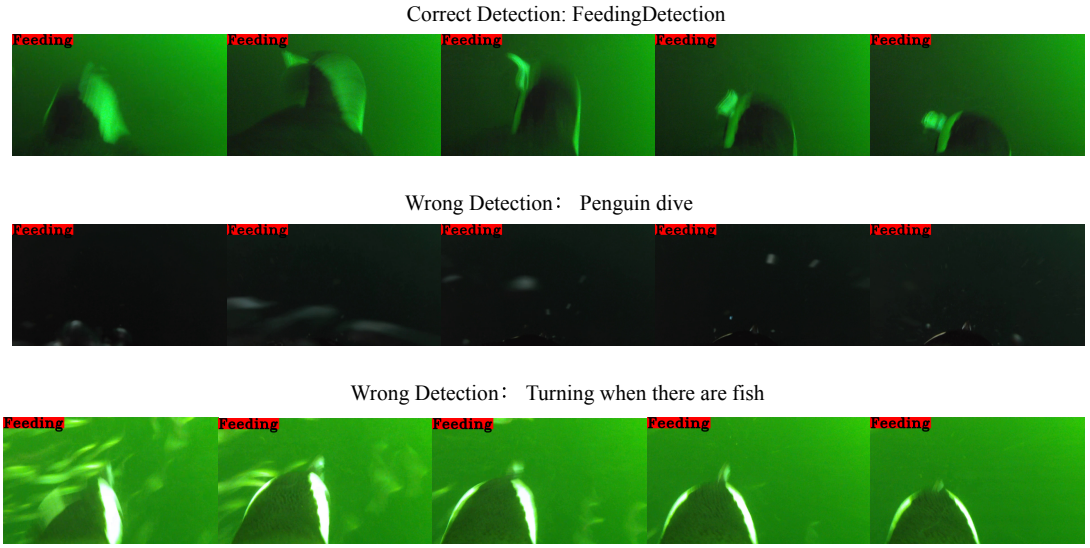


Figure 4.6: **Example of behaviour detection**: These three examples show that the model is not very good at distinguishing predation events from non-predation events, especially when the penguin have large range of motion.

4.3.3 Compare the LSTM-based Dual-Stream Network with I3D

I3D is a 3D convolutional neural network model using a two-stream network structure, which is described in Section 2.15. Compared to the behavioural detection model in this project, the two models are similar in that they process the temporal and spatial information in the video separately by using a two-stream network structure. But the difference is that I3D uses two 3D convolutional neural networks to extract spatial and temporal information. The behavioural detection model in this project is based on YOLOv5 and ResNet18 to extract spatial information and capture temporal information using LSTM.

The I3D model was loaded from the GluonCV library [14] and trained with the LSTM behaviour detection model on the same training dataset as the LSTM behaviour detection model by using the Python training script from the GluonCV library. Comparing the performance of the two models on the dataset for this project, it can be seen that in Table 4.16, I3D may not be applicable in the behavioural recognition task of penguin predation, with an average accuracy of only 58.55% on its test set. In contrast, the LSTM behavioural detection model performed better with an average accuracy of 79% and an accuracy of 80% on predation event detection. This difference is explained by the pre-trained YOLOv5 penguin detection model and the ResNet-18 fish detection model that better captures the spatial information of objects in the penguin diving video. In addition, the LSTM behavioural detection model combines spatial information and temporal information from the optical flow video into the LSTM, which has performed better on the temporal task, as described in Section 2.5.1. However, I3D only uses convolutional operations, which might not be able to capture this information properly.

But there are some limitations to this experiment that need to be noted. I used the existing training code for I3D directly, and used the parameters suggested by GluonCV, which means I did not to fine-tune the parameters to find the best model. Therefore, this might be an unfair experiment. In addition, the dataset for this project was designed for a two-stream model, and the length of each short video does not meet the recommendation of GluonCV to contain more than 32 frames, so this might not be a good demonstration of the ability of I3D on penguin predation events.

	Feeding Accuracy	Swimming Accuracy	Average Accuracy
I3D Network	55.2%	61.9%	58.55%
Dual Stream Network	74%	92%	83%

Table 4.16: **Comparison of the dual stream model and the I3D model**: The dual stream model performs better in detecting both predation and non-predation events.

In summary, based on the performance of the model on this project dataset, the spatial information provided by YOLOv5 and ResNet-18, combined with the temporal information of LSTM, might can better identify penguin predation behaviour in this project dataset. However, it is important to note that the accuracy of the LSTM model is dependent on the accuracy of the YOLOv5 and ResNet-18 models. If the accuracy of the YOLOv5 and ResNet-18 models is not good, it might bring some noise to the last LSTM layer in the model.

Chapter 5

Conclusion

This project is based on a deep learning approach to detect African penguins and their predation events in underwater videos. First, three datasets were created from the provided video dataset of African penguins to be used in the overall project model. After that, the YOLOv5 was used to detect and localise the location of the African penguins and fish in the video. Subsequently, the dual-stream network was used to detect the predation events of the penguins. In the network, there are two streams, which are the spatial stream and the temporal stream. The spatial stream is implemented by YOLOv5 and ResNet-18. The local detector based on YOLOv5 is responsible for providing information about the location of penguins and fish. And the global detector implemented by ResNet-18 mainly provides the spatial information of the fish in the video. In addition, the temporal stream uses the dense optical flow algorithm to extract the motion information from the video. The information from both streams is eventually fed into a behavioural detector built by LSTM to detect predation events in the video. Table 5.1 shows the final results of the three models built in this project.

Model	Penguin Accuracy	Fish Accuracy	Bubble Accuracy
YOLOv5	99%	70.2%	74.2%
Model	Have Fish Accuracy	No Fish Accuracy	Average Accuracy
ResNet-18	91.8%	90.2%	91%
Model	Feeding Accuracy	Swimming Accuracy	Average Accuracy
LSTM	74%	92%	83%

Table 5.1: Achievements of the final model in the project

It can be seen that the local detector based on the YOLOv5 model can effectively detect the location of penguins and fish in the video. And the final composed LSTM behavioural detection model also has the ability to detect some feeding and non-feeding behaviour of penguins in the video. To summarise, this project provides a highly credible model for penguin detection and a first step in attempting to detect predation events of African penguins in underwater video.

5.1 Future Work

Although the entire project has achieved its overall objectives, there still have some limitations that should be noted. The following section described these limitations and gives some direction for future development.

5.1.1 Expend the Dataset

The dataset built for this project does not contain a large enough number of underwater pictures and videos. Therefore, the trained model might not have enough generalisation ability and cause the model to perform not as expected in real-world applications. In addition, the test dataset has a small volume which does not reflect the actual accuracy of the model in a real penguin diving video. Therefore, extending the

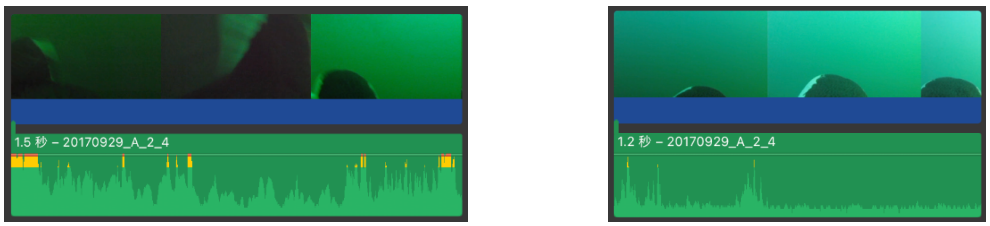
dataset is a good solution to this limitation. For example, adding other species of penguin dive videos to the dataset or combining earlier captured penguin dive videos with existing datasets to make a new dataset.

5.1.2 More Classification of the Penguin Behaviour

Another limitation is that this project only considered the predatory and non-predatory behaviour of penguins in the behavioural detection model. This is not sufficient to describe the underwater activities of African penguins. This probably leads to the model not learning some appropriate features. For example, as mentioned in Section 4.3.2, the act of turning when there are fish is considered a predation event by the model. One effective approach is to give more category definitions of penguin behaviour during the model training process, such as diving, accelerated pursuit, turning and so on. This would allow the model to learn more specific features about penguin predation events.

5.1.3 Combine the Audio

Models can improve their performance by learning more features from the data. This project only uses the visual data from the video, but features of the predation event can also be captured in the audio data from the video. By comparing the sound wave maps of the penguin predation event and the penguin swimming clip in the video, it can be seen that the sound of the predation event is much louder than when the penguin is swimming. Therefore, adding more types of features might be an effective way to improve the performance of the model in detecting penguin predation events.



Feeding Behaviour Swimming

Figure 5.1: **Audio of predation event**: This image shows a comparison of the audio graphs of the penguins in the video during the swimming and predation events.

Bibliography

[1] Laith Alzubaidi, Jinglan Zhang, Amjad J Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, José Santamaría, Mohammed A Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions. *Journal of big Data*, 8:1–74, 2021.

[2] Jason Brownlee. *Deep learning for computer vision: image classification, object detection, and face recognition in python*. Machine Learning Mastery, 2019.

[3] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6299–6308, 2017.

[4] Advanced Computing Research Centre. High performance computing. http://www.bristol.ac.uk/acrc/high-performance-computing/, 2023. Accessed: April 22, 2023.

[5] RK Chandana and AC Ramachandra. Real time object detection system with yolo and cnn models: A review. *arXiv preprint arXiv:2208.00773*, 2022.

[6] Comet. Comet: Supercharging machine learning. https://www.comet.com/docs/v2/, 2023. Accessed: April 22, 2023.

[7] RJM Crawford, R Altwegg, BJ Barham, PJ Barham, JM Durant, BM Dyer, D Geldenhuys, AB Makhado, L Pichegru, PG Ryan, et al. Collapse of south africa's penguins in the early 21st century. *African Journal of Marine Science*, 33(1):139–156, 2011.

[8] Robert JM Crawford, Peter J Barham, Les G Underhill, Lynne J Shannon, Janet C Coetzee, Bruce M Dyer, T Mario Leshoro, and Leshia Upfold. The influence of food availability on breeding success of african penguins spheniscus demersus at robben island, south africa. *Biological Conservation*, 132(1):119–125, 2006.

[9] Robert JM Crawford, William J Sydeman, Sarah Ann Thompson, Richard B Sherley, and Azwianewi B Makhado. Food habits of an endangered seabird indicate recent poor forage fish availability off western south africa. *ICES Journal of Marine Science*, 76(5):1344–1352, 2019.

[10] Tom Dietterich. Overfitting and undercomputing in machine learning. *ACM computing surveys (CSUR)*, 27(3):326–327, 1995.

[11] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[12] Mitchell Fennell, Christopher Beirne, and A Cole Burton. Use of object detection in camera trap image identification: Assessing a method to rapidly and accurately classify human and animal detections for research and application in recreation ecology. *Global Ecology and Conservation*, 35:e02104, 2022.

[13] GitHub. GitHub: Github: Let's build from here. https://github.com, 2023. Accessed: April 22, 2023.

[14] GluonCV. I3D: Getting started with pre-trained i3d models on kinetcis400. https://cv.gluon.ai/build/examples_torch_action_recognition/demo_i3d_kinetics400.html, 2023. Accessed: April 22, 2023.

[15] Shreyank N Gowda and Chun Yuan. Colornet: Investigating the importance of color spaces for image classification. In *Computer Vision–ACCV 2018: 14th Asian Conference on Computer Vision, Perth, Australia, December 2–6, 2018, Revised Selected Papers, Part IV 14*, pages 581–596. Springer, 2019.

[16] Mohamed Haggag, Mohsen M Tantawy, and Magdy MS El-Soudani. Implementing a deep learning model for intrusion detection on apache spark platform. *IEEE Access*, 8:163660–163672, 2020.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[18] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.

[19] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8):2, 2012.

[20] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[21] Javier Iranzo Sánchez. *A comparative study of Neural Machine Translation frameworks for the automatic translation of open data resources*. PhD thesis, Universitat Politècnica de València, 2018.

[22] Iza Sazanita Isa, Mohamed Syazwan Asyraf Rosli, Umi Kalsom Yusof, Mohd Ikmal Fitri Maruzuki, and Siti Noraini Sulaiman. Optimizing the hyperparameter tuning of yolov5 for underwater detection. *IEEE Access*, 10:52818–52831, 2022.

[23] Peiyuan Jiang, Daji Ergu, Fangyao Liu, Ying Cai, and Bo Ma. A review of yolo algorithm developments. *Procedia Computer Science*, 199:1066–1073, 2022.

[24] Michael Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Eighth Annual Conference of the Cognitive Science Society, 1986*, pages 513–546, 1986.

[25] Yeong Ming Keat. Video object avoidance implementation on embedded platform. IRC, 2015.

[26] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[27] Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

[28] Wuyungerile Li and Dalai Tang. *Mobile Wireless Middleware, Operating Systems and Applications: 9th EAI International Conference, MOBILWARE 2020, Hohhot, China, July 11, 2020, Proceedings*, volume 331. Springer Nature, 2020.

[29] Jingsai Liang. Confusion matrix: Machine learning. *POGIL Activity Clearinghouse*, 3(4), 2022.

[30] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[31] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

[32] Vanesa Lopez-Vazquez, Jose Manuel Lopez-Guede, Simone Marini, Emanuela Fanelli, Espen Johnsen, and Jacopo Aguzzi. Video image enhancement and machine learning pipeline for underwater animal detection and classification at cabled observatories. *Sensors*, 20(3):726, 2020.

[33] Enrique S Marquez, Jonathon S Hare, and Mahesan Niranjan. Deep cascade learning. *IEEE transactions on neural networks and learning systems*, 29(11):5475–5485, 2018.

[34] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

[35] AM McInnes and PA Pistorius. Up for grabs: prey herding by penguins facilitates shallow foraging by volant seabirds. *Royal Society open science*, 6(6):190333, 2019.

[36] Andrew Y Ng. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78, 2004.

[37] Mohammad Sadegh Norouzzadeh, Anh Nguyen, Margaret Kosmala, Alexandra Swanson, Meredith S Palmer, Craig Packer, and Jeff Clune. Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning. *Proceedings of the National Academy of Sciences*, 115(25):E5716–E5725, 2018.

[38] Otto Obrookes. GitHub: 2stream-action-recognition. https://github.com/obrookes/2stream-action-recognition/tree/master, 2023. Accessed: April 22, 2023.

[39] OpenCV. OpenCV: Home page. https://opencv.org, 2023. Accessed: April 22, 2023.

[40] Maria Gemel B Palconit, Vincent Jan D Almero, Marife A Rosales, Edwin Sybingco, Argel A Bandala, Ryan Rhay P Vicerra, and Elmer P Dadios. Towards tracking: Investigation of genetic algorithm and lstm as fish trajectory predictors in turbid water. In *2020 IEEE REGION 10 CONFERENCE (TENCON)*, pages 744–749. IEEE, 2020.

[41] Mihaela Pop, Maxime Sermesant, Pierre-Marc Jodoin, Alain Lalande, Xiahai Zhuang, Guang Yang, Alistair Young, and Olivier Bernard. *Statistical Atlases and Computational Models of the Heart. ACDC and MMWHS Challenges: 8th International Workshop, STACOM 2017, Held in Conjunction with MICCAI 2017, Quebec City, Canada, September 10-14, 2017, Revised Selected Papers*, volume 10663. Springer, 2018.

[42] PyTorch. Pytorch documentation pytorch documentation. https://pytorch.org/docs/stable/index.html. Accessed: April 18, 2023.

[43] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[44] Roboflow. Roboflow: Give your software the power to see objects in images. https://roboflow.com, 2023. Accessed: April 22, 2023.

[45] A. Rosenfeld. Computer vision: basic principles. *Proceedings of the IEEE*, 76(8):863–868, 1988.

[46] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.

[47] Faizaan Sakib. Great ape behaviour recognition using deep learning. *University of Bristol*, 2020.

[48] Alzayat Saleh, Marcus Sheaves, and Mostafa Rahimi Azghadi. Computer vision and deep learning for fish classification in underwater habitats: A survey. *Fish and Fisheries*, 23(4):977–999, 2022.

[49] Ahmad Salman, Ahsan Jalal, Faisal Shafait, Ajmal Mian, Mark Shortis, James Seager, and Euan Harvey. Fish species classification in unconstrained underwater environments based on deep learning. *Limnology and Oceanography: Methods*, 14(9):570–585, 2016.

[50] Richard B Sherley, Robert JM Crawford, Andrew D de Blocq, Bruce M Dyer, Deon Geldenhuys, Christina Hagen, Jessica Kemper, Azwianewi B Makhado, Lorien Pichegru, Desmond Tom, et al. The conservation status and population decline of the african penguin deconstructed in space and time. *Ecology and Evolution*, 10(15):8506–8516, 2020.

[51] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.

[52] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[53] Devis Tuia, Benjamin Kellenberger, Sara Beery, Blair R Costelloe, Silvia Zuffi, Benjamin Risse, Alexander Mathis, Mackenzie W Mathis, Frank van Langevelde, Tilo Burghardt, et al. Perspectives in machine learning for wildlife conservation. *Nature communications*, 13(1):792, 2022.

[54] Ultralytics. YOLOv5: A state-of-the-art real-time object detection system. https://docs.ultralytics.com, 2021. Accessed: April 22, 2023.

[55] Zhen Wang, Haolu Liu, Guangyue Zhang, Xiao Yang, Lingmei Wen, and Wei Zhao. Diseased fish detection in the underwater environment using an improved yolov5 network for intensive aquaculture. *Fishes*, 8(3):169, 2023.

[56] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. *Advances in neural information processing systems*, 30, 2017.

[57] Mingyu Yang. Reliable penguin detection in underwater footage using deep learning. *University of Bristol*, 2022.

[58] Xue Ying. An overview of overfitting and its solutions. In *Journal of physics: Conference series*, volume 1168, page 022022. IOP Publishing, 2019.

[59] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, 30(11):3212–3232, 2019.

[60] Yangfan Zhou, Xin Wang, Mingchuan Zhang, Junlong Zhu, Ruijuan Zheng, and Qingtao Wu. Mpce: a maximum probability based cross entropy loss function for neural network classification. *IEEE Access*, 7:146331–146341, 2019.

Appendix A

YOLO Hyperparameter

```
lr0: 0.01  # initial learning rate (SGD=1E-2, Adam=1E-3)
lrf: 0.01  # final OneCycleLR learning rate (lr0 * lrf)
momentum: 0.937  # SGD momentum/Adam beta1
weight_decay: 0.0005  # optimizer weight decay 5e-4
warmup_epochs: 3.0  # warmup epochs (fractions ok)
warmup_momentum: 0.8  # warmup initial momentum
warmup_bias_lr: 0.1  # warmup initial bias lr
box: 0.05  # box loss gain
cls: 0.5  # cls loss gain
cls_pw: 1.0  # cls BCELoss positive_weight
obj: 1.0  # obj loss gain (scale with pixels)
obj_pw: 1.0  # obj BCELoss positive_weight
iou_t: 0.20  # IoU training threshold
anchor_t: 4.0  # anchor-multiple threshold
# anchors: 3  # anchors per output layer (0 to ignore)
fl_gamma: 0.0  # focal loss gamma (efficientDet default gamma=1.5)
hsv_h: 0.015  # image HSV-Hue augmentation (fraction)
hsv_s: 0.7  # image HSV-Saturation augmentation (fraction)
hsv_v: 0.4  # image HSV-Value augmentation (fraction)
degrees: 0.0  # image rotation (+/- deg)
translate: 0.1  # image translation (+/- fraction)
scale: 0.5  # image scale (+/- gain)
shear: 0.0  # image shear (+/- deg)
perspective: 0.0  # image perspective (+/- fraction), range 0-0.001
flipud: 0.0  # image flip up-down (probability)
fliplr: 0.5  # image flip left-right (probability)
mosaic: 1.0  # image mosaic (probability)
mixup: 0.0  # image mixup (probability)
copy_paste: 0.0  # segment copy-paste (probability)
```

Code Listing A.1: List of YOLOv5 hyperparameter from YOLOv5 official [54]

Appendix B

GitHub Repo

The latest version of this thesis code can be found here: https://github.com/Kejia928/predationDetector