



University of Southern Queensland
Faculty of Health, Engineering and Sciences

Real Time Drowning Detection System Using Machine Vision and Learning

Dissertation submitted by
Fern Proctor

In fulfilment of the requirements of
Course ENG4111/4112 – Research Project

towards the degree of
Bachelor of Engineering (Electrical and Electronic)

Submitted: October 2023

This page intentionally left blank

Abstract

Globally, there are an estimated 236,000 drownings per year (World Health Organisation, 2014), and in Australia, drowning deaths form a grim tally over the summer months despite the introduction of robust legislation and education campaigns. Swimming pools are the most common location for drowning accidents (AIHW, 2023) and as such, any tools that may be able to reduce these occurrences could have substantial impact.

There have been significant advances in the fields of machine vision and learning in the past decade resulting in the development of new algorithms and hardware which can be applied to novel applications. Of particular interest is the ‘You Only Look Once’ (YOLO) algorithm which offers real time detection speeds with impressive accuracy. With a solution looking for a problem to solve, this project aimed to develop a real time drowning detection system using machine learning and vision. In particular, the system was designed to be used in a residential pool setting with little technical knowledge required for installation and setup.

A dataset was acquired and supplemented with additional images. Several versions of the Yolo algorithm were examined and trained using the custom dataset. Image augmentation and hyperparameter tuning were among some of the methods used to improve the model’s accuracy prior to it being deployed on an Oak-D Lite, an edge-AI camera where neural inference is done onboard rather than on a separate device. Finally, the system was deployed and tested in real time as well as monitored remotely through a mobile device. The testing demonstrated that real time drowning detection was feasible using YoloV8 and the Oak-D Lite.

Future works include improving and increasing the size of the dataset used for training as well as experimenting with various iterations of the Yolo algorithms deployed on the Oak-D Lite, in particular smaller versions to compare their accuracy with YoloV8-m used in the final system. In addition, further investigation of hyperparameter tuning and image augmentation could be beneficial as well as development of a mobile interface with the ability to notify users of a drowning event by means of a notification or via smart devices such as watches.

Limitations of Use

University of Southern Queensland

Faculty of Health, Engineering and Sciences

ENG4111/ENG4112 Research Project

Limitations of Use

The Council of the University of Southern Queensland, its school of engineering, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Health, Engineering & Sciences or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitled “Research Project” is to contribute to the overall education within the student’s chosen degree program. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

Certification of Dissertation

University of Southern Queensland

Faculty of Health, Engineering and Sciences

ENG4111/ENG4112 Research Project

Certification of Dissertation

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.

Portions of the research in this paper use the Water Behaviour dataset collected under the Electrical Engineering department at Rochester Institute of Technology Dubai.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

F. Proctor



Acknowledgements

I would like to acknowledge and thank Dr. Tobias Low for his initial proposal of the Real Time Drowning Detection System as well as his consistent and valuable guidance throughout this work. I would also like to thank the Rochester Institute of Technology in Dubai for providing their dataset as well as Katie for offering to jump in and add to mine. Finally, I would like to thank my family, friends, and colleagues for their support over the last five and a half years, in particular my mum, dad and Tark for their untiring backing and being my personal cheer squad.

Table of Contents

| | |
|--|-----------|
| 1. Introduction | 1 |
| 1.1 Background | 1 |
| 1.2 Computer and Machine Vision | 1 |
| 1.3 Machine Learning and Algorithms | 2 |
| 1.3.1 Object Detection | 3 |
| 1.3.2 Convolutional Neural Networks | 4 |
| 1.3.3 You Only Look Once | 5 |
| 1.3.4 Metrics in Object Detection Algorithms | 6 |
| 1.4 Project Justification | 8 |
| 1.5 Project Aims | 8 |
| 1.6 Research Questions | 9 |
| 2. Literature Review | 10 |
| 2.1 Drowning Prevention Techniques | 10 |
| 2.1.1 Sensor Solutions | 10 |
| 2.1.2 Machine Vision Solutions | 11 |
| 2.2 Commercial Drowning Detection Systems | 14 |
| 2.3 Drowning Detection Using Machine Vision | 16 |
| 2.3.1 Hardware | 16 |
| 2.3.2 Algorithms | 17 |
| 2.3.3 Datasets | 19 |
| 2.4 Machine Vision Developments | 21 |
| 2.5 Knowledge Gaps | 23 |
| 3 Methodology | 24 |
| 3.1 Research Methodology | 25 |
| 3.2 Task Analysis | 25 |
| 3.2.1 Dataset Acquisition and Preparation | 25 |

| | |
|--|-----------|
| 3.2.2 Metrics | 25 |
| 3.2.3 Algorithm Selection | 26 |
| 3.2.4 Build, Train, and Deploy Model | 26 |
| 3.3 Experimental Design | 27 |
| 3.3.1 Experiment 1 – Qualitative Testing of Yolo Models | 27 |
| 3.3.2 Experiment 2 – Quantitative Testing of Yolo Models on a Custom Dataset | 27 |
| 3.3.3 Experiment 3 – Improving the Yolo Model | 29 |
| 3.3.4 Experiment 4 – Model Deployment on the Oak-D and Raspberry Pi | 29 |
| 3.3.5 Experiment 5 – Real time detection and alert of drowning events | 29 |
| 3.4 Resource Analysis | 30 |
| 3.4.1 Hardware | 30 |
| 3.4.2 Software | 31 |
| 3.4.3 Dataset | 31 |
| 3.4.4 Test Location | 32 |
| 3.4.5 Project Cost | 32 |
| 3.5 Project Consequential Effects | 32 |
| 3.5.1 Sustainability | 33 |
| 3.5.2 Ethics | 33 |
| 3.6 Risk Assessment | 34 |
| 3.6.1 Risks Identified | 34 |
| 4. Development and Experimental Setup | 35 |
| 4.1 Dataset Acquisition and Preparation | 35 |
| 4.1.1 Dataset Acquisition | 35 |
| 4.1.2 Dataset Preparation | 36 |
| 4.2 Experiment 1 – Qualitative Testing of Yolo Models | 38 |
| 4.3 Experiment 2 – Quantitative Testing of Yolo Models on a Custom Dataset | 39 |
| 4.3.1 YoloV7 | 40 |

| | |
|--|-----------|
| 4.3.2 YoloV8 | 41 |
| 4.4 Experiment 3 – Improving the Yolo Model | 42 |
| 4.4.1 Inclusion of background images | 42 |
| 4.4.2 Increasing the Model Size | 42 |
| 4.4.3 Image Augmentation and Parameter Tuning | 42 |
| 4.4.5 Increasing Training Epochs | 43 |
| 4.5 Experiment 4 – Deployment on the Oak-D Lite and Raspberry Pi | 44 |
| 4.5.1 Creating a .blob file | 44 |
| 4.5.2 Deploying the Model | 44 |
| 4.6 Experiment 5 – Real Time Detection and Alert | 45 |
| 4.6.1 Drowning Detection Interface | 45 |
| 4.6.2 Remote Drowning Detection | 46 |
| 4.6.3 Experimental Set Up | 46 |
| 5. Results and Analysis | 48 |
| 5.1 Dataset | 48 |
| 5.2 Experiment 1 – Qualitative Testing of Yolo Models | 49 |
| 5.2.1 YoloV3 | 49 |
| 5.2.2 YoloV7-tiny | 50 |
| 5.2.3 YoloV8 | 51 |
| 5.2.4 Detection Speeds | 52 |
| 5.3 Experiment 2 - Quantitative Testing of Yolo Models on a Custom Dataset | 53 |
| 5.3.1 YoloV7-tiny | 53 |
| 5.3.2 YoloV8-s | 56 |
| 5.4 Experiment 3 – Improving the Yolo Model | 60 |
| 5.4.1 Background Images | 60 |
| 5.4.2 Increasing the Model Size | 63 |
| 5.4.3 Image Augmentation and Parameter Tuning | 64 |

| | |
|--|-----------|
| 5.4.4 Training Epochs | 67 |
| 5.5 Experiment 4 – Deployment on the Oak-D Lite | 70 |
| 5.6 Experiment 5 – Real Time Detection and Alert | 71 |
| 6. Discussion, Conclusion and Further Work | 76 |
| 6.1 Discussion | 76 |
| 6.2 Conclusion | 78 |
| 6.3 Further Work | 79 |
| References | 81 |
| Appendix A | 87 |
| Appendix B – Risk Assessment | 90 |
| Appendix C – Dataset Release Agreement | 93 |
| Appendix D – Google Colab Notebooks and Python Script | 94 |
| YoloV7-tiny Custom | 94 |
| Data File | 94 |
| Configuration File | 94 |
| Google Colab Notebook | 96 |
| YoloV8-s Custom | 97 |
| Data File | 97 |
| Google Colab Notebook | 97 |
| YoloV8-m Custom – Final Model | 99 |
| Datafile | 99 |
| Google Colab Notebook | 99 |
| Python Script | 100 |

List of Tables

| | |
|---|----|
| Table 1: A performance comparison of object detection models (Kaur and Singh, 2022) | 18 |
| Table 2 Yolo models and their performance on the Oak-D Lite (Luxonis, 2023) | 30 |
| Table 3: Project Costs | 32 |
| Table 4: A breakdown of the image dataset | 36 |
| Table 5: The number of images in each dataset for training and testing. | 39 |
| Table 6: Additional training parameters for the customYoloV7 model. | 41 |
| Table 7: Additional training parameters for the custom YoloV8 model. | 42 |
| Table 8: The image augmentation parameters to be used during for the model training..... | 43 |
| Table 9: The test results using the custom YoloV7-tiny best weights..... | 54 |
| Table 10: The test results using the custom YoloV8-s best weights. | 57 |
| Table 11: Key metrics for YoloV8-m trained on the improved dataset over 24 epochs. | 63 |
| Table 12: The image augmentation parameters used in the experiments. | 66 |

Table of Figures

| | |
|--|----|
| Figure 1: A block diagram of a machine vision system (Labudzki et al, 2014) | 2 |
| Figure 2: A timeline of object detection algorithm development (Zou et al, 2023)..... | 4 |
| Figure 3: The various layers of a Convolutional Neural Network (Mathworks, 2023)..... | 5 |
| Figure 4: Image processing and object detection using the YOLO algorithm (Redmon et al, 2016) | 6 |
| Figure 5: Instinctive Drowning Response behaviours (Great Lakes Surf Rescue Project, 2013) | 10 |
| Figure 6: Segmented silhouettes of various swimming and drowning behaviours for use in the behavioural recognition system (Eng et al, 2008). | 12 |
| Figure 7a and b: Angel Eye user interface and notification device (Angel Eye, 2023)..... | 15 |
| Figure 8: Drowning stages detected by the Yolo algorithm used by Handalage et al (Handalage et al, 2021). | 19 |
| Figures 9a and b: Drowning and Swimming detections using ResNet50 (Hasan et al, 2021).20 | |
| Figure 10: The Oak-D Lite edge AI camera. In the centre is the colour camera with two stereo cameras either side (Luxonis, 2023)..... | 21 |
| Figure 11: A comparison of speed and mAP of Yolo versions 5 to 8 (Ultralytics, 2023). | 28 |
| Figure 12: The folder directory structure of the provided dataset with the number of videos 35 | |
| Figure 13: Drawing a bounding box on a training image using Labellmg. | 37 |
| Figure 14: A text file associated with a labelled image. | 37 |
| Figure 15: The class declaration file where drowning is 0, idle is 1 and swimming is 2. | 38 |
| Figures 16a and b: The two images for qualitative testing, left ‘Kicking Horse’ and right ‘Skateboards’. | 39 |
| Figure 17: The process for training a custom Yolo model using Google Colab..... | 40 |
| Figure 18: A custom data file for YoloV7. | 41 |
| Figure 19: The custom data file for YoloV8. | 41 |
| Figure 20: The pseudo code for deploying the Yolo model in Experiment 4. | 44 |
| Figure 21: The equipment set up for Experiment 4 - deploying the Yolo model. | 45 |
| Figure 22: Pseudocode for experiment 5. | 46 |
| Figures 23a and b: The experimental setup with the Oak-D Lite, Raspberry Pi and power bank..... | 47 |
| Figure 24: A histogram showing the occurrences of the three classes in the training dataset. 48 | |
| Figure 25: The occurrence of two classes in a training image (swimming and idle). | 49 |

| | |
|--|----|
| Figures 26a and b: YoloV3 detections run on the test images trained on the COCO dataset. . | 49 |
| Figures 27a and b: YoloV7-tiny detections run on the test images trained on the COCO dataset. | 50 |
| Figures 28a and b: YoloV8-m detections run on the test images trained on the COCO dataset. | 51 |
| Figure 29: A comparison of detection times of the Yolo models tested..... | 52 |
| Figure 30: YoloV7-tiny losses during training over 24 epochs. | 53 |
| Figure 31: Precision, recall and mAP during training of YoloV7-tiny over 24 epochs. | 54 |
| Figure 32: Correct detection of swimming with confidence of 0.60. | 54 |
| Figure 33: Detection of swimming and drowning in the drowning state. | 55 |
| Figure 34: Incorrect detection of swimming in the idle state. | 55 |
| Figure 35: YoloV8-s losses during training over 24 epochs. | 56 |
| Figure 36: Precision, recall and mAP during training of YoloV8-s over 24 epochs..... | 57 |
| Figure 37: Incorrect detection of idle in the swimming state. | 57 |
| Figure 38: Incorrect Detection of swimming in the drowning state. | 58 |
| Figure 39: Incorrect detection of swimming in the idle state. | 58 |
| Figure 40: A comparison of key metrics between YoloV8-s trained on the original and improved dataset with background images..... | 60 |
| Figures 41a and b: The confusion matrices for the YoloV8-s models. On the left the model trained with no background images and on the right the model trained with background images. | 61 |
| Figure 42: A comparison of YoloV8-s and YoloV8-m key metrics. | 63 |
| Figure 43: A comparison of total losses between YoloV8-s and YoloV8-m..... | 63 |
| Figure 44: A comparison of batch sizes during training of YoloV8-m. | 64 |
| Figure 45: Key metrics for the three learning rates tested. | 65 |
| Figure 46: Total losses for the three learning rates tested..... | 65 |
| Figure 47: Key metrics with various image augmentation methods implemented..... | 66 |
| Figure 48: A comparison of key metrics across three models, the original YoloV8-m trained for 24 epochs, the improved 24 epoch version and the final version trained for 300 epochs.. | 67 |
| Figure 49: The confusion matrix for the final model..... | 68 |
| Figures 50a and b: The precision-recall curve and F1 confidence curve for the final model.. | 68 |
| Figures 51a and b: Images from the test dataset, on the left the drowning class and on the right swimming, which when taken out of context could prove difficult for even a human to classify..... | 69 |

| | |
|--|----|
| Figures 52a and b: Correct detections in the drowning and swimming states..... | 70 |
| Figures 53a and b: Correct detection of the idle state and an incorrect detection of swimming in the drowning state..... | 70 |
| Figures 54a and b: Detection of the drowning state and the alert holding despite the loss of the drowning bounding box at 1080p resolution. | 71 |
| Figures 55a and b: Correct detection of idle and swimming at 1080p resolution. | 72 |
| Figures 56a and b: Correct detection of drowning at a higher resolution and the alarm test holding despite the loss of the bounding box..... | 72 |
| Figures 57a and b: Correct detections of the swimming state at higher resolution. | 73 |
| Figures 58a and b: An incorrect detection of swimming in the drowning state and a missed detection of drowning at higher resolution. | 73 |
| Figure 59: Successful detection and remote monitoring using the VNC from the Raspberry Pi to iPhone. | 74 |

Glossary

| | | |
|-------|---|---------------------------------------|
| CNN | = | Convolutional Neural Network |
| CPU | = | Computer Processing Unit |
| GPU | = | Graphical Processing Unit |
| IDR | = | Instinctive Drowning Response |
| mAP | = | Mean Average Precision |
| YOLO | = | You Only Look Once |
| R-CNN | = | Regional Convolutional Neural Network |
| RNN | = | Recurrent Neural Network |
| YOLO | = | You Only Look Once |

1. Introduction

1.1 Background

Globally, there are an estimated 236,000 drownings per year (World Health Organisation, 2014) and in Australia swimming pools were the leading location of drowning deaths for children between the ages of 0-4 between 2002 and 2018 (Royal Lifesaving Australia, 2018). For each drowning death in Australia, there are an additional three people hospitalised with drowning related injuries costing the Australian economy \$1.24 billion annually (Peden et al, 2021). In Australia, drowning deaths are generally trending downwards with the introduction of robust legislation such as pool fences and signage, though experts agree that a multifaceted approach to reducing drowning deaths is required. In half of the drowning deaths of children in private residential swimming pools, there was adult supervision, however, lapses in this supervision contributed to the deaths (Peden et al, 2019).

One study examining the drowning deaths of 447 children under the ages of 4, 53.5% of the deaths occurred in swimming pools with 86.6% of these being private residential swimming pools (Peden et al, 2019). In over half of the cases, there were more than two supervisors present thus demonstrating that quality supervision is key to preventing drowning deaths. Leading causes of distractions when supervising were household duties (indoors and outdoors), talking and socialising. When there were two or more supervisors present, miscommunication occurred resulting in the lapse (Peden et al, 2019).

1.2 Computer and Machine Vision

Computer vision and machine vision are phrases which are frequently used interchangeably, though machine vision is the practical application of computer vision principles. Computer vision is the processing of an image using a variety of methods to extract information, whereas machine vision generally requires additional hardware and software to respond to the information gathered, such as the movement of a robotic arm in automated manufacturing (Labudzki et al, 2014). As shown in Figure 1, a machine vision system contains several sub systems; optical acquisition hardware (such as a camera), where scene constraints such as lighting and positioning must be considered, a pre-processing stage, segmentation of the image, feature extraction, classification/interpretation and finally actuation (Labudzki et al, 2014).

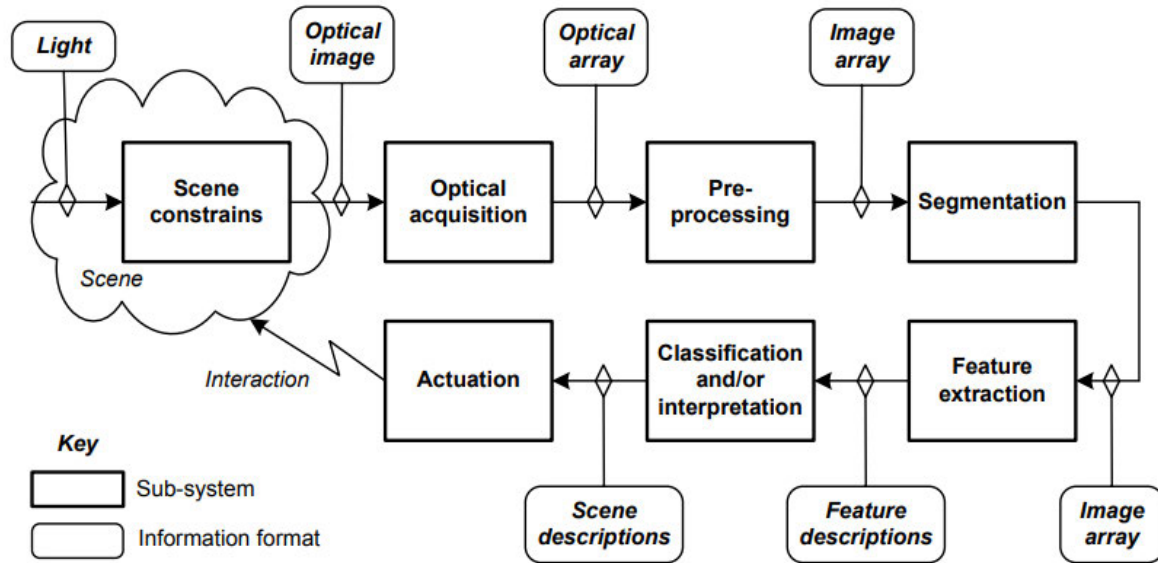


Figure 1: A block diagram of a machine vision system (Labudzki et al, 2014)

Machine vision systems are an established technology with wide ranging applications. Examples include barcode scanners and defect detection systems used in manufacturing. A commonality with classic machine vision systems is the controlled conditions they operate in. Barcodes by design are standardised, high contrast and easily interpreted. Defect detection systems in manufacturing plant can be installed with optimum lighting and positioning. Until recently, machine vision systems struggled with random variation and uncontrollable variables, however developments within the field of machine learning with Convolutional Neural Networks (CNN) means these systems are now able to automatically learn from data to develop rules for complex machine vision (Smith et al, 2021).

1.3 Machine Learning and Algorithms

The enigmatic qualities of machine learning have drawn the field into sharp focus over the past several years. Attention grabbing headlines covering the defeat of Go master Lee Sedol by AlphaGo and his ultimate retirement claimed, ‘machines cannot be defeated’ (The Guardian, 2019). However, rather than attempting to ‘defeat’ machines, it is important to recognise the opportunities machine learning presents.

Algorithms are a set of instructions for achieving a task. Often, algorithms are associated with computers, however, they have existed for millennia, with the first algorithms dating back to the ancient Babylonians (Louridas, 2020). Computers are simply an effective way of implementing and quickly executing an algorithm using a programming language.

Machine learning harnesses the vast amounts of data now available to automatically create algorithms for a given task (Alpaydin, 2021). Recommendation engines in common applications such as Netflix and Spotify learn from user's preferences to make suggestions. Machine learning and vision have been combined in medical and agricultural fields to great effect to improve patient outcomes and productivity (Smith et al, 2021). Several novel machine learning algorithms for vision and object detection have been proposed. Convolutional Neural Networks (CNN) and the You Only Look Once (YOLO) are two algorithms which are commonly used in vision applications due to their speed and accuracy.

1.3.1 Object Detection

Object detection algorithms are the cornerstone of a number of real-world applications such as autonomous driving and facial recognition. However, the significant progress in object detection in the last 15 years should be acknowledged. In the early 2010's, computers struggled to differentiate between a cat and a dog. Traditional object detection methods were bespoke systems often focusing on the use of filters, de-noising and edge finding for identification and classification of objects (Lakshmana et al, 2021).

With advancements in deep learning models and detection, the speed and accuracy of object detection has increased significantly, though even then require expensive GPUs to run. As figure 2 shows, from 2014 onwards and beyond the 'traditional' object detection methods, there were two distinct splits: one stage and two stage detectors. One stage detectors localise a region of interest and detection in the same step. Two stage detectors first run a 'Region Proposal Network' to find areas of interest, these are processed and enhanced before the second stage classifies these objects. Generally, two stage detectors have high precision though in comparison to single stage detectors are slow and computationally intensive. Single stage detectors can be deployed in real-time applications though experience reduced performance when detecting dense and small objects (Zou et al, 2023). Two detectors are of particular interest in this research; CNNs and YOLO.

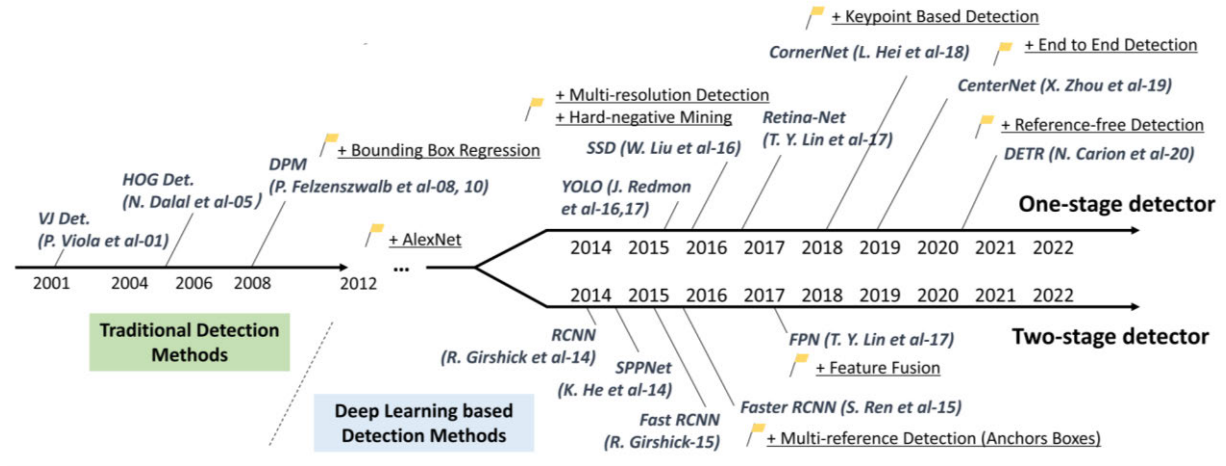


Figure 2: A timeline of object detection algorithm development (Zou et al, 2023)

1.3.2 Convolutional Neural Networks

Convolutional Neural Networks are a two-stage detector where numerous layers are used for object classification and localisation. The key layers are the convolution layer and pooling layer which produce feature maps, the fully connected layer which is a high-level reasoning layer for linear classification and a SoftMax layer which assigns objects into their various classes (Malhotra et al, 2020). Algorithmic improvements are made by performing several iterations to produce more accurate results.

The convolution layer uses several filters to enhance various features within the image. This is often used in conjunction with a rectified linear unit (ReLU), sometimes referred to as activation, which maps negative values to zero while maintaining positive values. Only the activated values progress to the next layer. The pooling layer simplifies the output by reducing the number of parameters that need to be learned (Mathworks, 2023). These are repeated over many layers to detect different features. Once the features have been learned, the network progresses to the classification layer, of which the Softmax function is a part of. This layer outputs a vector of n -dimensions (where n denotes the number of classes which can be predicted) and estimates the probabilities of each image class being present.

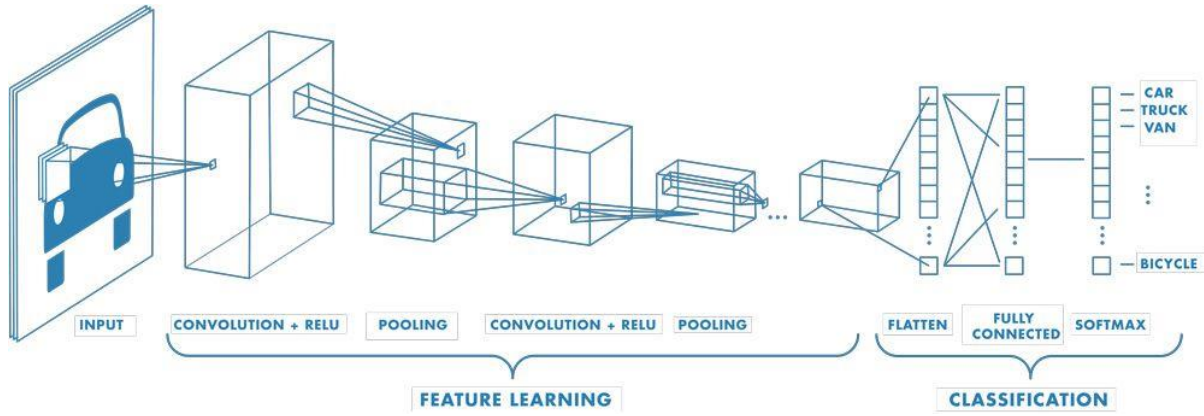


Figure 3: The various layers of a Convolutional Neural Network (Mathworks, 2023)

There are several variations to the standard CNN. These are Regional CNN (R-CNN) which localizes 2000 regions of interest within an image before applying the CNN layers, Fast R-CNN in which the image is processed to create a convolutional feature map before regions of proposal are identified. Finally, Faster R-CNN discards the selective search algorithm and instead allows the CNN to learn the region proposals. Faster R-CNN is significantly faster than both previous methods and as such can be used for real time object detection (Gandhi, 2018).

1.3.3 You Only Look Once

You Only Look Once (YOLO) is an algorithm which approaches object detection as a regression problem, assessing the image from the pixels to bounding box coordinates and class probabilities (Redmon et al, 2016). As a result, and unlike CNNs, complex pipelines are not required which significantly increases speed to the point where it can be used as a real time object detection system.

YOLO is based on the Darknet architecture and divides the image into a number of S by S cells and a single neural network is applied to the image. If the central point of an object is in a particular cell, it is responsible for detection of that object. This initial network layer extracts features from the object whilst the fully connected layers predict output probabilities and coordinates. A vector is produced for each bounding box with five predictions: x , y , w , h and confidence. The (x,y) coordinates denote the centre of the box in relation to the bounds of the particular grid cell. The width and height (w,h) are the dimensions of the bounding box relative to the entire image and the confidence score is the intersection over union (IOU) of an object.

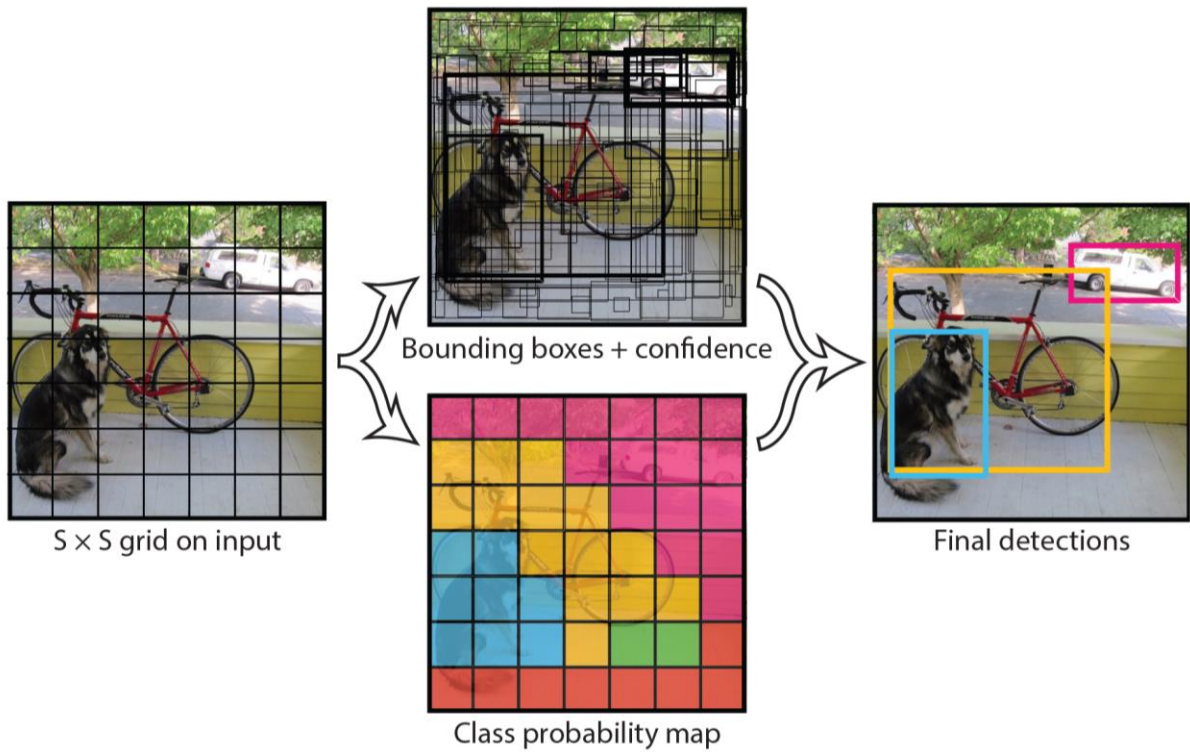


Figure 4: Image processing and object detection using the YOLO algorithm (Redmon et al, 2016)

There have been several iterations and improvements made to the YOLO algorithm since its initial release in 2016. YOLOv2 (or YOLO9000) offered greater accuracy and speeds and YOLOv3 replaced computationally intensive Softmax functions with independent logistics classifiers (Shah et al, 2022). Joseph Redmon, often credited with the creation of the YOLO algorithm, stopped working on it beyond version 3. However, YOLOv4 was developed offering improved performance and has been widely used in research. YOLO versions 5 to 8 have been developed though more research is needed to analyse their effectiveness (Shah et al, 2022).

1.3.4 Metrics in Object Detection Algorithms

There are several key metrics which are used to measure the effectiveness of object detection algorithms and in turn to compare them. The most fundamental of these is the following:

- i) True Positive (TP) – The correct detection of a bounding box
- ii) False Positive (FP) – The incorrect detection of a non-existent object or the misplaced detection of an existing object.
- iii) False Negative (FN) – A bounding box that failed to be detected around an object.

The correctness of a prediction is measured using Intersection Over Union (IOU). This is the overlap between a predicted bounding box and the ground truth bounding box, that is, the

bounding box which was manually placed around the object in the validation dataset.

Mathematically, IOU can be defined as follows:

$$IOU = \frac{\text{Area of Intersection}}{\text{Area of Union}}$$

If the IOU is greater than a defined threshold value, the detection is deemed to be positive.

Precision and recall are also commonly used metrics. Precision is the ability of the detector to identify relevant objects correctly, equivalent to the percentage of correct true predictions.

Recall is the model's ability to correctly identify all ground-truth bounding boxes.

Mathematically precision and recall can be defined as follows:

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{\text{All Detections}}$$

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{\text{All Ground Truths}}$$

A Precision-Recall curve can be plotted using these values. If the number of false positives is low, the model's precision will be high. However, there is the potential for many positives to have not been detected thus the number of false negatives will increase meaning lower recall. The same may apply to a greater number of true positives, in this case recall will increase though more false positives may be detected lowering the recall. A well performing model will ideally have high precision and high recall, thus the area under the precision-recall curve will be greater. Conversely, low precision and low recall results in a smaller area. This relationship is defined as the Average Precision and is only applicable to each individual class of object. Mathematically, it can be defined as follows:

$$\text{Average Precision (AP)} = \int_{r=0}^1 p(r)dr$$

To measure the Average Precisions across all classes in an object detection algorithm, another metric is introduced. The mean average precision (mAP) is the mean of the average precision for all classes, often this is used to gauge the accuracy of the entire algorithm.

Mathematically this is defined as follows:

$$mAP = \frac{1}{k} \sum_i^k AP_i$$

where k = number of classes

It should be noted that there are several variants of these metrics being utilised and it is important to understand the different definitions used in context. These are as follows:

- i) AP – the Average Precision is measured using various IOU thresholds. Generally, this range is between 50-95% which increments by 5%. This is defined as AP@50:5:95. Alternatively single values may be used which are commonly 50% and 75% which report as AP50 and AP75.
- ii) AP Across Scales – Average Precision for differently sized objects classed as small (32 x 32 pixels), medium (up to 96 x 96 pixels) and large (greater than 96 x 96 pixels).
- iii) Average Recall (AR) – maximum recall values over a fixed number of detections per image averaged over IOU and class.
- iv) AR Across Scales – AR determined over different sized objects as defined in AP Across Scales. Often referred to as AR-S (small), AR-M (medium) and AR-L (large).

(Padilla et al, 2020)

1.4 Project Justification

There are clear opportunities to research and prototype a drowning detection system using machine vision and learning given the recent advancements in the field. In addition, there are no market ready solutions for residential pool settings using only surface cameras. The commercial solutions currently available are bespoke designs with specific install requirements for different pools. The release of AI-enabled cameras such as the Oak-D Lite present additional opportunities to investigate the deployment of real time object detection algorithms for use in drowning detection.

1.5 Project Aims

The aims of this project are as follows:

- 1) Research object detection algorithms which may be used in a real time drowning detection system.
- 2) Gather a dataset with which to train the selected algorithms. Select an algorithm to be deployed based on appropriate parameters.
- 3) Deploy the algorithm on hardware which is then able to make detections using only surface cameras in a residential pool setting.
- 4) Create an interface which can alert other users to a potential drowning event.

1.6 Research Questions

There are several questions which this research project shall aim to answer:

- 1) What is the most effective algorithm which can be used in a real time drowning detection system?
- 2) What size dataset is needed to train the selected algorithm and gain accurate results?
- 3) Where can a dataset for a drowning detection system be sourced?
- 4) What method and hardware should be used to train the algorithm?
- 5) What is the most appropriate hardware to implement the algorithm on to build a real time drowning detection system?
- 6) How can the effectiveness of the drowning detection system be measured? Both quantitatively and qualitatively.

2. Literature Review

2.1 Drowning Prevention Techniques

Many novel approaches have been proposed for drowning prevention techniques. Technology for drowning prevention largely fall into two categories: image-based processing and methods employing sensors and embedded systems (Shehata et al, 2021). The reviewed literature indicates that image-based processing is an effective method for drowning detection. Embedded systems and wearable sensors are also a low-cost and reasonably effective solution with monitoring of motion, heart rate, blood oxygen level and depth being possible (Jalalifar et al. 2022). However, a key disadvantage of a sensor-based system is that it still relies on the swimmer wearing the device with discomfort being a key disadvantage noted, particularly among children (Alshbatat et al, 2021).

2.1.1 Sensor Solutions

Several novel sensor-based solutions have been proposed for drowning detection systems. One system utilised a device which detected 3-axis acceleration and pressure underwater and was tested being worn on various positions on the body including the head, chest, waist, wrists, and ankles (Konishi et al, 2022). The design was based upon the Instinctive Drowning Response (IDR) behaviour generally exhibited by persons drowning as shown in figure 5. Such behaviour includes the mouth being at water level, facing the shore or exit of a body of water, the head being titled back, the body being vertical and a ladder climbing motion (Great Lakes Surf Rescue Project, 2013). The prototype produced results whereby when IDR behaviours were demonstrated by the various lifesavers wearing the devices, similar pressure waveforms were seen. However, further research was needed to distinguish between similar actions such as IDR, standing and non-drowning behaviours.

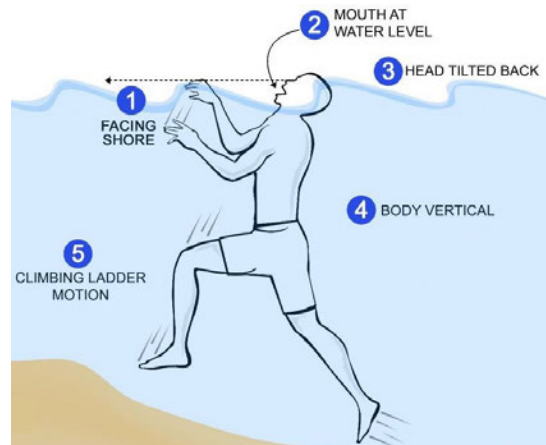


Figure 5: Instinctive Drowning Response behaviours (Great Lakes Surf Rescue Project, 2013)

Jalalifar et al proposed a multi-sensor device to detect swimmers in distress. Their design included the ability to measure four parameters which included heart rate, blood oxygen saturation, acceleration, and water depth. Should any of the measured values cross a pre-defined threshold for a certain period, an output indicating a potentially drowning swimmer is displayed, in addition a message can be sent via Wi-Fi to a phone or laptop to notify lifesavers (Jalalifar et al, 2022). The device was designed to be worn on the wrist, and with the addition of a battery pack and case was noticeably bulky. However, the initial test provided some promising results, the system performed well and was able to provide accurate data even in harsh conditions. The limitations of the system included the bulkiness of the device as well as communication limitations as a mesh network was needed.

Finally, a system named the Falling and Drowning Detection (FaDD) framework was developed. This used smartphone technology and embedded sensors including an accelerometer, gyroscope, magnetometer, and GPS in conjunction with machine learning models (Alqahtani et al, 2022). Firstly, the framework determines if a person is falling and then predicts a drowning situation. The data was then processed by three different machine learning models to assist in event identification. They found that the gyroscope did not provide useful data due to the reduction in gravity underwater and so this was removed. The magnetometer and accelerometer were used to determine a fall and then further analysis determined whether it was a drowning event. The authors found that there were differences in the data acquired from the accelerometer and magnetometer in each situation. For drowning, the signals were subtle whereas in a fall only, a significant spike was seen in readings. They found that the system had a 98% accuracy rate in event identification. However, a noted limitation was poor cellular and Wi-Fi signal propagation underwater. In addition, this system would be limited to certain situations such as an accidental fall into a body of water as opposed to the user being in the water and then starting to drown.

2.1.2 Machine Vision Solutions

Accurate identification of drowning utilising machine vision poses several challenges. One issue is the occurrence of high levels of noise when identifying foregrounds and behaviour recognition due to the random nature of movement on the surface of a body of water (Eng et al, 2008). Methods have been proposed for modelling dynamic aquatic backgrounds, recognising behaviours associated with drowning and addressing moderate levels of crowding within swimming pools in a paper by Eng et al in 2008. Of key interest with this proposed system is the utilisation of only surface cameras and thus offers the potential for

earlier detection of drowning events given that underwater cameras are more likely to detect drowning at a later stage, once the swimmer has sunk to the bottom (Eng et al, 2008). Since then, several advances have been made with the use of image processing and machine learning algorithms for drowning detection.

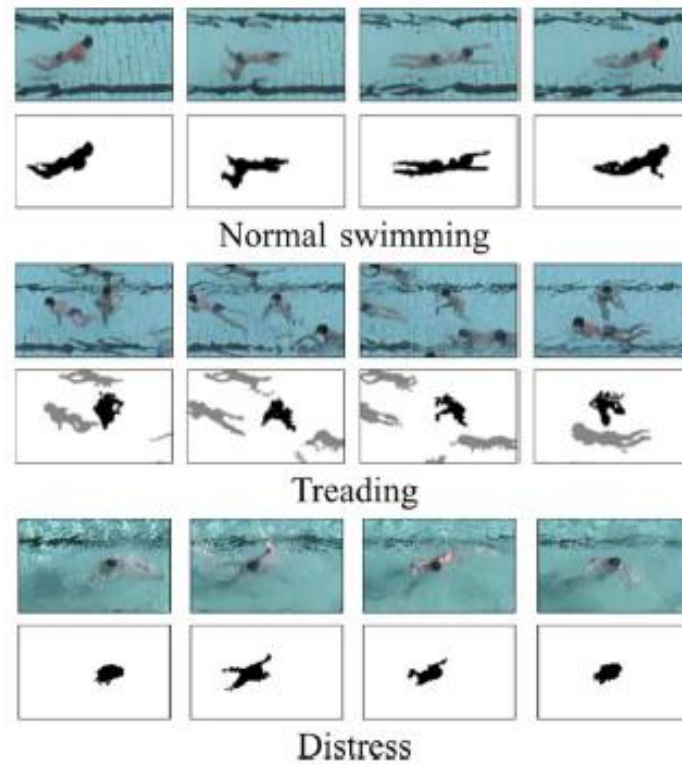


Figure 6: Segmented silhouettes of various swimming and drowning behaviours for use in the behavioural recognition system (Eng et al, 2008).

A system examined by Salehi et al utilised existing security cameras surrounding a swimming pool for drowning detection utilising HSV (Hue, Saturation, Value) colour spaces to track individuals and detect any potential drowning events. Water ripples proved to be challenging to the image recognition software but the HSV colour space contour detection proved to be a robust method for drowning detection with an average drowning detection delay of 1.53 seconds and minimal false alarms sent by the algorithm (Salehi et al, 2014). The performance of the system was impressive given that no machine learning algorithms had been used, though with advancements in this field since 2014, there are clear opportunities to improve on these results.

One novel system proposed utilised a customised YOLO algorithm to detect drowning in three stages, each defined by a particular behaviour associated with drowning such as a climbing ladder motion or the head being underwater (Handalage et al, 2021). The system also only utilises surface cameras with the aim of recognising struggling motions before

drowning occurs. Individuals are identified using object detection through the YOLO algorithm and then a CNN human detection algorithm determines if they are exhibiting drowning behaviour. Skeletal sketching using OpenPose is used to recognise a person's pose, the CNN is then used to identify hazardous or non-hazardous behaviour to identify possible drowning events. Using test images, the algorithm achieved a total accuracy of 85.6% when identifying drowning events (Handalage et al, 2021). The model was trained in a Google Colab notebook with a dataset of approximately 5000 images with weight files generated every 100 iterations. The model was then implemented on a NVIDIA Jetson Nano board to run multiple neural models. The authors acknowledged that the system could be improved through better hardware to assist with speed and accuracy as well as the addition of multiple cameras.

Lei et al developed a drowning behaviour detection system utilising the YoloV4 algorithm and eight underwater cameras. The authors adapted YoloV4 to a custom model to improve the accuracy of target detection. The system was based in PyTorch (used in YoloV5) and Darknet (YoloV3) frameworks in Windows 10. Two classes were given (drowning or swimming) and it was trained for 10,000 iterations. They experimented with different versions of Yolo from 3 to 5. They found YoloV4 and V5 performed similarly but V3 struggles due to reflections on the water surface and some swimmers being mistaken for drownings. For instance, the Mean Average Precision (mAP) of YoloV3 was 84.37% and 92.41% and 90.32% for V4 and V5 respectively (Lei et al, 2022). They found that all algorithms struggled with higher density pools, which aligns with Yolo's known shortcomings in detecting dense, small objects with their customised YoloV4 algorithm performing best with different pool angles. They concluded that an accurate real time drowning system was possible using underwater cameras and a customised YoloV4 framework.

Another system focused on inadequate supervision instead of drowning behaviours. This solution collected a dataset of 38,000 images of distracted parents or caregivers around a swimming pool and then implemented three different CNNs for classification and detection. The dataset was split 8:2 for training and testing (30,000 images for training and 8000 for testing) with seven detectable classes chosen, such as 'in the water and distracted' and 'out of the water distracted'. They then went on to develop an alert system in the form of voice alert, pager, or a wearable device. The three CNNs selected were VGG-19, ResNet-50 and Inception-v3 with an accuracy of 94%, 98% and 90% respectively (Cepeda-Pacheco and Domingo, 2022). Given the notification network was implemented using 5G, the authors did

acknowledge that privacy and security could be an issue with potential eavesdropping on the network and access to underage images by malicious parties. They emphasised the need for robust security measures for this system to be an accepted method for child drowning prevention.

A holistic rescue system using machine vision and an intelligent lifebuoy was proposed by Yang et al. They collected a dataset of approximately 3000 images which was split in a ratio of 8:1:1 for training, verification, and testing. The authors made some assumptions given that the lifebuoy was operating in open water and not swimming pools. For example, detection of any human features was considered a drowning situation as people shouldn't be present in these open bodies of water. Since it was not necessary to detect drowning behaviours, the model focused on detection of upper body parts such as the head, arms, and hands. YoloV4 was chosen due to its real time detection abilities. The model was trained on a NVIDIA GPU with a learning rate of 0.00261 and 40,000 iterations. They found the system performed well under testing but found some key limitations, notably that the YoloV4 model was too large to be deployed on the low-cost embedded system on the lifebuoy, thus compression and acceleration of the model was to be followed up in further works (Yang et al, 2021).

A low cost and effective solution using limited computational hardware was proposed by Pavithra et al using a Pi camera, Raspberry Pi 3 and a buzzer for alerting. The system used a timer approach as opposed to identifying drowning behaviours, thus if a person stopped being detected for a certain period, the buzzer would alarm. A R-CNN algorithm was used for detection and trained on a large dataset, though the authors do not specify the particulars of this dataset. They concluded that the system was 99% accurate for detecting missing persons in the water (Pavithra et al, 2021). However, the system had some limitations, including potential false detections, and the timer approach may not always detect drowning events that occur for an extended period before submersion. Additionally, the system's reliance on missing persons detection implies that drowning events will be detected in later stages, such as when the victim is already unconscious, potentially resulting in adverse outcomes.

2.2 Commercial Drowning Detection Systems

There are several commercial camera-based drowning detection systems available. One such system is the Angel Eye which utilises cameras above and below the water to detect all the swimmers in a pool. A three-dimensional model is created, and a drowning detection algorithm activates an alarm based on anomalous events (Angel Eye, 2022). In addition,

alerts are sent to smart watches or phones worn by lifeguards to notify of a potential event. Currently, the Angel Eye is available for commercial swimming pools and each system is uniquely configured for each application requiring a design study and professional installation (Angel Eye, 2022).

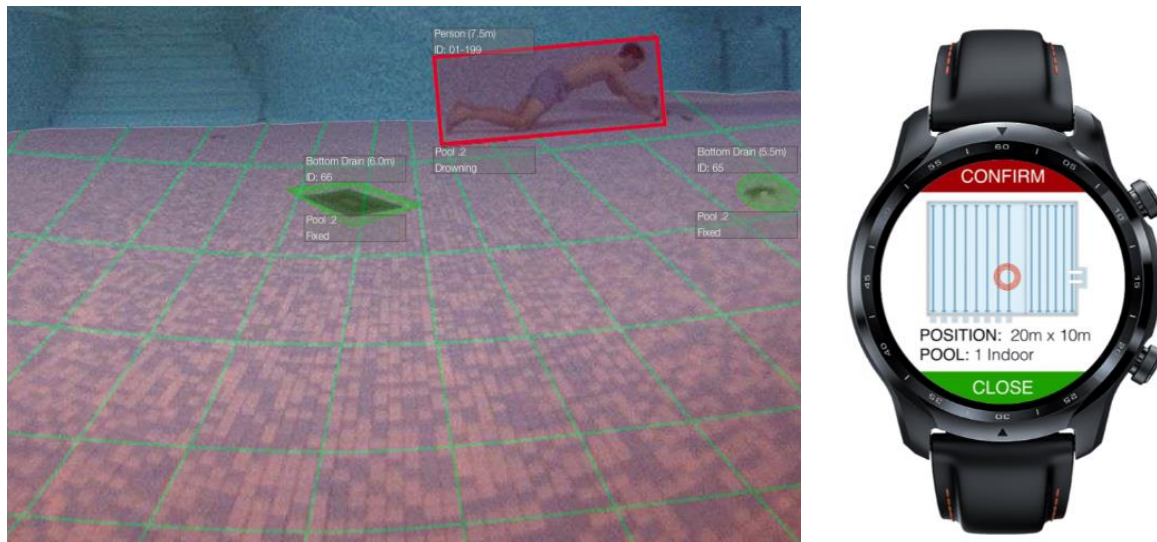


Figure 7a and b: Angel Eye user interface and notification device (Angel Eye, 2023).

Another similar solution available is the Poseidon drowning prevention system. Like the Angel Eye, it utilises cameras above and below the water to identify swimmers in distress which then alerts lifeguards. The company states that ‘unique and precise’ software code is used to differentiate between a person drowning and simply being still (Poseidon, 2022). Also like the Angel Eye, each system is custom built for the swimming pool it is installed in. A clear disadvantage of both solutions is that they are bespoke systems suited for commercial swimming pools in addition to requiring cameras above and below the water to accurately identify swimmers in distress.

Wave drowning detection systems offer a commercial and home-based drowning detection system utilising wearable sensors which link to a local hub and then to a mobile phone application (Wave drowning Detection Systems, 2022). One key advantage of this system is that is easy to configure and can be used in a variety of settings including swimming pools and lakes. However, as previously noted it requires the effective use of wearables which can often be problematic among children due to discomfort. Thus, there is currently no market ready solution utilising one camera above the water line in a swimming pool to detect potential drowning events, particularly in non-commercial settings.

2.3 Drowning Detection Using Machine Vision

Numerous prospects exist for drowning detection systems through the utilisation of machine vision, and the progress in object detection algorithms further enhances these possibilities. This section shall assess machine vision solutions including the hardware and algorithms used in their implementation.

2.3.1 Hardware

Many of the drowning detection systems reviewed utilised computers such as the Raspberry Pi for deployment in field. Pavithra et al used a Pi Camera and Raspberry Pi for their drowning detection system which was able to detect and track swimmers in a pool. Alshbatat et al also made use of a Raspberry Pi and Pixy cam in their system which communicated with an Elekstar controller for deployment of a rescue system. It should be noted that their drowning detection system was simplified to identify swimmers based on people wearing yellow vests. Thus, detection of a yellow object is assumed to be a swimmer. In addition, the system was concept tested in a laboratory only and not in a swimming pool. Nonetheless, the results from lab experiments were promising and proved the concept.

In addition to hardware required for deployment of drowning detection systems, further hardware is often required to train the algorithms. Graphical Processing Units (GPUs) are now widely used in machine learning applications due to their improved performance over Computer Processing Units (CPUs). In one experiment conducted using machine learning for webpage classification, GPUs were found to complete test cases 4 to 5 times faster than CPUs (Buber and Diri, 2018). In addition, it was found that running time was shortened as the number of cores was increased in the GPUs.

Handalage et al utilised Google Colab and the freely available virtual GPUs to create and train their models, saving weight files every 100 iterations. Once the model was trained, it was implemented on a NVIDIA Jetson Nano board running a Quad-core ARM CortexA57 processor. The Jetson nano was also equipped with a 128 Core GPU making it suited to machine learning applications and running neural models in parallel (Handalage et al, 2021). Other researchers followed a similar methodology, Yang et al trained a custom YoloV4 model on a NVIDIA GeForce GTX1080 GPU (Yang et al, 2021) and Niu et al also used a NVIDIA GeForce RTX 2080Ti GPU (Niu et al, 2022).

2.3.2 Algorithms

Both Yolo and CNN algorithms have been used in drowning detection systems and both offer promising results. There are advantages and disadvantages to each type of algorithm and this section shall review their use in drowning detection systems.

Five types of CNN algorithms were assessed for their effectiveness in a drowning detection system by Shatnawi et al in 2023. They selected SqueezeNet, GoogleNet, AlexNet, ShuffleNet and ResNet 50 for testing on their dataset. Their dataset was limited to 200 images taken from Google and two classes were to be detected: drowning and swimming. To avoid overfitting, they utilised data augmentation techniques to increase the quantity of imagery in their dataset. Rotation and scaling were found to be effective methods. Their experiments were implemented in MATLAB and six metrics were used to evaluate the various models. These were accuracy, recall, precision, specificity, F1 and the Mathew Correlation Coefficient (MCC). It is notable that no metric for measuring the speed of detection was used. They concluded that ResNet 50 performed the best with accuracy and training time (Shatnawi et al, 2023).

Another approach used pose estimation algorithms. Firstly, OpenPose was used to label joints on images of swimmers. VGG-19, a CNN used in OpenPose was replaced with Thin-MobileNet to reduce the size of the network model and calculation times (31 seconds to 14). A shallow Recurrent Neural Network (RNN) was then used to recognize drowning actions (Jian and Wang, 2021). The authors found that the model had an accuracy of 89.4% though there was the potential for detections to be missed due to bubbles generated by the swimmer. Again, no reference was made to detection times in the final testing of the model.

Speed and accuracy are critical measures in a real time drowning detection system. Consequently, it is important to compare these metrics in various algorithms. R-CNN and Fast R-CNN are known for their accuracy and ability to detect small objects, a feat that Yolo algorithms often struggle with. However, the former are less suited to real time applications due to the speed of detection, an area where Yolo excels (Malhotra and Garg, 2020). Joseph Redmon, the initial creator of Yolo, acknowledges the lagging accuracy of Yolo when compared to CNNs like RetinaNet (an AP of 33.0 compared to RetinaNet's 40.8). However, he points out that RetinaNet takes 3.8 times longer to process an image (Redmon and Farhadi, 2018). Since his final version of Yolo (YoloV3) there have been significant improvements in the subsequent iterations of the Yolo models. As table 1 below

demonstrates, CNN models generally perform better than Yolo models with regards to mAP metrics, however, cannot match Yolo in speed.

| Method | FPS | mAP@0.5 | mAP@0.5,0.95 |
|-------------|------|---------|--------------|
| Fast RCNN | 0.03 | 35.9 | 19.7 |
| Faster RCNN | 5 | 42.7 | 21.9 |
| Mask RCNN | - | 62.3 | 39.8 |
| YoloV2 | 40 | 44.0 | 21.6 |
| YoloV3 | 45 | 51.5 | 28.2 |
| YoloV4 | 31 | 64.9 | 43.0 |
| YoloV5 | 140 | 68.9 | 50.7 |

Table 1: A performance comparison of object detection models (Kaur and Singh, 2022)

It is not to say that CNN object detection models do not have their place, for instance they are well suited to medical imaging detections where real time speeds are not necessary (Kaur and Singh, 2022). However, where real time speeds are required, Yolo models are the most suitable choice (Malhotra and Garg, 2020).

Niu et al adapted a YoloV4 framework to a custom model. YoloV4 was the algorithm of choice as it is a single stage target detection algorithm with excellent speed and accuracy. They replaced the ReLu functions with a Meta-CON activation function in conjunction with a CBAM module. They then tested this model on a constructed swimming video dataset. They found their model performed marginally better than the YoloV4 alone with a mAP of 86.92%, which was an increase of 1.82% on the original method (Niu et al, 2022). The dataset contained a total of 12588 images (both above and underwater) which had been taken from footage and labelled using Labellmg. Training and testing were performed at a ratio of 9:1 with both CPUs and GPUs being used. The dataset contained people swimming as well as life savers simulating drowning behaviours.

Yang et al also selected YoloV4 as their algorithm of choice and customised it to suit their needs. They introduced a new output scale from the low-level feature map layer to improve the detection of small objects. They also improved the non-maximum suppression of YoloV4, which reduced the number of bounding boxes resulting in more accurate positioning information for their intelligent lifebuoy (Yang et al, 2021).

YoloV5 was used to create a drowning detection system using surface and underwater cameras. It was found to be the most suited algorithm for their needs as real time detection was a requirement (Vestinov et al, 2023). The authors considered several other versions of

Yolo, including YoloV7, YoloR and YoloX but after their initial experiments, found YoloV5 was the best for their purposes. Again, dataset acquisition was a challenge due to the nature of drowning events, but they utilised augmentation techniques to increase the size of the dataset. After the model was trained, the team tested the system by simulating drownings and found that their model successfully detected all events. They did note some limitations because of their dataset. For example, an underwater vacuum cleaner was consistently identified as a person and false positives occurred in night frames (Vestinov et al, 2023).

As previously mentioned, Handalage et al used YoloV3 for identifying swimmers and then a CNN determined if they were exhibiting drowning behaviour using OpenPose. Their system had an accuracy of 85.6% however they did not refer to any detection times (Handalage et al, 2021). Lei et al also utilised a customised YoloV4 and underwater cameras for identification of drowning behaviours having experimented with various versions of Yolo (from 3 to 5). Their results found that YoloV4 performed best and was capable of being used in a real time drowning detection system (Lei et al, 2021). Another previously mentioned solution by Yang et al also utilised YoloV4 for detection of swimmers in distress, again it performed well though was limited in deployment by its size (Yang et al, 2021). Thus, it is clear from the reviewed literature that Yolo is capable of being used in a real time drowning detection system. In addition, little research has been conducted beyond YoloV5 in these systems and as such, opportunities exist to investigate this further.

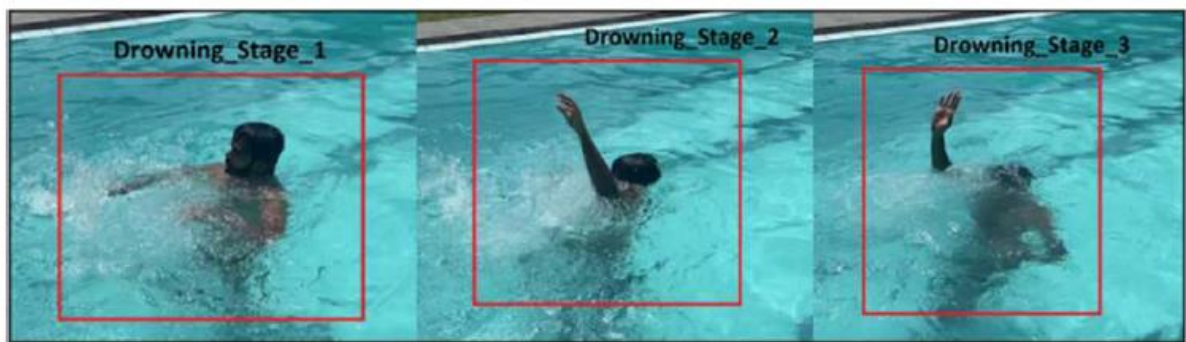


Figure 8: Drowning stages detected by the Yolo algorithm used by Handalage et al (Handalage et al, 2021).

2.3.3 Datasets

In object detection, there are several open-source datasets which have been used as a benchmark for training algorithms. Until 2017, the de facto standard was the Pascal VOC dataset. This contained 20 object classes through 17,125 images. However, since 2017, the benchmark has become the Microsoft COCO (Common Objects in Context) dataset (Miller et al, 2022). The COCO dataset contains 80 object categories with the 2017 dataset being split

into three different categories. These are train, which contains 118,000 images, validation which contains 5,000 images and test which contains 20,000 images. The benefit of the COCO dataset is that it provides standardised metrics including mean Average Precision (mAP) which can be used to compare various object detection models (Jocher and Waxmann, 2023), a leaderboard on the dataset website compares the performance of a variety of models.

A limitation of many of the systems proposed was the availability of a dataset on which to train a detection algorithm. By nature, drowning events are unpredictable and uncommon, and so there was a reliance on mimicking drowning behaviours to train models. In addition, reasonably large datasets are required for meaningful results, often in the order of 10,000 to 40,000 images as indicated by the reviewed literature. Thus, acquisition or creation of an appropriate dataset will be a key challenge of this research.

Generation of a dataset for drowning behaviours was proposed by Hasan et al using both surface and underwater cameras. This dataset was tested using algorithms to detect the early stages of drowning rather than looking for a drowned person with the underwater camera. Drowning recognition is achieved using Deep Neural Networks (DNNs) which have been pretrained on a large image set, which formed part of their dataset. These were then adapted for water behaviour recognition for the purpose of identifying behaviours associated with drowning. Two methods were then used to train deep learning models for drowning detection: scene recognition and pose estimation methods with pose estimation found to be the most effective for drowning detection (Hasan et al, 2021). Finally, a test dataset was used to assess the effectiveness of the models used and their dataset for training them.



Figures 9a and b: Drowning and Swimming detections using ResNet50 (Hasan et al, 2021).

Given the sporadic nature of drowning events and the challenges in acquiring such a dataset highlighted in this review, a gap exists in the development of a suitable dataset for drowning behaviours. There is an opportunity to investigate the size needed to effectively train a machine vision model for drowning detection.

2.4 Machine Vision Developments

Many machine vision systems have been implemented using standard cameras and rely on peripheral CPUs or processors to for object detection. However, there have been developments in affordable computer vision hardware. For instance, Luxonis have released an OAK-D (OpenCV AI Kits- Depth) which is a three-camera device with stereo depth and a high-resolution colour camera. In addition, the device contains Neural Network inferencing and computer vision capabilities (Luxonis, 2023). This allows much of the resource intensive neural network processing to be done on camera rather than on a peripheral device, such as a Raspberry Pi potentially offering better real-time response with neural network models.

The OAK-D was a crowdfunded camera released in 2021 and has since been the centre of an annual competition hosted by OpenCV and Microsoft attracting 1400 submissions a year (Bliss, 2021) with users demonstrating implementation of the Oak-D in their projects.



Figure 10: The Oak-D Lite edge AI camera. In the centre is the colour camera with two stereo cameras either side (Luxonis, 2023).

One novel use of the Oak-D camera was in a non-invasive real-time monitoring system in neonatal intensive care units in Spain named the Neocam. The system proposed used an Oak-D due to its on-board processing and ability to perform several video analysis tasks of clinical interest in real-time speeds (Ruize-Zafra et al, 2023). They used several algorithms for various tasks, for body and face detection two CNNs (ResNet) were used, for facial expression classification MobileNetV2 was implemented and for pose estimation BlazePose body was selected. The use of these networks allowed for monitoring of infant breathing rate

(due to the mono-cameras of the OAK-D), motor activity monitoring and emotional status. Their system had positive results, with the algorithms for face and pose detection functioning correctly 90% of the time (Ruize-Zafra et al, 2023). A survey conducted amongst clinical staff found that they considered the system to be safe and that they would like more time to work with it. In addition, the breathing rate detected by the camera was within 6% of measurements obtained by probes. Some of the system's flaws were that it struggled with infants covered or partially covered by blankets or sub-optimal lighting levels. In addition, key concerns from staff were related to privacy and image control. In all, the Neocam demonstrated the impressive capabilities of the Oak-D in a practical setting.

Another system implementing the Oak-D was a computer vision-based assistance system for the visually impaired. The OAK-D allowed the system to avoid using expensive and power intensive GPUs normally needed for deep learning algorithms. As such, it was capable of being worn inconspicuously by the intended user and was non-intrusive. The system was able to detect people, cars, traffic lights, yellow pavements that aid the blind, traffic signs, pedestrian crossings, and speed limits. In addition, the semantic image segmentation models were able to detect roads, curbs, and road markings (Mahendran et al, 2021). SSD-MobileNet was used for object detection and a hybrid lightweight semantic segmentation model was used for area detection. Five open-source datasets were used for training including the Google Open Image dataset, in addition the authors collected and labelled several thousand custom images from walking around their local area at various times of the day (Mahendran et al, 2021).

The hardware included the OAK-D connected to a small host computing unit such as a Raspberry Pi which was able to be placed in a backpack and the camera was embedded in an appropriately designed vest. The authors noted that the semantic image segmentation models were not able to be effectively run in parallel with the other models selected and as such were only run at the user's request. A proposed solution was to obtain more OAK-D cameras so it would be possible to run segmentation models in parallel, however at the time of publication the OAK-D was still a Kickstarter project and supply was limited (Mahendran et al, 2021). However, the system addressed common challenges experienced by the visually impaired on a daily basis and was also a non-obtrusive design. In addition, the authors were confident that the computing device could be eliminated and replaced with a mobile device or edge device such as a Nvidia Jetson.

At the time of writing, there was no literature indicating that a drowning detection system has been proposed or designed utilising devices such as the OAK-D Lite. As such, there is an opportunity to investigate this.

2.5 Knowledge Gaps

Having conducted the literature review, several knowledge gaps were identified which shall form the foundation of this research project. They are listed below:

- 1) There are no solutions which utilise only surface cameras in a non-commercial setting such as a residential pool.
- 2) Yolo algorithms have proven to be effective in drowning detection systems, however little research has been conducted beyond YoloV5. As a result, there is an opportunity to investigate newly released algorithms beyond YoloV5 in a drowning detection system.
- 3) Drowning datasets have proven to be a major challenge identified in all the systems studied. A knowledge gap exists in the acquisition and development of a suitable dataset to effectively train an object detection model.
- 4) Affordable cameras with embedded AI processing capabilities are now available. The literature review indicated that cameras such as the Oak-D have not been utilised in drowning detection systems. As such, an opportunity exists to explore this knowledge gap.

3 Methodology

Several knowledge gaps have been identified in the literature review. Experiments shall be developed and conducted to address these. To reiterate, the gaps are as follows:

- 1) *Surface Cameras*: There are no solutions which utilise only surface cameras in a residential pool and existing market ready solutions are bespoke designs for commercial swimming pools. Development of a simple surface camera for use in a residential pool may ultimately reduce drowning events.
- 2) *Yolo Algorithm Developments*: Yolo algorithms have proven to be effective in drowning detection systems, however little research has been conducted beyond YoloV5. As a result, there is an opportunity to investigate newly released algorithms beyond YoloV5 in a drowning detection system. Of particular interest is YoloV7 and YoloV8 which are adaptations of YoloV4 and V5 respectively.
- 3) *Dataset Acquisition and Development*: Drowning datasets have proven to be a major challenge identified in all the systems studied. A knowledge gap exists in the acquisition and development of a suitable dataset to effectively train an object detection model. High quality, diverse and often large datasets are needed to effectively train machine vision algorithms. The acquisition and development of such a dataset shall be investigated in this project.
- 4) *Cameras with Embedded AI functionality*: Affordable cameras with embedded AI processing capabilities are now available. The literature review indicated that cameras such as the Oak-D have not been utilised in drowning detection systems. As such, an opportunity exists to explore this knowledge gap. Of particular interest is the Oak-D Lite which is a lightweight version of the Oak-D at a lower cost.

To reduce the scope of this project, an Oak-D Lite camera has been selected as the surface camera for use. This shall assist in addressing the first and last knowledge gaps as well as the fifth research question ‘What is the most appropriate hardware to implement the algorithm on to build a real time drowning detection system?’. This project shall focus on the acquisition and development of a suitable dataset as well as an appropriate Yolo algorithm for the drowning detection system.

3.1 Research Methodology

This section shall develop a methodology and specific experiments which aim to close the knowledge gaps identified and answer the project research aims.

3.2 Task Analysis

The research task can be broken down into several distinct stages.

3.2.1 Dataset Acquisition and Preparation

The first stage of the project is to acquire and prepare a dataset. It may also be necessary to create my own dataset if a suitable one cannot be sourced. The dataset will also need to be appropriately labelled for the algorithm which will be trained. The literature indicates that LabelImg is a free and open-source program which can be used for this purpose.

3.2.2 Metrics

It is necessary to define metrics and parameters which shall be used to measure the effectiveness of the chosen models to address the sixth research question ‘How can the effectiveness of the system be measured?’. Key Metrics which shall be used are as follows:

- i) Mean Average Precision (mAP) - This metric shall be used to quantitatively compare the different models and their capabilities. This will also assist in selecting the best performing weight file once a model has been trained.
- ii) Speed of Detection – This shall be measured in milliseconds. It is important to know the time it takes for models to run their detections as this will be a critical component of a real time drowning detection system.
- iii) Average Precision (AP) – this metric is used to compare precision and recall in single classes. It will be important for understanding model’s performance in detail.
- iv) True Positive (TP) – The correct detection of a bounding box which will assist in gaining an in depth understanding of a model’s performance.
- v) False Positive (FP) – The incorrect detection of a non-existent object or the misplaced detection of an existing object, again this will assist with understanding a model’s performance and the quality of the dataset.
- vi) False Negative (FN) – A bounding box that failed to be detected around an object, as per the previous two metrics.

3.2.3 Algorithm Selection

The second stage of the research and first experiment is to select an appropriate Yolo algorithm. It is important that the algorithm is accurate whilst being deployable on the Oak-D Lite and Raspberry Pi. Key features needed are:

- 1) Precision – the algorithm needs to be accurate and capable of meaningful detections.
- 2) Speed – the algorithm shall be able to make detections in real time.
- 3) Size – the algorithm shall be deployable on the Oak-D Lite and the Raspberry Pi.

It may be appropriate to select two or three algorithms and then compare their relative performances against the above specifications.

3.2.4 Build, Train, and Deploy Model

Once the algorithms have been selected, they shall be built in Google Colab and trained on the acquired dataset using the virtual machines and GPU available through Colab. The models shall use appropriate measures such as mAP and speed to select the most suitable. If necessary, improvements will be made to the final model to increase mAP and speed capabilities.

Once the final model has been selected, it will then be deployed onto the Oak-D Lite and Raspberry Pi and tested in real time. This testing will be qualitative in nature to determine its accuracy and effectiveness in real time.

3.3 Experimental Design

Model testing and development will involve several experiments. The design of these is as follows.

3.3.1 Experiment 1 – Qualitative Testing of Yolo Models

The first experiment will involve deploying pretrained Yolo Models. The baseline model shall be YoloV3 as this is the final version developed by Joseph Redmon, the initial creator of Yolo. Versions 7 and 8 shall also be tested. These models will be trained on the COCO dataset and two images will be used for comparison of the model's effectiveness. These images will contain objects which are included in the COCO dataset (person, snowboard, skis, backpack, skateboard). The main aims of this experiment are:

- 1) To gain an understanding of how these models are built and deployed in Google Colab.
- 2) To qualitatively compare the abilities of each model and to gain an understanding of their object detection capabilities and assist in the selection of models for further testing.

Hypothesis

The expected outcome of this experiment is that larger and newer Yolo versions will have greater detection capabilities. It is expected that YoloV3 will perform poorly in comparison to later versions.

This experiment shall also assist in answering research question 4, 'What method and hardware should be used to train the algorithm?'. This experiment shall confirm whether Google Colab is the appropriate platform for training.

3.3.2 Experiment 2 – Quantitative Testing of Yolo Models on a Custom Dataset

Two models shall be selected for this experiment; YoloV7 and YoloV8. YoloV7 shall be selected as it was developed by the team behind YoloV4 and thus is likely to succeed any versions of it. In addition, the release of YoloV7 included a peer reviewed academic paper. YoloV8 shall also be tested, it was developed by Ultralytics who created YoloV5, there was some criticism around the release of YoloV5 as it did not have a peer reviewed paper accompanying it. However, the literature review indicates that it has been widely implemented with good results. Both YoloV7 and V8 are new models having been released in July 2022 and January 2023 respectively. Thus, they offer an exciting opportunity to

investigate their capabilities. The figure below compares the various later versions of Yolo for speed and mAP50-95.

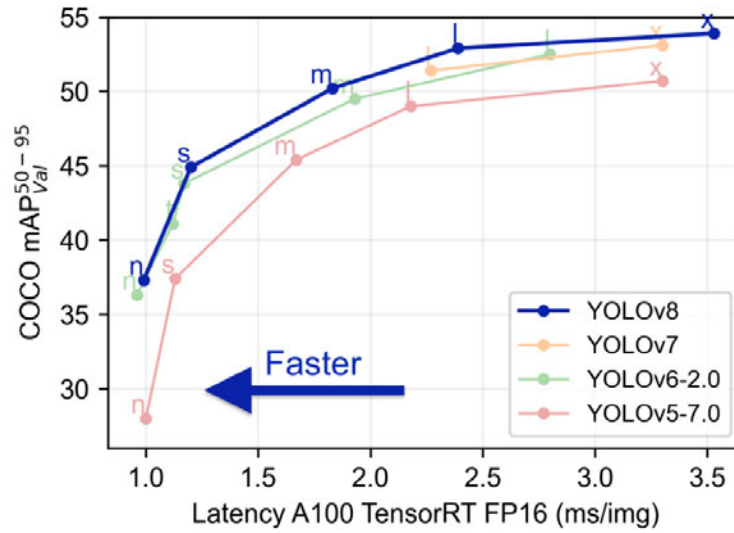


Figure 11: A comparison of speed and mAP of Yolo versions 5 to 8 (Ultralytics, 2023).

Additionally, both YoloV7 and V8 are compatible with the RCV2 architecture of the Oak-D Lite and can therefore be deployed in field.

Mean Average Precision (mAP) and speed shall be the metrics measured during this experiment. The aims of this experiment are as follows:

- 1) To compare three models trained on the custom drowning dataset and their capabilities.
- 2) To select the best model for deployment on the Oak-D Lite. mAP and speed shall be the key metrics.
- 3) To achieve mAP scores similar to or better than those achieved in the COCO dataset baseline testing.
- 4) To observe the effectiveness of the custom dataset when used to train a Yolo model.

Hypothesis

It is predicted that the YoloV7 and V8 models shall perform well. Literature indicates that YoloV8 slightly outperforms V7 and so the same is expected in this experiment.

This experiment shall answer the first research question, ‘What is the most effective algorithm that can be used in a real time drowning detection system?’.

3.3.3 Experiment 3 – Improving the Yolo Model

Having selected the most appropriate Yolo model, this experiment will be focused on improving mAP scores and speed. This may be through improvement of the dataset or other techniques such as fine-tuning model parameters and training for more iterations/epochs. The aim of this experiment is as follows:

- 1) To improve the performance of the Yolo model. This will be observed through changes in the mAP and speed.

Hypothesis

It is predicted that the methods implemented in this experiment will result in some improvement in the performance of this model.

3.3.4 Experiment 4 – Model Deployment on the Oak-D and Raspberry Pi

Having selected and improved the algorithm of choice. The next step is to deploy the model on the Oak-D and Raspberry Pi. Once deployed, the model shall be tested in real-time at a swimming pool to confirm it is able to make accurate detections. The aim of this experiment is as follows:

- 1) To deploy the chosen algorithm on the Oak-D Lite and Raspberry Pi and make real time detections.

Hypothesis

It is predicted that the system can make accurate detections in real time as the algorithm will have been selected with the end deployment in mind.

3.3.5 Experiment 5 – Real time detection and alert of drowning events

Once the model has been deployed effectively on the Oak-D Lite and Raspberry Pi, it will be necessary to make real time detections and alert of any potential drowning events. It is envisaged that the alert system will be in the basic form of a web browser or mobile interface. The aim of this experiment is as follows:

- 1) To prove that the algorithms and hardware is capable of real time drowning detections and alerts.

Hypothesis

It is predicted that the system will be capable of making real time detections and that development of a basic alert system is possible. Ultimately creation of an application for use on smart devices would be desirable, however this is beyond the scope of this project. A simple alert system shall be sufficient to address the research aims and knowledge gaps.

3.4 Resource Analysis

Several resources are required for successful completion of this project. Budgetary constraints have not been a key restriction of this project though efforts have been made to minimise costs where possible. The following resources have been selected.

3.4.1 Hardware

Oak-D Lite: The Oak-D Lite has been selected as the camera to be used in this research project. The Oak-D has produced promising results in systems reviewed in the literature. In addition, these cameras have not yet been used in a drowning detection system. The Oak-D Lite was specifically selected due to its lower price with the specifications being appropriate for this project. The Oak-D Lite is built upon the Robotics Vision Core 2 (RVC2) which can run several AI models, including custom models (Luxonis, 2023). The performance of the following Yolo models on the RCV2 are tabulated below:

| Model Name | Size | FPS | Latency (ms) |
|------------|---------|------|--------------|
| YoloV6n R2 | 416x416 | 65.5 | 29.3 |
| YoloV6n R2 | 640x640 | 29.3 | 66.4 |
| YoloV6t R2 | 416x416 | 35.8 | 54.1 |
| YoloV6t R2 | 640x640 | 14.2 | 133.6 |
| YoloV6m R2 | 416x416 | 8.6 | 190.2 |
| YoloV7t | 416x416 | 46.7 | 37.6 |
| YoloV7t | 640x640 | 17.8 | 97 |
| YoloV8n | 416x416 | 31.3 | 56.9 |
| YoloV8n | 640x640 | 14.3 | 123.6 |
| YoloV8s | 416x416 | 15.2 | 111.9 |
| YoloV8m | 416x416 | 6 | 273.8 |

Table 2 Yolo models and their performance on the Oak-D Lite (Luxonis, 2023)

Raspberry Pi 4: A Raspberry Pi 4 has been selected as the central computing unit for the drowning detection system. Raspberry Pi units have been used extensively in machine vision projects and are compatible with the Oak-D Lite. In addition, it is supported by a large amount of documentation and is simple to use. A Raspberry Pi 4 with 8GB of RAM was purchased as this should offer enough computing power for the purposes of this project.

Microsoft Surface Laptop: A Microsoft Surface laptop with 8GB of RAM and Windows 10 installed shall be used as the PC for development of this research project. Given that the training will be carried out on GPUs in virtual machines in Google Colab this laptop shall suffice for the requirements of this project.

3.4.2 Software

LabelImg: LabelImg is an open source and freely available program used for labelling datasets in PASCAL and YOLO formats. As such it shall be used in this project for labelling images forming the dataset.

Google Colab: Google Colab shall be used for training and development of algorithms. It has been chosen due to its compatibility with Google Drive as well as offering free access to GPUs which are required for training algorithms. Additional GPU access can be purchased if required. In addition, reviewed literature indicated it has been used effectively in past machine vision project with good results.

Google Drive: Google Drive shall be used alongside Google Collab for development of machine vision models as it offers a convenient platform to upload datasets. It offers 15GB of storage for free and the option of upgrading to more storage if needed for a negligible price.

Visual Studio Code: Visual Studio Code shall be used for creating and editing python files associated with algorithm development. It has been selected due to its ease of use and being freely available.

Windows Suite: The windows suite of programs including Word, Excel and Visio shall be used extensively through this project for collation and recording of results and for word processing and image creation.

Free Video to JPG Converter: Again, this software is freely available to download. This will be used to split the dataset videos into JPG files in preparation for labelling.

3.4.3 Dataset

Access to a drowning detection dataset was freely provided by the team at the Rochester Institute of Technology in Dubai used in their paper entitled ‘A Water Behaviour Dataset for an Image-Based Drowning Solution’ which addresses the third research question; ‘Where can a dataset be sourced?’.

3.4.4 Test Location

Access to a pool for testing is required. Access to such a pool is freely available at the unit complex I am a resident of, restricted access to this location is not envisaged and as such does not pose a risk to the completion of this project.

3.4.5 Project Cost

The project costs have been tabulated below:

| Item | Cost | Comments |
|--------------------------------|------------------|---|
| Raspberry Pi 4 8GB starter kit | \$ 289.00 | Included accessories such as a power supply and USB leads |
| Oak D-Lite Camera | \$ 287.00 | Includes USB A to C cable for camera |
| Laptop | \$ - | Currently own an appropriate laptop |
| Labelling | \$ - | Free to download and use |
| Google Collab | \$ - | Free to access and use |
| Google Drive | \$ - | Free to access and use |
| Visual Studio Code | \$ - | Free to access and use |
| Windows Suite | \$ - | Free to access and use |
| Free Video to JPG Converter | \$ - | Free to access and use |
| Dataset | \$ - | Freely provided by Rochester Institute of Technology |
| Test Location | \$ - | Access is freely available |
| Total Estimated Cost | \$ 586.00 | |

Table 3: Project Costs

A total project cost of \$586 is largely consumed by the Raspberry Pi 4 and Oak-D Lite camera. The cost is not prohibitive and is able to be self-funded for the purposes of this research.

3.5 Project Consequential Effects

The consequential effects of this project encompassing sustainability, ethics and risks have shall be discussed and assessed in this section. A cornerstone used for guiding the impacts of this projects is the Engineers Australia Code of Ethics. The guideline highlights four key areas, which are as follows:

- 1) Demonstrate integrity.
- 2) Practice competently.
- 3) Exercise leadership.
- 4) Promote sustainability.

(Engineers Australia, 2023)

3.5.1 Sustainability

This project shall promote sustainability in as far as it is applicable. This includes any effects of the project which have the potential to adversely impact the health, safety, and well-being of the community (Engineers Australia, 2023). Though this project shall be largely limited to demonstrating proof of concept of a real time drowning detection system, should the project be successful in its research aims, it offers an additional tool for lifesavers and in pool supervision. Ultimately, such a system is designed to be used in conjunction with existing controls, but it would offer a promising and potentially affordable opportunity to further reduce the risks and impact of drowning events.

3.5.2 Ethics

There are several ethical considerations surrounding this project. Two key considerations are privacy (including images used in the dataset) and use of the drowning detection system which directly tie in with two principles from the Engineers Australia Code of Ethics; demonstrating integrity and practicing competently.

The dataset provided by Rochester Institute of Technology release agreement contained the following conditions which shall be adhered to:

- 1) The dataset will not be further distributed, published, copied, or further disseminated in anyway or form whatsoever, whether for profit or not. This includes further distributing, copying or disseminating to a facility or organization unit in the requesting university, organization, or company.
- 2) The videos will only appear in technical reports, technical papers, and technical documents reporting on water behaviour research. There will be no more than 8 images used at a time in a publication.
- 3) All documents and papers that report on research that uses the Water Behaviour dataset will acknowledge the use of the Water Behaviour dataset. Use of the Water Behaviour dataset will be acknowledged as follows: "Portions of the research in this paper use the Water Behaviour dataset collected under the Electrical Engineering department at Rochester Institute of Technology Dubai" and citation to:

S. Hasan, J. Joy, F. Ahsan, H. Khambaty, M. Agarwal and J. Mounsef "A Water Behavior Dataset for an Image-Based Drowning Solution," In 2021 IEEE Green Energy and Smart Systems Conference (IGESSC), pp. 1-5, 2021.

A copy of the release agreement has been included in Appendix C.

In addition, the supplementary images used in the dataset, which do not form part of the set provided by the Rochester Institute of Technology, shall only be used for the purposes of this research project. Explicit permission was sought from the volunteers prior to their use, and it was ensured the volunteers understood the scope and use of these images. The images shall not be further distributed without their permission.

It was also decided that the scope of the drowning detection system shall be limited to adults only, and as such images of only adults have been used in the dataset.

Finally, this drowning detection system is a research project and as such shall not be used in place of appropriate supervision in swimming pools. It shall not be distributed or shared beyond what is required to for fulfilment of this research project.

3.6 Risk Assessment

A risk assessment was conducted to identify any hazards and implement controls for these risks.

3.6.1 Risks Identified

Several risks were identified including poor housekeeping resulting in trip and other hazards, adverse weather, and the use of electrical/electronic equipment around swimming pools. Controls were developed and implemented, which have been documented in the risk assessment in Appendix B. In addition, two other critical risks were identified, which are as follows:

- 1) *Working around swimming pools* – Working around swimming pools exposes personnel to wet and slippery surfaces and the potential for drowning. Only competent swimmers are to be used for data collection and there shall be at least one spotter in place whilst persons are in the swimming pools. In addition, the pool shall have standard pool fencing and signage as per Queensland legislation.
- 2) *Use of Project as a Drowning Detection System* – There is the potential for persons to misunderstand the intention of the drowning detection system as a research project and prototype and thus to mistakenly use it in place of appropriate supervision around swimming pools. The project is not to be distributed for public use and it shall be emphasised that this work is to develop a prototype and proof of concept rather than a market-ready drowning detection system.

4. Development and Experimental Setup

This section shall address the development and setup of experiments, including data set acquisition and equipment.

4.1 Dataset Acquisition and Preparation

4.1.1 Dataset Acquisition

As the reviewed literature indicated, a significant challenge of this project would be acquiring a dataset which can be used to train an object detection algorithm. Contact was made with several authors of papers reviewed however only one team was able to provide their dataset. The authors of the paper titled ‘A Water Behaviour Dataset for an Image-Based Drowning Solution’ from the Rochester Institute of Technology in Dubai graciously provided the dataset they developed for this research.

Access was provided to videos filmed using overhead and underwater cameras. For this project, only the overhead camera footage was to be utilised. There were two main folders entitled ‘test’ and ‘train’ and within these were four folders denoting the four actions filmed which were ‘swim’, ‘drown’, ‘idle’ and ‘misc’. The directory was structured as follows:

```
test/
├── te_overhead/
│   └── te_o_label_1.mp4 (9 videos: 4 drown, 2 idle, 3 swim)
train/
├── tr_overhead/
│   ├── tr_o_swim/ 27 videos
│   ├── tr_o_drown/ 18 videos
│   ├── tr_o_idle/ 10 videos
│   └── tr_o_misc/ 17 videos
```

Figure 12: The folder directory structure of the provided dataset with the number of videos

The dataset was inspected and observed to be filmed in different swimming pools at various times of the day. Several people were filmed, and the surface dataset consisted of 47 videos. The persons participating in filming were all males in their early 20s of middle eastern ethnicity (Hasan et al, 2022). A noted limitation of this dataset is the homogeneity of the subjects being filmed. As such, additional footage was added to the dataset to include female subjects of Caucasian ethnicity filmed in Mackay, Queensland.

A further 20 videos were created with females as the subject. Filming was done using a GoPro Hero 7 Black from various angles around the test pool. Two female subjects were used performing the three actions noted in the previous dataset (drowning, swimming, idle). 16 videos were to be used for the training/validation set and 4 were selected for the testing set.

4.1.2 Dataset Preparation

For simplicity, the two video datasets shall be denoted as the ‘Dubai Dataset’ (from Rochester Institute of Technology) and the additional dataset as the ‘Mackay Dataset’. Processing of the datasets followed the same methodology.

The first step was to extract images from the video files. ‘Free Video to JPG Converter’ software was used to do this. One frame was extracted every 10-50 frames. This was determined to be adequate for acquiring a suitable number of images whilst maintaining diversity. For example, extracting every frame would have resulted in a significant number of similar images. The extraction rate varied due to there being an imbalance in the amount of video for each class. For instance, there was more footage provided for ‘swimming’ classes and to keep the dataset balanced, frames were extracted less frequently. This process was repeated for each video. The breakdown of images from each is tabulated below:

| Train | | | Test | | |
|---------|--------------|-------------|---------|--------------|------------|
| Dataset | Class | Images | Dataset | Class | Images |
| Mackay | Swim | 868 | Mackay | Swim | 162 |
| | Idle | 477 | | Idle | 40 |
| | Drown | 246 | | Drown | 117 |
| Dubai | Swim | 955 | Dubai | Swim | 96 |
| | Idle | 966 | | Idle | 114 |
| | Drown | 1012 | | Drown | 70 |
| | Total | 4524 | | Total | 599 |

Table 4: A breakdown of the image dataset

A total dataset size of approximately 5100 images was chosen to strike a balance between practicality and accuracy. During the labelling process, it was possible to label between 300-400 images an hour depending on the number of labels needed in the image. Thus, 5100 images equates to between 13 and 17 hours of labelling time. Though a larger dataset is desirable, the limits of this research project dictate that any larger than 5100 images become impractical given time constraints.

The next step in the preparation process was labelling of the dataset using LabelImg. Once the three classes had been created in the program, it was a simple process of drawing bounding boxes around the objects and selecting the correct label in YOLO format. LabelImg automatically saved the text file associated with the image.

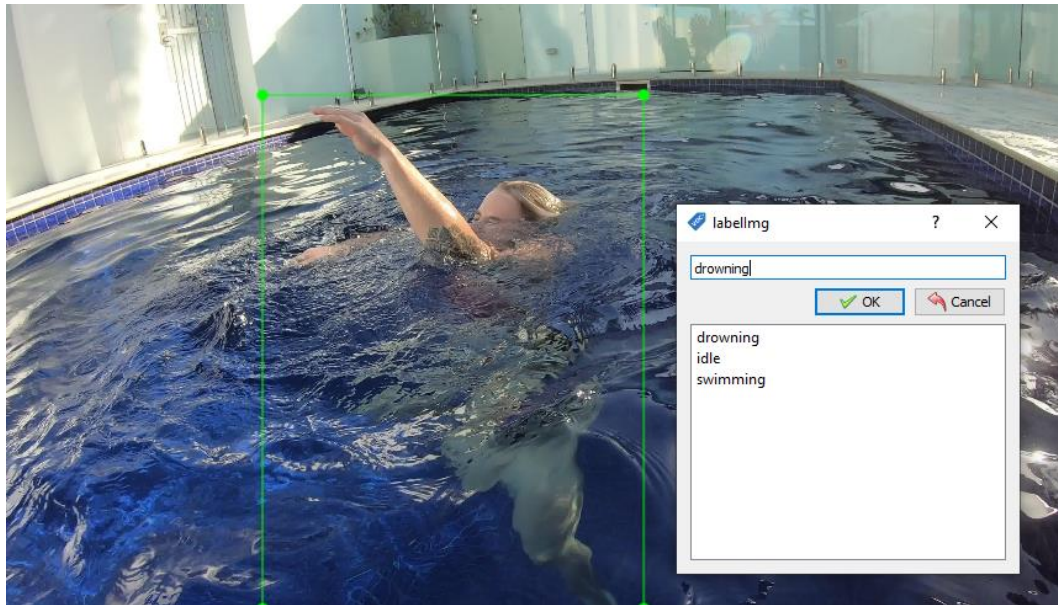


Figure 13: Drawing a bounding box on a training image using Labellmg.

Some key principles were followed in the labelling process as outlined by Nelson (2020):

- 1) *Label all objects of interest* – Thus all persons appearing in images should be labelled with the appropriate class. This avoids the introduction of false negatives within the model.
- 2) *Label the entire object* – In particular, this relates to objects which may be obscured by other objects in the image. It is recommended that the entire object of interest is included in the bounding box, including the blocked section. In addition, bounding boxes can overlap should two objects of interest be in close proximity.
- 3) *Create Tight Bounding Boxes* – it is critical that the entire image is encompassed within the bounding box, however it is also important that excess, irrelevant pixels are not included.

Upon completion of the labelling process, each image containing an object had a text file associated with it. Some images had several objects whilst the occasional one had no objects present. Figure 7 shows a label text file, the first value (an integer) represents the class, in this case 'idle'. The following two numbers are the x and y values of the centre of the bounding box. The final two values are the width and the height of the bounding box. If there were two objects in an image, there would be a line for each in the file.

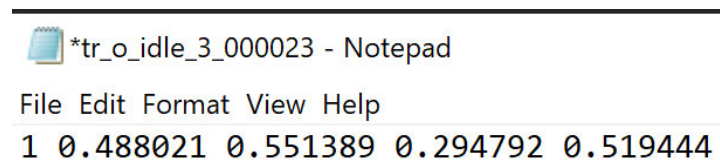


Figure 14: A text file associated with a labelled image.

Earlier versions of Yolo, including V3, require a class file which aligns the first integer of the text file with its class. Later versions of Yolo also require a similar declaration of classes, but this is generally captured in a configuration file and not a separate class file. LabelImg automatically generates the class file, as shown below.

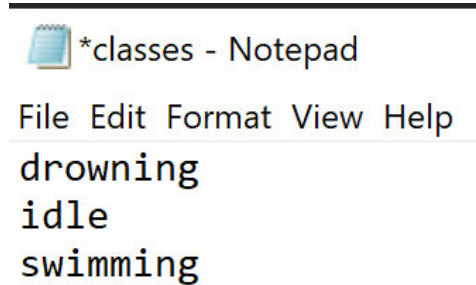


Figure 15: The class declaration file where drowning is 0, idle is 1 and swimming is 2.

4.2 Experiment 1 – Qualitative Testing of Yolo Models

In this experiment, qualitative testing of various Yolo models was carried out. The aims of the experiment were:

- 1) To gain an understanding of how these models are built and deployed in Google Colab.
- 2) To qualitatively compare the abilities of each model and to gain an understanding of their object detection capabilities and assist in the selection of models for further testing.

Google Colab notebooks exist in the model's respective Github repository. The models have been trained on the COCO dataset which can detect 80 object categories. Two images were selected for comparison purposes, their selection was based on them containing several object categories that are part of the COCO dataset on which the models are pretrained on. For reference, the two images have been named 'Kicking Horse' and 'Skateboards'.



Figures 16a and b: The two images for qualitative testing, left 'Kicking Horse' and right 'Skateboards'.

As shown, the first image contains several objects capable of being detected including 'person', 'snowboard', 'backpack' and 'skis'. The second image has the 'person' and 'skateboard' objects.

4.3 Experiment 2 – Quantitative Testing of Yolo Models on a Custom Dataset

Though the initial dataset was split into two test/train groups, an additional subgroup was created from the test data. Images were randomly selected and used as the validation dataset. These images are used in the model training to validate the model's development. Though there is no fixed standard for the ratio split of the dataset, Young et al used a train/validate/test split of 8:1:1. Nui et al used a train/test split of 9:1. Thus, a train/validate/test split of approximately 8:1:1 was used in these experiments. The final image tally is tabulated below.

| Dataset | Number of Images |
|----------|------------------|
| Train | 4003 |
| Validate | 521 |
| Test | 599 |

Table 5: The number of images in each dataset for training and testing.

Though the particulars of deploying each model were slightly different as the versions varied, the general structure of the Google Colab notebook for training a custom model was the same. The following diagram shows the process undertaken.

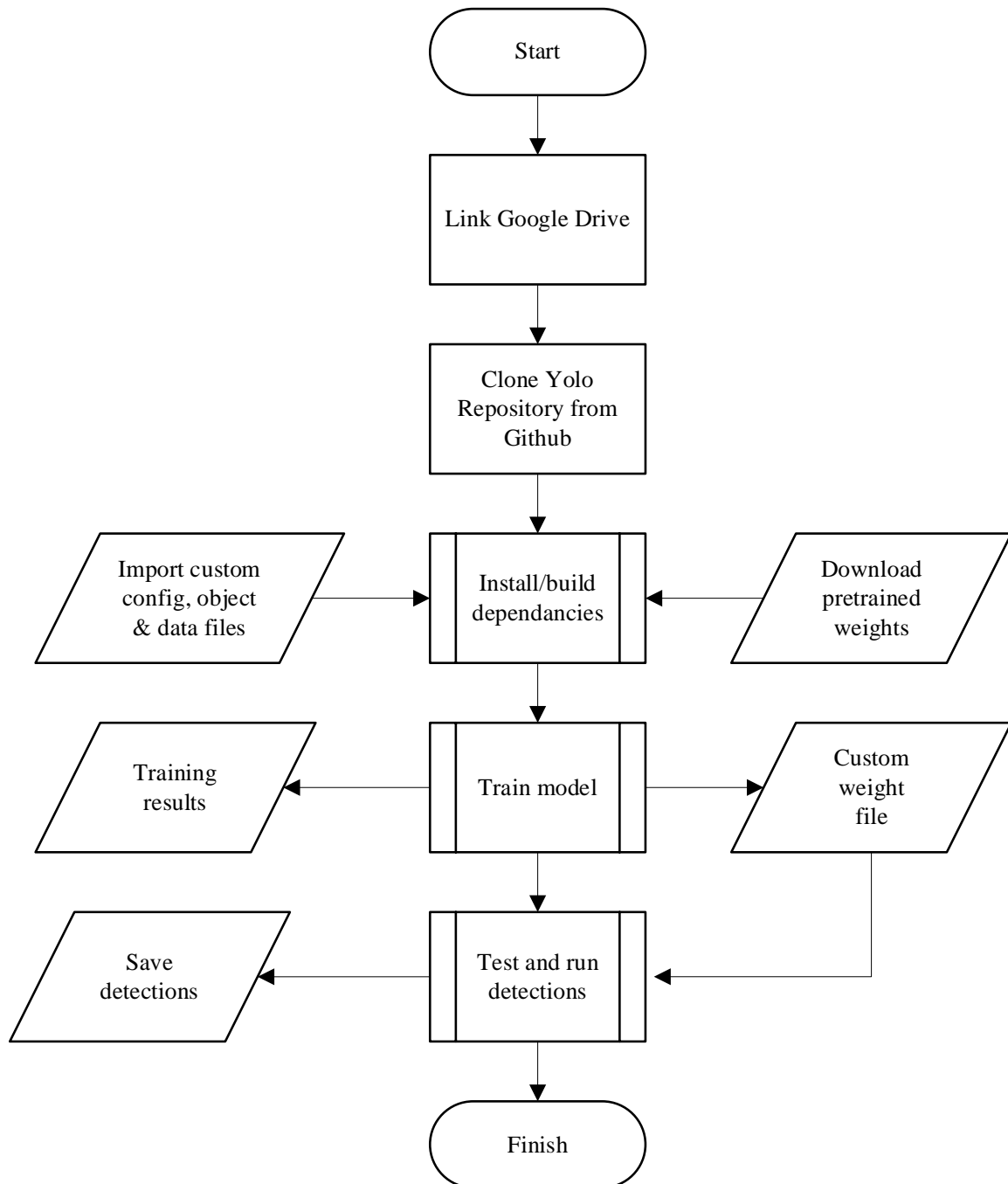



Figure 17: The process for training a custom Yolo model using Google Colab.

4.3.1 YoloV7

Each model required custom data and configuration files. The data file stores the directory addresses of the various image datasets for train, validate and test as shown in the following figure. This file is stored in the model's data folder.

 custom_data - Notepad
File Edit Format View Help
directories for train, validate and test images
train: ./data/train
val: ./data/validate
test: ./data/test

number of classes
nc: 3

class names
names: ['drowning','idle','swimming']

Figure 18: A custom data file for YoloV7.


In addition to the data file, changes needed to be made to the configuration file. In this case, it was changing the number classes from the default value to 3, to reflect training on the custom dataset. No hyperparameters were adjusted from the default in this experiment however some additional parameters were set before starting training. These were the default values taken from the Github repository and were as follows:

| Parameter | Value |
|------------|---------|
| Batch Size | 16 |
| Image Size | 640x640 |
| Epochs | 24 |

Table 6: Additional training parameters for the custom YoloV7 model.

4.3.2 YoloV8

YoloV8 setup was similar to V7. A file declaring the directory addresses for the image sets was created and placed in the data folder. This is shown in the following figure.

 *yolov8custom - Notepad
File Edit Format View Help
#Custom Dataset
path: /content/gdrive/MyDrive/Yolov7/yolov7/data # dataset root dir
train: ./train # train images
val: ./test # val images
test: ./test #test images

Classes
names:
0: drowning
1: idle
2: swimming

Figure 19: The custom data file for YoloV8.

Unlike YoloV7, it was not necessary to declare the number of classes in a configuration file. Several additional parameters were declared prior to starting training. Again, these were the recommended parameters taken from the Ultralytics Github repository and are tabulated below.

| Parameter | Value |
|------------|---------|
| Batch Size | 16 |
| Image Size | 640x640 |
| Epochs | 24 |

Table 7: Additional training parameters for the custom YoloV8 model.

4.4 Experiment 3 – Improving the Yolo Model

YoloV8 was selected as the model to be used in the drowning detection system based on its accuracy and training speeds. Attempts will be made to improve the model’s accuracy using several techniques.

4.4.1 Inclusion of background images

Inclusion of background images may assist in reducing false positives, with approximately 0-10% of the dataset containing background images (Ultralytics, 2023). In the case of the drowning detection dataset, these images would be empty swimming pools. These images were taken from a google images search and manually checked to confirm they contained no people as unlabelled objects could potentially impact the model’s accuracy. 221 images were acquired with 177 for the training dataset, 22 for the validate set and 22 for the test set which equated to approximately 4% of each set containing background images.

4.4.2 Increasing the Model Size

Initial testing was conducted using YoloV8-s to effectively compare it to YoloV7-tiny. However, the Oak-D Lite can run YoloV8-m which offers the potential for further accuracy.

4.4.3 Image Augmentation and Parameter Tuning

Image augmentation is an effective method for increasing the training dataset through techniques such as image rotations, transformations, blurring and other methods. In YoloV8, these parameters can be set as part of the hyperparameter file and will be investigated.

In addition, several parameters can be adjusted which may improve the model’s performance. This includes the batch size and learning rate. Increasing the batch size can lead to faster convergence should processor memory allow. The default batch size is 16. Two additional sizes shall be tested; 32 and 64.

The learning rate dictates how many weights are updated throughout training in response to the model error. Two parameters determine this, $lr0$ and lrf . These are multiplied together to give a final learning rate which by default is 0.0001. A high learning rate can lead to overshooting the best weights whilst a low learning rate can lead to slow convergence.

YoloV8 has several image augmentation options. Evidentially, given the number of variables which can be passed for augmentation, many experiments could be carried out to find the optimum combination. However, to remain within the scope of this thesis, several translational parameters have been selected for experimentation, as guided by Vestinov et al in their 2023 paper. They adjusted HSV values, random resizing of images, random cropping, horizontal flipping, Gaussian blur, and grayscale. In these experiments, HSV values were not changed in addition to perspective, copy-paste and image mix-up due to the nature of the dataset. For example, mixing up an image of drowning and swimming could cause more confusion to the model given their potential similarities and HSV values are set by default. The following values were selected for experimentation as they offer enough variety for meaningful experimentation whilst remaining within experimental scope.

| Parameter | Description | Default | Experiment 1 | Experiment 2 |
|-------------|--|---------|--------------|--------------|
| hsv_h | image HSV-Hue augmentation (fraction) | 0.015 | 0.015 | 0.015 |
| hsv_s | image HSV-Saturation augmentation (fraction) | 0.7 | 0.7 | 0.7 |
| hsv_v | image HSV-Value augmentation (fraction) | 0.4 | 0.4 | 0.4 |
| degrees | image rotation (+/- deg) | 0 | 180 | 90 |
| translate | image translation (+/- fraction) | 0.1 | 0.8 | 0.4 |
| scale | image scale (+/- gain) | 0.5 | 0.8 | 0.4 |
| shear | image shear (+/- deg) | 0 | 180 | 90 |
| perspective | image perspective (+/- fraction) | 0 | 0 | 0 |
| flipud | image flip up-down (probability) | 0 | 0.8 | 0.4 |
| fliplr | image flip left-right (probability) | 0.5 | 0.8 | 0.4 |
| mosaic | image mosaic (probability) | 1 | 1 | 1 |
| mixup | image mixup (probability) | 0 | 0 | 0 |
| copy_paste | segment copy-paste (probability) | 0 | 0 | 0 |

Table 8: The image augmentation parameters to be used during for the model training.

4.4.5 Increasing Training Epochs

Initially, the models were only trained for 24 epochs to gain an understanding of their capabilities within a reasonable time frame. However, as indicated in the literature, training is generally carried out for several hundred epochs. As such, this will be applied to further develop the model and the model shall be trained for 300 epochs.

4.5 Experiment 4 – Deployment on the Oak-D Lite and Raspberry Pi

This experiment shall focus on deploying the final model on the Oak-D Lite and Raspberry Pi and demonstrate proof of concept for an operational system.

4.5.1 Creating a .blob file

Once model training has been completed, a directory of weight files is created. Depending on how the model has been configured, these may be every 100 epochs. All Yolo models automatically select the ‘best’ file based on training parameters as well as the final weight file. The best file, as determined during training shall be deployed on the Oak-D Lite.

Luxonis have created a tool for conversion of the weight files (.pt file) to a .blob and JSON (JavaScript Open Notation) for deployment of Yolo models to Oak devices. This tool was used, and the files downloaded.

4.5.2 Deploying the Model

A python script was then developed for deploying the model including the JSON and .blob files. This script was based on an experiment developed by Depth AI and Luxonis and was available on their Github repository (Luxonis, 2023), the final version is available in Appendix D. The basic program layout is demonstrated in the following pseudo code:

```
# Load configuration - JSON and .blob files
model_path = ".blob file path"
config_path = "JSON file path"
config = load_config(config_path)
nnConfig = config["nn_config"]

# Initialize camera and YOLO model
camera = init_camera(nnConfig)
yolo_model = init_yolo_model(model_path, nnConfig)

# Main loop for detections
while True:
    # Capture the frame
    frame = capture_frame(camera)
    # Make a detection
    detections = infer_yolo(yolo_model, frame)
    # Display the frame and detection
    display_frame(frame, detections)

    # A function for recording, saving and exiting the program
    handle_user_input()
```

Figure 20: The pseudo code for deploying the Yolo model in Experiment 4.

The model was then deployed using the Raspberry Pi and Oak D-Lite camera in field. The experimental setup is shown in the following figure:

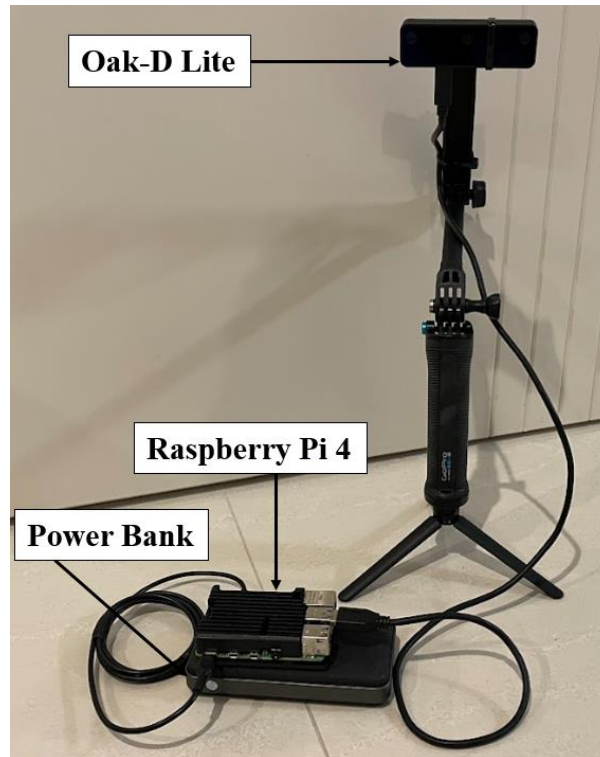


Figure 21: The equipment set up for Experiment 4 - deploying the Yolo model.

A frame rate of 5fps was used with an initial confidence interval of 0.5.

4.6 Experiment 5 – Real Time Detection and Alert

This final experiment is to deploy the real time drowning detection system and create and appropriate alert upon the detection of a drowning event.

4.6.1 Drowning Detection Interface

Improvements shall be made to the initial code used in experiment 4. Firstly, text will be used to alert the system user of a drowning event. This shall be simply displayed in green, if no drowning is detected and if drowning is detected it shall change to red and display ‘Drowning Detected!’. This text will hold for at least two seconds before resetting, i.e., a non-drowning behaviour must be detected for at least two seconds before it will reset. Any drowning bounding boxes shall be red, and idle and swimming bounding boxes shall be green.

```

# Load configuration - JSON and .blob files
model_path = ".blob file path"
config_path = "JSON file path"
config = load_config(config_path)
nnConfig = config["nn_config"]

# Initialize camera and YOLO model
camera = init_camera(nnConfig)
yolo_model = init_yolo_model(model_path, nnConfig)

# Main loop for detections
while True:
    # Capture the frame
    frame = capture_frame(camera)

    # Make a detection
    detections = infer_yolo(yolo_model, frame)

    # Drowning Detection Handling
    drowning_detection(detections)
    if drowning:
        display('Drowning Detected!')
    if not drowning & drowning_timer > 3:
        display('Drowning not Detected')

    # Display the frame and detection
    display_frame(frame, detections)

    # A function for recording, saving and exiting the program
    handle_user_input()
    # r key for record
    # s key for stop
    # q key for quit

```

Figure 22: Pseudocode for experiment 5.

4.6.2 Remote Drowning Detection

A simple Virtual Network Computing (VNC) application was used for creating remote viewing of the drowning detection system. The program used was Real VNC as it was compatible with the Raspberry Pi 4, the apple iPhone as well as the windows PC. Two programs had to be downloaded; the VNC server and viewer on the Raspberry Pi and the VNC viewer on the devices used for remote viewing. The apple iPhone was used as a Wi-Fi hotspot to enable the cloud VNC connection.

4.6.3 Experimental Set Up

In a similar fashion to experiment 4, the Oak-D Lite was mounted on an adjustable camera mount. The Raspberry Pi was powered by a power bank and connected to the Oak-D. Care was taken to ensure the hardware did not get wet during testing. The setup was moved around the pool to test various angles as well as being tested at various times of the day (midday in full sunlight and late afternoon with less sunlight).



Figures 23a and b: The experimental setup with the Oak-D Lite, Raspberry Pi and power bank.

Learnings were taken from experiment 4 to further inform this experiment. The confidence interval was dropped to 0.3 from 0.5 to see if this improved the drowning detection instances. In addition, initial testing was done at lower resolution, this was increased to the maximum (13 megapixels) that the Oak-D Lite RGB camera is capable of. Again, this was done to see if there were noticeable improvements in detections.

5. Results and Analysis

This section shall present observations and results as well as offer analysis.

5.1 Dataset

Observation

Once the dataset was created, analysis was carried out to further understand the class representation. Given that there can be several objects in one image, an imbalance in class occurrence was expected. This is shown in the following figure.

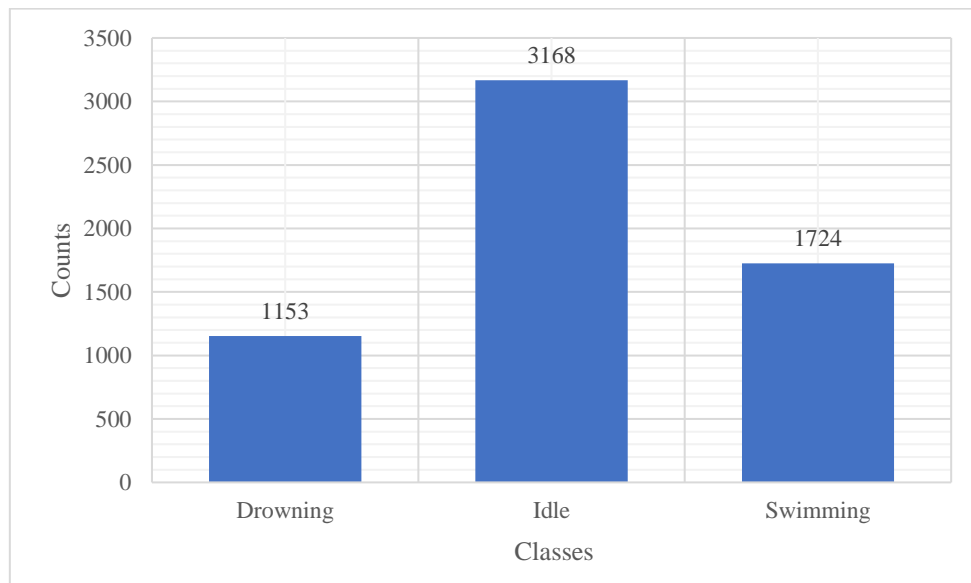


Figure 24: A histogram showing the occurrences of the three classes in the training dataset.

As the histogram shows, idle is significantly over-represented and instances of swimming and drowning occur more than recorded in the initial image split. In total, there are 6045 object occurrences across the three classes.

Analysis

It is not unusual for machine vision datasets to have an imbalance, and there are methods for addressing imbalance through dataset augmentation techniques. Intuitively, it makes sense that 'idle' is over-represented. During the labelling process, many 'swimming' actions initially start in the 'idle' state, for example, a person holding the side of the pool. In addition, in images with several objects present, 'idle' actions were common. The figure below shows two classes being labelled in a 'swimming' training image.



Figure 25: The occurrence of two classes in a training image (swimming and idle).

Understanding the class distribution through the initial training dataset is critical to improving the object detection algorithm and understanding any bias the model may have. As such, there is the opportunity to use data augmentation techniques on the training dataset to improve the final Yolo version.

5.2 Experiment 1 – Qualitative Testing of Yolo Models

The two test images were used to compare the different versions of Yolo models. As mentioned, YoloV3 acted as a baseline with different versions of YoloV7 and YoloV8 being used. The YoloV7 and V8 models selected were all able to be deployed on the Oak-D Lite as per the product specifications.

5.2.1 YoloV3



Figures 26a and b: YoloV3 detections run on the test images trained on the COCO dataset.

Observation

YoloV3 produced good results when run on the two test images. The model was able to make reasonable predictions detecting all the objects present with good confidence. The only notable error was the detection of skis instead of a snowboard.

Analysis

YoloV3 produced encouraging results, particularly with the detection of smaller objects such as the person in the background of the skateboard image. The erroneous detection of a snowboard instead of skis was the only fault and perhaps a difficult detection to make given it is being carried on a backpack. Though it did successfully detect the backpack, the bounding box encompasses the snowboard as well. It should be noted that accurate detections were expected, given this is a larger sized Yolo model. It is too big to be deployed on the Oak-D Lite and the detections are slower when compared to the smaller versions.

5.2.2 YoloV7-tiny



Figures 27a and b: YoloV7-tiny detections run on the test images trained on the COCO dataset.

Observation

The YoloV7-tiny model detected most of the objects in the test images. It was unable to detect the backpack as YoloV3 was. Like YoloV3, it mistakenly detected skis instead of a snowboard in the 'Kicking Horse' image. It also did not detect the backpack as YoloV3 had.

Analysis

Given this is the smallest sized YoloV7 model available, the detections were not dissimilar from the detections made by YoloV3. Though it failed to detect the backpack, all other

objects were detected correctly with the exception of the snowboard. In addition, it was still able to detect smaller objects such as the people in the background of the ‘skateboard’ image. Given this is the smallest YoloV7 model it was expected to be the least accurate. However, its performance is still impressive when compared to YoloV3 with the additional benefit of being deployable on the Oak-D Lite and having real time detection capabilities.

5.2.3 YoloV8



Figures 28a and b: YoloV8-m detections run on the test images trained on the COCO dataset.

Observation

Three versions of YoloV8 were tested; nano, tiny and medium, all of which can be deployed on the Oak-D Lite. The detections made by the models were similar, with a notable difference being an incremental increase in the bounding box confidence as the model size increased. Again, all key objects were detected except for the backpack and snowboard (except for YoloV8-n). YoloV8-n was able to detect the snowboard, albeit with a low confidence score (0.26) in addition to the detection of skis in the same area.

Analysis

The three YoloV8 versions were able to detect all key objects with increased confidence as the model size increased. Notably, YoloV8-n was able to detect the snowboard with low confidence, but the larger models did not. This was anomalous as the expectation would be for the larger models to have more accurate detections. In addition, the detection speeds remained similar across the three models, with YoloV8-n offering slightly superior speeds (18ms less in one detection compared to YoloV8-m) though not significantly better. The speeds of the three versions all offered real time detection capability.

5.2.4 Detection Speeds

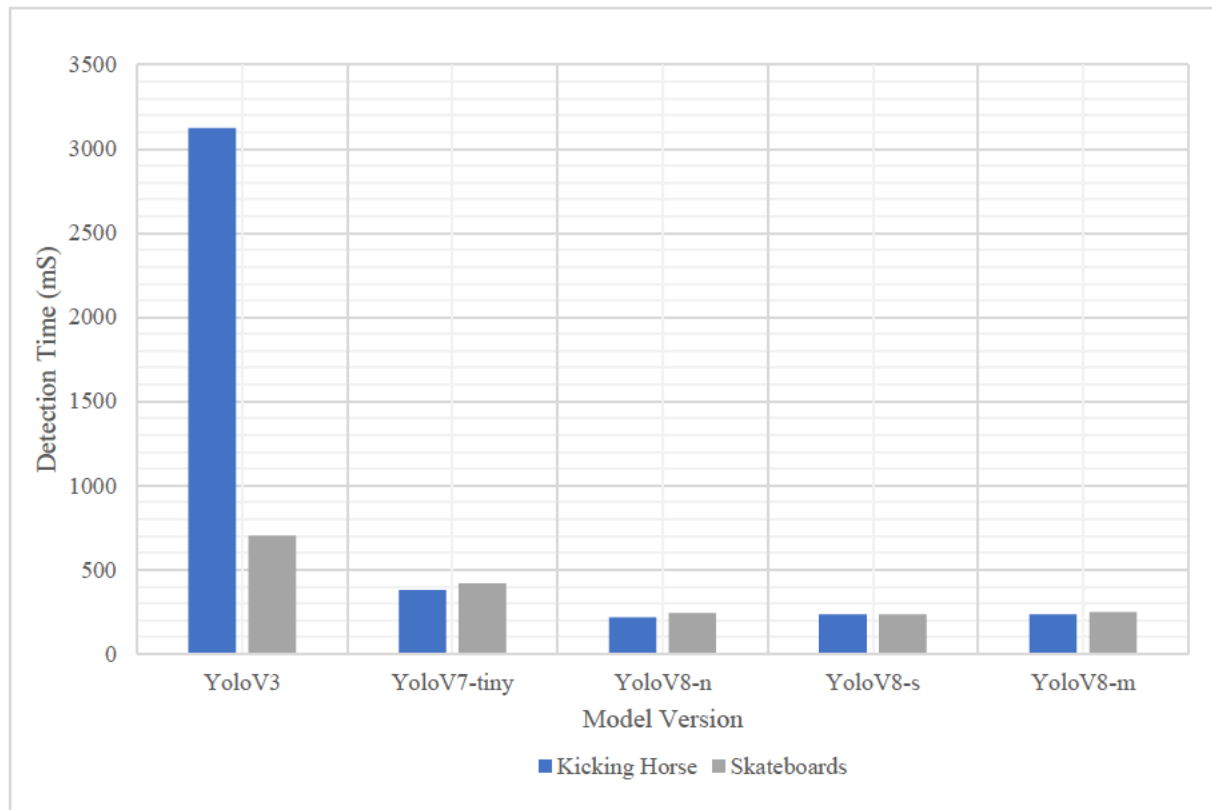


Figure 29: A comparison of detection times of the Yolo models tested.

Observation

As expected, YoloV3 had significantly longer detection times than the newer models. YoloV7-tiny also had slightly longer detection times than YoloV8 (almost double when compared to YoloV8-n). The detection times for the three YoloV8 models were all similar with YoloV8-n performing slightly better.

Analysis

YoloV3 was the largest model tested, and as such it was expected to have the longest detection times. Notably, detections in the 'Kicking Horse' image take significantly longer. There are several possible reasons, firstly, the detections made in the image were of relatively low confidence compared to the other images. It is also detecting three object classes (skis, backpack, person) compared to only two in the skateboard image (skateboard, person). These could result in more computation and processing and thus longer detection times.

Improvements in YoloV7 and V8 have resulted in comparable detection capabilities compared to YoloV3 but with significantly faster detection speeds. The detection speeds of YoloV7 and V8 models would be considered real time and are deployable on the Oak-D Lite.

Further experiments need to be conducted to determine the best model for the drowning detection system as these experiments indicate that YoloV7 and V8 perform similarly.

5.3 Experiment 2 - Quantitative Testing of Yolo Models on a Custom Dataset

This experiment focuses on training the two Yolo models on a custom dataset to determine the best model for the drowning detection system. YoloV7 and V8 were the models chosen as they can be deployed on the Oak-D Lite and demonstrated promising results in the previous experiment.

5.3.1 YoloV7-tiny

YoloV7-tiny was trained for 24 epochs (starting at epoch 0) on the train/validate image dataset. As the following figures demonstrate, during training the losses gradually decrease whilst mAP, precision and recall all increase indicating the model is successfully learning. In addition, no overfitting is observed during the training.

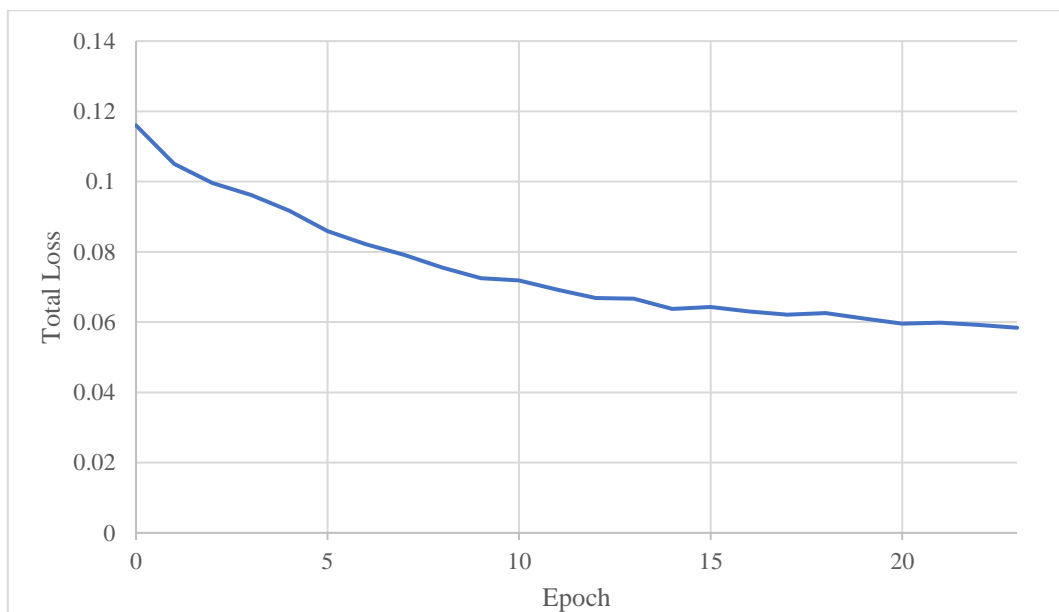


Figure 30: YoloV7-tiny losses during training over 24 epochs.

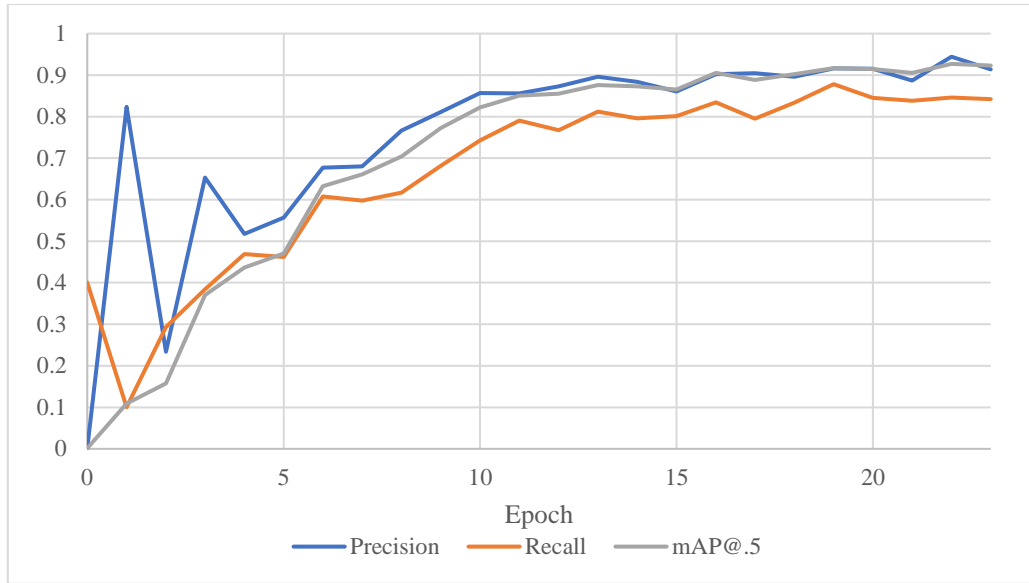


Figure 31: Precision, recall and mAP during training of YoloV7-tiny over 24 epochs.

Throughout the training, weight files were saved every epoch, YoloV7 automatically determines the best weight file based off several metrics including precision, recall and mAP. This file was then used to test the model on the test images. The results are tabulated below.

| Best Weights | | | | | |
|--------------|--------|-----------|--------|---------|-------------|
| Images | Images | Precision | Recall | mAP 0.5 | mAP 0.5:.95 |
| All | 599 | 0.63 | 0.495 | 0.549 | 0.32 |
| Drowning | 599 | 0.738 | 0.394 | 0.239 | 0.294 |
| Idle | 599 | 0.645 | 0.613 | 0.667 | 0.465 |
| Swimming | 599 | 0.505 | 0.479 | 0.443 | 0.203 |

Table 9: The test results using the custom YoloV7-tiny best weights.

In addition, detections were carried out on three test images. Each image shows a particular class of either idle, swimming or drowning.



Figure 32: Correct detection of swimming with confidence of 0.60.



Figure 33: Detection of swimming and drowning in the drowning state.



Figure 34: Incorrect detection of swimming in the idle state.

Observation

Training was conducted at a rate of approximately 2 seconds per iteration, and as such, training for 24 epochs was time consuming. The training metrics indicate that the model's learning rate progressed well and reasonable metrics for mAP, precision and recall were achieved using the best weight files.

The detections run on the test images were inaccurate, with only the correct detection of the swimming state. The model detected both swimming and drowning in the drowning state and failed to detect the idle action, instead mistaking it for swimming.

Analysis

Given the model was only trained for 24 epochs, the initial results are promising. The literature review indicated that models were trained for hundreds of epochs (generally in the range of 300), thus there is significant room to improve. The training data also suggests that the model would further benefit from more training time as there is no indication of over fitting and a loss rate of around 0.01 is desirable.

In addition, detections run on the test images are also encouraging. The model was able to detect the drowning state and despite also detecting swimming, the model did not miss a potential drowning event. The mAP scores show that the model struggles to accurately detect drowning (mAP 0.5 of 0.239) and swimming (mAP 0.5 of 0.443). Intuitively, this is logical given the classes can appear similar depending on the image. Increases in these metrics would see overall improvement in the model.

5.3.2 YoloV8-s

YoloV8-s was trained for 24 epochs (starting at epoch 0) on the train/validate image dataset. Like YoloV7, the losses gradually decrease indicating improvement in the model's learning. Of interest is the fact that precision, recall and mAP start significantly higher than observed in YoloV7-tiny. Again, these improve over the epochs indicating the model is successfully learning.

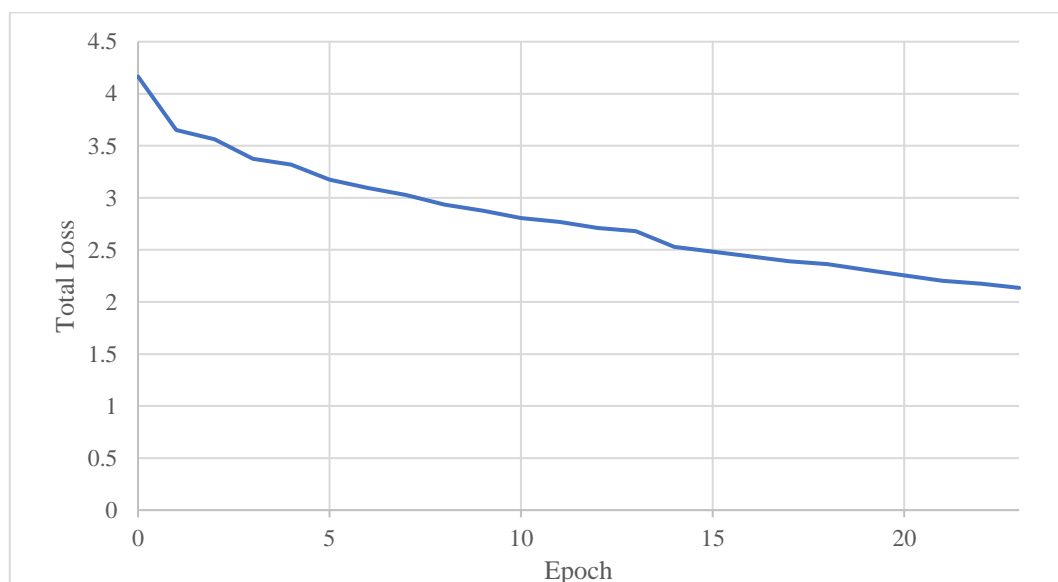


Figure 35: YoloV8-s losses during training over 24 epochs.

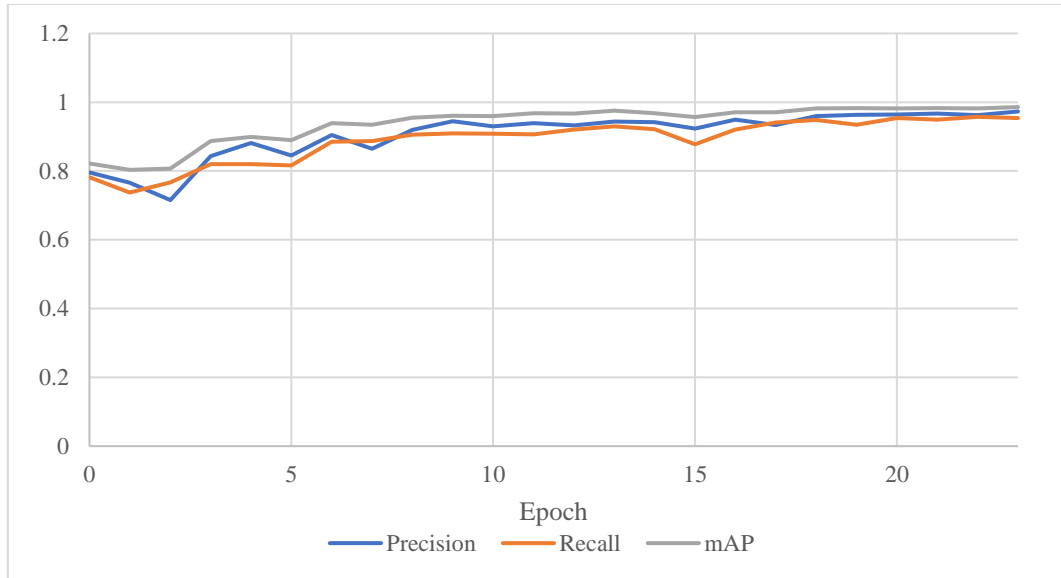


Figure 36: Precision, recall and mAP during training of YoloV8-s over 24 epochs.

Like the YoloV7-tiny training, weight files were saved every epoch and the best weight file selected (this was the final weight file). Again, this is the weight file that was used to test the model.

| Best and Last Weights | | | | | |
|-----------------------|--------|-----------|--------|-------|-------------|
| Class | Images | Precision | Recall | mAP50 | mAP .50:.95 |
| All | 599 | 0.579 | 0.595 | 0.598 | 0.425 |
| Drowning | 599 | 0.685 | 0.456 | 0.535 | 0.343 |
| Idle | 599 | 0.534 | 0.739 | 0.761 | 0.634 |
| Swimming | 599 | 0.518 | 0.589 | 0.498 | 0.299 |

Table 10: The test results using the custom YoloV8-s best weights.

Again, detections were done on the same test images used in the YoloV7-tiny testing.

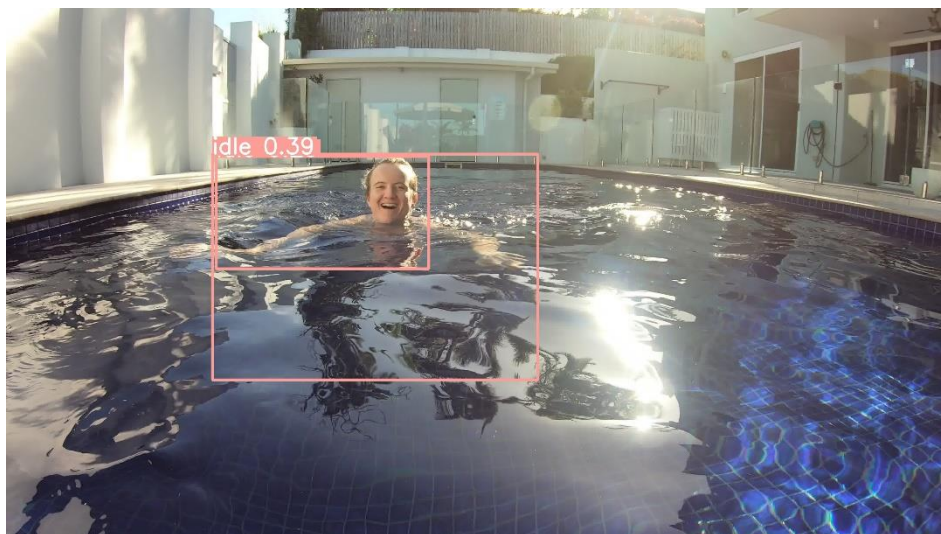


Figure 37: Incorrect detection of idle in the swimming state.



Figure 38: Incorrect Detection of swimming in the drowning state.



Figure 39: Incorrect detection of swimming in the idle state.

Observations

The first significant observation was the training time of the model. The model was able to train around 4 to 5 iterations a second, a significant improvement in training time when compared to YoloV7-tiny (2 seconds per iteration). There was also improvement in the key metrics when tested on the test images, notably the drowning and swimming classes. The training losses also start at a far higher value than observed in YoloV7-tiny as well as precision, recall and mAP.

When detections were run on the same test images, the model proved to be inaccurate and failed to make any correct detections. The confidence scores for the two swimming actions were high.

Analysis

Despite the improvements in metrics, the model still struggled to accurately make detections on the test images. However, the model's capabilities are encouraging given the losses being measured. As figure 35 shows, the losses start significantly higher than observed in YoloV7-tiny and there was a considerable difference in losses after 24 epochs. This suggests, that with more training, the losses could be lowered further which would see improvement in the model's performance. Given the speed of training, further training of this model is feasible and practical. Interestingly, in figure 37, there are two occurrences of bounding boxes in this image. Noticeably, the larger bounding box has encompassed water reflections which was previously highlighted in systems examined in the literature review.

These experiments assist in answering the second research question, 'What size dataset is needed to train the selected algorithm and gain accurate results?'. Though literature indicated a larger dataset was desirable (over 10,000 images), practicality dictated that this was not feasible. However, meaningful results have been achieved with a dataset of approximately 5000 images.

5.4 Experiment 3 – Improving the Yolo Model

YoloV8 was selected as the model to be further developed due to the speed of training and the opportunity to further reduce losses. In addition, initial testing was carried out using YoloV8-s, the Oak-D Lite can run the larger V8-m, offering an additional opportunity to increase accuracy. Several improvements were identified with the results presented in the following sections.

5.4.1 Background Images

Background images were added to the dataset to reduce the instances of false positives.

YoloV8-s was trained on this improved dataset and then compared with the original YoloV8-s model.

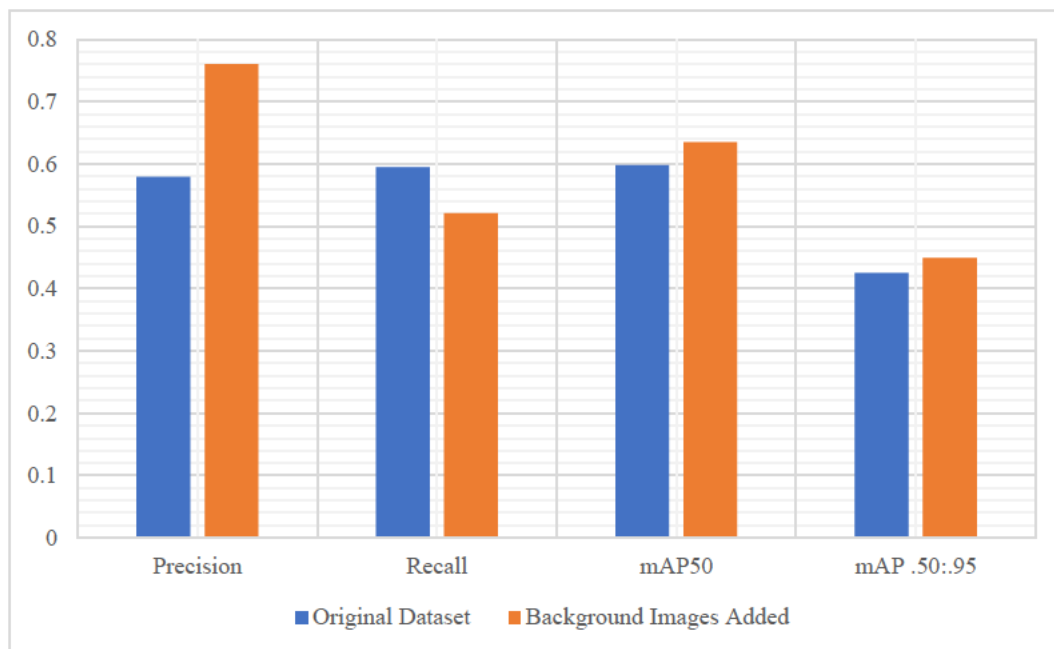
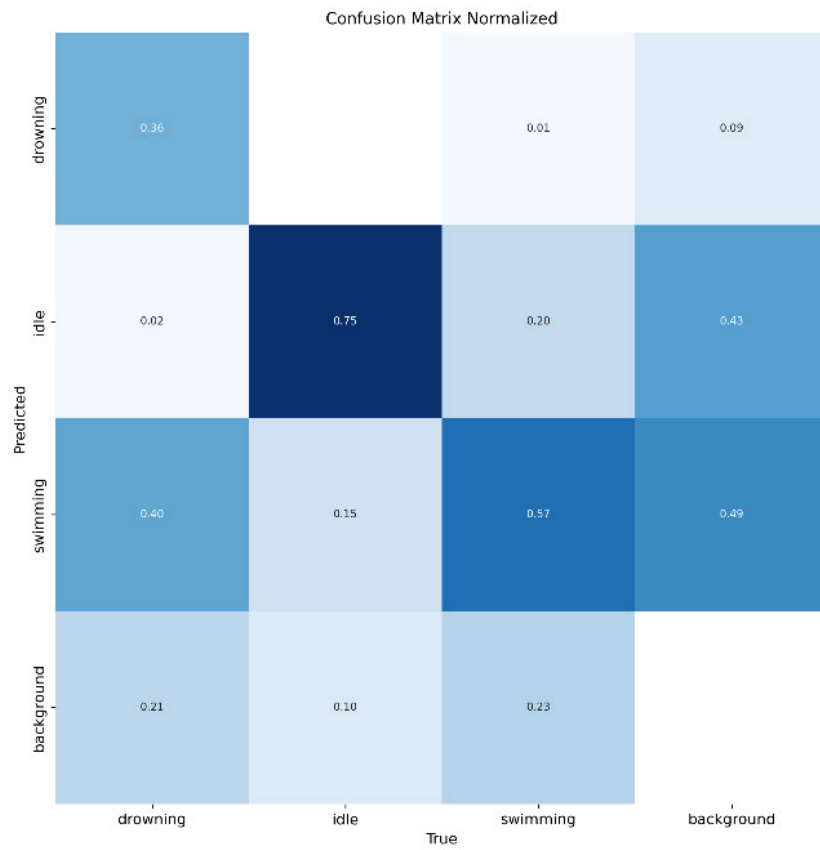
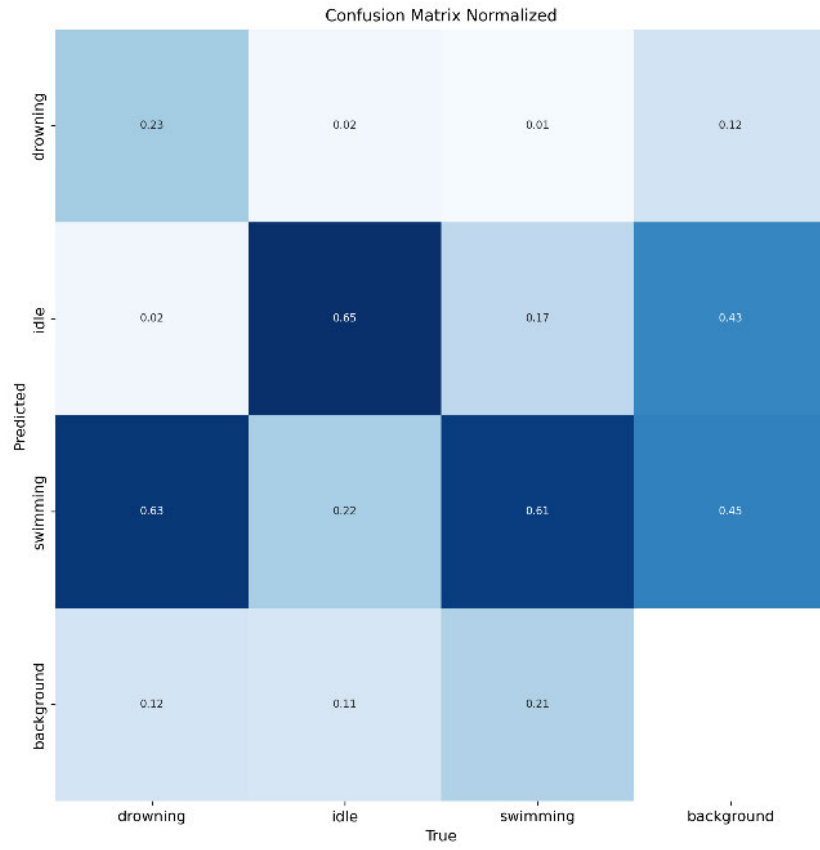


Figure 40: A comparison of key metrics between YoloV8-s trained on the original and improved dataset with background images.



Figures 41a and b: The confusion matrices for the YoloV8-s models. On the left the model trained with no background images and on the right the model trained with background images.

Observation

As figure 40 shows, there is improvement in three of the metrics apart from recall in which there is a slight reduction. There is significant improvement in precision and subtle improvement across the two mAP scores. The confusion matrices do not show significant changes in the model, though the matrix for the model trained with background images does show a general reduction in false positives, with a slight reduction in true positives for swimming.

Analysis

Precision is the number of true positives divided the sum of true and false positives which describes the number of correct positive identifications. Recall is the number of true positives divided by the sum of true positives and false negatives describing the model's ability to correctly identify a class. Given that the model trained on no background images has slightly higher recall, this is perhaps the reason for the confusion matrices not demonstrating a significant change in the model's detection. However, the matrices do demonstrate a subtle reduction in false positives with the inclusion of background images which was to be expected. Overall, the model's performance has improved with the inclusion of even a small number of background images.

5.4.2 Increasing the Model Size

The next experiment was to increase the size of the model from YoloV8-s to YoloV8-m. Again, the model was trained for 24 epochs and key metrics compared. This model was also trained on the dataset which included background images. The results are shown below.

| Yolo V8-m | | | | | |
|-----------|--------|-----------|--------|-------|-------------|
| Class | Images | Precision | Recall | mAP50 | mAP .50:.95 |
| All | 622 | 0.653 | 0.573 | 0.625 | 0.44 |
| Drowning | 622 | 0.734 | 0.434 | 0.6 | 0.368 |
| Idle | 622 | 0.657 | 0.765 | 0.794 | 0.662 |
| Swimming | 622 | 0.569 | 0.521 | 0.482 | 0.29 |

Table 11: Key metrics for YoloV8-m trained on the improved dataset over 24 epochs.

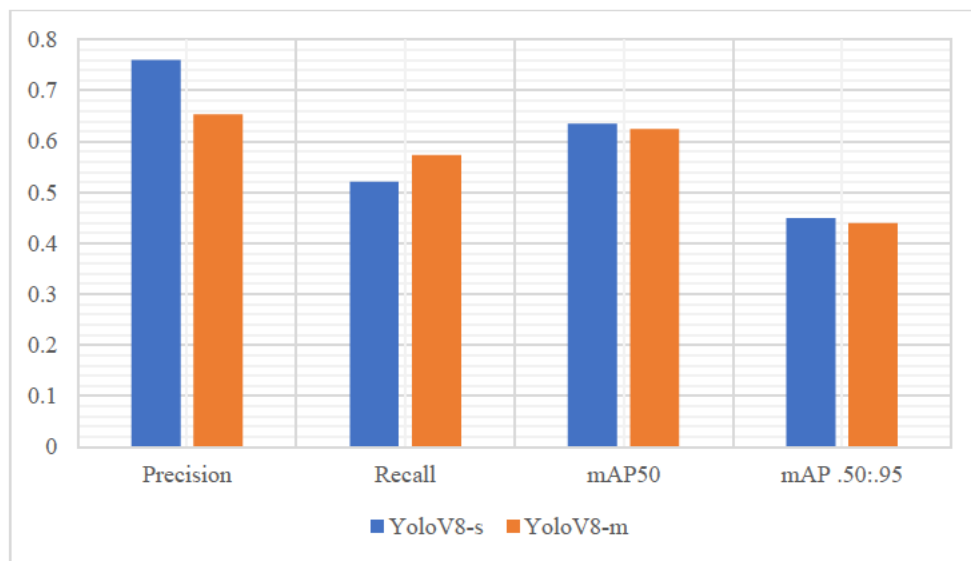


Figure 42: A comparison of YoloV8-s and YoloV8-m key metrics.

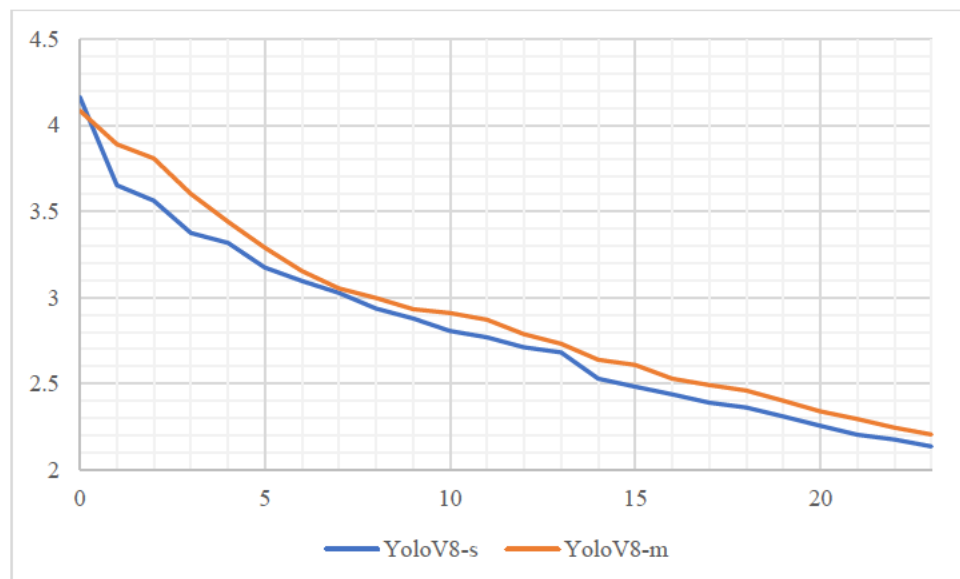


Figure 43: A comparison of total losses between YoloV8-s and YoloV8-m.

Observation

The key metrics for both models are similar by comparison when trained over 24 epochs. There are slight differences in precision and recall between the two models, with YoloV8-s having higher precision and lower recall and vice versa for YoloV8-m. YoloV8-m does trend slightly higher with training losses in comparison to YoloV8-s.

Analysis

There appears to be negligible difference between the two models when comparing key metrics, however, the losses for YoloV8-m trend consistently higher than YoloV8-s. This indicates that the model may benefit from being trained for more epochs. As such, it seems advantageous to continue improvements based on YoloV8-m.

5.4.3 Image Augmentation and Parameter Tuning

Batch Size

Several batch sizes were tested using the YoloV8-m model, with the three sizes being 16, 32 and 64. The results are shown below.

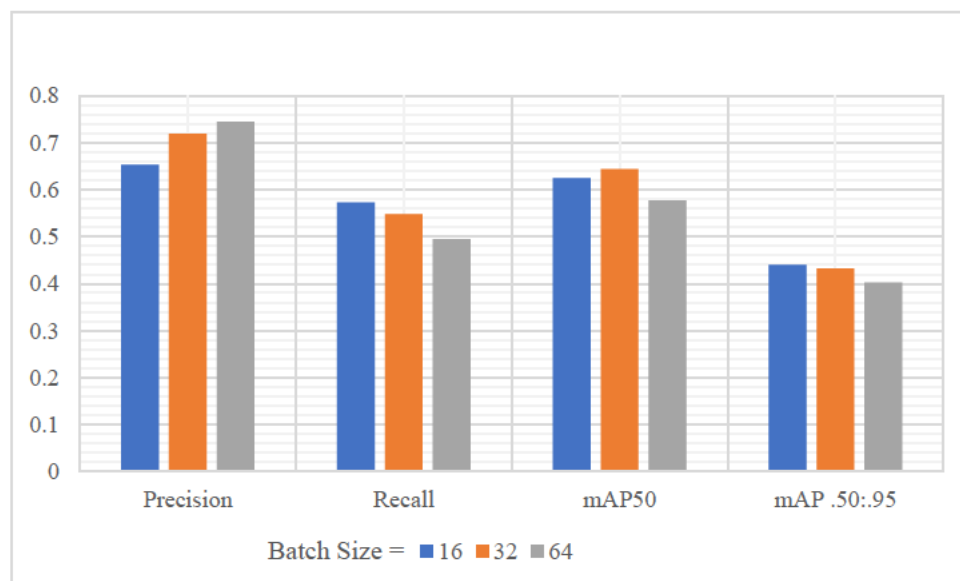


Figure 44: A comparison of batch sizes during training of YoloV8-m.

Observation

As the batch size increases, so does the precision whilst recall decreases. There is a slight increase in mAP50 with a batch size of 32, and a very slight reduction in mAP.50:95 as the batch size increases.

Analysis

Increasing the batch size can potentially lead to faster convergence should processor memory allow. In this instance, there is improvement with a batch size of 32 compared to 16 but little gain when increasing to 64. As a result, the optimal batch size for this model appears to be 32 and is the value that shall be used for the proceeding experiments.

Learning Rate

Three learning rates were tested and compared: 0.01, 0.005 and 0.001 and results collated. The batch size was 32 and YoloV8-m was the model trained.

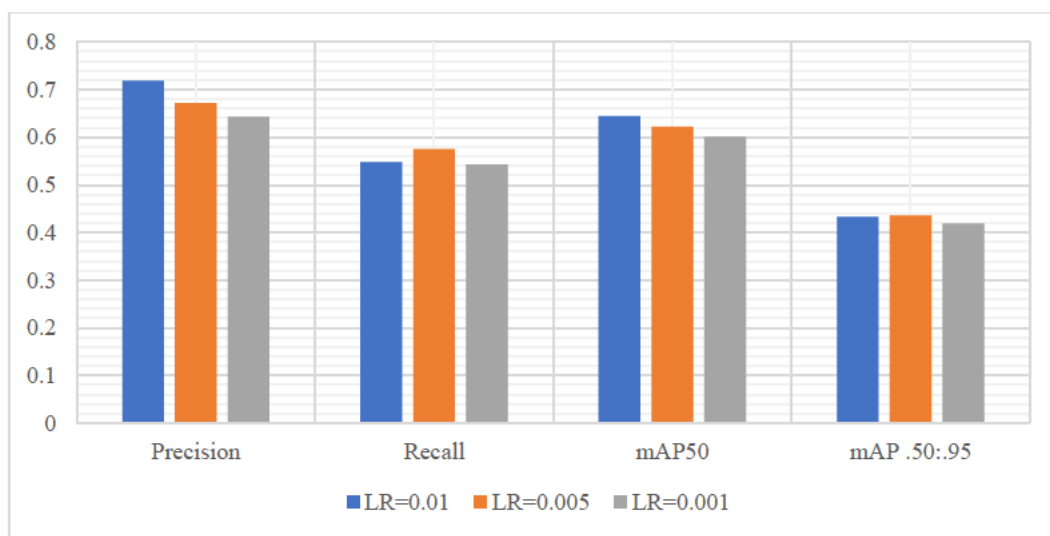


Figure 45: Key metrics for the three learning rates tested.

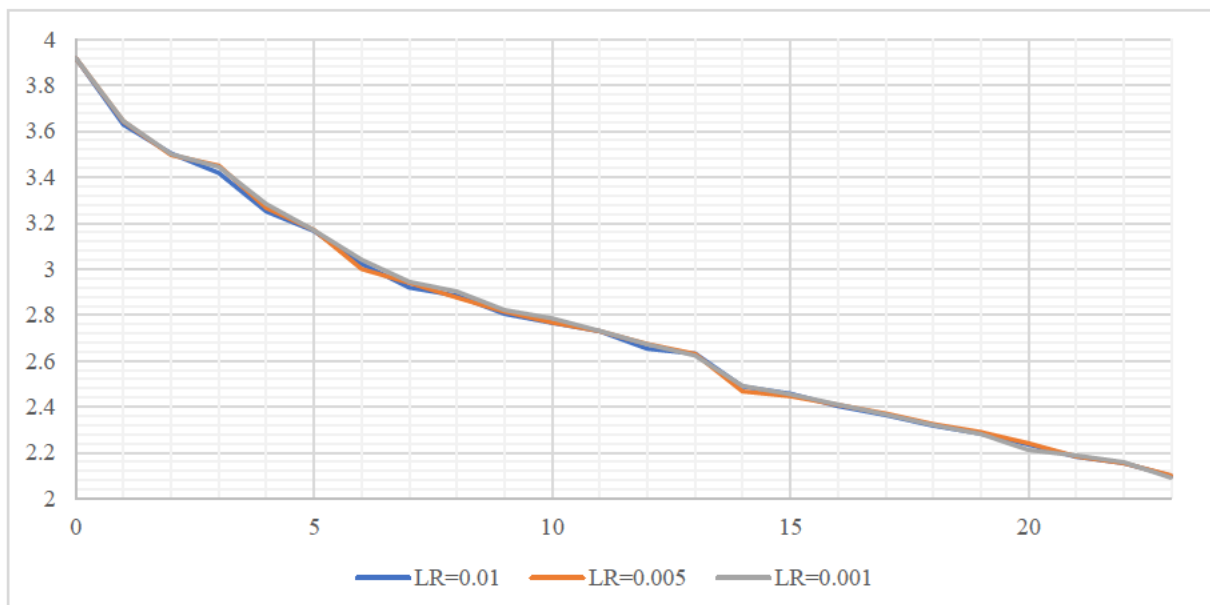


Figure 46: Total losses for the three learning rates tested.

Observation

In general, the key metrics for the three learning rates tested decreased as the learning rate decreased except for recall which increased with a learning rate of 0.05. The total losses for the learning rates, remained consistent throughout the epochs. In addition, the training time for each model was similar.

Analysis

Given that as the learning rate decreases, model convergence time increases, seeing the key metrics decrease is not surprising. I would expect to see these values increase as the number of training epochs increases. In addition, losses trend consistently between the different values tested. Due to there being no noticeable increase in training time with a lower learning rate, a learning rate of 0.001 shall be used through the proceeding experiments.

Image Augmentation

To reiterate, the values changed in this experiment are as follows:

| Parameter | Description | Default | Experiment 1 | Experiment 2 |
|-----------|-------------------------------------|---------|--------------|--------------|
| degrees | image rotation (+/- deg) | 0 | 180 | 90 |
| translate | image translation (+/- fraction) | 0.1 | 0.8 | 0.4 |
| scale | image scale (+/- gain) | 0.5 | 0.8 | 0.4 |
| shear | image shear (+/- deg) | 0 | 180 | 90 |
| flipud | image flip up-down (probability) | 0 | 0.8 | 0.4 |
| fliplr | image flip left-right (probability) | 0.5 | 0.8 | 0.4 |

Table 12: The image augmentation parameters used in the experiments.

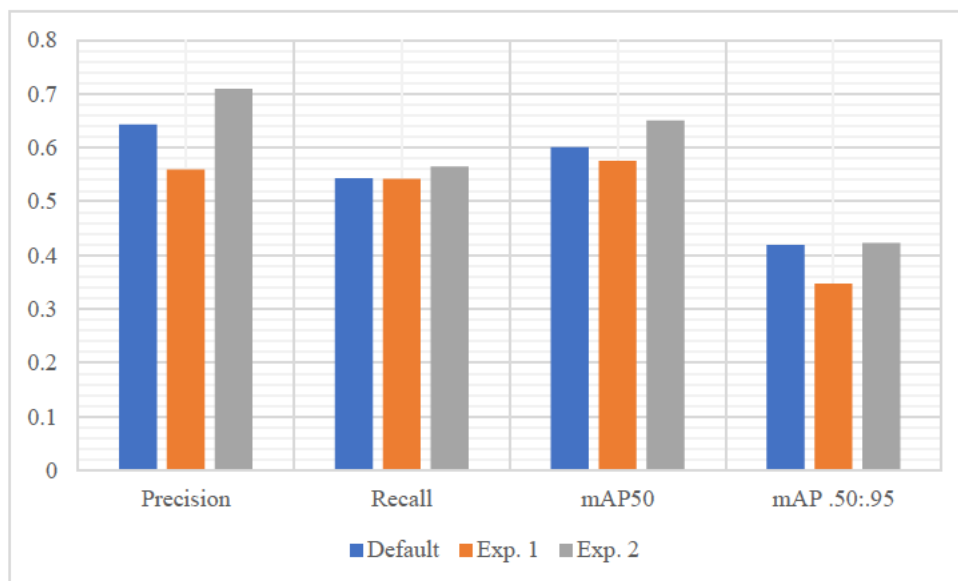


Figure 47: Key metrics with various image augmentation methods implemented.

Observation

Default augmentation values and the values used in experiment 1 offer little in the way of conclusive differences, across three of the metrics the values used in experiment 1 result in poorer results than the default. Of interest is the significant improvement seen with the values in experiment 2, which performs best across all metrics.

Analysis

The values chosen in experiment 2 were lower than those in experiment 1. This suggests that ‘less is more’ and perhaps too much augmentation adds little benefit beyond a certain threshold. There are notable improvements from the default values to those used in experiment 2 and as such, augmentation offers an opportunity to further improve model accuracy. Evidentially, further experimentation with augmentation parameters would be desirable though this falls outside the scope of this thesis.

5.4.4 Training Epochs

The final improvement to be implemented was to train the model for more epochs. The model was trained for 300 epochs as guided by the literature review. Metrics were compared to those previously captured in the final image augmentation experiment (experiment 2) as well as the original, unimproved YoloV8-m model trained for 24 epochs as these produced the best results for comparison. In addition, the confusion matrix, precision-recall curve and F1 curve generated when training the model have also been included for analysis.

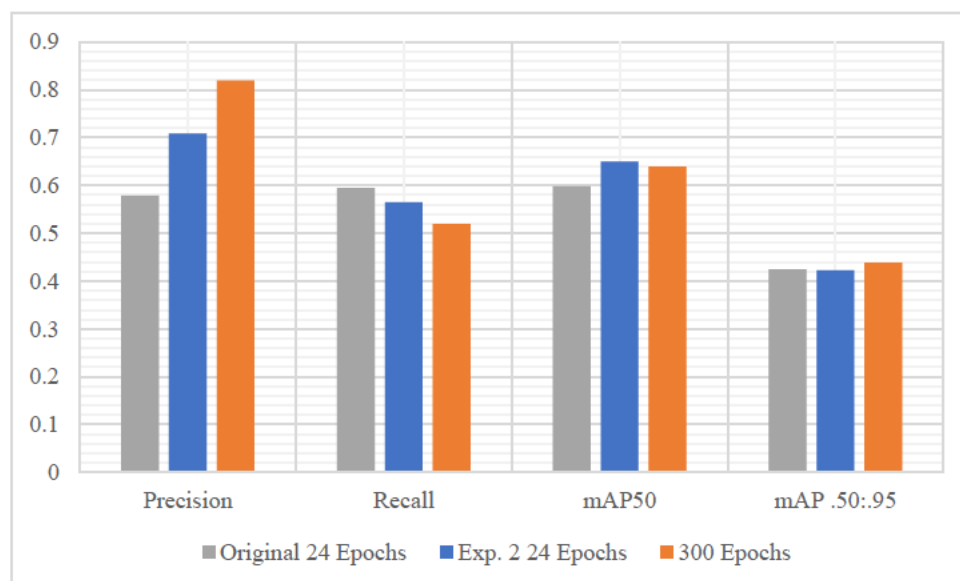


Figure 48: A comparison of key metrics across three models, the original YoloV8-m trained for 24 epochs, the improved 24 epoch version and the final version trained for 300 epochs.

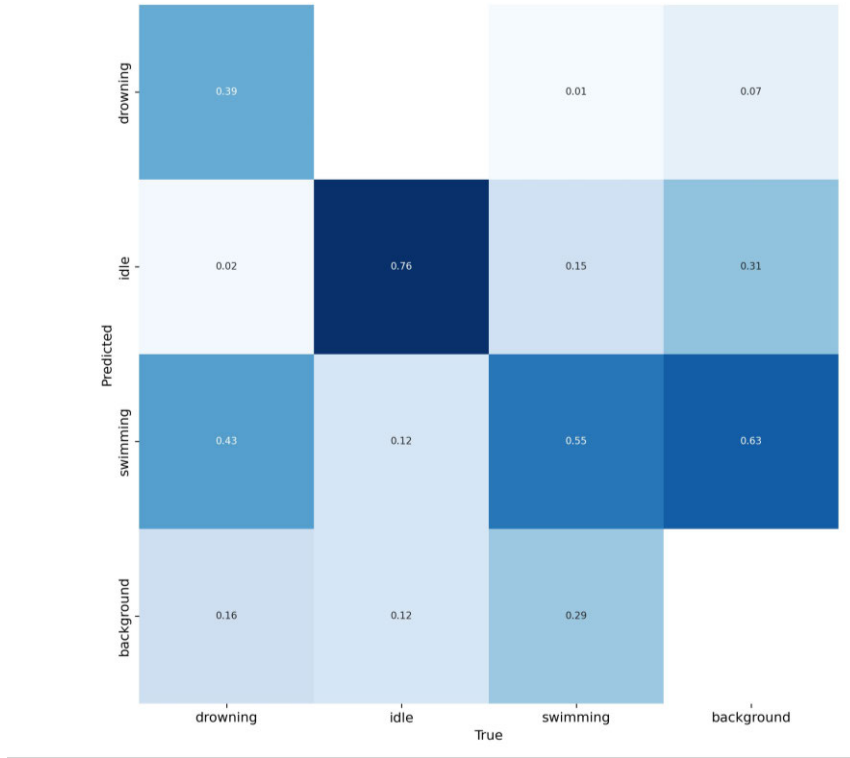
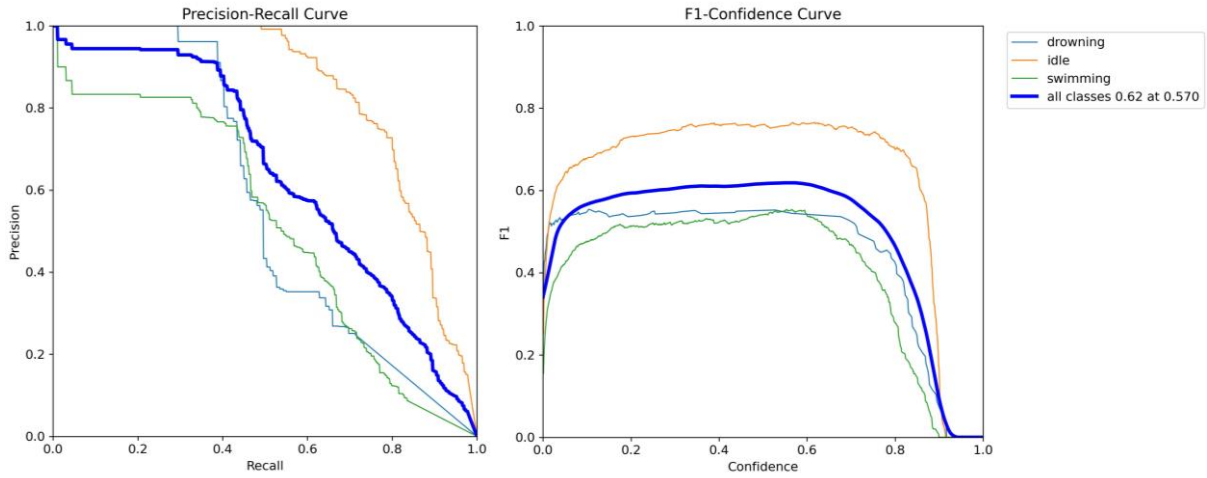


Figure 49: The confusion matrix for the final model.



Figures 50a and b: The precision-recall curve and F1 confidence curve for the final model.

Observation

With reference to figure 48, in general there has been improvement across most metrics with the increase in training epochs except for recall, which decreases with the final model. The mAP scores are slightly lower for the 300 epoch model when compared to the previous, improved version which was trained for 24 epochs.

The confusion matrix also shows promising results, though there is a high occurrence of the model detecting swimming in a drowning state (0.43) in addition to detecting swimming in

background images (0.63). The matrix shows that the model is good at predicting the idle state (0.76).

Finally, the precision-recall curve shows excellent results in the idle state, with lower values for precision and recall for drowning and swimming. This is also reflected in the F1 curve, where the model again excels with detecting the idle class but struggles more with swimming and drowning.

Analysis

The most significant improvement after training the model for 300 epochs can be seen in the precision score, though recall gradually decreases. This means that the model is making fewer predictions (due to low recall) but more predictions are correct (high precision). One potential solution would be to lower the required confidence score for a detection. Alternatively, the final interface could compensate for this with a timer; if a potential drowning is detected, it shall be assumed to be a drowning unless not detected for a certain period. Examination of the confusion matrix, precision-recall curve and F1 confidence curve shows that the model can accurately detect idle though struggles with drowning and swimming. Intuitively, this makes sense, the images of drowning and swimming can appear similar when not in context, as the following images show. In addition, the analysis of the dataset showed that the idle state was overrepresented when compared to drowning and swimming thus giving the model some bias.

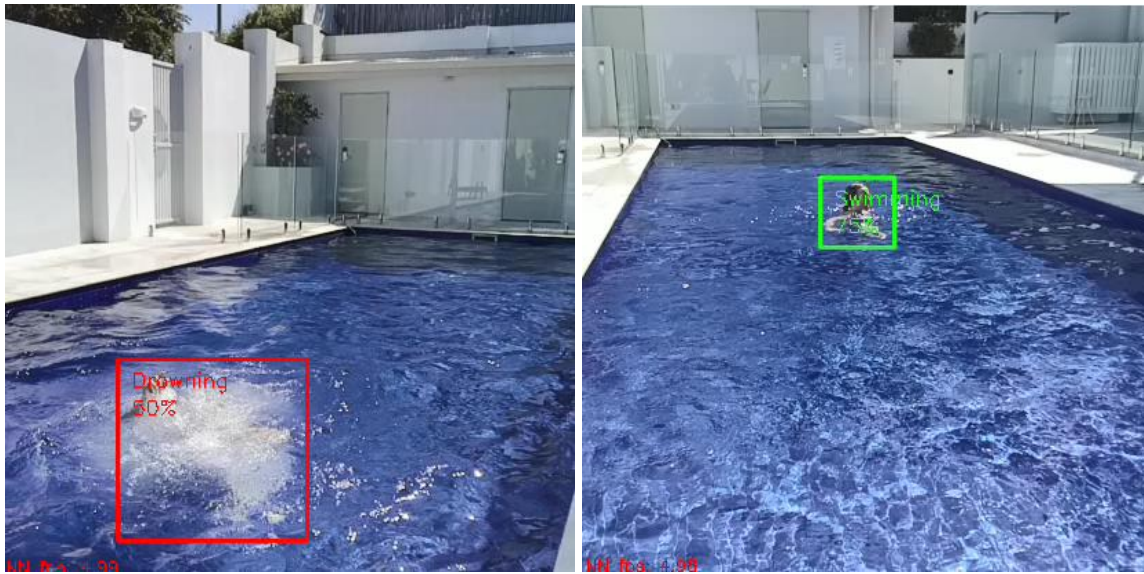


Figures 51a and b: Images from the test dataset, on the left the drowning class and on the right swimming, which when taken out of context could prove difficult for even a human to classify.

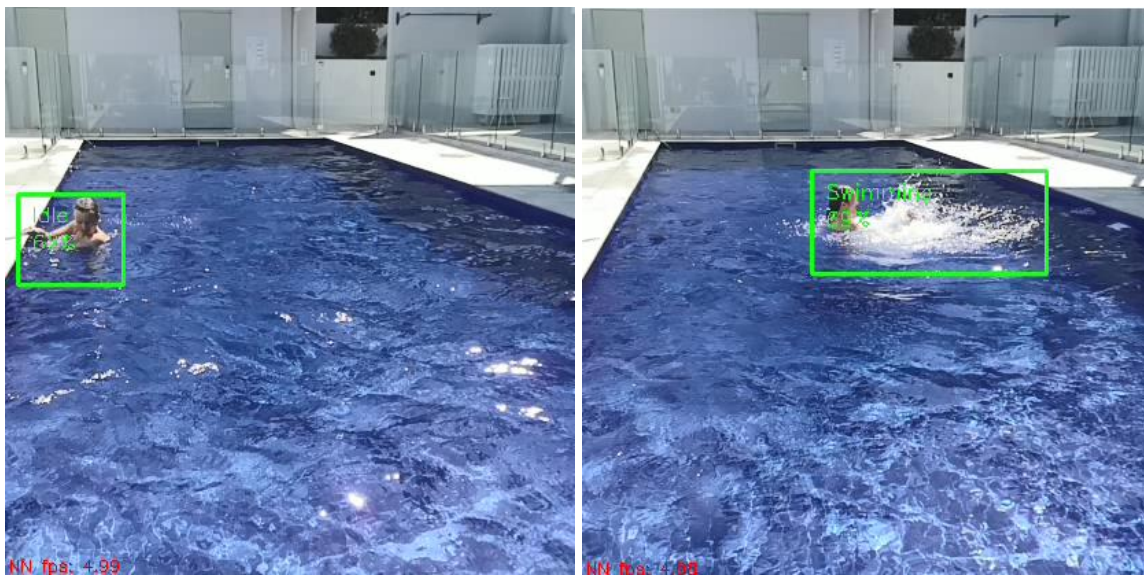
Though additional improvements to the model shall not be made beyond this experiment, the results indicate that increasing the size and diversity of the dataset could improve the model's performance. Increasing the instances of drowning and swimming in the training data could see significant improvement in these metrics.

5.5 Experiment 4 – Deployment on the Oak-D Lite

The final model was deployed on the Oak-D Lite and Raspberry Pi and tested in field. This experiment set out to confirm that successful deployment was possible and further direct the changes needed for the final deployment in experiment 5.



Figures 52a and b: Correct detections in the drowning and swimming states.



Figures 53a and b: Correct detection of the idle state and an incorrect detection of swimming in the drowning state.

Observation

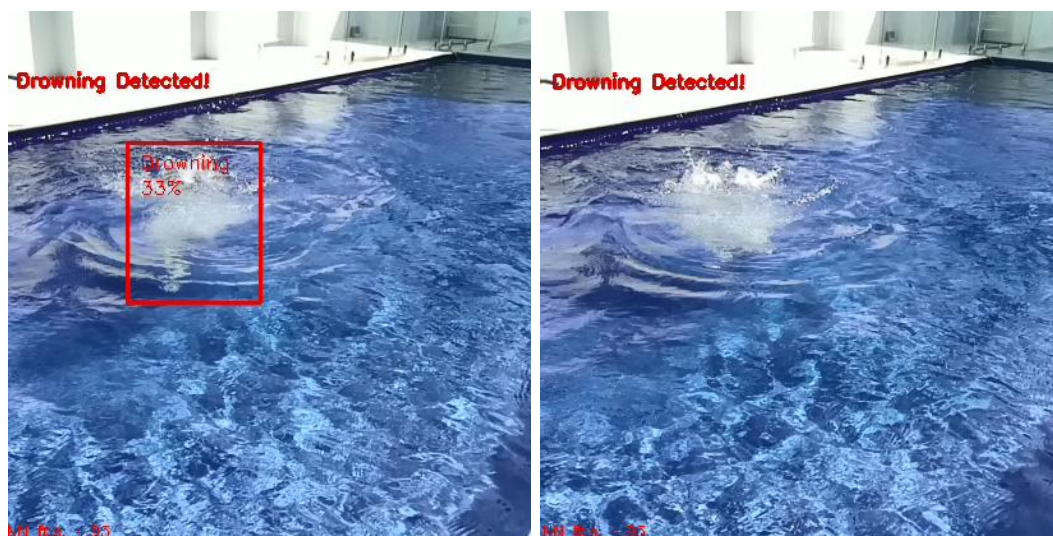
The frame rate for the video feed was between 4-5 fps. As the images above show, the resolution was quite low. In general, the model was able to detect swimming, idle and drowning however was prone to confusing swimming and drowning. It also generally detected the idle state as swimming. It was also able to make detections from various areas of the pool under varying lighting conditions. The confidence interval was set to 0.5.

Analysis

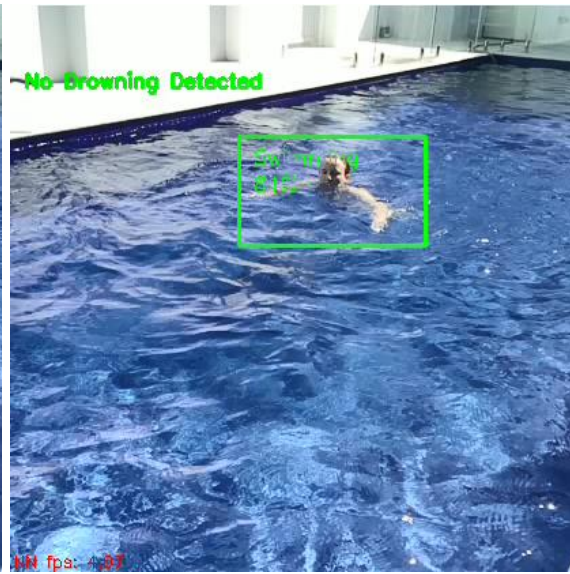
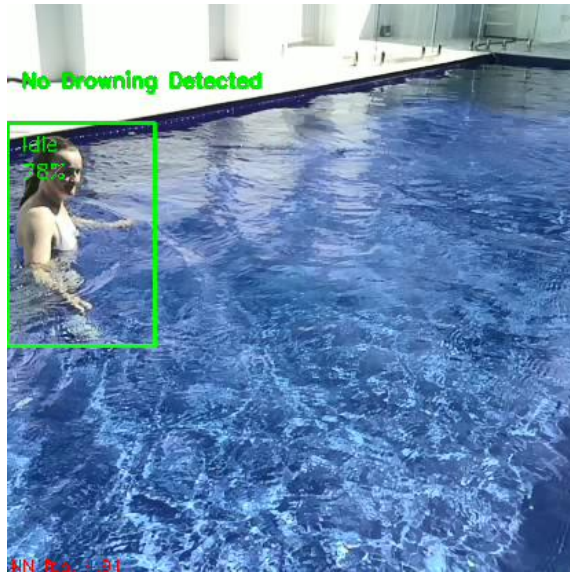
The previous experiments demonstrated that the model had problems differentiating between drowning and swimming, and as such it is not surprising this is also the case in real time. There are several adjustments which can be made to the final version of the detection system such as better indication of the drowning state, and this indication holding for several seconds before it is cleared of being in the drowning state. In addition, it may be worthwhile lowering the confidence interval of the detection system and increasing the resolution to see if this assists with detections.

5.6 Experiment 5 – Real Time Detection and Alert

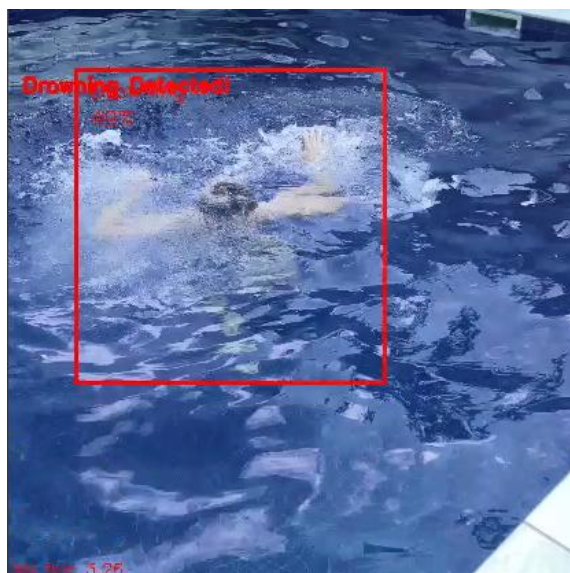
This experiment set out to demonstrate a real time detection system alerting of a drowning event. In addition, proof of concept was to be demonstrated for a remote interface which could also be used for monitoring drowning events. Testing was initially done at a lower resolution (1080p as per experiment 4) then increased to 13MP which is the Oak-D Lite's maximum resolution.



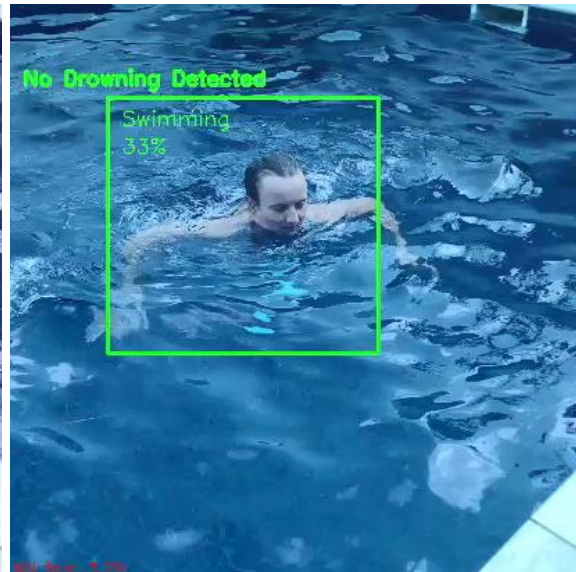
Figures 54a and b: Detection of the drowning state and the alert holding despite the loss of the drowning bounding box at 1080p resolution.



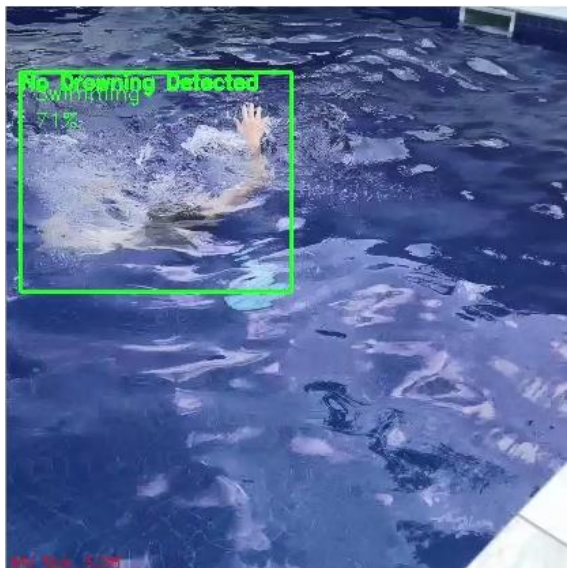
Figures 55a and b: Correct detection of idle and swimming at 1080p resolution.



Figures 56a and b: Correct detection of drowning at a higher resolution and the alarm test holding despite the loss of the bounding box.



Figures 57a and b: Correct detections of the swimming state at higher resolution.



Figures 58a and b: An incorrect detection of swimming in the drowning state and a missed detection of drowning at higher resolution.

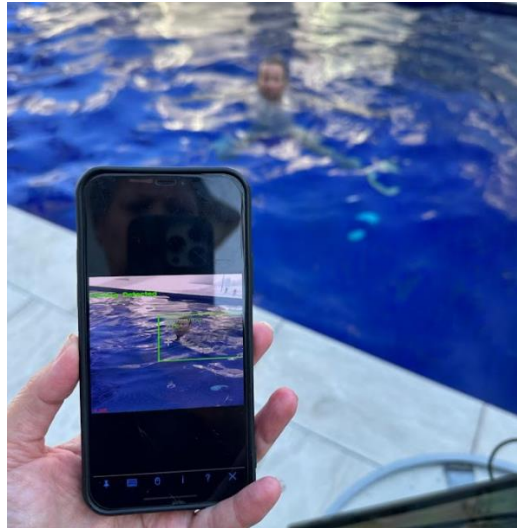


Figure 59: Successful detection and remote monitoring using the VNC from the Raspberry Pi to iPhone.

Observation

In a similar fashion to experiment 4, the system was still prone to confusing the drowning and swimming states and occasionally missed drowning events. However, it still made correct detection as figures 54 to 57 demonstrate. It was tested under varying lighting conditions and from differing perspectives around the pool and was still able to make good detections. As the resolution increased, the frame rate noticeably dropped. At 1080p, the frame rate was consistently between 4-5 fps. When increased to the maximum resolution of 13MP, the frame rate remained around 3 fps. Detections did not seem to improve noticeably as the resolution was increased.

The drowning detection alert banner functioned as intended, alarming if drowning was detected, and holding for the fixed amount of time (2 seconds but easily adjusted in the python script). The VNC method of remote monitoring via the VNC server functioned well so that the interface could be viewed from a variety of devices including an iPhone and windows laptop.

Analysis

Given the same weight files were used in experiment 5 from experiment 4, detection performance was similar. The system often struggled with drowning and swimming. Increasing the resolution didn't seem to impact the quality of detections, though the frames started to lag noticeably as the frame rate decreased to 3 fps. Referring to the Luxonis documentation on the Oak-D Lite, the maximum fps available for the Yolov8-m is 6 fps, thus a frame rate of 5 fps is at the higher end of what the Oak-D Lite is capable of. In future, it

would be worthwhile experimenting with smaller versions of the Yolo algorithm which can be operated at a higher frame rate to see if the quality of detections improves. For example, YoloV6n R2 at 416x416 is capable of a frame rate of 65.5fps (Luxonis, 2023), in addition YoloV7-t at 416x416 is capable of 46.7 fps (Luxonis, 2023) which is significantly higher than the 6 fps of YoloV8-m.

The alert banner worked well and assisted in compensating for detections. It was designed to prioritise any drowning detection and alert the user of a potential drowning event, which is one of the ultimate aims of this system. In addition, dropping the confidence interval from 0.5 to 0.3 helped in increasing drowning detections. It is also encouraging that the positioning of the camera didn't significantly impact the consistency of detections offering flexibility in where such a system could be installed.

6. Discussion, Conclusion and Further Work

This section shall summarise the results and extract a final analysis from all the pooled results through all experiments. Opinions shall be drawn on how much of the knowledge gap and research aims were addressed and solved.

6.1 Discussion

Experiments 1 and 2 demonstrated the capabilities of several Yolo models including YoloV3, V7 and V8. The results achieved in the first experiment were impressive with the detections in the initial test images being very promising. This formed a benchmark for what the Yolo algorithm is capable of. However, it should be remembered that the Yolo models used in experiment 1 have been trained on the COCO dataset which consists of 118,000 train images, 5000 validation and 20,000 test. The custom dataset used was far more modest in size consisting of 4003 train, 521 validate and 599 test and as such, the results in experiments 3 through to 5 had to be evaluated in this context. The final mAP50 of the model deployed was 64% with a mAP50-95 of 43.9%, in context, the Ultralytics YoloV8-m release trained on the benchmark COCO val2017 dataset achieved a mAP50-95 of 50.2% (Ultralytics, 2023). Thus, achieving such metrics on a small dataset is very encouraging.

The changes made to the final Yolo version including adding background images to the dataset, increasing the model size, image augmentation, parameter tuning and increasing the training epochs improved the model's performance across all metrics. Evidentially there is room for further improvement, notably increasing the dataset size, additional experimentation with augmentation and parameter tuning and further extending the training time, however this falls outside the scope of this project due to time restraints. In addition, there will continue to be difficulties in sourcing a dataset for this system due to the sporadic nature of drowning events. However, further diversifying the dataset through various people, locations and conditions would add to its quality and ultimately the capabilities of the trained model. Focusing on increasing the number of drowning and swimming training images would also see improvements in the model, the idle state was over-represented in the dataset and was consistently detected at a more accurate rate as a result.

An additional method for dataset improvement may be by acquiring training images using the Oak-D Lite as well as devices such as the GoPro. This would ensure consistency between image framing, resolution, and quality. It was notable that the HSV and contrast qualities observed in the images acquired from the Oak-D Lite were dissimilar to that of the GoPro

and the Dubai dataset. Eng et al (2008) noted that their system missed some detections due to low contrast between swimmers and backgrounds. As such, this system could be experiencing similar issues.

Real time deployment and testing saw the model continue to occasionally struggle with differentiating between drowning and swimming though lowering the confidence interval from 0.5 to 0.3 helped to improve drowning detections. Both Eng et al (2008) and Lei et al (2021) experienced similar issues with their systems. Eng et al (2008) found water disturbances lead to greater false positives and Lei et al (2021) found their Yolo algorithm mistook part of the drowning behaviour for swimming as it detected water surface reflection as swimming resulting in higher false positives. The random nature of water movement and reflection is a significant challenge in drowning detection systems. A method of compensating may be by the use of pose-estimation algorithms in conjunction with Yolo. Hasan et al (2022) found that when comparing scene classification and pose estimation algorithms, pose estimation were impacted less by scene factors such as lighting and water movement. Interestingly, YoloV8 does have pose estimation capabilities, thus there would be an opportunity to leverage this for improved model performance in future work.

It was also notable that there was intermittent changing of states from drowning to swimming, and vice versa during drowning actions. The interface aimed to compensate for this by triggering a timer so that a non-drowning event had to be detected for a minimum of two seconds before it would change state. However, the use of temporal and recurrent algorithms may assist in adding context to the system. For example, if the previous frames were a drowning event and the algorithm determines that the current frame is a swimming class with low confidence, it is therefore likely to be drowning. Recurrent Neural Networks (RNN) are known for their ‘memory’ where the current output considers previous inputs and outputs, thus putting the current frame into context. A Recurrent Yolo model was proposed in 2019, where the model’s capabilities extended to Long Short-Term Memory (LSTM). This resulted in the model having excellent object tracking ability, even when the target (a pedestrian) was obscured from view (Yun and Kim, 2019). The ability for the drowning detection model to track swimmers with obstructions such as water movement and splashing could result in improved performance as it was notable that there were occasional missed detections. YoloV8 also includes object tracking abilities, in addition to pose estimation. Utilising these additional features in conjunction with the standard Yolo object detection model could see significant improvement in model performance.

The frame rate was problematic and there was some lag in the system. This was due to the size of the model used, YoloV8-m has a maximum frame rate of 6 fps and as such, there would be value in experimenting with smaller models of Yolo which are able to have higher frame rates at increased resolution. In addition, increasing the frame rate may assist in detection accuracy, more frames result in more images to run detections on and so perhaps a smaller model would prove to be more accurate in real time as a result. Increasing the resolution of images would also be beneficial so the model is better able to identify features.

The alert interface though modest, was effective and demonstrated remote monitoring of the drowning detection system. There is significant room for such an interface to improve, with simple modifications compensating for drowning detection adding value to the system.

Developing the system to include notifications to smart devices such as watches and phones would be beneficial. The ultimate goal of this system was to improve supervision around swimming pools and to act as an additional aid, and as such even this prototype tested was capable of alerting of potential drowning events.

6.2 Conclusion

The aim of this project was to develop a real time drowning detection system using a surface mounted camera in a residential pool setting, which has been achieved and demonstrated. In addition, the project aims outlined in section 1.5 were also addressed. Firstly, several algorithms used in real time drowning detection systems were researched, with Yolo algorithms consistently appearing as the most appropriate. Secondly, an appropriate dataset was acquired and developed, with experiments conducted to assist in the selection and development of an appropriate algorithm, in this case YoloV8m. Finally, the algorithm was deployed on appropriate hardware and tested in a residential setting, as demonstrated in experiments 4 and 5. Additionally, a simple interface was developed to inform users of a potential drowning event.

Four knowledge gaps relating to the research aims were also identified. Firstly, there were no systems available in a non-commercial setting utilising only surface mounted cameras. This research has proven that such a system is feasible with an Oak-D Lite, Raspberry Pi and Yolo algorithm. Secondly, algorithms beyond YoloV5 had little peer-reviewed research conducted on them, particularly in instances of drowning detection systems. This project has again demonstrated that YoloV7 and V8 are appropriate for drowning detection. Furthermore, there is scope for further experimentation to determine their suitability for deployment. For

example, assessing the performance of YoloV8-n or YoloV7-t in real time when deployed on the Oak-D Lite. Leveraging the pose estimation and object tracking abilities of YoloV8 offer an exciting opportunity to further improve detections.

A significant challenge of this research was acquiring an appropriate dataset upon which to train the algorithms. The third knowledge gap acknowledged this, and this was addressed two-fold. Firstly, the initial dataset acquired from the Rochester Institute of Technology provided a sizable and quality dataset upon which the model could be trained. However, the homogenous nature of the subjects in the dataset dictated that further diversification was needed, hence the additional ‘Mackay’ dataset. As mentioned, further expansion and variation of the dataset would be advantageous to algorithm training, and given the limitations of the dataset used, the final results achieved exceeded expectations.

The final knowledge gap addressed the use of the Oak-D Lite in a drowning detection system. Again, experiments 4 and 5 demonstrated that the hardware is well suited to such a task, with a significant limitation being the size of the algorithm deployed. As mentioned, experimenting with smaller versions of Yolo with higher frame rate capabilities would be valuable for further assessing the Oak-D Lite’s performance. In addition, given the natural development of technology with time, it is expected that further iterations of the Oak-D Lite shall only be more capable and thus more suited to a real time drowning detection system.

6.3 Further Work

As mentioned, there are several areas where further work could be conducted. These are as follows:

1) *Dataset Improvement* – Increasing the size of the dataset used to train the algorithm would only improve its performance. Particular focus on the acquisition of drowning and swimming imagery would be advantageous. In addition, varying the people, location and conditions would also improve the model’s performance. Inclusion of more background images would also be beneficial, 4% of the dataset was background imagery though literature indicated that this could be as high as 10% if needed. Acquiring imagery using the Oak-D Lite could also be beneficial to ensure training is completed using imagery with similar qualities.

2) *Parameter tuning and Image Augmentation* – Further experimentation with training parameter tuning and image augmentation offers near limitless opportunity. The research conducted demonstrated that experimentation in this area added value, though the scope of this project limited the ability of finding the optimal values.

3) *Real Time Deployment of Different Models* – Driven by improvements in metrics saw the deployment of YoloV8-m, though the low frame rate was a limiting factor for resolution and detections. Experimenting with smaller versions of Yolo, and even older versions, would offer valuable insight into the optimal model for real time deployment on the Oak-D Lite.

4) *Algorithm Development* – Two key features have been identified for areas of development with the YoloV8 algorithm; pose estimation and object tracking. Previous literature indicates that pose estimation provided good results in drowning detection systems. Built in object tracking algorithms in YoloV8 may assist in providing the model with temporal information and assist in reducing confusion between swimming and drowning behaviours. It is clear that there is significant opportunity to further develop and experiment with the full abilities of YoloV8 in a drowning detection system.

5) *Developing the Detection Interface* – The final interface demonstrating the drowning detection system was simple and as such, there is the opportunity to further develop this. A simple alert banner helped to compensate for intermittent loss of detections. There is scope to develop this into a more user friendly interface, perhaps including a mobile application with a notification system.

References

- Alpaydin, E., 2021, *Machine Learning*, The MIT Press, Cambridge, Massachusetts
- Alqahtani, A., Alsubai, S., Sha, M., Peter, V., Almadhor, A. S., & Abbas, S., 2022, Falling and drowning detection framework using smartphone sensors. *Computational Intelligence and Neuroscience*
- Alshbatat, A.I.N., Alhameli, S., Almazrouei, S., Alhameli, S. and Almarar, W., 2020, Automated vision-based surveillance system to detect drowning incidents in swimming pools. *Advances in Science and Engineering Technology International Conferences (ASET)* (pp. 1-5). IEEE, viewed online 20th September 2022 <<https://ieeexplore.ieee.org/abstract/document/9118248>>
- AngelEye, 2022, Installation, *AngelEye* , viewed online 21st September 2022 <<https://www.angeleye.tech/en/en-installation/>>
- Australian Institute of Health and Welfare, 2023, Injury in Australia: Drowning and Submersion, *Australian Institute of Health and Welfare*, viewed online 1st October 2023 <<https://www.aihw.gov.au/reports/injury/drowning-and-submersion>>
- Bliss, L., 2021, How a Bike Safety Bot Became a Building Block for Computer Engineers, *Bloomberg News*, viewed online 10th July 2023 <<https://www.bloomberg.com/news/articles/2021-03-11/oak-d-device-paves-the-way-for-future-spatial-ai>>
- Buber, E., & Diri, B., 2018, Performance Analysis and CPU vs GPU Comparison for Deep Learning, *In 2018 6th International Conference on Control Engineering & Information Technology (CEIT)*, (pp. 1-6). Istanbul, Turkey
- Cepeda-Pacheco, J. C., & Domingo, M. C., 2022, Deep learning and 5G and beyond for child drowning prevention in swimming pools, *Sensors*, 22(19), 7684. doi: 10.3390/s22197684
- Eng, H.L., Toh, K.A., Yau, W.Y. and Wang, J., 2008. DEWS: A live visual surveillance system for early drowning detection at pool. *IEEE transactions on circuits and systems for video technology*, 18(2), pp.196-210, viewed online 18th September 2022 <<https://ieeexplore.ieee.org/abstract/document/4399966>>
- Engineers Australia, Code of Ethics and Guidelines on Professional Conduct, 2023, *Engineers Australia*, Barton ACT, viewed online 6th June 2023

<<https://www.engineersaustralia.org.au/sites/default/files/2022-08/code-ethics-guidelines-professional-conduct-2022.pdf>>

Gandhi, R., 2018, R-CNN, Fast R-CNN, Faster R-CNN, YOLO – Object Detection Algorithms; Understanding Object Detection Algorithms, *Towards Data Science*, viewed online 29th April 2023 < <https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e>>

Great Lakes Surf Rescue Project, 2013, Signs of Drowning, *Great Lakes Surf Rescue Project*, viewed online 13th May 2023, < <https://glsrc.org/signs-of-drowning/>>

Handalage, U., Nikapotha, N., Subasinghe, C., Prasanga, T., Thilakarthna, T. and Kasthurirathna, D., 2021, Computer Vision Enabled Drowning Detection System. In *2021 3rd International Conference on Advancements in Computing (ICAC)* (pp. 240-245). IEEE, viewed online 5th October 2022 < <https://ieeexplore.ieee.org/abstract/document/9671126>>

Hasan, S., Joy, J., Ahsan, F., Khambaty, H., Agarwal, M. and Mounsef, J., 2021, A Water Behavior Dataset for an Image-Based Drowning Solution. In *2021 IEEE Green Energy and Smart Systems Conference (IGESSC)* (pp. 1-5). IEEE, viewed online 6th October 2022 < <https://ieeexplore.ieee.org/abstract/document/9618700>>

Jalalifar, S., Kashizadeh, A., Mahmood, I., Belford, A., Drake, N., Razmjou, A. and Asadnia, M., 2022. A smart multi-sensor device to detect distress in swimmers. *Sensors*, 22(3), p.1059. viewed online 25th September 2022 < <https://www.mdpi.com/1424-8220/22/3/1059>>

Jocher G., Waxmann S, 2023, COCO Dataset, *Ultralytics*, viewed 20th June 2023, <<https://docs.ultralytics.com/datasets/detect/coco/>>

Kaur, R., Singh, S., 2023, A comprehensive review of object detection with deep learning, *Digital Signal Processing*, viewed online 23rd June 2023 <<https://www.sciencedirect.com/science/article/pii/S1051200422004298>>

Konishi, N., Ishigaki, Y., Nakada, T., Nemoto, W., Inuma, S., Hoshino, T. & Ohkawara, K., 2022, Development and demonstration of a prototype system for the early detection of drowning. *2022 International Electrical Engineering Congress (iEECON)*, 9-11 March 2022 2022. 1-2.

Labudzki, R., Legutko, S., & Raos, P., 2014, The essence and applications of machine vision. *Tehnicki Vjesnik*, viewed online 19th April 2023 <

https://www.researchgate.net/profile/Stanislaw-Legutko/publication/286283684_The_essence_and_applications_of_machine_vision/links/568d5a7908aeaa1481ae4d9b/The-essence-and-applications-of-machine-vision.pdf

Lakshmana V, Gorner M, Gillard R, 2021, Practical Machine Learning for Computer Vision: End to End Machine Learning for Images, *O'Reilly Media*, Sebastopol California.

Lei, F., Zhu, H., Tang, F., & Wang, X., 2022, Drowning behaviour detection in swimming pool based on deep learning. *Signal, Image and Video Processing*, 1-8.

Louridas, P, 2020, Algorithms, *The MIT Press*, Cambridge, Massachusetts

Luxonis, 2023, gen2-yolo, *Github*, accessed online 10th August 2023
<<https://github.com/luxonis/depthai-experiments/tree/master/gen2-yolo>>

Luxonis, 2023, Oak-D, *Luxonis*, viewed online 10th July 2023
<<https://shop.luxonis.com/collections/oak-cameras-1/products/oak-d>>

Mahendran, J. K., Barry, D. T., Nivedha, A. K., & Bhandarkar, S. M., 2021, Computer Vision-based Assistance System for the Visually Impaired Using Mobile Edge Artificial Intelligence. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (pp. 2418-2427). Nashville, TN, USA

Malhotra, P. & Garg, E. 2020, Object Detection Techniques: A Comparison, *7th International Conference on Smart Structures and Systems (ICSSS)*

Mathworks, 2023, What is a Convolutional Neural Network? Three things you need to know, *Mathworks*, viewed 29th April 2023, < <https://au.mathworks.com/discovery/convolutional-neural-network-matlab.html>>

Miller D., Goode G., Bennie C., Moghadam P., Jurdak R., 2022, Why Object Detectors Fail: Investigating the Influence of the Dataset, *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 4823-4830

Nelson, J., 2020, How to Label Images for Computer Vision Models, *Roboflow*, viewed online 11th July 2023 < <https://blog.roboflow.com/tips-for-how-to-label-images/>>

Niu, Q., Wang, Y., Yuan, S., Li, K., & Wang, X., 2022,. An indoor pool drowning risk detection method based on improved YOLOv4. In *2022 IEEE 5th Advanced Information*

Management, Communicates, Electronic and Automation Control Conference (IMCEC) (pp. 1559-1563). Chongqing, China.

Padilla, R., Netto, S. L., & da Silva, E. A. B., 2020, A Survey on Performance Metrics for Object-Detection Algorithms. *In 2020 International Conference on Systems, Signals and Image Processing (IWSSIP)* (pp. 237-242). Niteroi, Brazil

Pavithra, P., Nandini, S., Nanthana, A., Aslam, N.T., & Praveen, P.K., 2021, Video Based Drowning Detection System. *In 2021 International Conference on Design Innovations for 3Cs Compute Communicate Control (ICDI3C)* (pp. 203-206). Bangalore, India.

Peden, A. E., Franklin, R. C. & Clemens, T. 2021. Can child drowning be eradicated? A compelling case for continued investment in prevention, *Acta Paediatrica*, viewed online 1st October 2022 <<https://onlinelibrary-wiley.com/doi/full/10.1111/apa.15618>>

Peden, A. E., Scarr, J. P. & Mahony, A. J. 2021. Analysis of fatal unintentional drowning in Australia 2008–2020: implications for the Australian Water Safety Strategy, *Australian and New Zealand journal of public health*, 45, 248-254, viewed online 30th September 2022 <<https://onlinelibrary-wiley-com.ezproxy.usq.edu.au/doi/10.1111/1753-6405.13124>>

Peden, A. E. & Franklin, R. C. 2020. Causes of distraction leading to supervision lapses in cases of fatal drowning of children 0–4 years in Australia: A 15-year review. *Journal of paediatrics and child health*, 56, 450-456, viewed online 30th September 2022 <<https://onlinelibrary-wiley-com.ezproxy.usq.edu.au/doi/full/10.1111/jpc.14668>>

Poseidon by Maytronics 2022, Poseidon Drowning Prevention, *Maytronics*, viewed online 21st September 2022 <<https://drowningprevention.com.au/>>

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

Redmon, J. and Farhadi, A., 2018. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.

Royal Life Saving Australia 2018, Trends in Child Drowning Over the Last 25 Years Research Report, *Royal Life Saving Australia*, Sydney New South Wales, viewed online 2nd October 2022 <

https://www.royallifesaving.com.au/__data/assets/pdf_file/0005/37526/RLS_ChildDrowning_25yrReport.pdf>

Ruiz-Zafra, A., Precioso, D., Salvador, B., Lubián-López, S.P., Jiménez, J., Benavente-Fernández, I., Pigueiras, J., Gómez-Ullate, D., & Gontard, L.C., 2023, NeoCam: An edge-cloud platform for non-invasive real-time monitoring in neonatal intensive care units. *IEEE Journal of Biomedical and Health Informatics*.

Salehi, N., Keyvanara, M. and Monadjemmi, S.A., 2016. An automatic video-based drowning detection system for swimming pools using active contours. *Int. J. Image, Graph. Signal Process*, 8(8), pp.1-8, viewed online 5th October 2022

<https://www.researchgate.net/profile/Nasrin-Salehi-5/publication/305877670_An_Automatic_Video-based_Drowning_Detection_System_for_Swimming_Pools_Using_Active_Contours/links/5f4092ea6fdccdb82eac83/An-Automatic-Video-based-Drowning-Detection-System-for-Swimming-Pools-Using-Active-Contours.pdf>

Shah, R.M., Sainath, B. and Gupta, A., 2022, July. Comparative Performance Study of CNN-based Algorithms and YOLO. In *2022 IEEE International Conference on Electronics, Computing and Communication Technologies* (pp. 1-6). IEEE.

Shatnawi, M., Albreiki, F., Alkhoori, A., & Alhebshi, M, 2023, Deep Learning and Vision-Based Early Drowning Detection. *Information*, 14(1), 52

Shehata, A.M., Mohamed, E.M., Salem, K.L., Mohamed, A.M., Salam, M.A. and Gamil, M.M., 2021, May. A Survey of Drowning Detection Techniques. In *2021 International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC)* (pp. 286-290). IEEE, viewed online 19th September 2022 < <https://ieeexplore.ieee.org/abstract/document/9447677>>

Smith, M. L., Smith, L. N. & Hansen, M. F. 2021. The quiet revolution in machine vision - a state-of-the-art survey paper, including historical review, perspectives, and future directions. *Computers in Industry*, 130, 103472.

The Guardian, 2019, Go game master quits saying machines ‘cannot be defeated’, *The Guardian*, 27th November, viewed 29th April 2023, <<https://www.theguardian.com/world/2019/nov/27/go-game-master-quits-saying-machines-cannot-be-defeated>>

Ultralytics, 2023, 'Tips for Best Training Results', *Ultralytics*. Viewed online 31st July 2023 <https://docs.ultralytics.com/yolov5/tutorials/tips_for_best_training_results/>

Vestnikov, R., Stepanov, D., & Bakhshiev, A., 2023, Development of Neural Network Algorithms for Early Detection of Drowning in Swimming Pools. *Paper presented at the 2023 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM)*, Sochi, Russian Federation, pp. 820-824

Wang, C.Y., Bochkovskiy, A., & Liao, H.Y.M., 2023, YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. *In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 7464-7475).

Wave Drowning Detection Systems 2022, Wave W10, *Wave Drowning Detection Systems*, viewed online 20th September 2022 <<https://www.wavedds.com/w10>>

World Health Organisation 2014, Global Report on Drowning: Preventing a Leading Killer, *World Health Organisation*, Geneva, Switzerland, viewed online 5th October 2022 <<https://www.who.int/publications/i/item/global-report-on-drowning-preventing-a-leading-killer>>

Yang, D., Cao, Y., Feng, Y., Lai, X., & Pan, Z., 2021, Drowning detection algorithm for intelligent lifebuoy. *In Proceedings of the 2021 International Conference on Unmanned Systems (ICUS)* (pp. 512-519). doi: 10.1109/ICUS52573.2021.9641291.

Yun, S., & Kim, S., 2019, Recurrent YOLO and LSTM-based IR single pedestrian tracking. *In 2019 19th International Conference on Control, Automation and Systems (ICCAS)*, Jeju, Korea (South) (pp. 94-96).

Zou, Z., Chen, K., Shi, Z., Guo, Y. and Ye, J., 2023. Object detection in 20 years: A survey. *Proceedings of the IEEE*.

Appendix A

ENG4111/4112 Research Project

Project Specification

For: Fern Proctor

Title: Real Time Drowning Detection System Using Machine Vision

Major: Electrical and Electronic Engineering

Supervisors: Dr. Tobias Low

Enrollment: ENG4111 – EXT S1, 2023

ENG4112 – EXT S2, 2023

Project Aim: To develop a real time drowning detection system using a surface mounted camera in a residential pool setting.

Programme: Version 1, 20th February 2023

- 1) Conduct background research drowning detection systems and their implementation. This would include systems which are commercially available.
- 2) Review systems and algorithms which utilise machine learning and vision for detecting people and behavior patterns.
- 3) Research and secure an appropriate dataset for use in the model for drowning detection. If an existing set cannot be found, research drowning behaviors and create a dataset based on simulated images/videos.
- 4) Investigate and select appropriate hardware which can be used to implement a drowning detection system.
- 5) Investigate and select an appropriate user interface for alerting personnel of a drowning event.

- 6) Select a suitable software development environment in which an appropriate algorithm can be implemented.
- 7) Develop a prototype of the system using the selected hardware.
- 8) Develop a machine vision/learning algorithm for drowning detection.
- 9) Deploy the prototype and algorithm in a suitable environment. Record and collect the data for analysis.
- 10) Analyse and evaluate the data. Assess the effectiveness of the prototype.

ENG4111/4112 Research Project

Project Resources

For: Fern Proctor

Title: Real Time Drowning Detection System Using Machine Vision

Major: Electrical and Electronic Engineering

Supervisors: Dr. Tobias Low

Enrollment: ENG4111 – EXT S1, 2023


ENG4112 – EXT S2, 2023

This document outlines the project resources that are likely to be required for successful completion.

Version 1, 20th February 2023

| Item | Estimated Cost | Comments |
|-----------------------------|-----------------|--|
| Raspberry Pi 4 8GB | \$123 | Currently out of stock at many retailers |
| Oak D-Lite Camera | \$277 | In stock – Core Electronics |
| Open CV | \$0 | Free to download for Windows/Linux |
| Python 3 | \$0 | Free to download for Windows/Linux |
| Laptop | \$0 | Currently own an appropriate laptop |
| Test Image Set | \$-- | Need to source or generate own set |
| Test location (pool) | \$0 | Pool available for use |
| Wifi Antenna | \$10 | Must use Raspberry Pi Compute Module 4 kit |
| Enclosure | \$40 | To be confirmed – select appropriate enclosure |
| Raspberry Pi Power Supply | \$16.45 | Available Core Electronics (official Power Supply) |
| Misc Hardware | \$-- | To be confirmed |
| Total Estimated Cost | \$466.45 | |

Appendix B – Risk Assessment

| NUMBER | RISK DESCRIPTION | TREND | CURRENT | RESIDUAL |
|---|---|---|-----------------------|----------|
| 2486 | Risk Management Plan - Real Time Drowning Detection System |  | Low | Low |
| DOCUMENTS REFERENCED | | | | |
| ISO 31000 Risk management | | | | |
| RISK OWNER | RISK IDENTIFIED ON | LAST REVIEWED ON | NEXT SCHEDULED REVIEW | |
| Fern Proctor | 24/05/2023 | | | |
| RISK FACTOR(S) | EXISTING CONTROL(S) | PROPOSED CONTROL(S) | OWNER | DUE DATE |
| Hazards: 1) Hazards as a result of poor house keeping around work area 2) Slip and trip hazards 3) Hazards as a result of damaged or broken equipment | Control: 1) Ensure house keeping is maintained throughout project. 2) Clear any potential slip or trip hazards 3) Inspect equipment for damage prior to use. | | | |

| | | |
|--|---|--|
| <p>Hazards: 1) Electrical cabling used for powering devices 2) Faulted devices causing short circuit 3) Electrical equipment being used around swimming pools.</p> | <p>Control: Equipment in use to be to Australian Standards Inspect equipment prior to use Equipment not to be used within the footprint (inside fence) of swimming area if it is supplied by 230VAC. Adequate protection - earth leakage and circuit breakers.</p> | |
| <p>Hazards: 1) Sun exposure 2) Heat stress due to adverse weather conditions 3) Other hazards related to adverse weather conditions - rain, lightening</p> | <p>Control: Sunscreen to be used, adequate clothing for sun protection where possible.</p> <p>Control: Task rotation, rest breaks as required. Work not to be conducted in adverse weather (lightening, storms).</p> | |
| <p>Hazards: Persons in swimming pool for data collection and testing purposes. Wet and slippery surfaces around swimming pool</p> | <p>Control: Competent swimmers only to be used in data collection and testing. Spotters in place to monitor swimmers.</p> | |

| | | |
|---|---|--|
| | <p>Control: Signage and barricades in place around swimming pool as per Queensland standards.</p> <p>Control: No running around pool area. Eyes on path and awareness when accessing swimming pools.</p> | |
| Hazard: Loss of project work due to equipment failure | <p>Control: Back up project using Google Drive. Both personal and university account. Minimum of weekly back ups to be done.</p> | |
| Risk of project being used as a drowning detection system as substitution for adequate supervision around swimming pools. | <p>Control: Project information and hardware not to be distributed publicly for use. Emphasis on project being a prototype and not to be used in place of appropriate supervision in swimming pool settings.</p> | |

Appendix C – Dataset Release Agreement

Water Behavior Dataset Release Agreement


To advance the state-of-the-art in water behavior recognition, the Water Behavior dataset will be made available to researchers in water behavior recognition on a case-by-case basis only. All requests for the Water Behavior dataset must be submitted by email to jmbcad@rit.edu. To receive a copy of the dataset, the researcher must sign this document and thereby agree to observe the restrictions listed herein. Failure to observe the restrictions in this document will result in access being denied for the balance of the Water Behavior dataset and being subject to civil damages in the case of publication of images that have not been approved for release, a violation of restriction 3 below. The researcher(s) agrees to the following restrictions on the dataset:

1. The dataset will not be further distributed, published, copied, or further disseminated in any way or form whatsoever, whether for profit or not. This includes further distributing, copying or disseminating to a facility or organization unit in the requesting university, organization, or company.
2. The videos will only appear in technical reports, technical papers, and technical documents reporting on water behavior research. There will be no more than 8 images used at a time in a publication.
3. All documents and papers that report on research that uses the Water Behavior dataset will acknowledge the use of the Water Behavior dataset. Use of the Water Behavior dataset will be acknowledged as follows: "Portions of the research in this paper use the Water Behavior dataset collected under the Electrical Engineering department at Rochester Institute of Technology Dubai" and citation to:

S. Hasan, J. Joy, F. Ahsan, H. Khambaty, M. Agarwal and J. Mounsef
" A Water Behavior Dataset for an Image-Based Drowning Solution,"
In 2021 IEEE Green Energy and Smart Systems Conference (IGESSC), pp.
1-5, 2021.

I hereby agree to the aforementioned restrictions:

Name: fern PROCTOR

Signature: 

Appendix D – Google Colab Notebooks and Python Script

This appendix contains the configuration files and Google Colab notebooks used to train the custom models as well as the final python script used in deployment of the Oak-D Lite.

YoloV7-tiny Custom

Data File

```
# directories for train, validate and test images
train: ./data/train
val: ./data/validate
test: ./data/test

# number of classes
nc: 3

# class names
names: [ 'drowning','idle','swimming' ]
```

Configuration File

```
# parameters
nc: 3 # number of classes
depth_multiple: 1.0 # model depth multiple
width_multiple: 1.0 # layer channel multiple

# anchors
anchors:
  - [10,13, 16,30, 33,23] # P3/8
  - [30,61, 62,45, 59,119] # P4/16
  - [116,90, 156,198, 373,326] # P5/32

# yolov7-tiny backbone
backbone:
  # [from, number, module, args] c2, k=1, s=1, p=None, g=1, act=True
  [[-1, 1, Conv, [32, 3, 2, None, 1, nn.LeakyReLU(0.1)]], # 0-P1/2

    [-1, 1, Conv, [64, 3, 2, None, 1, nn.LeakyReLU(0.1)]], # 1-P2/4

    [-1, 1, Conv, [32, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
    [-2, 1, Conv, [32, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
    [-1, 1, Conv, [32, 3, 1, None, 1, nn.LeakyReLU(0.1)]],
    [-1, 1, Conv, [32, 3, 1, None, 1, nn.LeakyReLU(0.1)]],
    [[-1, -2, -3, -4], 1, Concat, [1]],
    [-1, 1, Conv, [64, 1, 1, None, 1, nn.LeakyReLU(0.1)]], # 7

    [-1, 1, MP, []], # 8-P3/8
    [-1, 1, Conv, [64, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
    [-2, 1, Conv, [64, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
    [-1, 1, Conv, [64, 3, 1, None, 1, nn.LeakyReLU(0.1)]],
    [-1, 1, Conv, [64, 3, 1, None, 1, nn.LeakyReLU(0.1)]],
    [[-1, -2, -3, -4], 1, Concat, [1]],
    [-1, 1, Conv, [128, 1, 1, None, 1, nn.LeakyReLU(0.1)]], # 14
```

```

[-1, 1, MP, []], # 15-P4/16
[-1, 1, Conv, [128, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
[-2, 1, Conv, [128, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
[-1, 1, Conv, [128, 3, 1, None, 1, nn.LeakyReLU(0.1)]],
[-1, 1, Conv, [128, 3, 1, None, 1, nn.LeakyReLU(0.1)]],
[[-1, -2, -3, -4], 1, Concat, [1]],
[-1, 1, Conv, [256, 1, 1, None, 1, nn.LeakyReLU(0.1)]], # 21

[-1, 1, MP, []], # 22-P5/32
[-1, 1, Conv, [256, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
[-2, 1, Conv, [256, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
[-1, 1, Conv, [256, 3, 1, None, 1, nn.LeakyReLU(0.1)]],
[-1, 1, Conv, [256, 3, 1, None, 1, nn.LeakyReLU(0.1)]],
[[-1, -2, -3, -4], 1, Concat, [1]],
[-1, 1, Conv, [512, 1, 1, None, 1, nn.LeakyReLU(0.1)]], # 28
]

# yolov7-tiny head
head:
[[-1, 1, Conv, [256, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
[-2, 1, Conv, [256, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
[-1, 1, SP, [5]],
[-2, 1, SP, [9]],
[-3, 1, SP, [13]],
[[-1, -2, -3, -4], 1, Concat, [1]],
[-1, 1, Conv, [256, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
[[-1, -7], 1, Concat, [1]],
[-1, 1, Conv, [256, 1, 1, None, 1, nn.LeakyReLU(0.1)]], # 37

[-1, 1, Conv, [128, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
[-1, 1, nn.Upsample, [None, 2, 'nearest']],
[21, 1, Conv, [128, 1, 1, None, 1, nn.LeakyReLU(0.1)]], # route backbone
P4
[[-1, -2], 1, Concat, [1]],

[-1, 1, Conv, [64, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
[-2, 1, Conv, [64, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
[-1, 1, Conv, [64, 3, 1, None, 1, nn.LeakyReLU(0.1)]],
[-1, 1, Conv, [64, 3, 1, None, 1, nn.LeakyReLU(0.1)]],
[[-1, -2, -3, -4], 1, Concat, [1]],
[-1, 1, Conv, [128, 1, 1, None, 1, nn.LeakyReLU(0.1)]], # 47

[-1, 1, Conv, [64, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
[-1, 1, nn.Upsample, [None, 2, 'nearest']],
[14, 1, Conv, [64, 1, 1, None, 1, nn.LeakyReLU(0.1)]], # route backbone
P3
[[-1, -2], 1, Concat, [1]],

[-1, 1, Conv, [32, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
[-2, 1, Conv, [32, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
[-1, 1, Conv, [32, 3, 1, None, 1, nn.LeakyReLU(0.1)]],
[-1, 1, Conv, [32, 3, 1, None, 1, nn.LeakyReLU(0.1)]],
[[-1, -2, -3, -4], 1, Concat, [1]],
[-1, 1, Conv, [64, 1, 1, None, 1, nn.LeakyReLU(0.1)]], # 57

[-1, 1, Conv, [128, 3, 2, None, 1, nn.LeakyReLU(0.1)]],
[[-1, 47], 1, Concat, [1]],

[-1, 1, Conv, [64, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
[-2, 1, Conv, [64, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
[-1, 1, Conv, [64, 3, 1, None, 1, nn.LeakyReLU(0.1)]],

```

```

[-1, 1, Conv, [64, 3, 1, None, 1, nn.LeakyReLU(0.1)]],
[[-1, -2, -3, -4], 1, Concat, [1]],
[-1, 1, Conv, [128, 1, 1, None, 1, nn.LeakyReLU(0.1)]], # 65

[-1, 1, Conv, [256, 3, 2, None, 1, nn.LeakyReLU(0.1)]],
[[-1, 37], 1, Concat, [1]],

[-1, 1, Conv, [128, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
[-2, 1, Conv, [128, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
[-1, 1, Conv, [128, 3, 1, None, 1, nn.LeakyReLU(0.1)]],
[-1, 1, Conv, [128, 3, 1, None, 1, nn.LeakyReLU(0.1)]],
[[-1, -2, -3, -4], 1, Concat, [1]],
[-1, 1, Conv, [256, 1, 1, None, 1, nn.LeakyReLU(0.1)]], # 73

[57, 1, Conv, [128, 3, 1, None, 1, nn.LeakyReLU(0.1)]],
[65, 1, Conv, [256, 3, 1, None, 1, nn.LeakyReLU(0.1)]],
[73, 1, Conv, [512, 3, 1, None, 1, nn.LeakyReLU(0.1)]],

[[74,75,76], 1, IDetect, [nc, anchors]], # Detect(P3, P4, P5)

```

Google Colab Notebook

```

#mount drive
from google.colab import drive
drive.mount('/content/gdrive')

# %cd /content/gdrive/MyDrive
import os
if not os.path.isdir("Yolov7"):
    os.makedirs("Yolov7")

#move to newly created YoloV7 directory

# %cd Yolov7

#clone repo yolov7
!git clone https://github.com/WongKinYiu/yolov7.git
# %cd /content/gdrive/MyDrive/Yolov7/yolov7
"""At this point, drop the image and label files into the data folder on
google drive"""

# download Yolov7-Tiny Weights
#!wget
https://github.com/WongKinYiu/yolov7/releases/download/v0.1/yolov7.pt this
is the standard yolo7 version

# Below is the Tiny Version of YoloV7
!wget https://github.com/WongKinYiu/yolov7/releases/download/v0.1/yolov7-
tiny.pt

!pip install PyYAML==5.4.1

#Train the YoloV7-tiny model
!python train.py --device 0 --batch-size 16 --data data/custom_data.yaml --
img 640 640 --cfg cfg/training/yolov7tinycustom.yaml --weights '' --name
yolov7tinycustomweights --hyp data/hyp.scratch.tiny.yaml --epochs 24

# %cd /content/gdrive/MyDrive/Yolov7/yolov7
#In the event of an error, use this command to resume training from the
last epoch.

```

```

!python train.py -resume

#Once training has finished, change to test mode
!python test.py --data data/custom_data.yaml --img 640 --weights
runs/train/yolov7tinycustomweights/weights/last.pt --task 'test'

# Run evaluation on an image
!python detect.py --weights
runs/train/yolov7tinycustomweights/weights/last.pt --conf 0.1 --source
/content/gdrive/MyDrive/images/test_idle_000012.jpg

#Run a detection on a video
!python detect.py --weights
runs/train/yolov7tinycustomweights/weights/last.pt --conf 0.25 --img-size
640 --source /content/gdrive/MyDrive/images/te_o_swim_5.mp4

#Run a detection on a different video

!python detect.py --weights
runs/train/yolov7tinycustomweights/weights/last.pt --conf 0.25 --img-size
640 --source /content/gdrive/MyDrive/images/te_o_drown_5.mp4

```

YoloV8-s Custom

Data File

```

#Custom Dataset
path: /content/gdrive/MyDrive/Yolov7/yolov7/data # dataset root dir
train: ./train # train images
val: ./test # val images
test: ./test #test images

# Classes
names:
  0: drowning
  1: idle
  2: swimming

```

Google Colab Notebook

```

Fern YoloV8

#mount drive

from google.colab import drive
drive.mount('/content/gdrive')

# %cd /content/gdrive/MyDrive

import os

if not os.path.isdir("yolov8"):
    os.makedirs("yolov8")

#move to newly created YoloV8 directory

# %cd yolov8

!git clone https://github.com/ultralytics/ultralytics.git

!pip install ultralytics

import ultralytics

```



```

ultralalytics.checks()

from ultralytics import YOLO

!pwd

# Load a model

model = YOLO("yolov8m.pt")      # load a pretrained model (recommended for
training)

# %cd

!yolo task=detect mode=predict model=yolov8m.pt conf=0.25 source =
/content/gdrive/MyDrive/images/skateboards.jpg save=True

!pwd

# Train YOLOv8n on Custom Dataset for 24 epochs (6000 Iterations)

#!yolo train model=yolov8s.pt data=yolov8custom.yaml epochs=24 imgsz=640

# Alternative model trial

#!yolo train data=yolov8custom.yaml model=yolov8s.yaml
pretrained=yolov8s.pt epochs=24 imgsz=640

#Another alternative trial

!yolo train data=yolov8custom.yaml model=yolov8m.pt pretrained=yolov8m.pt
epochs=300 imgsz=416 batch =32 lr0 =0.001 lrf =0.001 degrees = 90 translate
= 0.4 scale = 0.4 shear =0.4 flipud = 0.4 fliplr = 0.4 patience =0

!yolo mode=train resume
model=/content/gdrive/MyDrive/yolov8/runs/detect/train15/weights/last.pt
data=yolov8custom.yaml epochs=300 imgsz=416 batch =32 lr0 =0.001 lrf
=0.001 degrees = 90 translate = 0.4 scale = 0.4 shear =0.4 flipud = 0.4
fliplr = 0.4 patience =0

!yolo mode=val
model=/content/gdrive/MyDrive/yolov8/runs/detect/train15/weights/best.pt
data=yolov8custom.yaml epochs=24 imgsz=416

!yolo detect mode=val
model=/content/gdrive/MyDrive/yolov8/runs/detect/train15/weights/best.pt
data=yolov8custom.yaml

!yolo detect mode=val
model=/content/gdrive/MyDrive/yolov8/runs/detect/train15/weights/best.pt
data=yolov8custom.yaml

!yolo task=detect mode=predict
model=/content/gdrive/MyDrive/yolov8/runs/detect/train15/weights/best.pt
conf=0.25 source =
/content/gdrive/MyDrive/images/test_swimming_2_000041.jpg

!yolo task=detect mode=predict
model=/content/gdrive/MyDrive/yolov8/runs/detect/train15/weights/best.pt
conf=0.25 source = /content/gdrive/MyDrive/images/te_o_swim_5.mp4

```

YoloV8-m Custom – Final Model

Datafile

```
#Custom Dataset
path: /content/gdrive/MyDrive/Yolov7/yolov7/data # dataset root dir
train: ./train # train images
val: ./test # val images
test: ./test #test images

# Classes
names:
  0: drowning
  1: idle
  2: swimming
```

Google Colab Notebook

```
"""Fern YoloV8

#mount drive
from google.colab import drive
drive.mount('/content/gdrive')
# %cd /content/gdrive/MyDrive
import os
if not os.path.isdir("yolov8"):
    os.makedirs("yolov8")
#move to newly created YoloV8 directory
# %cd yolov8
!git clone https://github.com/ultralytics/ultralytics.git
!pip install ultralytics
import ultralytics
ultralytics.checks()
from ultralytics import YOLO
!pwd
# Load a model
model = YOLO("yolov8m.pt") # load a pretrained model (recommended for
training)
# %cd
!yolo task=detect mode=predict model=yolov8m.pt conf=0.25 source =
/content/gdrive/MyDrive/images/skateboards.jpg save=True
!pwd
# Train YOLOv8n on Custom Dataset for 300 epochs
!yolo train data=yolov8custom.yaml model=yolov8m.pt pretrained=yolov8m.pt
epochs=300 imgsz=416 batch =32 lr0 =0.001 lrf =0.001 degrees = 90 translate
= 0.4 scale = 0.4 shear =0.4 flipud = 0.4 fliplr = 0.4 patience =0
```

```

!yolo mode=train resume
model=/content/gdrive/MyDrive/yolov8/runs/detect/train15/weights/last.pt
data=yolov8custom.yaml epochs=300 imgsz=416 batch =32 lr0 =0.001 lrf
=0.001 degrees = 90 translate = 0.4 scale = 0.4 shear =0.4 flipud = 0.4
fliplr = 0.4 patience =0
!yolo mode=val
model=/content/gdrive/MyDrive/yolov8/runs/detect/train15/weights/best.pt
data=yolov8custom.yaml epochs=24 imgsz=416
!yolo detect mode=val
model=/content/gdrive/MyDrive/yolov8/runs/detect/train15/weights/best.pt
data=yolov8custom.yaml
!yolo detect mode=val
model=/content/gdrive/MyDrive/yolov8/runs/detect/train15/weights/best.pt
data=yolov8custom.yaml
!yolo task=detect mode=predict
model=/content/gdrive/MyDrive/yolov8/runs/detect/train15/weights/best.pt
conf=0.25 source =
/content/gdrive/MyDrive/images/test_swimming_2_000040.jpg
!yolo task=detect mode=predict
model=/content/gdrive/MyDrive/yolov8/runs/detect/train15/weights/best.pt
conf=0.25 source = /content/gdrive/MyDrive/images/te_o_swim_5.mp4

```

Python Script

The following is the Python script which was deployed on the Raspberry Pi for the user interface.

```

# Experiment 5 - Interface and Alert System for Drowning Detection System
# This code has been adapted from https://github.com/luxonis/depthai-experiments/tree/master/gen2-yolo/device-decoding

# Import libraries
from pathlib import Path
import sys
import cv2
import depthai as dai
import numpy as np
import time
import argparse
import json
import blobconverter

```

```

#Define FPS
fps = 5

# parse arguments
parser = argparse.ArgumentParser()
parser.add_argument("-m", "--model", help="Provide model name or model path
for inference",
                    default='model/yolov8-300e-
best_openvino_2022.1_6shave.blob', type=str)
parser.add_argument("-c", "--config", help="Provide config path for
inference",
                    default='json/yolov8-300e-best.json', type=str)
args = parser.parse_args()

# Get config
configPath = Path(args.config)
if not configPath.exists():
    raise ValueError("Path {} does not exist!".format(configPath))

# Load JSON file for metadata
with configPath.open() as f:
    config = json.load(f)
nnConfig = config.get("nn_config", {})

# Get input shape
if "input_size" in nnConfig:
    W, H = tuple(map(int, nnConfig.get("input_size").split('x')))

# extract metadata from JSON file
metadata = nnConfig.get("NN_specific_metadata", {})
classes = metadata.get("classes", {})
coordinates = metadata.get("coordinates", {})
anchors = metadata.get("anchors", {})
anchorMasks = metadata.get("anchor_masks", {})
iouThreshold = metadata.get("iou_threshold", {})
confidenceThreshold = metadata.get("confidence_threshold", {})

print(metadata)

# Get labels
nnMappings = config.get("mappings", {})

```

```

labels = nnMappings.get("labels", {})

# get model path from .blob file
nnPath = args.model
if not Path(nnPath).exists():
    print("No blob found at {}. Looking into DepthAI model
zoo.".format(nnPath))
    nnPath = str(blobconverter.from_zoo(args.model, shaves = 6, zoo_type =
"depthai", use_cache=True))
# sync outputs
syncNN = True

# Create pipeline to Oak-D Lite
pipeline = dai.Pipeline()

# Define sources and outputs
camRgb = pipeline.create(dai.node.ColorCamera)
detectionNetwork = pipeline.create(dai.node.YoloDetectionNetwork)
xoutRgb = pipeline.create(dai.node.XLinkOut)
nnOut = pipeline.create(dai.node.XLinkOut)

xoutRgb.setStreamName("rgb")
nnOut.setStreamName("nn")

# Properties
camRgb.setPreviewSize(W, H)

# Use this to change the camera resolution
camRgb.setResolution(dai.ColorCameraProperties.SensorResolution.THE_13_MP)

camRgb.setInterleaved(False)
camRgb.setColorOrder(dai.ColorCameraProperties.ColorOrder.BGR)
camRgb.setFps(fps)

# Network specific settings
detectionNetwork.setConfidenceThreshold(confidenceThreshold)
detectionNetwork.setNumClasses(classes)
detectionNetwork.setCoordinateSize(coordinates)
detectionNetwork.setAnchors(anchors)
detectionNetwork.setAnchorMasks(anchorMasks)
detectionNetwork.setIouThreshold(iouThreshold)

```

```

detectionNetwork.setBlobPath(nnPath)
detectionNetwork.setNumInferenceThreads(2)
detectionNetwork.input.setBlocking(False)

# Linking
camRgb.preview.link(detectionNetwork.input)
detectionNetwork.passthrough.link(xoutRgb.input)
detectionNetwork.out.link(nnOut.input)

# Connect to Oak-D Lite and start pipeline
with dai.Device(pipeline) as device:

    # Output queues will be used to get the rgb frames and nn data from the
    outputs defined above
    qRgb = device.getOutputQueue(name="rgb", maxSize=4, blocking=False)
    qDet = device.getOutputQueue(name="nn", maxSize=4, blocking=False)

    frame = None
    detections = []
    startTime = time.monotonic()
    counter = 0
    color2 = (0, 0, 255)

    # nn data, being the bounding box locations, are in <0..1> range - they
    need to be normalized with frame width/height
    def frameNorm(frame, bbox):
        normVals = np.full(len(bbox), frame.shape[0])
        normVals[::2] = frame.shape[1]
        return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)

    def displayFrame(name, frame, detections):
        class_colors = {
            'Drowning': (0, 0, 255),    # Red for Drowning
            'Idle': (0, 255, 0),        # Green for Idle
            'Swimming': (0, 255, 0)     # Blue for Swimming
        }

        for detection in detections:
            bbox = frameNorm(frame, (detection.xmin, detection.ymin,
detection.xmax, detection.ymax))

            #Get the class label

```

```

        class_label = labels[detection.label]

        #Colour of label
        color=class_colors.get(class_label, (0,0,255))

        #Draw the bounding boxes
        cv2.putText(frame, class_label, (bbox[0] + 10, bbox[1] + 20),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, color)
        cv2.putText(frame, f"{int(detection.confidence * 100)}%",
(bbox[0] + 10, bbox[1] + 40), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color)
        cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]),
color, 2)

        # Show the frame
        cv2.imshow(name, frame)


# Initialize recording variables
recording = False
out = None


# Initialize drowning detection timer variables
drowning_timer = 0
drowning_threshold = 2
drowning_detected = False
last_detection_time = time.monotonic() # Initialize
last_detection_time
drowning_display_timer = 0
drowning_reset_time = 2 # Get rid of drowning alert after 5 seconds


# Create a named window with the fullscreen flag
cv2.namedWindow("Drowning Detection System", cv2.WINDOW_NORMAL)
cv2.setWindowProperty("Drowning Detection System",
cv2.WND_PROP_FULLSCREEN, cv2.WINDOW_FULLSCREEN)


while True:
    inRgb = qRgb.get()
    inDet = qDet.get()

    if inRgb is not None:
        frame = inRgb.getCvFrame()

```

```

        cv2.putText(frame, "NN fps: {:.2f}".format(counter /
(time.monotonic() - startTime)),
                    (2, frame.shape[0] - 4), cv2.FONT_HERSHEY_SIMPLEX, 0.4,
color2)

if inDet is not None:
    detections = inDet.detections
    counter += 1

    # Check if 'Drowning' is detected in the current frame
    drowning_detected = False # Reset drowning detection flag
    for detection in detections:
        if labels[detection.label] == 'Drowning':
            drowning_detected = True
            break

    if drowning_detected:
        drowning_timer += 1 / fps # Increase timer based on FPS
        last_detection_time = time.monotonic()
    else:
        drowning_timer = 0 # Reset timer

    # Check if the timer threshold is met and reset the timer
    if time.monotonic() - last_detection_time >=
drowning_reset_time and drowning_timer >= drowning_threshold:
        drowning_timer = 0

    # Update the separate timer for displaying "Drowning Detected!"
text
    if drowning_detected:
        drowning_display_timer = time.monotonic()

    # Code to hold drowning detection alert even after loss of bounding
box
    if frame is not None:
        if drowning_detected and drowning_timer >= drowning_threshold:
            cv2.putText(frame, "Drowning Detected!", (10, 60),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
            elif time.monotonic() - last_detection_time <=
drowning_reset_time:

```



```

        cv2.putText(frame, "Drowning Detected!", (10, 60),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
    else:
        cv2.putText(frame, "No Drowning Detected", (10,60),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
    displayFrame("Drowning Detection System", frame, detections)

# Recording
if recording:
    if out is None:
        fourcc= cv2.VideoWriter_fourcc(*"H264")
        out = cv2.VideoWriter('output.mp4', fourcc, fps,
(frame.shape[1], frame.shape[0]))
        out.write(frame)

# Some hot key commands for control
key = cv2.waitKey(1)
if key == ord('q'):
    break
elif key == ord('r'):
    if not recording:
        recording = True
        out = None
        print("Recording started.")
elif key == ord('s'):
    if recording:
        if out is not None:
            out.release()
            print("Recording stopped and saved as 'output.mp4'.")
            recording = False

```