School of Computing

FACULTY OF ENGINEERING AND PHYSICAL SCIENCES

UNIVERSITY OF LEEDS

Final Report

Pose Estimation and Neural Network-based Classification of Drowning Incidents in Video Footage

Kacper Adam Roemer

Submitted in accordance with the requirements for the degree of BSc/MEng Computer Science with Artificial Intelligence

2022/2023

COMP3931 Individual Project

The candidate confirms that the following have been submitted*:*

Items	**Format**	**Recipient(s) and Date**
Final Report	*PDF file*	*Uploaded to Minerva (02/05/23)*
Link to online code repository	*URL https://github.com/kaperoo/Drowning-Detection*	*Sent to supervisor and assessor (02/05/23)*

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of student) Kacper Roemer

Summary

Each year, drowning incidents are considered to be a global issue. They result in many injuries and have a big psychological impact on both pool workers and victims' families. One way of addressing this issue involves creating an automatic drowning detection system capable of accurately distinguishing between regular water activities and potentially life-threatening situations. To solve this problem, different computer vision techniques could be applied to classify individual behaviours in video footage

The goal of this project was to develop a drowning detection system with the help of computer vision and deep learning techniques. Different human pose estimation methods were examined, and one was selected to analyse and extract individuals' body poses from video footage. In addition, three CNN-based models were constructed to process body keypoints, obtained with the pose estimation algorithm and predict the activity class. Finally, pose estimation, data processing and neural network classification were merged to create the promised drowning detection system

The project's deliverable is the code for the proposed drowning detection system with 4 promised neural network models. This approach demonstrates the feasibility of using computer vision to classify water-related activities. A modified version of the program could serve as a tool to support lifeguards in monitoring swimming pools.

Table of Contents

Chapter 1 - Introduction and Background Research

1.1 Introduction

Drowning accidents are a major global issue each year, with approximately 236,000 lives lost [1]. In fact, drowning ranks high among the leading causes of unintentional injury-related deaths [2]. In order to reduce accidents and enhance safety measures in aquatic environments dependable drowning detection system should be introduced. Although swimming pool drownings account for just a fraction of these incidents [3], such a tragedy can have a significant emotional impact on the victims' families and pool staff [4]. In addition, swimming facilities might be closed because of legal issues and negative publicity stemming [5].

Lifeguards and pool attendants often face challenges like the repetitive nature of their duties, which may lead to reduced situational awareness [6]. The crowded conditions in the swimming pools can also make it difficult for lifeguards to spot someone in an emergency situation [7]. These elements can hinder the timely detection and response to drowning incidents, which is crucial, as each minute of submersion increases the likelihood of severe injury or death [8]. It is expected that an automated system capable of accurately identifying drowning events using video footage could prove invaluable in assisting lifeguards with pool monitoring [9].

The primary goal of this project is to develop a detection system as a prevention method for the drowning problem. To achieve this, several human pose estimation techniques will be examined. The best one will be chosen to be incorporated into the solution. Three neural networks will be built with the purpose of classification of estimated body poses. These models will be trained on a chosen dataset to facilitate their ability to predict the class of subjects' behaviour in the input footage. Both, pose estimation and neural networks will be evaluated and combined to form three versions of the promised drowning detection system. The performance of all versions will be measured and discussed. By harnessing these state-of-the-art technologies, the proposed system should aim to support lifeguards in swiftly detecting drowning incidents and reducing the risk of tragic incidents.

The report aims to thoroughly examine the issue and review previous drowning detection techniques, approaches to pose estimation, classification methods, and relevant datasets in Chapter 1. The subsequent chapter covers all decisions made in creating a drowning recognition system as well as the methodology described in detail. The system's performance results and evaluation will occupy Chapter 3 with its presentation and analysis, followed by a discussion of its limitations and suggestions for future work to improve the proposed system's capabilities further in Chapter 4.

1.2 Previous Approaches to Automatic Drowning Detection

1.2.1 Sensor-Based Drowning Recognition: WAVEDDS Wearable System

Wave Drowning Detection Systems is a private company specialised in sensor-based drowning detection systems. They approach the problem by detecting prolonged submersion times of swimmers, particularly children [10]. To measure the submersion time, a headband equipped with water sensors must be worn by the swimmer. If the submersion exceeds a predetermined threshold, the device sends vibrating signals to bracelets worn by lifeguards, alerting them about a potential incident. However, the system has its limitations. First, wearing the headband at all times in the water might be met with resistance from swimmers [12]. Secondly, the predetermined submersion thresholds might not be able to represent drowning risk accurately due to different capabilities for holding a breath [11]. Moreover, having enough devices for all swimming pool attendees might not be feasible due to their high price.

1.2.2 Video-Based Drowning Detection System Using Active Contours

Another approach to drowning detection includes using video-based systems, such as the one implemented in the study titled "An Automatic Video-Based Drowning Detection System for Swimming Pools using Active Contours" [13]. The system described in this paper employs real-time video analysis from cameras installed around the swimming pool, ensuring complete coverage of the pool area. This method involves converting video frames to the HSV colour space, which separates luminance data from colour information, making it suitable for tracking purposes [14]. A threshold is then applied to the HSV values to create a binary image, which distinguishes swimmers from the background. Contour detection is used to find object outlines in the binary image [15], and the algorithm selects and tracks the contour with the largest area, representing the swimmer in consecutive frames. This approach aims to detect a swimmer missing in video frames for a predefined period of time, triggering an alarm to the rescue team if above a certain threshold. By using cameras and computer vision techniques, it removes the need for specially designed wearable sensors. However, this method may have limitations in terms of its performance under varying lighting conditions or water turbidity [16]. It also suffers from the same issue as the previously described approach, where the predetermined submersion thresholds might not accurately represent the individuals’ capabilities for holding a breath. Additionally, it may struggle to accurately track multiple individuals in crowded pool environments, which could lead to false alarms or missed detections.

1.2.3 Drowning Detection Using Neural Networks

In [17], the authors propose two methods for drowning recognition that utilise deep neural networks (DNNs) and a custom dataset. The authors explore scenarios for incorporating pre-trained neural networks: fine-tuning the parameters of existing DNNs and detecting body keypoints with a Deep High-Resolution Network (HRNet) [18] to train a generic DNN.

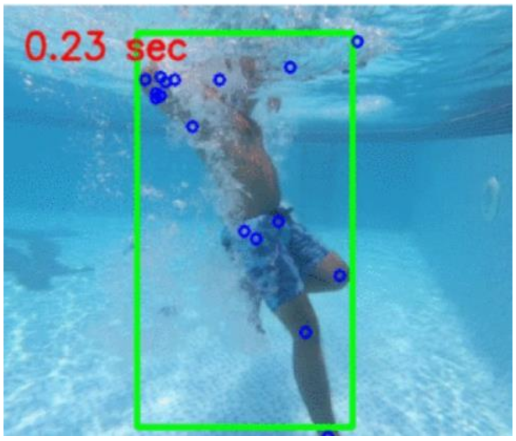


Figure 1.1: Pose Estimation using HRNet in underwater footage [17].

Method 1: The first method involves performing standard video scene classification on the proposed dataset. The approach trains a DNN using transfer learning to predictively label video frames and classify different human activities in the water. The authors test three DNN architectures: ResNet50 [19], VGG16 [20], and MobileNet [21], all pre-trained on the ImageNet dataset. By fine-tuning the parameters of the neural networks, the authors adapt the generic filters trained on ImageNet to the drowning recognition problem.

Method 2: The second method utilises the HRNet to detect and compute body pose keypoints (fig 1.1). The model has been pre-trained on the COCO train2017 dataset [22], which contains over 50,000 images and 150,000 person instances labelled with 17 keypoints. The detected keypoints are then classified using a DNN of 5 layers with (50,50,50) neurons in the hidden layers to classify the three different human water activities (swim, drown, idle). The authors did not specify the exact architecture of this network

Described methods achieved 96.85% and 98.4% accuracy respectively. While the first approach offers valuable insights into using DNNs for drowning recognition, it may have limitations regarding generalisation to various environments and camera perspectives. However, using the second approach, the classification by DNNs occurs solely on keypoints, reducing the impact of the environment on the inference.

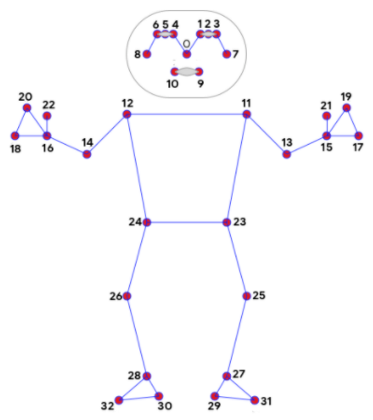
1.3 Pose Estimation Approaches

Pose estimation, a computer vision technique, tracks spatial positions and orientations of key body features within an image or video frame [23]. Its primary goal is to capture the relative positions of keypoints such as shoulders, elbows, wrists, hips, knees, and ankles to understand and represent human structure and movement. Pose estimation is crucial in various applications, including human-computer interaction [24], augmented reality [25], motion analysis [26], sports analytics [27], and safety systems [17]. It enables the analysis and classification of human activities, allowing the development of intelligent systems that effectively monitor and alert users about specific situations. One example of such a system could be the drowning detection system. By identifying and tracking keypoints, the proposed method could determine swimmers' positions and motions, differentiating between normal activities and potential drowning incidents. Analysing body poses would allow the system to recognise unusual or erratic movements, indicating distress [28].

1.3.1 MediaPipe Pose Estimation

MediaPipe is an open-source, cross-platform framework developed by Google that offers a suite of machine-learning models for solving a variety of real-time computer vision tasks, including human pose estimation [29]. Its pose estimation module utilises a neural network-based approach called BlazePose to detect and track 33 body keypoints in 2D images and videos [30].

BlazePose operates in two stages: first, it employs a lightweight, single-shot detector (SSD) [31] to locate the body's bounding box within the image. This detector is trained on a large dataset created by Google, containing various poses and body types. The specifics of the dataset, such as the number of images, the distribution of poses, and the exact sources, are not publicly disclosed by Google. The second stage involves a more complex neural network estimating the 33 keypoints' positions (fig 1.1) within the bounding box. This approach focuses computational resources on the region of interest, enhancing keypoint detection efficiency and accuracy.



0. nose	17. left_pinky
1. left_eye_inner	18. right_pinky
2. left_eye	19. left_index
3. left_eye_outer	20. right_index
4. right_eye_inner	21. left_thumb
5. right_eye	22. right_thumb
6. right_eye_outer	23. left_hip
7. left_ear	24. right_hip
8. right_ear	25. left_knee
9. mouth_left	26. right_knee
10. mouth_right	27. left_ankle
11. left_shoulder	28. right_ankle
12. right_shoulder	29. left_heel
13. left_elbow	30. right_heel
14. right_elbow	31. left_foot_index
15. left_wrist	32. right_foot_index
16. right_wrist	

Figure 1.1: Pose Estimation Landmarks used by MediaPipe [32]

Advantages include real-time performance on various devices, robust keypoint detection from diverse datasets, and easy integration due to MediaPipe's open-source nature. However, disadvantages involve being limited to 2D keypoints, which may not accurately analyse complex swimming movements, and sensitivity to occlusions, potentially reducing drowning detection performance.

1.3.2 HRNet Pose Estimation

The High-Resolution Network (HRNet) is a state-of-the-art human pose estimation approach widely used in various computer vision tasks [18]. Unlike traditional methods that predict keypoints in a low-resolution feature map before upsampling them, HRNet maintains high-resolution representations throughout the network using a multi-resolution parallel framework. This design captures both high-level semantic information and low-level fine-grained details, leading to more accurate keypoint detection.

HRNet extracts features from input images at multiple scales and fuses them through high-to-low and low-to-high resolution connections (fig 1.2). This multi-scale feature fusion allows the network to effectively utilise information from different resolutions, improving pose estimation performance. The output is a set of 2D keypoint coordinates for each body part in the input image. HRNet is commonly trained on the MS COCO dataset [33], producing 17 keypoint outputs (shown in figure 1.1).

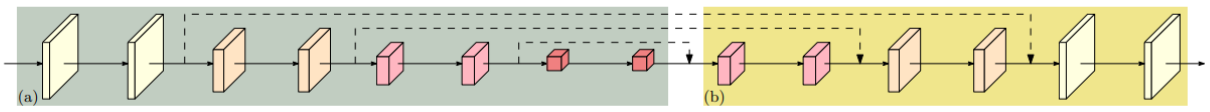


Figure 1.2: High-to-low (a) and low-to-high resolution connections in HRNet [18]

Advantages include high accuracy, demonstrated in various benchmarks and competitions [34], [35], fine-grained detail capture, and versatility, as HRNet's architecture can be adapted for different tasks. Disadvantages involve computational complexity, potentially limiting real-time performance on resource-constrained devices, and sensitivity to occlusions, which could impact drowning detection performance.

1.3.3 YOLOv7 Pose Estimation

YOLOv7 is a recent advancement in the popular "You Only Look Once" (YOLO) family of object detection algorithms [36]. It introduces improvements such as an enhanced backbone network, better activation functions, and improved data augmentation techniques. These enhancements result in a more accurate and efficient pose estimation approach, making it promising for drowning detection applications.

YOLOv7-pose is a pose estimation implementation of YOLOv7 [37]. It divides an input image into a grid, predicting bounding boxes and class probabilities for each grid cell. It uses a single convolutional neural network (CNN) for object detection and keypoint localisation simultaneously, requiring only a single pass through the network for predictions. Similarly to HRNet, YOLOv7 is trained on the MS COCO dataset [33], which results in a robust algorithm predicting body poses consisting of 17 keypoints as shown in Fig 1.3.

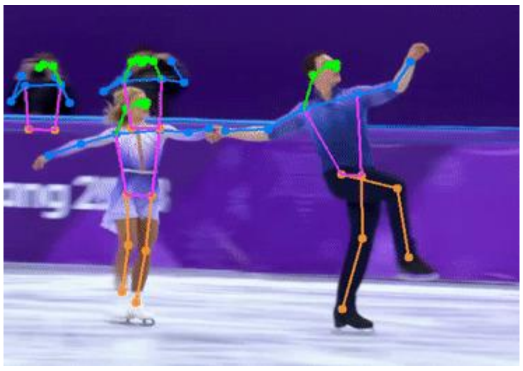


Figure 1.3: An example of YOLOv7 pose estimation inference on a video frame [38]

Advantages include real-time performance, improved accuracy compared to earlier versions of YOLO algorithms (fig 1.4) and MediaPipe [39], and scalability, as YOLOv7 offers multiple variants with different accuracy and computational complexity trade-offs. It also is able to do multi-person pose estimation, which is essential in drowning detection systems for swimming pools. Disadvantages involve less precise keypoint localisation compared to HRNet and sensitivity to occlusions, similar to other pose estimation approaches.

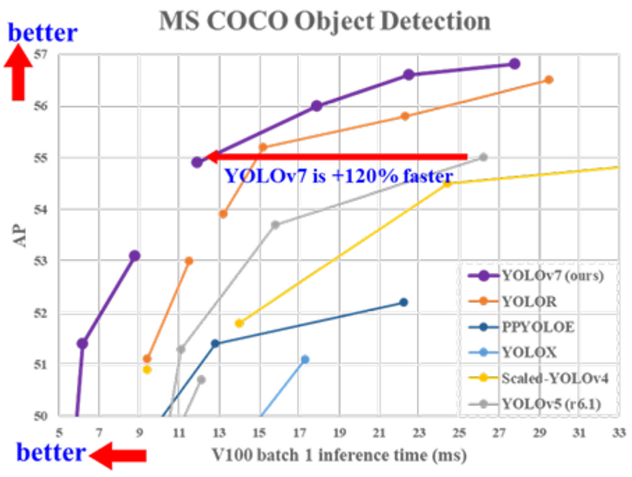


Figure 1.4: Comparison of YOLOv7 with other object detectors [36].

1.4 Neural Network-Based Classification

Classification in machine learning involves predicting the category or class label of an input based on its features. As a type of supervised learning [40], models are trained on datasets containing input-output pairs, with the output being the associated class label [41]. The approach is applied in various practical applications such as spam detection [42], medical diagnosis [43], and image recognition [44].

Artificial Neural Networks (ANNs) were first discussed in 1944 by researchers at the University of Chicago [45]. They were supposed to mimic the electrochemical properties of neurons in our brains to learn complex patterns and relationships between input features and class labels [46]. ANNs are built with interconnected layers of nodes that process and transmit information through connections with assigned weights [47]. During training, these weights are adjusted to minimise the error between the predicted and true class labels in the dataset [48].

Neural network-based classifiers are increasingly popular because of their ability to model non-linear relationships and exceptional performance in various domains [49]. Deep learning, as a subfield of machine learning, focuses on deep neural networks (DNNs). DNNs consist of multiple hidden layers allowing them to extract increasingly complex and abstract features from the input data [50]. Deep architectures have been important in achieving state-of-the-art performance in many classification tasks, including image and video classification, which are highly relevant to the proposed drowning detection system [19].

The deep neural networks which have shown significant success in activity recognition are Recurrent Neural Networks (RNNs) capable of processing temporal data [51], Convolutional Neural Networks (CNNs) specialised in spatial reasoning [52], and combinations of both [53].

1.4.1 Convolutional Neural Networks (CNNs)

CNNs are specialised deep learning architecture for processing grid-like data, like images and videos [54]. CNNs excel in various computer vision tasks, such as image classification, object detection [55], and semantic segmentation [56]. The main component of a CNN is the convolutional layer, which applies filters or kernels to input data, capturing local patterns and features. These filters are learned during training, allowing the network to discover relevant features for a given task automatically [57].

CNN architectures stack multiple convolutional layers [58], capturing more complex and abstract features. In the convolution layer, a kernel is slid over the input data to calculate the dot product between the kernel's weights and the underlying data at each position. Three types of convolutional layers are used in real-life applications, namely 1D, 2D, and 3D convolutional layers [59]. The main difference lies in the dimensionality of the accepted input and filters. Most widely known are 2D convolutional layers allowing for the processing of images [60]. The less computationally expensive than 2D layers, due to working with a vector-like input, are 1D layers with applications in text-to-speech translations [61]. 3D convolutional layers can be employed to learn spatiotemporal features in videos [62] To manage spatial dimensions and computational complexity, pooling layers are often introduced between convolutional layers [63]. These layers perform downsampling operations, like max or average pooling, to aggregate information and retain essential features [64]. Near the network's end, one or more fully connected layers are used for high-level reasoning and final classification [65]. An example of CNN with 2D convolutional layers is shown in Figure 1.5.

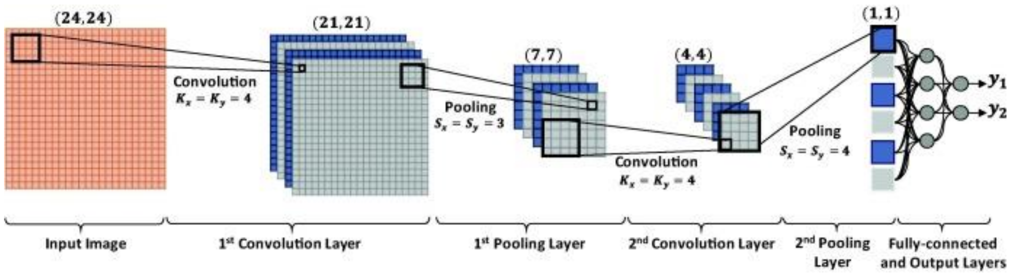


Figure 1.5: An example of CNN with 2 2D convolutional layers and 1 fully-connected layer [61]

1.4.2 Recurrent Neural Networks (RNNs)

RNNs are designed for handling sequential data, making them ideal for tasks involving time series, text, and other data with temporal dependencies [66]. RNNs maintain an internal state or memory through feedback connections, allowing them to capture and exploit time-varying patterns [67].

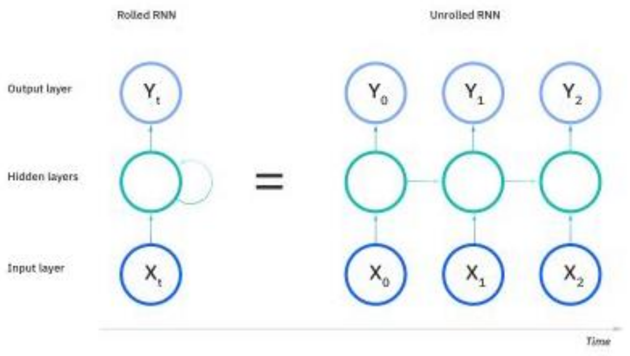


Figure 1.6: An example of a one-to-one RNN [68]

The hidden state, a crucial component of RNNs, is updated at each time step based on the current input and previous hidden state. This enables RNNs to retain information from previous time steps and use it for later predictions (figure 1.6). The RNNs can be divided into four main categories with regard to the number of inputs and outputs: one-to-one, one-to-many, many-to-one and many-to-many [69]. However, training RNNs can be challenging due to issues like vanishing and exploding gradients [70]. Advanced architectures like Long Short-Term Memory (LSTM) [71] networks and Gated Recurrent Units (GRUs) [72] address these challenges with gating mechanisms for more effective learning and retention of long-range dependencies.

1.4.3 Combination of CNNs and RNNs

A good way of achieving more powerful classification models is by combining convolutional layers and recurrent layers in the model architecture [73]. This approach has found significant success in the quantification of DNA [74], sequential reasoning [75] and object detection [76]. CNN-RNN neural network architecture, which is a combination of CNNs and RNNs, is a fusion of spatial and temporal capabilities of both types of networks. In this ensemble of neural network layers, the CNN part first extracts the spatial features from the input data. The extracted feature vectors are then fed through the RNN part of the network, which models the temporal dependencies and relationships between them over time. This allows the CNN-RNN model to exploit the strengths of both types of networks to classify sequences of input into categories. These sequences could represent consecutive frames from video footage representing some kind of action presented by a human individual, like the water activities in the proposed drowning detection system.

1.5 Datasets

An important aspect of a neural network-based classification detection system is a large, good-quality dataset [77]. Often times dataset collection can be expensive and time-consuming, especially when desired data consists of scarcely occurring events [78]. Drowning incidents are rarely happening in front of the camera, hence collecting a diverse and well-balanced dataset, which is essential for the training and evaluation of neural network models, can pose a challenge.

1.5.1 Lifeguard.IO Custom Drowning Dataset

Lifeguard.IO is a drowning detection system built by researchers for the purpose of the HackHarvard 2017 hackathon [79]. This project [80] recognised the lack of publicly available datasets containing drowning events and took the initiative to create the dataset to train a machine-learning model for detecting drowning incidents in swimming pool footage [81]. The researchers first gathered all available videos on the internet featuring drowning incidents to build their custom dataset. They isolated the drowning actions from these videos, resulting in a collection of labelled drowning events. Additionally, they extracted non-drowning actions, such as swimming or idle behaviour, from the same videos. This approach ensured that drowning and non-drowning actions were captured under similar environmental conditions, providing a more realistic training set for the model.

1.5.2 Water Behavior Dataset for an Image-Based Drowning Solution

The “Water Behavior” dataset was collected under the Electrical Engineering department at Rochester Institute of Technology in Dubai [17]. It is specifically designed for training and testing models to recognise drowning incidents in swimming pool environments. This dataset consists of short videos depicting various human activities in water, captured using two different cameras for above-water and underwater scenes.

The researchers engaged an experienced team of lifeguards to record simulated drowning events in a controlled swimming pool environment. Each video in the dataset supposedly shows only one person performing one of three different activities: swimming, drowning, or staying idle. The dataset includes a total of 91 videos, averaging 57 minutes each. The videos are split into 47 overhead and 44 underwater scenes, with subcategories for each activity.

1.6 Background research summary

By considering the three mentioned approaches to the development of a drowning detection system, a method utilised in [17] seems to be most feasible in terms of costs and complexity. It does not require all swimming pool goers to wear potentially uncomfortable and expensive sensors, as in [10]. Additionally, much less complex hardware architecture is needed than in [13]. Advanced computer vision techniques such as pose estimation and neural network classification can lead to a robust system invariant to environmental changes.

For the pose estimation part of the drowning detection system, YOLOv7-pose seems like the best choice. YOLOv7 is a compromise between the real-time performance of Google's MediaPipe and the accuracy of HRNet [39]. Thanks to its single-pass detection mechanism described in [36], YOLOv7 should be able to process the input data with reasonable execution time while maintaining good accuracy in its predictions. Furthermore, YOLOv7 offers scalability with multi-person pose estimation [37], which is essential if the proposed system were to be implemented in a real-life environment.

Regarding the classification section of the system, three alternative deep neural network models will be designed to settle which one delivers superior outcomes when evaluated on the test data. Starting with a straightforward CNN estimating predictions from keypoints generated based on a single frame and finishing with both CNN and CNN-RNN that produce predictions considering features derived from sequences of frames. Introduced models ought to undergo a thorough evaluation and comparison with a baseline model. Having done that, the most effective architectural approach for the drowning detection system will be identified. There is potential in the ensemble of convolutional and recurrent layers for leading to a more efficient and dependable solution addressing drowning, as it already showed promising developments in human activity recognition [53].

The “Water Behaviour” dataset proves to be a good choice when considering its diverse range of swimming activities and drowning incidents. This aspect delivers an excellent basis for training and evaluating the proposed methods. Looking into the positive features it holds, there is the appearance of only one person per video in the dataset. As the focus is directed on core techniques and simplicity, this project considers single-person pose estimation and classification. This choice enables concentrating on developing a robust and accurate system for individual drowning detection, which can later be expanded to handle multiple individuals in a more complex scenario.

Chapter 2 - Methods

2.1 Development Tools

Python and JavaScript are high-level programming languages commonly used for machine-learning applications [82]. They allow for a level of abstraction with the cost of slower execution [83]. Due to Python's wide choice of open-source data manipulation and machine-learning libraries, and a personal preference, it has been chosen as a programming language for this project. Additionally, the code was developed using Microsoft's free, open-source code editor "VS Code".

2.1.1 Libraries

PyTorch is a popular open-source deep learning framework developed by Meta's AI research lab [84]. It provides libraries for developing a wide range of machine-learning applications. PyTorch and other deep learning frameworks, such as TensorFlow or Keras, offer well-written documentation and remarkable support from the community. Crucially, YOLOv7 is implemented using PyTorch. Thus, this will be the machine-learning library of choice for this project. NumPy [85] and OpenCV [86] libraries will be used for handling data, including tasks such as data processing and data augmentation. Additionally, OpenCV will be used for displaying the classification results in the final version of the drowning detection system. Scikit-learn [87] offers a range of functions to calculate evaluation metrics which will be used to evaluate neural network models. Matplotlib [88] and Seaborn [89] will be used to display results from the testing of the proposed system.

2.1.2 Version Control

For this project, Git will be used for version control. Git allows for reverting to old system versions in case of critical errors. It also can be used in the system console, speeding up the development process. The two most popular options for hosting the central repository are GitHub and GitLab. Due to higher experience with the former and a personal preference, GitHub was chosen.

2.1.3 Experimental Setup

It is generally recommended to train Neural Networks using a General Processing Unit (GPU) due to the large number of calculations in learning algorithms [90]. All models were trained on Nvidia GeForce GTX 1070 graphics card. Nvidia provides an interface for programming with GPUs called Compute Unified Device Architecture (CUDA) [91]. CUDA can be accessed directly from the PyTorch library, which allows the user to move more computationally expensive calculations onto the graphics card.

2.2 Dataset Description

As mentioned in section 1.5.2, the “Water Behavior” dataset contains videos of different water activities simulated with help from a professional lifeguard team. Access to the data has been granted for this project after signing a release agreement. The dataset comprises four classes: swimming, drowning, idle, and miscellaneous. Due to the ambiguity of represented activities and the small number of videos, the last category was omitted from the classification task. All videos were recorded either under the water or overhead. The dataset is split into two sets: 90% intended for training and 10% intended for testing. Each video represents one type of activity, with its label in the file's title.

All videos in the dataset were captured at a rate of 30 fps. The overhead footage was recorded with a resolution of 1920x1080, while the underwater footage was recorded at a resolution of 1920x1440. Summing up, in the test set, there are 51 videos representing swimming (22,168 frames), 32 showing simulated drowning (13,430 frames), and 21 presenting idle behaviour (15,817 frames).

2.3 YOLOv7 Pose Estimation

YOLOv7 is a state-of-the-art object detection algorithm. Its yolov7-pose implementation of the pose estimation method [37] is available from the creator's GitHub repository [92]. In addition to pre-trained models, it also contains Python modules for handling and displaying data. The “yolov7-w6-pose” is a model pre-trained on the MS COCO dataset [33], [37], capable of predicting 17 keypoint locations based on input images [93]. At the time of writing this report, a new version of YOLO, namely YOLOv8, has been made available for public use by Ultralytics [94]. However, due to the known bugs and lack of documentation, YOLOv7 will continue to be used throughout this project.

2.3.1 Pre-processing the data

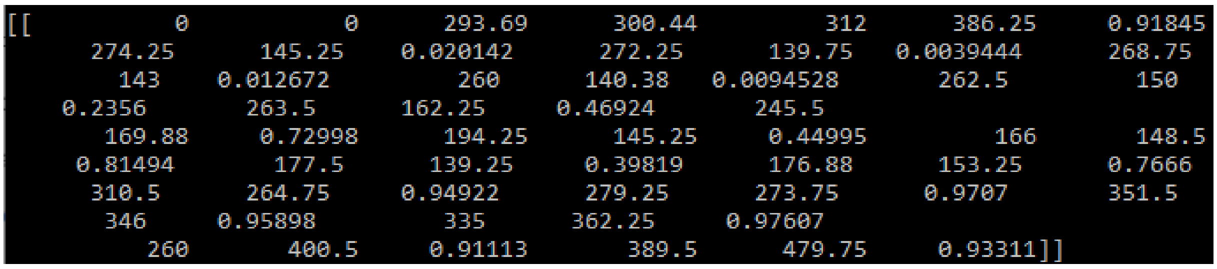
Before processing the dataset using YOLOv7 for body pose estimation, several adjustments need to be made to the video frames to optimise the performance and speed of the model. First, the frames need to be resized to 640x384 resolution in order to reduce the computational requirements and expedite the inference time. Next, the colour channel has to be converted using OpenCV’s ‘COLOR_BGR2RGB’ filter. The original video frames were recorded in the BGR colour space, however, YOLOv7 performs better with images in RGB colour space. Lastly, the images have to be converted to PyTorch tensors, as the yolov7-w6-pose model only accepts input in this form.

2.3.2 Extracting Keypoints

To extract keypoints from frames of the videos in the dataset, the ‘yolov7-w6-pose’ must first be loaded onto the GPU using the ‘device’ and ‘load’ functions from Pytorch’s ‘torch’ library. The model is then set to evaluation mode and optimised for faster inference time by converting it to a half-precision floating-point format [95]. Each frame from the input footage is sequentially passed for inference with the model.

A non-maximum suppression filter (NMS) has to be applied to the output of inference of each frame to retain only the most relevant keypoints. NMS uses the confidence and the Intersection over Union (IoU) thresholds to merge all possible predictions for a single object into one accurate prediction [96]. Furthermore, the output has to be passed through the ‘output_to_keypoint’ function from YOLOv7’s ‘utils.plots’ library to convert the model output to the desired format.

As mentioned in section 1.6, this project focuses on a single-person action classification. Thus, in cases where more than one set of keypoints is detected in a frame due to multiple individuals present or an inaccurate detection, only the pose of the person with the highest confidence score is retained. If only one person is detected, all keypoints are kept. When no person is detected, the frame is skipped.



```
[[          0           0      293.69      300.44         312      386.25     0.91845
      274.25      145.25    0.020142      272.25      139.75   0.0039444      268.75
         143    0.012672         260      140.38   0.0094528       262.5         150
     0.2356       263.5      162.25     0.46924       245.5
      169.88     0.72998      194.25      145.25     0.44995         166       148.5
     0.81494       177.5      139.25     0.39819      176.88      153.25      0.7666
      310.5      264.75     0.94922      279.25      273.75      0.9707       351.5
        346     0.95898         335      362.25     0.97607
         260       400.5     0.91113       389.5      479.75     0.93311]]
```

Figure 2.1: An example output from YOLOv7’s pose estimation inference on a frame.

2.3.3 Inference Output

The output from the inference is a one-dimensional tensor consisting of 58 values (figure 2.1). The first 7 values contain the following information: batch number, class label of the detected person, x-coordinate of the centre of the bounding box, y-coordinate of the centre of the bounding box, the width of the bounding box, the height of the bounding box and confidence score of the detection. The remaining 51 values are related to the estimated body pose. This includes the x and y coordinates and confidence values for each of the 17 key features detected. For the classification task, only values representing keypoints are needed. However, the first 7 values are preserved, as they are useful for drawing the bounding boxes around subjects during real-time inference on videos.

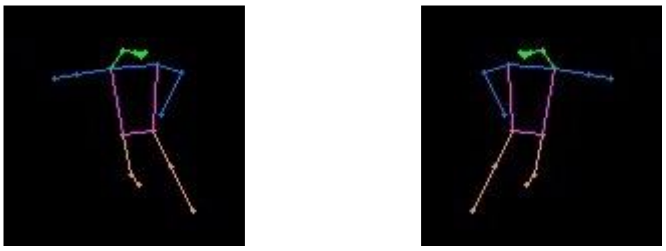
2.4 Data Preparation

Data generated with YOLOv7 inference needs to be further processed before being used in classification tasks or training of neural network models. As mentioned in the section above, the first 7 seven values in the one-dimensional tensor are not needed to describe the skeleton of a detected pose. Hence, they are split from the rest of the values representing keypoints. The remaining tensor of length 51 is then reshaped to the shape of a 17x3 2-dimensional matrix.

Normalisation: To ensure that the dimensions are consistent, the x and y values of each keypoint had to be normalised. Normalisation is crucial to obtain good classification results and improve the training time [97]. This process reduces the impact of variations in the size and position of individuals in the footage. To achieve this, the keypoints are centred within the 640x384 frame by calculating the displacement vector with regard to the centre of the detected pose and adjusting all detected features in the opposite direction.

2.4.1 Preparing Data for Neural Network Training

Data augmentation: A process of artificially increasing the training set by creating modified copies of existing data is called data augmentation [98]. Its purpose is to improve the model's generalisation capabilities. In the context of this project, the size of a training set is increased by creating horizontally mirrored copies of the normalised keypoints. This is achieved by first creating copies of all tensors describing detected poses. In copied tensors, values representing the x coordinate of each keypoint are changed to $w - x$ where w is the width of the frame. This technique effectively doubles the number of data points available for training the neural network. Figure 2.2 shows a visual representation of this technique of data augmentation. The dataset size could be quadrupled by additionally mirroring the keypoints vertically. However, due to the nature of the classification task, modifying the data in such a way may negatively influence the models' accuracy.



(a) The original keypoints. (b) Augmented keypoints.

Figure 2.2: A visual representation of data augmentation in the training set.

Saving pre-processed data: Finally, the normalised and augmented keypoints are saved as tensors with a '.pt' file extension. The labels extracted from the file names are saved in a separate file. Stored data can be easily accessed using PyTorch's functions in the 'torch' module. In addition, storing the keypoints as tensors eliminates the need to execute the YOLOv7 inference each time the model is trained. This leads to significantly reduced overall training time.

2.5 Neural Network Models

PyTorch framework allows for easy implementation of custom neural network models by subclassing a 'nn.Module' class from the 'torch' library. A class created this way needs to implement the following two methods:

1. __init__: Method initialising parameters of the neural network and containing all its architecture layers.
2. forward: Method describing data flow through the neural network.

The architecture of the neural network is described with layers. Some layers are used for classification and hold weights inferred from the training process. Other layers in the architecture are used for data transformation between the hidden layers.

2.5.1 Neural Network Components

Fully connected layers: Fully connected layers, also known as linear layers, transform the input data by applying weights and optional biases producing a linear output [99]. The mathematical operation performed in each fully connected layer can be represented as: $y = Wx + b$, where: x is the input vector, W is the weight matrix, b is the bias vector and y is the output vector. In this project, the linear layers will be used to produce the output from data processed by previous convolutional or recurrent layers. Additionally, the baseline model will consist only of this type of layers.

Convolutional layers: Convolutional layers are a key component of CNNs, allowing them to capture spatial information from the input. As was already mentioned in Chapter 1, convolutional layers can be divided into 3 categories: Conv1D, Conv2D and Conv3D. The differences lay in the dimensionality of kernels and input data. Since the input data is 17x3 2-dimensional tensors, 2D convolutional layers are the best choice. The 3D layers could be utilised in models that process inputs in the form of sequences of frames, treating the sequence length as a third dimension to try and capture the spatial as well as temporal relations [100]. However, using 3-dimensional layers is more computationally expensive, hence the final decision to utilise Conv2D layers in this project.

For the purpose of this project, all convolutional layers will employ 3x3 kernels. This is due to the shape of the input data (17x3) allowing for a maximum 3x3 size of a square kernel. Additionally, 1-by-1 padding will have to be applied to the input data at each subsequent convolutional layer to prevent further shrinking of the tensor. Padding works by prepending and appending a specified amount of rows and columns containing 0 values to the input data. After sliding the 3x3 kernel over padded data, its shape returns to its original form. More information about kernels and padding in convolutional layers is in [101], [102].

Recurrent layer: As mentioned in Chapter 1, recurrent layers allow models to capture temporal information in the input data. However, in the backpropagation part of training, the recurrent layers can cause gradients to become too big or too small. Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) are types of RNN architecture commonly used to address this issue [103]. Both LSTM and GRU approaches utilise mechanisms called “gates” to control the flow of information and gradients through the network. This allows the network to adaptively retain or discard information from previous time steps, which mitigates the vanishing gradient issue. In addition, gradient clipping addresses the problem of exploding gradients by limiting the maximum gradient value [104]. More on LSTM and GRU can be found in (ref).

Since the classification task requires the models to be trained on a relatively small dataset, GRU seems to be the best choice. Its fewer parameters make it less prone to overfitting and less computationally expensive [105]. LSTM perform better on tasks involving long-term temporal dependencies and complex models. However, the task of water activity recognition is inherently short-term and requires low execution times. Hence, GRU was chosen as the recurrent layer in this project.

Non-linear activation layers: Non-linear activation layers, or activation functions, add nonlinearity to the model, allowing networks to capture complex patterns and relationships in the data [106]. Commonly used activation layers include Rectified Linear Unit (ReLU), Hyperbolic Tangent (tanh), and Sigmoid functions. While ReLU sets negative values to zero, Sigmoid maps input values to a range between 0 and 1, and tanh adjusts them to a range between -1 and 1. However, Sigmoid and tanh are prone to vanishing gradient issues. Thus, for the proposed models, ReLU is preferred due to its computational efficiency and ability to address the vanishing gradient problem.

2.5.2 Neural Network Architectures

Three CNN-based models are created with the purpose of classification of the prepared data. Each successive model architecture presented in this section modifies the previous one. The aim is to see how the addition of individual elements affects the performance of these networks Each proposed model employs 3 convolutional layers to avoid excessive complexity, which comes with many hidden layers [107]. Additionally, a generic 50x50x50 baseline model is created to evaluate the performance of the proposed models as proposed in [17].

2.5.2.1 Baseline model

The baseline model (Figure 2.3) is a simple feedforward neural network. Its architecture consists of an input layer with 51 neurons, three hidden layers with 50 neurons each, and an output layer with three neurons corresponding to three classes. Additionally, after each layer, the data is passed through the activation function to introduce non-linearity to the model. This neural network accepts data in the form of tensors representing a person's position generated from just one video frame.

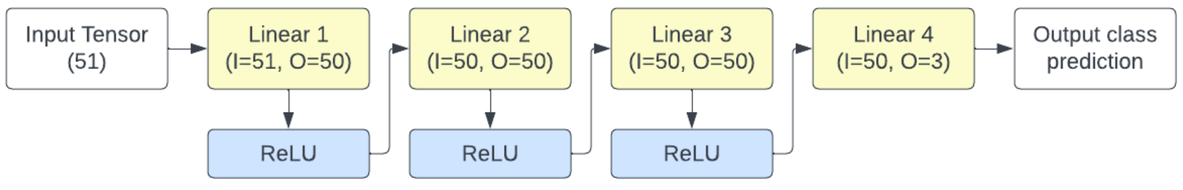


Figure 2.3: Architecture diagram of the Baseline model (I-Input, O-Output).

2.5.2.2 CNN model on single frames (CNN-SINGLE)

The CNN-SINGLE (figure 2.4) is a simple CNN-based neural network. It consists of three convolutional layers, followed by two fully connected layers to produce output. This model predicts the water activity classes based on single frame input. Unlike the baseline model, CNN-SINGLE employs convolutional layers to exploit spatial information in the input data. The assumption is that the model is able to determine if a person is drowning by just looking at their body pose described by keypoints. Output from the convolutional layers is passed through two fully connected layers to map the spatial features into 3 classes. Additionally, activation functions are applied to the output of each hidden layer to introduce non-linearity to the model.

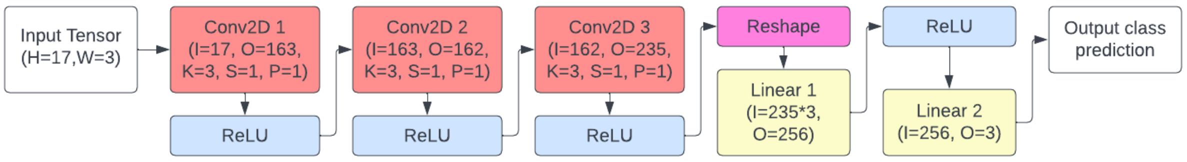

Input Tensor (H=17,W=3)
Conv2D 1 (I=17, O=163, K=3, S=1, P=1)
ReLU
Conv2D 2 (I=163, O=162, K=3, S=1, P=1)
ReLU
Conv2D 3 (I=162, O=235, K=3, S=1, P=1)
ReLU
Reshape
Linear 1 (I=235*3, O=256)
ReLU
Linear 2 (I=256, O=3)
Output class prediction

Figure 2.4: Architecture diagram of the CNN-SINGLE model (H-Height, W-Width, K-Kernel, S-Stride, P-Padding).

2.5.2.3 CNN model on sequences of frames (CNN-SEQ)

The CNN-SEQ model shown in Figure 2.5 is a modification of the previous CNN-SINGLE model. It also features three convolutional layers and two fully connected layers with ReLU activation functions between the hidden layers. However, the architecture has been adapted to accept data in sequence form. The model processes the current and also previous frames from the video at the same time to see if this helps to improve performance. The output of this model is a list of predictions for all frames in the sequence. The result of the classification is the most frequent class in this list. This model aims to see whether prediction based on multiple previous frames achieves better accuracy than classification based solely on the estimated pose.

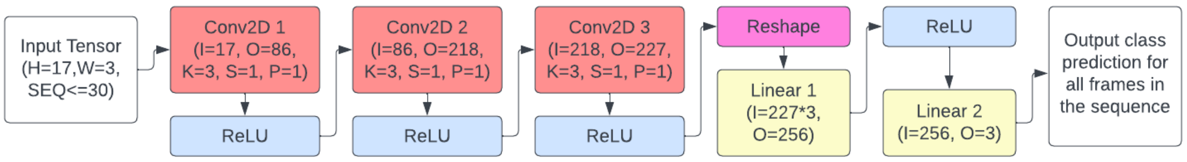

Input Tensor (H=17,W=3, SEQ<=30)
Conv2D 1 (I=17, O=86, K=3, S=1, P=1)
ReLU
Conv2D 2 (I=86, O=218, K=3, S=1, P=1)
ReLU
Conv2D 3 (I=218, O=227, K=3, S=1, P=1)
ReLU
Reshape
Linear 1 (I=227*3, O=256)
ReLU
Linear 2 (I=256, O=3)
Output class prediction for all frames in the sequence

Figure 2.5: Architecture diagram of the CNN-SEQ model (SEQ-Length of the input sequence).

2.5.2.4 CNN-RNN model

The CNN-RNN model in Figure 2.6 is a further modification of the CNN-SEQ model. It incorporates a Gated Recurrent Unit (GRU) layer to better capture temporal dependencies in sequential data, such as keypoint sequences generated from video frames. The architecture now includes the original convolutional layers, the GRU layer, the fully connected layers and the activation layers between them. In this model, each sequence element is first processed through the CNN part of the network exploiting its spatial relations. Then the resulting feature maps are flattened and fed into the GRU layer to find existing temporal relations. The sequence elements are passed through the network incrementally, updating the recurrent layer at each step with its previous hidden state (dotted arrow). Finally, after processing the entire sequence, the output of the last GRU layer is passed through fully connected layers to map the feature vectors into 3 classes.

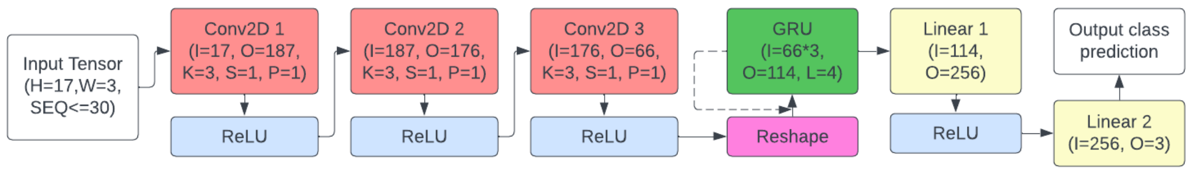

Input Tensor (H=17,W=3, SEQ<=30)
Conv2D 1 (I=17, O=187, K=3, S=1, P=1)
ReLU
Conv2D 2 (I=187, O=176, K=3, S=1, P=1)
ReLU
Conv2D 3 (I=176, O=66, K=3, S=1, P=1)
ReLU
Reshape
GRU (I=66*3, O=114, L=4)
Linear 1 (I=114, O=256)
ReLU
Linear 2 (I=256, O=3)
Output class prediction

Figure 2.6: Architecture diagram of the CNN-RNN model (L-Number of recurrent layers).

2.5.3 Hyperparameter Optimisation

Model hyperparameters were found in the process of hyperparameter optimisation. These are distinct from regular model parameters, as they are not learned during training and must be set externally. Two popular libraries for automatic hyperparameter optimisation in PyTorch are Optuna [108] and Ray Tune [109]. The former is a user-friendly, adaptable library offering a range of optimisation techniques. The latter is a distributed library for hyperparameter tuning with advanced features such as parallel execution and fault tolerance. Due to its user-friendliness and lightweight, Optuna has been chosen to tune the model configurations for this project. Hyperparameters needing optimisation in the three proposed neural networks are the learning rate, batch size, number of epochs, the number of neurons per layer, sequence length, number of recurrent layer channels, and the number of hidden recurrent layers.

Model	Learning rate	Batch	epoch	Conv1 channels	Conv2 channels	Conv3 channels	Sequence length	GRU channels	GRU layers
CNN-SINGLE	0.00024	30	43	163	162	235	—	—	—
CNN-SEQ	3.56e-05	16	45	86	218	227	30	—	—
CNN-RNN	9.14e-05	16	64	187	176	66	30	114	4

Table 2.1: Hyperparameter optimisation results for each model

Techniques featured in the Optuna library include Tree-structured Parzen Estimator (TPE), Gaussian Process, CMA-ES, and simpler techniques such as grid and random search. The TPE optimisation technique was chosen for this project because of its ability to prune weak trials, significantly reducing the optimisation execution times [110]. Moreover, TPE is a Bayesian optimisation technique which builds a probabilistic model based on previous trials. This allows it to converge to good hyperparameter settings faster than a grid or random search. The hyperparameter optimisation process is described in Appendix D. The final hyperparameters for each model are shown in Table 2.1.

2.5.4 Model Training

The proposed models need to be trained on the prepared training data in order to adjust their layers' weights. The training procedure in PyTorch is as follows:

1. The model instance is created on GPU
2. The KeypointDataset (subclass of PyTorch's 'Dataset') is created with a path to the saved keypoint and label files.
3. A 'DataLoader' class instance is created, taking the dataset instance and batch size as inputs.
4. Two nested loops are created. The outer loop stands for the number of epochs the model is meant to be trained on to improvise its predictive performance. The inner loop iterates over the data-label pairs in the DataLoader instance.
    a. With each passing of the inner loop, the data and labels from DataLoader are sent to the GPU for faster calculations. In addition, if required by the model, the input data is also reshaped.
    b. Optimiser gradients are cleared to ensure that the parameters are being updated based on the current iteration only.
    c. The data is passed through the model to obtain the output logits.
    d. A loss function calculates the loss based on the output's logits and actual labels.
    e. A backward pass is performed to calculate the loss gradients with respect to model parameters
    f. Model parameters are updated using the optimiser.
    g. To indicate training progress, the running loss is printed in the console every 100 iterations or every epoch.
5. Finally, the trained model is saved for future testing.

Loss functions evaluate the difference between the predictions generated by the model and the actual target values or labels [111]. These functions help steer the optimisation process by offering a metric to minimise throughout the training phase. CrossEntropyLoss and MSELoss are two commonly utilised loss functions in PyTorch [112]. CrossEntropyLoss is particularly well-suited for classification tasks as it measures the gap between predicted class probabilities and true class labels. This loss function combines the logarithm of the softmax function [113] with the negative log-likelihood, leading to a higher penalty for incorrect class predictions. Mean Squared Error Loss, computes the squared differences between the predicted and target values, averaging them across all instances. This loss function is typically employed for regression tasks.In CNN-based models, CrossEntropyLoss is the better choice, especially for classification tasks, since it effectively quantifies the difference between predicted class probabilities and actual class labels.

Optimisers are critical in refining the model's parameters by minimising the loss function, thereby enhancing the model's performance [114]. A comparison of different optimisers is presented in [115]. Stochastic Gradient Descent (SGD) and Adam are two popular optimisers in this regard. SGD refines the model parameters based on the loss function's gradient with respect to each parameter, using a fixed learning rate. However, this approach may lead to slow convergence or getting trapped in local minima. Adam is an adaptive optimisation technique that combines the benefits of other optimisers like AdaGrad and RMSProp. By adjusting the learning rate for each parameter individually, Adam achieves faster convergence and improved performance. For water activity classifiers, Adam might be the superior optimiser choice due to its adaptive learning rate and capability to effectively handle sparse gradients, potentially resulting in quicker convergence and better performance.

2.5.5 Model Testing

The three trained models and the baseline were tested on a test partition of the dataset that was not used during training. The evaluation metrics of each model were measured to assess their ability to make accurate predictions on previously unseen data. This helps identify potential issues such as overfitting, where the model learns to perform well on the training data but fails to generalise to new instances. The testing results will be discussed in Chapters 3 and 4.

2.6 Integration of the Drowning Detection System

The final Drowning Detection System promised in this report is created by merging the pose estimation algorithm, data preparation, and neural network model into a single program as shown in Figure 2.8. Three distinct system versions were developed using the classification models discussed in section 2.5, along with an additional baseline model for comparison and evaluation. This end-to-end pipeline combines pose estimation, data preparation and classification tasks in real-time inference.

During real-time testing, the video frames are processed sequentially, with the pose estimation algorithm extracting keypoint information from each frame. The extracted information is then normalised with the technique described in 2.4. The resulting data is then fed into one of the pre-trained neural networks, which predicts the water activity being shown in the video. The system then displays the frame with a bounding box with colour and text based on the prediction.

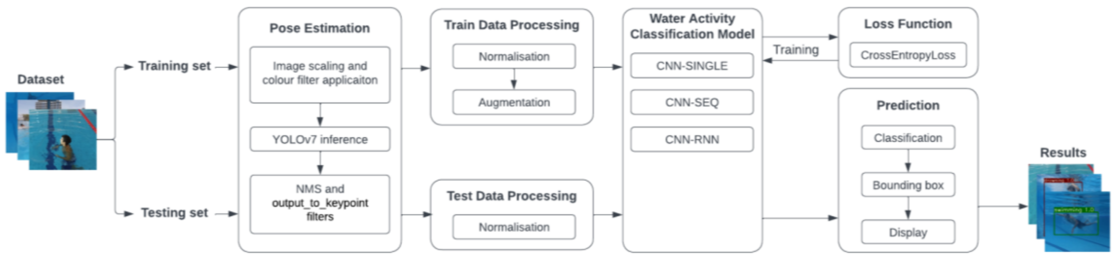


Figure 2.7: The working pipeline of the Drowning Detection System.

Chapter 3 - Results

3.1 Pose Estimation

As mentioned in Chapter 2, the body-pose keypoint extraction has been done with the pose estimation version of the YOLOv7 detector algorithm. Accurate keypoint detection was needed in order to provide quality data for neural network models to train on. For the purpose of training the classification models, the inference was run over all frames in the training set. Figure 3.1 shows a successful body-pose estimation of an individual performing “idle” water activity.

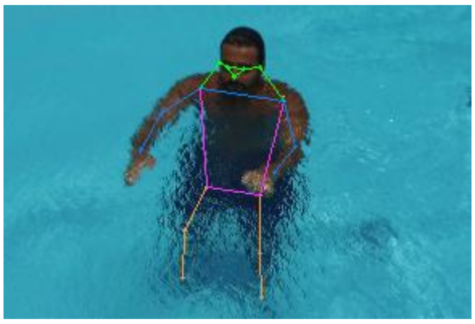


Figure 3.1: A result of pose estimation inference on a frame from the test set.

However, not all frames from the training partition of the dataset were used for body-pose extraction. Figure 3.2 shows the number of body poses extracted from frames in the dataset by the label. In total, 40,822 body-pose skeletons were extracted out of 51,415 frames in the dataset. This means that 10,593 frames did not include a person to extract the skeleton from or that the pose estimation algorithm failed to detect the keypoints.

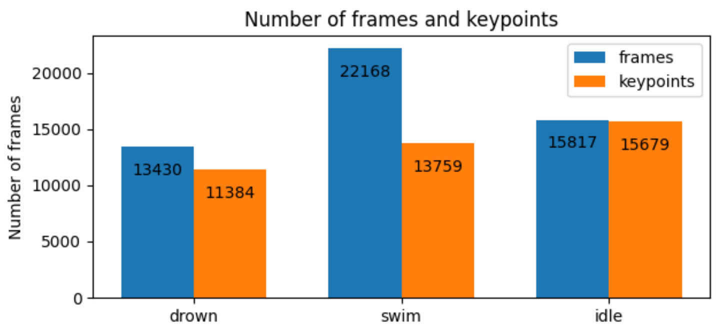


Figure 3.2: Bar chart showing the difference between available frames and keypoints extracted from each class.

The algorithm has extracted keypoints from over 99% of frames with an “idle” label and from 85% of frames labelled “drown”. A significantly lower percentage of extracted keypoints comes from the frames labelled “swim”, where the pose had been detected in only 62% of instances. A possible explanation includes the overheads in the videos presenting swimming action where the swimmer has not yet entered the camera's field of view or has just left it.

In addition to x and y coordinates, the YOLOv7 algorithm assigns confidence to each detected body part. Confidence in a keypoint means how certain the model is about its existence in the specified place of the frame. Figure 3.3 shows the percentage of detected features with confidence over 50% by feature class.

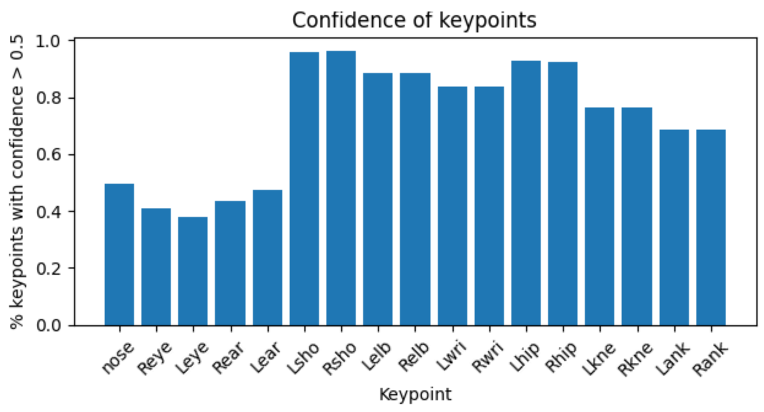
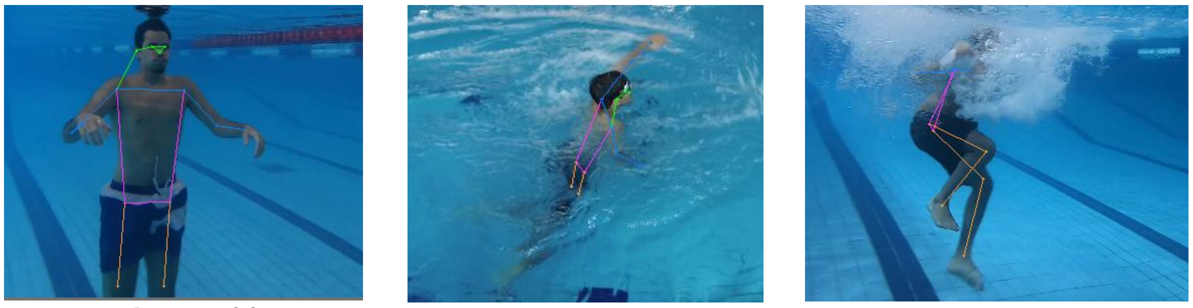


Figure 3.3: Histogram showing the percentage of keypoints with confidence higher than 0.5.

The algorithm had the least confidence in keypoints located on the head of the individuals in the video footage. This is mainly caused by the fact that multiple videos in the dataset show the water activities performed facing away from the camera, making it impossible to detect features on the face of the subject. Furthermore, it can be noticed that the closer to the centre of the body, the higher the confidence in a keypoint class. The cause of this might be the clipping of the subject limbs out of the frame of the video. This happens when the person performing a water activity is too close to the camera and cannot be entirely shown in the frame. Other explanations include the occlusions created by the water, especially in the overhead footage. Occlusions due to the water reflection and turbulence might be the reason why the confidence is lower for keypoints in the lower parts of the body than for the higher ones. Figure 3.4 shows examples of incomplete pose estimations on the data from the test set. It was also discovered that YOLOv7 struggled with correctly estimating the poses of upside-down individuals, causing a "jittering" effect where the body skeleton appeared to be moving, even though the subject was standing still.



(a) Limbs out of frame. (b) Water turbulence. (c) Water splashes.

Figure 3.4: Flawed pose estimation with its causes.

3.2 Classification Results

The proposed neural network classification models were tested on test data partition from the "Water Behavior" dataset. The Scikit-learn library was used to calculate the evaluation metrics to evaluate and compare models against each other and the baseline. The performance of each model is measured based on the classification results of the entire test video and the classification results of single frames.

3.2.1 Evaluation Metrics

Accuracy measures the proportion of correct predictions against the total number of predictions. It can be misleading in the case of an imbalanced dataset where the majority class dominates the results.

Precision measures the proportion of correct class predictions with respect to all instances predicted as belonging to that class made by a model. It highlights how well the model correctly identifies correct instances out of all instances it predicts as belonging to a given class.

Recall measures the proportion of correct class predictions among all actual instances of this class in the dataset. It quantifies the ability of a model to identify correct instances out of all correct instances. The recall is especially relevant in scenarios where the cost of false negatives is high, such as the risk of misidentifying the drowning incident in the drowning detection system.

The **F1 Score** is a harmonic mean of precision and recall. By considering both precision and recall, the F1 score provides a balanced assessment of a model's performance, emphasising the importance of correctly identifying positive instances without producing excessive false positives.

3.2.2 Classification results on frames

The results of classification based on single frames consider each frame from the input video. The classification class had to match the label to be successfully classified into a correct class. This way allows evaluation of the model performance on a lower time scale than with entire videos. Testing logs can be seen in Appendix C.3. Table 3.1 shows evaluation metrics for three proposed models and the baseline.

Model	Baseline	CNN-SINGLE	CNN-SEQ	CNN-RNN
Accuracy	0.69	0.70	0.74	0.80
Precision	0.66	0.67	0.70	0.75
Recall	0.71	0.71	0.75	0.76
F1 Score	0.67	0.67	0.71	0.76

Table 3.1: Evaluation Metric scores achieved by models.

The model which achieved the greatest scores in evaluation metrics is the CNN-RNN model. Following it is the CNN-SEQ model, achieving 6% lower accuracy and 5% lower precision and f1 scores. The recall score, however, is similar for both models, with only a 1% difference. The performance of the CNN-SINGLE model was only slightly better than the base model.

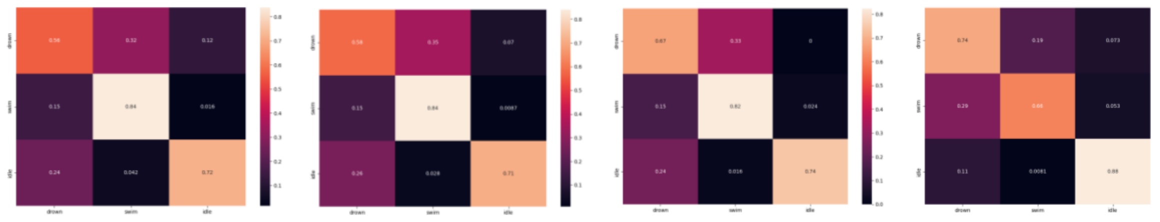
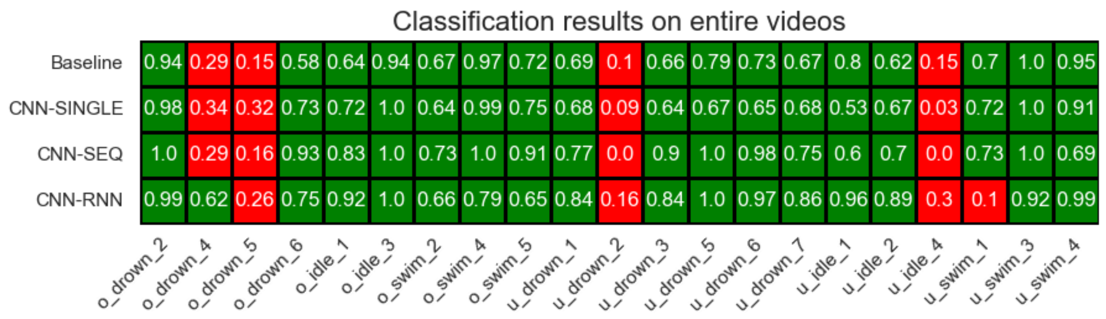

(a) Baseline. (b) CNN-SINGLE. (c) CNN-SEQ. (d) CNN-RNN.

Figure 3.5: Model Classification Confusion Matrices.

Confusion matrices are shown in Figure 3.5 (larger versions shown in Appendix C.1). These matrices had to be normalised due to the disproportionate number of frames belonging to each class in the test set. The models which classified the swim class with the highest accuracy are the CNN-SINGLE model, the CNN-SEQ model, and the baseline model, with respective accuracies of 88%, 88% and 86%. The only model which achieved an accuracy of over 70% in classifying the “drown” class is the CNN-RNN model. This model also achieved the highest accuracy in classifying the “idle” class.

3.2.3 Classification results on videos

Classification results of entire videos are considered in Figure 3.6. The class of the videos is defined as a class which appears most frequently in class predictions of its frames. The confusion matrices for each model are available in Appendix C.2.


Classification results on entire videos

	o_drown_2	o_drown_4	o_drown_5	o_drown_6	o_idle_1	o_idle_3	o_swim_2	o_swim_4	o_swim_5	u_drown_1	u_drown_2	u_drown_3	u_drown_5	u_drown_6	u_drown_7	u_idle_1	u_idle_2	u_idle_4	u_swim_1	u_swim_3	u_swim_4
Baseline	0.94	0.29	0.15	0.58	0.64	0.94	0.67	0.97	0.72	0.69	0.1	0.66	0.79	0.73	0.67	0.8	0.62	0.15	0.7	1.0	0.95
CNN-SINGLE	0.98	0.34	0.32	0.73	0.72	1.0	0.64	0.99	0.75	0.68	0.09	0.64	0.67	0.65	0.68	0.53	0.67	0.03	0.72	1.0	0.91
CNN-SEQ	1.0	0.29	0.16	0.93	0.83	1.0	0.73	1.0	0.91	0.77	0.0	0.9	1.0	0.98	0.75	0.6	0.7	0.0	0.73	1.0	0.69
CNN-RNN	0.99	0.62	0.26	0.75	0.92	1.0	0.66	0.79	0.65	0.84	0.16	0.84	1.0	0.97	0.86	0.96	0.89	0.3	0.1	0.92	0.99

Figure 3.6: Classification results on entire videos

In Figure 3.6, cell colours depend on whether the video class was predicted correctly, with green for correct predictions and red for wrong ones. Furthermore, the cell values show the percentage of correct predictions made by the model for frames of each video.

Overall, It can be noticed that most of the models struggled with correctly classifying the same four videos: “o_drown_4”, “o_drown_5”, “u_drown_2”, and “u_idle_4”. An exception to that is the CNN-

RNN model, which correctly classified the “o_drown_5” video, but instead failed with “u_swim_1”. The CNN-SEQ and CNN-RNN models were more confident in their predictions than the CNN-SINGLE and baseline models, which can be seen in the higher cell values for correctly classified videos.

Upon further inspection of wrongly classified videos with the “drown” label, it was noticed that in all 3 of them, the YOLOv7 algorithm struggled to accurately detect the body pose of the individuals simulating the water activity. Furthermore, the “o_drown_4” and “o_drown_5” videos included frames a the beginning and end of the footage where the subject wasn’t simulating the activity from the label. The individual in the “u_drown_2” video was upside-down, and as discussed earlier, YOLOv7 struggles with pose estimation in such cases leading to “jitter” in predictions. This jitter could have led to the video being classified by all models as “swim” due to its false motion. These observations might factor in the poor performance of all classification models.

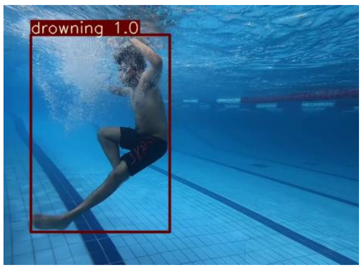
3.2.4 Evaluation of the Neural Network Models

Undoubtedly, the best-performing model based on evaluation metrics is the CNN-RNN model consisting of a combination of convolutional and recurrent layers. This model sacrifices the accuracy in predicting the “swimming” class to achieve better overall accuracy. In terms of precision, recall and f1 scores, this model significantly outperforms the baseline and most of the remaining models. The second best-performing model is CNN-SEQ. This model achieves high accuracy in predicting the “swim” class. Its recall score is close to the one of the CNN-RNN model.

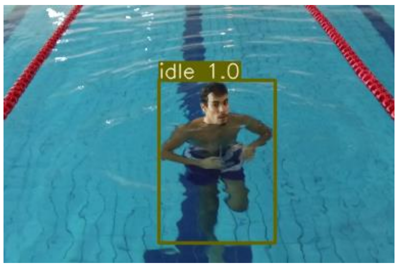
Overall, the models didn’t perform as well as expected. The accuracy results were nowhere near the ones achieved in [17]. One cause of this poor performance might be the not-so-big size of the training data. The models were trained on 81,644 frames from just 103 videos. It was also discovered that some of the videos might include frames which do not represent the water activity from the label at the start and end of the footage. The quality of the training data could have also been impacted by inaccurate or incomplete YOLOv7 pose estimation.

3.3 Integrated Drowning Detection System

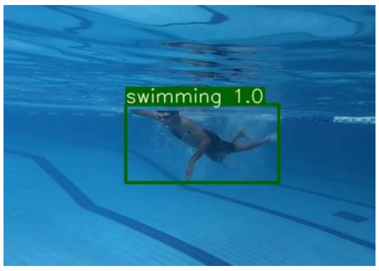
The proposed drowning detection system is created by integrating the pose estimation algorithm with a neural network classification model. This system displays the footage (in this case, the videos from the test partition of the dataset), puts a bounding box around the detected person in the frame and displays the result of the classification. Additionally, The colour of the bounding box depends on the predicted class. The red colour represents “drowning”, yellow represents “idle”, and the green represents “swimming”. This had been done to quickly draw the attention of the person monitoring the swimming pool in case of a potential drowning incident. Figure 3.8 shows how the system displays each of the three classes.



(a) Drowning prediction.



(b) Idle prediction.



(c) Swimming prediction.

Figure 3.8: An example of successful classification in the proposed system.

3.3.1 Latency in the Predictions

A crucial factor in the effectiveness of the drowning detection system is latency and fluency in class prediction. To evaluate the performance of the proposed system, times of the following four execution stages were measured: YOLOv7 inference, data preparation, pose classification and display of the results. Additionally, the frame rate of the system was calculated to compare it to one of the input videos (30 FPS). Table 3.2 shows mentioned measurements for three versions of the proposed classification models and a baseline model. The values in the table are the average times it took the different system versions to process and display 30 frames of the video footage.

Model	YOLOv7 inference	Data pre-processing	NN Model inference	Displaying results	Total	FPS
Baseline	1.60	0.02	0.02	0.48	2.12	15.08
CNN-SINGLE	1.71	0.02	0.04	0.42	2.19	14.71
CNN-SEQ	1.72	0.02	0.05	0.44	2.23	14.56
CNN-RNN	1.55	0.02	0.53	0.44	2.54	12.39

Table 3.2: Overhead times in the proposed system by the model used.

Depending on the model, the speed of the visualisation varies. The system built on the baseline model offers the least latency in displaying the results. Following closely are the CNN-SINGLE and CNN-SEQ models, with only a slight increase in latency. The increase in overheads is visible in a CNN-RNN model, where the inference time of the neural network is ten times that of the CNN-SEQ model. The overall latency is the biggest in the system employing the CNN-RNN model. Its neural network inference time is ten times that of the CNN-SEQ model. The data preparation times stay the same for all models because the method does not change significantly. It must be noted that the experiments were done on Nvidia GTX 1070 GPU. A significant factor contributing to the latency of all system versions is the YOLOv7 inference time, which stays around 1.7 seconds for every 30 frames (1 second of input video).

3.3.2 Prediction Jittering

Another important factor influencing the effectiveness of the drowning detection system is the jittering of predictions. Jitter measures how often the predicted class changes, potentially misclassifying the input and falsely drawing the attention of the person monitoring the swimming pool. To measure jitter, the number of class changes per 30 frames of each system was counted while running predictions on the test data. Additionally, the average uninterrupted display time of a class was calculated. Table 3.3 shows the jittering in proposed systems employing three models and the baseline system.

Model	Baseline	CNN-SINGLE	CNN-SEQ	CNN-RNN
Changes per 30 frames	3.05	2.67	0.36	0.39
Avg uninterrupted time (frames)	9.85	11.22	83.33	77.46

Table 3.3: Frequency of changes of predicted class and the average uninterrupted display time.

All three models developed for the purpose of this project reduced the jitter in the final system when compared to the baseline. The greatest improvement is visible in the CNN-SEQ model and the CNN-RNN model. This is probably due to the fact that the classification of each frame in these models is partly dependent on up to 30 previous frames, considerably reducing variations in predicted classes.

3.3.3 Evaluation of the Drowning Detection System

The best-performing version of the drowning detection system, with respect to the lowest latency and jitter, is built on the CNN-SEQ model. It has one of the lowest overheads in displaying the classification results while having the lowest number of predictions changed per 30 frames. When taking into account the results from section 3.2, the system, which has the best performance while still maintaining low jittering and latency, is the one built on the CNN-RNN model. With this model, the system's accuracy increases by 6% when compared to the CNN-SEQ and 11% when compared to the system employing the baseline model.

The performance of all proposed systems is greatly influenced by the inference time of the YOLOv7 pose estimation algorithm, which on average, takes 1.7 seconds per inference on 30 frames. Jittering, while low in some instances, still exists in all system versions. This could cause false alarms if the system were to be used as an aid by the person monitoring the swimming pool.

Chapter 4 - Discussion

4.1 Conclusions

This project demonstrated the feasibility of constructing a drowning detection system using a pose estimation algorithm and neural network classification model to assist lifeguards in monitoring swimming pools. The proposed method is more convenient than some previous approaches to creating such a system because it eliminates the need for wearable sensors and strategically placed cameras to track and monitor individuals' behaviour in swimming pools.

The "Water Behavior" dataset used in this project enabled the training of the system's neural network components. It was created with the assistance of a professional lifeguard team to ensure accurate simulation of water activities, particularly drowning. However, the dataset has certain limitations, as it only includes young male adults, lacking diversity and omitting women and children. Diversified data is essential for accurate model training and classification. Additionally, despite the dataset description stating that each instance contained only one person performing a water activity, some frames included two individuals showing two different behaviours, potentially causing inaccurate predictions in the drowning detection system.

YOLOv7 was employed for pose estimation in the proposed system due to its compromise between fast inference and accuracy when compared to other methods. Its PyTorch implementation facilitated easy integration into the drowning detection system, providing real-time keypoint extraction. It managed to extract body poses from most video frames in the dataset providing around 80,000 instances (with data augmentation) to train the models. This allowed the system to be partially invariant to the environment changes, as the models were trained only on the keypoint representation of the body poses without including information such as the colour of the background. Although YOLOv7's pose estimation was generally accurate, it occasionally struggled with estimating poses obscured by water splashes and reflections and failed to estimate the skeletons of upside-down subjects accurately.

The project successfully experimented with and built neural network classification models for activity classification. Three CNN-based models were proposed, with the best-performing model being the combined CNN-RNN model, capable of processing spatiotemporal information. Each model was built on the basis of the previous one, improving its accuracy in predicting water behaviours. Despite showing promise in classifying water actions, the models' accuracy was lower than expected, leaving room for improvement. From testing in Chapter 3, it could be seen that the models struggled with the same types of videos from the testing set. One possible reason for the not-so-excellent performance includes poor data quality as a result of flawed pose estimation with YOLOv7 or a small and imbalanced dataset.

The final system integrated the YOLOv7 pose estimation algorithm, data processing methods, and a classification model. It is capable of extracting an individual's body pose from video footage and predicting their water activity in a single pipeline. The system's latency and fluency were significantly impacted by YOLOv7’s inference time, which significantly reduced the number of frames displayed per second. A more powerful GPU may resolve the slow execution time of the inference. Moreover, systems built on classification models that processed sequences as input showed less jitter in class predictions, potentially leading to fewer false alarms. Unfortunately, due to the not ideal accuracies achieved by the classification models, the system should only be considered a tool to aid lifeguards in their jobs, but it should not be completely relied on when it comes to detecting emergencies.

4.2 Limitations

Hardware Costs

One of the limitations includes the cost of deploying such a system. The proposed drowning detection system requires a camera overlooking the swimming pool from an overhead or underwater angle. Moreover, it requires a machine capable of hosting this system with a good-quality GPU to work fluently in real time. Even though these hardware requirements are more feasible than those of other discussed systems, their cost might still be too high, especially for small, public swimming pools.

Robustness

In its current state, the developed system is limited to the classification of the actions of only one person in the frame. This means that in a real swimming pool environment, the actions of only one of the multiple persons would be tracked and classified. This, of course, is not desired as the person in trouble might be among all the other people not tracked by the system. Thus, for deployment in a real-life environment, multi-person detection needs to be employed.

Access to the dataset

Additionally, if anyone were interested in improving the proposed drowning detection system, they would have to request access to the “Water Behavior” dataset personally. Due to the signed release agreement, the dataset cannot be published along with the code and report for this project. This means that unless an alternative dataset can be found, proposed neural network models cannot be trained or tested. Saved models can still be used to predict water behaviours in the original footage, however, they have not been tested on any data outside the mentioned dataset.

4.3 Ideas for future work

There is a lot of room to improve the proposed drowning detection system. In its current state, the system cannot be used reliably to replace lifeguards. However, there are several potential ways to improve drowning detection based on pose estimation and neural network classification.

Improvement to the model architectures

In the short term, more testing and hyperparameter optimisation can be done to improve the classification results of the neural network models. More optimisation trials could potentially help to find more suitable hyperparameters as well as optimise those that were not considered in this project, such as the number of hidden layers, stride sizes, etc. Furthermore, more experimentation with the model architectures can lead to a good compromise between inference time and the accuracy of predictions. Other convolutional layers (Conv1D and Conv3D) could be experimented with to see if they would yield better results. The models could be built, trained and tested on a much more powerful architecture than the one used for the purpose of this project, such as Google Colab, Microsoft Azure or AWS servers.

Live system

A live system could be developed and employed in a real swimming pool environment. In order to do that, the system should have access to the live CCTV feed of the swimming pool. This could be done by changing the footage input source in the code for the system from test data to live camera footage. Moreover, a system deployed in a real-life environment should be able to perform multiperson classification. Fortunately, the pose estimation implementation of YOLOv7 already supports multiperson inference. Hence, the way the system processes and feeds the data to a neural network model could be changed to classify the actions of more than just one individual per frame and plot all predictions at the same time.

Larger Dataset

In the long term, a larger, well-balanced dataset could be developed to provide data for neural network model training. A dataset consisting of footage from various environments with different lighting, including individuals of different genders, ages and body physiques, could possibly lead to better classification results in real-world scenarios. It would also help if each frame were labelled separately, allowing for a more accurate data description. The new dataset could combine the simulated videos recorded by professional lifeguards with videos from real incidents recorded by the swimming pool facilities or found on the internet.

References

[1] WHO, ‘Drowning’, Nov. 17, 2014. https://www.who.int/news-room/fact-sheets/detail/drowning (accessed Apr. 17, 2023).

[2] WHO, ‘Drownings’, 2021. https://platform.who.int/mortality/themes/theme-details/topics/indicator-groups/indicator-group-details/MDB/drownings (accessed Apr. 17, 2023).

[3] A. R. Pelletier and J. Gilchrist, ‘Fatalities in swimming pools with lifeguards: USA, 2000–2008’, *Inj. Prev.*, vol. 17, no. 4, pp. 250–253, Aug. 2011, doi: 10.1136/ip.2010.029751.

[4] J. Bierens and A. Scapigliati, ‘Drowning in swimming pools’, *Microchem. J.*, vol. 113, pp. 53–58, Mar. 2014, doi: 10.1016/j.microc.2013.10.003.

[5] R. I. L. L.-J. Rosenfeld, ‘The Owner’s Liability for a Swimming Pool Drowning’, *Lexology*, Sep. 03, 2020. https://www.lexology.com/library/detail.aspx?g=f50250ce-2831-4909-85cd-ad52caa98e02 (accessed Apr. 17, 2023).

[6] M. Martin, G. Sadlo, and G. Stew, ‘The phenomenon of boredom’, *Qual. Res. Psychol.*, vol. 3, no. 3, pp. 193–211, Jan. 2006, doi: 10.1191/1478088706qrp066oa.

[7] D. Whitney and D. M. Levi, ‘Visual crowding: a fundamental limit on conscious perception and object recognition’, *Trends Cogn. Sci.*, vol. 15, no. 4, pp. 160–168, Apr. 2011, doi: 10.1016/j.tics.2011.02.005.

[8] P. Suominen, C. Baillie, R. Korpela, S. Rautanen, S. Ranta, and K. T. Olkkola, ‘Impact of age, submersion time and water temperature on outcome in near-drowning’, *Resuscitation*, vol. 52, no. 3, pp. 247–254, Mar. 2002, doi: 10.1016/S0300-9572(01)00478-6.

[9] F. Lei, H. Zhu, F. Tang, and X. Wang, ‘Drowning behavior detection in swimming pool based on deep learning’, *Signal Image Video Process.*, vol. 16, no. 6, pp. 1683–1690, Sep. 2022, doi: 10.1007/s11760-021-02124-9.

[10] WAVEDDS, ‘Intro | WAVE Drowning Detection Systems | United States’, *wavedds*, 2021. https://www.wavedds.com (accessed Apr. 08, 2023).

[11] S. B. Gay, C. L. Sistrom, C. A. Holder, and P. M. Suratt, ‘Breath-holding capability of adults. Implications for spiral computed tomography, fast-acquisition magnetic resonance imaging, and angiography’, *Invest. Radiol.*, vol. 29, no. 9, pp. 848–851, Sep. 1994.

[12] M. B. Johnson and K. A. Lawson, ‘Evaluation of the WAVE Drowning Detection SystemTM for use with children’s summer camp groups in swimming pools: A prospective observational study’, *Int. J. Crit. Illn. Inj. Sci.*, vol. 12, no. 4, p. 184, Jan. 2022, doi: 10.4103/ijciis.ijciis_24_22.

[13] N. Salehi, M. Keyvanara, and S. Monadjemmi, ‘An Automatic Video-based Drowning Detection System for Swimming Pools Using Active Contours’, *Int. J. Image Graph. Signal Process.*, vol. 8, pp. 1–8, Aug. 2016, doi: 10.5815/ijigsp.2016.08.01.

[14] S. Saravanakumar, A. Vadivel, and C. G. S. Ahmed, ‘Multiple object tracking using HSV color space’, in *Proceedings of the 2011 International Conference on Communication,*

Computing & Security, in ICCCS ’11. New York, NY, USA: Association for Computing Machinery, Feb. 2011, pp. 247–252. doi: 10.1145/1947940.1947993.

[15] ‘An Overview of Contour Detection Approaches’. https://mi-research.net/en/article/doi/10.1007/s11633-018-1117-z (accessed Apr. 26, 2023).

[16] M. R. Paulsen and W. F. McClure, ‘Illumination for Computer Vision Systems’, *Trans. ASAE*, vol. 29, no. 5, pp. 1398–1404, 1986, doi: 10.13031/2013.30328.

[17] 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)*, Nov. 2021, pp. 1–5. doi: 10.1109/IGESSC53124.2021.9618700.

[18] K. Sun, B. Xiao, D. Liu, and J. Wang, ‘Deep High-Resolution Representation Learning for Human Pose Estimation’, presented at the Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 5693–5703. Accessed: Apr. 08, 2023. [Online]. Available: https://openaccess.thecvf.com/content_CVPR_2019/html/Sun_Deep_High-Resolution_Representation_Learning_for_Human_Pose_Estimation_CVPR_2019_paper.html

[19] K. He, X. Zhang, S. Ren, and J. Sun, ‘Deep Residual Learning for Image Recognition’. arXiv, Dec. 10, 2015. doi: 10.48550/arXiv.1512.03385.

[20] K. Simonyan and A. Zisserman, ‘Very Deep Convolutional Networks for Large-Scale Image Recognition’. arXiv, Apr. 10, 2015. doi: 10.48550/arXiv.1409.1556.

[21] A. G. Howard *et al.*, ‘MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications’. arXiv, Apr. 16, 2017. doi: 10.48550/arXiv.1704.04861.

[22] T.-Y. Lin *et al.*, ‘Microsoft COCO: Common Objects in Context’. arXiv, Feb. 20, 2015. doi: 10.48550/arXiv.1405.0312.

[23] E. Nishani and B. Çiço, ‘Computer vision approaches based on deep learning and neural networks: Deep neural networks for video analysis of human pose estimation’, in *2017 6th Mediterranean Conference on Embedded Computing (MECO)*, Jun. 2017, pp. 1–4. doi: 10.1109/MECO.2017.7977207.

[24] H. Liu, T. Liu, Z. Zhang, A. K. Sangaiah, B. Yang, and Y. Li, ‘ARHPE: Asymmetric Relation-Aware Representation Learning for Head Pose Estimation in Industrial Human–Computer Interaction’, *IEEE Trans. Ind. Inform.*, vol. 18, no. 10, pp. 7107–7117, Oct. 2022, doi: 10.1109/TII.2022.3143605.

[25] E. Marchand, H. Uchiyama, and F. Spindler, ‘Pose Estimation for Augmented Reality: A Hands-On Survey’, *IEEE Trans. Vis. Comput. Graph.*, vol. 22, no. 12, pp. 2633–2651, Dec. 2016, doi: 10.1109/TVCG.2015.2513408.

[26] W. Bao, T. Niu, N. Wang, and X. Yang, ‘Pose estimation and motion analysis of ski jumpers based on ECA-HRNet’, *Sci. Rep.*, vol. 13, no. 1, Art. no. 1, Apr. 2023, doi: 10.1038/s41598-023-32893-x.

[27] K. M. Kulkarni and S. Shenoy, ‘Table Tennis Stroke Recognition Using Two-Dimensional Human Pose Estimation’, presented at the Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 4576–4584. Accessed: Apr. 26, 2023. [Online]. Available:

https://openaccess.thecvf.com/content/CVPR2021W/CVSports/html/Kulkarni_Table_Tennis_Stroke_Recognition_Using_Two-Dimensional_Human_Pose_Estimation_CVPRW_2021_paper.html

[28] Q. Nie, ‘Human Pose Estimation and Action Recognition Using Deep Learning’, phd, The Chinese University of Hong Kong (Hong Kong), 2020.

[29] C. Lugaresi *et al.*, ‘MediaPipe: A Framework for Building Perception Pipelines’. arXiv, Jun. 14, 2019. doi: 10.48550/arXiv.1906.08172.

[30] V. Bazarevsky, I. Grishchenko, K. Raveendran, T. Zhu, F. Zhang, and M. Grundmann, ‘BlazePose: On-device Real-time Body Pose tracking’. arXiv, Jun. 17, 2020. doi: 10.48550/arXiv.2006.10204.

[31] A. Kumar, Z. J. Zhang, and H. Lyu, ‘Object detection in real time based on improved single shot multi-box detector algorithm’, *EURASIP J. Wirel. Commun. Netw.*, vol. 2020, no. 1, p. 204, Oct. 2020, doi: 10.1186/s13638-020-01826-x.

[32] ‘google/mediapipe’. Google, Apr. 26, 2023. Accessed: Apr. 26, 2023. [Online]. Available: https://github.com/google/mediapipe/blob/507ed0d91de4d82fbc6053e510c42514f2307221/docs/solutions/pose.md

[33] T.-Y. Lin *et al.*, ‘Microsoft COCO: Common Objects in Context’, in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., in Lecture Notes in Computer Science. Cham: Springer International Publishing, 2014, pp. 740–755. doi: 10.1007/978-3-319-10602-1_48.

[34] K. W. Zovaco Founder, ‘Human Pose Estimation: OpenPose vs. HRNet’, *Medium*, Oct. 20, 2020. https://kartikwason.medium.com/human-pose-estimation-openpose-vs-hrnet-e8fa37768929 (accessed Apr. 26, 2023).

[35] J. Wang, S. Jin, W. Liu, W. Liu, C. Qian, and P. Luo, ‘When Human Pose Estimation Meets Robustness: Adversarial Algorithms and Benchmarks’, presented at the Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 11855–11864. Accessed: Apr. 26, 2023. [Online]. Available: https://openaccess.thecvf.com/content/CVPR2021/html/Wang_When_Human_Pose_Estimation_Meets_Robustness_Adversarial_Algorithms_and_Benchmarks_CVPR_2021_paper.html

[36] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, ‘YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors’. arXiv, Jul. 06, 2022. doi: 10.48550/arXiv.2207.02696.

[37] D. Maji, S. Nagori, M. Mathew, and D. Poddar, ‘YOLO-Pose: Enhancing YOLO for Multi Person Pose Estimation Using Object Keypoint Similarity Loss’. arXiv, Apr. 14, 2022. doi: 10.48550/arXiv.2204.06806.

[38] https://www.linkedin.com/in/david-landup-859455144, ‘Real-Time Pose Estimation from Video in Python with YOLOv7’, *Stack Abuse*, Oct. 04, 2022. https://stackabuse.com/real-time-pose-estimation-from-video-in-python-with-yolov7/ (accessed Apr. 26, 2023).

[39] ‘YOLOv7 Pose vs MediaPipe in Human Pose Estimation’, Oct. 18, 2022. https://learnopencv.com/yolov7-pose-vs-mediapipe-in-human-pose-estimation/

(accessed Apr. 25, 2023).

[40] A. Singh, N. Thakur, and A. Sharma, ‘A review of supervised machine learning algorithms’, in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, Mar. 2016, pp. 1310–1315.

[41] P. Zhang, ‘Neural Networks for Classification: A Survey’, *Syst. Man Cybern. Part C Appl. Rev. IEEE Trans. On*, vol. 30, pp. 451–462, Dec. 2000, doi: 10.1109/5326.897072.

[42] Y. Ren and Y. Zhang, ‘Deceptive Opinion Spam Detection Using Neural Network’, in *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, Osaka, Japan: The COLING 2016 Organizing Committee, Dec. 2016, pp. 140–150. Accessed: Apr. 26, 2023. [Online]. Available: https://aclanthology.org/C16-1014

[43] F. Amato, A. López, E. M. Peña-Méndez, P. Vaňhara, A. Hampl, and J. Havel, ‘Artificial neural networks in medical diagnosis’, *J. Appl. Biomed.*, vol. 11, no. 2, pp. 47–58, Jan. 2013, doi: 10.2478/v10136-012-0031-x.

[44] G. Lou and H. Shi, ‘Face image recognition based on convolutional neural network’, *China Commun.*, vol. 17, no. 2, pp. 117–124, Feb. 2020, doi: 10.23919/JCC.2020.02.010.

[45] ‘Explained: Neural networks’, *MIT News | Massachusetts Institute of Technology*, Apr. 14, 2017. https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414 (accessed Apr. 26, 2023).

[46] J. J. Hopfield, ‘Neural networks and physical systems with emergent collective computational abilities.’, *Proc. Natl. Acad. Sci. U. S. A.*, vol. 79, no. 8, pp. 2554–2558, Apr. 1982.

[47] J. Schmidhuber, ‘Deep Learning in Neural Networks: An Overview’, *Neural Netw.*, vol. 61, pp. 85–117, Jan. 2015, doi: 10.1016/j.neunet.2014.09.003.

[48] I. N. da Silva, D. Hernane Spatti, R. Andrade Flauzino, L. H. B. Liboni, and S. F. dos Reis Alves, ‘Artificial Neural Network Architectures and Training Processes’, in *Artificial Neural Networks : A Practical Course*, I. N. da Silva, D. Hernane Spatti, R. Andrade Flauzino, L. H. B. Liboni, and S. F. dos Reis Alves, Eds., Cham: Springer International Publishing, 2017, pp. 21–28. doi: 10.1007/978-3-319-43162-8_2.

[49] H. K. Lam *et al.*, ‘A study of neural-network-based classifiers for material classification’, *Neurocomputing*, vol. 144, pp. 367–377, Nov. 2014, doi: 10.1016/j.neucom.2014.05.019.

[50] H. Wang and B. Raj, ‘On the Origin of Deep Learning’. arXiv, Mar. 02, 2017. doi: 10.48550/arXiv.1702.07800.

[51] D. Singh *et al.*, ‘Human Activity Recognition using Recurrent Neural Networks’, 2017, pp. 267–274. doi: 10.1007/978-3-319-66808-6_18.

[52] Md. M. Islam, S. Nooruddin, F. Karray, and G. Muhammad, ‘Human activity recognition using tools of convolutional neural networks: A state of the art review, data sets, challenges, and future prospects’, *Comput. Biol. Med.*, vol. 149, p. 106060, Oct. 2022, doi: 10.1016/j.compbiomed.2022.106060.

[53] S. Zhu, R. G. Guendel, A. Yarovoy, and F. Fioranelli, ‘Continuous Human Activity Recognition With Distributed Radar Sensor Networks and CNN–RNN Architectures’, *IEEE Trans. Geosci. Remote Sens.*, vol. 60, pp. 1–15, 2022, doi: 10.1109/TGRS.2022.3189746.

[54] N. Sharma, V. Jain, and A. Mishra, ‘An Analysis Of Convolutional Neural Networks For Image Classification’, *Procedia Comput. Sci.*, vol. 132, pp. 377–384, Jan. 2018, doi: 10.1016/j.procs.2018.05.198.

[55] K. Li, W. Ma, U. Sajid, Y. Wu, and G. Wang, ‘Object Detection with Convolutional Neural Networks’. arXiv, Dec. 04, 2019. doi: 10.48550/arXiv.1912.01844.

[56] M. Alam, J.-F. Wang, C. Guangpei, L. Yunrong, and Y. Chen, ‘Convolutional Neural Network for the Semantic Segmentation of Remote Sensing Images’, *Mob. Netw. Appl.*, vol. 26, no. 1, pp. 200–215, Feb. 2021, doi: 10.1007/s11036-020-01703-3.

[57] P. Gavrikov and J. Keuper, ‘CNN Filter DB: An Empirical Investigation of Trained Convolutional Filters’, in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2022, pp. 19044–19054. doi: 10.1109/CVPR52688.2022.01848.

[58] L. Alzubaidi *et al.*, ‘Review of deep learning: concepts, CNN architectures, challenges, applications, future directions’, *J. Big Data*, vol. 8, no. 1, p. 53, Mar. 2021, doi: 10.1186/s40537-021-00444-8.

[59] S. Verma, ‘Understanding 1D and 3D Convolution Neural Network | Keras’, *Medium*, Apr. 05, 2022. https://towardsdatascience.com/understanding-1d-and-3d-convolution-neural-network-keras-9d8f76e29610 (accessed Apr. 27, 2023).

[60] ‘2D Convolution in Image Processing - Technical Articles’. https://www.allaboutcircuits.com/technical-articles/two-dimensional-convolution-in-image-processing/ (accessed Apr. 27, 2023).

[61] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman, ‘1D convolutional neural networks and applications: A survey’, *Mech. Syst. Signal Process.*, vol. 151, p. 107398, Apr. 2021, doi: 10.1016/j.ymssp.2020.107398.

[62] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, ‘Learning Spatiotemporal Features with 3D Convolutional Networks’. arXiv, Oct. 06, 2015. doi: 10.48550/arXiv.1412.0767.

[63] R. Nirthika, S. Manivannan, A. Ramanan, and R. Wang, ‘Pooling in convolutional neural networks for medical image analysis: a survey and an empirical study’, *Neural Comput. Appl.*, vol. 34, no. 7, pp. 5321–5347, Apr. 2022, doi: 10.1007/s00521-022-06953-8.

[64] H. Gholamalinezhad and H. Khosravi, ‘Pooling Methods in Deep Neural Networks, a Review’. arXiv, Sep. 16, 2020. doi: 10.48550/arXiv.2009.07485.

[65] S. H. S. Basha, S. R. Dubey, V. Pulabaigari, and S. Mukherjee, ‘Impact of Fully Connected Layers on Performance of Convolutional Neural Networks for Image Classification’, *Neurocomputing*, vol. 378, pp. 112–119, Feb. 2020, doi: 10.1016/j.neucom.2019.10.008.

[66] Z. C. Lipton, J. Berkowitz, and C. Elkan, ‘A Critical Review of Recurrent Neural Networks for Sequence Learning’. arXiv, Oct. 17, 2015. doi: 10.48550/arXiv.1506.00019.

[67] R. DiPietro and G. D. Hager, ‘Chapter 21 - Deep learning: RNNs and LSTM’, in *Handbook of Medical Image Computing and Computer Assisted Intervention*, S. K. Zhou, D. Rueckert, and G. Fichtinger, Eds., in The Elsevier and MICCAI Society Book Series. Academic Press, 2020, pp. 503–519. doi: 10.1016/B978-0-12-816176-0.00026-0.

[68] ‘What are Recurrent Neural Networks? | IBM’. https://www.ibm.com/topics/recurrent-neural-networks (accessed Apr. 27, 2023).

[69] ‘Types of Recurrent Neural Networks (RNN) in Tensorflow’, *GeeksforGeeks*, Jan. 27, 2022. https://www.geeksforgeeks.org/types-of-recurrent-neural-networks-rnn-in-tensorflow/ (accessed Apr. 27, 2023).

[70] R. Pascanu, T. Mikolov, and Y. Bengio, ‘On the difficulty of training Recurrent Neural Networks’. arXiv, Feb. 15, 2013. doi: 10.48550/arXiv.1211.5063.

[71] S. Hochreiter and J. Schmidhuber, ‘Long Short-Term Memory’, *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[72] M. Ravanelli, P. Brakel, M. Omologo, and Y. Bengio, ‘Improving speech recognition by revising gated recurrent units’. arXiv, Sep. 29, 2017. doi: 10.48550/arXiv.1710.00641.

[73] X. Zhang, F. Chen, and R. Huang, ‘A Combination of RNN and CNN for Attention-based Relation Classification’, *Procedia Comput. Sci.*, vol. 131, pp. 911–917, Jan. 2018, doi: 10.1016/j.procs.2018.04.221.

[74] D. Quang and X. Xie, ‘DanQ: a hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences’, *Nucleic Acids Res.*, vol. 44, no. 11, p. e107, Jun. 2016, doi: 10.1093/nar/gkw226.

[75] C. Xu *et al.*, ‘Recurrent Convolutional Neural Network for Sequential Recommendation’, in *The World Wide Web Conference*, in WWW ’19. New York, NY, USA: Association for Computing Machinery, May 2019, pp. 3398–3404. doi: 10.1145/3308558.3313408.

[76] M. Liang and X. Hu, ‘Recurrent Convolutional Neural Network for Object Recognition’, presented at the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 3367–3375. Accessed: Apr. 27, 2023. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2015/html/Liang_Recurrent_Convolutional_Neural_2015_CVPR_paper.html

[77] T. Linjordet and K. Balog, ‘Impact of Training Dataset Size on Neural Answer Selection Models’. arXiv, Jan. 29, 2019. doi: 10.48550/arXiv.1901.10496.

[78] E. Romani, ‘How to estimate data collection costs for your Data Science project’, *Medium*, Feb. 27, 2022. https://towardsdatascience.com/how-to-estimate-data-collection-costs-for-your-data-science-project-8938ca9acc5f (accessed Apr. 27, 2023).

[79] ‘HackHarvard 2017’, *HackHarvard 2017*. https://hackharvard-2017.devpost.com/ (accessed Apr. 27, 2023).

[80] ‘lifeguard.io’, *Devpost*, Oct. 22, 2017. https://devpost.com/software/lifeguard-io (accessed Apr. 08, 2023).

[81] sudipta swarnakar, ‘LifeGuard.IO’. Apr. 10, 2023. Accessed: Apr. 27, 2023. [Online]. Available: https://github.com/sswarnak77/LifeGuard.IO

[82] M. Pruciak, ‘Python vs JavaScript: Which Is Better For Machine Learning Project?’ https://www.ideamotive.co/blog/python-vs-javascript (accessed Apr. 27, 2023).

[83] ‘Difference between High Level and Low level languages’, *GeeksforGeeks*, May 24, 2019. https://www.geeksforgeeks.org/difference-between-high-level-and-low-level-languages/ (accessed Apr. 27, 2023).

[84] A. Paszke *et al.*, ‘PyTorch: An Imperative Style, High-Performance Deep Learning Library’. arXiv, Dec. 03, 2019. doi: 10.48550/arXiv.1912.01703.

[85] C. R. Harris *et al.*, ‘Array Programming with NumPy’, *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020, doi: 10.1038/s41586-020-2649-2.

[86] I. Culjak, D. Abram, T. Pribanic, H. Dzapo, and M. Cifrek, ‘A brief introduction to OpenCV’, in *2012 Proceedings of the 35th International Convention MIPRO*, May 2012, pp. 1725–1730.

[87] F. Pedregosa *et al.*, ‘Scikit-learn: Machine Learning in Python’. arXiv, Jun. 05, 2018. doi: 10.48550/arXiv.1201.0490.

[88] J. D. Hunter, ‘Matplotlib: A 2D Graphics Environment’, *Comput. Sci. Eng.*, vol. 9, no. 3, pp. 90–95, May 2007, doi: 10.1109/MCSE.2007.55.

[89] M. Waskom, ‘seaborn: statistical data visualization’, *J. Open Source Softw.*, vol. 6, p. 3021, Apr. 2021, doi: 10.21105/joss.03021.

[90] J. Dsouza, ‘What is a GPU and do you need one in Deep Learning?’, *Medium*, Dec. 26, 2020. https://towardsdatascience.com/what-is-a-gpu-and-do-you-need-one-in-deep-learning-718b9597aa0d (accessed Apr. 27, 2023).

[91] I. Buck, ‘GPU computing with NVIDIA CUDA’, in *ACM SIGGRAPH 2007 courses*, in SIGGRAPH ’07. New York, NY, USA: Association for Computing Machinery, Aug. 2007, pp. 6-es. doi: 10.1145/1281500.1281647.

[92] K.-Y. Wong, ‘Official YOLOv7’. Apr. 27, 2023. Accessed: Apr. 27, 2023. [Online]. Available: https://github.com/WongKinYiu/yolov7

[93] ‘WongKinYiu/yolov7 at pose’. https://github.com/WongKinYiu/yolov7/tree/pose (accessed Apr. 27, 2023).

[94] G. Jocher, A. Chaurasia, and J. Qiu, ‘YOLO by Ultralytics’. Jan. 2023. Accessed: Apr. 08, 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[95] ‘Half-precision floating-point format’, *Wikipedia*. Apr. 27, 2023. Accessed: Apr. 30, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Half-precision_floating-point_format&oldid=1151975144

[96] A. Neubeck and L. Van Gool, ‘Efficient Non-Maximum Suppression’, in *18th International Conference on Pattern Recognition (ICPR’06)*, Aug. 2006, pp. 850–855. doi: 10.1109/ICPR.2006.479.

[97] J. Sola and J. Sevilla, ‘Importance of input data normalization for the application of neural networks to complex industrial problems’, *IEEE Trans. Nucl. Sci.*, vol. 44, no. 3, pp. 1464–1468, Jun. 1997, doi: 10.1109/23.589532.

[98] S. Yang, W. Xiao, M. Zhang, S. Guo, J. Zhao, and F. Shen, ‘Image Data Augmentation

for Deep Learning: A Survey'. arXiv, Apr. 18, 2022. doi: 10.48550/arXiv.2204.08610.

[99] A. MOAWAD, 'Linear layers explained in a simple way', *DataThings*, Jul. 28, 2020. https://medium.com/datathings/linear-layers-explained-in-a-simple-way-2319a9c2d1aa (accessed Apr. 30, 2023).

[100]M. Fayyaz *et al.*, '3D CNNs with Adaptive Temporal Feature Resolutions'. arXiv, Aug. 11, 2021. doi: 10.48550/arXiv.2011.08652.

[101]M. Dwarampudi and N. V. S. Reddy, 'Effects of padding on LSTMs and CNNs'. arXiv, Mar. 18, 2019. doi: 10.48550/arXiv.1903.07288.

[102]Z. Sun, M. Ozay, and T. Okatani, 'Design of Kernels in Convolutional Neural Networks for Image Classification', Nov. 2015.

[103]Y. Hu, A. Huber, J. Anumula, and S.-C. Liu, 'Overcoming the vanishing gradient problem in plain recurrent networks'. arXiv, Jul. 05, 2019. doi: 10.48550/arXiv.1801.06105.

[104]J. Zhang, T. He, S. Sra, and A. Jadbabaie, 'Why gradient clipping accelerates training: A theoretical justification for adaptivity'. arXiv, Feb. 10, 2020. doi: 10.48550/arXiv.1905.11881.

[105]S. Yang, X. Yu, and Y. Zhou, 'LSTM and GRU Neural Network Performance Comparison Study: Taking Yelp Review Dataset as an Example', in *2020 International Workshop on Electronic Communication and Artificial Intelligence (IWECAI)*, Jun. 2020, pp. 98–101. doi: 10.1109/IWECAI50956.2020.00027.

[106]S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, 'Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark'. arXiv, Jun. 28, 2022. doi: 10.48550/arXiv.2109.14545.

[107]M. Uzair and N. Jamil, 'Effects of Hidden Layers on the Efficiency of Neural networks', in *2020 IEEE 23rd International Multitopic Conference (INMIC)*, Nov. 2020, pp. 1–6. doi: 10.1109/INMIC50486.2020.9318195.

[108]T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, 'Optuna: A Next-generation Hyperparameter Optimization Framework', in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, in KDD '19. New York, NY, USA: Association for Computing Machinery, Jul. 2019, pp. 2623–2631. doi: 10.1145/3292500.3330701.

[109]'Hyperparameter tuning with Ray Tune — PyTorch Tutorials 2.0.0+cu117 documentation'. https://pytorch.org/tutorials/beginner/hyperparameter_tuning_tutorial.html (accessed Apr. 30, 2023).

[110]T. Yu and H. Zhu, 'Hyper-Parameter Optimization: A Review of Algorithms and Applications'. arXiv, Mar. 12, 2020. doi: 10.48550/arXiv.2003.05689.

[111]K. Janocha and W. M. Czarnecki, 'On Loss Functions for Deep Neural Networks in Classification'. arXiv, Feb. 18, 2017. doi: 10.48550/arXiv.1702.05659.

[112]A. Kumar, 'Mean Squared Error vs Cross Entropy Loss Function', *Data Analytics*, Apr. 03, 2023. https://vitalflux.com/mean-squared-error-vs-cross-entropy-loss-function/ (accessed Apr. 30, 2023).

[113] J. S. Bridle, ‘Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters’, in *Proceedings of the 2nd International Conference on Neural Information Processing Systems*, in NIPS’89. Cambridge, MA, USA: MIT Press, Jan. 1989, pp. 211–217.

[114] Musstafa, ‘Optimizers in Deep Learning’, *MLearning.ai*, Feb. 12, 2022. https://medium.com/mlearning-ai/optimizers-in-deep-learning-7bf81fed78a0 (accessed Apr. 30, 2023).

[115] C. Desai, ‘Comparative Analysis of Optimizers in Deep Neural Networks’, Oct. 2020.

[116] GNU, ‘The GNU General Public License v3.0 - GNU Project - Free Software Foundation’, Jun. 29, 2007. https://www.gnu.org/licenses/gpl-3.0.en.html (accessed Apr. 25, 2023).

[117] BCS, ‘BCS Code of Conduct for members - Ethics for IT professionals | BCS’, 2023. https://www.bcs.org/membership-and-registrations/become-a-member/bcs-code-of-conduct/ (accessed Apr. 25, 2023).

Appendix A - Self Appraisal

A.1 Critical self-evaluation

The main goals of this project were to develop and evaluate a drowning detection system and its elements. These goals were successfully achieved. Pose estimation techniques were explored, and one was chosen to extract body poses. Three CNN-based neural network models capable of making predictions based on these extracted poses were built and evaluated. Finally, three versions of working drowning detection systems were created by integrating the pose estimation method with each of the developed models. The system was evaluated, and the best version was chosen. Even though the evaluation metrics were not excellent, the developed system demonstrated the potential to classify water-related activities correctly.

However, there were some initial goals which I did not manage to reach due to the hardware and time limitations. The main challenge I faced in this project was setting my goals too high with respect to time and resources available. Goals such as developing a custom test set by simulating three kinds of water activity myself (swim, idle and drown) were not feasible due to the amount of time and work they required in order to gain a permit to record at a local swimming pool. Moreover, I wanted to try and compare pose estimation methods in the proposed system, but I found out that it required much more time than I could spend and computational power my local machine did not have. Furthermore, gaining access to the "Water Behavior" dataset took much longer than expected. I got official access in the middle of January after 2 months of communication with the authors. This considerably delayed my ability to start the implementation of the proposed system.

In hindsight, I should have set my goals earlier and evaluated them more thoroughly at the beginning of the project. I should have considered all potential risks and limitations of time and hardware to avoid situations where I realised that some of my goals were impossible to reach. Additionally, I spent a disproportionate amount of time working on the small details of the implementation. This time I should have spent on writing the final report.

A.2 Personal reflection and lessons learned

Overall, the aim of developing a drowning detection system has been achieved. I'm satisfied with the results, even though the classification accuracies were lower than expected. I have always had an interest in machine learning, especially in the computer vision field, and this project allowed me to explore the subject in more detail. By working on this project, I gained a deeper understanding of machine learning in computer vision systems. I enjoyed working with state-of-the-art deep learning methods and learned many valuable lessons.

Through the development of the system, I learned how to utilise popular deep-learning tools. Python machine-learning libraries such as PyTorch or Sklearn were essential in the development and evaluation of my neural network models. Hyperparameter optimisation with the Optuna library considerably improved the performance of these models. This knowledge will be useful in my future projects and can possibly be used in my future job if I ever become an AI/ML developer.

I also understood the importance of background research and well-written documentation. Without a throughout background research, I would never be able to understand the inner workings of neural networks. If it were not for the research papers and online guides, I would not be able to develop a working drowning detection system. Tools such as google scholar or stack overflow proved very useful when searching for relevant information. I went even as far as answering my first question relating to YOLOv7 pose estimation on stack overflow.

Finally, this project showed me how important time management skills are. I have learned to break down each problem into smaller, more approachable tasks. This allowed me to plan out the next steps while still keeping some spare time for eventual emergencies. However, in any future projects, I will definitely start earlier with a better outline and evaluation of the things I want to achieve to alleviate the stress of an incoming deadline.

A.3 Legal, social, ethical, and professional issues

A.3.1 Legal issues

Regarding legal issues within this project, it was important to gain official permission to use the "Water Behavior" dataset. This dataset contained videos of simulated water activities by a professional team of swimming pool lifeguards [17]. Obtaining this data was essential for the training and evaluation of the proposed drowning detection system. I satisfied the legal requirement by signing a release agreement with the authors of the said dataset (release agreement in Appendix B). I followed the restrictions by not distributing the data, not showing more than 8 images and making sure to reference the authors in the body of the report.

Furthermore, the YOLOv7 pose estimation algorithm used in this project is open-source and legal for me to use under the GNU General Public License v3.0 [116].

A.3.2 Social issues

Using the drowning detection system in a real swimming pool environment can potentially reduce the vigilance of lifeguards by creating a false sense of security. Human observers could get used to the system making accurate predictions and alerting them in case of emergency, leading to losing attention to the events occurring in the water. This could potentially endanger lives in case of a system emergency or any unforeseen circumstance. This issue can be solved by constantly reminding the users that the developed system is there only to aid them in their jobs and not replace them.

A.3.3 Ethical issues

The main ethical issue regarding this project is privacy. The data set used in this project contained videos of professional lifeguards simulating water-related activities. All of the individuals agreed to use their images for research purposes by researchers who signed the release agreement. As described in section A.3.1, I fulfilled this requirement.

Privacy concerns might also arise if the system were to be employed in a real-life environment. The proposed drowning detection system requires CCTV-like monitoring of the swimming pool in order to analyse and classify the behaviours of people in the water. This might lead to some concerns regarding the usage and storage of data. However, by design, the system does not store any data. After extracting the keypoints with YOLOv7 and displaying the classification result, frames are discarded.

Lastly, a major issue in machine learning systems is algorithmic bias and fairness. This issue arises when artificial intelligence systems do not treat everyone fairly or perform worse when used by certain groups of people. Even though the training data for the proposed drowning detection system consisted only of young male adults of Middle Eastern nationality, there should not be any bias caused by the ethnicities of the users. The neural network models were trained solely on the data from YOLOv7 pose estimation inference, which did not include any information about the nationality of the subject. The YOLOv7 algorithm should also be free of bias as it was trained on a diverse MS COCO 2017 dataset [22].

A.3.4 Professional issues

The drowning detection system has been developed in accordance with the British Computer Society (BCS) code of conduct [117]. The code is well-commented with descriptive variables and function names. Version control was done using Git and GitHub to keep track of changes made throughout the project. Feedback provided by the project supervisor and assessor was considered in order to deliver a high-quality final product.

Appendix B - External Materials

External Pose Estimation Algorithm used in the proposed system:

- YOLOv7-pose [36] [37]
- GitHub repository: https://github.com/WongKinYiu/yolov7

Guide I used to do pose estimation with yolov7:

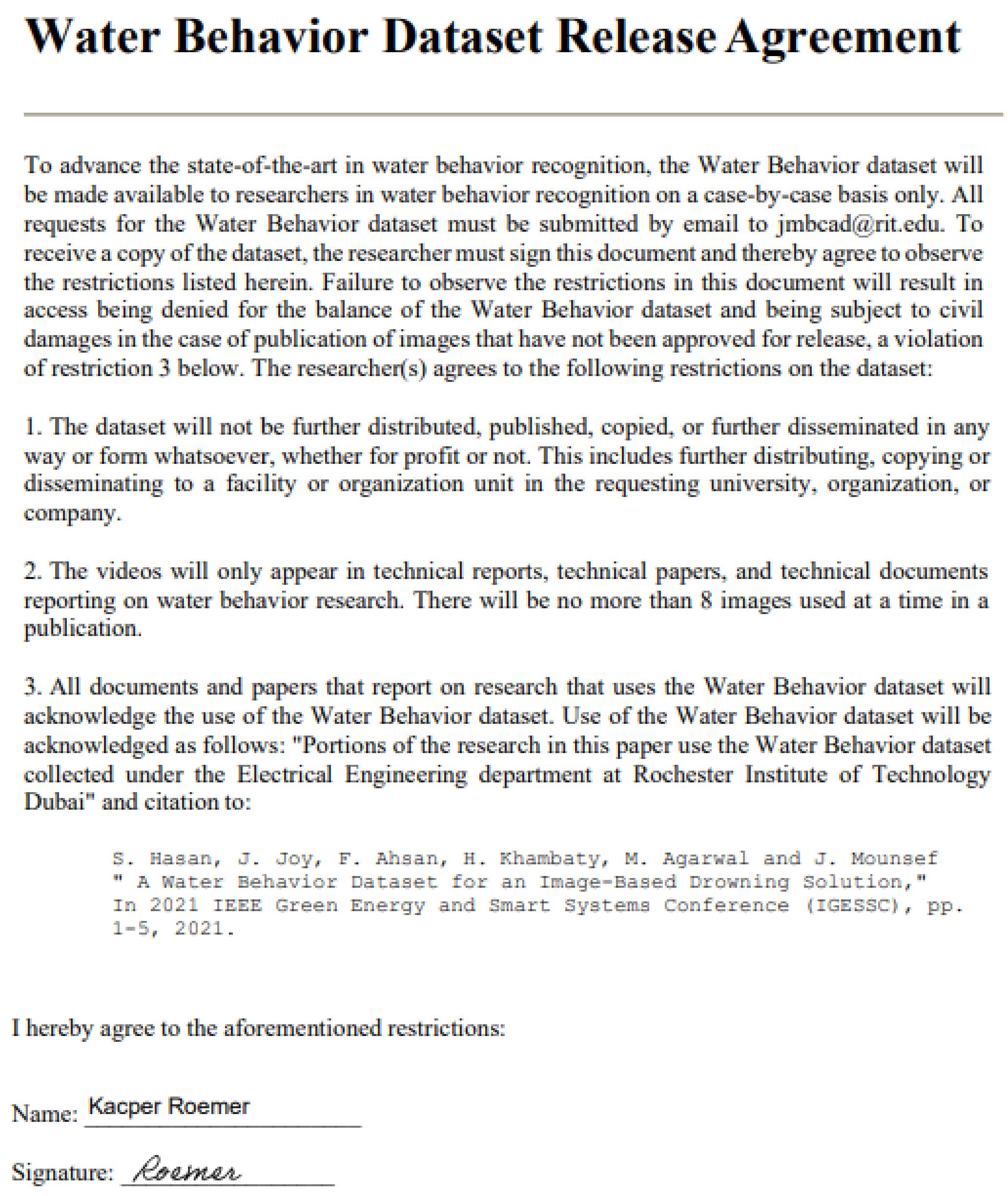
- https://stackabuse.com/real-time-pose-estimation-from-video-in-python-with-yolov7/

Optuna library used for Hyperparameter tuning [108]:

- https://optuna.org/

Dataset used for training and testing:

- A Water Behavior Dataset for an Image-Based Drowning Solution [17]


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: Kacper Roemer

Signature: Roemer

Figure 3.1: A “Water Behavior” dataset release agreement

Appendix C - Additional Content

C.1 Normalised Confusion matrices on single frames

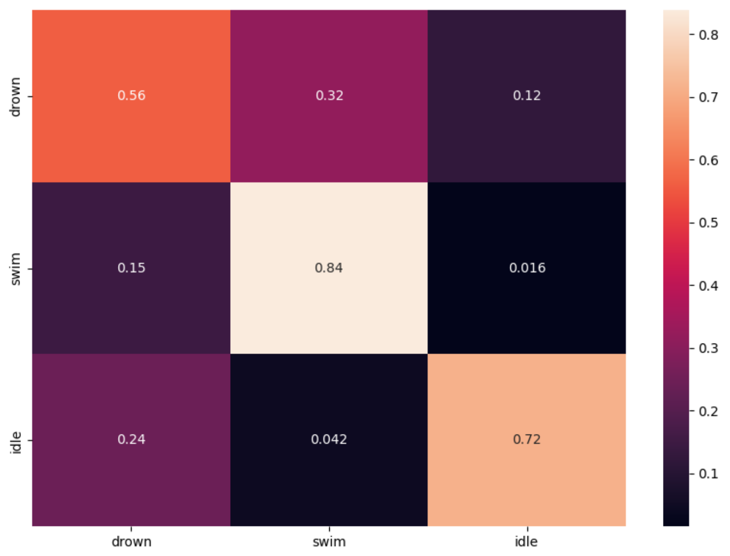


Fig C.1.1 - Normalised confusion matrix of the baseline model

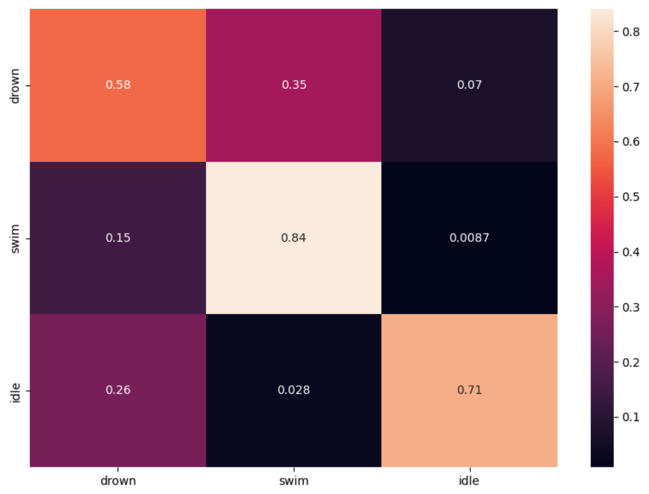


Fig C.1.2 - Normalised confusion matrix of the CNN-SINGLE model

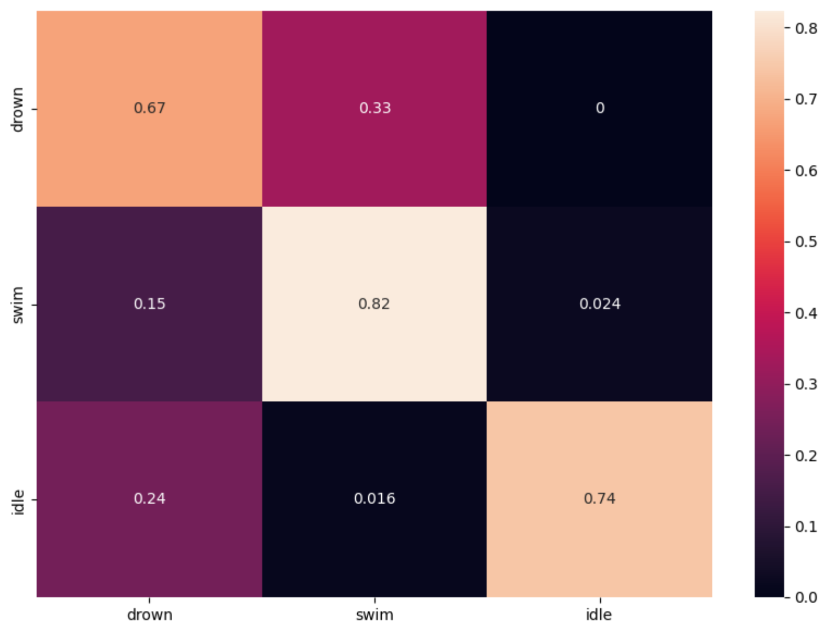


Fig C.1.3 - Normalised confusion matrix of the CNN-SEQ model

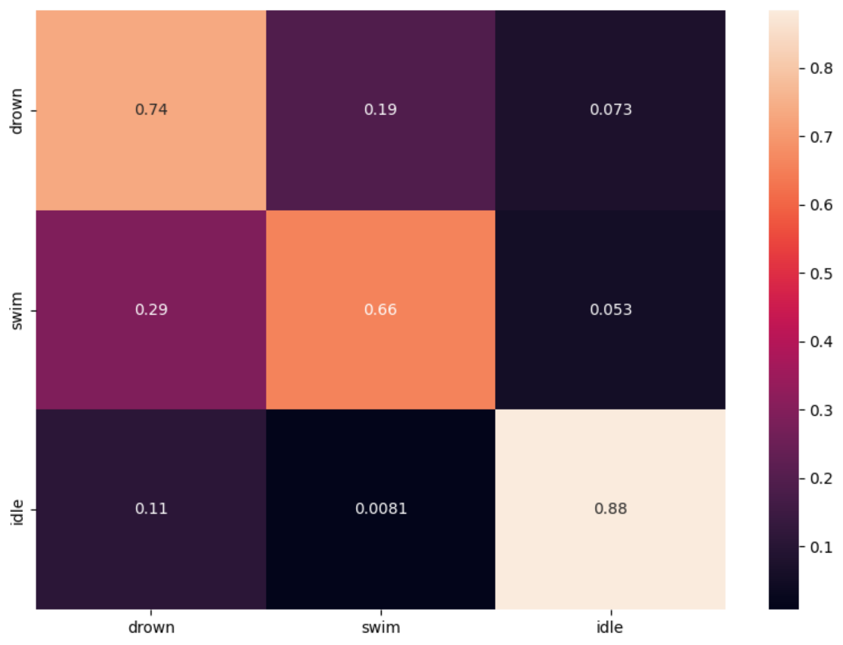


Fig C.1.4 - Normalised confusion matrix of the CNN-RNN model

C.2 Confusion Matrices on Entire Videos

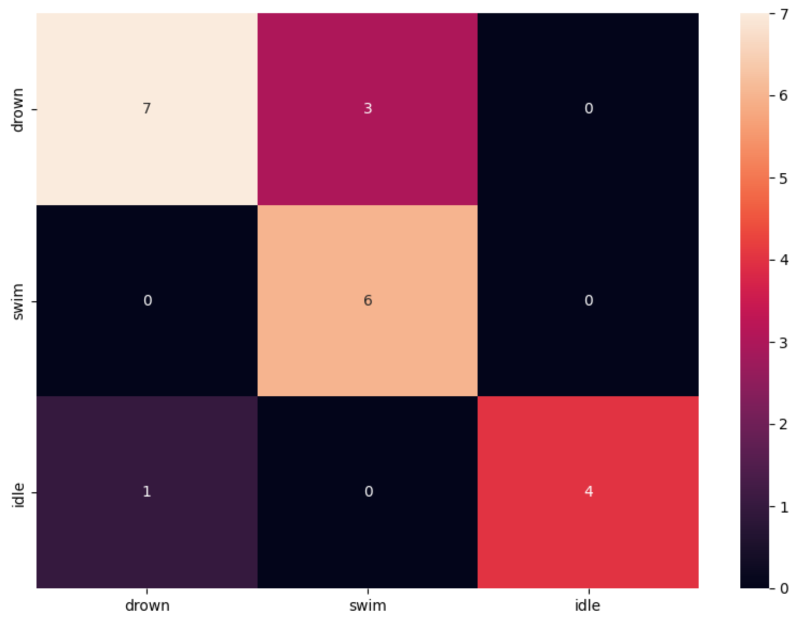


Fig C.2.1 - Confusion matrix of the baseline model

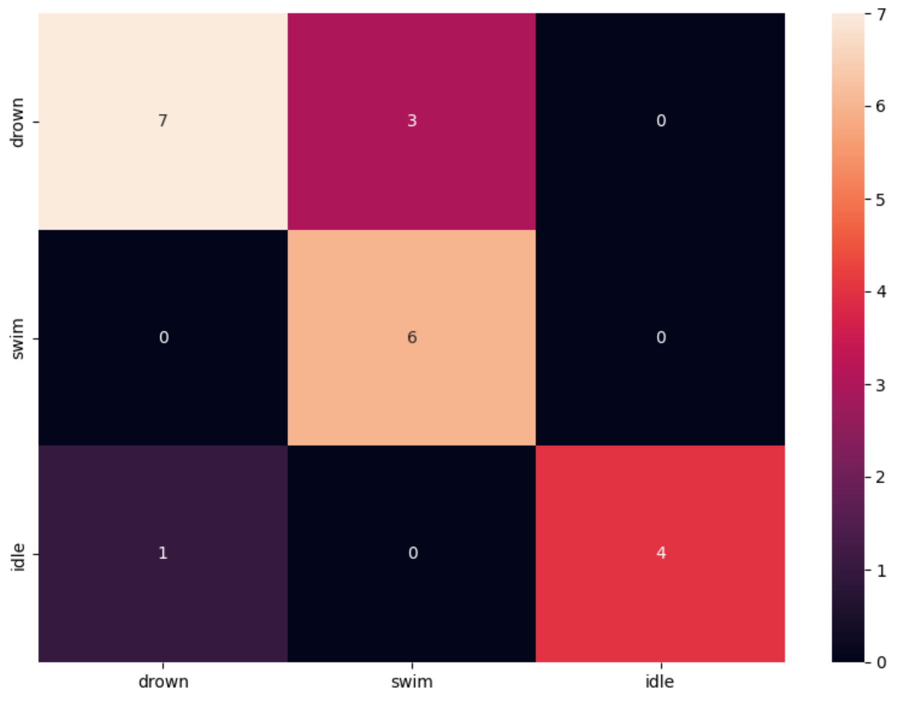


Fig C.2.2 - Confusion matrix of the CNN-SINGLE model

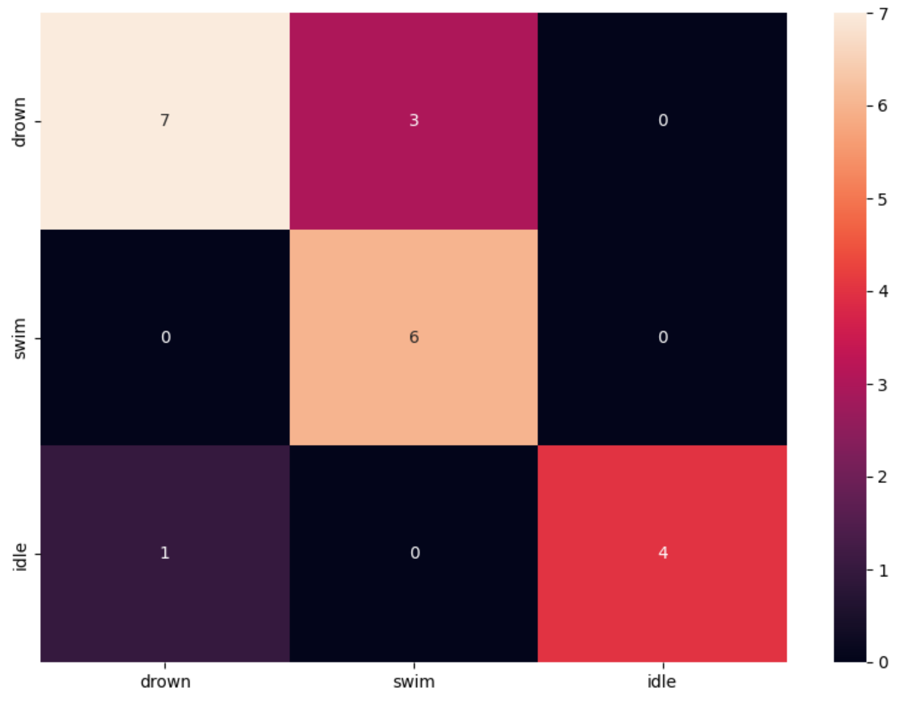


Fig C.2.3 - Confusion matrix of the CNN-SEQ model

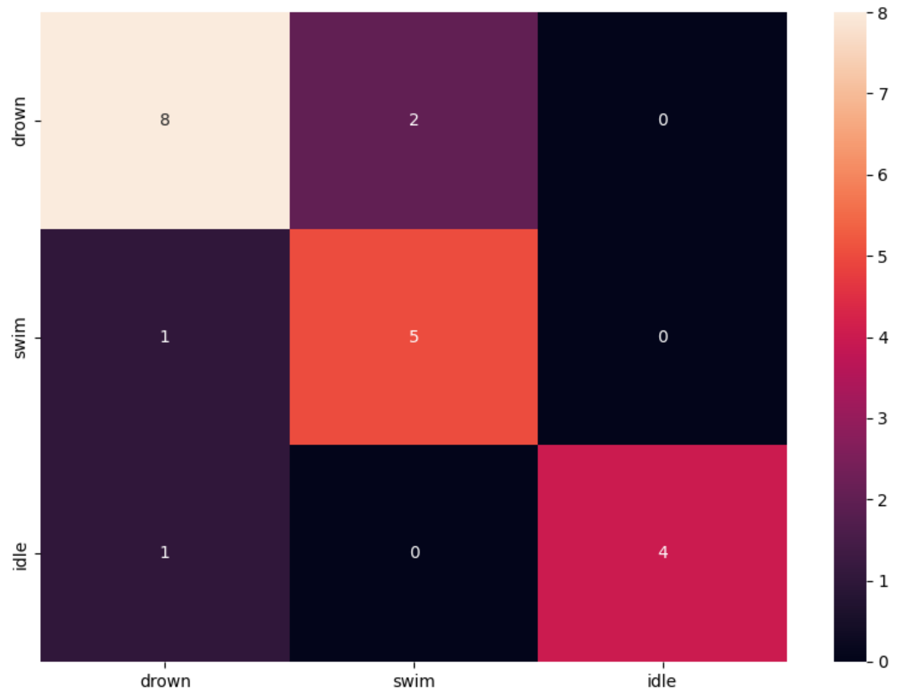
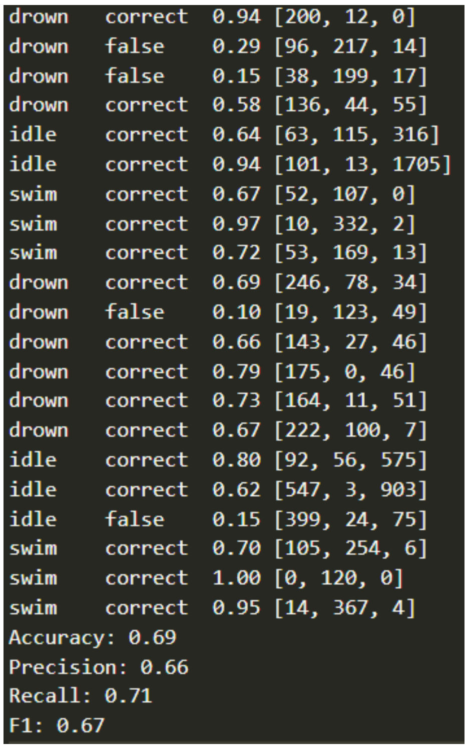


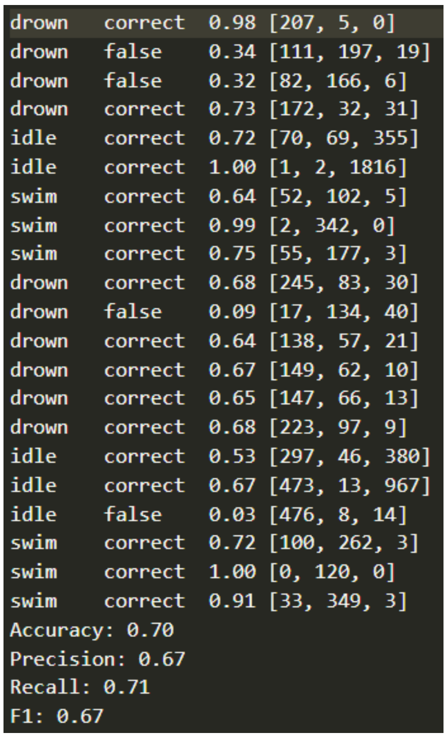
Fig C.2.4 - Confusion matrix of the CNN-RNN model

C.3 Testing logs



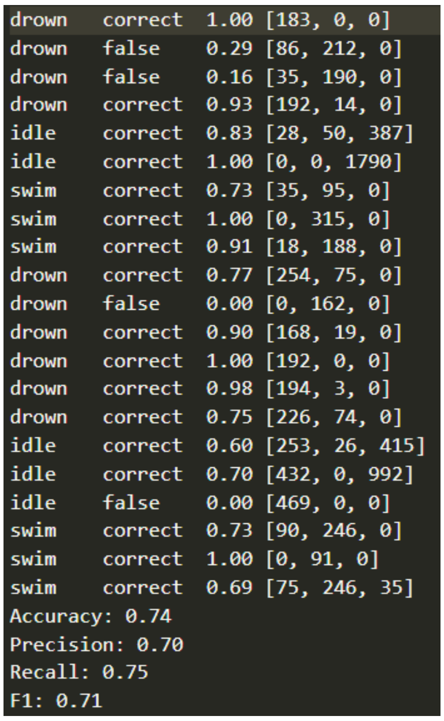
```
drown    correct  0.94 [200, 12, 0]
drown    false    0.29 [96, 217, 14]
drown    false    0.15 [38, 199, 17]
drown    correct  0.58 [136, 44, 55]
idle     correct  0.64 [63, 115, 316]
idle     correct  0.94 [101, 13, 1705]
swim     correct  0.67 [52, 107, 0]
swim     correct  0.97 [10, 332, 2]
swim     correct  0.72 [53, 169, 13]
drown    correct  0.69 [246, 78, 34]
drown    false    0.10 [19, 123, 49]
drown    correct  0.66 [143, 27, 46]
drown    correct  0.79 [175, 0, 46]
drown    correct  0.73 [164, 11, 51]
drown    correct  0.67 [222, 100, 7]
idle     correct  0.80 [92, 56, 575]
idle     correct  0.62 [547, 3, 903]
idle     false    0.15 [399, 24, 75]
swim     correct  0.70 [105, 254, 6]
swim     correct  1.00 [0, 120, 0]
swim     correct  0.95 [14, 367, 4]
Accuracy: 0.69
Precision: 0.66
Recall: 0.71
F1: 0.67
```

Figure C.3.1 Baseline testing logs



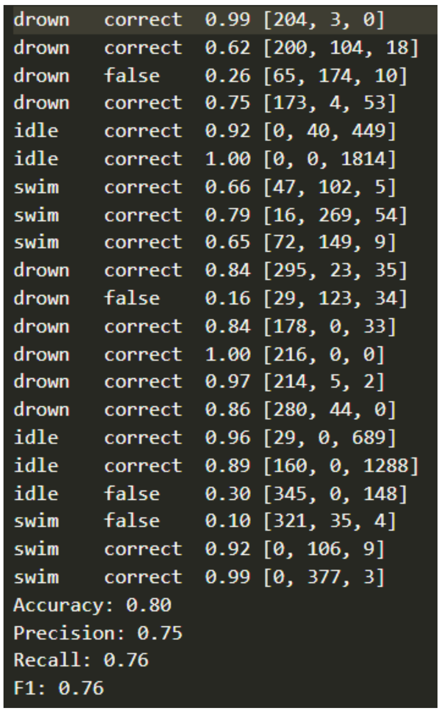
```
drown    correct  0.98 [207, 5, 0]
drown    false    0.34 [111, 197, 19]
drown    false    0.32 [82, 166, 6]
drown    correct  0.73 [172, 32, 31]
idle     correct  0.72 [70, 69, 355]
idle     correct  1.00 [1, 2, 1816]
swim     correct  0.64 [52, 102, 5]
swim     correct  0.99 [2, 342, 0]
swim     correct  0.75 [55, 177, 3]
drown    correct  0.68 [245, 83, 30]
drown    false    0.09 [17, 134, 40]
drown    correct  0.64 [138, 57, 21]
drown    correct  0.67 [149, 62, 10]
drown    correct  0.65 [147, 66, 13]
drown    correct  0.68 [223, 97, 9]
idle     correct  0.53 [297, 46, 380]
idle     correct  0.67 [473, 13, 967]
idle     false    0.03 [476, 8, 14]
swim     correct  0.72 [100, 262, 3]
swim     correct  1.00 [0, 120, 0]
swim     correct  0.91 [33, 349, 3]
Accuracy: 0.70
Precision: 0.67
Recall: 0.71
F1: 0.67
```

Figure C.3.2 CNN-SINGLE testing logs



```
drown    correct  1.00 [183, 0, 0]
drown    false    0.29 [86, 212, 0]
drown    false    0.16 [35, 190, 0]
drown    correct  0.93 [192, 14, 0]
idle     correct  0.83 [28, 50, 387]
idle     correct  1.00 [0, 0, 1790]
swim     correct  0.73 [35, 95, 0]
swim     correct  1.00 [0, 315, 0]
swim     correct  0.91 [18, 188, 0]
drown    correct  0.77 [254, 75, 0]
drown    false    0.00 [0, 162, 0]
drown    correct  0.90 [168, 19, 0]
drown    correct  1.00 [192, 0, 0]
drown    correct  0.98 [194, 3, 0]
drown    correct  0.75 [226, 74, 0]
idle     correct  0.60 [253, 26, 415]
idle     correct  0.70 [432, 0, 992]
idle     false    0.00 [469, 0, 0]
swim     correct  0.73 [90, 246, 0]
swim     correct  1.00 [0, 91, 0]
swim     correct  0.69 [75, 246, 35]
Accuracy: 0.74
Precision: 0.70
Recall: 0.75
F1: 0.71
```

Figure C.3.3 CNN-SEQ testing logs



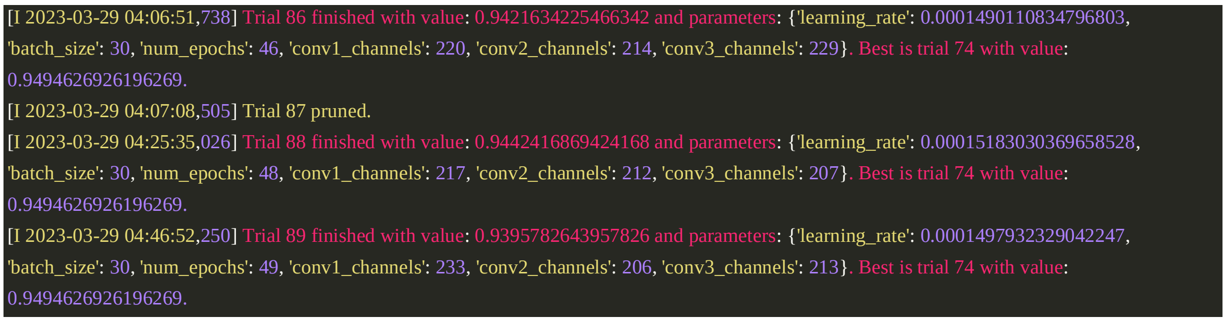
```
drown    correct  0.99 [204, 3, 0]
drown    correct  0.62 [200, 104, 18]
drown    false    0.26 [65, 174, 10]
drown    correct  0.75 [173, 4, 53]
idle     correct  0.92 [0, 40, 449]
idle     correct  1.00 [0, 0, 1814]
swim     correct  0.66 [47, 102, 5]
swim     correct  0.79 [16, 269, 54]
swim     correct  0.65 [72, 149, 9]
drown    correct  0.84 [295, 23, 35]
drown    false    0.16 [29, 123, 34]
drown    correct  0.84 [178, 0, 33]
drown    correct  1.00 [216, 0, 0]
drown    correct  0.97 [214, 5, 2]
drown    correct  0.86 [280, 44, 0]
idle     correct  0.96 [29, 0, 689]
idle     correct  0.89 [160, 0, 1288]
idle     false    0.30 [345, 0, 148]
swim     false    0.10 [321, 35, 4]
swim     correct  0.92 [0, 106, 9]
swim     correct  0.99 [0, 377, 3]
Accuracy: 0.80
Precision: 0.75
Recall: 0.76
F1: 0.76
```

Figure C.3.4: CNN-RNN testing logs

Appendix D - Hyperparameter optimisation with Optuna

To tune the hyperparameters of proposed neural network models, the following steps were taken:

1. The training set was further divided into a train set and validation set (ratio 80:20)
2. An objective function was created to train and validate the model.
3. Ranges for each of the hyperparameters to be optimised were defined inside the objective function using ‘trial.suggest_*’ methods from Optuna module.
4. Optuna study was created with the direction set to maximize the objective function.
5. Study was optimised for 100 trials for each model.
6. For every trial:
    a. An instance of a model was created with hyperparameters suggested by the trial.
    b. The model was trained using the suggested number of epochs and suggested batch size.
    c. The trained model was evaluated on the validation set after each epoch by calculating accuracy.
    d. The accuracy was reported to the trial using ‘trial.report’
    e. If necessary, the trial was pruned with ‘trial.should_prune()’.
7. After completing the optimisation, the values of ‘best_trial’ were saved.



```
[I 2023-03-29 04:06:51,738] Trial 86 finished with value: 0.9421634225466342 and parameters: {'learning_rate': 0.0001490110834796803,
'batch_size': 30, 'num_epochs': 46, 'conv1_channels': 220, 'conv2_channels': 214, 'conv3_channels': 229}. Best is trial 74 with value:
0.9494626926196269.
[I 2023-03-29 04:07:08,505] Trial 87 pruned.
[I 2023-03-29 04:25:35,026] Trial 88 finished with value: 0.9442416869424168 and parameters: {'learning_rate': 0.00015183030369658528,
'batch_size': 30, 'num_epochs': 48, 'conv1_channels': 217, 'conv2_channels': 212, 'conv3_channels': 207}. Best is trial 74 with value:
0.9494626926196269.
[I 2023-03-29 04:46:52,250] Trial 89 finished with value: 0.9395782643957826 and parameters: {'learning_rate': 0.0001497932329042247,
'batch_size': 30, 'num_epochs': 49, 'conv1_channels': 233, 'conv2_channels': 206, 'conv3_channels': 213}. Best is trial 74 with value:
0.9494626926196269.
```

Figure D.1: An example of console output during hyperparameter optimisation with Optuna