2022

A STEP TOWARDS AUTOMATING REAL-TIME COLLECTION OF ECOLOGICAL DATA ON OBSERVATIONAL PLATFORMS : A PILOT STUDY ON DEEP-SEA BENTHIC SPECIES Syringammina fragilissima

Browne, Erin

http://hdl.handle.net/10026.1/19507

http://dx.doi.org/10.24382/818
University of Plymouth

UNIVERSITY OF PLYMOUTH

A STEP TOWARDS AUTOMATING REAL-TIME COLLECTION OF ECOLOGICAL DATA ON OBSERVATIONAL PLATFORMS : A PILOT STUDY ON DEEP-SEA BENTHIC SPECIES *Syringammina fragilissima*

By

Erin Browne

A thesis submitted to the University of Plymouth in partial fulfilment for the degree

of

RESEARCH MASTERS

School of Biological and Marine Sciences

March 2022

Acknowledgements

From this work I would like to say a particular thank you to my primary supervisor Professor Kerry Howell. Kerry has gone above and beyond in supporting this work and put massive funds to getting me the required equipment via Mission Atlantic. This brings me on to thank Professor Louise Allcock who let me join the *Celtic Explorer* this summer to test the methods developed in a real-time scenario given scientist spaces were limited. A further thank you to the scientists and technical staff aboard the *Celtic Explorer* who helped deployment of the experiment and collection of data.

To achieve many of the technical programming steps outlined in the thesis I would like to extend my thanks to David Hutchinson, a key driver in making this work possible and whom I am very grateful for. For technical questions in terms of the functioning of the CNNs used and providing resources and guidance to understand how to best apply these analytical tools, I appreciate the insight knowledge given to me by my secondary supervisors Dr Chunxu Li and Dr Oliver Tills.

I am also grateful for the preparation of annotated datasets in order to train my CNNs which encompassed effort amongst many members of the deep-sea CRU past and present, but particularly Nicola Foster and Nils Piechaud. In addition, I would like to say a further thank you to Nils for his continuous advice and zoom calls during a very isolating masters year given the current circumstance of COVID-19. He has supported and motivated me and I am extremely grateful for all of his contributions.

Many thanks again to Professor Kerry Howell and David Hutchinson for reading and correcting my thesis draft.

Many thanks to contributors via online open-source support platforms and developers of key open-source packages in R, Python and C++ that are uncited.

Authors Declaration

At no time during the registration for the degree of Research Masters has the author been registered for any other University award without prior agreement of the Doctoral College Quality Sub-Committee.

Work submitted for this research degree at the University of Plymouth has not formed part of any other degree either at the University of Plymouth or at another establishment.

The following programme of advanced study included attendance of relevant scientific modules and seminars; presenting at conferences and a research cruise was undertaken to perform the methods described in the study.

Modules attended

- Postgraduate Research Skills and Methods (MAR 513)
- Artificial Vision and Deep Learning (AINT 515Z)

Cruise Participation

- RV *Celtic Explorer*, Resources of Rockall Bank and Fangorn Bank, July – August 2021, cruise reference CE21010, 3.5 weeks.

Conference Presentation

- Applied Deep Learning in Real-Time Analysis of ROV Livestream. The 16th International Deep Sea Biology Symposium (Brest - France), September 2021.

Word count of main body of thesis: 27,664

Signature: [illegible]

Date: 16/03/2021

Abstract

A STEP TOWARDS AUTOMATING REAL-TIME COLLECTION OF ECOLOGICAL DATA ON OBSERVATIONAL PLATFORMS : A PILOT STUDY ON DEEP-SEA BENTHIC SPECIES *Syringammina fragilissima*

Erin Browne

Despite current efforts to study deep-sea life, there is a dependency on technological advancements to asses it at a scale and pace required to inform effective management and conservation. Data collection platforms, such as Remotely Operated Vehicles (ROVs) are routinely applied to studying the deep sea due to their ability to collect large image-based datasets. Thus, data collection in the deep sea is becoming less of a problem in comparison to data interpretation, where terabytes of video data is collected during expeditions and manual interpretation of this is currently the standard procedure. Deep learning (DL), a sub-field of artificial intelligence (AI) has potential to address this particular issue thanks to its ability to analyse vast image-based datasets with minimal human interaction required, often exceeding efficiency, and a classification accuracy near-equivalent with human experts. This thesis investigates how DL, in particular Convolutional Neural Networks (CNNs), have progressed to the point of potential application in deep-sea research for detection and classification of organisms in image-based datasets. Different methodological approaches to training "off-the-shelf" CNNs on ROV datasets are assessed, with the aim to inform marine scientists, with little background in computer science, what steps and considerations are required when using CNNs for such tasks. In conjunction, a potential pipeline to perform real-time detection and classification during research expeditions is outlined. The research conducted in this thesis suggests CNN architectures perform differently given different training approaches and training image datasets, each with their own trade-offs. Maximum performances were achieved using the You Only Look Once (YOLO) version 3 architecture and a train from scratch (TS) approach, with improvements seen when using the pre-processed training image dataset. This gave 93% recall and

63% precision in detection of areas of presence-absence, and a strong correlation of estimated counts of *S. fragilissima* with manual counts (73%). Overall, results suggest that classifiers performance was mostly affected by the architecture type (version 3 and 4) and pre-processing steps chosen. However difference in the standard computer vision (CV) metrics assessed are minimal, meaning more simplistic approaches could be used, streamlining the procedure for non-experts. The pipeline for real-time detection and classification of *Syringammina fragilissima* on ROV livestream on-board a vessel performed efficiently at 25 frames per second (FPS) requiring no more than 12GB video RAM (VRAM) of a NVIDIA GeForce RTX 3090 Graphical Processing Unit (GPU); making it an achievable and cost-effective set-up for scientist on lower budgets. Despite these results it is noted that even the maximum performing classifier stills attains false positives and false negatives, meaning for reliable ecological metrics a degree of human intervention to check the data is required. This suggests that the described pipeline can achieve real-time detection on-board a vessel, however the training of the classifier impacts its performance. Thus, making the dataset used to train the CNNs integral in its performance. Studies in understanding the impact of pre-processing of training imagery datasets is a key area to focus on in the future regarding improvements to “off-the-shelf” CNNs as these are more user-friendly to implement than having to design a personalised CNN architecture. This provides a stepping stone for most non-experts in using such advanced analytical tools, and could lead to major increases in data availability for conservation and management of the deep seas.

Contents

List of Tables

List of Figures

the magnitude the classifiers under or over predicts the number of *S. fragillisima* on average over the sample size.

Figure 9| Examples of V3TS classifiers detections for false positives (A and B) and true positives (C and D).

Glossary

Image: An image in this context consists of a matrix of pixels pertaining to a vector of three colour channels; red, green and blue.

Segmentation: This is a computer vision term for individually labelling each pixel in an image or a group of pixels in an image.

Annotation: In the field of computer vision this simply means that an individual, usually a human expert, labels an area within the image into a class or group that object pertains to. This can be done usually by labelling individual pixels (segmentation) or using bounding boxes around the object.

Classification/ Identification: Both words in the context of computer vision mean to assign a group of pixels in an image (i.e. an object) to a class that the classifier has been trained on. In the context of ecologists this is interpreted as a taxonomic grouping, however in computer vision this doesn't necessarily have to be a strict taxonomic grouping it can be more vague, e.g. crab.

Detection: When a classifier predicts an object that it is trained on occurs within the unseen images or videos.

Regions of interest (RoIs): An area within an image that a classifier indicates has some similarity in terms of features pertaining to the data it was trained on.

Training data: In the context of deep learning and this study, training data pertains to a subset (usually 75-80%) of the total number of images annotated by experts, used to train the classifier.

Convolutional Neural Network (CNN): A type of deep neural network that utilises convolutional layers to extract feature maps of useful information to be passed to the fully connected layer. By doing so its reduces computational requirements by removing redundant features much faster, which is particularly relevant when dealing with image data.

Layer: Within a neural network a layer consists of an organised amount of small individual units called nodes or neurons e.g. input layer, output layer or hidden layer. In CNNs layers also mean where filters are applied to the original image or to feature maps deeper in the CNN architecture.

Architecture: Order of mathematical processes that occur within a neural network upon the input data (image or video) in order to extract information.

Node: In the context of machine learning these nodes represents the function of neurons in the human brain and often occur in layers of more than one within the hidden layers of a network. Here data annotated from the training images are inputted, transformed and saved at the weights alongside a bias value and then outputted to the next layers of nodes.

Class imbalance: This refers to the number of training images assigned per class not being even, i.e. the classifier does not receive an even representation of each class based on the quantity of examples it was given to learn off.

Classifier: In the context of this study a classifier refers to a CNN used to classify objects in image or video data. In chapter 2 classifiers differ given different architectures, training approaches or pre-processing of training data.

Framework: Otherwise known as deep learning frameworks are the building blocks for designing, training and validating deep neural networks (e.g. CNNs) via high-level APIs. This can be interpreted as one large library containing many modules and libraries used in the process of deep learning.

Backbone: Often these form part of the framework, whereby most deep learning frameworks use a feature extractor (backbone) and object detector (head), DarkNet-53 is the backbone associated with the Darknet framework and YOLO is the head.

Localisation: Obtaining a location within an image pixel matrix of a target object (i.e. class).

Epoch/training cycle: These are used intermittently but have the same meaning, whereby 1 epoch or 1 training cycle is when the classifier has done one complete cycle through all the training images to extract information. It does this repeatedly until it is trained.

Augmentation: Enlarging the size of a training image dataset by processing (e.g. rotating, brightening, flipping etc.) the original images (i.e. pre-processed) and including them in the training process of the classifier with the originals.

Chapter 1: Literature review – Progression of Artificial Intelligence sub-field - Deep Learning for driving automation of detection and classification of organisms in marine science

1.1 Evolution of Machine Learning to Deep Learning: *object detection and classification*

Traditional machine learning is a subfield of Artificial Intelligence (AI) in which machines are taught to parse data, learn from it and then apply what is learned to make intuitive decisions on a given goal (LeCun *et al*., 1998) using a set of algorithms. Initial steps in the development of machine learning began with training a machine to interpret data (e.g. image-based) using an architecture known as an Artificial Neural Network (ANN) (McCulloch & Pitts, 1943). This architecture was inspired by the structure and function within the human brain; where neurons pass on information between each other to visualise and associate objects with a name (i.e. 'class' in AI, or 'identification' in biology). ANNs replicate this via a network of nodes whereby the input image-based data contains information at the pixel level. That is then passed to the next layer of nodes in order to build a picture of what is occurring in the image. The classifier does this by interpreting these pixels into groups of features based on similarity (e.g. colour change, edges or corners) that could help associate that object with a class. Whilst this structure in theory could be an alternative to humans manually analysing problems, the progression toward its implementation in practice has seen many challenges. An initial approach developed by McCulloch & Pitts (1943) requires the user to hard-code features associated with a class to be learned from the data. This is a problem due to the skillset and time needed to produce these.

The single-layer perceptron network by Rosenblatt (1958, 1961) allowed the network to self-learn the features of each class from training data. In the perceptron (Figure 1), the

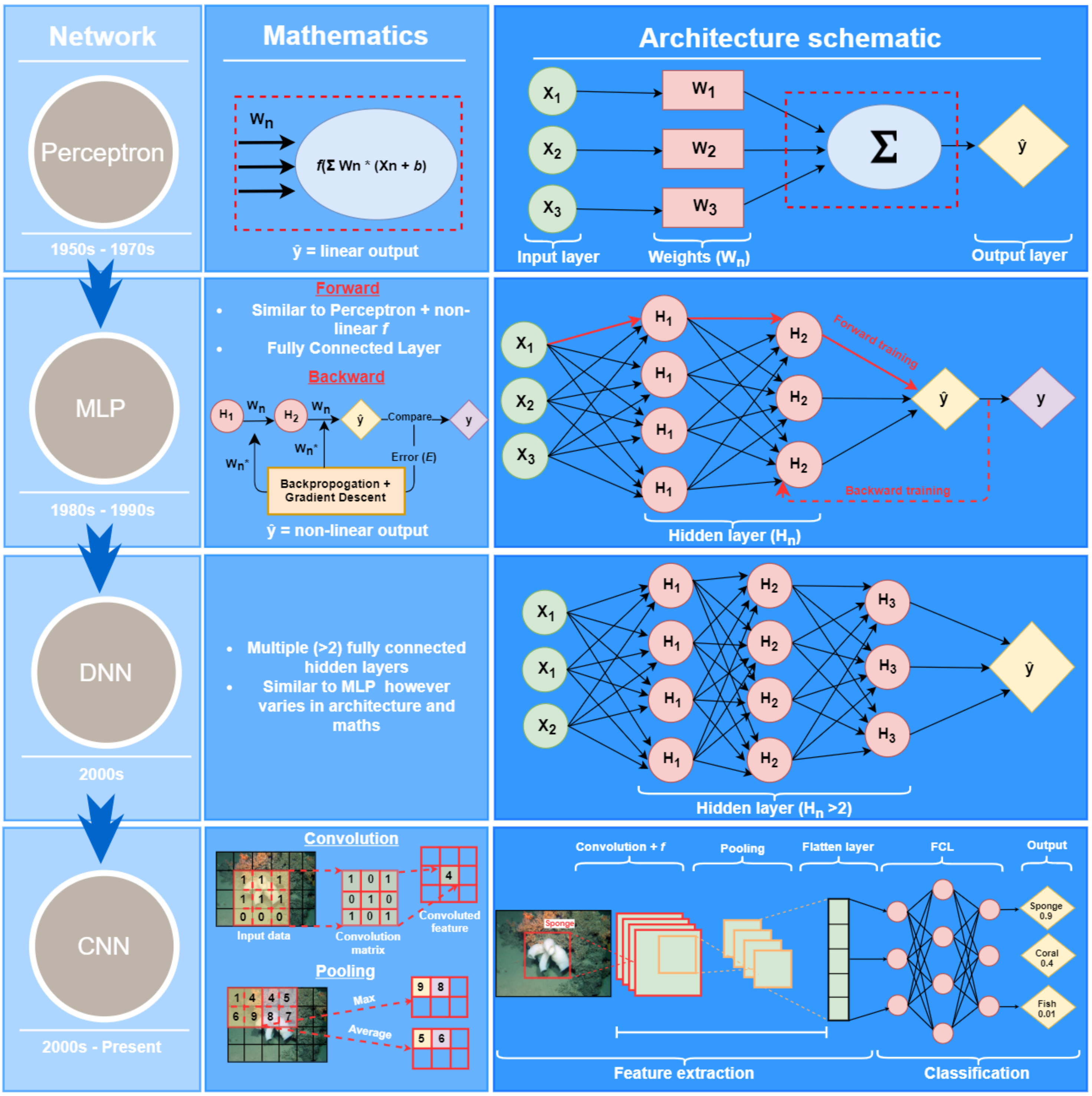


Figure 1| Visual interpretation of the evolution of traditional machine learning algorithms to Deep Learning (DL) networks used for object detection and classification, in terms of mathematics and architecture.

parameters (or features) learnt from the input layer of nodes are shared to the output layer, and each node connection has an associated weight (W_n) where parameters are learned. These weights are multiplied and summed per layer with the input nodes (X_i) alongside a predefined bias (b) term, known as the activation threshold. If this threshold is met, an activation function (e.g. step function, f) is initiated and the networks predictions are compared with the expected prediction, here an error between the two is calculated and the user can update the weights and bias to improve the networks performance (Figure 1). However, the application of the perceptron network in real-life problems (i.e. usually non-linear) was restricted to linear problems (i.e. input X_n features and the label y is linearly related, Figure 1) due to its famous failing in the XOR problem (Minsky and Papert, 1969). Meaning its usage across diverse image-based data was restricted.

The development of the Multi-Layer feedforward Perceptron network (MLP) between the mid-1980s to early 1990s reinvigorated the progression of ANNs in solving non-linear classification problems (Rumelhart *et al*., 1988; Werbos 1974, 1990; LeCun *et al*., 1998). Essentially this involved additional layers of perceptron's in the architecture (Figure 1), forming the hidden layers (H_n), or otherwise known as a Fully Connected Layer (FCL). In this layer the output for H_1 is the input for H_2 , where the mathematics remains the same as in the perceptron at each node, however it applies a non-linear activation function (e.g. Sigmoid, f) instead of the threshold activation function. This process is known as feedforward training. Again, the output from the hidden layer is passed to the output layers nodes where the predicted output ($\hat{y}$) is compared with the expected output (y). An error metric (E) is computed based on the fit between the predicted and expected. Unlike a perceptron, MLP utilises a 'supervised' learning technique for training called backpropagation, that takes the error and propagates it backwards through the network in order to calculate gradients, the gradients are calculated based on the loss function with respect to the weights (*W_n*) in the network, this is then inputted into an optimisation algorithm (e.g. Gradient Descent) to update the weights (*W_n**) and bias at each node; removing error in predictions in a self-training manner. Providing there is sufficient training data this allows MLP networks to deal with more complex non-linear problems.

Throughout the 1990s- early-2000s a new bottleneck emerged for ANNs; that being ANNs (i.e. MLPs) use one perceptron for each data input (i.e. pixel in an image) and this produces an unmanageable amounts of weights to be processed for large or complex images, thus computational restraints stagnated the usage of the ANNs to image-based analysis. Therefore, ANNs became less popular, whilst popularity of other subfields of AI grew, one being Computer Vision (CV).

The goal of CV is to mimic the human visual system. CV uses feature-based algorithms in order to detect and classify objects by applying mathematical filters (with user-inputted thresholds) over the raw input pixels in what is known as a sliding window, extracting features and separating them into classes based on their commonality (LeCun and Bengio, 1995; Goodfellow *et al*., 2016; Abroyan and Hakobyan, 2016). By applying these filters with an associated threshold to the image pixel matrix, pixels that meet the threshold are kept and pixels that do not can be removed from the input image. The pixels kept often pertain to features such as edges or corners. This process reduces the computational burden during feature extraction by removing large amounts of redundant pixels quickly. However, this approach solely cannot achieve the self-learning manner that ANNs can. In addition, feature extraction used to classify different groups still requires a degree of manually sorting, defining and fine-tunning based on a plethora of parameters, thus also necessitating the skills of a CV expert.

Since 2005, higher optimization of hardware and the development of Graphical Processing Units (GPUs), has facilitated ANN architectures to increase in complexity, allowing machines to process input data (e.g. image pixels) over more than two hidden layers, permitting larger more complex images to be analysed more autonomously (Bohte and Nguyen, 2016). These ANNs are known as Deep Neural Networks (DNNs; Figure 1), and have evolved into their own independent subfield known as Deep Learning (DL) (Hinton and Salakhutdinov, 2006; LeCun *et al*., 2015). Put simply, DNNs are mathematically similar to MLPs, however they perform calculations over extra hidden layers. This takes more processing power, time and more example training data (Shin *et al.*, 2016) in order to ensure the network does not overfit the predictions, creating bias outputs (Rice *et al.,* 2020). The DL subfield, and the diversity in network architecture, given its application, has exploded in recent years. Particularly in the application of object detection and classification tasks, with the most applicable neural networks being Convolutional Neural Networks (CNNs, Figure 1) (Voulodimos *et al*., 2018).

The potential of CNNs in object detection and classification problems came to light with the birth of AlexNet in 2012, achieving state-of-the-art performance in correctly labelling objects in the largest dataset still to be produced, ImageNet (Krizhevsky *et al*., 2012). This, resulted in CNNs becoming ubiquitous, which has subsequently led to the development of techniques which can organize, store, and analyse large volumes of data autonomously (Bohte and Nguyen, 2016; Christin *et al*., 2019). CNNs come in a variety of different architectures and, to an extent, have variations in their mathematical approaches. However, the fundamental processes within a CNN, compared to a DNN, occur in two stages 1) feature extraction and 2) classification. Within feature extraction three key mathematical processes happen, 1) convolution, 2) activation function and 3) pooling, before then being flattened into a single vector feature, and passed into the FCL for the classification stage (Figure 1).

At the beginning of a CNN a convolutional layer (or convolution matrix/filter) is applied over the image matrix in a 'sliding window'-like manner with thresholding (taken from the CV approach) to remove large amounts of redundant pixels and output a feature map (e.g. edge or point information). Multiple of these can be produced based on the filter count and are inputs into the next convolutional layer. A non-linear activation function (f) is applied to these feature maps, and if the features extracted match those in the training input data, the f is activated and those parameters are stored by the weights. That subset of the feature map is then passed onto the next layer in the CNN where the same process occurs, and this happens over a given number of convolutional layers, whilst retaining spatial context. This process differs from MLPs feedforward training, where all previous nodes are connected. By passing on only a subset of the feature map, the number of parameters to be learned from the input data is reduced, with further reductions through a technique known as parameter sharing. Lastly to note, convolutional layers may be manipulated for optimisation using a variety of hyper-parameters: filter size, output depth, stride and padding.

The pooling layers in a CNN occur successively between convolutional layers in order to reduce the spatial size of the input data or feature map for the next layer (removing redundant features). It does this by applying a filter (e.g. max or average pooling) on the feature maps from the convolutional layer, and reduces for example 4 pixels into 1 based on averaging them or taking the maximum value. This process is known as down-sampling, and results in faster training and controls overfitting. Aforementioned, higher-order features (flattened single vectors) are passed to one or more FCLs that compute probabilities or scores for each class labelled in the input data. This is used as the network output layer, whereby the predictions of the network are compared with the expected and, based on activation functions, a certain class label is given, alongside a probability or confidence score that represents how certain the network is of that prediction based on the features it has learned to associate with that class (Yamashita *et al*., 2018; Zhao *et al*., 2019). To conclude, CNNs offer a method to detect and classify objects in large sets of images without human intervention, and thus are end-to-end learners (Serbetci and Akgul, 2020). As a result of this design they have outperformed previous efforts in traditional machine learning and CV approaches.

1.2 Convolutional Neural Network: ***classifier trade-offs***

CNNs have brought significant advancements in object detection and classification over a plethora of fields, and this diversity in application has resulted in a diversity of CNN architectures. This is because there is no single robust CNN architecture that addresses all possible problems associated with a given task, such as variations in viewpoints, speed, multiple scales, occlusions, lighting conditions, background noises, size and exposure of the target object/organism, limited data and community structure (Mandal *et al*., 2018; Zhao *et al*., 2019). Each architecture deals with these issues through a series of different activation and loss functions, parameter optimisation, regularisation, and other architectural features (Khan *et al*., 2020). Thus, architectural variability comes with trade-offs in performance in detection and classification with different datasets for a given task, meaning consideration of these are required to choose the best candidates. Discussed below are the most up-to-date, relevant and well-performing CNNs that encompass both object detection and classification and with the potential for use in real-time applications.

Most modern classifiers are trained and tested on large, generic training datasets (e.g. ImageNet and COCO) (Deng *et al*., 2009; Bochkovskiy *et al*., 2020) at world-wide competitions (e.g. ImageNet Large Scale Visual Recognition Challenge; Russakovsky *et al*., 2015), and aim to localise the object in the image, alongside its classification, allowing for a fairer comparison in their performance. One of the most popular of these architectures, used for both object detection and classification, is the Regional-based CNN (R-CNN; Girshick *et al*., 2014),

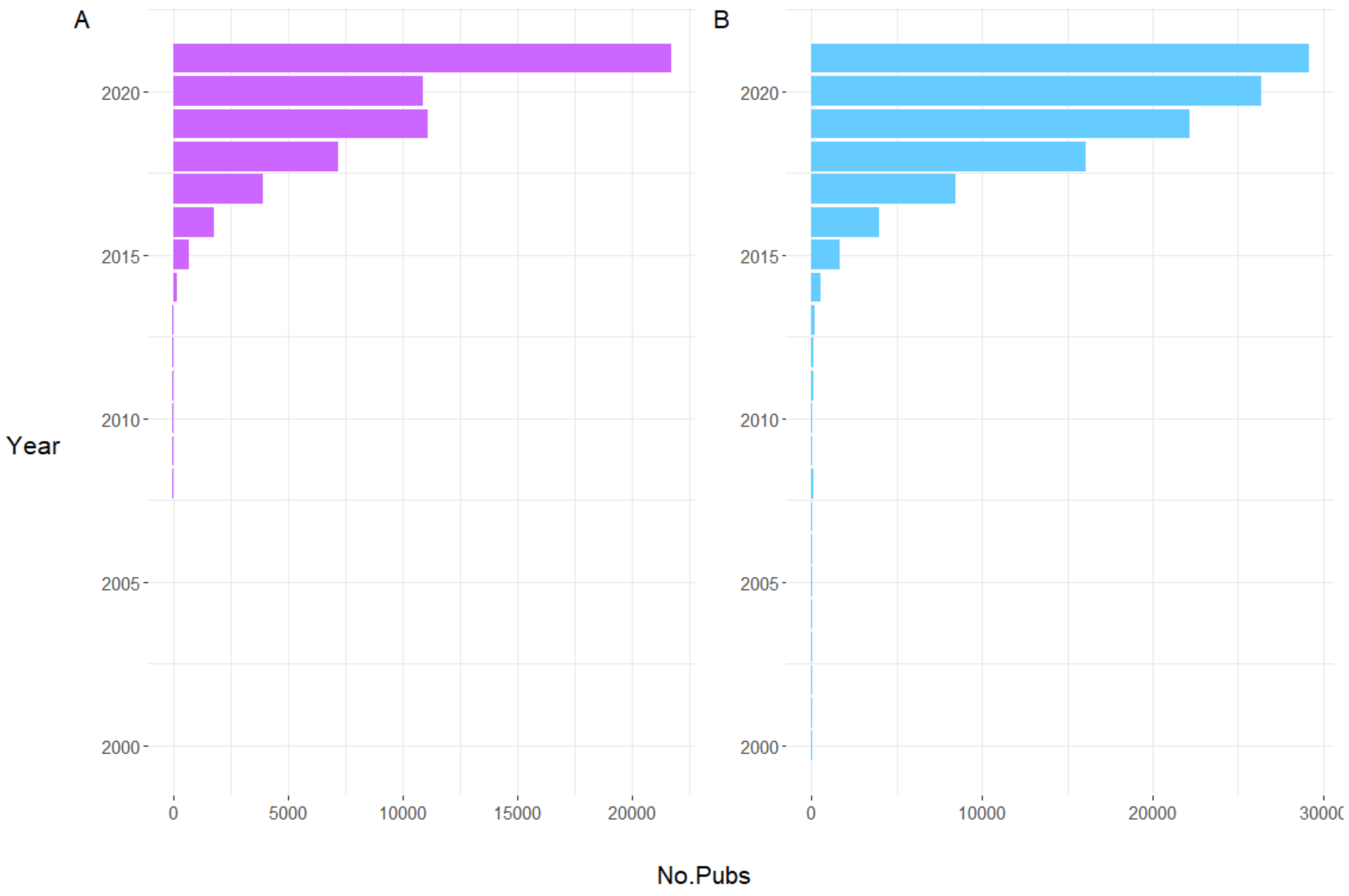


Figure 2| Number of publications on Google Scholar using search terms "CNN" and **A)** "*object detection*" or **B)** "*image classification*", indicating the boom in using CNNs across a multitude of fields for these tasks.

accounting for around 50% of object detection publications seen in Figure 2. This architecture combines Region Proposal Networks (RPNs) with CNN architecture. The RPN first decides the likelihood of object locations, producing multiple Regions of Interests (RoI). These are often warped into a uniform size, and RoIs are then passed individually through the CNN to refine object locations by extracting features. It can do this for multiple objects over various sizes in one given image (Montserrat *et al.*, 2017). R-CNN are therefore computationally complex, since every RoI is processed individually through the CNN as many times as its detected in the RPN, hence it can take much longer in order to train the CNN (e.g. 47 secs per RoI; Montserrat *et al.*, 2017) compared to architectures without RPN implemented in this manner.

The development of the Fast R-CNN (Girshick *et al.*, 2014) improved speed by first passing the input image to the CNN to generate a convolutional feature map, and from this using an RoI pooling layer and a bounding box regressor to create multiple RoIs per image. These are then passed through the CNNs FCL to generate RoI feature vectors, thus reducing the computational burden by avoiding passing each individual RoI through the series of convolutional layers in a CNNs architecture (Girshick, 2015). Following this, Faster R-CNN was produced by Ren *et al.* (2016) with the main goal being to speed up detection for real-time scenarios. This involved eliminating the selective search algorithm that was originally used to scan over a convolutional feature map to find out the RoI proposals, making the previous process slow and time-consuming. Instead Ren *et al.* (2016) utilise a confidence threshold in the FCL RoIs to ensure those of non-interest are discarded, yielding a reduction in processing time and thus improving its near-real time performance (Montserrat *et al.*, 2017).

Despite these improvements, Faster R-CNN cannot keep pace with real time video, which is typically shot at 24 frames per second (FPS), due to the two phase process involved (i.e. RPN and CNN) in its architecture, also known as a two-stage detector. In comparison, one-stage detectors are capable of undergoing detection and classification in one single pass over the input data, rather than using two networks combined. One of the most prominent one-stage detectors for object detection tasks is the You Only Look Once (YOLO) series of CNNs (Redmon *et al*., 2016). It is noted that the Faster R-CNN, and its later incarnations that attain more precision (Mask R-CNN and Cascade R-CNN; He *et al*., 2017 and Cai and Vasconcelos, 2018), outperformed the original YOLO architecture (Montserrat *et al*., 2017) in terms of accuracies in near-real time detection and classification on smaller objects. Mask R-CNN uses a pixel-level segmentation process to extract image features at a higher resolution, and Cascade R-CNN address issues related to overfitting and quality mismatch, in order to achieve this (Qiang *et al.,* 2020).

Now multiple one-staged detector incarnations of YOLO exist, most improving detection speed at the cost of accuracy (e.g. Fast YOLO, YOLO9000 and YOLOv3) (Shafiee *et al*., 2017; Redmon and Farhadi, 2017 and 2018). However, the most recent, YOLOv4, is aimed at striking a balance between speed and accuracy previously not addressed (Bochkovskiy *et al*., 2020); achieving a near 10% improvement in speed and accuracy in real-time performance (30 FPS) compared to YOLOv3 (Redmon and Farhadi, 2018), making it effective on most video datasets. On the other hand, other CNN architectures focus even more so on the accuracy of the detection and classification of the objects in more complex data (e.g. multiple objects of varying size in an image), but in doing so they compromise on speed due to the depth of their architecture often exceeding 100 layers (Ning *et al*., 2017; Khan *et al*., 2020). A few prominent series of architectures leading the field are Inception, introduced by Szegedy *et al*. (2014), which allowed the extraction of features over various spatial scales (small and larger objects), and ResNet (He *et al*., 2016) which is near equivalent in performance (McNeely-White *et al*., 2019).

The following examples are the most recent and relevant versions, and are easily compared due to being trained and tested on the ImageNet validating dataset, these include: InceptionV4 and a hybrid, Inception-ResNet-V2 (Szegedy *et al*., 2017), both of which succeeded the previous forerunner - Xception (Chollet, 2017). Both have similar, deeper architectures (i.e. more layers) and computational cost (Khan *et al*., 2020), and as a result they can achieve even higher detection and classification accuracies. This is largely a result of the introduction of specialized reduction blocks, which are used to change the width and height of the RoIs to ensure it is uniform (a requirement for CNNs), but instead this process is done in a manner that retains the width: height ratio of input RoIs so as to reduce the loss/distortion of smaller objects (Szegedy *et al*., 2017). However, Inception-ResNet-V2 did exceed accuracies at lower epoch, making it faster to train in practice when compared to InceptionV4 (Szegedy *et al*., 2017).

More recently ResNeXt-101 has provided improvement on the Inception Network (27 layers depth), albeit a similar architecture, it has more (101) and wider layers (Xie *et al*., 2017), allowing it to perform more complex tasks, alongside a major architectural introduction of cardinality (Szegedy *et al*., 2015). Previously, Inception networks used a common architectural property known as the split-transform-merge strategy, simply meaning it can perform like a large and dense layer CNN with considerably less computational complexity (Xie *et al*., 2017). However, this required the customisation of filter size and number in order to achieve high accuracy. Therefore, adapting the Inception architecture for new tasks means significant re-design of many factors and hyper-parameters for each filter (Xie *et al*., 2017). However, cardinality allows the ResNeXt-101 architecture to fix these parameters (i.e. no customising or re-designing) to a measurable dimension that is of central importance, in addition to width and depth of filters, removing the need to customise. Thus, reducing the labour involved in re-design per task. In fact it has been emphasized in the literature that increasing accuracy while maintaining or reducing complexity is rare, meaning this branch of particularly deep CNNs are important to consider (Xie *et al*., 2017). However, two main concerns observed with deeper and wider architectures are the high computational cost and memory requirement (Bianco *et al*., 2018). This makes it very challenging to deploy these state-of-the-art CNNs in resource-constrained environments and real-life scenarios (e.g. on livestream videos feeds; Khan *et al*., 2020), restricting the applicability of CNNs in low memory and time constrained applications (Wen *et al.,* 2020). Overall, this highlights the prominent trade-off when choosing a CNN-base architecture, that being the speed-accuracy trade-off (Huang *et al*., 2017; Li *et al*., 2019).

Based on these architectures and, using classifier comparison studies (Huang *et al*., 2017; Xie *et al*., 2017; Sanchez *et al*., 2020; Khan *et al*., 2020), a summary of the potential candidates best suited for object detection and classification in real-time, can be seen in Table 1. These CNNs are readily available online and utilise many different backbones (e.g. Darknet-53) whereby they can be re-trained from scratch (TS) or using transfer learning (TL). TL is when lower layers of the network are frozen (often associated with localisation) and the upper layers are re-trained on the users desired dataset. It is discussed in more detail later in the thesis.

Table 2| Summary of trade-offs between potential CNN architectures for object detection and classification. Where top-5 accuracy is the 5 highest probabilities that the CNN produces the expected answer, and real-time speed is represented by inference time (time in seconds for a trained classifier to process one image) on a conventional GPU (between 8-16GB VRAM).

Architectures (Year)	**Architecture Depth (No. of layers)**	**Top-5 accuracy (3sf)**	**Inference time/ secs (2sf)**	**Main trade-off**	**Reference**
Cascade R-CNN (2018)	Varies with CNN backbone (e.g. VGG16 is 16)	0.925	0.075	Deals with overfitting and mismatch to increase detection hypothesis ***vs.*** high accuracies only achieved if presented with high quality input data	Cai and Vasconcelos (2018)
YOLOv4 (2020)	9-53 (backbone dependant e.g. Tiny YOLO or DarkNet53)	0.952	≥0.015	Accuracy increases with depth of backbone (e.g. Darknet-53) ***vs.*** the cost of decreasing in speed	Bochkovskiy *et al*. (2020)
InceptionV4 (2016)	70	0.95	0.016	Deep hierarchies and multi-level feature representation ***vs.*** Slow in	Szegedy *et al*. (2017)

				learning (high epoch ≈ training number)	
ResNeXt-101 (2017)	101	0.966	0.016	Easier parameter customisation due to homogeneity in layers ***vs.*** High computational costs (very deep)	Xie *et al*. (2017)
Inception-ResNetV2 (2016)	572	0.953	0.064	Optimal accuracy ***vs.*** Slow and requires high memory	Szegedy *et al*. (2017)

1.3 Using Deep Learning for non-specialists: *progression and accessibility*

Despite development toward an intuitive method for faster detection and classification, attaining human-level accuracies still remains a challenge in the field of DL (Goodfellow *et al*., 2016). Related to this is the significant expertise required to create an appropriate architecture to improve accuracies. To address this computer science specialists have dedicated time and effort into building a variety of pre-made CNN architectures to address many scenarios (Sultana *et al*., 2018). The abundance of these now-available pre-made CNNs means those lacking in expertise can bypass the process of needing to build a CNN, instead they can simply re-train them using a dataset applicable to their problem-scenario (Rampasek and Goldenberg, 2016).

In order to re-train these pre-made networks, large, annotated (label and position of objects in an image) datasets are required, and creation of such datasets can often be tedious and time consuming due to the human effort needed to annotate these objects manually. However, annotation software (e.g. BioImage Indexing, Graphical Labelling and Exploration (BIIGLE; Ontrup *et al.*, 2009; Schoening *et al*., 2009), Yolo_Label (Yakovlev and Lisovychenko, 2020); VARS Annotation Assistance (Schlining and Stout, 2006); Marine Image Annotations (Schoening *et al*., 2016); CoralNet (Beijbom *et al*., 2015)), and generic large image datasets (e.g. ImageNet (Deng *et al*., 2009); COCO (Bochkovskiy *et al*., 2020); Pascal VOC (Everingham *et al.,* 2010)) have been produced by the computer science and ecology community to support the annotation and training process. Additionally, the production of such as Anaconda (Anaconda, 2016) and Docker containers (e.g. Singularity; Kurtzer *et al.*, 2017) has improved the ability to manage and utilise DL networks by packaging installations and code into virtual environments and containers, allowing for version control. Various Application Programming Interfaces (APIs) or DL training frameworks have been produced, from TensorFlow and Keras (Géron, 2019) to Darknet and Video and Image Analytics in Marine Environments (VIAME; Dawkins *et al*., 2018), all using multiple interactive languages (e.g. Python, C++). These frameworks package DL libraries into a single convenient program making it easier for the user to create a personalised project using their preferred programming language. In terms of writing, organising and storing code using these frameworks, many Integrated Development Environments (IDEs) have been produced (e.g. Spyder; Raybaut, 2009), over multiple languages with pre-written online code to support non-experts (Eglen *et al*., 2017).

In terms of training and testing DL networks two primary approaches can be taken, either TS or using TL (Shin *et al.,* 2016). Training a pre-made network from scratch simply means to retrain every layer from the feature extraction in the lower layers to the higher layers used for classification. It is argued that TS is more successful than TL in terms of object detection, as the DL network learns lower level features that are more applicable to the end-users data, often making it perform better (Xuhong *et al.,* 2018). However, it takes much more processing power to TS and can take longer to train (Shin *et al.,* 2016). In addition, the size of the training dataset and distribution over the number of classes must be carefully considered in order to prevent training a biased network (Hensman and Masko, 2015; Langenkämper *et al.,* 2018; Durden *et al.,* 2021). On the other hand, TL uses a pre-trained CNN architecture, built and trained originally on a large generic dataset and repurposed into a detector or classifier capable of performing well on data on which it was not originally trained (e.g. Han *et al*., 2018; Hussain *et al*., 2018). This involves re-training the upper convolutional layers with a dataset subject to the users interest, whilst the lower layers remain the same, and are based on the original larger generic datasets (Montserrat *et al*., 2017). The development of TL is a major contributor to the recent peak in popularity of CNNs over many disciplines for object detection and classification (Figure 2), as is does not require high-end hardware (< 8GB VRAM GPU) (Hastie *et al*., 2009; Ghahramani, 2015; Rajamaran *et al*., 2018) and can be performed on online free software (e.g. Google Collab, Bisong, 2019).

Improving the accessibility of DL as an analytical tool for non-specialists is important. However, using CNNs in object detection and classification tasks that are specific to end-users' needs requires tailored made annotated training images to achieve high performance. Manual annotation of training datasets is often still required in fields that rely on experts in order to detect and classify useful information from image and video data. In marine ecology recent awareness in the benefits of using CNNs in order create an autonomous approach for processing large amounts of image and video-data has sparked initial efforts in amassing large annotated training datasets applicable to the ecosystem surveyed (e.g. Christin *et al*., 2019; Boulais *et al*., 2020). In addition there have been efforts to develop standard reference image libraries to support consistent classification and annotation of image and video-data between human observers, in order to improve the quality of training datasets (Howell *et al*., 2019). The integration of DL into non-expert fields shows real promise in automating data processing of complex image and video data, with CNN architectures constantly being optimised, and annotated training datasets becoming more accessible for specific fields (e.g. medicine, ecology, security). In addition, interdisciplinary research furthers the understanding of how to utilise and improve these advanced analytical tools for specific purposes.

1.4 Application of Deep Learning in Marine Ecology: *object detection and classification*

Interest in utilising CNNs for ecological sciences has boomed with regards to processing ecological survey data more consistently and efficiently (Hampton *et al*., 2013;Diesing, 2016; Christin *et al*., 2019). For example, organisms could be found, counted and studied either in the laboratory or natural environments, at a speed greater than human analysis (Weinstein, 2018). The desire to speed up the analysis process is driven by the exponential increase in rate and quantity of digital data (e.g. images or videos) now collected and stored from expeditions and experiments (Dunbabin and Marques, 2012). Human analysis alone leaves a huge deficit in the ability to extract key information required to support management of ecological systems. Camera systems are often used as a method for studying marine ecosystems (Durden *et al.,* 2016a). They can collect masses of digital data, and analysis of such data has become a major bottleneck in marine ecology, causing delays in provision of up-to-date information (e.g. species abundance, distribution and biodiversity) on the health and services of our ecosystems and their response to change (e.g. climate, food, hunting, fishing). Utilisation of DL for detection and classification of ecological data has grown in recent years, accounting for 4.5-6% of the increase seen in Figure 2 (using the additional search term "marine ecology"). These techniques have the potential to address this major bottleneck in marine ecological science (Goodwin *et al.,* 2021).

Some initial restrictions seen in using CNNs for this application are already being addressed by the marine ecology community. For example, standardised classification public databases (Howell *et al*., 2019) for labelling organisms are in development and large expert annotated training datasets (e.g. CoralNet, Beijbom *et al*., 2015; FathomNet, Katija *et al*., 2021a) to improve the features learned during training. This coincides with advancements in architecture design (Goodwin *et al.,* 2021), alongside well-documented research in dealing with other problems such as class imbalance (Moniruzzaman *et al*., 2017 ; Langenkämper *et al.,* 2018; Durden *et al.,* 2021), tracking and monitoring (Jäger *et al*., 2017; Mandal et al., 2018; Katija et al., 2021b) and multi-class detection (Goodwin *et al.,* 2021; Liu *et al.,* 2021). These studies have noted instances yielding accuracy levels sometimes surpassing humans (e.g. Table 2).

Table 2| Examples of approaches used to train and improve different CNN architectures for objective detection and classification on four major target groups in marine ecology 2014-present.

Target group	**Methodological approaches**	**CNN architectures**	**Range of accuracy achieved**	**Author (publication year)**

Fish	RGB colour space, sliding window for object tracking, TL to learn characteristic features (edges, pixel intensities), classification	Fast R-CNN, Soft max Classifier with Deep Network (CNN), R-CNN with a hierarchical parametric classifier, ResNet-152 network, YOLO classifier, Inception classifier, DeepFish architecture (CNN), YOLOv3, Cross-pooled FishNet.	~55.0-99.27%	Chatfield *et al*. (2014) Li et al. (2015) Villon *et al*. (2016) Salman *et al*. (2016) Qin *et al.* (2016) Shafait *et al.* (2016) Sung *et al*. (2017) Jäger *et al*. (2017) Rathi *et al*. (2017) Matabos *et al.* (2017) Villon *et al*. (2018) Siddiqui *et al*. (2018) Salmon *et al*. (2019) Raza and Hong (2020) Mathur *et al*. (2020) Knausgård *et al*. (2021)
Plankton	Shapes and rotational symmetry, Multi scale Architecture, TL (e.g. reduce Class Imbalance, increase taxonomic identification,	ConvNNet inspired by OxfordNet Deep CNN inspired by GoogleNet CIFAR 10 CNN ZooPlanktoNet inspired by	~48.7%-94.8%	Py *et al*. (2016) Lee *et al*. (2016) Dai *et al*. (2016) Libreros *et al.* (2018) Mandal *et al*. (2018)

	asses fish abundance), Data Augmentation to increase the dataset	AlexNet and VGGNet, Faster R-CNN, Inception module.		Mitra *et al*. (2019)
Coral	Colour Shape Texture feature Descriptors, Texton and colour based handcrafted features Spatial Pyramid Pooling (SPP), Tensorflow Object Detection API	Supervised CNNs, VGGNet, Faster RCNN with NasNet, Inception V2, ResNet101, Mask R-CNN, ResNet60, VGG16 and 19.	~44.0-98.0%	Elawady (2015) Mahmood *et al*. (2016) Mahmood *et al*. (2017) Jaisakthi *et al*. (2019) Arendt *et al*. (2020) Picek *et al*. (2020) Lumini *et al*. (2020) Raphael *et al*. (2020)
Other benthic organisms	TL for taxonomic classification, Data augmentation to reduce class imbalance and increase dataset size, image segmentation	Inception V3 classifier, AlexNet, Google Inception classifiers, Mask R-CNN, CGG-16 and U-Net,	~67.0-95.0%	Marburg and Bigham 2016 (deep sea benthic macrofauna) Raitoharju *et al*. 2018 (freshwater invertebrates) Zurowietz *et al*. 2018 (deep sea benthic megafauna).

				macroinvertebrates) Langenkämper *et al.* 2018 (benthic megafauna) Piechaud *et al*. 2019 (deep sea benthic megafauna) Han *et al*. 2020 (sea cucumbers) Shashidhara *et al*. 2020 (deep sea worms) Durden *et al.* 2021 (benthic megafauna)

1.4.1 Fish ecology: *object detection and classification*

The field of fish ecology accounts for a significant proportion of the work already carried out in marine ecology using CNNs in detection and classification tasks (Figure 3). This is largely driven by the fishing sector in order to create transparency of practices and sustainability in the fishing industry (Munim *et al*., 2020; Probst, 2020). As a result, this has allowed them to address practical constraints found in previous methods for detection and classification (Villon *et al*., 2016). These include, classification of data with varying obstacles, for example using front illumination with backlight to identify fish in high turbidity environments (Zhang *et al.,* 2016; Shafait *et al.,* 2016), where light propagation is limited. Complex background noise, such as 3D complex habitats, occlusion and sediment type, are another major obstacle to wider use that is repeatedly addressed by different studies given the variability between each species being studied and, their study conditions (Shafait *et al.,* 2016; Liu *et al*., 2018; Wang *et al.,* 2022). One popular method used to address this is to pre-process the training dataset to highlight RoIs (Zhao *et al.,* 2018) to improve feature extraction. Variance in organism exposure, position and size, can affect CNN performance. There is a great deal of research that has been produced which addresses this issue, predominantly by varying angles and distance of the camera to target species to gather more contextualised training datasets

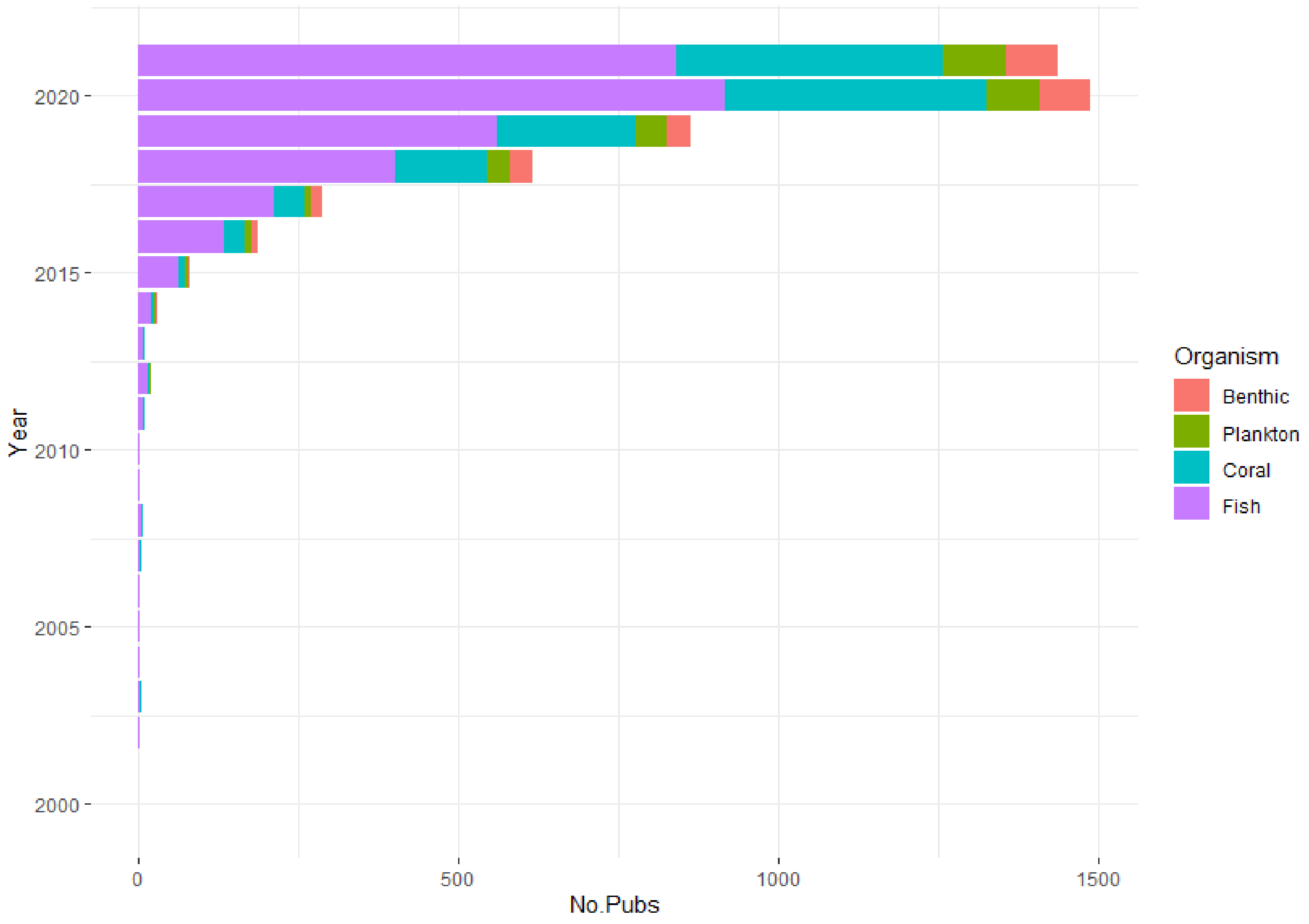


Figure 3| Number of publications on Google Scholar using search terms "CNN", "object detection" "classification", and "*benthic*" or "*coral*" or "*plankton*" or "*fish*", over the last two decades. Indicating the boom in using CNNs for such detection and classification tasks amongst the major marine ecology fields.

(Jäger *et al*., 2017; Rathi *et al*., 2017; Tharwat *et al.,* 2018; Salman *et al*., 2019). There is now evidence to suggest that real-time recognition of fish taxa (Qin *et al*., 2016; Sung *et al*., 2017; Salman *et al*., 2016; Mathur *et al*., 2020), and quantifying fish abundance from cameras deployed in the field is possible (Matabos *et al.,* 2017 ; Hong Khai *et al.,* 2022). Many of these advancements have been a result of open-source training databases being more species specific (e.g. Fish4Knowledge, Fisher *et al.,* 2016; NorFish, Crescitelli *et al.,* 2021) in order to compensate for accuracy lost at the higher taxonomic levels when detecting multiple classes, particularly with low morphological variability and highly complex backgrounds. From the methodological progression in applying CNNs to fish ecology, high performances have been achieved across the board. Ditria *et al.* (2020) achieved mean Average Precision (mAP) scores of 92.5-93.4% on the target species, *Girella tricuspidate*, with the incorporation of citizen science annotation contributions using both image and video datasets. Multi-class fish detection has also seen exceptional results with Knausgård *et al.* (2021) achieving 99.3% classification accuracy and, Raza and Hong (2020) 91.3%, over 4 different species. Whilst a study by Siddiqui *et al.* (2018) achieved 94.3% classification accuracy across 16 fish species by exploiting pre-trained CNNs on generic databases, and in doing so limited the amount of training images required per class (ranging from only 42-91 each). The classification accuracies obtained suggest that data processing using this method can obtain human level expertise (Culverhouse *et al.,* 2003; Schoening *et al.,* 2012; Durden *et al.,* 2016b). Pipelines to improve the ease of use and outcomes of this tool for fish ecologists are becoming more established (Li *et al.,* 2022).

1.4.2 Plankton ecology: *object detection and classification*

In plankton ecology, CNNs have been used to a lesser extent (Figure 3) for detecting and classifying species compared to fish ecology. One reason for this is the resolution required to accurately identify many taxonomic groupings, such as bacteria and picoeukaryotes (0.2-2μm) (Bureš *et al.,* 2021; Irisson *et al.,* 2022). The majority of studies on the classification of plankton in image datasets emerged post-2015 (Irisson *et al.,* 2022), with Luo *et al.* (2018) contributing the first publication achieving this using a CNN. In this study they attained an overall accuracy of 87%, although, the precision attained was only 55% over 38 broad taxa. Interestingly most CNNs have not displayed large increases in accuracy over the years (usually ranging between 70-80%), but accuracy has been demonstrated over a more diverse range of taxa (~100 classes) (Irisson *et al.,* 2022). Some studies identifying and enumerating plankton in *in-situ* planktonic imagery (providing a representation scene) with smaller numbers of taxa (7 classes), but with higher taxonomic resolution and more complex morphology, have achieved accuracies of up to 94.5%, on pre-trained ResNet50 CNN (Cheng *et al.,* 2019). This is particularly relevant when using such techniques in the field, where high levels of debris, flocs and microplastics found in water samples can cause misidentification of some plankton species, and human analysis still attains high error (73.7 – 75% accuracy) (Kelly *et al.,* 2002; First and Drake, 2012). A study by Libreros *et al.* (2018) on diatoms managed to utilise the symmetry, shape, geometry and texture of undesired elements (high turbidity caused by debris, flocs, etc.) and target species, with segmentation, to assign features relevant to each class. In doing so, the best accuracy attained was 99%, and similar, but more extensive (between phytoplankton, zooplankton and detritus, minerals etc.) results have been replicated by Rivas-Villar *et al.* (2021). Considering the performances attained, the application of real-time classification *in-situ* has more associated challenges compared to classification using microscopy in controlled laboratory conditions or on image datasets.

The vast spatial and temporal distribution of plankton further restricts the application of real-time classification in the field as collection methods have not yet been adapted for use with

AI, and robust onsite pipelines have not yet been fully established within the research community (Irisson *et al.,* 2022). However, this is beginning to be addressed, for example, Bergum *et al.* (2020) trained a CNN with a potential to average 66.6% precision (using Mask R-CNN) on the copepod class, *C. finmarchicus*, and created a pipeline viable for deployment on a lightweight autonomous underwater vehicle (LAUV). To achieve full automation in such a way requires the CNNs to have to perform in real-time, meaning that a whole transect, or time series of data can be analysed at once. This requires the CNN to be able to identify a target species over a number of frames consistently, in which the target species vary morphologically. Studies toward the full automation of classification are enabling us to understand how well a CNN would perform in this scenario. For example, Irisson *et al.* (2022) assessed this over a time-series equating to 850,000 images over 60 classes (detritus and plankton). The CNN attained an accuracy, mean precision, and recall of 67%, 69% and 78%, respectively, but the optimum performances tended to be associated with biological taxa that were abundant and distinctive in shape. In addition, detritus was still being detected as biological taxa. This further highlights that the production of large, balanced and diverse (e.g. lighting, occlusion, angles, texture, size) datasets over multiple planktonic taxa, and objects that can be mistaken for plankton, are key to improving CNN performance when applying it to image or video datasets. Contributions to open source training data being made by plankton ecologists (e.g. ZooScanNet, Elineau *et al.,* 2018; WHOI-Plankton dataset, Sosik *et al.,* 2021; Verhaegen *et al.,* 2021) are beginning to bring solutions to this issue, but further contributions are required.

1.4.3 Benthic ecology, shallow to deep: *object detection and classification*

Benthic ecology has seen nearly as much research as fish ecology, however the majority of studies are on shallow water corals (Figure 3), leaving significant knowledge gaps for automating classification of many other benthic marine habitats and species. In fact, reef corals alone are the second most researched marine organisms after fish when it comes to automating the analysis of large image and video datasets, with huge surges in relevant studies post-2017 (Figure 3). This is related to their known importance in terms of ecosystem service provision and widely documented vulnerability to anthropogenic stressors including climate change. . To-date, research on applying CNNs to coral detection and classification has achieved accuracies of up to 84-99% on imagery datasets (8-10 classes) (Lumini *et al*., 2020; Raphael *et al*., 2020). Classes are based largely on coral morphology (e.g. branching, boulders, texture) to distinguish down to fine-scale taxonomic level. In addition, most studies include algae as a class due to the close ecological relationship and proximity with coral. Studies have shown retention of these high accuracies on CNNs trained with up to 15 classes, and a degree of augmentation, in order to increase and balance training datasets for better network performance (Gómez-Ríos *et al.,* 2019; Jaisakthi *et al*., 2019; Picek *et al*., 2020). Research in coral ecology has dealt with a range of constraints in order to achieve the results seen today (Table 2). For example dealing with variations in illumination (Beijbom *et al.,* 2015; Arendt *et al*., 2020), image blur (Picek *et al*., 2020), class imbalance (Lumini *et al*., 2020), occlusion (due to corals being a complex 3D habitat) (Lopez-Vazquez *et al.,* 2020; Hopkinson *et al.,* 2020), variation in camera equipment (e.g. angle of view, resolution, distances) (Beijbom *et al.,* 2015; Hopkinson *et al.,* 2020) and, combinations of *in-situ* and processed (e.g. fluorescence images) imagery used in training (Beijbom *et al.,* 2016). Addressing these restraints has seen CNNs being applied to more elaborate tasks, such as quantifying key benthic substrata for coral reef monitoring (Christin *et al*., 2019), identify diseases on coral (Ani Brown Mary and Dharma, 2019) and identifying, then counting different coral species using a combination of CNNs (for identification) and trackers (for counting) in images on an autonomous underwater vehicle (AUV) (Modasshir *et al.,* 2018).

Studies across individual-based benthic fauna have seen more varying results, with accuracies on freshwater macroinvertebrates reaching 74-76% over 64 classes (Raitoharju *et al*., 2018), and in marine fauna, precisions of 70% for sea cucumbers (Han *et al*., 2020), 67% for deep sea worms (Shashidhara *et al*., 2020), 75% on 7 different deep sea taxa (e.g. xenophyophores, sponges, anemones) (Piechaud *et al.,* 2019) and 89% accuracy over 10 deep sea taxa (e.g. anemone, crab, coral, sea star) (Marburg and Bigham, 2016). A more recent study by Durden *et al.* (2021) looking at 25 different benthic invertebrate megafaunal classes, attained 94% accuracy, but in addition provided results in terms of ecological metrics (diversity and faunal composition), whilst also dealing with class imbalance. Framing the performances of CNN-based classifiers in ecological terms provides more context and usability of the generated output for non-computer scientists, such as ecologists. It is noted that the comparison of accuracies in detection and classification tasks across these two benthic fields requires consideration due to the fact corals tend to be large colonies covering vast areas compared to individual-based benthos. This brings inherent differences in what biological aims and metrics (e.g. area covered vs. individual counts) are being achieved and the practicalities involved in detecting and classifying between these two different genres. For example, individual-based benthic fauna tend to have more distinct classes but often more different morphologies, meaning classification and quantification of the same individual requires classifiers to be more robust to account for this (Schoening et al., 2012).

In the marine ecology community it is becoming more evident that the use of CNNs for automated detection and classification of organisms within image and video-based datasets is gaining traction. This is particularly relevant to areas of ecology that rely heavily on image and video data to study their habitat, such as the deep-sea. There are a plethora of photographic and non-photographic techniques to collect this data, however the most common data collection platforms are cameras mounted to remotely operated vehicles (ROVs), AUVs and Drop cameras, as well as stationary cameras (e.g. mounted to buoys) (Morris *et al.,* 2014). These are robust collections methods that can acquire terabytes of data at a time and as a result large image and video-based datasets need to be analysed and quantified into useful ecological metrics (e.g. diversity, abundance, density) for it to be meaningful to ecologists (Durden *et al.,* 2021). However, using these CNN-based approaches over current manual analysis, means accuracy and reliability is required. Currently many issues still prevail, such as in intra-class identification accuracies are not consistent amongst classes nor consistent across inter-class level, and often decrease with increasing class number (Piechaud *et al*., 2019; Durden *et al.,* 2021). However, it does outweigh manual analysis in terms of efficiency, nor has it been noted to perform worse at detection rates compared to humans (mainly only classification), making it a tool worth further exploration. In order to achieve full automation using these tools (i.e. receiving meaningful ecological data during actual collection/camera deployment) pipelines outlining deployment in real-time settings is required as these are still rare. It is acknowledge there has been major steps toward achieving given the substantial increase in online support documentation, guidance and awareness around the need for large marine datasets for training (e.g. FathomNet; Katija *et al.,* 2021a) to mention a few. Although, there remains an assumed level of knowledge that is required in applying these tools, and often many different approaches can be taken. Thus, it is useful to understand how to narrow-down what are the key steps required, as well as exploring simpler alternatives, with considerations and limitations for non-experts.

1.5 Conclusion and thesis aims

Understanding trade-offs between approaches to take when beginning to use deep-learning technologies, as well as having cost-effective and reproducible pipelines for use in the field, could allow the global scientific community to benefit from the application of these tools to help unlock data, and fill gaps in knowledge.

The potential to integrate real-time classification with camera systems deployed in the field is beneficial though provision of immediate numeric data, which may facilitate more targeted exploration while in the field, as well as significantly reducing the post-cruise analysis time. This is particularly true with respect to ROVs, due to their popularity of use in deep-sea data collection (Whitt *et al.*, 2020).

Further steps are still required before this potential can be realised, as there are currently no documented accessible pipelines produced for real-time deployment of deep-learning on camera systems in the field. In addition, the results and overall benefits have not been extensively studied within the wider frame of ecological research, nor on a realistic scale. Approaches to fine tune these tools may be become tedious and over complicated for biologists to perform, and eventually this could outweigh the simplicity of manual analysis. However, there is rapidly growing interest in this area, and the community is already beginning to familiarise itself with the tools in order to achieve this.

This thesis will develop and test a basic pipeline for real-time classification of a single faunal class/species of xenophyophore (*Syringammina fragilissima*) in ROV livestream video. This study will quantify deep-learning classifier performance in identifying presence-absence, and abundance of *S. fragilissima*. In addition, it will investigate how different approaches to training a CNN may impact the performance in the context of real-time deployment. The overall aim of this work is to provide guidance to other benthic ecologists considering using CNNs in the context of classification of video datasets for quantifying ecological metrics, and highlights considerations to make when training your CNN for a given task (**Chapter 2**).

Chapter 2: Automating detection, classification and ecological data extraction in real-time during ROV deployment: A pilot study on deep-sea benthic species *Syringammina fragilissima*

2.1 Introduction

The deep sea encompasses the most extensive ecosystem on planet earth, stretching from 200 meters depth at the start of the bathyal to the hadal at 6000 meters and deeper. In total it encompasses over 50% of Earth's surface (Harris *et al.,* 2014) and supports a diverse range of fauna. Given the vast extent of the deep sea, it is not surprising that gaps in knowledge of currently known species occur, not to mention the plethora of potentially undiscovered taxa (Howell *et al.,* 2021). Additionally, as the Anthropocene era progresses, climate change and biodiversity loss is at the forefront, having potentially serious consequences for deep-sea organisms (Paulus, 2021) and the ecosystem services they provide, including nutrient cycling, carbon sequestration, and food provision (Armstrong *et al.,* 2012; Paulus, 2021). Human activities such as fishing, and the development of a new deep-sea mining industry are placing deep-sea ecosystems under increased pressure (Kung *et al.,* 2021). Therefore, it is imperative that we understand the diversity and distribution of deep-sea species, such that we can forecast the potential impacts of human activities on deep sea ecosystems.

The Food and Agriculture Organization of the United Nations (FAO) has taken steps to implement management strategies that help to protect deep-sea ecosystems from the damaging effects of bottom fishing. One of these steps is the designation of bottom trawl closures where vulnerable species groups, communities and habitats, termed Vulnerable Marine Ecosystems (VMEs) are known or likely to occur (FAO, 2007). One example of a VME are summits and flanks of seamounts that support VME-indicator taxa such as corals, sponges and xenophyophores. Xenophyophores are large rhizarian protists found below 500 meters and in soft sediment areas. They are around 5-20cm in size, white in colour (or covered with light coloured sand/mud sediment) and a complex round lump of white ribbon-like structures. In the North Atlantic a particular species, *Syringammina fragilissima* aggregates to form a structural habitat and is an important autogenic ecosystem engineer (Levin *et al.,* 1986). *S. fragilissima* aggregations support high densities and species richness of meiofaunal and macrofaunal organisms (Buhl-Mortensen *et al.,* 2010). However, the extent of their distribution is still unclear, as is the case for many other VME-indicator taxa (Ashford *et al*., 2014).

In order to quantify the extent of VME-indicator taxa, exploration of the deep sea remains imperative. In the late 19th century exploration had halted because of the intimidating scale of the task and lack of equipment developed in order to do so effectively (Danovaro *et al.,* 2014). The main discoveries then where made using semiquantitative dredges, box corers and trawls that are costly to collect and process and difficult to store (Clark *et al.,* 2016). The 21st century saw the introduction of submersibles with manipulators, hybrid ROVs, landers, drop cameras and even AUVs. These are now commonly used in deep sea scientific and biological exploration (Danovaro *et al.,* 2014; Levin *et al.,* 2019). These technologies acquire video and image datasets that are easily stored, less invasive and more cost-effective. They generate vast quantities (terabytes to petabytes) of image-based data on a single scientific voyage (Schoening *et al.,* 2018). However, before this video and image data can be useful, it must first be processed to extract ecological information (e.g. species diversity, density, abundance) that may then form the basis of decision-making. This has created a bottleneck in marine ecology, whereby high volumes of data need to be manually analysed by taxonomic experts, a process that is very labour and time intensive. In addition, manual analysis is highly inconsistent between observers and over time (Durden *et al.,* 2016b). Thus, both the pace and quality of data analysis are compromised by current practices.

CNNs (or classifiers) for CV tasks have emerged as a potentially useful tool in the field of marine ecology (Goodwin *et al.,* 2021). The most common practice is to train a classifier for a custom dataset and test its performance on unseen images to see how well it can identify objects (e.g. organisms) within them. This practice tends to be carried out using archived pre-collected datasets. However, there is the potential to apply these tools to real-time data collection. ROVs are a popular observational platform used to generate image-based data as a result of cost-efficiency and capability of providing high-resolution spatiotemporal data on individual organisms (Kuhnz et al., 2014). Cameras mounted to ROVs generate video data that is transmitted in real-time from the seafloor to a manned surface vessel via an umbilical, offering the possibility of applying classifiers in real-time. Integration of these tools on camera systems mounted to data collection platforms in the field could provide real-time interpretation of data. In theory this could significantly reduce or aid post-cruise analysis; saving time and money that could be utilised for more innovative science and, allow more targeted exploration while at sea by providing initial context of study sites species diversity, abundance and density or even indicate new species.

Recent studies have shown promise in the possibility of gathering real-time ecological data via observational platforms. For example, Katija *et al.*, (2021b) developed pipelines to run classifiers on AUV observation platforms to detect and track deep-sea pelagic jelly fish. Tseng and Kuo (2020) attained high recall (98% and 94%) and precision (94% and 77%) scores for both detection and counting of fish species in videos from electronic monitoring systems. Fewer studies have been performed on benthic fauna due to higher occlusion and morphological variability (Katija *et al.,* 2021b; Liu and Wang, 2021). It is now established that these classifiers can obtain classification accuracies equivalent to an expert taxonomists, whilst also being cost and time effective for imagery and real-time video datasets too. This provides hope for wider and more diverse applications across other taxa, such as deep-sea benthic fauna. However, replicating use of these tools in the wider scientific community is still a major area for improvement (Piechaud *et al.,* 2019). Viable examples need to be described in order to inform the potential application of these tools to this area. Greater collaboration between ecologists, computer scientist and engineers could help develop more user friendly tools that rely less on strong computer science expertise and programming abilities. Integration of these fields in this manner could help unlock real-time gathering of ecological data that could support sustainable management of deep-sea ecosystems, such as VMEs.

This study aims to develop, test and asses a simple, novel pipeline to run a YOLO classifier over a 95 minute transect of ROV livestream at sea for the detection, classification and extraction of ecological information (presence-absence, enumeration) on a single VME target species, *Syringammina fragilissima*. The target species was chosen due to it previously attaining high performances in a multi-class image classification task of various deep-sea taxa (7 to 52 classes) (Piechaud *et al.,* 2019). More specifically, this study assesses the performance of different classifiers by comparing classifier architecture (YOLO versions 3 vs. 4), two different training image datasets (unprocessed images vs. processed (variations in resolution, brightness and size)), and two different training approach (TL vs. TS, totalling 8 different classifiers beginning tested. To assess their performance standard CV metrics (precision, recall, accuracy and F_1 score) in an ecological context (presence-absence, accuracy in counting individuals observed) are used. Assessing these approaches and interpretating in this manner aims to provide guidance and consideration to future applications of CNN classifiers to benthic ecology in a real-time context.

2.2 Methods

The methodology of this study is as follows: collect images, pre-process these and then using two different architecture (YOLOv3 and 4), two different training approaches (TL and TS) and comparing the unprocessed images and processed images, train 8 different classifiers to be applied to ROV livestream. The classifiers are then applied to detect and classify the target species *S. fragilissima* and is then assessed using standard CV metrics in an ecological context.

2.2.1 Image data collection and annotation

The dataset used to train the DL classifiers in this study was provided by Howell as interlaced video frame-grabs pre-annotated using the BIIGLE software. Video was collected by the Natural Environment Research Council's (NERC) ROV *Isis* camera (Insite Pacific Mini Zeus camera - 1920*1080i, RGB-colour, 50 FPS), in June 2016 as part of the NERC funded DeepLinks (JC136) research project for which Howell was Principal Investigator. The video stills were extracted from 3 replicate 750m video transects at 500, 800 and 1200 meter depths, across three different sites, Anton Dohrn Seamount (ADS), North Rockall Bank (NRB) and Rosemary Bank (RB). These sites where chosen as they matched the species *S.fragilissima* and its respective habitat. The video stills were extracted at 20 second and 1minute time intervals (Appendix A2.1) in order to annotate as much of the videos as possible, and each frame was annotated by a single observer within the BIIGLE 2.0 software (Langenkämper *et al.*, 2017) using a regional catalogue of Operational Taxonomical Units (OTU) developed by Howell and Davies (2016). In total, 10,500 *S. fragilissima* individuals were manually annotated from these frames using the label OTU261, with their location within a still image depicted using a circle

(X, Y centre point co-ordinates and radius) or rectangle (X,Y points for length, rotation and width). Each *S. fragilissima* was visually inspected using the “Largo” evaluation tool in BIIGLE 2.0, to ensure consistency in the quality of training images in order to reduce error.

For the purpose of this study, whereby the classifiers were tested on real-time ROV livestream, these imagery datasets were chosen due to their consistency in the *S. fragilissima* species and the data collection platform used (standard High Definition (HD) 1920*1080i resolution stills ROV). To mitigate issues with occlusion *S. fragilissima* was a desirable VME due to them being found distributed in patches over soft sediment regions were 3D habitat complexity and background noise is negligible.

2.2.2 Hardware requirements

Two computers were used to undertake the study. Deinterlacing of the raw video data to extract frames for annotation was carried out on an AMD Ryzen 5 3600 CPU (6 cores), 32G of RAM (machine A). Pre-processing of imagery data, training classifiers and running real-time detection was carried out on an NVIDIA GeForce RTX 3090 GPU (10,496 CUDA cores and 328 tensor cores), 24G of VRAM hosted on an intel i7 Core with 64G of RAM (machine B).

2.2.3 Pre-processing of training data

Ffmpeg software was used to deinterlace the video transects in order to remove the zig-zag artefacts associated with moving objects in the original interlaced video feed, which might reduce classifier performance. Videos were re-encoded as Apple prores (1920*1080p at 25FPS) using bwdif filter and prores_ks encoder. From the deinterlaced transects, the annotated frames from section 2.2.1 were re-extracted at the same 20 seconds and 1 minute time intervals using a custom python script ‘extract_frames_N.py’ (Figure 4A; Appendix A2.2). Corresponding annotations.csv created on the BIIGLE software in section 2.2.1 were obtained.

The deinterlaced image data then underwent two pre-processing pipelines outlined in Figure 4C to give two different datasets for later use. One pre-processing pipeline used the API - Video and Imagery Analytics for the Marine Environment (VIAME; Dawkins *et al.,* 2018) to pre-process the imagery used in deep-learning classifier building and testing. Initially a custom python script 'xls2csv.py' (Appendix A2.2) was used to convert the BIIGLE formatted annotations.csv to VIAME format - groundtruth.csv (Figure 4B). In the VIAME format all annotation locations are depicted as bounding boxes (X, Y top left and X, Y bottom right) only, a requirement for the CNNs used in this study.

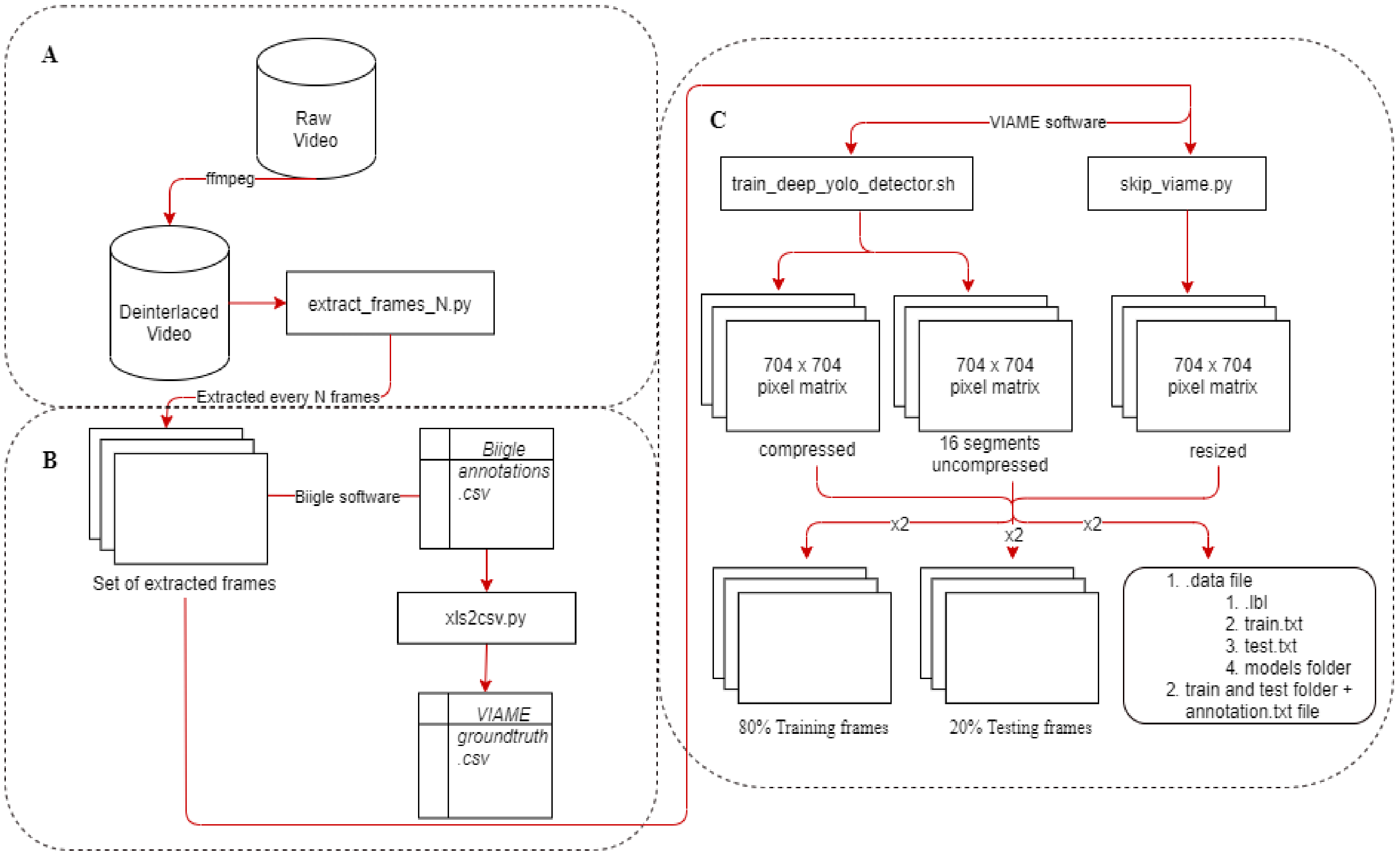


Figure 4| Two pre-processing pipelines for training and validation datasets, where A and B were carried out for both, and in C the images are processed using either the VIAME software (pre-processed dataset) or custom python script - 'skip_viame.py' (unprocessed dataset).

To produce the ‘processed images’ dataset the shell script, ‘train_deep_yolo_detector.sh’ located in the VIAME API Version 0.15.1 was ran using bash shell for Linux Operating Systems (OS); it calls upon the following shell script - ‘train_yolo_wtf_704.viame_csv.conf’. In this shell script the following processes are pre-defined and performed as standard by the VIAME API; 1) compressing the original images (1920x1080p) into a 704x704 pixel matrix, 2) rescaling and brightening the images by a factor of x1.25 and x0.6 respectively, and 3) splitting the rescaled image into x15 704x704 overlapped uncompressed pixel matrices (to improve resolution of smaller *S. fragilissima*), plus x1 whole rescaled image. To produce the ‘unprocessed images’ dataset a custom python script (‘skip_viame.py’, Appendix A2.2) bypassed steps 2 and 3, only compressing (letterbox compression) the original image into the 704x704 pixel matrix. Allowing assessment on the effects of augmenting the training dataset for better real-time performance. Both datasets are then randomly split 80/20 into training and validation datasets based on the occurrences of *S. fragilissima* rather than the number of images.

Both pipelines output the following 1) .lbl (label) file of the class names (in this study OTU261 = *S. fragilissima*), 2) .data file outlining the location of files (.lbl file) and folders (classifiers folder where weights containing trained parameters are outputted to every 10,000 training cycles) required for training and, 3) a training (80%) and validation (20%) dataset in folders, each with a corresponding YOLO formatted annotation.txt file outlining the class (OTU261 [1] or nothing [0]) and bounding box co-ordinates as a floating point number. Each custom-made python script was developed and edited using the Spyder IDE (Raybaut, 2009) and for version control (of Python language, packages and libraries) git was used (source: https://git-scm.com/). Anaconda virtual environments (Anaconda, Vers 2.4.0, 2021) were used to perform the pre-processing steps.

2.2.4 Training classifiers

Two different classifiers, YOLO versions 3 (Redmon and Farhadi, 2018) and 4 (Bochkovskiy *et al.,* 2020), were trained twice each with the training dataset created from the two pre-processing pipelines A and B (Figure 4). Training the classifier required compilation of the DL framework Darknet which contained YOLO networks backbone (Darknet-53), onto machine B. Darknet is an open source neural network framework written in C, it requires compatible versions of CUDA and cuDNN (DNN library with optimised fast GPU implementations) to be installed in order to activate the GPU for computation; for machine B this was CUDA 11.3 and cuDNN 8.2.1. In this study using Darknet was optimal as it is the specified framework built by Redmon and Farhadi (2018) for YOLO architectures in order to gain the speed required for real-time performance, in addition to accuracy.

For each dataset and each classifier two training approaches were used, TS and TL (outlined in Chapter 1). Therefore, in total, eight classifiers were trained: two different classifiers using two different pre-processing methods, and two different training approaches. For training these classifiers, alongside the outputs from Figure 4C, YOLOv3 and v4 weight files (where learnt parameters are stored corresponding to features of the target class) and corresponding configuration files (outlining classifiers architecture) were acquired from an online GitHub repository (source: https://github.com/AlexeyAB/darknet; Appendix A2.3). For the TL approach lower layers (i.e. Darknet-53 backbone = initial 53 layers) of the network are pre-trained on the MS COCO dataset (328k training images over 80 classes). Bash shell commands outlined in the 'AlexeyAB' GitHub account (source: https://github.com/AlexeyAB/darknet) were utilised to begin the training process on machine B. Due to time constraints fine-tuning of hyperparameters was limited, thus to ensure consistency all classifiers were trained with the same hyper-parameters set out as standard in the 'AlexeyAB' GitHub account: batch size

(64), subdivisions (16), training cycles (45,000) and training cycle steps at 80% (36,000) and 90% (40,500) of total training cycles (Bochkovskiy *et al.,* 2020), learning rate (0.001). A nomenclature (Table 3) outlines the classifiers with consideration to their training datasets, YOLO architecture and training approach.

Table 3| Nomenclature of classifiers names and characteristics. The different classifiers names are a combination of group name, pre-processing and training approach. Group name (A and B) defines the classifier architecture being used (YOLO version 3 or 4). Both groups use the 'processed imagery' (V) and 'unprocessed imagery' (P) training datasets, as well as both training methods, TL and TS. Resulting in 8 classifiers in two different groups.

	Groups	
	A	**B**
Classifier version	3	**4**
Pre-processing method	V	P
Training approach	TL	TS
Classifiers names	**V3TL, V3TS, P3TL, P3TS**	**V4TL, V4TS, P4TL, P4TS**

2.2.5 Assessing the classifiers training process

To assess the performance of the 8 classifiers (Table 3) during training, average training loss and mean Average Precision (mAP) were calculated. The average training loss value (Complete-Intersection over Union [CIoU] loss) assesses the distance between bounding boxes centre points predicted by the classifiers, after each training cycle, with the ground-truth (annotated) bounding boxes outlined in the annotations.txt (Figure 4C). In addition, CIoU inspects the overlapping area and aspect ratio between both bounding boxes. If the classifier is training well the CIoU loss is expected to drop exponentially, then stabilise over the course of training cycles. Stabilised values suggested by Bochkovskiy *et al.* (2020) range from 0.05 for a small classifier with an easy dataset, to 3.0 for a large classifier with a difficult dataset (e.g. multiple classes).

The mAP score is calibrated every 4 training cycles, where a steady increase indicates the classifier training is stable. The following performance metrics are used to calculate mAP:

True positives (TP): the number of correct detections of a ground-truth bounding box;

False positives (FP): the number of incorrect detections of a non-existent object or a detection misplaced from the ground-truth bounding boxes;

False negative (FN): the number of undetected ground-truth bounding boxes;

In order to define what a correct detection is the Intersection over Union (IoU) metric is used. IoU measures the overlapping area between the predicted bounding box (B_p) and the ground-truth bounding box (B_{gt}) from the training dataset, and divided by the area of union between them. It can be defined using the following equation (Padilla *et al.,* 2020),

$$IoU = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})}. \quad (1)$$

The IoU has an associated threshold (t) that is pre-defined by the user, thus a correct detection can be classified as IoU $\geq t$ and incorrect if IoU $< t$. In this case IoU was set to 0.5; meaning 50% overlap between B_p and B_{gt} is required for a detection to be counted as a TP.

From the number of TP, FP, FN detections made by the classifiers precision P and recall R are calculated respectively and defined as

$$P = \frac{TP}{TP + FP}, \quad (2)$$

$$R = \frac{TP}{TP + FN}. \quad (3)$$

Precision is the percentage of TPs within all the predictions (i.e. rate of FPs), where IoU threshold is 0.5. Whilst recall is the percentage of TPs amongst all the given ground truths. During each training cycle the classifiers precision and recall values are plotted on a P-R curve, where the area under the P-R curve (AUC) indicates the classifier's performance; a good performance would be indicative of a high precision with increasing recall. To increase the accuracy of AUC, the curve is interpolated using an 11-pointed average precision (AP, Everingham *et al.,* 2010) defined as:

$$AP = \frac{1}{11} \sum_{R \in \{0, 0.1, \ldots.1\}} P_{interp}(R), \qquad (4)$$

Where,

$$P_{interp}(R) = \max P(\tilde{R}),\ \tilde{R} \geq R. \quad (5)$$

Here the maximum precision $P_{interp}(R)$ at 11 equally spaced recall levels [0,0.1,...,1] is averaged. Then mAP is calculated to give a measure of an object detectors performance given multiple classes are represented in the training dataset. Thus, it can be defined as:

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i, \qquad (6)$$

Where the AP_i is simply the AP at each 11 point interval (i) on the P-R curve over all classes (N) (Padilla *et al.,* 2020). In this study N =1 making $mAP \approx AP$.

Once trained (after the 45,000 training cycles) the 20% validation dataset was used for testing. The final outputted average loss value and mAP score from the training process was used in the assessment of the overall performance of each classifier (Appendix A2.4).

2.2.6 Pipeline for real-time deployment: *in-situ* and synthetic ship

2.2.6.1 Hardware and software design: *in-situ* and synthetic ship

Two design approaches were developed in order to run the classifiers on the ROV livestream at sea (*in-situ*) and on land (synthetic ship); both are outlined in Figure 5. For the *in-situ* design,

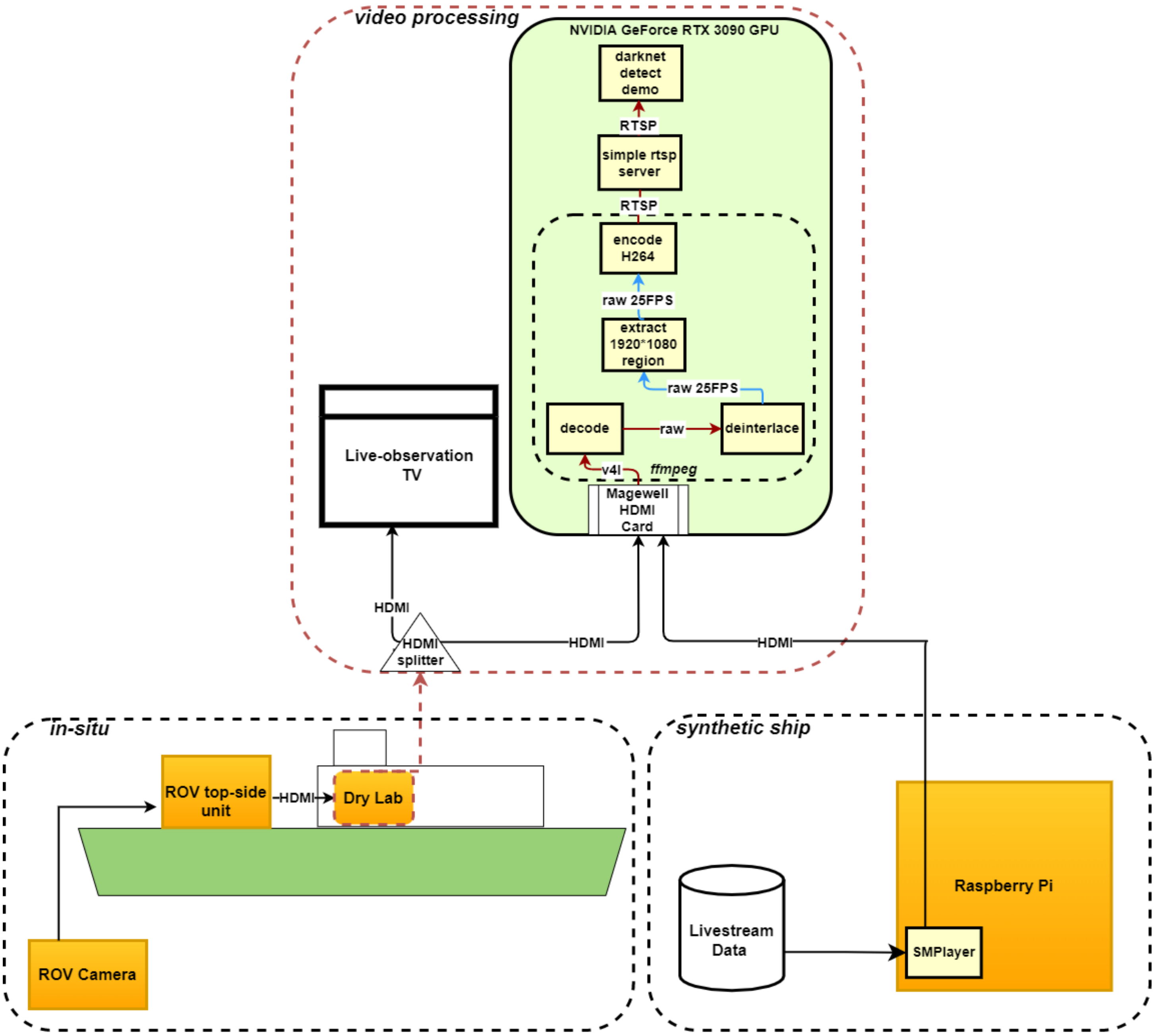


Figure 5| Schematic design of the two approaches for deploying classifiers on ROV livestream at sea (*in-situ*) and on land (*synthetic ship*). Both use the same *video processing* steps, expect the live-observation television (TV) required for scientists observation aboard the vessel.

ROV Livestream was fed into the ROV top-side unit on-board the vessel, then passed via HDMI into the dry lab (as standard practice). The livestream was split and sent to a live-observation television, and into a Magewell Pro Capture HMDI card plugged into machine B, via a HDMI splitter (Figure 5: *in-situ* design). The capture card read the ROV livestream into machine B to where ffmpeg software was used again to produce deinterlaced ROV livestream (see section 2.2.3) in order to mimic the imagery the classifiers were trained with. Retaining the speed of 25 FPS meant classifiers were detecting and classifying at a 'real-time' performance. The video stream was then compressed using a 'hevc-nvec' H.264 encoder (using dedicated hardware within the Nvidia GPU). Real Time Streaming Protocol (RTSP) is used to transfer the compressed, 25 FPS, ROV livestream from the server (Magewell Pro Capture HDMI card) to the endpoint device (Darknet detector demo; built into the Darknet framework (source: https://github.com/AlexeyAB/darknet; Bochkovskiy *et al.,* 2020)).

At the time of *in-situ* deployment (section 2.2.6.2) only 1 of the 8 classifiers had been trained to an acceptable standard to test at sea. Thus, all remaining classifiers were tested using the synthetic ship design. Here, the livestream data (video) collected for the transect analysed by the 1 classifier *in-situ* (section 2.2.6.2) was played back via the HDMI output on a Raspberry Pi 4 model B at the same frame rate (59.95 FPS) using SMPlayer software, essentially acting as the equivalent of the ROV livestream *in-situ*. The video was then passed to the Magewell Pro Capture HMDI card where it followed the same processing steps as the *in-situ* design (Figure 5: *video processing* steps). For both pipelines, a minimum of 12GB VRAM was required from machine B GPU to run the classifiers via the Darknet detector (in demo mode) on ROV livestream. Darknet outputs both a video stream showing detections as bounding boxes, and a JSON stream with exact times and co-ordinates of each detection. Communication amongst the software outlined in this pipeline was designed using custom Python and Shell scripts outlined in Appendix A2.5.

2.2.6.2 In-situ test site and livestream data collection

Using the *in-situ* ship design (Figure 5), classifier V3TS (Table 3) was run on ROV livestream during the 'Resources of Rockall Bank' research cruise (CE21010) in August 2021. The classifier ran along a 1 kilometre (km) transect where the presence of *S. fragilissima* was predicted by

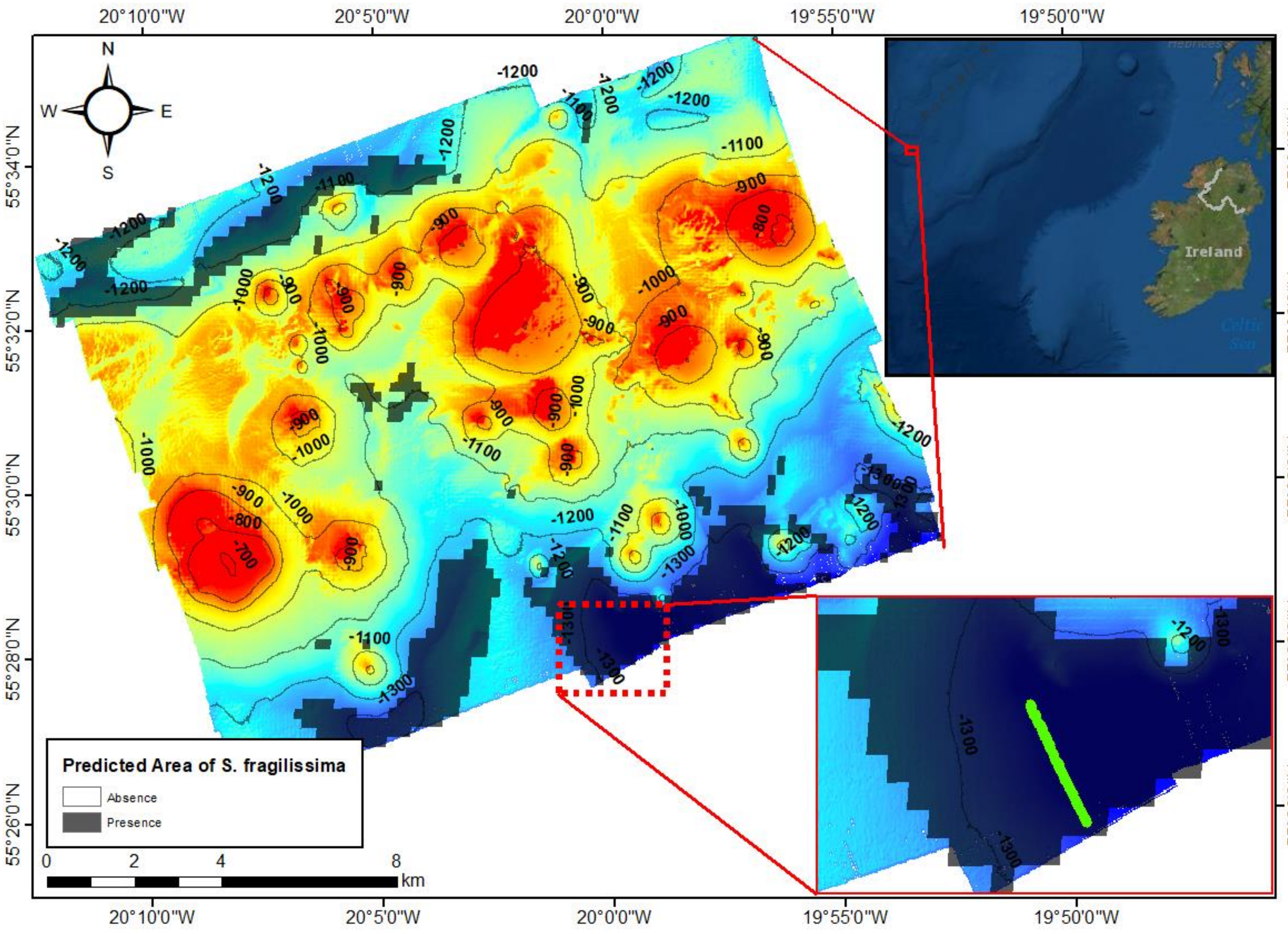


Figure 6| Location of 1km transect (green line) North-East (NE) of Fangorn Bank, in the NE Atlantic used for testing the pipeline for real-time deployment of the V3TS classifiers. Blackened pixels (p) indicate areas of predicted *S. fragilissima* presence (Graves *et al.,* 'in prep'). Maps bathymetry layers were processed on QPS Qimera v2.4.2 and generated on ArcGIS v10.9, and indicate depth ranges.

models created by Graves *et al.* (in prep) (Figure 6). The site was located on the North-East (NE) side of Fangorn Bank in the NE Atlantic Ocean. The transect equated to approximately 1.35 hours (h) of ROV livestream data, therefore, 1.35 hours of detection data collected for *S. fragilissima* at an average ROV speed of 0.1ms^{-1} at around 1300m water depth of seabed (Figure 6). This ROV livestream data was collected and stored for use in the synthetic ship design (Figure 5), which was used to test all remaining classifiers.

2.2.7 Analysis of real-time performance: presence-absence and estimated counting

A custom python script (Appendix A2.5) was used to output a .csv file for detections made by each classifier during real-time analysis of the ROV livestream data. This csv file provided the frame number in which a detection was made (i.e. 1 - 142,710, as 95.14 minutes $\approx$ 142,710 frames at 25 FPS), its bounding box co-ordinates (pixels (p)), associated confidence score (ranges 0 – 1), and timestamp within the livestream data (0-95.14 minutes). In order to ground-truth the classifiers' detections, manual annotation of the entire transect (95.14 minutes) was carried out by one observer. Whereby, areas of *S. fragilissima* presence were timestamped and then converted into frame numbers. Therefore, each frame consisted of a manual annotation of present (1) or absent (0). Conversion from timestamp to frame number simplified the coding process, and made it easier to directly compare manual detections with those output by the classifiers.

Possible predictions of the classifiers differ from the ones defined during training (section 2.2.5) and are outlined in the Table 4 for context.

Table 4| Possible predictions of the classifiers when compared to the manual annotations. Comparison of predictions with the manual annotations is done at the individual frame level (totalling 142,710 frames) for the confidence threshold, and then in 1s (25 frames) increments (totalling 5708s) for the tracking threshold and presence-absence (P-A) analysis (both explained below).

Prediction type	Description	
	Confidence threshold	**Tracking threshold + P-A**
True Positives (TP)	Classifier correctly identifies presence of *S. fragilissima* per frame	Classifier correctly identifies presence of *S. fragilissima* per second
True Negatives (TN)	Classifier correctly identifies absence of *S. fragilissima* per frame	Classifier correctly identifies absence of *S. fragilissima* per second
False Negatives (FN)	Classifier misses actual presence of *S. fragilissima* per frame	Classifier misses actual presence of *S. fragilissima* per second
False Positives (FP)	Classifier incorrectly identifies presence of *S. fragilissima* per frame	Classifier incorrectly identifies presence of *S. fragilissima* per second

2.2.7.1 Removing noise using thresholding optimisation: confidence and tracking

Initially no confidence threshold score was implemented for the real-time analysis of the livestream. This threshold is a confidence value the classifier needs to obtain before recording a positive detection of an *S. fragilissima* to the .csv file. To reduced noise in the dataset (i.e. FP detections), confidence threshold optimisation methods were explored using the 'optimal.threshold' function in the "PresenceAbsence" package in R (Freeman and Moisen, 2008). The initial confidence values for each classifier (i.e. no confidence threshold) were compared with the manual annotated observations using three appropriate optimisation methods; predicted prevalence is equal to observed prevalence ('PredPrev=Obvs'), minimal distance between ROC plot and (0,1) ('MinROCdist') and sensitivity is equal to the specificity

(‘Sens=Spec’). From these an optimal confidence threshold was outputted along with a recall and specificity score between 0 and 1. The updated threshold for the confidence score set for each classifier during real-time analysis was based on the optimisation method that attained the highest recall score (i.e. retaining as many positive detections that matched manual observations of presence) following Dujon *et al.* (2021). The classifiers were then re-run with the optimised confidence threshold implemented and a new .csv file generated.

In order to reduce noise further, and based on the theory that a classifier used for real-time detection tracks a TP detection more consistently over frames than FPs (Bashir and Porikli, 2006) a ‘tracking’ threshold was implemented. To achieve this without implementing a tracker into the pipeline, manual and classifier generated detections were binned into 1-second increments (25 frames) for the whole transect (5708 seconds). The number of frames in those increments containing positive detections (i.e. 1-25) with respect to them being either a TP or FP detection was visualised in a violin plot. A subsample (N=100 per detection) of the TP and FP detections, were investigated and analysed using the T-test from the “rstatix” package in R (Kassambara, 2021) to test for a significant difference in the number of frames in a one second increment each type of positive was detected over. Confirmation of a significant difference informed implementation of a tracking threshold value. This value was based on the lower quantile value of the FP detections per classifier (e.g. 1-25), and was chosen in order to maintain as many TPs whilst removing large areas of FPs, changing them to true negatives (TN) detections (frames correctly detected to have no *S. fragilissima* present).

2.2.7.2 Calculating classifier performance in detecting areas of presence-absence *S. fragilissima*

Classifiers were then assessed post-thresholding based on their ability to highlight areas of *S. fragilissima* presence-absence at 1-second increments (as described in section 2.2.7.1). The different possible predictions of the classifier are detailed in Table 4. The respective number of each type of prediction (the confusion matrix) was used to calculate the following standard performance metrics using the 'confusion_matrix' function in the 'cvms' package in R (Jeyaraman *et al.,* 2019), these metrics are described as follows:

Recall (sensitivity or true positive rate) quantifies the proportion of areas (1s increments along transect) of *S. fragilissima* in the transect (Figure 6) correctly identified. It varies between 0 and 1, were 1 means all areas are identified.

$$Recall = \frac{TP}{TP + FN}\,. \quad (7)$$

Precision (positive predictive value) quantifies the proportion of TPs among all the positive predictions for areas of *S. fragilissima*. A value of 1 indicates all the positive detections for areas of *S. fragilissima* are in fact areas of *S. fragilissima*.

$$Precision = \frac{TP}{TP + FP}\,. \quad (8)$$

Accuracy quantifies the number of all correct predictions (TP + TN) for areas of *S. fragilissima* presence or absence with respect to the total predictions made. A value of 1 implies no false predictions (FP + FN) and all correct predictions are identified.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}\,. \quad (9)$$

F$_1$ Score quantifies the harmonic mean of precision and recall, meaning a value of 1 indicates perfect precision and recall (as defined in equation 7 and 8).

$$F_1 = \frac{TP}{TP + \frac{1}{2}(FP + FN)} = 2 \cdot \frac{precision \cdot recall}{precision + recall}. \quad (10)$$

Each metric gave an estimation of the overall performance of each classifier in order to determine the best performing in real-time analysis, as well as being fitting for an ecological setting.

2.2.7.3 Estimating classifier performance at counting individual *S. fragilissima*

Based on the overall performance of each classifier, the optimal classifier was chosen for further analysis regarding its ability to estimate counts of individual *S. fragilissima*. From the transect (5708s), 100 1-second (25 frames) increments were taken evenly along the transect. For each 1-second increment the quantity of *S. fragilissima* present was manually counted by a the same human observer. The best performing classifier's predictions for counts of individual *S. fragilissima* (with both thresholding methods still implemented) were also quantified.

In order to evaluate how well the classifier predicted counts matched manual counts, a linear regression analysis was performed using the "tidyverse" package (Wickham, 2017), alongside the associated error for the regression. The error indicates the degree to which the classifier over or under predicts the quantity of *S. fragilissima*. This is calculated using the root mean square error (RMSE).

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y_i})^2}, \quad (11)$$

Where, N is the number of 1-second increments (N = 100), i is the increment number being compared (1 – 100th), y_i is the manual count of *S. fragilissima* at the i^{th} data point, whilst $\widehat{y_i}$ is the corresponding predicted count made by the classifier.

2.3 Results

The results outline 1) classifier performance using different confidence thresholds in order to record a positive detection, 2) number of frames in a 1 second increment (i.e. 1-25) containing positive detections, either a TPs or FPs. 3) classifier performance at recalling areas of presence-absence, and 4) optimal classifier performance in counting individuals.

2.3.1 Thresholding: *confidence*

Confidence threshold values varied marginally amongst all threshold optimisation methods, ranging from 0.02 to 0.09 (Table 5). The highest fluctuations in threshold values occurred using both the *Sens=Spec* and *MinROCdist* methods (0.02-0.07 and 0.02-0.09, respectively). The *PredPrev=Obs* methods threshold values remained more consistent, ranging from 0.02-0.03.

Table 5| Threshold optimisation methods; '*Sens=Spec*', '*PredPrev=Obs*' and '*MinROCdist*' confidence threshold values and their associated recall and specificity performance metrics for the 8 classifiers outlined in Table 3.

Classifier	Threshold Optimisation Method	Confidence Threshold Value	Recall	Specificity
V3TS	Sens=Spec	0.03	0.481	0.577
	PredPrev=Obs	0.02	0.629[1]	0.418
	MinROCdist	0.05	0.481	0.577
V3TL	Sens=Spec	0.02	0.607[=1]	0.455
	PredPrev=Obs	0.02	0.607[=1]	0.455

	MinROCdist	0.03	0.452	0.624
P3TS	Sens=Spec	0.07	0.575	0.557
	PredPrev=Obs	0.03	0.710^1	0.386
	MinROCdist	0.08	0.557	0.578
P3TL	Sens=Spec	0.07	0.576	0.565
	PredPrev=Obs	0.03	0.708^1	0.403
	MinROCdist	0.09	0.545	0.601
V4TS	Sens=Spec	0.02	0.548$^{=1}$	0.525
	PredPrev=Obs	0.02	0.548$^{=1}$	0.525
	MinROCdist	0.02	0.548$^{=1}$	0.525
V4TL	Sens=Spec	0.02	0.538$^{=1}$	0.530
	PredPrev=Obs	0.02	0.538$^{=1}$	0.530
	MinROCdist	0.02	0.538$^{=1}$	0.530
P4TS	Sens=Spec	0.03	0.497	0.604
	PredPrev=Obs	0.02	0.623^1	0.460
	MinROCdist	0.03	0.497	0.604
P4TL	Sens=Spec	0.04	0.576	0.622
	PredPrev=Obs	0.02	0.715^1	0.457
	MinROCdist	0.04	0.576	0.622

PredPrev=Obvs was chosen as the optimal method, based on the all classifiers consistently retaining the highest recall values (0.538 - 0.715) using this method, although the specificity values were consistently lower (0.386 - 0.530). Choosing a threshold optimisation method based on the highest recall retained as many true positive detections by the classifiers. In this case retaining areas of known presence of *S. fragilissima* was the priority rather than removing all false positive detections.

2.3.2 Thresholding: *tracking*

Consistently, each classifier showed a significant difference in terms of the distribution of true positive versus false positive detections based on the number of frames in that second it was

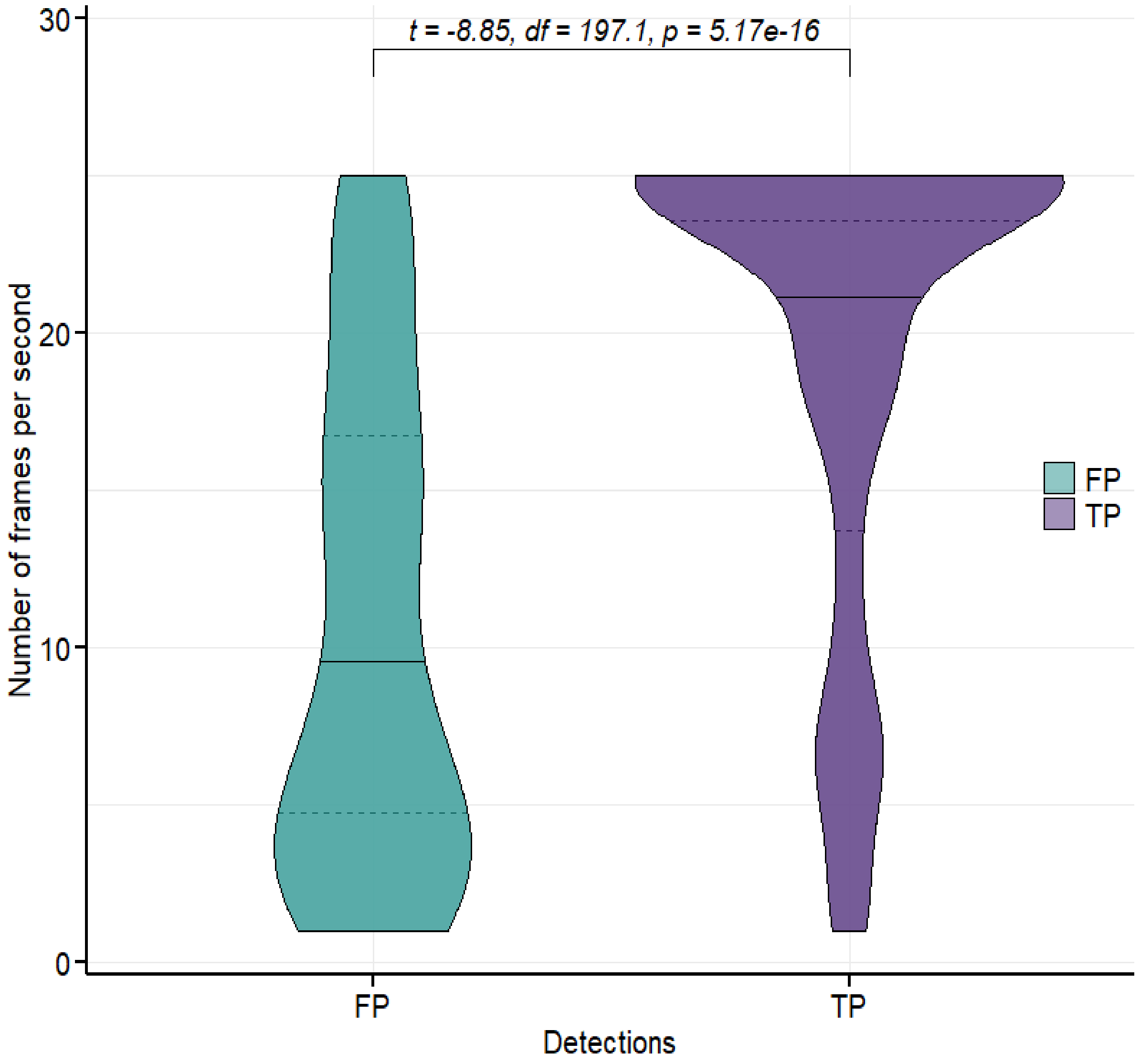


Figure 7| The number of frames in a 1-second increment (25 frames) a *S. fragillissima* is tracked for by the V3TS classifier (best performing classifier in terms of recall; Table 5) given the detection is a false positive or true positive, where N = 100 randomly sampled detections each.

tracked for (Appendix A2.6). Figure 7 displays this relationship for the best performing classifier (V3TS, Table 6), where true positive detections made by the classifier are associated with being present in more frames ($p_{adj} < 0.0001$). Indicating the classifier tends to track true positive detections for longer. However, the cost of implementing a tracking threshold with any overlap between false positives and true positive means a loss of true positive detections. Thus, to reduce the amount of noise and minimise the loss of true positive detections the tracking threshold was set based on the lower quantile value of the false positives detections for all classifiers (V3TS = 5 *1sf*), before being analysed for real-time performance (Section 2.3.3).

2.3.3 Classifiers performance: validation and real-time presence-absence

Classifiers trained using the 'processed images' dataset consistently outperformed those trained using the 'unprocessed images' dataset, with the top 3 $mAP_{@50}$ scores during training being 69.9%, 68.9% and 66.6% for classifiers V4TL, V3TL and V3TS, respectively (Table 6). However, despite V4TL being the top $mAP_{@50}$ score, the average loss value was higher (1.02) than V3TS and V3TL (0.314 and 0.246, respectively), indicating the classifiers training had more error associated with the parameters being learnt. This could indicate why it did not perform as well on the recall metric during real-time performance compared to V3TS and V3TL (Table 6), despite the higher $mAP_{@50}$ score during training.

Table 6| Performance metrics assessed during i) the validation of the training process and ii) the real-time performance (25 FPS) of each classifier on unseen ROV Livestream. Standard metrics for evaluating the classifiers training are used; average loss value and mean Average Precision (mAP), where the IoU (section 2.2.6, equation 1) is equal or greater than 50% overlap. Real-time performance metrics are performed post-thresholding (section 2.3.1 and

2.3.2) in order to remove noise. In all metrics the top (1), second (2) and third (3) scoring are indicated by the superscript and highlighted colour.

	Training		**Real-Time Performance**			
Classifiers	**Average Loss Value**	**mAP@50 (%)**	**Recall**	**Accuracy**	**Precision**	**F_1 Score**
V3TS	0.314	66.6^3	0.933^1	0.661	0.629	0.751
V3TL	0.246^3	68.9^2	0.909^2	0.629	0.608	0.728
P3TS	0.178^1	40.1	0.841	0.706^2	0.690^3	0.758^3
P3TL	0.240^2	39.2	0.838	0.709^1	0.695^2	0.761^1
V4TS	1.52	64.2	0.816	0.679	0.671	0.736
V4TL	1.02	69.9^1	0.899^3	0.686	0.657	0.759^2
P4TS	0.556	38.1	0.863	0.677	0.657	0.746
P4TL	0.256	30.6	0.756	0.694^3	0.699^1	0.726

The lowest loss values (errors) during the training process were associated with classifiers trained using the 'unprocessed images' training dataset (P3TS; 0.178 and P3TL; 0.240), however all classifiers trained using the 'unprocessed images' training dataset consistently retained the lowest mAP@50 score (Table 6) despite consistently performing the best in real-time performance metrics; accuracy, precision and F_1 score (Table 6). In terms of ecological surveying, the recall metric is deemed most important to perform well in as it better to have all known areas than potentially missing some (Dujon *et al.* (2021) and for attaining more training imagery to improve classifier performances. This suggests a degree of pre-processing of training imagery is an important consideration in classifiers ability to predict areas of *S. fragillisima* presence in real-time ROV livestreams using the pipeline outlined in section 2.2.7,

Figure 5. This is further supported by the high variability in top scoring training and real-time performance metrics for the training approach used (i.e. TS versus TL).

YOLOv3 outperformed the more up-to-date YOLOv4 architecture (Table 6) in terms of training and real-time performance metrics. This suggests YOLOv4 architecture size and mathematical complexity may be excessive considering the task at hand, where only a single-class with quite simple morphology is being detected and classified. Overall performance of the classifiers indicates that using either V3TS or V3TL provides the most consistent recall for automating the process of finding as many areas of *S. fragillisima* presence as possible. However, V3TS marginally surpassed V3TL in its real-time performance for all metrics (Table 6). Therefore, it was chosen for further analysis regarding its ability to estimate counts of individual *S. fragilissima.*

2.3.4 Classifiers performance: estimating number of individuals in real-time

The V3TS classifiers estimated capability in correctly detecting the number of individual occurrences of *S. fragillisima* in comparison to a human observer is 73% ($R^2 = 0.73$, $p < 0.001$; Figure 9). Human observers counted a total of 130 individuals with the V3TS classifier counting a total of 174. The V3TS classifier tends to correlate more closely with manual observations

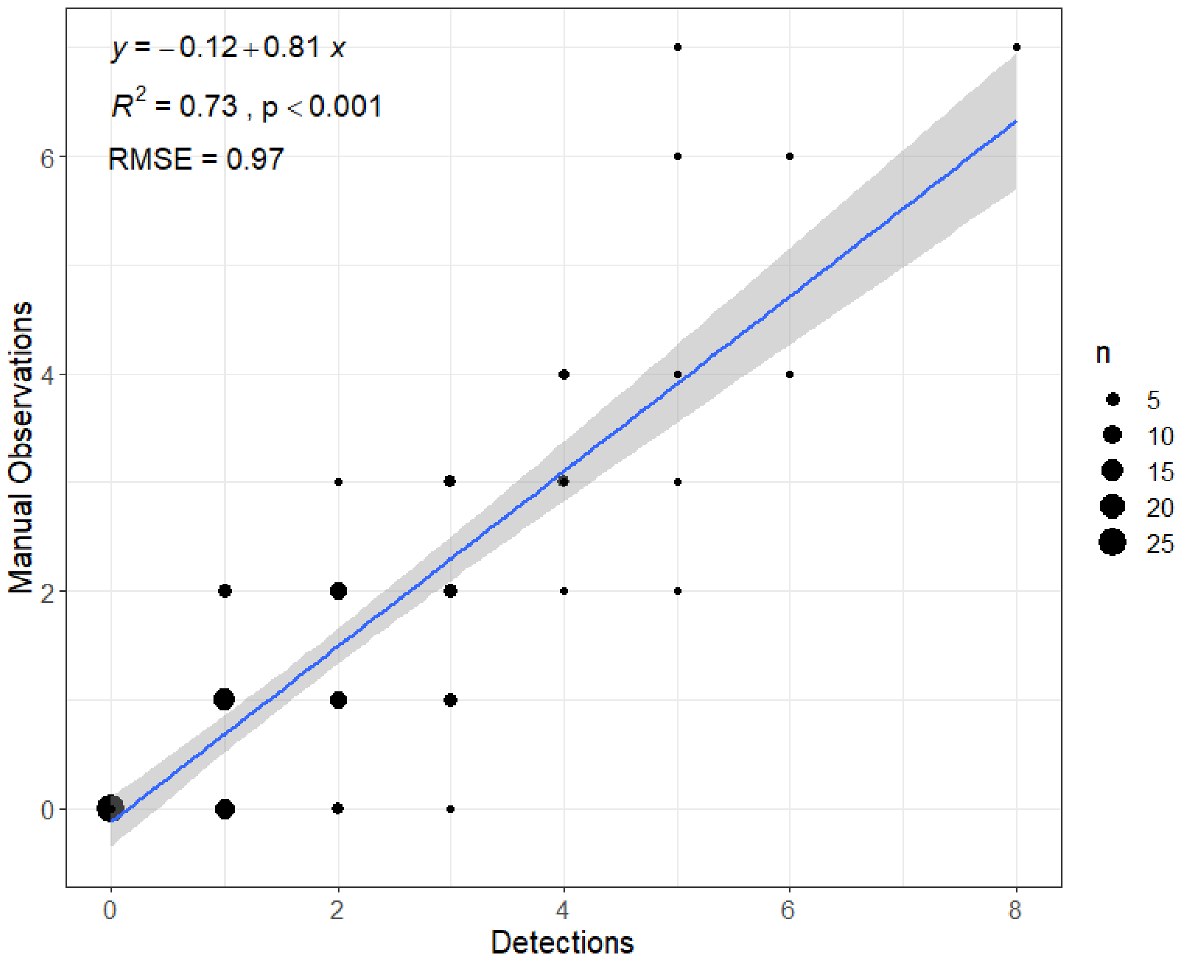


Figure 8| Relationship between the number of *S. fragillisima* detected by the V3TS classifier (Detections) and the number manually counted by a human observer (Manual Observations), over 100 evenly spaced, 1s increments of the whole transect (N = 100). Point size indicates the number (n) of datapoints for a given number of *S. fragillisima*. Root mean square error (RMSE) quantifies the magnitude the classifiers under or over predicts the number of *S. fragillisima* on average over the sample size.

when there is a lower quantity (0-4) of *S. fragillisima* in a 1-second increment than higher quantities, indicated by the variance around the correlation line seen in Figure 8.

Additionally, this would explain the root mean square error (RMSE) of $\pm$0.97, where the classifier over predicts the quantity of *S. fragillisima* when higher quantities are present. However, this could be indicative of more data points for 1-second increments of lower counts and a larger or more representative sample of the transect could further clarify this relationship.

Despite V3TS classifier's high recall (0.933; Table 6), alongside a strong relationship for correctly counting individual occurrences of *S. fragillisima* ($R^2 = 0.73$; Figure 8), it is recognised that a high proportion of false positives are also detected, where precision is the second lowest scoring (0.629) of all classifiers (Table 6). This is further evident in Figure 8 where V3TS tends to over predict on average 1 (RMSE = 0.97) *S. fragillisima* than manually observed given a sub-sample of the data (N = 100s). Visual insight into the kind of positive detections V3TS makes indicates higher false positive rates could be indicative of the possible features learnt during the classifiers training process that associated it with being an *S. fragillisima*. Figure 9

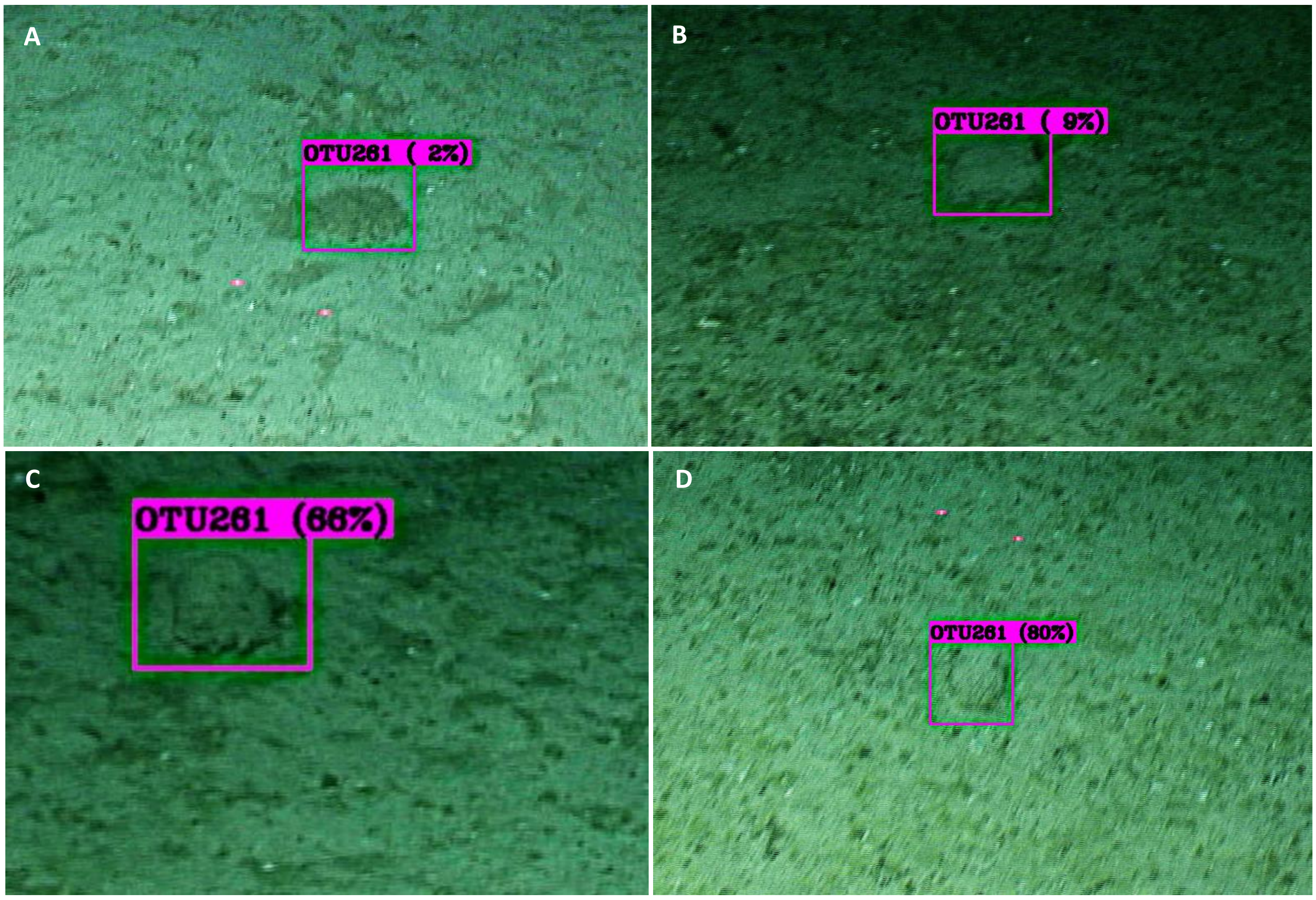


Figure 9| Examples of V3TS classifiers detections for false positives (A and B) and true positives (C and D).

displays two examples of false positive detections (Figure 9A and B) compared to true positive detections (Figure9C and D); all of which have key common features (rounded shape with darker edges compared to the background sediment). This suggests greater knowledge of the features learnt during the training process may be helpful in understanding classifier performance, and pre-processing of the training dataset to enhance important features may be a necessary step in future pipelines.

2.4 <u>Discussion</u>

The main objectives of this study were to 1) assess performance of different classifiers comparing classifier architecture, pre-processing of training imagery, and training approach, to inform future application of CNN classifiers to benthic ecology; and 2) test a novel, low-tech pipeline for the application of CNN classifiers on ROV livestream at sea for identifying taxa. To the authors knowledge this study is this first to attempt to use these tools on a ROV observation platforms on a scientific expedition in deep-sea benthic ecology, using solely 'off-the-shelf' CNNs architectures, open-source software and libraries.

2.4.1 <u>Overall performances</u>

The classifiers in this study achieved a maximum performance of 93% (0.933) recall and 63% (0.629) precision at a real-time performance of 25 FPS, for identifying presence-absence areas of *S. fragillisima*. In other studies, performances achieved using manual analysis by experts range from 50 to 95 % in benthic organisms (Beijbom *et al.,* 2015). Performances for automated classification on imagery datasets using CNNs have seen ranges from 78 to 98% over various benthic megafauna (Marburg and Bigham, 2016; Xia *et al.,* 2018; Piechaud *et al.,* 2019; Durden *et al.,* 2021; Lütjens and Sternberg, 2021), noted they are multi-classification studies rather than single class as in this study. Real-time performance has followed suit, however the speed (FPS) these classifiers can achieve detection varies. For example, Han *et* *al.* (2020) achieved 90% mAP$_{70}$ at 58 milliseconds (ms) for detection of benthic marine fauna (e.g. scallops, sea urchin and sea cucumbers), with the potential of running at 17 FPS on ROV platforms. Liu *et al.* (2021) attained a better performance (precision 96.32% and RMSE 8.84) for detecting benthic fauna such as shrimp, mussels and crabs. However, the speed for real-time detection is limited (191ms). There are few studies that have demonstrated the potential of using these classifiers on real-time observational platforms deployed in the field, and fewer that retain statistical information (e.g. presence, absence or counting) or run over realistic sampling periods. The best comparable study to have achieved this in the deep sea is by Katija *et al.* (2021b) who achieved real-time detection and tracking (at 1-second intervals) of pelagic organisms on AUV platforms at sea during a 5hr continuous observation period. However, it was observed that tracking, which enables the ability to count, is impeded with high occlusion and overlap, particularly when applied to benthic habitats, and performances did decrease the longer the observation period continued.

Most studies do not attain high accuracy without a degree of misclassification or detection (false negatives or false positives). In fact, there is no acceptable baseline error rate for using these tools to extract ecological information when installed on observation platforms for real-time application such as this. One reason for this is that a high degree of variance in human annotators still persists (Culverhouse *et al.,* 2003), as human experts may perform well with classification but not necessarily detection (Durden *et al.,* 2016b; Durden *et al.,* 2021). Especially in a real-time scenarios where keeping track of multiple objects or multiple of the same object is difficult, if not impossible. Therefore, measuring a classifiers ability to perform against a human annotator can impact the precision and recall metrics, whereby the classifier attains high FPs (i.e. precision) but in actual fact it is picking up on individuals missed by the human, making the classifier look worse than it actually is. This could be indicative of the overestimated individual counts seen in this study and highlights that the interpretation of classifier outputs requires a degree of consideration and cannot always be taken at face value.

Overall, the consensus is that to make this a worthwhile tool for a marine scientist to use as an alternative to current manual efforts; classification results must be near-equivalent to those achieved manually, a significant degree of automation of effort for implementation is needed, as well as reproducibility by non-experts. Therefore, even with the best performing classifier in this study, and the pipeline for running on a real-time observational platform being feasible and simplistic, reducing or checking false detections requires manual elimination post-cruise, and the approach is limited to detecting sections of video with *S. fragillisima* presence, rather than accurate count data. Thus, is it not good enough to be considered a suitable replacement for extracting real-time ecological information (e.g. abundance or density), but could speed up or aid manual analysis. Given the simplicity of this pipeline, many improvements (e.g. classifier training, incorporating associated metadata (e.g. pan-tilt, zoom) into a tracker) could be implemented to increase the accuracy in attaining quantifiable ecological measurements and reduce false detections in order to move towards full automation (Katija et al., 2021b; Lütjens and Sternberg, 2021).

2.4.2 Impact of classifiers performance with variations in training method

In this study, the overall performance scores for training the classifiers and their real-time recall of presence-absence areas of *S. fragilissima*, was affected the most by the classifiers architecture (YOLOv3 vs. YOLOv4). The pre-processing steps applied to the training imagery dataset (varying resolution, brightness and zoom vs. none) was the next most impactful. Performance scores were the least impacted with regards to the training approach used (TS vs. TL). This may be a result of the simple structural morphology and low variability in the appearance of *S. fragilissima*, concurring with the findings of Piechaud *et al.* (2019) who classified *S. fragilissima* in an AUV imagery dataset. They found *S. fragilissima* was detected well, despite it often being covered in a light dusting of sediment. Pan (2020) suggested that lower level features learnt for detection from pre-training on large generic training image datasets could suffice for objects with simple morphology or similarly shaped to the classes in the generic dataset (Guo *et al.,* 2021). Research suggests there are many advantages of using TL as it often achieves the same classification accuracies as manual efforts and TS (e.g. Oztel *et al.,* 2019), it requires a lower number of training images for good performance (Shu, 2019), is less computationally expensive (meaning it requires less advanced hardware, Pan, 2020) and, is more simple to perform in practice for non-experts, making the pipeline more reproducible.

In this study YOLOv3 attained the majority of the top-three scoring performance metrics indicating that when choosing an architecture to train for a given task, newer, faster (FPS) and mathematically more complex is not always best (Goodwin et al., 2021). In fact classifier performance is largely data-driven (Christin *et al.,* 2019), where the number of classes and class complexity are key influencing factors (Favret and Sieracki, 2016; Piechaud *et al.,* 2019). Training dataset size can imped overfitting with respect to the architecture complexity, as well as speed of classifiers performance in real-time (FPS). Hardware for running the classifier is influential in their suitability and performance for a given task (Khan *et al.,* 2020). In this study, where only a single class is considered and the frame-rate for real-time detection is sufficient, a more advanced architecture may be unnecessary.

Pre-processing of the training image dataset using the VIAME API (increasing brightness and multi-level resolution – processed image dataset) consistently attained the highest mAP and recall scores, with only marginal improvement in the other three performance metrics (F1 score, precision and accuracy) when using the ’ unprocessed-image’ dataset. Neither effects are quantified with respect to classifier performance. However, in combination they seem to gain marginal improvements in classifiers retention of true positives and, trains more steadily (Appendix A2.4). It is thought that the increasing of brightness could potentially enhance the quality and features of the target class in the training images as illumination in the images was uneven, making organisms in the extremities and background less distinct (Verhaegen *et al.,* 2021). However, a study by Hou *et al.* (2020) assessing the effect of increasing the brightness and rotation of training imagery for detecting the giant panda (*Ailuropoda melanoleuca*), found a slight decrease in accuracy when training with only brighter images (1-3%), this could explain the slight drop in accuracy (3%) seen in this study. Other studies have noted classifier robustness is more likely to be increased as a result of training with images experiencing a combination of dark and light (e.g. Taylor and Nitschke, 2018; Shorten and Khoshgoftaar, 2019; Hou *et al.*, 2020; Enkvetchakul and Surinta, 2021); particularly when deploying on real-time video in the field, due to often high variability of illumination over a changing scene.

As *S. fragilissima* only reaches a maximum diameter of 10-20cms (Jun and Taheri-Araghi, 2015), training classifiers solely on the ‘unprocessed’ image dataset images could lead to drawbacks with scaling as the model receives fewer pixels of information to build a picture of the target class (*S. fragilissima*) which could impede predictions made on unseen data. It may also mean the classifiers learns wrong key identifying features, for example size (Bureš *et al.,* 2021). It is noted that the consistency of size of the target class in the training image dataset can often become a feature CNNs use to make a positive detection. This is particularly the case when training a YOLO network, where performances on detecting objects in unseen data are often impacted by the area the object occupies in the image, whereby object size in the training image must be similar to the data it is being applied to (Jeong *et al.,* 2018). During the *in-situ* ROV deployment this became evident, as the V3TS classifier locked onto *S. fragilissima* much better when the camera was zoomed in than when on the default level of zoom. This indicates that the classifier may have learnt more features associated with *S. fragilissima* in the full-resolution segments (e.g. size), and so when zoomed in on unseen data the features resemble more closely those learned during training. Since these effects were not quantified this is speculation based on observations made during the study.

Regardless, numerous studies have supported and highlighted the influence pre-processing can have on classifier performance, and suggests that pre-processing of imagery may be the most optimal and manageable way for non-experts to enhance feature extraction capabilities of the classifier without having to re-design classifiers architectures (Lumini and Nanni, 2019). Overall, in this study, training and real-time performance clearly was impacted by pre-processing, however the degree of these effects could be considered marginal. Further consideration of what pre-processing steps to take would be advantageous in future studies to understand the full effects of how it could enhance features of the target class to improve classifiers performance on unseen data (Ditria *et al.,* 2021).

2.4.3 Classifier performance estimating individual counts of *S. fragilissima*

In ecology, informative statistics gathered from video datasets may include species diversity, number, or behaviour. In the context of VME there is a need to understand the density of VME indicator taxa in defining the VME (ICES, 2016). Thus, it is important to extract count data in addition to detecting presence. The V3TS classifier performed well at recalling the number of *S. fragilissima* compared to manual counting, attaining a significant and good correlation of 73%. However, classifier performance was notably worse for frames with higher numbers of manually counted *S. fragilissima*, indicating possible limitations in the classifiers ability to count individuals in frames of dense aggregations. The approach used in this study was limited by the lack of incorporation of a robust, CV tracker. Individual *S. fragilissima* may occur in multiple frames for an extended period of time, meaning with this approach there are challenges with double counting of individuals. A tracker follows individuals through frames mitigating this double counting problem. Lütjens and Sternberg (2021) describe a method that uses deep-learning networks to detect and classify glass sponges, soft coral and brittle stars, with the incorporation of a robust tracker to automate the counting. They attained relatively good count errors, ranging only from 7-20%. However, the application of trackers are still restricted to short observation periods, whereby the tracker starts to be impeded by high levels of occlusion, overlap and changes in object morphology with respect to the objects movements and camera angle throughout the video (Kandimalla *et al.,* 2022). Katija *et al.* (2021b) has tried to address the issues of applying trackers for longer observation periods by creating a three-layered robust tracker. This successfully tracked a single target siphonophore (*Lychnagalma* sp.) for a +5 hour observation period. However, the effects of occlusion, overlap and background complexity are less of an obstacle in the pelagic realm, and whilst it was stated the tracker is applicable to benthic habitats, the accuracy of tracking is expected to diminish over longer observation periods Katija *et al.* (2021b), thus reducing its ability to be applied to count data. Therefore, further development is needed to ensure that pipelines are robust enough to fully and accurately automate detecting, classifying and counting of target species to augment and in some cases replace current manual efforts; particularly with respect to benthic habitats and realistic observation periods in order to allow these tools to collect ecologically meaningful data in real-time at the scale required.

2.5 Conclusion

The key findings in this study indicate consideration of CNN architecture, followed by the pre-processing and then training approach (TS vs. TL), are important factors when training a classifier. It also suggests that the application of a trained YOLOv3 classifier to a ROV livestream in the field is possible and achieves good performances in terms of detecting presence-absence areas of the target taxon, whilst counts offer promise for future applications of DL and CV techniques to live collection of ecological data. Overall, the pipeline can deliver faster interpretation of large video datasets and aid in augmenting data to improve further classifier training. However, it has not achieved full automation with respect to collecting reliable real-time ecological data (i.e. count data) desired for informing management and conservation decisions. An incorporation of manual checking of the classifiers outputs are still required to eliminate false detections and double counts. However, these issues could be addressed or improved by incorporation of a robust CV tracker to mitigate double counting and, optimising the classifiers performance during training to increase its robustness over a wider range scenarios (e.g. change in organism appearance, size, illumination).

While this does not immediately address all the challenges seen with large datasets in marine ecology, it provides an initial and realistic baseline that could be adopted and improved on by ecologists to help them begin using these tools to aid their research. The development of a fully automated system to collect and analyse videos to extract real-time ecological data on observation platforms is gaining momentum (Katija *et al.,* 2021b). With collaboration between computer scientists, ecologists and engineers, the creation of such a tool could reduce the workload on more monotonous tasks, and increase the pace of studying the natural world providing the evidence to support the development of effective management and conservation measures.

Chapter 3: Closing discussion: limitations, future work and the wider scientific community

AI technologies are becoming increasingly popular throughout the world for various applications. As a result, a huge scope of knowledge, practical usage and approaches are now available, but the field remains largely impenetrable for non-experts such as marine ecologists. As discussed in **Chapter 1,** classifiers mathematical complexity, alongside the increase in popularity across a multitude of fields, has driven the production of standardised classifiers and training datasets in order to increase the useability of this tool for detecting and classifying objects within large image datasets. Application in the field of marine ecology has found CNNs are now able to achieve replicable and accurate results compared to humans. However, implementation of these tools has largely focused on detecting and classifying objects within datasets (e.g. production of presence-absence data). There is a desire to move to extraction of quantitative ecological data, such as abundance or density. This is especially the case for those working in the deep sea, where a significant amount of data are collected via image and video transects using ROVs. The aims of this thesis were to investigate what has been achieved to-date in the application of DL to the field of marine ecology (**Chapter 1**) and how we might apply DL to speed up the rate of data collection, ultimately to provide data to inform sustainable management of the deep sea (**Chapter 2**).

The major findings of this thesis (**Chapter 2**) are 1) that it is possible for non-experts to use these technologies *in-situ* and at relatively low-cost and 2) there are a number of approaches that can be taken to improve classifier performance (**Chapter 1 and 2**) before deploying it in the field. This includes how it is trained, what it is trained with and what architectures may provide benefits for a given task. From this thesis and other research it can be concluded that, given that performance of classifiers trained on a customised dataset (TS) does not vary massively from that achieved using classifiers trained on a generic dataset (TL), TL may be the most useful approach to train these classifiers. This benefits ecologists as it requires less GPU time to train, meaning hardware requirements are less expensive and training requires fewer example images. This study, in line with the wider scientific community, also finds that pre-processing of imagery is important to optimise classifier performance (Zhuang *et al.,* 2017; Jeong *et al.,* 2018; Riaboff *et al.,* 2019; Shahriar and Li, 2020) and consideration of the CNN architecture used is also key (Malde *et al.,* 2020).

This study has highlighted the potential application of these technologies within the field of deep-sea ecology specifically in reducing costs through reducing time spent on manual processing of image-based data. However, major shortfalls with respect to data quality were observed, and data collected required a fair amount of post-processing, particularly for count data. At present DL is limited to aiding manual processing efforts, rather than eradicating them.

It must be noted that this thesis was restricted in terms of time available to optimise classifiers performance and to implement a more effective counting pipeline. The field of deep-learning is moving rapidly and during this study period alone (1-year) new tools (e.g. robust CV trackers) were developed and incorporated into pipelines in order to deal with retaining accurate (avoiding double counts) and automated counting of target classes over long periods of observation (Katija *et al.,* 2021b). Yet, it remains the case that no study has fully achieved a robust method to incorporate detecting, classifying and counting during real-time deployment on observational platforms to attain real-time ecological data.

To make this a more accessible option for ecologists, and to allow comparability between approaches, further research could benefit from using appropriate and standardised statistical metrics to describe the performance of the classier for assessing real-time detecting, tracking and counting (e.g. Multi-Object tracking Accuracy (MOTA), Multi-Object Tracking Precision (MOTP), Higher Order Tracking Accuracy (HOTA), Wang *et al.,* 2020; Luiten *et al.,* 2021). Thus, allowing for better comparison and optimisation of classifiers. To help improve trackers to automated counting during deployment whilst preventing double counts, cameras system metadata (e.g. pan-tilt, zoom, speed) could be incorporated into the tracker to help with sudden changes occurring in the field of view (FOV) that could lead to miscalculations or bias in the ecological data (Katija *et al.,* 2021b). These trackers would also ideally be able to tell individuals that return to the frame in order to ensure no double counts occur. In addition, during field expeditions the goal is often to attain data on more than one target species with varying ranges of size. A pipeline that could deal with classifying and tracking multiple target species is required, particularly for trying to count in highly heterogenous habitats for realistic survey periods (Katija *et al.,* 2021b).

Understanding how classifiers interpret features of target classes during the training process could help in the development and / or training of better classifiers. For example, what features are learnt and what are made redundant? How similar are these features to the key features a human annotator would assign? Which features do not vary greatly over various changes in optics or at the intra-species level? Answering these questions could help guide researchers to develop pre-processing steps that could help highlight these features for improved detection and more consistent tracking (Durden *et al.,* 2021).

In the deep-sea, the integration of these technologies to interpret data during the deployment of observational platforms could help to quantify ecological metrics in real-time, allow for more targeted exploration, and potentially reduce costs through reducing post manual processing of data. This would allow more time and funds to be spent on more innovative sciences rather than repetitive, tedious tasks. In addition, with more collaboration between ecologists, computer scientists and engineers pipelines could be developed for various other observational platforms used throughout other marine and terrestrial ecology fields (Goodwin *et al.,* 2021; Tuia *et al.,* 2022). Analysing data in this manner ensures the maximum amount of data collected is being interpreted into useful ecological metrics. Attaining it in real-time means data can be more rapidly delivered into knowledge pathways at the pace required to respond to rapid fluctuations induced by climate change, and to more rapidly advance understanding of data poor areas such as the deep sea, in the face of increasing anthropogenic use.

Appendices

4.1 Appendix A2:

Appendix accompanying **Chapter 2**.

4.1.1 A2.1: Training images collected

This document outlines the number of frame grabs (or images) annotated from ROV video transects collected in three locations; ADS, NRB and RB over 3 different depths. Each transect was either manually annotated every 60 seconds or every 20 and 60 seconds.

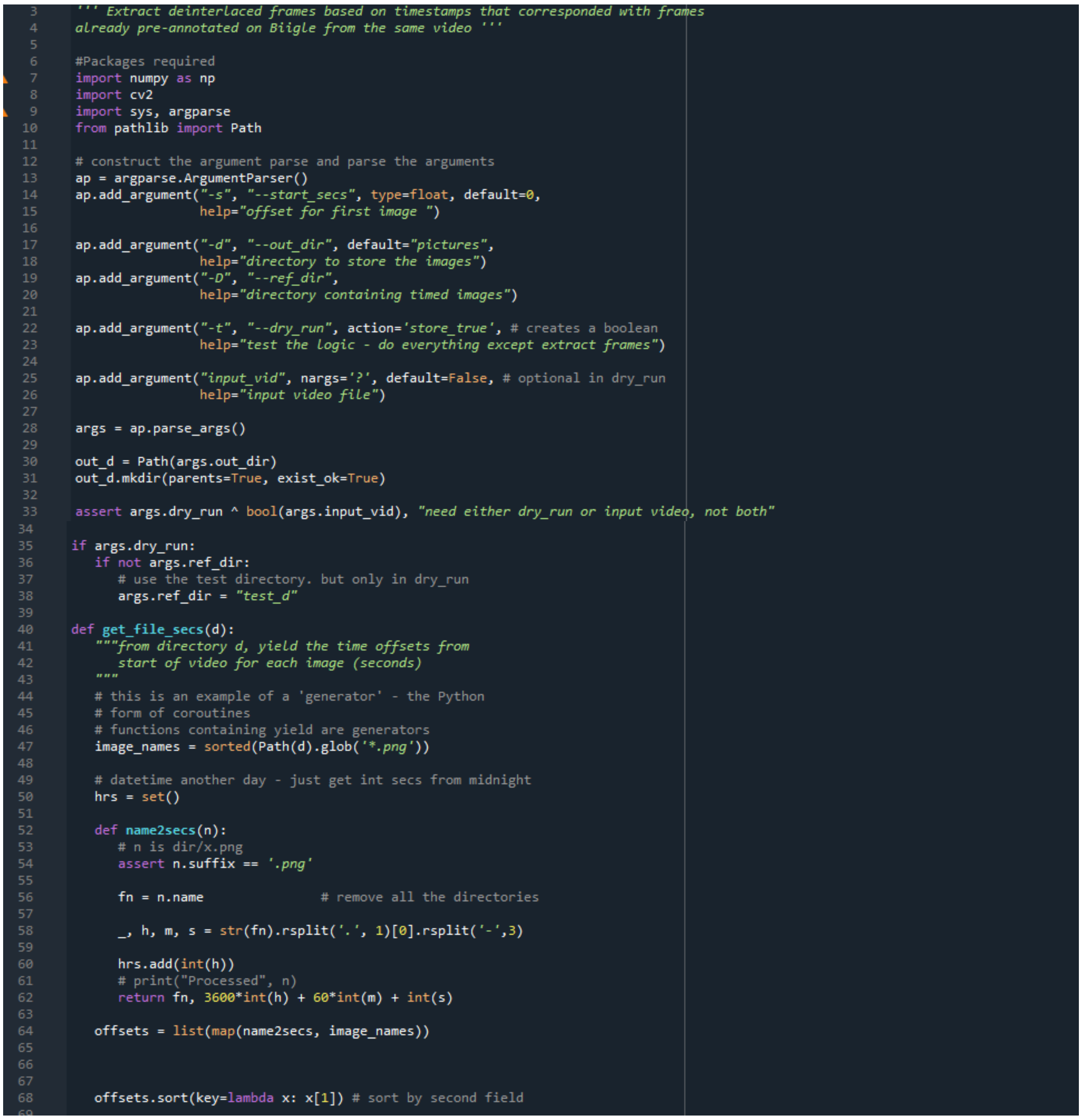
A2.1 Table 1| Number of images (frames) annotated for classifier training with their respective locations (ADS, NRB and RB), depths the data were collected, and annotation time intervals for each transect.

Locations	No. of Transects	Annotation Time Interval (seconds)	Depth (meters)	Total No. of Frames Annotated
ADS	3	60	1200	194
	3	20 and 60	800	522
	3	X2 60 and x1 20 and 60	500	360
RB	3	60	1200	163
NRB	3	20 and 60	1200	852
Total				**2,091**

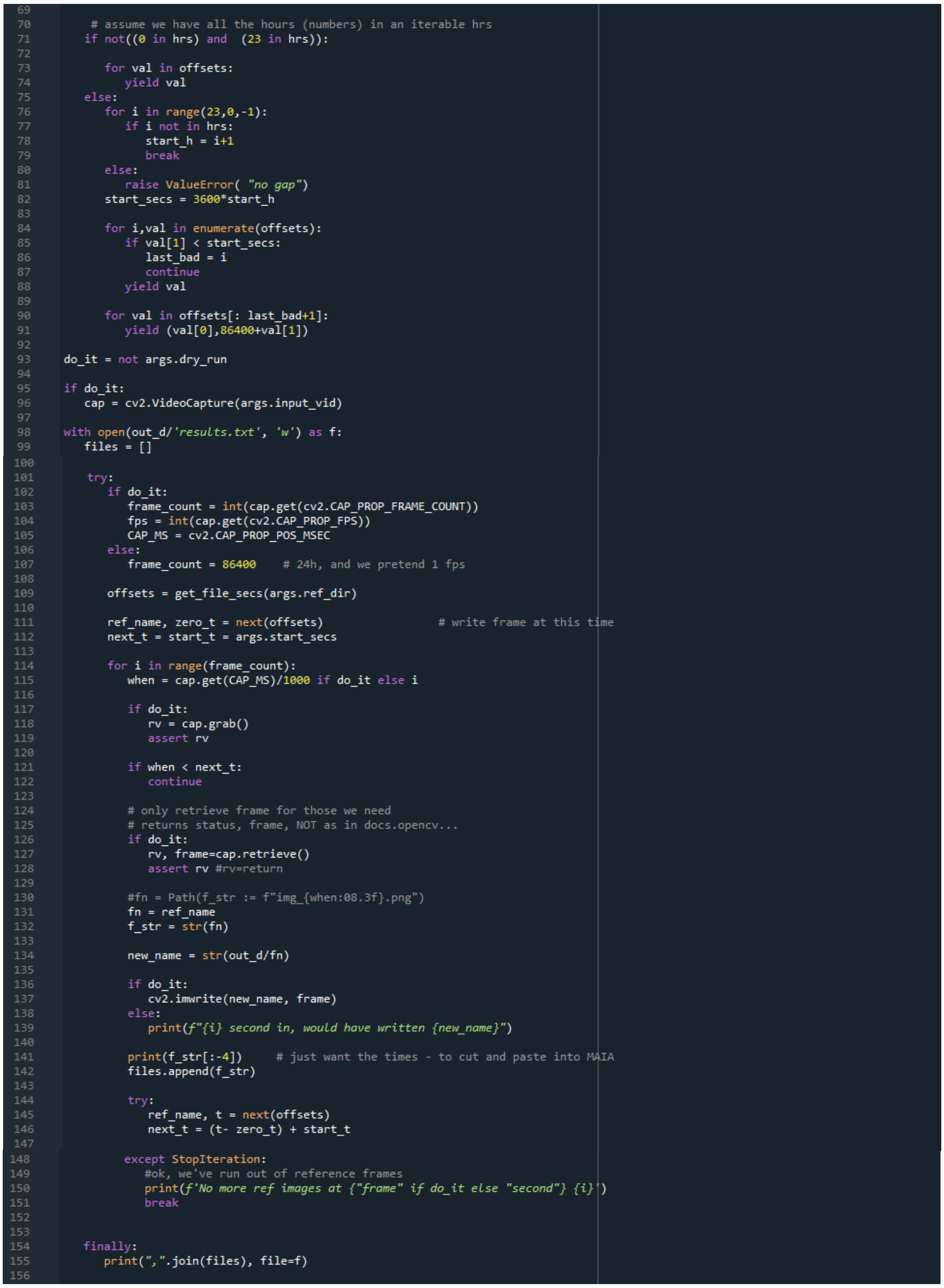
4.1.2 A2.2: Python codes used for pre-processing of training datasets

This document describes the python scripts used for pre-processing outlined in the methods (section 2.2.3).

1. '**extract_frames_N.py**': Extract frames from a deinterlaced video to improve training image quality by reducing interlacing.

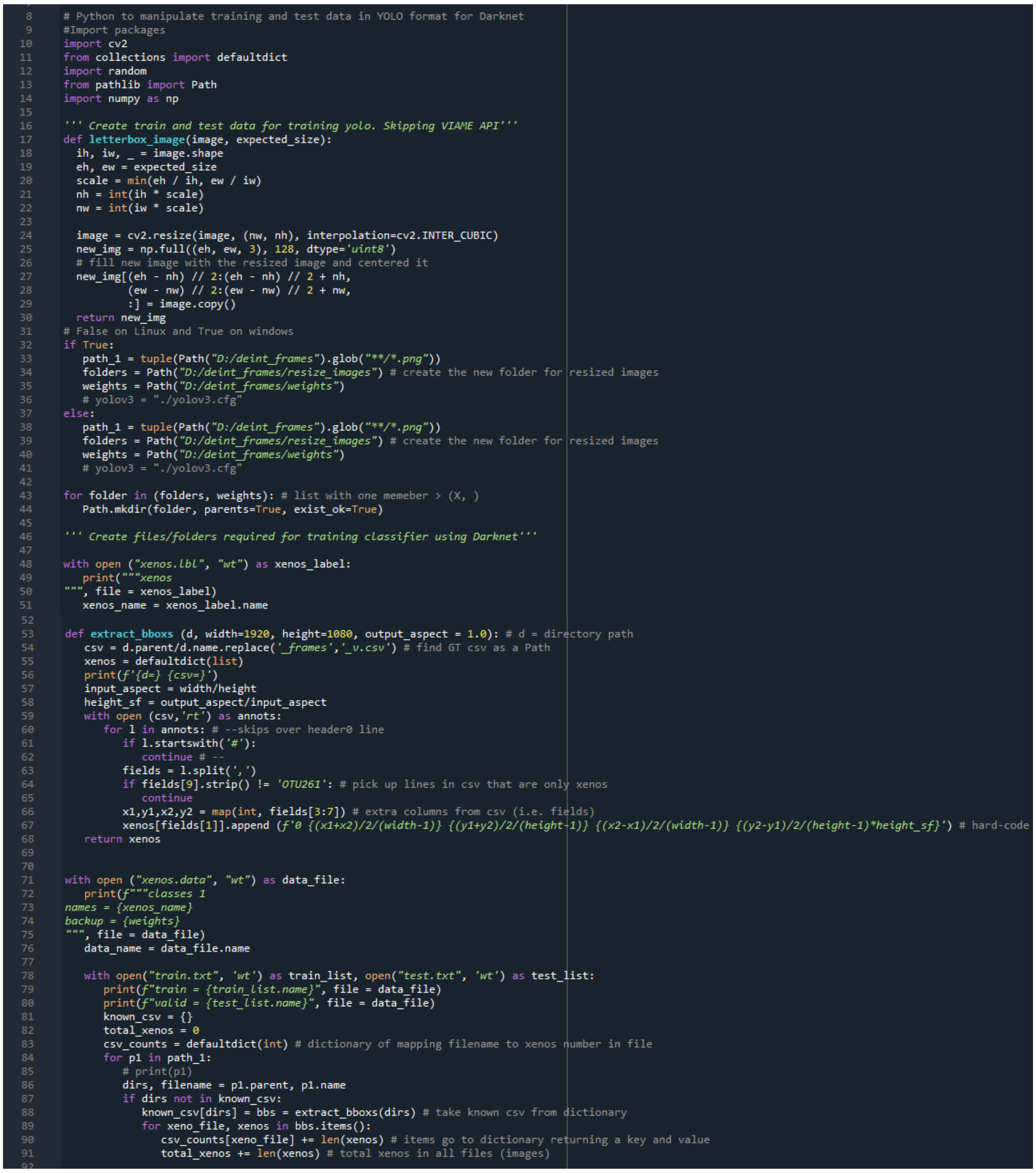


```python
''' Extract deinterlaced frames based on timestamps that corresponded with frames
already pre-annotated on Biigle from the same video '''

#Packages required
import numpy as np
import cv2
import sys, argparse
from pathlib import Path

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-s", "--start_secs", type=float, default=0,
                help="offset for first image ")

ap.add_argument("-d", "--out_dir", default="pictures",
                help="directory to store the images")
ap.add_argument("-D", "--ref_dir",
                help="directory containing timed images")

ap.add_argument("-t", "--dry_run", action='store_true', # creates a boolean
                help="test the logic - do everything except extract frames")

ap.add_argument("input_vid", nargs='?', default=False, # optional in dry_run
                help="input video file")

args = ap.parse_args()

out_d = Path(args.out_dir)
out_d.mkdir(parents=True, exist_ok=True)

assert args.dry_run ^ bool(args.input_vid), "need either dry_run or input video, not both"

if args.dry_run:
   if not args.ref_dir:
      # use the test directory. but only in dry_run
      args.ref_dir = "test_d"

def get_file_secs(d):
   """from directory d, yield the time offsets from
      start of video for each image (seconds)
   """
   # this is an example of a 'generator' - the Python
   # form of coroutines
   # functions containing yield are generators
   image_names = sorted(Path(d).glob('*.png'))

   # datetime another day - just get int secs from midnight
   hrs = set()

   def name2secs(n):
      # n is dir/x.png
      assert n.suffix == '.png'

      fn = n.name                 # remove all the directories

      _, h, m, s = str(fn).rsplit('.', 1)[0].rsplit('-',3)

      hrs.add(int(h))
      # print("Processed", n)
      return fn, 3600*int(h) + 60*int(m) + int(s)

   offsets = list(map(name2secs, image_names))



   offsets.sort(key=lambda x: x[1]) # sort by second field
```

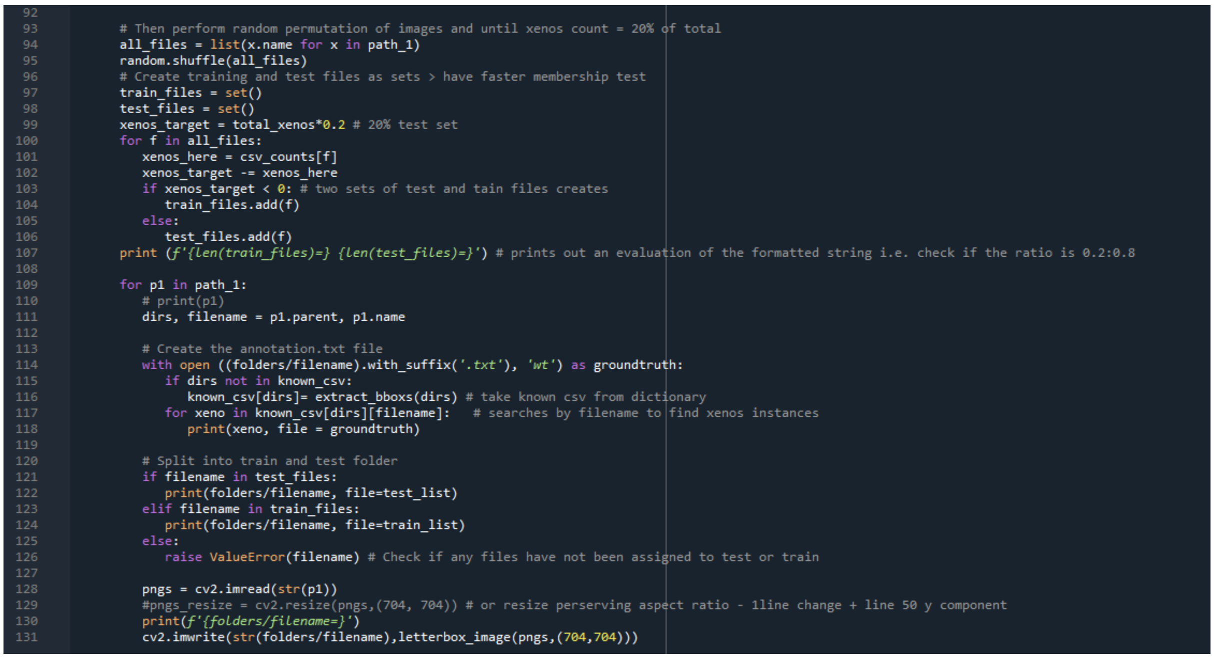


```python
    # assume we have all the hours (numbers) in an iterable hrs
   if not((0 in hrs) and  (23 in hrs)):

      for val in offsets:
         yield val
   else:
      for i in range(23,0,-1):
         if i not in hrs:
            start_h = i+1
            break
      else:
         raise ValueError( "no gap")
      start_secs = 3600*start_h

      for i,val in enumerate(offsets):
         if val[1] < start_secs:
            last_bad = i
            continue
         yield val

      for val in offsets[: last_bad+1]:
         yield (val[0],86400+val[1])

do_it = not args.dry_run

if do_it:
   cap = cv2.VideoCapture(args.input_vid)

with open(out_d/'results.txt', 'w') as f:
   files = []

   try:
      if do_it:
         frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
         fps = int(cap.get(cv2.CAP_PROP_FPS))
         CAP_MS = cv2.CAP_PROP_POS_MSEC
      else:
         frame_count = 86400    # 24h, and we pretend 1 fps

      offsets = get_file_secs(args.ref_dir)

      ref_name, zero_t = next(offsets)                 # write frame at this time
      next_t = start_t = args.start_secs

      for i in range(frame_count):
         when = cap.get(CAP_MS)/1000 if do_it else i

         if do_it:
            rv = cap.grab()
            assert rv

         if when < next_t:
            continue

         # only retrieve frame for those we need
         # returns status, frame, NOT as in docs.opencv...
         if do_it:
            rv, frame=cap.retrieve()
            assert rv #rv=return

         #fn = Path(f_str := f"img_{when:08.3f}.png")
         fn = ref_name
         f_str = str(fn)

         new_name = str(out_d/fn)

         if do_it:
            cv2.imwrite(new_name, frame)
         else:
            print(f"{i} second in, would have written {new_name}")

         print(f_str[:-4])     # just want the times - to cut and paste into MAIA
         files.append(f_str)

         try:
            ref_name, t = next(offsets)
            next_t = (t- zero_t) + start_t

         except StopIteration:
            #ok, we've run out of reference frames
            print(f'No more ref images at {"frame" if do_it else "second"} {i}')
            break


   finally:
      print(",".join(files), file=f)
```

2. ‘**skip_viame.py**’: Takes full-resolution (1080p) training images and applies letterboxing effect (same as VIAME API) in order to allow the CNN to process the whole image (as CNNs required a square pixel matrix for training). It bypasses the manipulation of brightness, resolution and zoom taken by the VIAME API, in order to assess the effects this has on classifier performance.

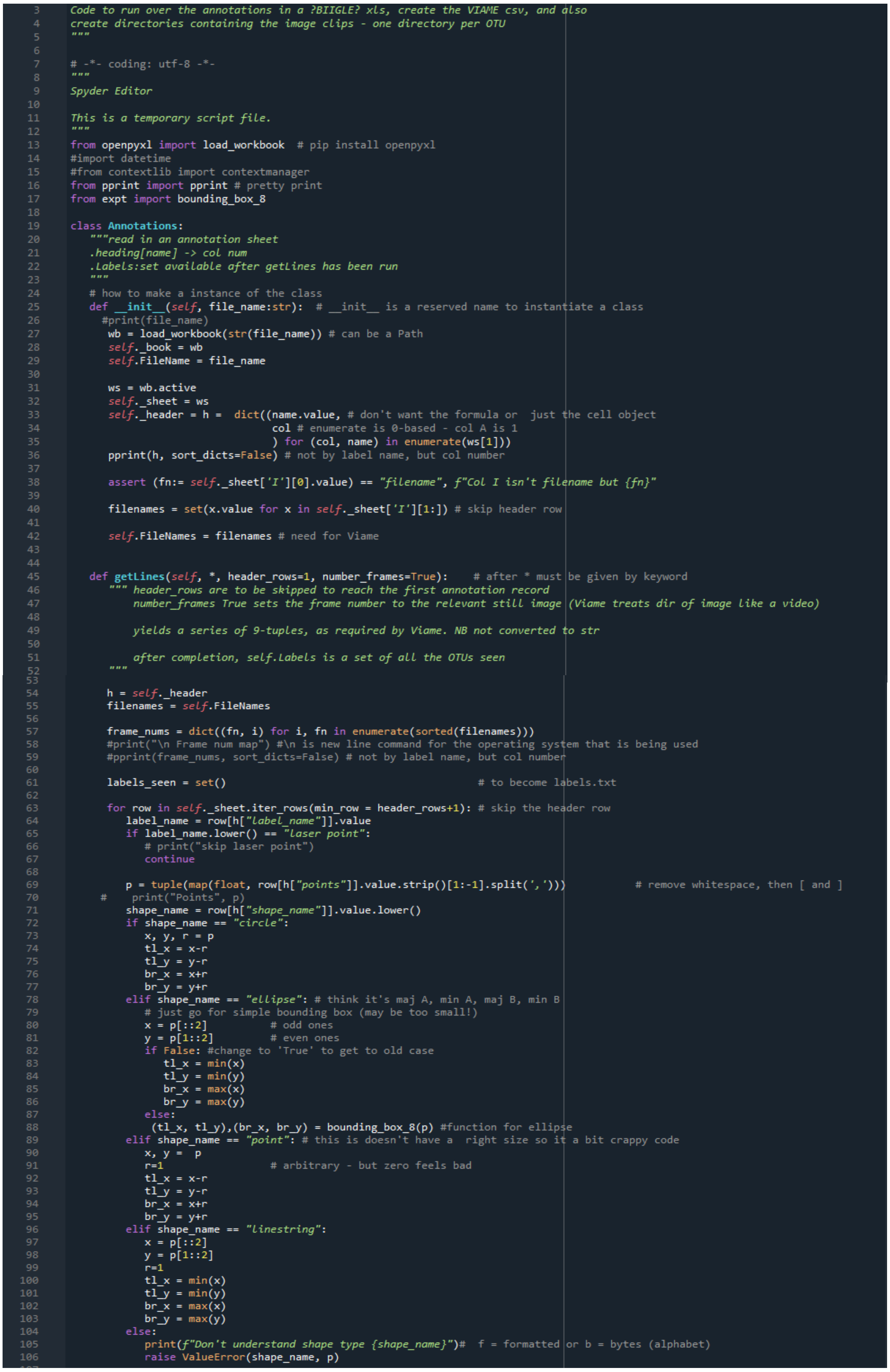


```python
# Python to manipulate training and test data in YOLO format for Darknet
#Import packages
import cv2
from collections import defaultdict
import random
from pathlib import Path
import numpy as np

''' Create train and test data for training yolo. Skipping VIAME API'''
def letterbox_image(image, expected_size):
  ih, iw, _ = image.shape
  eh, ew = expected_size
  scale = min(eh / ih, ew / iw)
  nh = int(ih * scale)
  nw = int(iw * scale)

  image = cv2.resize(image, (nw, nh), interpolation=cv2.INTER_CUBIC)
  new_img = np.full((eh, ew, 3), 128, dtype='uint8')
  # fill new image with the resized image and centered it
  new_img[(eh - nh) // 2:(eh - nh) // 2 + nh,
          (ew - nw) // 2:(ew - nw) // 2 + nw,
          :] = image.copy()
  return new_img
# False on Linux and True on windows
if True:
   path_1 = tuple(Path("D:/deint_frames").glob("**/*.png"))
   folders = Path("D:/deint_frames/resize_images") # create the new folder for resized images
   weights = Path("D:/deint_frames/weights")
   # yolov3 = "./yolov3.cfg"
else:
   path_1 = tuple(Path("D:/deint_frames").glob("**/*.png"))
   folders = Path("D:/deint_frames/resize_images") # create the new folder for resized images
   weights = Path("D:/deint_frames/weights")
   # yolov3 = "./yolov3.cfg"

for folder in (folders, weights): # list with one memeber > (X, )
   Path.mkdir(folder, parents=True, exist_ok=True)

''' Create files/folders required for training classifier using Darknet'''

with open ("xenos.lbl", "wt") as xenos_label:
   print("""xenos
""", file = xenos_label)
   xenos_name = xenos_label.name

def extract_bboxs (d, width=1920, height=1080, output_aspect = 1.0): # d = directory path
   csv = d.parent/d.name.replace('_frames','_v.csv') # find GT csv as a Path
   xenos = defaultdict(list)
   print(f'{d=} {csv=}')
   input_aspect = width/height
   height_sf = output_aspect/input_aspect
   with open (csv,'rt') as annots:
      for l in annots: # --skips over header0 line
         if l.startswith('#'):
            continue # --
         fields = l.split(',')
         if fields[9].strip() != 'OTU261': # pick up lines in csv that are only xenos
            continue
         x1,y1,x2,y2 = map(int, fields[3:7]) # extra columns from csv (i.e. fields)
         xenos[fields[1]].append (f'0 {(x1+x2)/2/(width-1)} {(y1+y2)/2/(height-1)} {(x2-x1)/2/(width-1)} {(y2-y1)/2/(height-1)*height_sf}') # hard-code
   return xenos


with open ("xenos.data", "wt") as data_file:
   print(f"""classes 1
names = {xenos_name}
backup = {weights}
""", file = data_file)
   data_name = data_file.name

   with open("train.txt", 'wt') as train_list, open("test.txt", 'wt') as test_list:
      print(f"train = {train_list.name}", file = data_file)
      print(f"valid = {test_list.name}", file = data_file)
      known_csv = {}
      total_xenos = 0
      csv_counts = defaultdict(int) # dictionary of mapping filename to xenos number in file
      for p1 in path_1:
         # print(p1)
         dirs, filename = p1.parent, p1.name
         if dirs not in known_csv:
            known_csv[dirs] = bbs = extract_bboxs(dirs) # take known csv from dictionary
            for xeno_file, xenos in bbs.items():
               csv_counts[xeno_file] += len(xenos) # items go to dictionary returning a key and value
               total_xenos += len(xenos) # total xenos in all files (images)
```

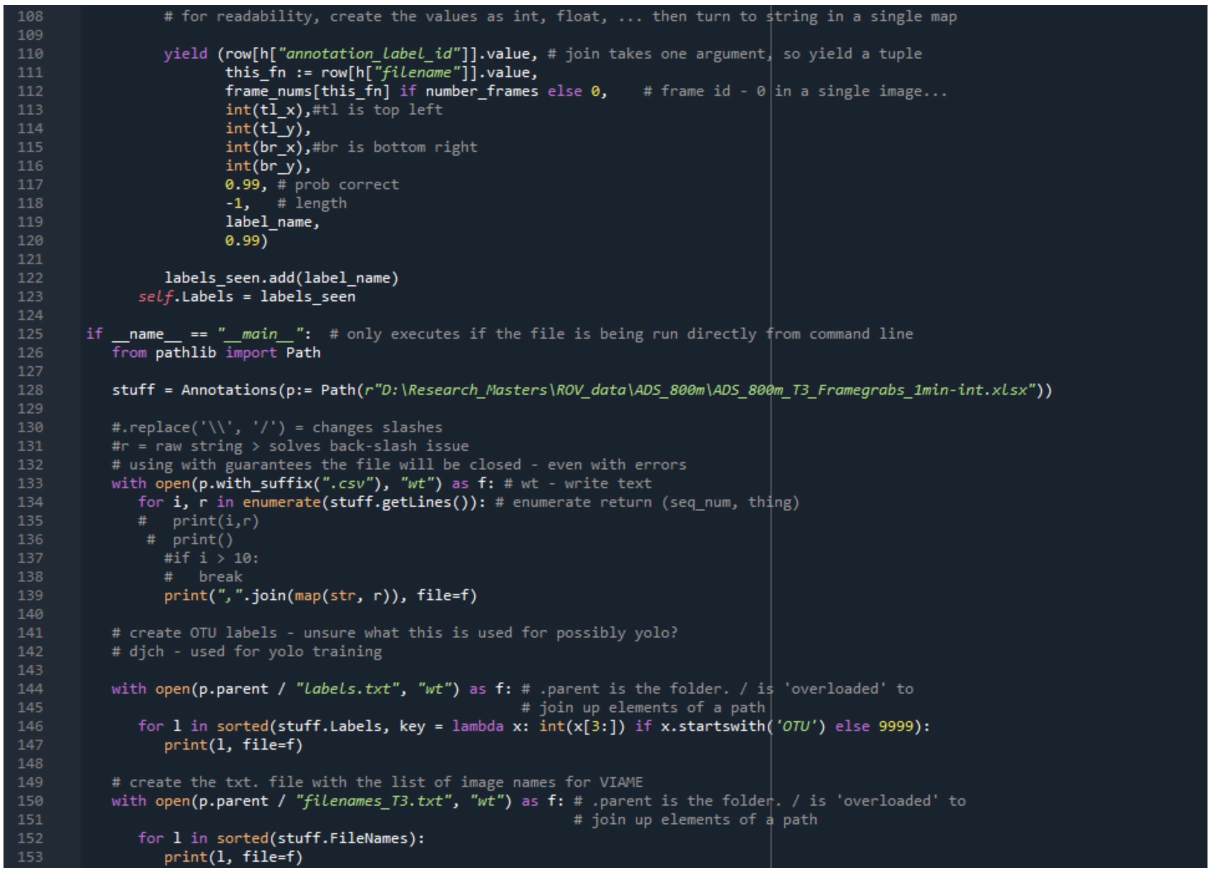


```python
      # Then perform random permutation of images and until xenos count = 20% of total
      all_files = list(x.name for x in path_1)
      random.shuffle(all_files)
      # Create training and test files as sets > have faster membership test
      train_files = set()
      test_files = set()
      xenos_target = total_xenos*0.2 # 20% test set
      for f in all_files:
         xenos_here = csv_counts[f]
         xenos_target -= xenos_here
         if xenos_target < 0: # two sets of test and tain files creates
            train_files.add(f)
         else:
            test_files.add(f)
      print (f'{len(train_files)=} {len(test_files)=}') # prints out an evaluation of the formatted string i.e. check if the ratio is 0.2:0.8

      for p1 in path_1:
         # print(p1)
         dirs, filename = p1.parent, p1.name

         # Create the annotation.txt file
         with open ((folders/filename).with_suffix('.txt'), 'wt') as groundtruth:
            if dirs not in known_csv:
               known_csv[dirs]= extract_bboxs(dirs) # take known csv from dictionary
            for xeno in known_csv[dirs][filename]:   # searches by filename to find xenos instances
               print(xeno, file = groundtruth)

         # Split into train and test folder
         if filename in test_files:
            print(folders/filename, file=test_list)
         elif filename in train_files:
            print(folders/filename, file=train_list)
         else:
            raise ValueError(filename) # Check if any files have not been assigned to test or train

         pngs = cv2.imread(str(p1))
         #pngs_resize = cv2.resize(pngs,(704, 704)) # or resize perserving aspect ratio - 1line change + line 50 y component
         print(f'{folders/filename=}')
         cv2.imwrite(str(folders/filename),letterbox_image(pngs,(704,704)))
```

3. **‘xls2csv.py’**: Converts the already tabulated manual annotations from the BIIGLE API (created prior to this study), for the corresponding deinterlaced training imagery, into a VIAME annotation csv. This is required to create the outputted files and folders in a YOLO format for training.



```python
Code to run over the annotations in a ?BIIGLE? xls, create the VIAME csv, and also
create directories containing the image clips - one directory per OTU
"""

# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""
from openpyxl import load_workbook  # pip install openpyxl
#import datetime
#from contextlib import contextmanager
from pprint import pprint # pretty print
from expt import bounding_box_8

class Annotations:
   """read in an annotation sheet
   .heading[name] -> col num
   .Labels:set available after getLines has been run
   """
   # how to make a instance of the class
   def __init__(self, file_name:str):  # __init__ is a reserved name to instantiate a class
     #print(file_name)
      wb = load_workbook(str(file_name)) # can be a Path
      self._book = wb
      self.FileName = file_name

      ws = wb.active
      self._sheet = ws
      self._header = h =  dict((name.value, # don't want the formula or  just the cell object
                                col # enumerate is 0-based - col A is 1
                                ) for (col, name) in enumerate(ws[1]))
      pprint(h, sort_dicts=False) # not by label name, but col number

      assert (fn:= self._sheet['I'][0].value) == "filename", f"Col I isn't filename but {fn}"

      filenames = set(x.value for x in self._sheet['I'][1:]) # skip header row

      self.FileNames = filenames # need for Viame


   def getLines(self, *, header_rows=1, number_frames=True):    # after * must be given by keyword
      """ header_rows are to be skipped to reach the first annotation record
          number_frames True sets the frame number to the relevant still image (Viame treats dir of image like a video)

          yields a series of 9-tuples, as required by Viame. NB not converted to str

          after completion, self.Labels is a set of all the OTUs seen
      """

      h = self._header
      filenames = self.FileNames

      frame_nums = dict((fn, i) for i, fn in enumerate(sorted(filenames)))
      #print("\n Frame num map") #\n is new line command for the operating system that is being used
      #pprint(frame_nums, sort_dicts=False) # not by label name, but col number

      labels_seen = set()                                        # to become labels.txt

      for row in self._sheet.iter_rows(min_row = header_rows+1): # skip the header row
         label_name = row[h["label_name"]].value
         if label_name.lower() == "laser point":
            # print("skip laser point")
            continue

         p = tuple(map(float, row[h["points"]].value.strip()[1:-1].split(',')))           # remove whitespace, then [ and ]
     #    print("Points", p)
         shape_name = row[h["shape_name"]].value.lower()
         if shape_name == "circle":
            x, y, r = p
            tl_x = x-r
            tl_y = y-r
            br_x = x+r
            br_y = y+r
         elif shape_name == "ellipse": # think it's maj A, min A, maj B, min B
            # just go for simple bounding box (may be too small!)
            x = p[::2]          # odd ones
            y = p[1::2]         # even ones
            if False: #change to 'True' to get to old case
               tl_x = min(x)
               tl_y = min(y)
               br_x = max(x)
               br_y = max(y)
            else:
             (tl_x, tl_y),(br_x, br_y) = bounding_box_8(p) #function for ellipse
         elif shape_name == "point": # this is doesn't have a  right size so it a bit crappy code
            x, y =  p
            r=1                 # arbitrary - but zero feels bad
            tl_x = x-r
            tl_y = y-r
            br_x = x+r
            br_y = y+r
         elif shape_name == "linestring":
            x = p[::2]
            y = p[1::2]
            r=1
            tl_x = min(x)
            tl_y = min(y)
            br_x = max(x)
            br_y = max(y)
         else:
            print(f"Don't understand shape type {shape_name}")#  f = formatted or b = bytes (alphabet)
            raise ValueError(shape_name, p)
```



```python
         # for readability, create the values as int, float, ... then turn to string in a single map

         yield (row[h["annotation_label_id"]].value, # join takes one argument, so yield a tuple
                this_fn := row[h["filename"]].value,
                frame_nums[this_fn] if number_frames else 0,    # frame id - 0 in a single image...
                int(tl_x),#tl is top left
                int(tl_y),
                int(br_x),#br is bottom right
                int(br_y),
                0.99, # prob correct
                -1,   # length
                label_name,
                0.99)

         labels_seen.add(label_name)
      self.Labels = labels_seen

if __name__ == "__main__":  # only executes if the file is being run directly from command line
   from pathlib import Path

   stuff = Annotations(p:= Path(r"D:\Research_Masters\ROV_data\ADS_800m\ADS_800m_T3_Framegrabs_1min-int.xlsx"))

   #.replace('\\', '/') = changes slashes
   #r = raw string > solves back-slash issue
   # using with guarantees the file will be closed - even with errors
   with open(p.with_suffix(".csv"), "wt") as f: # wt - write text
      for i, r in enumerate(stuff.getLines()): # enumerate return (seq_num, thing)
      #   print(i,r)
       #  print()
         #if i > 10:
         #   break
         print(",".join(map(str, r)), file=f)

   # create OTU labels - unsure what this is used for possibly yolo?
   # djch - used for yolo training

   with open(p.parent / "labels.txt", "wt") as f: # .parent is the folder. / is 'overloaded' to
                                                  # join up elements of a path
      for l in sorted(stuff.Labels, key = lambda x: int(x[3:]) if x.startswith('OTU') else 9999):
         print(l, file=f)

   # create the txt. file with the list of image names for VIAME
   with open(p.parent / "filenames_T3.txt", "wt") as f: # .parent is the folder. / is 'overloaded' to
                                                        # join up elements of a path
      for l in sorted(stuff.FileNames):
         print(l, file=f)
```

4.1.3 A2.3: Sources for configuration and weights file

This documents where the 'off-the-shelf' classifier architectures weights files (used for storing the parameters learnt during training) and corresponding configuration files are sourced. These weights are either pre-trained on larger imagery datasets (lower level features already learned and stored in weights file), also known as TL. Or the weights have no pre-trained features, therefore all features learned and stored in the weights files are directly from the training imagery created in this study.

A2.3 Table 1| Sources for weights and configuration files for TS (*VIAME API GitHub*) and TL (*AlexeyAB GitHub*).

Classifier	Weight files		Configuration files	
	AlexeyAB GitHub	**VIAME API GitHub**	**AlexeyAB GitHub**	**VIAME API GitHub**
YOLOv3	https://pjreddie.com/media/files/yolov3-spp.weights	Install VIAME v 0.15.1 Directory: */viame/configs/pipelines/models/yolo_v3_seed.wt	https://raw.githubusercontent.com/AlexeyAB/darknet/master/cfg/yolov3-spp.cfg	Install VIAME v 0.15.1 Directory: */viame/configs/pipelines/models/yolo_train.cfg
YOLOv4	https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.conv.137	Install VIAME v 0.17.2 Directory: */viame/configs/models/yolo_seed.wt	https://raw.githubusercontent.com/AlexeyAB/darknet/master/cfg/yolov4.cfg	Install VIAME v 0.17.2 Directory: */viame/configs/models/yolo_train.cfg

4.1.4 A2.4: Supporting graphs for training of classifiers

Graphs produced by Darknet detector during training process of each classifier. They display the fluctuations in average loss value and mean Average Precision (mAP) with Intersection over Union (IoU) at 50, during the 45,000 epochs. The rate in fluctuations of mAP and the steadiness in the exponential decay of the loss function, indicates if a classifier is training well. Where, large fluctuations can indicate poor performance on unseen data.

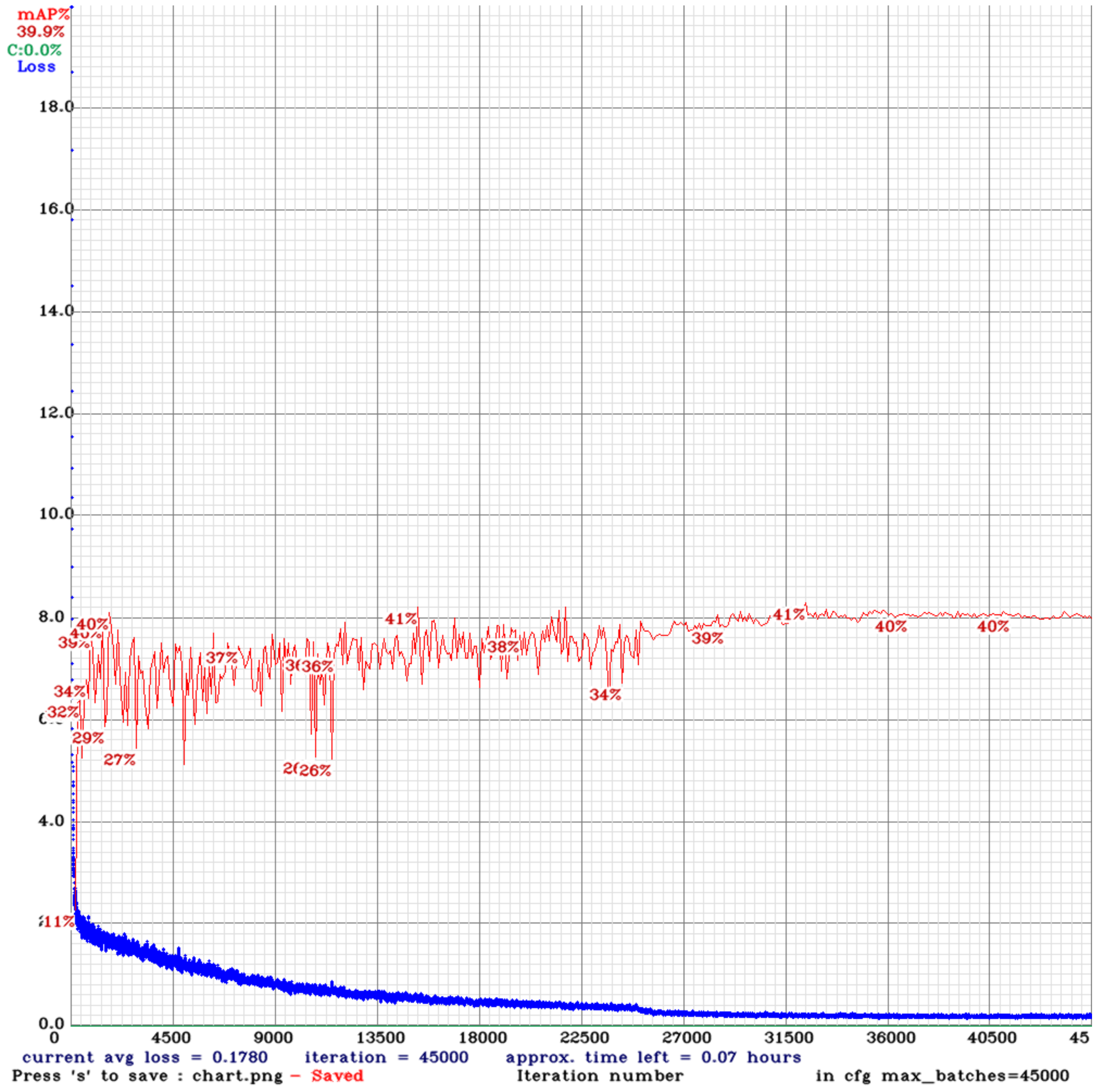


Figure 1| P3TS classifier

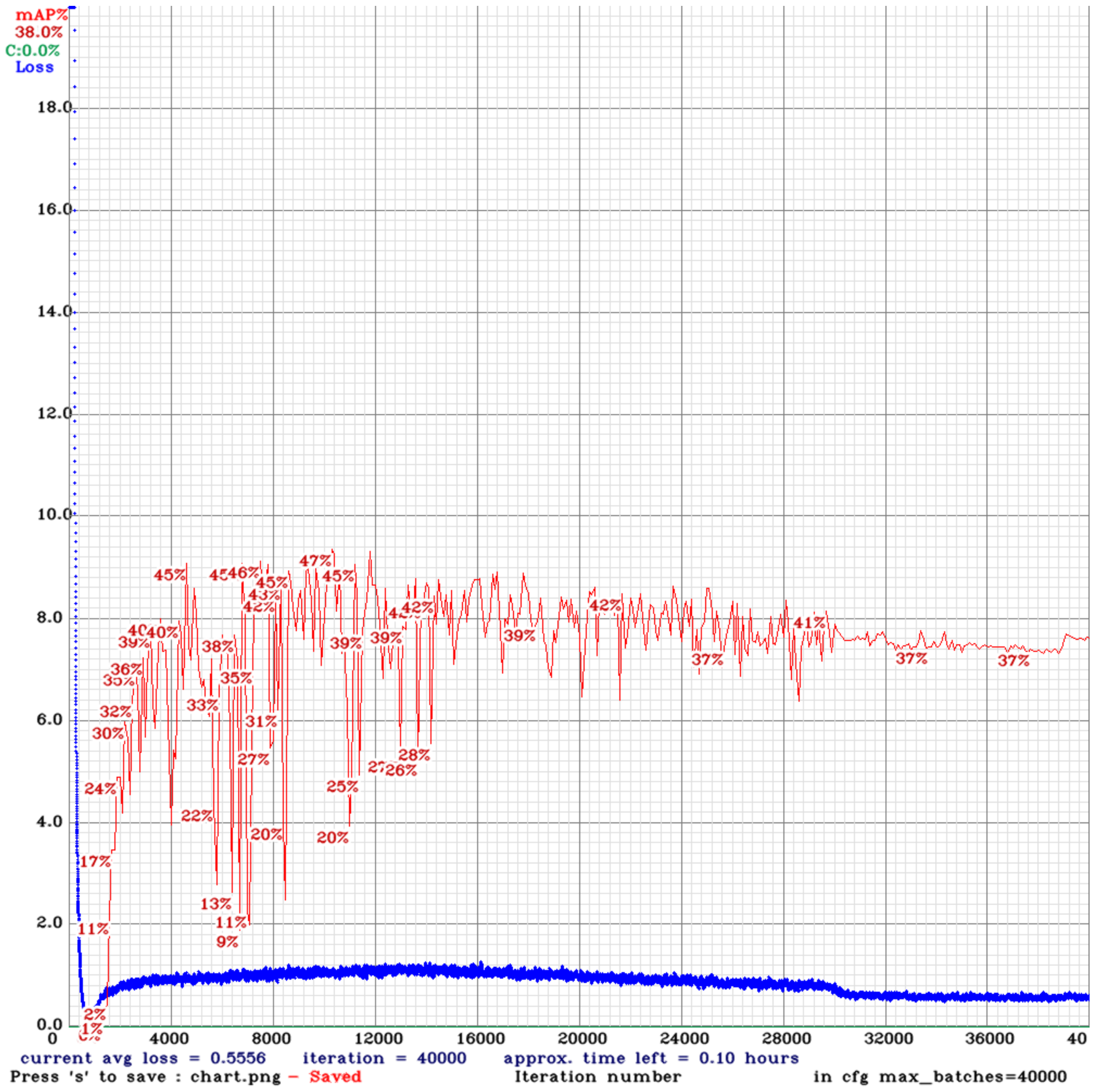


Figure 2| P4TS classifier

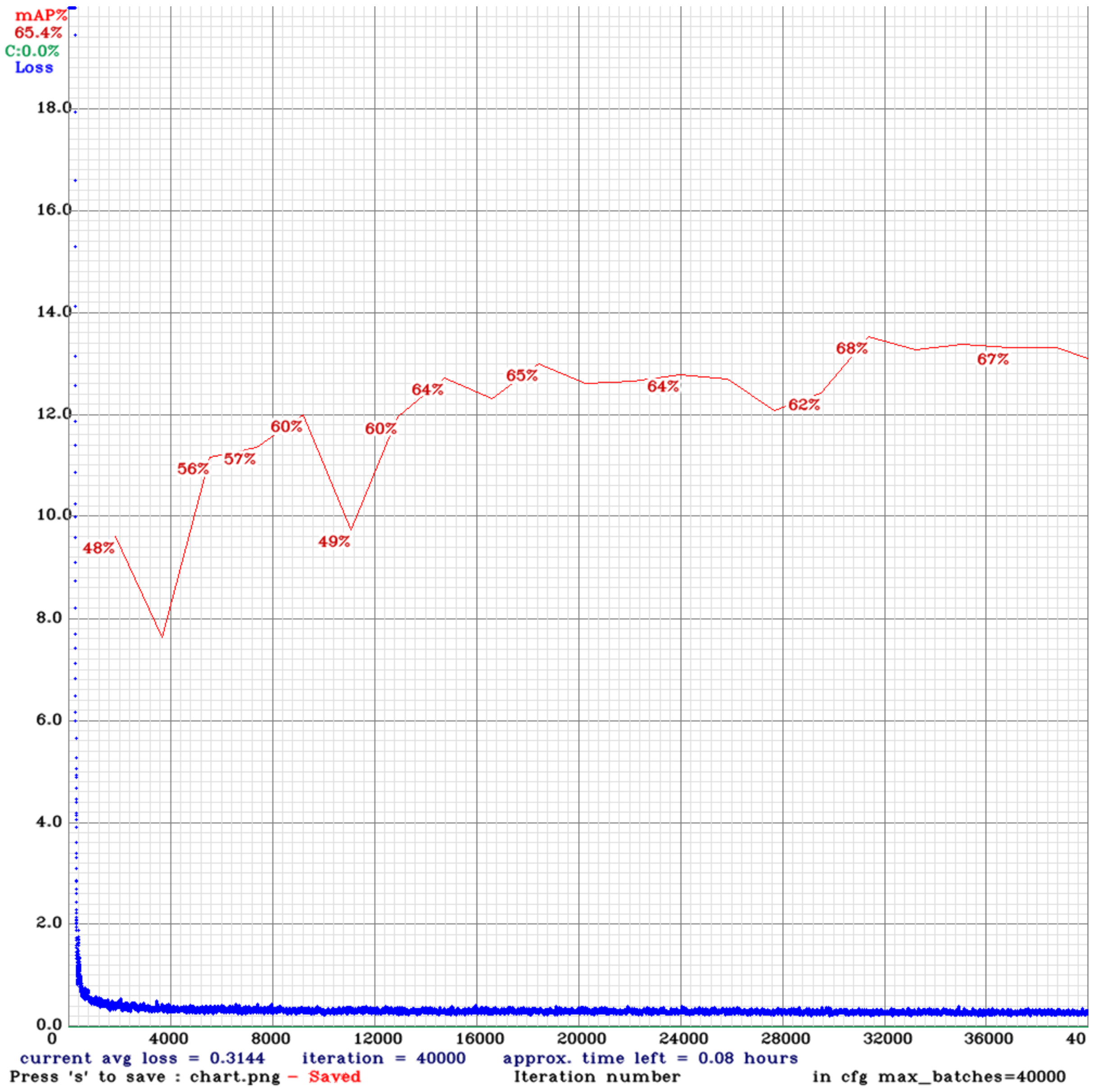
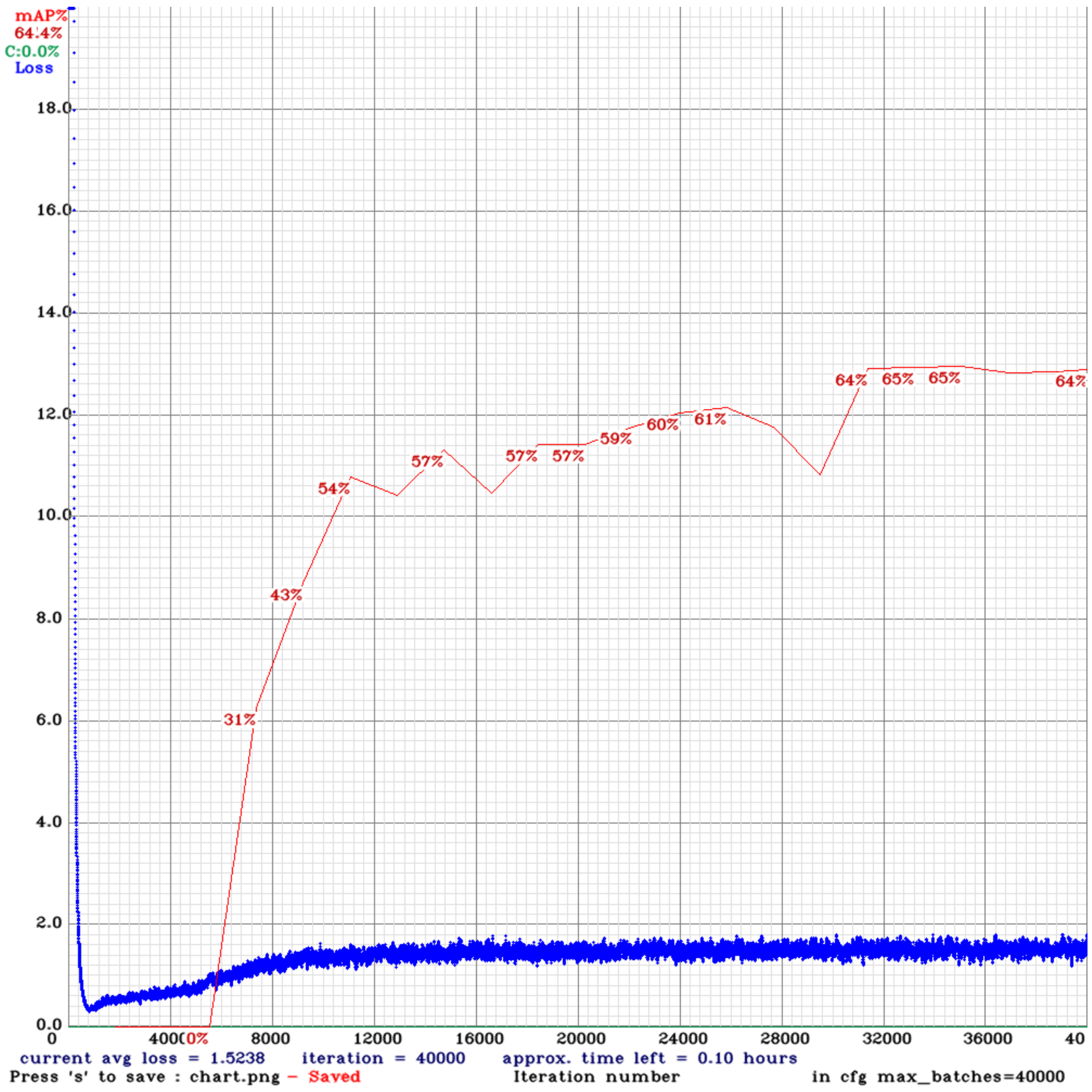


Figure 3| V3TS classifier



current avg loss = 1.5238 iteration = 40000 approx. time left = 0.10 hours

Press 's' to save : chart.png – Saved in cfg max_batches=40000

Figure 4| V4TS classifier

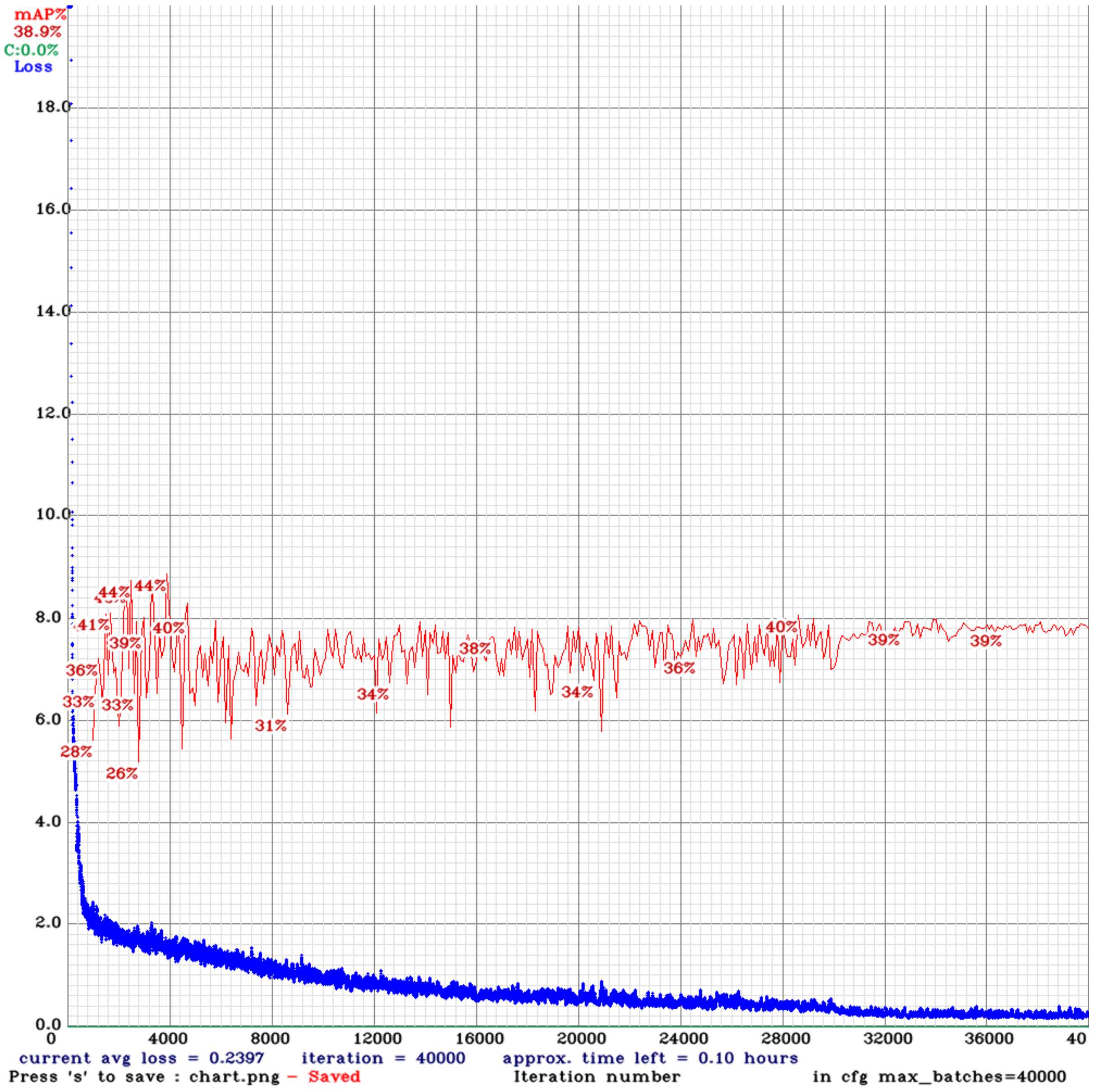


Figure 5| P3TL classifier

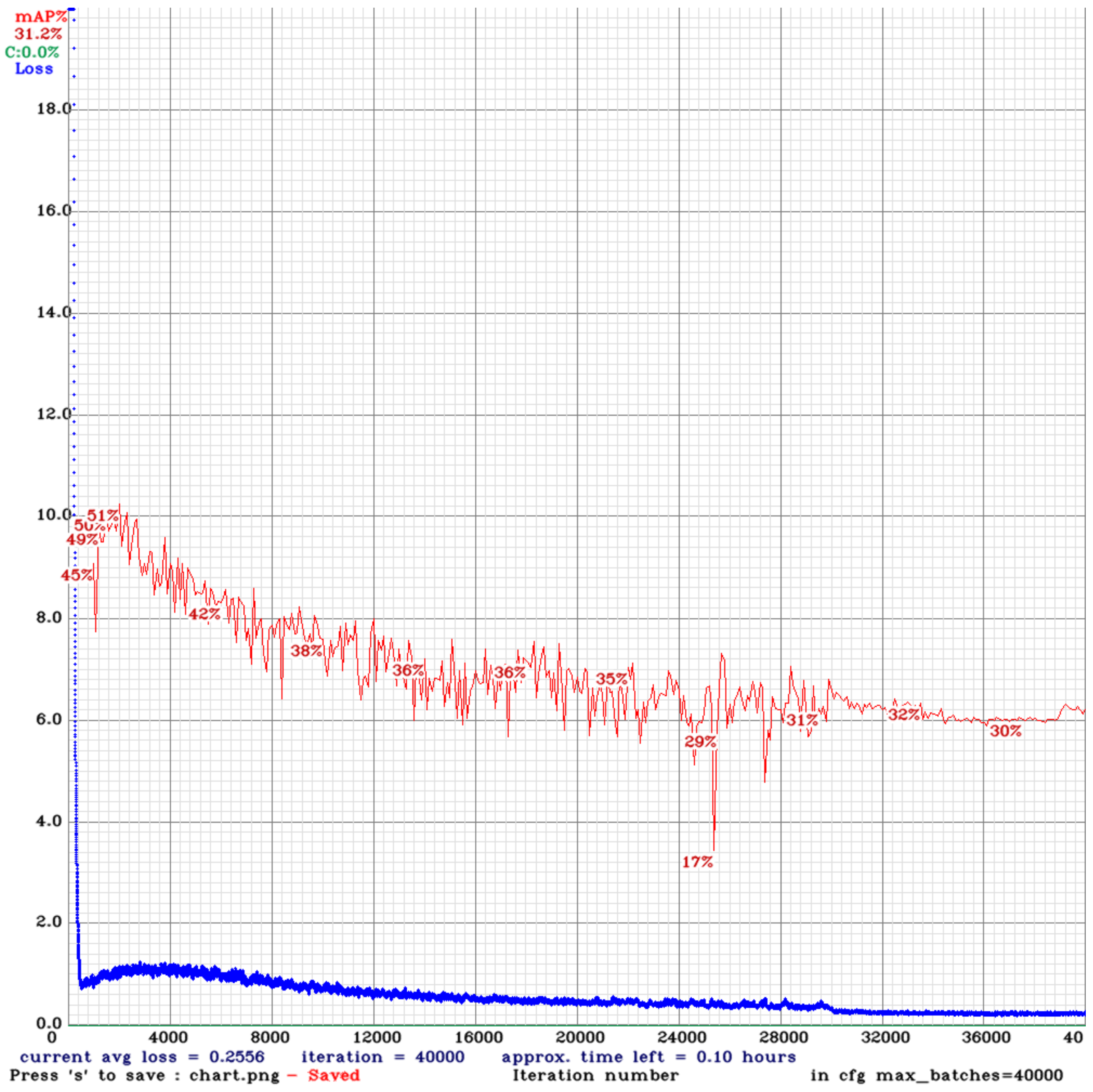


Figure 6| P4TL classifier

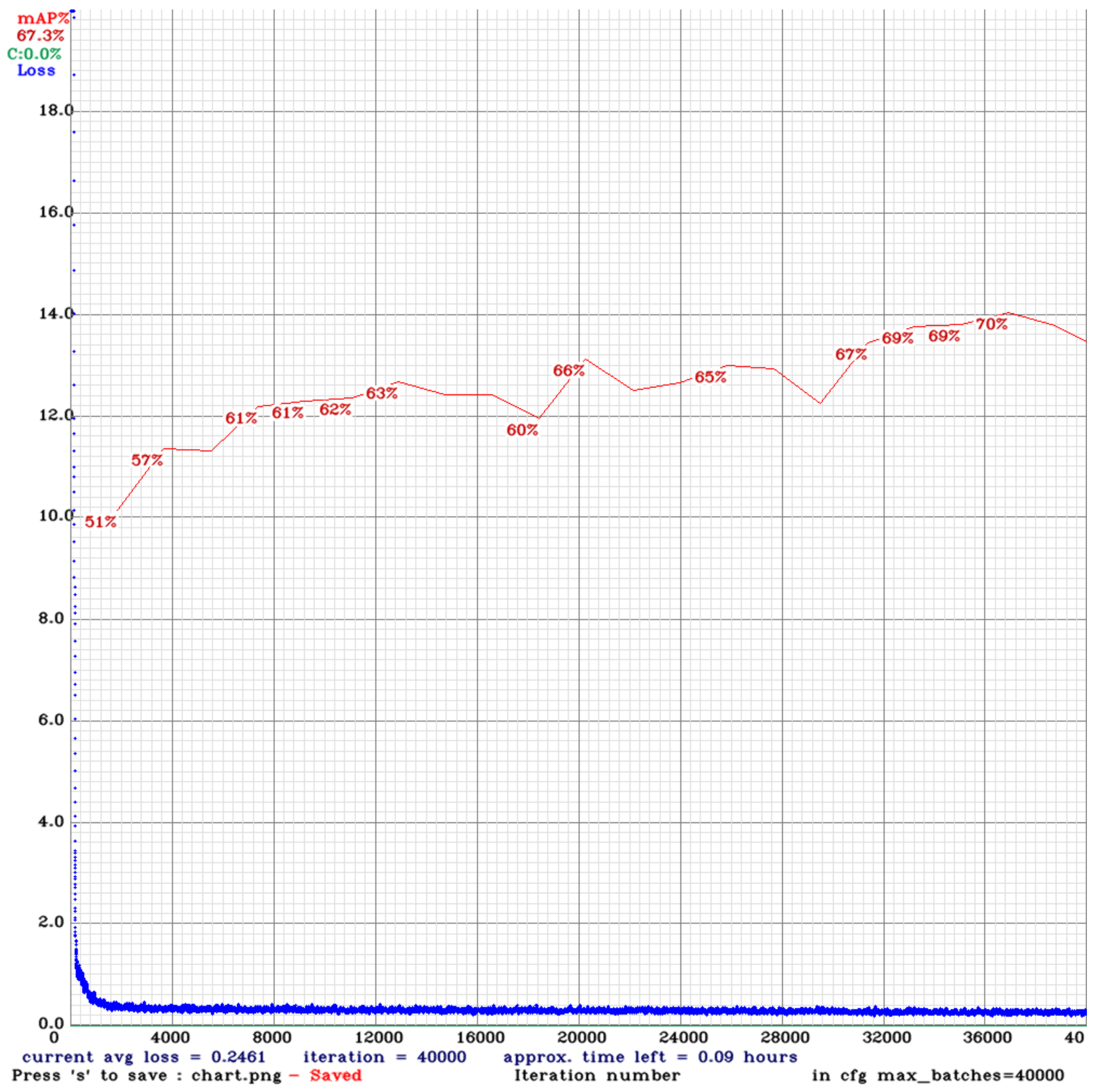


Figure 7| V3TL classifier

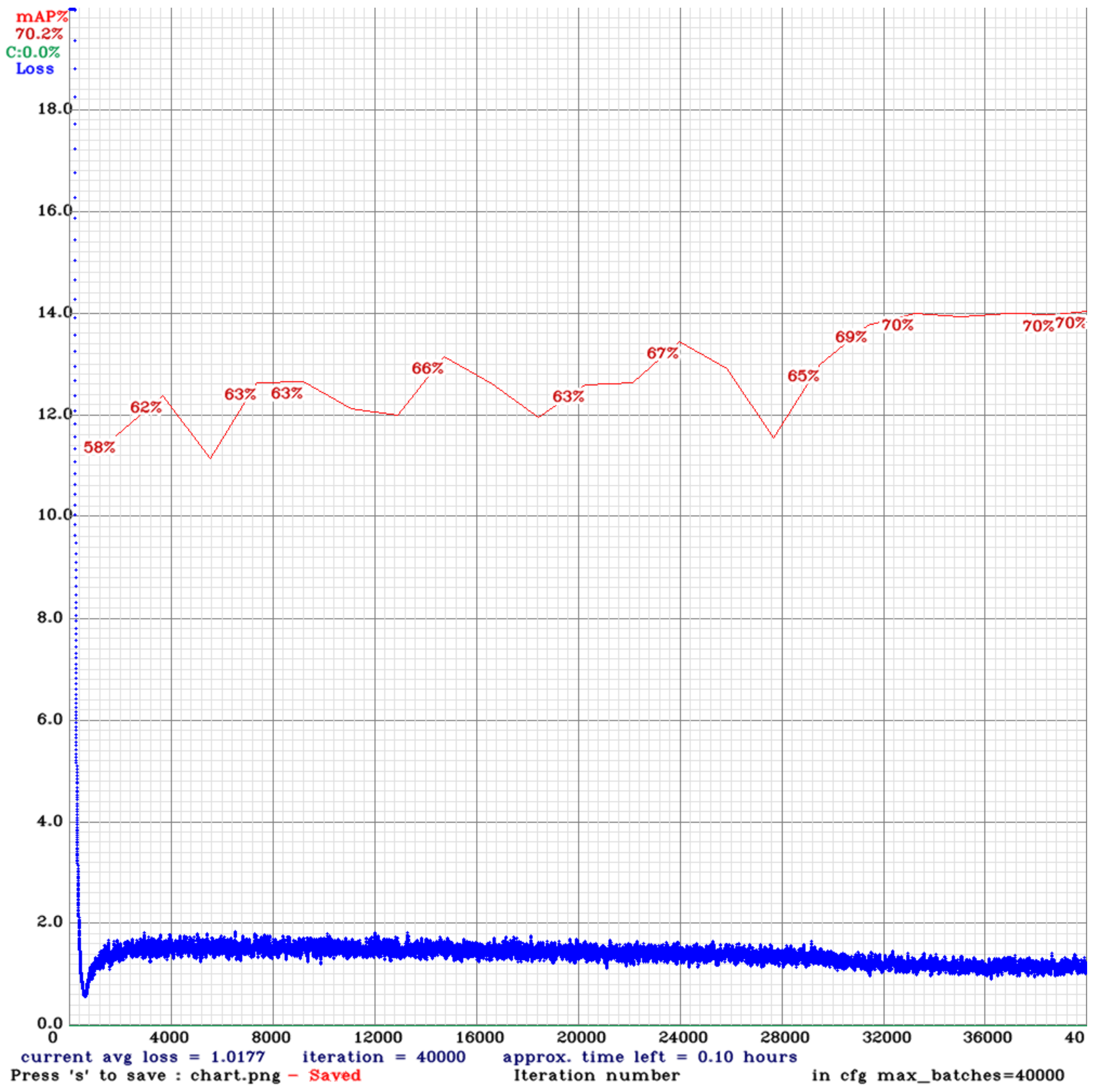
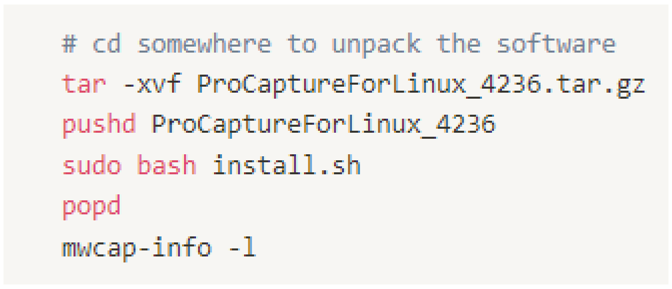


Figure 8| V4TL classifier

4.1.5 A2.5: Codes to run classifier on ROV observation platforms livestream

This outlines the two codes developed for running the classifier on the ROV livestream and, saving the detection outputs into a csv file for later analysis, as well as installation process required. Both are ran via the git bash terminal, one is written in Python programming language and the other in bash script, and are described as follows.

1. Install hardware, install software, run mwcap-info to check Magewell Capture card is responding there. Use git bash terminal window to run.



```
# cd somewhere to unpack the software
tar -xvf ProCaptureForLinux_4236.tar.gz
pushd ProCaptureForLinux_4236
sudo bash install.sh
popd
mwcap-info -l
```

2. Install ffmpeg using the following command line.

   - sudo apt-get install ffmpeg

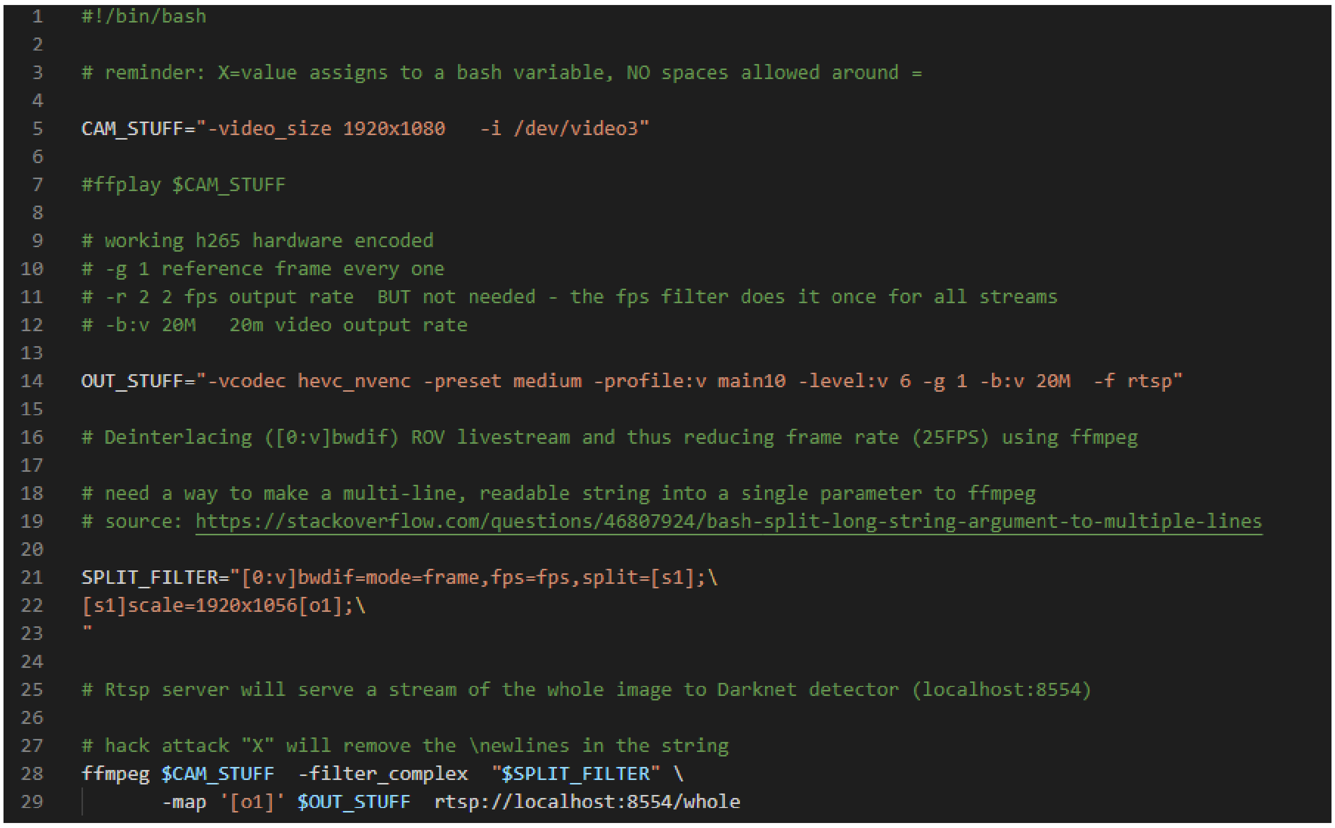
3. Install and set-up RTSP streamer using the following command line.

   Download 'rtsp-simple-server_v0.16.4_linux_amd64.tar.gz' and unpack in working directory (cd) via a bash shell terminal window using following code.

   - cd rtsp-simple-server

- ./rtsp-simple-server ./rtsp-simple-server.yml

4. ‘**run.capture.sh**’: This allows the real-time streaming protocol (RTSP) transfer compressed, deinterlaced ROV livestream from the H.264 encoder to the Darknet detector that is used to run the classifier.



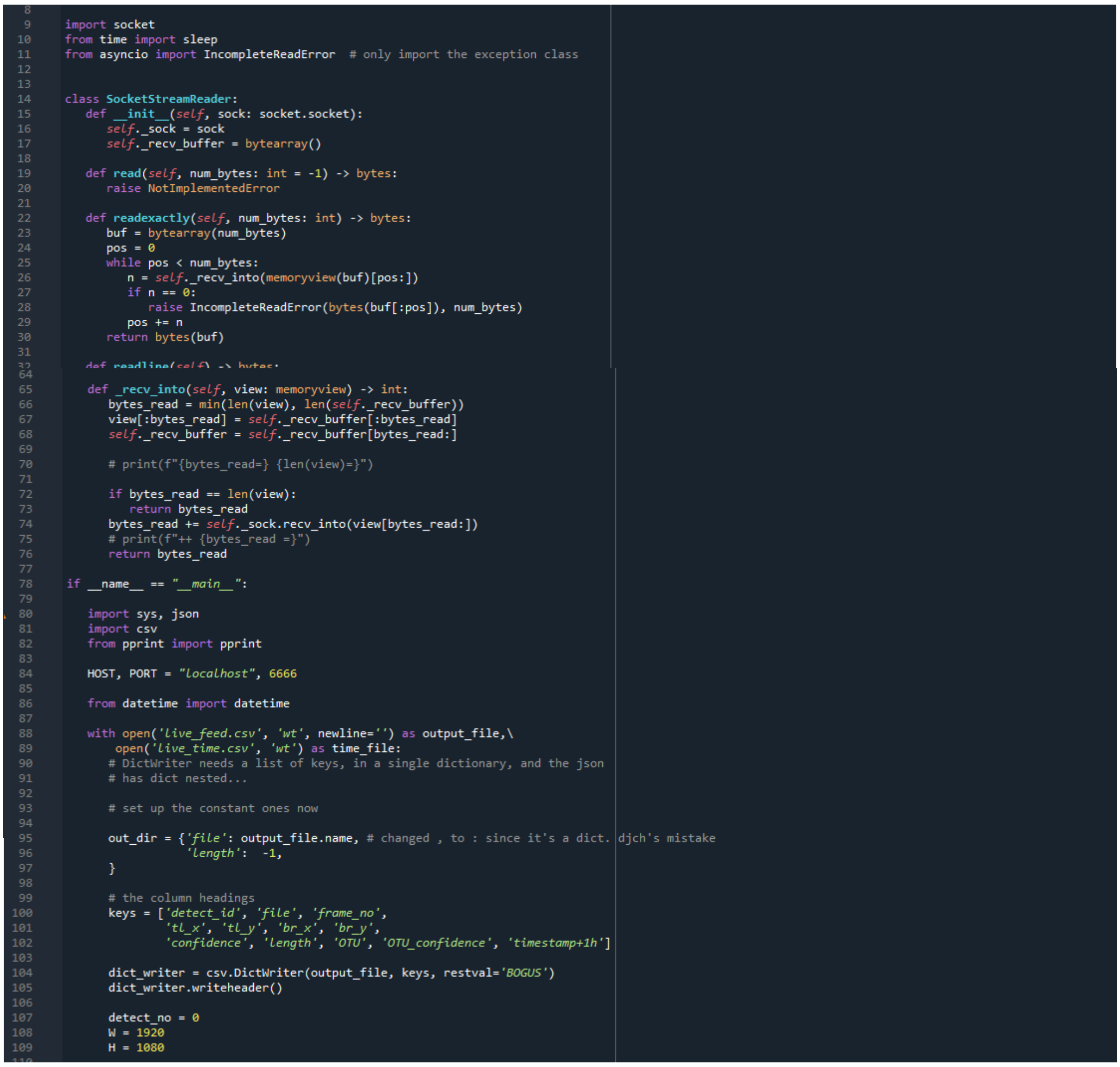
```bash
#!/bin/bash

# reminder: X=value assigns to a bash variable, NO spaces allowed around =

CAM_STUFF="-video_size 1920x1080   -i /dev/video3"

#ffplay $CAM_STUFF

# working h265 hardware encoded
# -g 1 reference frame every one
# -r 2 2 fps output rate  BUT not needed - the fps filter does it once for all streams
# -b:v 20M   20m video output rate

OUT_STUFF="-vcodec hevc_nvenc -preset medium -profile:v main10 -level:v 6 -g 1 -b:v 20M  -f rtsp"

# Deinterlacing ([0:v]bwdif) ROV livestream and thus reducing frame rate (25FPS) using ffmpeg

# need a way to make a multi-line, readable string into a single parameter to ffmpeg
# source: https://stackoverflow.com/questions/46807924/bash-split-long-string-argument-to-multiple-lines

SPLIT_FILTER="[0:v]bwdif=mode=frame,fps=fps,split=[s1];\
[s1]scale=1920x1056[o1];\
"

# Rtsp server will serve a stream of the whole image to Darknet detector (localhost:8554)

# hack attack "X" will remove the \newlines in the string
ffmpeg $CAM_STUFF  -filter_complex  "$SPLIT_FILTER" \
       -map '[o1]' $OUT_STUFF  rtsp://localhost:8554/whole
```

5. Run the following bash script in another git bash terminal window to allow Darknet detector to start running the classifier over the 1) saved video transect d or 2) ROV livestream, and save each frame of the video with detections found by the classifier to a folder on the machine B for further analysis.

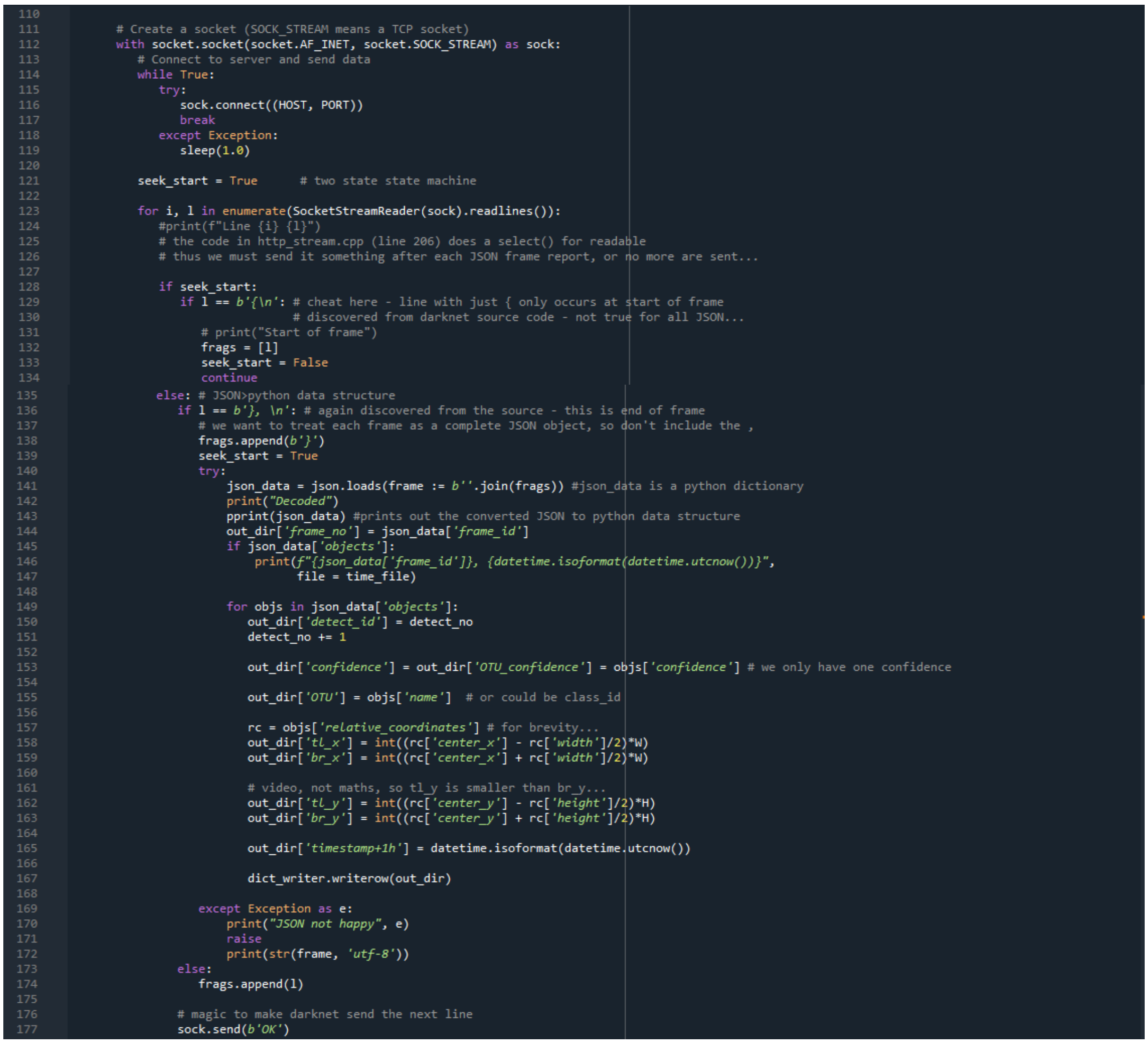
➢ **./darknet detector demo ./deep_training/'classifier'.data ./deep_training/'classifier'.cfg ./deep_training/models/'classifier'.weights ./'videofile.mov or livestream' -thresh 0.X -prefix ./Frames_Detected/Frame_No -json_port 6666**

The data, weights and configuration files are generated during the training process with the final weights generated being those used in this study.

6. '**transform_data.py**': This takes the JSON formatted Darknet detections and prints them out in real-time onto a Excel.csv document with timestamps to be used for analysing ecological information.



```python

import socket
from time import sleep
from asyncio import IncompleteReadError  # only import the exception class


class SocketStreamReader:
   def __init__(self, sock: socket.socket):
      self._sock = sock
      self._recv_buffer = bytearray()

   def read(self, num_bytes: int = -1) -> bytes:
      raise NotImplementedError

   def readexactly(self, num_bytes: int) -> bytes:
      buf = bytearray(num_bytes)
      pos = 0
      while pos < num_bytes:
         n = self._recv_into(memoryview(buf)[pos:])
         if n == 0:
            raise IncompleteReadError(bytes(buf[:pos]), num_bytes)
         pos += n
      return bytes(buf)

   def readline(self) -> bytes:

   def _recv_into(self, view: memoryview) -> int:
      bytes_read = min(len(view), len(self._recv_buffer))
      view[:bytes_read] = self._recv_buffer[:bytes_read]
      self._recv_buffer = self._recv_buffer[bytes_read:]

      # print(f"{bytes_read=} {len(view)=}")

      if bytes_read == len(view):
         return bytes_read
      bytes_read += self._sock.recv_into(view[bytes_read:])
      # print(f"++ {bytes_read =}")
      return bytes_read

if __name__ == "__main__":

   import sys, json
   import csv
   from pprint import pprint

   HOST, PORT = "localhost", 6666

   from datetime import datetime

   with open('live_feed.csv', 'wt', newline='') as output_file,\
        open('live_time.csv', 'wt') as time_file:
      # DictWriter needs a list of keys, in a single dictionary, and the json
      # has dict nested...

      # set up the constant ones now

      out_dir = {'file': output_file.name, # changed , to : since it's a dict. djch's mistake
                 'length':  -1,
      }

      # the column headings
      keys = ['detect_id', 'file', 'frame_no',
              'tl_x', 'tl_y', 'br_x', 'br_y',
              'confidence', 'length', 'OTU', 'OTU_confidence', 'timestamp+1h']

      dict_writer = csv.DictWriter(output_file, keys, restval='BOGUS')
      dict_writer.writeheader()

      detect_no = 0
      W = 1920
      H = 1080
```



```python
# Create a socket (SOCK_STREAM means a TCP socket)
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
   # Connect to server and send data
   while True:
      try:
         sock.connect((HOST, PORT))
         break
      except Exception:
         sleep(1.0)

   seek_start = True      # two state state machine

   for i, l in enumerate(SocketStreamReader(sock).readlines()):
      #print(f"Line {i} {l}")
      # the code in http_stream.cpp (line 206) does a select() for readable
      # thus we must send it something after each JSON frame report, or no more are sent...

      if seek_start:
         if l == b'{\n': # cheat here - line with just { only occurs at start of frame
                         # discovered from darknet source code - not true for all JSON...
            # print("Start of frame")
            frags = [l]
            seek_start = False
            continue
      else: # JSON>python data structure
         if l == b'}, \n': # again discovered from the source - this is end of frame
            # we want to treat each frame as a complete JSON object, so don't include the ,
            frags.append(b'}')
            seek_start = True
            try:
                json_data = json.loads(frame := b''.join(frags)) #json_data is a python dictionary
                print("Decoded")
                pprint(json_data) #prints out the converted JSON to python data structure
                out_dir['frame_no'] = json_data['frame_id']
                if json_data['objects']:
                    print(f"{json_data['frame_id']}, {datetime.isoformat(datetime.utcnow())}",
                          file = time_file)

                for objs in json_data['objects']:
                   out_dir['detect_id'] = detect_no
                   detect_no += 1

                   out_dir['confidence'] = out_dir['OTU_confidence'] = objs['confidence'] # we only have one confidence

                   out_dir['OTU'] = objs['name']  # or could be class_id

                   rc = objs['relative_coordinates'] # for brevity...
                   out_dir['tl_x'] = int((rc['center_x'] - rc['width']/2)*W)
                   out_dir['br_x'] = int((rc['center_x'] + rc['width']/2)*W)

                   # video, not maths, so tl_y is smaller than br_y...
                   out_dir['tl_y'] = int((rc['center_y'] - rc['height']/2)*H)
                   out_dir['br_y'] = int((rc['center_y'] + rc['height']/2)*H)

                   out_dir['timestamp+1h'] = datetime.isoformat(datetime.utcnow())

                   dict_writer.writerow(out_dir)

            except Exception as e:
                print("JSON not happy", e)
                raise
                print(str(frame, 'utf-8'))
         else:
            frags.append(l)

         # magic to make darknet send the next line
         sock.send(b'OK')
```

4.1.6 A2.6: Supporting results for applying a tracking threshold

This documents shows how the tracking threshold was tested and quantified in order to reduce the amount of false positive detections. Using a violin plot a visual representations of the distribution of false positive and true positive detections were made with respect to how long the classifier held that detection for over a 1 second (25 frames) time period. For example, one classifiers may detect many false positives, but they are only detected for short periods (e.g. 1 frame out of 25), whereas a true positive detection would be tracked for more (e.g. 20 out of 25 frames) as the classifier is likely more confident. This is an assumption, but it can help the elimination process of false positives be more efficient. We implement this by testing it there is a significant difference between the distribution of the data with respect to tracking longevity to assess whether it's worth using this method. Then if significant, the threshold to which we eliminate positive detections on is based on the lower quantile value of false positive detections. This ensure we remove little to no true positive detections whilst cleaning out a degree of false positive. Therefore, changing them to true negatives.

A2.6 Table 1 | T-test results that investigated if false positive and true positive detections have a significant differences between how long they are track for based on 1 second increments. And if significant, the frame value (X) out of 25 (i.e. 1-second increment) the tracking threshold that is implemented for each classifier.

Classifier	**T-Test**				**Tracking threshold (X/25 frames)**
	t	**df**	**p**	**p.adj**	
V3TL	-6.37	195	1.30×10^{-9}	1.30×10^{-9}	6
P3TS	-5.08	152	1.11×10^{-6}	1.11×10^{-6}	14
P3TL	-9.77	188	1.71×10^{-18}	1.71×10^{-18}	6

V4TS	-6.62	193	$3.43x10^{-10}$	$3.43x10^{-10}$	18
V4TL	-6.73	136	$4.35x10^{-10}$	$4.35x10^{-10}$	14
P4TS	-6.68	198	$2.36x10^{-10}$	$2.36x10^{-10}$	7
P4TL	-7.10	175	$3.07x10^{11}$	$3.07x10^{11}$	4

References

Abroyan, N.H. and Hakobyan, R.G. (2016) ‘A review of the usage of machine learning in real-time systems’, *Banber ANAU-Information technologies, electronics, radio equipment*, 19(1), pp. 46-54.

Anaconda Software Distribution (Conda). (2021) ‘Computer software. Version 2-2.4.0. (Austin, TX: Anaconda, 2016)’, source: *https://www.anaconda.com*.

Ani Brown Mary, N. and Dharma, D. (2019) ‘A novel framework for real-time diseased coral reef image classification’, *Multimedia Tools and Applications*, 78(9), pp. 11387-11425.

Ansari, S., Saad, A., Stahl, A. and Rajachandran, M. (2020) ‘Vision-based Real-time Zooplankton Detection and Classification using Faster R-CNN’, *Earth and Space Science Open Archive (ESSOAr): Ocean Sciences Meeting*, pp. 1-8.

Arendt, M. and Rückert, J. (2020) ‘The effects of colour enhancement and IoU optimisation on object detection and segmentation of coral reef structures’, *Conference and Labs of the Evaluation Forum (CLEF) (Working Notes)*.

Armstrong, C.W., Foley, N.S., Tinch, R. and van den Hove, S. (2012) ‘Services from the deep: Steps towards valuation of deep sea goods and services’, *Ecosystem Services*, 2, pp. 2-13.

Ashford, O.S., Davies, A.J. and Jones, D.O. (2014) ‘Deep-sea benthic megafaunal habitat suitability modelling: A global-scale maximum entropy model for xenophyophores’, *Deep Sea Research Part I: Oceanographic Research Papers*, 94, pp. 31-44.

Bashir, F. and Porikli, F. (2006) ‘Performance evaluation of object detection and tracking systems’, In *Proceedings 9th IEEE International Workshop on PETS*, pp. 7-14.

Beijbom, O., Edmunds, P.J., Roelfsema, C., Smith, J., Kline, D.I., Neal, B.P., Dunlap, M.J., Moriarty, V., Fan, T.Y., Tan, C.J. and Chan, S. (2015) ‘Towards automated annotation of benthic survey images: Variability of human experts and operational modes of automation’, *PloS one*, 10(7), p. e0130312.

Beijbom, O., Treibitz, T., Kline, D.I., Eyal, G., Khen, A., Neal, B., Loya, Y., Mitchell, B.G. and Kriegman, D. (2016) ‘Improving automated annotation of benthic survey images using wide-band fluorescence’, *Scientific reports*, 6(1), pp. 1-11.

Bergum, S., Saad, A. and Stahl, A. (2020) ‘Automatic in-situ instance and semantic segmentation of planktonic organisms using Mask R-CNN’, In *Global Oceans 2020: Singapore–US Gulf Coast*, pp. 1-8.

Bianco, S., Cadene, R., Celona, L. and Napoletano, P. (2018) ‘Benchmark analysis of representative deep neural network architectures’, *IEEE Access*, 6, pp. 64270-64277.

Bisong E. (2019) ‘Google Colaboratory. In: Building Machine Learning and Deep Learning Models on Google Cloud Platform’ *Apress, Berkeley, California CA*, source: *https://doi.org/10.1007/978-1-4842-4470-8_7*.

Bochkovskiy, A., Wang, C.Y. and Liao, H.Y.M. (2020) ‘Yolov4: Optimal speed and accuracy of object detection’, *arXiv preprint arXiv:2004.10934.*

Bohte, S., and Nguyen, H.S. (2016) ‘Modern Machine Learning: More with Less, Cheaper and Better’, *ERCIM News*, 107, pp, 16-17.

Buhl-Mortensen, L., Vanreusel, A., Gooday, A.J., Levin, L.A., Priede, I.G., Buhl-Mortensen, P., Gheerardyn, H., King, N.J. and Raes, M. (2010) ‘Biological structures as a source of habitat heterogeneity and biodiversity on the deep ocean margins’, *Marine Ecology*, 31(1), pp. 21-50.

Boulais, O., Woodward, B., Schlining, B., Lundsten, L., Barnard, K., Bell, K.C. and Katija, K. (2020) ‘FathomNet: An underwater image training database for ocean exploration and discovery’ *arXiv preprint arXiv:2007.00114*.

Bureš, J., Eerola, T., Lensu, L., Kälviäinen, H. and Zemčík, P. (2021) ‘Plankton Recognition in Images with Varying Size’, In *International Conference on Pattern Recognition*, pp. 110-120.

Cai, Z., and Vasconcelos, N. (2018) ‘Cascade r-cnn: Delving into high quality object detection’, In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6154-6162.

Chatfield, K., Simonyan, K., Vedaldi, A. and Zisserman, A. (2014) ‘Return of the devil in the details: Delving deep into convolutional nets’, *arXiv preprint arXiv:1405.3531*.

Cheng, K., Cheng, X., Wang, Y., Bi, H. and Benfield, M.C. (2019) ‘Enhanced convolutional neural network for plankton identification and enumeration’, *PLoS One*, 14(7), p.e0219570.

Chollet, F. (2017) ‘Xception: Deep learning with depthwise separable convolutions’, In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1251-1258.

Christin, S., Hervet, É. and Lecomte, N. (2019) 'Applications for deep learning in ecology', *Methods in Ecology and Evolution*, 10(10), pp. 1632-1644.

Clark, M. R., Consalvey, M. and Rowden, A. A. (2016) 'Biological Sampling in the Deep Sea', *John Wiley & Sons*, pp. 1-443.

Crescitelli, A.M., Gansel, L.C. and Zhang, H. (2021) 'NorFisk: fish image dataset from Norwegian fish farms for species recognition using deep neural networks', *Modelling Identification and Control*, 42(1), pp. 1-16.

Culverhouse, P.F., Williams, R., Reguera, B., Herry, V., González-Gil, S. (2003) 'Do experts make mistakes? A comparison of human and machine identification of dinoflagellates', *Marine Ecology Progress Series*, 247, 17-25.

Dai, J., Wang, R., Zheng, H., Ji, G. and Qiao, X. (2016) 'Zooplanktonet: Deep convolutional network for zooplankton classification', In *OCEANS 2016-Shanghai*, pp. 1-6.

Danovaro, R., Snelgrove, P.V. and Tyler, P. (2014) 'Challenging the paradigms of deep-sea ecology', *Trends in ecology & evolution*, 29(8), pp. 465-475.

Dawkins, M., Sherrill, L., Fieldhouse, K., Hoogs, A., Richards, B., Zhang, D., Prasad, L., Williams, K., Lauffenburger, N. and Wang, G. (2017) 'An open-source platform for underwater image and video analytics', In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 898-906.

Deng, J., Dong, W., Socher, R., Li, L.J., Li, K. and Fei-Fei, L. (2009) 'Imagenet: A large-scale hierarchical image database', In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248-255.

Diesing, M. (2016) ‘Mapping the Oceans: Progress in Semi-Automated Seabed Classification Required’, In *2016 Ocean Sciences Meeting*, *American Geophysical Union (AGU)*, pp. MG44A-1958.

Ditria, E.M., Lopez-Marcano, S., Sievers, M., Jinks, E.L., Brown, C.J. and Connolly, R.M. (2020) ‘Automating the analysis of fish abundance using object detection: optimizing animal ecology with deep learning’, *Frontiers in Marine Science*, 7, p.429.

Ditria, E.M., Connolly, R.M., Jinks, E.L. and Lopez-Marcano, S. (2021) ‘Annotated video footage for automated identification and counting of fish in unconstrained seagrass habitats’, *Frontiers in Marine Science*, 8, p.160.

Dujon, A.M., Ierodiaconou, D., Geeson, J.J., Arnould, J.P., Allan, B.M., Katselidis, K.A. and Schofield, G. (2021) ‘Machine learning to detect marine animals in UAV imagery: effect of morphology, spacing, behaviour and habitat’, *Remote Sensing in Ecology and Conservation*, 7(3), pp. 341-354.

Dunbabin, M. and Marques, L. (2012) ‘Robots for environmental monitoring: Significant advancements and applications’, IEEE *Robotics & Automation Magazine*, 19(1), pp. 24-39.

Durden, J.M., Schoening, T., Althaus, F., Friedman, A., Garcia, R., Glover, A.G., Greinert, J., Stout, N.J., Jones, D.O., Jordt, A. and Kaeli, J.W. (2016a) ‘Perspectives in visual imaging for marine biology and ecology: from acquisition to understanding’, In *Oceanography and Marine Biology*, pp. 9-80), CRC Press.

Durden, J.M., Bett, B.J., Schoening, T., Morris, K.J., Nattkemper, T.W. and Ruhl, H.A. (2016b) ‘Comparison of image annotation data generated by multiple investigators for benthic ecology’, *Marine Ecology Progress Series*, 552, pp.61-70.

Durden, J.M., Hosking, B., Bett, B.J., Cline, D. and Ruhl, H.A. (2021) ‘Automated classification of fauna in seabed photographs: The impact of training and validation dataset size, with considerations for the class imbalance’, *Progress in Oceanography*, 196, p.102612.

Eglen, S.J., Marwick, B., Halchenko, Y.O., Hanke, M., Sufi, S., Gleeson, P., Silver, R.A., Davison, A.P., Lanyon, L., Abrams, M. and Wachtler, T. (2017) ‘Toward standard practices for sharing computer code and programs in neuroscience’, *Nature neuroscience*, 20(6), pp. 770-773.

Elawady, M. (2015) ‘Sparse coral classification using deep convolutional neural networks’, *arXiv preprint arXiv:1511.09067*.

Elineau, A., Desnos, C., Jalabert, L., Olivier, M., Romagnan, J.B., Brandao, M., Lombard, F., Llopis, N., Courboulès, J., Caray-Counil, L. and Serranito, B. (2018) ‘Zooscannet: plankton images captured with the zooscan’.

Enkvetchakul, P. and Surinta, O. (2021) ‘Effective data augmentation and training techniques for improving deep learning in plant leaf disease recognition’, *Applied Science and Engineering Progress*.

Everingham, M., Van Gool, L., Williams, C.K., Winn, J. and Zisserman, A. (2010) ‘The pascal visual object classes (VOC) challenge’, *International journal of computer vision*, 88(2), pp. 303-338.

Food and Agriculture Organisation (FAO) and Japan Government Cooperative Programme. (2008) ‘Report of the FAO Workshop on Vulnerable Ecosystems and Destructive Fishing in Deep-sea Fisheries: *Rome*’, *Food & Agriculture Org*, Report number: 829.

Favret, C. and Sieracki, J.M. (2016) ‘Machine vision automated species identification scaled towards production levels’, *Systematic Entomology*, 41(1), pp. 133-143.

First, M.R. and Drake, L.A. (2012) ‘Performance of the human “counting machine”: evaluation of manual microscopy for enumerating plankton’, *Journal of plankton research*, 34(12), pp. 1028-1041.

Fisher, R.B., Chen-Burger, Y.H., Giordano, D., Hardman, L. and Lin, F.P. eds. (2016) ‘Fish4Knowledge: collecting and analysing massive coral reef fish video data’, *Heidelberg: Springer*, 104.

Freeman, E.A. and Moisen, G. (2008) ‘PresenceAbsence: An R package for presence absence analysis’, *Journal of Statistical Software*, 23(11), p.31.

Géron, A. (2019) ‘Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems’, *O'Reilly Media, Inc*.

Ghahramani, Z. (2015) ‘Probabilistic machine learning and artificial intelligence’, *Nature*, 521(7553), pp. 452-459.

Girshick, R., Donahue, J., Darrell, T. and Malik, J. (2014) ‘Rich feature hierarchies for accurate object detection and semantic segmentation’, In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580-587.

Girshick, R. (2015) ‘Fast r-cnn’, In *Proceedings of the IEEE international conference on computer vision*, pp. 1440-1448.

Gómez-Ríos, A., Tabik, S., Luengo, J., Shihavuddin, A.S.M., Krawczyk, B. and Herrera, F. (2019) 'Towards highly accurate coral texture images classification using deep convolutional neural networks and data augmentation', *Expert Systems with Applications*, 118, pp. 315-328.

Goodfellow, I., Bengio, Y. and Courville, A. (2016) 'Machine learning basics', *Deep learning*, 1, pp. 98-164.

Goodwin, M., Halvorsen, K.T., Jiao, L., Knausgård, K.M., Martin, A.H., Moyano, M., Oomen, R.A., Rasmussen, J.H., Sørdalen, T.K. and Thorbjørnsen, S.H. (2021) 'Unlocking the potential of deep learning for marine ecology: overview, applications, and outlook', *arXiv preprint arXiv:2109.14737*.

Hampton, S.E., Strasser, C.A., Tewksbury, J.J., Gram, W.K., Budden, A.E., Batcheller, A.L., Duke, C.S. and Porter, J.H. (2013) 'Big data and the future of ecology', *Frontiers in Ecology and the Environment*, 11(3), pp. 156-162.

Han, D., Liu, Q. and Fan, W. (2018) 'A new image classification method using CNN transfer learning and web data augmentation', *Expert Systems with Applications*, 95, pp. 43-56.

Han, F., Yao, J., Zhu, H. and Wang, C. (2020) 'Marine Organism Detection and Classification from Underwater Vision Based on the Deep CNN Method', *Mathematical Problems in Engineering*.

Harris, P.T., Macmillan-Lawler, M., Rupp, J. and Baker, E.K. (2014) 'Geomorphology of the oceans', *Marine Geology*, 352, pp.4-24.

Hastie, T., Tibshirani, R. and Friedman, J. (2009) ‘The elements of statistical learning: data mining, inference, and prediction’, *Springer Science & Business Media: New York*, 2, pp. 1-758.

He, K., Zhang, X., Ren, S. and Sun, J. (2016) ‘Deep residual learning for image recognition’, In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770-778.

He, K., Gkioxari, G., Dollár, P. and Girshick, R. (2017) ‘Mask r-cnn’, In *Proceedings of the IEEE international conference on computer vision*, pp. 2961-2969.

Hensman, P. and Masko, D. (2015) ‘The impact of imbalanced training data for convolutional neural networks’, *Degree Project in Computer Science, KTH Royal Institute of Technology*.

Hinton, G.E. and Salakhutdinov, R.R. (2006) ‘Reducing the dimensionality of data with neural networks’, *Science*, 313(5786), pp. 504-507.

Hong Khai, T., Abdullah, S.N.H.S., Hasan, M.K. and Tarmizi, A. (2022) ‘Underwater Fish Detection and Counting Using Mask Regional Convolutional Neural Network’, *Water*, 14(2), p.222.

Hopkinson, B.M., King, A.C., Owen, D.P., Johnson-Roberson, M., Long, M.H. and Bhandarkar, S.M. (2020) ‘Automated classification of three-dimensional reconstructions of coral reefs using convolutional neural networks’, *PloS one*, 15(3), p.e0230671.

Hou, J., He, Y., Yang, H., Connor, T., Gao, J., Wang, Y., Zeng, Y., Zhang, J., Huang, J., Zheng, B. and Zhou, S. (2020) ‘Identification of animal individuals using deep learning: A case study of giant panda’, *Biological Conservation*, 242, p.108414.

Howell, K. L. and Davies, J. S. (2016) 'Deep-sea species image catalogue, On-line version 2'. Source: *https://deepseacru.org/2016/12/16/deep-sea-species-image-catalogue/*.

Howell, K.L., Davies, J.S., Allcock, A.L., Braga-Henriques, A., Buhl-Mortensen, P., Carreiro-Silva, M., Dominguez-Carrió, C., Durden, J.M., Foster, N.L., Game, C.A. and Hitchin, B. (2019) 'A framework for the development of a global standardised marine taxon reference image database (SMarTaR-ID) to support image-based analyses', *PLoS One*, 14(12), p.e0218904.

Howell, K.L., Hilário, A., Allcock, A.L., Bailey, D., Baker, M., Clark, M.R., Colaço, A., Copley, J., Cordes, E.E., Danovaro, R. and Dissanayake, A. (2021) 'A decade to study deep-sea life', *Nature Ecology & Evolution*, 5(3), pp. 265-267.

Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S. and Murphy, K. (2017) 'Speed/accuracy trade-offs for modern convolutional object detectors', In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7310-7311.

Huck, N. (2019) 'Large data sets and machine learning: Applications to statistical arbitrage', *European Journal of Operational Research*, 278(1), pp. 330-342.

Hussain, M., Bird, J.J. and Faria, D.R. (2018) 'A study on cnn transfer learning for image classification', In *UK Workshop on Computational Intelligence*, Springer, Cham, pp. 191-202.

International Council for the Exploration of the Sea (ICES) 2016. Source: *https://www.ices.dk/data/Documents/VME/VME_2016_Data_Reporting_Guidance.pdf*.

Irisson, J.O., Ayata, S.D., Lindsay, D.J., Karp-Boss, L. and Stemmann, L. (2022) ‘Machine Learning for the study of plankton and marine snow from images’, *Annual review of marine science*, 14.

Jäger, J., Wolff, V., Fricke-Neuderth, K., Mothes, O. and Denzler, J. (2017) ‘Visual fish tracking: Combining a two-stage graph approach with CNN-features’, In *OCEANS 2017-Aberdeen* IEEE, pp. 1-6.

Jaisakthi, S.M., Mirunalini, P. and Aravindan, C. (2019) ‘Coral Reef Annotation and Localization using Faster R-CNN’, In *CLEF (Working Notes)*.

Jeong, H.J., Park, K.S. and Ha, Y.G. (2018) ‘Image preprocessing for efficient training of YOLO deep learning networks’, In *2018 IEEE International Conference on Big Data and Smart Computing (BigComp)* IEEE, pp. 635-637.

Jeyaraman, B.P., Olsen, L.R. and Wambugu, M. (2019) ‘Practical Machine Learning with R: Define, build, and evaluate machine learning models for real-world applications’, *Packt Publishing Ltd*, 5.

Jun, S. and Taheri-Araghi, S. (2015) ‘Cell-size maintenance: universal strategy revealed’, *Trends in microbiology*, 23(1), pp. 4-6.

Kandimalla, V., Richard, M., Smith, F., Quirion, J., Torgo, L. and Whidden, C. (2022) ‘Automated detection, classification and counting of fish in fish passages with deep learning’, *Frontiers in Marine Science*, p.2049.

Kassambara, A. (2021) ‘Pipe-Friendly Framework for Basic Statistical Tests’, *R Package Rstatix Version 0.7.0. Source: https://mran.microsoft. com/web/packages/rstatix/index.html*.

Katija, K., Orenstein, E., Schlining, B., Lundsten, L., Barnard, K., Sainz, G., Boulais, O., Woodward, B. and Bell, K.C. (2021a) ‘FathomNet: A global underwater image training set for enabling artificial intelligence in the ocean’, *arXiv preprint arXiv:2109.14646*.

Katija, K., Roberts, P.L., Daniels, J., Lapides, A., Barnard, K., Risi, M., Ranaan, B.Y., Woodward, B.G. and Takahashi, J. (2021b) ‘Visual tracking of deepwater animals using machine learning-controlled robotic underwater vehicles’, In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 860-869.

Kelly, M.G., Bayer, M.M., Hürlimann, j. and Telford, R.J. (2002) ‘Human error and quality assurance in diatom analysis’, In *Automatic diatom identification*, pp. 75-91.

Khan, A., Sohail, A., Zahoora, U. and Qureshi, A.S. (2020) ‘A survey of the recent architectures of deep convolutional neural networks’, *Artificial Intelligence Review*, 53(8), pp. 5455-5516.

Knausgård, K.M., Wiklund, A., Sørdalen, T.K., Halvorsen, K.T., Kleiven, A.R., Jiao, L. and Goodwin, M. (2021) ‘Temperate fish detection and classification: A deep learning based approach’, *Applied Intelligence*, pp. 1-14.

Kononenko, I. (2001) ‘Machine learning for medical diagnosis: history, state of the art and perspective’, *Artificial Intelligence in medicine*, 23(1), pp. 89-109.

Krizhevsky, A., Sutskever, I. and Hinton, G.E. (2012) ‘Imagenet classification with deep convolutional neural networks’, *Advances in neural information processing systems*, 25, pp. 1097-1105.

Kuhnz, L.A., Ruhl, H.A., Huffard, C.L. and Smith Jr, K.L. (2014) ‘Rapid changes and long-term cycles in the benthic megafaunal community observed over 24 years in the abyssal northeast Pacific’, *Progress in Oceanography*, 124, pp.1-11.

Kung, A., Svobodova, K., Lèbre, E., Valenta, R., Kemp, D. and Owen, J.R. (2021) ‘Governing deep sea mining in the face of uncertainty’, *Journal of Environmental Management*, 279, p.111593.

Kurtzer, G.M., Sochat, V. and Bauer, M.W. (2017) ‘Singularity: Scientific containers for mobility of compute’, *PloS one*, 12(5), p.e0177459.

Langenkämper, D., Zurowietz, M., Schoening, T. and Nattkemper, T. W. (2017) 'BIIGLE 2.0 - Browsing and Annotating Large Marine Image Collections', *Frontiers in Marine Science*, 4(83).

Langenkämper, D., Kevelaer, R.V. and Nattkemper, T.W. (2018) ‘Strategies for tackling the class imbalance problem in marine image classification’, In *International Conference on Pattern Recognition*, pp. 26-36, Springer, Cham.

LeCun, Y. and Bengio, Y. (1995) ‘Convolutional networks for images, speech, and time series’, *The handbook of brain theory and neural networks*, 3361(10), p.1995.

LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P. (1998) ‘Gradient-based learning applied to document recognition’, *Proceedings of the IEEE*, 86(11), pp. 2278-2324.

LeCun, Y., Bengio, Y. and Hinton, G. (2015) ‘Deep learning’, *Nature*, 521(7553), pp. 436-444.

Lee, H., Park, M. and Kim, J. (2016) ‘Plankton classification on imbalanced large scale database via convolutional neural networks with transfer learning’, In *2016 IEEE International Conference on Image Processing (ICIP)*, pp. 3713-3717.

Levin, L.A., DeMaster, D.J., McCann, L.D. and Thomas, C.L. (1986) ‘Effects of giant protozoans (class: Xenophyophorea) on deep-seamount benthos’, *Marine Ecology Progress Series*, 29, pp.99-104.

Levin, L.A., Bett, B.J., Gates, A.R., Heimbach, P., Howe, B.M., Janssen, F., McCurdy, A., Ruhl, H.A., Snelgrove, P., Stocks, K.I. and Bailey, D. (2019) ‘Global observing needs in the deep ocean’, *Frontiers in Marine Science*, 6, p.241.

Li, X., Shang, M., Qin, H. and Chen, L. (2015) ‘Fast accurate fish detection and recognition of underwater images with fast R-CNN’, In *OCEANS 2015 - MTS/IEEE Washington DC*, pp. 1–5.

Li, X., Zhou, Y., Pan, Z. and Feng, J. (2019) ‘Partial order pruning: for best speed/accuracy trade-off in neural architecture search’, In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9145-915.

Li, D., Wang, Q., Li, X., Niu, M., Wang, H. and Liu, C. (2022) ‘Recent advances of machine vision technology in fish classification’, *ICES Journal of Marine Science*.

Libreros, J., Bueno, G., Trujillo, M. and Ospina, M. (2018) ‘Automated identification and classification of diatoms from water resources’, In *Iberoamerican Congress on Pattern Recognition,* Springer, Cham, pp. 496-503.

Liu, S., Li, X., Gao, M., Cai, Y., Nian, R., Li, P., Yan, T. and Lendasse, A. (2018) 'Embedded online fish detection and tracking system via YOLOv3 and parallel correlation filter', In *OCEANS 2018 MTS/IEEE Charleston* IEEE, pp. 1-6.

Liu, T., Li, P., Liu, H., Deng, X., Liu, H. and Zhai, F. (2021) 'Multi-class fish stock statistics technology based on object classification and tracking algorithm', *Ecological Informatics*, 63, p.101240.

Liu, Y. and Wang, S. (2021) 'A quantitative detection algorithm based on improved faster R-CNN for marine benthos', *Ecological Informatics*, 61, p.101228.

Lopez-Vazquez, V., Lopez-Guede, J.M., Marini, S., Fanelli, E., Johnsen, E. and Aguzzi, J. (2020) 'Video image enhancement and machine learning pipeline for underwater animal detection and classification at cabled observatories', *Sensors*, 20(3), p.726.

Luiten, J., Osep, A., Dendorfer, P., Torr, P., Geiger, A., Leal-Taixé, L. and Leibe, B. (2021) 'Hota: A higher order metric for evaluating multi-object tracking', *International journal of computer vision*, 129(2), pp. 548-578.

Lumini, A. and Nanni, L. (2019) 'Deep learning and transfer learning features for plankton classification', *Ecological informatics*, 51, pp. 33-43.

Lumini, A., Nanni, L. and Maguolo, G. (2020) 'Deep learning for plankton and coral classification', *Applied Computing and Informatics*. https://doi.org/10.1016/j.aci.2019.11.004

Luo, J.Y., Irisson, J.O., Graham, B., Guigand, C., Sarafraz, A., Mader, C. and Cowen, R.K. (2018) 'Automated plankton image analysis using convolutional neural networks', *Limnology and Oceanography: Methods*, 16(12), pp. 814-827.

Lütjens, M. and Sternberg, H. (2021) 'Deep Learning based Detection, Segmentation and Counting of Benthic Megafauna in Unconstrained Underwater Environments', *IFAC-PapersOnLine*, 54(16), pp. 76-82.

MacLeod, N., Benfield, M. and Culverhouse, P. (2010) 'Time to automate identification', *Nature*, 467(7312), pp. 154-155.

Mahmood, A., Bennamoun, M., An, S., Sohel, F., Boussaid, F., Hovey, R., Kendrick, G. and Fisher, R.B. (2016) 'Coral classification with hybrid feature representations', In *2016 IEEE International Conference on Image Processing (ICIP)*, pp. 519-523).

Mahmood, A., Bennamoun, M., An, S., Sohel, F., Boussaid, F., Hovey, R., Kendrick, G. and Fisher, R.B. (2017) 'Deep learning for coral classification', In *Handbook of neural computation*, Academic Press, pp. 383-401.

Malde, K., Handegard, N.O., Eikvil, L. and Salberg, A.B. (2020) 'Machine intelligence and the data-driven future of marine science', *ICES Journal of Marine Science*, 77(4), pp. 1274-1285

Mandal, R., Connolly, R.M., Schlacher, T.A. and Stantic, B. (2018) 'Assessing fish abundance from underwater video using deep neural networks', In *2018 International Joint Conference on Neural Networks (IJCNN)* IEEE, pp. 1-6.

Marburg, A. and Bigham, K. (2016) 'Deep learning for benthic fauna identification', In *OCEANS 2016 MTS/IEEE Monterey* IEEE, pp. 1-5.

Matabos, M., Hoeberechts, M., Doya, C., Aguzzi, J., Nephin, J., Reimchen, T.E., Leaver, S., Marx, R.M., Branzan Albu, A., Fier, R. and Fernandez-Arcaya, U. (2017) 'Expert, Crowd, Students or Algorithm: who holds the key to deep-sea imagery ‘big data’ processing?’, *Methods in Ecology and Evolution*, 8(8), pp.996-1004.

Mathur, M., Vasudev, D., Sahoo, S., Jain, D. and Goel, N. (2020) ‘Crosspooled FishNet: transfer learning based fish species classification model’, *Multimedia Tools and Applications*, 79(41), pp. 31625-31643.

McCulloch, W.S. and Pitts, W. (1943) ‘A logical calculus of the ideas immanent in nervous activity’, *The bulletin of mathematical biophysics*, 5(4), pp. 115-133.

McNeely-White, D., Beveridge, J.R. and Draper, B.A. (2020) ‘Inception and ResNet features are (almost) equivalent’, *Cognitive Systems Research*, 59, pp. 312-318.

Minsky, M. and Papert, S. (1969) ‘An introduction to computational geometry’, *Cambridge tiass*, HIT, 479, p.480.

Mitra, R., Marchitto, T.M., Ge, Q., Zhong, B., Kanakiya, B., Cook, M.S., Fehrenbacher, J.S., Ortiz, J.D., Tripati, A. and Lobaton, E. (2019) ‘Automated species-level identification of planktic foraminifera using convolutional neural networks, with comparison to human performance’, *Marine Micropaleontology*, 147, pp.16-24.

Modasshir, M., Rahman, S., Youngquist, O. and Rekleitis, I. (2018) ‘Coral identification and counting with an autonomous underwater vehicle’, In *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 524-529.

Moniruzzaman, M., Islam, S.M.S., Bennamoun, M. and Lavery, P. (2017) ‘Deep learning on underwater marine object detection: A survey’, In *International Conference on Advanced Concepts for Intelligent Vision Systems*, Springer, Cham, pp. 150-160.

Montserrat, D.M., Lin, Q., Allebach, J. and Delp, E.J. (2017) ‘Training object detection and recognition CNN models using data augmentation’, *Electronic Imaging*, 2017(10), pp. 27-36.

Morris, K.J., Bett, B.J., Durden, J.M., Huvenne, V.A., Milligan, R., Jones, D.O., McPhail, S., Robert, K., Bailey, D.M. and Ruhl, H.A. (2014) ‘A new method for ecological surveying of the abyss using autonomous underwater vehicle photography’, *Limnology and Oceanography: Methods*, 12(11), pp.795-809.

Munim, Z.H., Dushenko, M., Jimenez, V.J., Shakil, M.H. and Imset, M. (2020) ‘Big data and artificial intelligence in the maritime industry: a bibliometric review and future research directions’, *Maritime Policy & Management*, 47(5), pp. 577-597.

Ning, C., Zhou, H., Song, Y. and Tang, J. (2017) ‘Inception single shot multi-box detector for object detection’, In *2017 IEEE International Conference on Multimedia & Expo Workshops (ICMEW*), pp. 549-554.

Ontrup, J., Ehnert, N., Bergmann, M. and Nattkemper, T.W. (2009) ‘BIIGLE-Web 2.0 enabled labelling and exploring of images from the Arctic deep-sea observatory Hausgarten’, In *OCEANS 2009-EUROPE* IEEE, pp. 1-7.

Oztel, I., Yolcu, G. and Oz, C. (2019) ‘Performance comparison of transfer learning and training from scratch approaches for deep facial expression recognition’, In *2019 4th International Conference on Computer Science and Engineering (UBMK)* IEEE, pp. 1-6.

Padilla, R., Netto, S.L. and da Silva, E.A. (2020) ‘A survey on performance metrics for object-detection algorithms’, In *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)* IEEE, pp. 237-242.

Pan, S.J. (2020) ‘Transfer learning’, *Learning*, 21, pp. 1-2.

Paulus, E. (2021) ‘Shedding Light on Deep-Sea Biodiversity—A Highly Vulnerable Habitat in the Face of Anthropogenic Change’, *Frontiers in Marine Science*, 8, p.667048.

Picek, L., Říha, A. and Zita, A. (2020) ‘Coral Reef annotation, localisation and pixel-wise classification using Mask R-CNN and Bag of Tricks’, In *CLEF (Working Notes)*.

Piechaud, N., Hunt, C., Culverhouse, P.F., Foster, N.L. and Howell, K.L. (2019) ‘Automated identification of benthic epifauna with computer vision’, *Marine Ecology Progress Series*, 615, pp. 15-30.

Probst, W.N. (2020) ‘How emerging data technologies can increase trust and transparency in fisheries’, *ICES Journal of Marine Science*, 77(4), pp. 1286-1294.

Py, O., Hong, H. and Zhongzhi, S. (2016) ‘Plankton classification with deep convolutional neural networks’, In *2016 IEEE Information Technology, Networking, Electronic and Automation Control Conference*, pp. 132-136.

Qiang, B., Chen, R., Zhou, M., Pang, Y., Zhai, Y. and Yang, M. (2020) ‘Convolutional Neural Networks-Based Object Detection Algorithm by Jointing Semantic Segmentation for Images’, *Sensors*, 20(18), p.5080.

Qin, H., Li, X., Liang, J., Peng, Y. and Zhang, C. (2016) ‘DeepFish: Accurate underwater live fish recognition with a deep architecture’, *Neurocomputing*, 187, pp. 49-58.

Raitoharju, J., Riabchenko, E., Ahmad, I., Iosifidis, A., Gabbouj, M., Kiranyaz, S., Tirronen, V., Ärje, J., Kärkkäinen, S. and Meissner, K. (2018) ‘Benchmark database for fine-grained image classification of benthic macroinvertebrates’, *Image and Vision Computing*, 78, pp. 73-83.

Rajaraman, S., Antani, S.K., Poostchi, M., Silamut, K., Hossain, M.A., Maude, R.J., Jaeger, S. and Thoma, G.R. (2018) 'Pre-trained convolutional neural networks as feature extractors toward improved malaria parasite detection in thin blood smear images', *PeerJ*, 6, p.e4568.

Rampasek, L. and Goldenberg, A. (2016) 'Tensorflow: Biology's gateway to deep learning?', *Cell systems*, 2(1), pp. 12-14.

Raphael, A., Dubinsky, Z., Iluz, D. and Netanyahu, N.S. (2020) 'Neural Network Recognition of Marine Benthos and Corals', *Diversity*, 12(1), p.29.

Rathi, D., Jain, S. and Indu, S. (2017) 'Underwater fish species classification using convolutional neural network and deep learning', In *2017 Ninth International Conference on Advances in Pattern Recognition (ICAPR)* IEEE, pp. 1-6.

Raybaut, P. (2009) 'Spyder-documentation'. *Source: pythonhosted.org*.

Raza, K. and Hong, S. (2020) 'Fast and accurate fish detection design with improved YOLO-v3 model and transfer learning', *International Journal of Advanced Computer Science and Applications*, 11, pp. 7-16.

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

Redmon, J. and Farhadi, A. (2017) 'YOLO9000: better, faster, stronger', In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7263-7271.

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

Ren, S., He, K., Girshick, R. and Sun, J. (2016) ‘Faster r-cnn: Towards real-time object detection with region proposal networks’, *IEEE transactions on pattern analysis and machine intelligence*, 39(6), pp. 1137-1149.

Riaboff, L., Aubin, S., Bedere, N., Couvreur, S., Madouasse, A., Goumand, E., Chauvin, A. and Plantier, G. (2019) ‘Evaluation of pre-processing methods for the prediction of cattle behaviour from accelerometer data’, *Computers and Electronics in Agriculture*, 165, p.104961.

Rice, L., Wong, E. and Kolter, Z. (2020) ‘Overfitting in adversarially robust deep learning’, In *International Conference on Machine Learning* PMLR, pp. 8093-8104.

Rosenblatt, F. (1958) ‘The perceptron: a probabilistic model for information storage and organization in the brain’, *Psychological review*, 65(6), p.386.

Rosenblatt, F. (1961) ‘Principles of neurodynamics. perceptrons and the theory of brain mechanisms’, *Cornell Aeronautical Lab Inc Buffalo NY*, Report Number: VG-1196-G-8.

Rumelhart, D.E., Hinton, G.E. and Williams, R.J. (1988) ‘Neurocomputing: Foundations of Research’, *Cambridge, MA, USA: MIT Press*, Editor James A. Anderson and Edward Rosenfeld, pp. 696–699.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. and Berg, A.C. (2015) ‘Imagenet large scale visual recognition challenge’, *International journal of computer vision*, 115(3), pp. 211-252.

Salman, A., Jalal, A., Shafait, F., Mian, A., Shortis, M., Seager, J. and Harvey, E. (2016) ‘Fish species classification in unconstrained underwater environments based on deep learning’, *Limnology and Oceanography: Methods*, 14(9), pp. 570-585.

Salman, A., Maqbool, S., Khan, A.H., Jalal, A. and Shafait, F. (2019) ‘Real-time fish detection in complex backgrounds using probabilistic background modelling’, *Ecological Informatics*, 51, pp. 44-51.

Sanchez, S.A., Romero, H.J. and Morales, A.D. (2020) ‘A review: Comparison of performance metrics of pretrained models for object detection using the TensorFlow framework’, *In IOP Conference Series: Materials Science and Engineering*, IOP Publishing, 844(1), p.012024.

Schlining, B.M. and Stout, N.J. (2006) ‘MBARI's video annotation and reference system’, In *OCEANS 2006* IEEE, pp. 1-5.

Schoening, T., Ehnert, N., Ontrup, J. and Nattkemper, T.W. (2009) ‘BIIGLE Tools–a Web 2.0 approach for visual Bioimage database mining’, In *2009 13th International Conference Information Visualisation* IEEE, pp. 51-56.

Schoening, T., Bergmann, M., Ontrup, J., Taylor, J., Dannheim, J., Gutt, J., Purser, A. and Nattkemper, T.W. (2012) ‘Semi-automated image analysis for the assessment of megafaunal densities at the Arctic deep-sea observatory HAUSGARTEN’, *PloS one*, 7(6), p.e38179.

Schoening, T., Osterloff, J. and Nattkemper, T.W. (2016) ‘RecoMIA—Recommendations for marine image annotation: Lessons learned and future directions’, *Frontiers in Marine Science*, 3, p.59.

Schoening, T., Köser, K. and Greinert, J. (2018) ‘An acquisition, curation and management workflow for sustainable, terabyte-scale marine image analysis’*, Scientific data*, 5(1), pp.1-12.

Serbetci, A. and Akgul, Y.S. (2020) ‘End-to-end training of CNN ensembles for person re-identification’, *Pattern Recognition*, 104, p.e107319.

Shafiee, M.J., Chywl, B., Li, F. and Wong, A. (2017) ‘Fast YOLO: A fast you only look once system for real-time embedded object detection in video’, *arXiv preprint arXiv:1709.05943*.

Shahriar, M.T. and Li, H. (2020) ‘A Study of Image Pre-processing for Faster Object Recognition’, *arXiv preprint arXiv:2011.06928*.

Shashidhara, B.M., Scott, M. and Marburg, A. (2020) ‘Instance segmentation of benthic scale worms at a hydrothermal site’, In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 1314-1323.

Shin, H.C., Roth, H.R., Gao, M., Lu, L., Xu, Z., Nogues, I., Yao, J., Mollura, D. and Summers, R.M. (2016) ‘Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning’, *IEEE transactions on medical imaging*, 35(5), pp. 1285-1298.

Shorten, C. and Khoshgoftaar, T.M. (2019) ‘A survey on image data augmentation for deep learning’, *Journal of big data*, 6(1), pp. 1-48.

Shu, M. (2019) ‘Deep learning for image classification on very small datasets using transfer learning’, *Creat. Components, Jan*.

Siddiqui, S.A., Salman, A., Malik, M.I., Shafait, F., Mian, A., Shortis, M.R. and Harvey, E.S. (2018) ‘Automatic fish species classification in underwater videos: exploiting pre-trained deep neural network models to compensate for limited labelled data’, *ICES Journal of Marine Science*, 75(1), pp. 374-389.

Sosik, H.M., Peacock, E.E. and Brownlee, E.F. (2021) 'WHOI Plankton: Annotated Plankton Images-Data Set for Developing and Evaluating Classification Methods', *Woods Hole Open Access Server.* Source: *https://darchive.mblwhoilibrary.org/handle/1912/7341.*

Sultana, F., Sufian, A. and Dutta, P. (2018) 'Advancements in image classification using convolutional neural network', In *2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)* IEEE, pp. 122-129.

Sung, M., Yu, S.C. and Girdhar, Y. (2017) 'Vision based real-time fish detection using convolutional neural network', In *OCEANS 2017-Aberdeen* IEEE, pp. 1-6.

Szegedy, C., Reed, S., Erhan, D., Anguelov, D. and Ioffe, S. (2014) 'Scalable, high-quality object detection', *arXiv preprint arXiv:1412.1441*.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A. (2015) 'Going deeper with convolutions', In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1-9.

Szegedy, C., Ioffe, S., Vanhoucke, V. and Alemi, A. (2017) 'Inception-v4, inception-ResNet and the impact of residual connections on learning', In *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1).

Taylor, L. and Nitschke, G. (2018) 'Improving deep learning with generic data augmentation', In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)* IEEE, pp. 1542-1547.

Tharwat, A., Hemedan, A.A., Hassanien, A.E. and Gabel, T. (2018) 'A biometric-based model for fish species classification', *Fisheries Research*, 204, pp. 324-336.

Tseng, C.H. and Kuo, Y.F. (2020) ‘Detecting and counting harvested fish and identifying fish types in electronic monitoring system videos using deep convolutional neural networks’, *ICES Journal of Marine Science*, 77(4), pp. 1367-1378.

Tuia, D., Kellenberger, B., Beery, S., Costelloe, B.R., Zuffi, S., Risse, B., Mathis, A., Mathis, M.W., van Langevelde, F., Burghardt, T. and Kays, R. (2022) ‘Perspectives in machine learning for wildlife conservation’, *Nature Communications*, 13(1), pp. 1-15.

Verhaegen, G., Cimoli, E. and Lindsay, D. (2021) ‘Life beneath the ice: jellyfish and ctenophores from the Ross Sea, Antarctica, with an image-based training set for machine learning’, *Biodiversity Data Journal*, 9.

Villon, S., Chaumont, M., Subsol, G., Villéger, S., Claverie, T. and Mouillot, D. (2016) ‘Coral reef fish detection and recognition in underwater videos by supervised machine learning: Comparison between Deep Learning and HOG+ SVM methods’, In *International Conference on Advanced Concepts for Intelligent Vision Systems*, Springer, Cham, pp. 160-171.

Villon, S., Mouillot, D., Chaumont, M., Darling, E.S., Subsol, G., Claverie, T. and Villéger, S. (2018) ‘A Deep Learning algorithm for accurate and fast identification of coral reef fishes in underwater videos’, *PeerJ Preprints*, p.e26818v1.

Voulodimos, A., Doulamis, N., Doulamis, A. and Protopapadakis, E. (2018) ‘Deep learning for computer vision: A brief review’, *Computational intelligence and neuroscience*.

Wang, C., Zheng, X., Guo, C., Yu, Z., Yu, J., Zheng, H. and Zheng, B. (2018) ‘Transferred parallel convolutional neural network for large imbalanced plankton database classification’, In *2018 OCEANS-MTS/IEEE Kobe Techno-Oceans (OTO)* IEEE, pp. 1-5.

Wang, Z., Zheng, L., Liu, Y., Li, Y. and Wang, S. (2020) ‘Towards real-time multi-object tracking’, In *European Conference on Computer Vision,* Springer, Cham, pp. 107-122.

Wang, N., Wang, Y. and Er, M.J. (2022) ‘Review on deep learning techniques for marine object recognition: Architectures and algorithms’, *Control Engineering Practice*, 118, p.104458.

Weinstein, B.G. (2018) 'A computer vision for animal ecology'. *Journal of Animal Ecology*, 87(3), pp. 533-545.

Wen, Y., Anderson, A., Radu, V., O'Boyle, M.F. and Gregg, D. (2020) ‘TASO: Time and Space Optimization for Memory-Constrained DNN Inference’, In *2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pp. 199-208.

Werbos, P. (1974) ‘Beyond regression: new tools for prediction and analysis in the behavioural sciences’, *Ph.D. dissertation, Harvard University*.

Werbos, P.J. (1990) ‘Backpropagation through time: what it does and how to do it’, *Proceedings of the IEEE*, 78(10), pp. 1550-1560.

Whitt, C., Pearlman, J., Polagye, B., Caimi, F., Muller-Karger, F., Copping, A., Spence, H., Madhusudhana, S., Kirkwood, W., Grosjean, L. and Fiaz, B.M. (2020) ‘Future vision for autonomous ocean observations’, *Frontiers in Marine Science*, p.697.

Wickham, H. (2017) 'Tidyverse: Easily install and load’tidyverse’packages', *R package version, 1(1)*.

Xia, C., Fu, L., Liu, H. and Chen, L. (2018) ‘In situ sea cucumber detection based on deep learning approach’, In *2018 OCEANS-MTS/IEEE Kobe Techno-Oceans (OTO)* IEEE, pp. 1-4.

Xie, S., Girshick, R., Dollár, P., Tu, Z. and He, K. (2017) 'Aggregated residual transformations for deep neural networks', In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1492-1500.

Xuhong, L.I., Grandvalet, Y. and Davoine, F. (2018) 'Explicit inductive bias for transfer learning with convolutional networks', In *International Conference on Machine Learning* PMLR, pp. 2825-2834.

Yakovlev, A. and Lisovychenko, O. (2020) 'An approach for image annotation automatization for artificial intelligence models learning', *Adaptive automatic control systems*, 1(36), pp. 32-40.

Yamashita, R., Nishio, M., Do, R.K.G. and Togashi, K. (2018) 'Convolutional neural networks: an overview and application in radiology', *Insights into imaging*, 9(4), pp. 611-629.

Zhang, D., Lee, D.J., Zhang, M., Tippetts, B.J. and Lillywhite, K.D. (2016) 'Object recognition algorithm for the automatic identification and removal of invasive fish', *Biosystems Engineering*, 145, pp. 65-75.

Zhao, J., Li, Y., Zhang, F., Zhu, S., Liu, Y., Lu, H. and Ye, Z. (2018) 'Semi-supervised learning-based live fish identification in aquaculture using modified deep convolutional generative adversarial networks', *Transactions of the ASABE*, 61(2), pp. 699-710.

Zhao, Z.Q., Zheng, P., Xu, S.T. and Wu, X. (2019) 'Object detection with deep learning: A review', *IEEE transactions on neural networks and learning systems*, 30(11), pp. 3212-3232.

Zhuang, P., Xing, L., Liu, Y., Guo, S. and Qiao, Y. (2017) 'Marine Animal Detection and Recognition with Advanced Deep Learning Models', In *CLEF (Working Notes)*.

Zurowietz, M., Langenkämper, D., Hosking, B., Ruhl, H.A. and Nattkemper, T.W. (2018) ‘MAIA—A machine learning assisted image annotation method for environmental monitoring and exploration’, *PloS one*, 13(11), p.e0207498.