UiA University of Agder

Master

Hierarchical Fish Species Detection in Real-Time Video Using YOLO

ESPEN STAUSLAND KALHAGEN,
ØRJAN LANGØY OLSEN

SUPERVISOR
Morten Goodwin

University of Agder, 2020
Faculty of Engineering and Science
Department of Information and Communication Technology

Abstract

Information gathering of aquatic life is often based on time consuming methods with a foundation in video feeds. It would be beneficial to capture more information in a cost effective manner from video feeds, and video object detection has an opportunity to achieve this. Recent research has shown promising results with the use of YOLO for object detection of fish. As under-water conditions can be difficult and fish species hard to discriminate, we propose the use of a hierarchical structures in both the classification and the dataset to gain valuable information. With the use of hierarchical classification and other techniques we present YOLO Fish. YOLO Fish is a state of the art object detector on nordic fish species, with an mAP of 91.8%. For a more stable video, YOLO Fish can be used with the object tracking algorithm SORT. This results in a complete fish detector for real-time video.

Table of Contents

List of Figures

List of Tables

Nomenclature

AP Average Precision – Metric for object detector performance.

Class smoothing Applying the most common class for a tracked object.

COCO Common Objects in Context – Object detection competition and dataset.

FPS Frames per Second – The number of frames that can be processed per second.

GFLOPS Giga floating point operations - How many giga floating point operations.

Hierarchical classification The use of hierarchical relations between classes to aid in classification.

IoU Intersection over Union – Metric for bounding box prediction similarity.

LRP Localization Recall Precision – Another metric for object detector performance that also takes localization error into account.

mAP Mean Average Precision – Another name for average precision.

MOT Multiple Object Tracking algorithms – Algorithms that track objects across frames.

NMS Non-Maximum Suppression – Algorithm for filtering abundant detections.

Object detection Localization and classification of objects in visual media.

oLRP Optimal LRP – The best achievable LRP based on the probability threshold where the algorithm performs the best.

Performance How well the algorithm performs at the task at hand.

Precision The proportion of selected objects that are relevant.

Recall The proportion of the relevant objects that are selected.

Soft-NMS Algorithm for filtering abundant detections where they are assigned lower probabilities based on overlap instead of completely removed.

SORT Simple Online Realtime Tracking – Algorithm for tracking multiple objects in video.

Speed The inference time of a network.

Taxa A node in the biological hierarchical system.

Taxonomy Biological hierarchical system.

VOC Visual Object Classes – Object detection competition and dataset.

WordTree An implementation of hierarchical classification in YOLO.

WT Initialism for WordTree.

YOLO You Only Look Once – Algorithm for object detection.

1 Introduction

Collecting information about aquatic life is highly important for sustainable and profitable marine life management [53][39]. Currently this information gathering is done manually with expensive techniques and limited information often with a basis in video feeds [39][55]. Despite previous research into deep learning for fish localization and classification in images, few algorithms have seen use in the real world. Previous algorithms struggle with handling uncertainties and background fish, reducing their usefulness. Additionally, video of fish is generated continuously by underwater cameras. For AI techniques to be cost effective and scalable, processing of real-time videos of fish would be useful. However, only limited research of real-time processing of underwater video has been conducted.

Over the past years AI have made great progress in localization and classification of objects in visual media. Object detection above water is able to determine objects accurately and quickly. Previous research has performed well in classification of fish in good conditions on unique looking fish [37]. However, for these techniques to be useful for data collection the algorithms need to perform well in difficult conditions where fish species are challenging to tell apart, even for humans, because of inherent properties of seeing underwater and because of fish looking very similar. Furthermore, for real-time processing of video the algorithm must also be fast.

As a result of the challenging conditions, applying object detection algorithms to this environment is difficult. To achieve the goals of the thesis, a novel approach for training artificial intelligence on biological fish species are applied together with state of the art object detection algorithms and techniques to create YOLO Fish. This novel approach takes advantage of biological hierarchies to deal with similarities between species and difficult conditions. It applies a more general class as certainty of species decreases to increase the confidence that can be placed in the results. Additionally, the employed techniques allow for real-time processing of video.

1.1 Problem Statement

This thesis assesses whether real-time fish detectors' performance can be improved. This is done through answering the following problem statement. Is it possible to use You Only Look Once (YOLO) [45] with hierarchical classification and various other techniques to improve performance? And, does the hierarchical properties yield benefits in cases where species can not be easily discerned?

The thesis evaluates the following hypotheses.

H1 Hierarchical clasification can be used to train on fish data where the species is not given, but rather a class of higher taxonomic rank is used.

H2 Hierarchical classification can be used to increase the precision and quality of fish prediction.

H3 The introduction of classless NMS mitigates the creation of erroneous bounding boxes as a result of hierarchical classification.

H4 The benefits of the improvements made in YOLOv3 can be utilized while also keeping the benefits of YOLOv2's WordTree by designing a new network that incorporates both.

H5 Soft-NMS improves the performance of YOLOv3 for detecting fish.

H6 It is possible to achieve state of the art accuracy in detection of Nordic fish species in real-time videos using the YOLO architecture.

H7 Using SORT for object tracking and assigning the most common class for each tracked object can increase detection stability on video.

1.2 Contributions

This thesis achieves state of the art performance for fish object detection by using various techniques in a novel way. The following list summarizes the contributions.

- An end-to-end model for real-time localization and classification of Nordic fish species in video named YOLO Fish.
- State of the art performance on real-time localization and classification of Nordic fish in video.
- The use of biological hierarchy for data annotation to make data generation significantly easier for marine life.
- A method that enables training and detection under water, where species is not always easily discernible.

2 Background

The purpose of this chapter is to introduce the reader to the theory and give an understanding of the technology that was used in this thesis. The chapter covers convolutional neural networks in general, the specifics of YOLO, some general techniques like hierarchical classification, NMS and Soft-NMS, and object tracking. Followed by an explanation of the performance measures and lastly some theory on biological taxonomies and how they relate to fish.

2.1 Convolutional Neural Networks

Convolutional neural networks are networks that mainly use convolutional layers. A convolutional layer is a layer that is based on the convolution operator. When working with image data, discrete 2D convolutions are used. But in most implementations cross correlation is used for efficiency [18]. This can be seen in equation 1 which shows the operation on a single channel input. S is the output layer, K is the filter kernel, I is the input image, m and n are the position in the filter, and i and j are the position in the image.

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{1}$$

A convolution can extract useful information out of an image given a filter. In a convolutional neural network, a filter is not specified, but is rather weights that are learnt during the training process, to extract features that it deems most important given the training set and the loss function.

A convolutional layer is defined by a number of filters, a stride and a padding. The number of filters is how many different filters should be learnt and also determines the output depth. The stride is the number of steps that is taken between each sample the filter makes. Padding is whether to add a padding of zeros around the input data and how large of a padding. Padding is usually used to avoid losing information near the edges.

A 2D convolution can be illustrated by filters that are 3D cubes that moves over a 3D image and each calculates one output value per step in its stride. Figure 1 shows an example of this where a filter is convolved with a portion of the input image to produce one value in the output. The image is 3D in the sense that there are two positional dimensions and one color dimension and the convolution is 2D in the sense that the filter moves in the 2D space of the input.

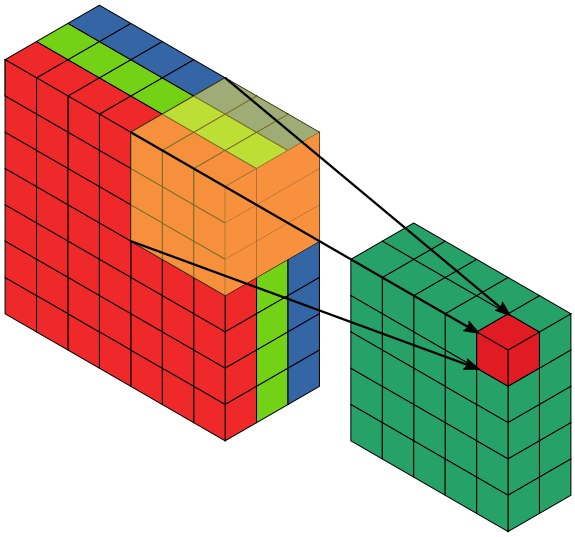


Figure 1: 2D convolution over a 3 channel image with 2 filters of size 3 resulting in an output with a depth of 2.

Convolutional layers have properties that makes them effective at image feature extraction, two of these are the parameter sharing and the sparse connectivity properties [18]. Parameter sharing means that each weight is used multiple times during a single pass. This comes from the fact that the weights make up the filters and the filters are moved across the image. Sparse connectivity is the opposite of dense connectivity and means that it doesn't have as many connections to the next layer. This is because each area the filter overlaps connects to just one node in the next layer, as seen in 1.

A third helpful property is that convolutional layers are equivariant to translation [18] and this means that if the input is translated, the output will also be translated. In practice, this means that the layer will detect features regardless of its position in the data. Note that the convolutional layer is not rotationally invariant.

The layers in a convolutional neural network are often laid out so that they become smaller and smaller in size, but larger in depth as you traverse deeper into the network. The first layers often extract low level features like edges and color tones, while the later layers often extract high level features like shapes and objects. This property makes them able to extract most of the features present in an image [61].

2.2 You Only Look Once (YOLO)

YOLO [45] is a state of the art algorithm for object detection. The algorithm has seen some incremental improvements since its first release with accompanying published papers. The versions are called YOLOv1 [45], YOLOv2 (YOLO9000) [43] and YOLOv3 [44].

There has been some significant changes from one version to the other and are

briefly summarized as follows. YOLOv2 is predicting how to move and scale some prior anchor boxes relatively, instead of directly predicting bounding boxes [43]. YOLO9000 added the WordTree concept to predict classes that are hierarchically organized and was designed to be able to classify over 9000 classes [43]. YOLOv3 added multi-class and multi-scale prediction [44].

The rest of this section will explain how YOLO works. This will be how the configuration used in our solution is implemented in the YOLO author's Darknet framework [42]. The reason why we distinguish between the different versions of YOLO is because in the Darknet framework features from the different versions can be both mixed and used simultaneous, and this is something we take advantage of in this project.

2.2.1 The Architecture

The YOLO architecture can be divided into a feature extractor and an object detector. The feature extractor relies on the Darknet-53 architecture in figure 2. This is a modern deep convolutional neural network that utilizes techniques as batch normalization, residual blocks (or shortcut connections) and leaky Rectified Linear Unit (ReLU). [44]

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	608 × 608
	Convolutional	64	3 × 3 / 2	304 × 304
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			304 × 304
	Convolutional	128	3 × 3 / 2	152 × 152
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			152 × 152
	Convolutional	256	3 × 3 / 2	76 × 76
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			76 × 76 → ③
	Convolutional	512	3 × 3 / 2	38 × 38
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			38 × 38 → ②
	Convolutional	1024	3 × 3 / 2	19 × 19
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			19 × 19

→ ①

Figure 2: The Darknet-53 feature extractor with an input layer of 608x608x3. In YOLOv3 the numbered connections are connected to the numbers in figure 3. This figure is a modified version of one found in [44].

The object detector part of YOLO is doing detections at different scales as seen in figure 3. After the feature extractor and at the smallest scale, the first prediction is done. Then it is up-sampled and concatenated with an earlier and higher resolution layer from the feature extractor. This is then used to do detection on the middle scale. The same process is repeated for the highest scale, and thus it ends up with a 3-scale detector.

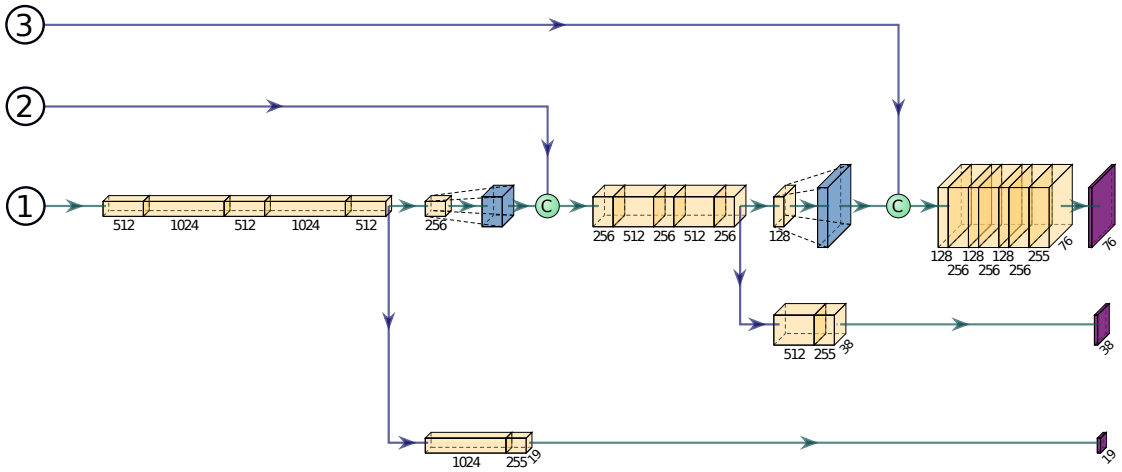


Figure 3: The YOLOv3 object detector. The yellow blocks are convolutional layers, the blue blocks are up-sampling layers, the 'C' operators are concatenations with a previous layer, and the purple blocks are YOLO prediction layers. The numbers are the dimensions of the output of each layer. It is connected to the Darknet-53 feature extractor via the numbered connections.

2.2.2 Activations and Weight Initialization

The YOLO architecture only consists of one type of layer with learnable weights, that is the convolutional layers. The activation function used for these layers are all leaky ReLU. The advantage of using ReLU over say logistic activation in the hidden layers is that logistic activation suffers from the vanishing gradient problem. However ReLU also introduces a problem in that the optimizer will not adjust the weights if the neuron never activates. This problem is mitigated by using a leaky ReLU activation as show in equation 2. This is because it allows for a small non-zero gradient when the unit otherwise would not be activated. [32]

$$\text{LeakyReLU}(x) = \begin{cases} x & x > 0 \\ 0.01x & \text{else} \end{cases} \tag{2}$$

Weight initialization is important for avoiding the problem of vanishing and exploding gradient. The choice of weight initialization is dependent upon which activation function is used. He et al. [19] has shown that He initialization works well with leaky ReLU. Under He initialization, the weights W are initialized randomly from the normal distribution as defined in equation 3. Fan in is the number of inputs from the previous layer. Because YOLO uses leaky ReLU this weight initialization is used.

$$W \sim \mathcal{N}\left(0, \frac{2}{\text{fan\_in}}\right) \tag{3}$$

2.2.3 Prediction and Priors

The YOLO detection layer is responsible for the predictions and in the Darket framework it goes under the name 'detection' in v1, 'region' in v2, and 'yolo' in v3. The detection layer works by dividing the input image into a SxS grid. And for each grid cell it tries to fit B bounding boxes that are defined by a position and a size, both relative, to the grid cell. For YOLOv3, S is defined as the input image size divided by 32. This comes from the transformation the convolutional layers performs on the input throughout the network.

The output is a R^4 tensor that for each bounding box in each grid cell has a vector that contains $B(5+C)$ elements as seen in equation 4. C is the number of classes. For each bounding box in a cell, the relative width, relative height, x offset and y offset, the detection confidence, and a confidence for each class is predicted.

$$P = \begin{bmatrix} t_x \\ t_y \\ t_w \\ t_h \\ P_o \\ P_1 \\ P_2 \\ \vdots \\ P_C \end{bmatrix} \tag{4}$$

The width and height of the bounding box isn't predicted directly, but rather it predicts how much to stretch prior anchor boxes in either dimension to fit the detection. Usually there is a relatively low number of prior anchor boxes, for instance 9 in v3. These prior anchor boxes are found by maximizing the average IoU they get with all the annotated bounding boxes in the dataset.

IoU is a metric for determining the accuracy of a bounding box prediction and its location as illustrated in figure 4. It is named intersection over union because it divides the intersection area by the total area to find the amount of overlap. A python implementation has been provided in listing 1 to give a better understanding of how it works.

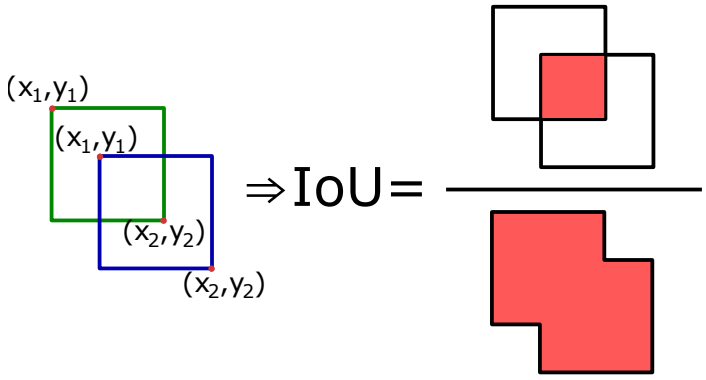


Figure 4: The IoU is calculated by dividing the area of the intersection by the area of the union.

```
def iou(box1, box2):
    inter_x1 = max(box1.x1, box2.x1)
    inter_x2 = min(box1.x2, box2.x2)
    inter_y1 = max(box1.y1, box2.y1)
    inter_y2 = min(box1.y2, box2.y2)

    inter_area = (inter_x2 - inter_x1) * (inter_y2 - inter_y1)
    box1_area = (box1.x2 - box1.x1) * (box1.y2 - box1.y1)
    box2_area = (box2.x2 - box2.x1) * (box2.y2 - box2.y1)
    union_area = box1_area + box2_area - inter_area

    iou = inter_area/float(union_area)
    return iou
```

Listing 1: A simplified Python implementation of IoU that assumes there is an overlap. Further modifications has to be done for this to be used in a real application.

Finding the prior anchor boxes with the best IoU overlap with the dataset is done with K-means. K-means normally optimises euclidean distance, but here the distance metric shown in equation 5 is used. B_1 and B_2 are the two bounding boxes that are compared. IoU maps to a number close to one when there is a high overlap and close to zero when there is a low overlap. Because we want our distance metric to have a low distance when the bounding box has a high overlap, we use 1 minus the IoU. How K-means clustering works can be seen in algorithm 1.

$$d(B_1, B_2) = 1 - \text{IoU}(B_1, B_2) \tag{5}$$

Algorithm 1: The K-means clustering algorithm used to find the best prior anchor boxes.

Data: B = list of sizes of the bounding boxes from the annotated labels
k = number of anchor boxes
Result: k anchor boxes that has the best average IoU over B

```
initialize k centroids to random bounding boxes from B;
while any changes to centroids do
    define k clusters over B based on their closest centroid;
    foreach cluster do
        calculate the mean of all points in the cluster;
    end
    assign the new centroids as the means of all clusters;
end
```

As mentioned earlier, the network doesn't directly predict the bounding boxes, but rather where relative to the cell it is located and how to scale the prior anchor box to fit it. To get the correct bounding box one has to do the transformation as seen in figure 5. This involves to logistically activate the location prediction and add it to the grid cell offset to get the bounding box center. To get the bounding box dimensions one has to multiply the prior anchor box size with the exponential function of the dimension prediction.

$$
\begin{aligned}
b_x &= \sigma(t_x) + c_x \\
b_y &= \sigma(t_y) + c_y \\
b_w &= p_w e^{t_w} \\
b_h &= p_h e^{t_h}
\end{aligned}
$$

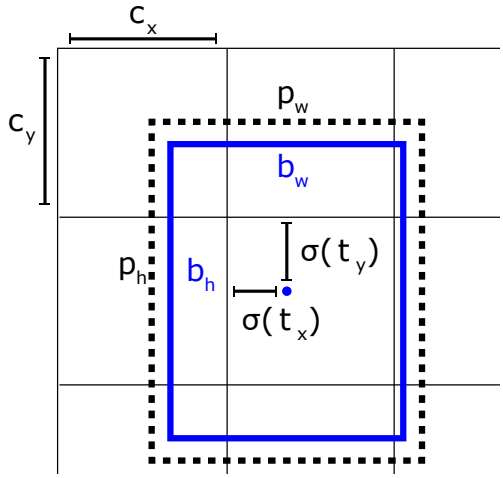


Figure 5: The transformation on the left is applied to the predicted values to get the correct bounding box. This is a figure from [43], modified to fit this section.

Using an SxS grid, each with B bounding boxes, results in a quite fine detection grid, which means that one single object may be predicted multiple times with varying IoU. This problem is mitigated by using an algorithm called non max suppression that outputs the final bounding boxes. This algorithm will be explained in section 2.4.

2.2.4 Loss Function

The loss function for YOLO that is minimized during training is shown in equation 6.

$$
\begin{aligned}
&\sum_{i=0}^{S^2}\sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \lambda'_{\text{coord}} \left[(x_{ij}-\hat{x}_{ij})^2 + (y_{ij}-\hat{y}_{ij})^2\right] \\
+&\sum_{i=0}^{S^2}\sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \lambda'_{\text{coord}} \left[(w_{ij}-\hat{w}_{ij})^2 + (h_{ij}-\hat{h}_{ij})^2\right] \\
&+\lambda_{\text{obj}} \sum_{i=0}^{S^2}\sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_{ij}-\hat{C}_{ij}\right)^2 \\
&+\lambda_{\text{noobj}} \sum_{i=0}^{S^2}\sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_{ij}-\hat{C}_{ij}\right)^2 \\
&+\lambda_{\text{class}} \sum_{i=0}^{S^2}\sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \sum_{c\in\text{classes}} \left(p_{ij}(c)-\hat{p}_{ij}(c)\right)^2
\end{aligned}
\tag{6}
$$

$\mathbb{1}_{ij}^{\text{obj}}$ is an indicator function, or boolean predicate function, that denotes whether a specific bounding box has done an object prediction. The same goes for $\mathbb{1}_{ij}^{\text{noobj}}$, when the bounding box has not done a prediction. A more verbose version of the predicate function is shown in equation 7.

$$
\mathbb{1}_{ij}^{\text{obj}} = \begin{cases} 1 & \text{if object is predicted in cell } i \text{ and box } j \\ 0 & \text{otherwise} \end{cases}
\tag{7}
$$

The algorithm is defined to have done a prediction if the IoU is higher than a given threshold. This is used to ensure that the coordinate and classification error for that anchor box is only penalized if it think there's an object in the prediction. YOLO is an end-to-end learning technique and as such there is no explicit logic that tells it to only predict a bounding box if it thinks there is an object. The network will thus predict some insignificant numbers for all bounding boxes that doesn't have objects. That is why it is important to not penalize this.

The loss function is mostly the sum of squared errors with some small deviations. The constituent parts of the loss function will now be explained. The first two terms in equation 6 are the localization loss and because of the object predicate, it is only used if the algorithm has decided that the current prediction is an object. Here λ'_{coord} is defined as $\lambda_{\text{coord}}(2 - w'_{ij}h'_{ij})$ in the Darknet framework. w'_{ij} and h'_{ij} is the width and height relative to the image size, and λ_{coord} is a parameter that specifies how much the localization error is going to be penalized in the loss function. This will effectively mean that the smaller the ground truth box is, the larger the factor will be. There are similar parameters for the other terms as

well. λ_{obj}, λ_{noobj} and λ_{class} determines how much each respective part of the loss function should be weighted in the total loss. This can be helpful to mitigate class imbalance problems such as images containing a lot more background than actual objects.

The next two terms are the confidence loss. This sums the squared error of the confidence given to the predictions. The term with $\mathbb{1}_{ij}^{\text{obj}}$ is used if an object has been predicted. Here the ground truth C_{ij} is defined as 1. If the rescore parameter of the region layer in the Darknet framework is used, then C_{ij} is defined as the IoU of the ground truth and the prediction. This means that the the optimizer will try to predict the IoU of a given prediction instead of the objectness/confidence. If no object has been predicted, the term with $\mathbb{1}_{ij}^{\text{noobj}}$ is used. Here the ground truth C_{ij} is defined as 1.

The last term is the classification loss. It sums up the squared error of the probabilities predicted for each class. The ground truth is 1 for the correct class and 0 for all others. If hierarchical classification is used, the ancestor classes of the ground truth labeled class, is also included. Here the ground truth is 1 for the labeled class and all its ancestors, while for the other classes it is 0.

2.3 Hierarchical Classification in YOLO

Most convolutional neural networks use a flat structure to classification, but a hierarchical structure can also be useful when modeling domains where hierarchical structures are the norm. Semantic relationships between words in a WordTree is one such domain where this structure can be useful [43]. This structure can be designed such that each node has a "is a" relationship to its parent. For example a fish is an animal, so thus 'fish' can have 'animal' as parent.

$$P(\text{Node}|\text{Parent}) \tag{8}$$

YOLO's WordTree uses hierarchical probabilities to apply the most specific class (furthest down in the tree) that still is above some hierarchical threshold. To do this it calculates conditional probabilities as in equation 8 for each node in the WordTree starting with the root. It does this recursively down the tree, following the node with highest probability, as seen in equation 9. At the point where the calculated probability is lower than the hierarchical threshold the recursion is stopped and the parent class is returned. As a result the most specific node that can confidently be selected is returned and the probability of this node is based on itself and its parents. [43]

$$\begin{aligned} & P(\text{Pollachius virens}) \\ = {} & P(\text{Pollachius virens}|\text{Pollachius}) \\ & \cdot P(\text{Pollachius}|\text{Gadidae}) \\ & \cdot P(\text{Gadidae}|\text{Fish}) \end{aligned} \tag{9}$$

Most softmax implementation is applied across all class probabilities to get a probability distribution, resulting in a single-class predictor. To be able to do hierarchical classification efficiently YOLO's WordTree applies multiple softmax across siblings as in 6. This makes each level stand alone and makes it possible for the detector to discriminate classes accurately. [43]

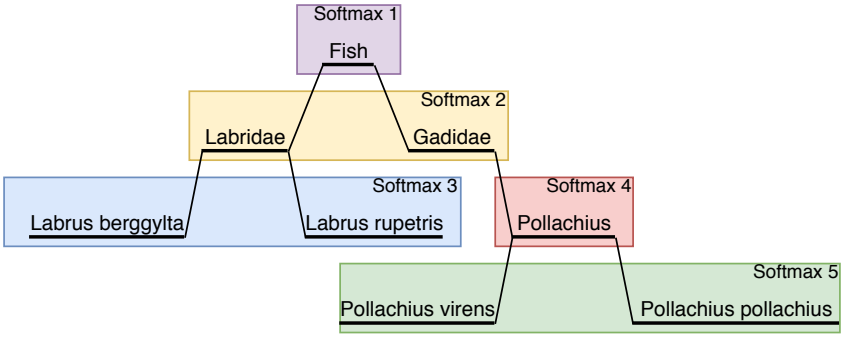


Figure 6: Softmax is applied across sibling nodes in WordTree.

2.4 NMS and Soft-NMS

Object detection mechanisms often produce a high number of bounding boxes with probabilities based on the features present in that area, often many more than the number of objects. For this reason non max suppression (NMS) is often used as a post processing technique to remove excessive predictions [6]. NMS takes a list of boxes with probabilities and it removes the boxes that overlap more than some threshold, keeping the boxes with the highest probabilities [6]. The standard NMS implementation only remove boxes where the overlapping boxes predict the same class. The NMS used by YOLO is detailed in algorithm 2.

Algorithm 2: Non-maximum suppression.

Data: B = list of bounding boxes
S = list of probabilities P_c for each class for each box
N_t = NMS threshold value
Result: List with overlapping bounding boxes suppressed

```
foreach class do
    while not all bounding boxes processed do
        select bounding box with highest P_c;
        if IoU greater than N_t and P_c is non-zero then
            P_c = 0;
    end
end
```

Bodla et al. [6] proposed Soft-NMS which gradually decays detection scores as the IoU increases. Instead of setting probability to 0, it applies either of the following functions.

$$f(P_c, \text{IoU}) = (1 - \text{IoU}) * P_c \tag{10}$$

$$f(P_c, \text{IoU}) = e^{\frac{-(\text{IoU}*\text{IoU})}{\sigma}} * P_c \tag{11}$$

The function in equation 10 is standard Soft-NMS where probability decays linearly and the function in equation 11 is Gaussian Soft-NMS where probability decays exponentially. This way no bounding are directly given a probability of 0. These can potentially be removed later with a probability threshold, as done when calculating oLRP, described in section 2.7.2, or before drawing bounding boxes as is common practice.

2.5 Multiple Object Tracking Algorithms

Object detection algorithms produce detections without association between frames, if this is needed Multiple Object Tracking (MOT) can be used after detections have been produced. MOT algorithms keep track of objects as they move in a video giving them unique identities. State of the art MOT algorithms also account for temporary occlusions and objects that temporarily leave the view frame. Most often MOT algorithms rely on object detectors to localize objects in a frame before it does tracking. Two types of MOT algorithms exists, online and offline. Offline trackers considers a whole finished video and therefore can "see into the future" while online trackers only consider current and past frames [52].

Two common metrics to report for MOT algorithms are MOTA and MOTP used in the CLEAR MOT challenge. MOTA measures the tracking accuracy. It accomplishes this by measuring the trackers ability to identify objects across frames, the ratio of false positives and the mismatches. Through this it measures object configuration errors. MOTP on the other hand measures the localization error of the tracker for the matched targets and objects. [3]

2.6 Simple Online and Realtime Tracking

Simple Online and Realtime Tracking (SORT) is a tracking-by-detection framework that can track multiple objects across frames after an object detector has localized objects [4]. SORT has been shown to work well on detections generated by YOLO [2].

SORT does association across frames only using the detection data and approximating the displacement of each object between frames linearly. First the velocity of the objects are solved using a Kalman filter. Then for each frame, it moves the boxes and associates new detections with the targets that already exist. This is done by computing an assignment cost matrix by calculating IoU between each detection and all moved targets. The assignment is solved using the Hungarian

algorithm with the IoU matrix. A minimum IoU threshold is also used before assignments. A maximum age threshold is set so targets can live past some number of frames without detection to account for occlusions and the previously computed linear velocity is used when the target cannot be associated with new detections. [4] A result of the use of a Kalman filter is that SORT also changes the bounding box size and location across frames.

SORT is fast and achieves 260 Hz on a modern CPU, therefore it pairs well with a real-time object detector like YOLO, making it possible to do tracking in real-time. [4]

2.7 Performance Measures

Measuring the performance of an object detection algorithm is more difficult than other algorithms such as image classification. This is because the performance consists of multiple factors like localization and classification and not just a single error. The image may also contain multiple objects. For this reason multiple performance measures have been developed, the relevant ones are presented here.

2.7.1 Precision Recall Curves

Precision recall curves are used as a performance measure for object detection tasks as it is not only able to measure an algorithm's ability to correctly classify objects, but also penalize it for wrongly detected and misclassified objects [12]. It is especially useful when assessing models trained with moderate to large class imbalance [10].

To create the precision recall curve it utilizes precision and recall numbers to compute a curve. As seen in figure 7 precision is the proportion of the objects that have been correctly identified. Recall is the proportion of objects that have been identified that are of the correct class. A detection is considered correct if the IoU is over 0.5 and the class is correctly predicted [12].

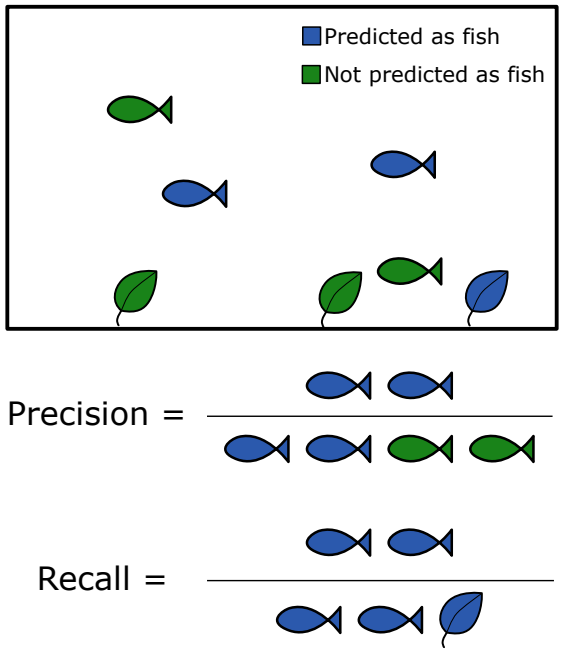


Figure 7: Illustration of an image that has been classified and how precision and recall is calculated for this image.

The precision and recall is plotted against each other to produce a measurement of precision at different recall levels as seen in figure 8. This is done by taking all the predictions, ordering them after probability. Then going through the list and calculating the average precision and average recall so far, for each prediction. The results of this are then plotted.

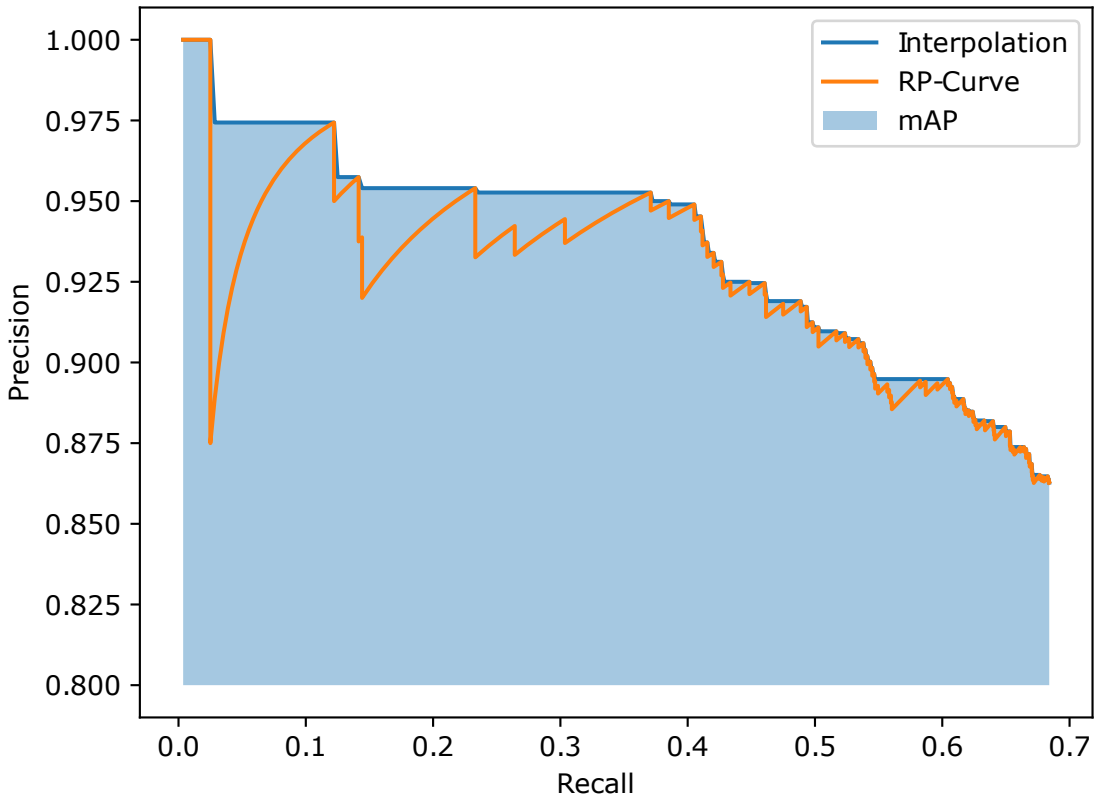


Figure 8: Precision recall curve. Orange is the actual curve, blue is interpolated and the blue area is the calculated mAP.

Several measures based on this curve exist, but they all calculate the area under the curve (AUC) in their own slightly different way. Calculating the area under the curve is an effective method for assessing the quality of the predictions across the different recall levels [10]. Most of these method use interpolation to reduce the impact of "wiggles" [12]. The AUC with interpolation is shown in light blue in figure 8.

This area under the curve metric is called average precision (AP) or mean avarage precision (mAP). The Pascal VOC challange used to calculate the average precision by sampling the curve at 11 different positions [12], and it was a common measure to report. However they now use all data points to calculate AP [12], this is the technique used in figure 8.

2.7.2 Localization Recall Precision

Oksuz et al. [36] notes several problems with the standard performance measure, AP. The most important being:

- Unable to distinguish different PR curves.
- Lack of directly measuring bounding box localization accuracy.

And most relevant for this thesis:

- Uses interpolation between neighboring recall values. [36]

Therefore Oksuz et al. [36] proposed a new performance metric called Localization Recall Precision (LRP). LRP is made up of three parts, a bounding box error part, a false positive (or precision) part, and a false negative (or recall) part. It is therefore able to represent both precision and recall errors, but also bounding box errors. It does this without calculating the area under the PR curve, hereby avoiding the need for interpolation. As opposed to mAP lower value indicates better performance with LRP. LRP is defined in equation 12.

$$\mathrm{LRP}(X, Y_s) = \left(\sum_{i=1}^{N_{\mathrm{TP}}} \frac{1 - \mathrm{IoU}(x_i, y_{x_i})}{1 - \tau} + N_{\mathrm{FP}} + N_{\mathrm{FN}} \right) / (N_{\mathrm{TP}} + N_{\mathrm{FP}} + N_{\mathrm{FN}}) \quad (12)$$

Where τ is the IoU threshold for predicitions being considered correct, and where N_{TP}, N_{FP}, and N_{FN} is the number of true positives, false positives and false negatives respectively.

In addition Oksuz et al. [36] introduce optimal LRP (oLRP) which is the minimum achievable LRP error. oLRP represents the best achievable configuration of bounding box localization and recall for the detector with predictions with IoU greater than $\tau = 0.5$ considered correct.

oLRP is calculated by finding the lowest error for all possible values of the probability threshold for considering a prediction correct, denoted s. oLRP is defined in equation 13.

$$\mathrm{oLRP} = \min_{s} \mathrm{LRP}(X, Y_s) \quad (13)$$

2.8 Fish Species Taxonomy

Classification of fish and other organisms is unique because it follows an established, strict, tree structure. Each element in this tree is called a taxa, and consists of a group of species. A given taxa have a more general category above it, and a more specific category below it. The most specific category is a species and specifies one organism type. These categories are based on the inherent traits of the organism or common features. [56] This means that species that are related in some manner often share a lot of visual features.

3 State of the Art

Object detection and classification in images are heavily researched. Fish detection is a branch of this that recently have experienced great improvements. Techniques for detection of fish in images is divided up in to two different branches, traditional image processing approaches and deep learning approaches. Deep learning approaches perform significantly better at fish detection [58].

3.1 Object Detection Using CNN

There are two big object detection competitions where researchers compete to come up with the best algorithms. These are PASCAL VOC and COCO. Researchers often use datasets from these to compare performance.

The PASCAL VOC 2007 and 2012 challenges are the most notable of the VOC challenges. The VOC datasets has 20 categories ranging from train to bottle. The VOC 2007 challenge has 5 011 images released for training and validation and 4 952 images reserved for testing [12]. Evaluation of performance is done by calculating mAP with 11-point TREC-style sampling [12]. The VOC 2012 challenge has 11 540 images released for training and validation [13]. Evaluation of performance is done by calculating mAP with interpolation using all data points as explained in section 2.7.1 [12].

The COCO challenges had object detection with bounding boxes up until 2017 with the 2018 and 2019 challenges being segmentation only. COCO 2017 has more than 200 000 images with 80 classes available. COCO uses mAP with 101-point interpolation at different IoU threshold, averaging the mAP with IoU ranging from 0.05 to 0.95 with 0.05 steps. More conventional mAP metrics with IoU 0.5 and 0.75 are also provided. [7][8][27]

There exist many techniques for deep vision systems, a literature review from 2018 found that the the following distribution of technologies are used.

- Convolutional Neural Networks: 66.07%
- Reccurent Neural Networks: 12.50%
- Deep Boltzmann Machine: 8.94%
- Other: 12.50% [1]

This shows that convolutional neural networks are by far the technique receiving the greatest amount of research. This is also reflected in recent advances in object detection in real-time videos where convolutional neural networks achieve the highest performance [1]. This performance is achieved by CNN for several reasons. CNNs have the ability to learn features with multi-stage structures from data automatically [24]. The deep architectures of CNNs provide significantly increased expressive power. CNNs can optimize for classification and localization at the same time, making it possible to create one model that that is responsible for the entire problem. [61]

There are mainly two approaches to object detection algorithms, region proposal based, and regression/classification based [1]. Region proposal approaches are used by, among others, R-CNN [17], fast R-CNN [16], faster R-CNN [47], R-FCN [9], SPP-net [20], FPN [28], and NAS-FPN [15]. Region proposal methods first generate a large number of proposed bounding boxes and then feed these into the network to classify the objects within them. Various region proposal methods exists but generally produce a very high number (300-2000 proposals) of bounding boxes that needs to be classified, and as a result this approach is significantly slower than regression/classification methods.

All versions of YOLO [45][43][44], SSD [31], DSSD [14], DSOD [49] and others use regression/classification methods for object localization and classification. With this method bounding box generation and classification is performed in one operation, using one pass through a single network that generate both bounding boxes and class probabilities for each of these bounding boxes. A common trait is that they can do detection significantly faster than region proposal networks. As seen in table 1 some of the networks are able to do detection in real-time or close to real-time on videos[44][31][31][49].

Name	Resolution	mAP (%)	Inference time (ms)
SSD	321 x 321	21.6	61 (M40)
DSSD	321 x 321	28.0	85 (M40)
R-FCN	600 x 600	29.9	85 (M40)
RetinaNet-50	500 x 500	32.5	73 (M40)
RetinaNet-101	500 x 500	34.4	90 (M40)
YOLOv3	320 x 320	28.2	22 (Titan X)
YOLOv3	416 x 416	31.0	29 (Titan X)
YOLOv3	608 x 608	33.0	51 (Titan X)
YOLOv4	416 x 416	41.2	10.4 (V100)
YOLOv4	512 x 512	43.0	12.0 (V100)
YOLOv4	608 x 608	43.5	16.1 (V100)
RefineDet R-101	832 x 500	34.4	90 (Titan X)
FPN R-50	640 x 640	37.0	37.5 (P100)
FPN R-101	640 x 640	37.8	51.1 (P100)
FPN R-50	1024 x 1024	40.1	73.0 (P100)
FPN R-101	1024 x 1024	41.1	83.7 (P100)
NAS-FPN R-50	640 x 640	39.9	56.1 (P100)
NAS-FPN R-50	1024 x 1024	44.2	92.1 (P100)

Table 1: Data collected from [29][44][15][5] showing mAP at COCO test-dev dataset for some notable networks with inference time less than 100 ms. Note that mAP reported here is with the COCO avaraged IoU method. Inference time measured on a M40 or Titan X which are comparable in performance, P100 which is significantly faster with about 60% more flops and V100 which has about 32% more flops than P100.

YOLO is considered the state of the art in object detection and classification in

real-time video, and YOLOv3 achieves almost the same accuracy as others while having significantly lower processing time. It achieves inference times from 22 ms to 51 ms depending on the resolution of the input layer, where higher resolution achieves higher accuracy at the price of inference time. This makes it possible to classify object is videos at more than 30 FPS with a modern graphics card. [44][43]

A recent improvement to YOLO, named YOLOv4 [5], sees an increase in both speed and performance. Their method is to divide the object detector architecture into 4 parts where they can swap out each part with solutions from different earlier research within object detection. It was found that using an architecture consisting of CSPDarknet-53 [54], SPP [20] and PAN [30], and YOLOv3 [44] as the detector mechanism yielded the best results. This resulted in an mAP of 43.5% with 62 FPS on a Tesla V100.

3.2 Object Detection of Fish Using CNN

Research on utilizing machine learning for fish images has been primarily focused on the fish4knowledge dataset which contains 27 730 fish [40] and is the largest publicly available dataset for fish. It contains images cropped around the fish with sizes ranging from 30x30 pixels to 250 x 250 pixels [37]. Almost all the images contain a side view of the fish. It was created from low resolution video of 320x240. This video is caught in tropical coral reefs off the coast of Taiwan, and as a result has very clear and calm waters. The dataset contains 23 different species, 97% of which are in the top 15 classes and 44% is in the top class [37]. Research on this has been focused on either fish object detection or fish species classification. The research in table 2 shows the accuracy of different convectional neural networks for classification of fish.

Name	Classification accuracy
Salman et al. [48]	94.00%
Siddiqui et al. [50]	94.30%
Olsvik et al. [37]	99.27%

Table 2: Accuracy of fish species classification in the fish4knowledge dataset.

Object detection and classification on fish has been done on data derived from the fish4knowledge dataset where Xiu Li et al. [58] in 2015 used Fast R-CNN to achieve an mAP of 81.4% at a speed of 311 ms per image, and Fast R-CNN with singular value decomposition to achive an mAP of 78.9% at 273 ms per image. This is further improved in 2017 by Li et al. [26] which achieved a 89.5% mAP at 89 ms per image.

Raza and Hong [41] did object detection of fish using a modified version of YOLOv3 on their dataset with 4 classes of aquatic life with one of them being for fish. By modifying the loss function and adding an extra detection scale to account for finer grained features they achieve an mAP of 91.3%.

Object detection of salmon in Norwegian fish farms has been done by Reithaug [46] where using SSD Inception V2 an mAP of 84.64% is achieved at 266 ms seconds per image. Classification of Nordic fish have been done by Olsvik et al. [37] where they attained an accuracy of 87.74% with a CNN-SENet and 90.20% with ResNet-50 on their dataset containing 867 fish across 4 classes. The classifier from Olsvik et al. [37] is expanded for object detection using YOLOv3 by Knausgård et al. [25] where YOLO is used for localization only, and classification is done with CNN-SENet. The localizer archives an mAP of 87% (without classifciation error) on their data. The classifier achieves a classification accuracy of 83.68% on the dataset of Knausgård et al. [25]. No mAP with both localization and classification is given.

3.3 Multiple Object Tracking Algorithms

MOT algorithms that focus on associating detections from a detector across frames is a active research area. Stability of Video Detection and Tracking (SORT) is a notable tracker that simultaneously achieves state of the art tracking accuracy while being fast and simple. SORT has had multiple improvements, notably Wojke et al. [57] integrated appearance information and used a CNN to discriminate different objects to avoid identity switches. This improves performance of SORT at a cost to speed and requiring a network to discriminate objects.

Person of Interest (POI) is an online tracker that performs better than SORT by creating a more advanced algorithm. POI utilizes motion, shape and appearance for association instead of just IoU for association with the Hungarian algorithm, among other differences. [59]

3.4 Stability of Video Detection and Tracking

Stability of localization and classification in video has received limited research, but Zhang and Wang [60] identified problems in current evaluation techniques of object detection in video. Zhang and Wang [60] presented a metric for measuring detection stability based on error. This metric uses MOT data as ground truth data and then evaluates the three following factors of stability. Fragmentation error is defined as the integrity of detections along a trajectory and measures how many times the detector loses and regains tracking of an object. Center position measures the stability of the center position to find errors in bounding box localization by measuring the variance. Big changes in the center position would indicate jittering in bounding box location from frame to frame. Scale and Ratio error measures the stability in the size and shape of the bounding box by measuring the variance. Big changes in size and shape of bounding boxes would lead to unpleasant results. All these metrics are then combined to produce a stability error. [60] As far as we know no stability metric that does not require ground truth data exists.

3.5 Hierarchical Classification

Normally, CNN operates on just simple categories, but animals are more complex and grouped into categories of categories by biologists, and therefore has a hierarchical structure. Hierarchical classifiers is a subset of classification algorithms, and are commonly used when the domain being modeled already have a hierarchical structure. Hierarchical classification can be divided into two different approaches, global classifiers and local classifiers. The difference between these are that in global classifiers one model considers the whole class hierarchy and in local classifiers there is more than one classifier arranged in some manner across the hierarchy. [51]

ImageNet introduced by Deng et al. [11] contains 14 million images and 1 million images with bounding boxes. The classes are organised into the a graph structure they call WordNet. Using this hierarchy and a tree-max classifier they show how hierarchical classification based on synset nodes can improve performance. This algorithm and other algorithms noted in [33], [34], [62] and [35] are local classifiers and utilize non-deep learning approaches and show that hierarchical classification can increase performance.

Redmon and Farhadi [43] utilises global hierarchical classification with Darknet-19 on the ImageNet dataset by transforming the WordNet into a tree structure called WordTree. It then predicts the probabilities as explained in section 2.2. This classifier achieves 71.9% accuracy on a subset of ImageNet containing 1 000 leaf classes, or 1 369 with the classes needed in the hierarchy. This is later extended to be trained on the top 9 000 classes from ImageNet for classification and the COCO dataset for detection to create an object detection algorithm that can detect over 9 000 classes. Interestingly Redmon and Farhadi [43] notes that this algorithm learns animal species better than other classes from the hierarchy.

Local hierarchical classification of fish is used by Huang et al. [22] with the fish4knowledge dataset where they use a Balance-Guaranteed Optimized Tree (BGOT) to construct a hierarchy of fish species based on the 66 features extracted from the image. This hierarchy is then used with a SVM at each level of the hierarchy. With this algorithm they achieve a classification average precision of 91.7% on a dataset containing 3 179 fish from 10 classes. Later in [23] they improve the algorithm and achieve an average precision of 65% on a dataset containing 25 150 fish from 15 classes.

4 Approach

This chapter will outline our approach to solving the problem of real time object deteciton of fish. It goes over the steps performed to create the object detection algorithm YOLO Fish.

4.1 Execution Process

To go from problem statement to final solution and conclusion, a method based on design science research is used, a common method in information technology research. The design science research method follows six steps where an artifact that can solve the problem is developed and then evaluated in terms of the observed results from using the artifact [38][21]. The phases in the design science research process are:

- Problem identification and motivation.
- Definition of objectives for solution.
- Design and development.
- Demonstration.
- Evaluation.
- Communication. [38]

This has been integrated in the larger process of the execution process, resulting in the process seen in figure 9.

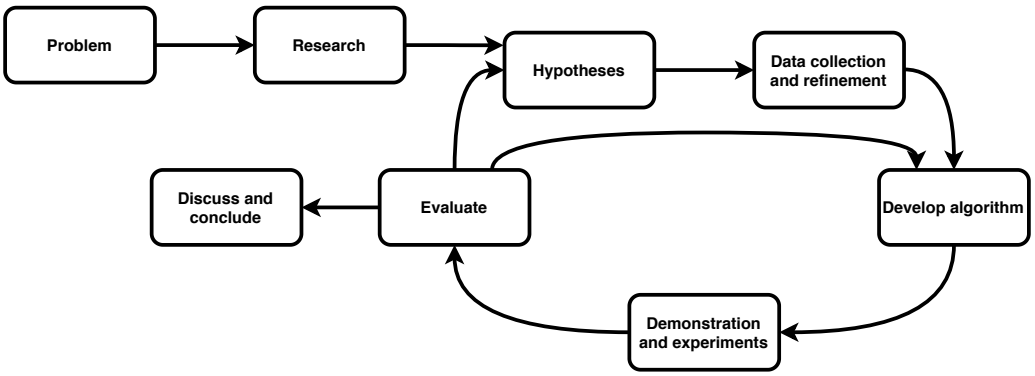


Figure 9: Extended version of the design science research method used for the project.

Initial research is done to figure out which algorithms best fit the problem. Theories are formulated that will aid in solving the problem. Data is collected and annotated based on the current hypotheses. The selected or modified algorithm is used as the artifact that the design science research method requires. This algorithm is then tested in the experiments to see whether it provides a verifiable contribution.

These experiments has the goal of isolating the problem to be able to rigorously evaluate the contribution. Atomizing the problem makes it possible to isolate the complexity [38]. The experiments constitute the demonstration phase. Evaluation of these experiments use both quantitative and qualitative results to assess whether the solution aids in meeting the objectives, and whether it verifies the hypotheses. Quantitative results are effective for evaluating if a new solution is better than the current solutions [38]. Qualitative methods are used to support solving problems not previously addressed [38]. The results of the experiments influence the hypotheses in an iterative manner that creates new or modified hypotheses or it may influence the algorithm to improve the solution.

Following this strategy the experiments in this thesis have either qualitative or quantitative objectives for the solution. For this, objectives are rationally inferred from the problem. These objectives are then compared with the observed results and evaluated to see how well they support the problem. [38]

4.2 Data Generation Strategy

Object detection in images and video requires data in the form of images with accompanying bounding boxes with assigned classes. Compared to image classification tasks, this requires a lot of manual labour as this kind of data is not easy to come by. Additionally, to be able to test hierarchical classification, the dataset needs be created with hierarchical classes.

The dataset needs to be created with hierarchical classes in mind because the hierarchy affects what labels are applied. This is because the algorithm needs to learn what features can distinguish certain species, and what features is common among them. Therefore the class needs to reflect the visible features of the fish. By using hierarchical classification with a dataset specifically created for this, the network can mimic the mental model humans have of fish and fish species. This makes it possible for it to apply classes based on the features that are actually visible and have a result that reflects how good the view of the fish is.

Redmon and Farhadi [43] uses hierarchical classification to combine two incompatible datasets. Just applying hierarchical classification without a dataset created for it works to combine the datasets, but does not help when the goal is to use this to account for the lack of visible features.

Because of the previously mentioned reasons, a custom dataset and data generation strategy is required. The images are collected from an underwater video stream and imported into an image labeling tool. To ensure the dataset is consistent we establish a set of strategies on how to annotate the data that is rooted in the theory behind YOLO. These are specifically based on how the loss is handled. We define the following approach for annotation.

1. Annotate all fish that are visible to humans, without the context of neighboring frames.
2. Apply the most specific species label that can confidently be discerned.

Since the loss function penalizes detections where it detects fish where there is none, a goal is to make sure all the fish that can possibly be detected are annotated with a bounding box.

On the other hand, with the use of a tree structure for classes, the annotation of classes have used the opposite tactic. Here only the most specific taxa that can very confidently be applied to the fish is used. This strategy is rooted in the design of YOLO where loss is only applied to the ground truth class and the above classes in the WordTree [43]. This does not penalize the model for predicting a more specific class than what is annotated. An example of this can be seen in figure 10, where *Pollachius pollachius* has been predicted, but the ground truth specifies Gadidae, so only the classes Gadidae and Fish have their loss backpropagated. This has the effect that it increases the confidence in the network's ability to predict the correct classes since the confidence of the class in the dataset is very high. It also makes it significantly easier to create the dataset since not all fish in an image needs to have a species specified.

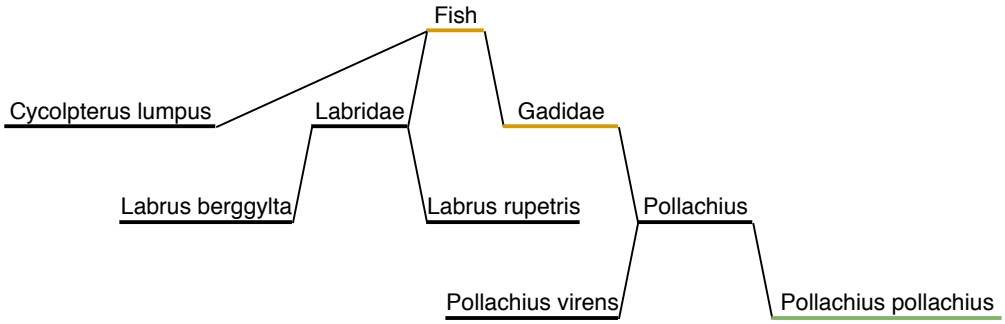


Figure 10: If the network predicts *Pollachius pollachius* (green), but the ground truth only specifies Gadidae, only the orange classes Gadidae and Fish, have their loss backpropagated.

4.3 Architecture Improvements

YOLO cannot be used out of the box because it is not tailored for the problem, and therefore has received various changes to become YOLO Fish. The networks are mostly used in their original architecture as explained in section 2.2.1, but the small changes that are made are explained here. The configuration files that detail the network and the modified version of YOLO is included in the appendix.

YOLO9000 is designed to incorporate WordTree during training to combine datasets [43], therefore the network is used as is except changing the number of classes from 9418 to 9.

YOLOv3 has several improvements as compared to both YOLO9000 and YOLOv2, these improvements could be utilized to increase performance. However YOLOv3 cannot be directly used with hierarchical data as the detection layers used are new and does not support hierarchical classes. Instead YOLOv3 is a multiclass detector.

To be able to utilize the improvements from YOLOv3 with the benefits of hierarchical classification, modifications to turn it into a single class detector that uses soft max across co-hyponym, and calculates absolute probability as explained in section 2.3 is performed. This is accomplished by using region layers instead of the YOLO layers, and using the WordTree.

By default YOLO is limited to classify at most 30 objects in each image, this has been upped to 90 to be able to detect the number of fish often present in the video.

The NMS algorithm only removes overlapping bounding boxes of the same class. When using hierarchical classification an object can be of several classes across the levels in the hierarchy at the same time. This causes a problem when used together with class dependant NMS as it does not remove objects where the classes are at different levels in the hierarchy. To solve this problem NMS has been modified to remove all overlapping boxes regardless of class.

Because NMS now removes all overlapping bounding boxes, too many bounding boxes may sometimes be removed. Therefore Soft-NMS is implemented to test whether it improves performance, and used in experiment 6.5.

4.4 Training

Because the problem is different than what YOLO was created to do it needs to be trained on new data. All the networks presented in this thesis are trained in the same way with the same parameters and settings, with the exception of the varying use of the WordTree and network resolution. The configuration files for the networks which detail all training configurations are included in appendix B.

The fish in the dataset very often share the same aspect ratios and sizes because the camera was mostly in the same position and fish tends to swim in the same ways. Therefore computing good anchor boxes significantly helps the localization. Nine anchor boxes that fit the training set are computed with K-means as explained in section 2.2, and this resulted in the boxes seen in figure 11.

Different clusters and associated bounding boxes can clearly be seen in figure 11 with many of the bounding boxes being small and square, small and wide, or small and tall. This is because fish often swim towards, parallel to or above the camera respectively. Two larger bounding boxes are also seen, these represent the fish that swim close to the camera, but there are significantly fewer of these. Nine anchor boxes is selected because more anchor boxes gave diminishing returns in IoU overlap and increased inference time. Fewer bounding boxes however reduced IoU overlap significantly, as seen in table 3. In the case of YOLOv3 each detection layer was responsible for three of the anchor boxes.

# anchor boxes	6	9	12
Avarage IoU	0.63	0.69	0.71

Table 3: How increasing the number of anchor boxes yields diminishing returns after nine. Anchor boxes are only given in a multiple of three to be appropriate for YOLOv3's three detection layers.

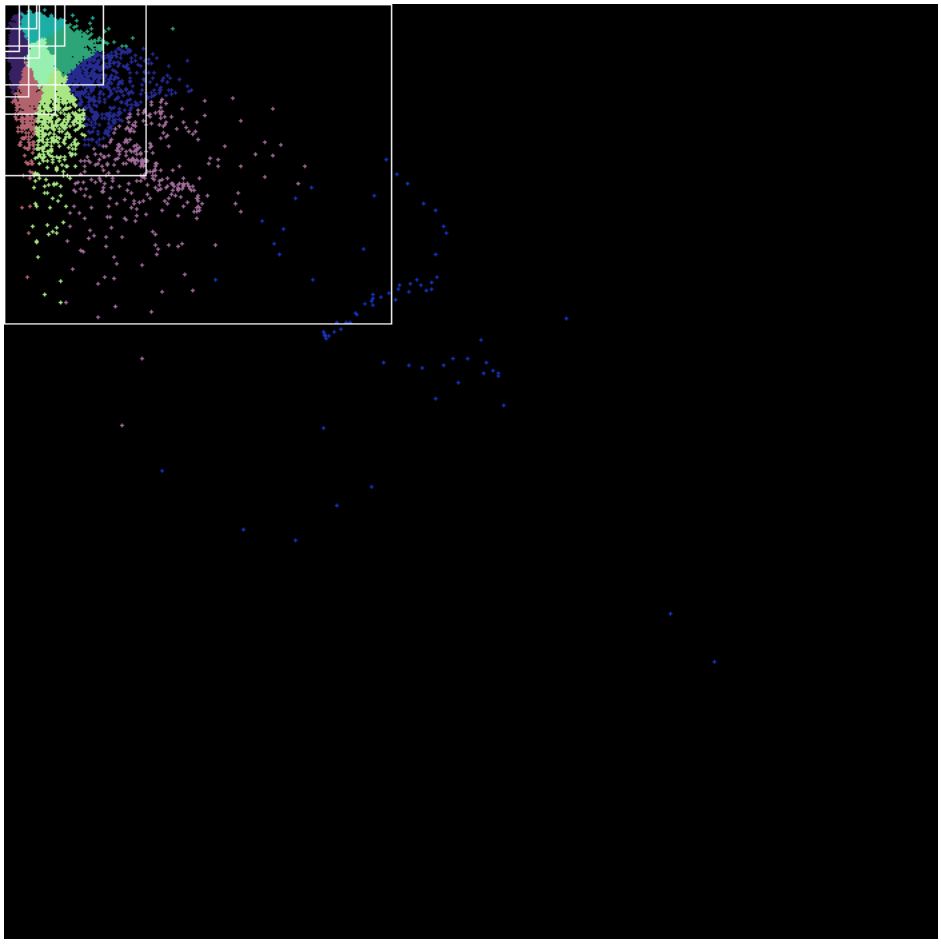


Figure 11: Visual representation of the clusters that was found with K-means. X- and Y-axis represents the width and height, the colors represents the different clusters, and the white boxes are the anchor boxes.

In addition to new anchor boxes some other training parameters are important to note. Letter boxing is used to keep image aspect ratio. If nothing else is noted a network resolution of 608x608 is used. While there are only seven classes of fish in the dataset as noted in section 5, intermediate classes are added to allow for the tree structure. The tree structure with all the classes is in figure 12. To keep things as similar as possible everything is trained with the nine classes, even if WordTree is not used. In addition these image augmentations are randomly applied during training:

- Image resizing
- Hue change
- Saturation change
- Exposure change
- Image cropping (jitter)

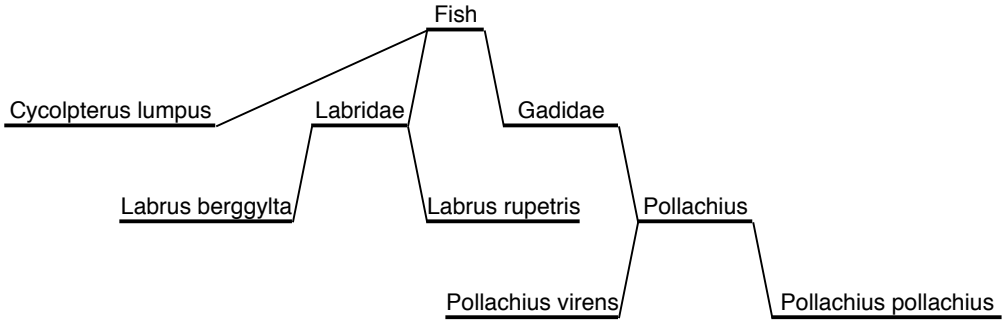
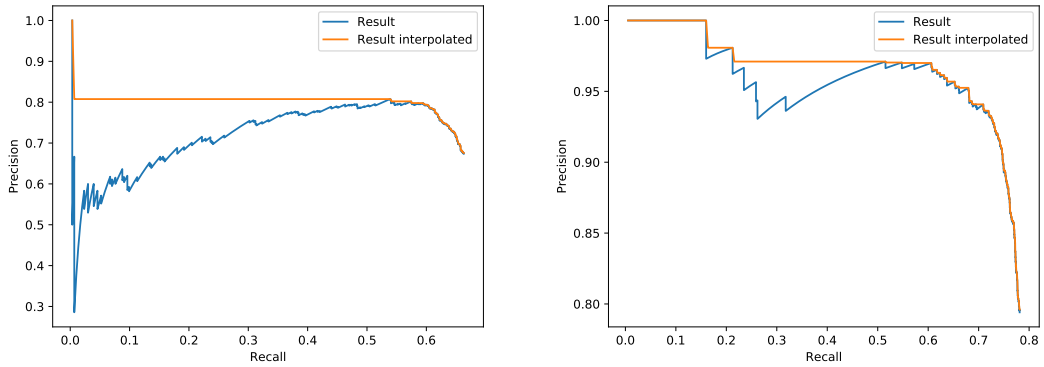


Figure 12: The WordTree used for training and validation.

4.5 Evaluation Methods

Performance measures are an effective way to quantitatively gauge how well the solution works, and selecting appropriate evaluation measures and methods that capture the problem is important. The standard measure for evaluating performance of object detectors is mAP [36], because of its use in prominent competitions like Pascal VOC [13].

However the mAP metric does not work well when evaluating predictions made while using the WordTree in the case when NMS includes too many boxes. Figure 13a is an example of the PR curve for predictions made by YOLO when using the WordTree. This graph is constructed by sorting by probability and then interpolating with the method explained in section 2.7.2. This construction method leads to large increases in the precision as the recall approaches 1 before it quickly falls off. The interpolation algorithm will then result in highly inflated area under the curve, as seen in the difference in the orange and blue line in figure 13a.



(a) PR curve when NMS is not removing enough bounding boxes.

(b) PR curve when NMS removes all bounding boxes that has a IoU greater than 0.5.

Figure 13: PR curve and mAP interpolation for predictions made with YOLO-WT.

This happens since NMS does not remove predictions made for the same fish when the classes are different as seen in figure 15a. This results in many bounding boxes for each fish with a high probability and leads to many errors with a low recall in figure 13a, and therefore misleading mAP.

For evaluation of the experiments multiple different metrics are therefore used. oLRP is used as a metric for assessing the performance of the network. This is mainly because it does not encounter the same problems as mAP, as it is able to have different results for different PR curves, as apposed to mAP, and various other advantages explained in section 2.7.2 [36]. For experiments where mAP provides representable results mAP is also used as it is the most widely reported measure and performs well for imbalanced datasets.

A custom implementation of the metrics had to be created for them to account for the hierarchical nature of the dataset. The script that calculates the different metrics is included in appendix A. oLRP is implemented as defined by Oksuz et al. [36] and mAP is implemented as used in Pascal VOC 2010 and onwards [13] with a change to what is considered correct. This change is that a prediction is also considered correct if the assigned class is a descendant node of the ground truth node in the hierarchy. This is done because, as far as we can know, the classification is correct. This is followed even if the network doesn't use hierarchical classification for the sake of fairness.

5 Dataset

A new dataset was needed for this project because no publicly available datasets fits the needs for the project. No large object detection dataset with multiple fish in each image was found. Additionally, to be able to show the effectiveness of hierarchical species classification, a dataset that is created with a hierarchical labeling strategy is needed. The labeling strategy used for this project is explained in section 4.2.

The dataset is a collection of underwater images with fish, and annotation files that specify the bounding box of each fish in the image, and a class. The images have a resolution of 1920x1080 and are taken from a camera that is located in Lindesnes, Norway. The pictures were captured in a period between February and March 2020. The dataset consists of 1879 images and corresponding annotation files. There are 7721 labeled fish in total and they are distributed over 7 classes as shown in table 4. The test set contains 10% of the images which equates to 188 images and the training set contain 1691 images. Each image has a varying number of fish in each image.

Species	Count
Fish	5810
Gadidae	791
Pollachius virens	537
Ctenolabrus rupestris	212
Pollachius pollachius	172
Cyclopterus lumpus	147
Labrus bergylta	52
Labridae	0
Pollachius	0
Total	7721

Table 4: Distribution of classes in the dataset.

There is one annotation file for each image where the annotation file and the image have the same name. Each annotation file contains the labels separated by newline. The labels are on the format specified in listing 2. The x, y, width and height are relative to the size of the image. A link to the dataset can be found in appendix A.

```
<object-class> <x> <y> <width> <height>
```

Listing 2: The annotation file format.

The images contain fish in motion and from many angles. This makes the fish in the images have very varied shapes, sizes, and visible features, as seen in figure 14. The fish can mostly be seen swimming above, towards, away, or parallel to the camera because of its location on the bottom of the seabed. The pictures were caught day, dusk, night, and dawn, and in both clear and turbid waters and therefore provides a wide variety of conditions. There is also a significant portion of the frame where seaweed can be seen and many of the pictures contain fish in the seaweed. This is a quite difficult dataset in contrast to for example fish4knowledge. This dataset more represents the real world's noisy and imperfect conditions that a real application would be exposed to.

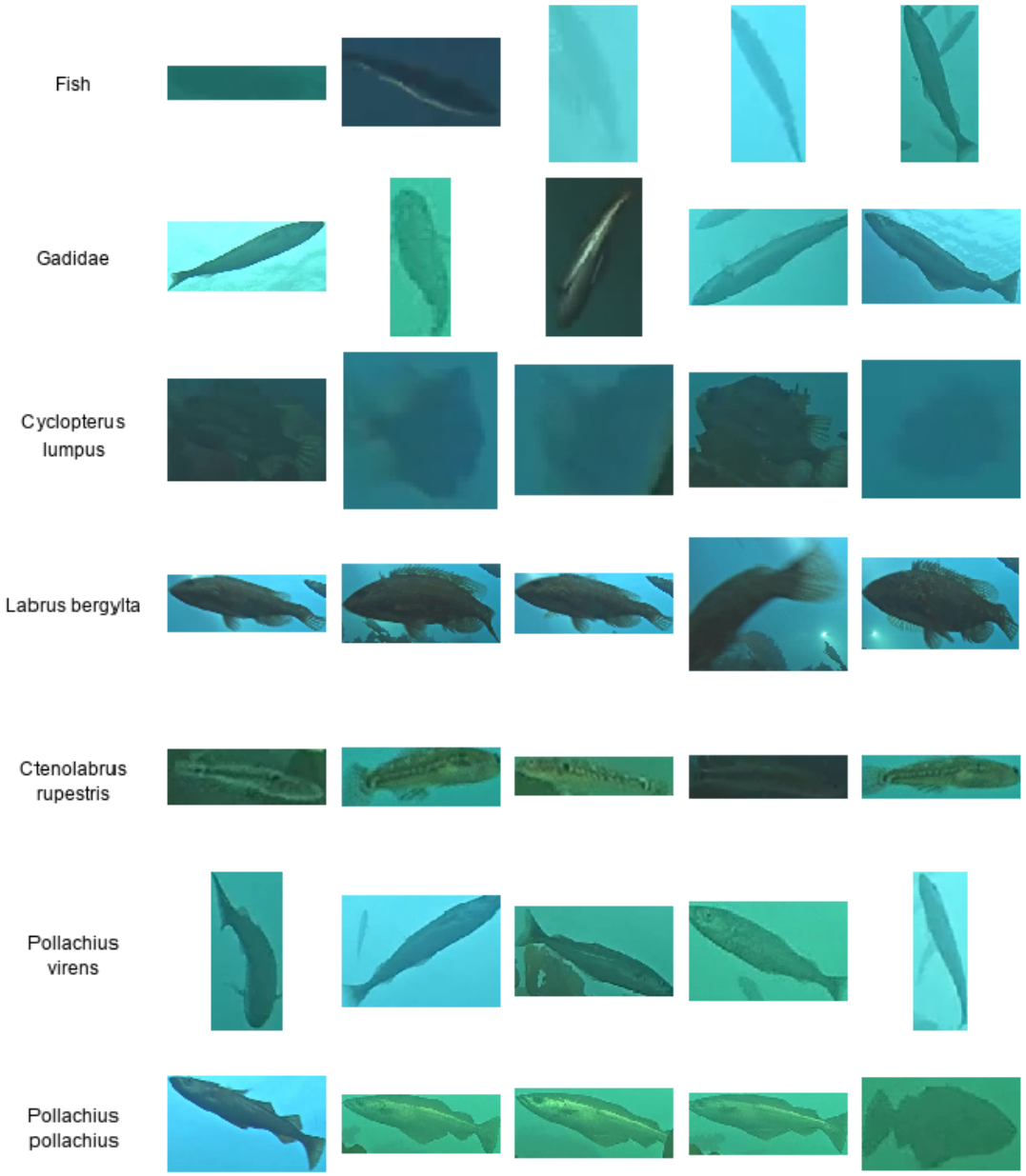


Figure 14: Examples of fish in the dataset from the different classes.

Some weaknesses with the dataset is that it is possible that some fish have been labeled wrongly, as we don't have the proper training in the identification of fish species. Another problem is that during this period of time, there was an abundance of *Pollachius pollachius* and *Pollachius virens*, which is of the *Pollachius* genus and Gadidae family. This means that we often had to use over-sampling techniques for other species. Thus these other species does not have labels in as many varied conditions. Thankfully, using hierarchical classification, it should still pick up on useful information.

6 Experiments

This chapter contains experiments based on the hypotheses and iteratively improves the performance by potentially confirming a hypothesis and using the previous improvement as a basis for the next experiment. The outline of the chapter is as follows:

- 6.1 tests H1 whether biological taxonomy can be used with YOLO's WordTree.
- 6.2 tests H2 to see if hierarchical classification improves performance.
- 6.3 tests H3 whether classless NMS can solve the bounding box problem created by hierarchical classification.
- 6.4 tests H4 to increase performance by utilizing the newer architecture of YOLOv3.
- 6.5 tests H5 and compares classless NMS with Soft-NMS.
- 6.6 tests H6 to evaluate the speed of the network.
- 6.7 tests H7 to evaluate the stability of detections across frames in video.

Each of these experiments starts with a explanation of the context and the problem that the algorithm aims to solve. Further the experiment method is presented followed by the results. Then each experiment discusses whether the results proves its hypothesis.

6.1 YOLO's WordTree on the Taxonomic Fish Hierarchy

When creating datasets that can be used for object detection and species classification of fish it is difficult to correctly assign classes to all the fish in an image. This is because an image is likely to contain clearly visible fish in the foreground and less visible fish in the background. Some datasets uses images that are cropped to fit a specific fish of interest, but this cannot be used to effectively train an object detection algorithm. To solve this, fish can be labeled using the biological taxonomy hierarchy, making it possible to always apply labels confidently.

This experiment tests whether it is possible to train YOLO, using YOLO's WordTree in combination with data labeled with taxas from the biological taxonomy hierarchy. And whether YOLO is able to detect fish and correctly assign classes when it is trained on a dataset that contains fish labeled using this hierarchy.

To verify this the YOLO9000 algorithm is trained on the dataset that contains hierarchically labeled fish, and the resulting model tested. Table 5 shows the model's performance on validation data. It displays number of fish labeled more and less specific than that of the ground truth. This makes it possible to see that the model is able to correctly assign the most specific class it can, over a probability threshold.

Species	Fraction correct	# more specific	# less specific
Fish	501/653	500	0
Gadidae	76/81	76	0
Cyclopterus lumpus	12/14	0	0
Labridae	0/0	0	0
Pollachius	0/0	0	0
Labrus bergylta	6/6	0	0
Ctenolabrus rupestris	23/27	0	0
Pollachius virens	65/69	0	65
Pollachius pollachius	14/14	0	14

Table 5: YOLO9000 with hierarchically predicted classes. Column two shows proportion of detected fish that was classified correctly for that label. Column three shows the number of fish classified correctly and that have been applied a more specific class in the hierarchy than the ground truth data. Column four shows number of fish classified with a less specific class, but a class still in its branch of the tree.

Table 5 shows that all Gadidae that were detected was assigned a more specific class than the ground truth data. Consulting figure 12 shows that Gadidae has multiple sub taxas which it shares many features with. This means the the network is able to learn feature information from other training examples and transfer this into being correct for the Gadidae class, and then apply that class.

The same goes for *Pollachius virens* and *Pollachius pollachius*, where the network has applied less specific classes compared to ground truth data, but still a parent class. Comparing those classes in figure 12 it becomes apparent that there are very few distinguishing features. The network has then not been confident enough to distinguish between the classes and then applied a less specific class, for instance *Pollachius*.

This proves that YOLO's classifier has been trained to label data according to the WordTree, and therefore hypothesis H1 is confirmed. It is able to identify the features of specific fish and generalize this information.

6.2 YOLO's WordTree for Increased Performance

Since using YOLO's WordTree is able to generalize classification to data with more specific labels, it is valuable to see what this adds to the predictions. To do this it is effective to compare it with a similar algorithm that does not use the WordTree to isolate the effect of the WordTree.

YOLO's WordTree uses the Region layer, a layer that, if not WordTree is used, assumes that classes are mutual exclusive [43]. This assumption does not hold for our dataset so comparing YOLO9000 with and without WordTree has some drawbacks. But since this is the most direct comparison possible, and since everything else is equal it is an effective way to isolate the improvements made by

the WordTree. Both networks are trained in the same way as detailed in section 4.4.

Table 6 shows oLRP, precision and recall of the algorithm at the probability threshold s, that provides the best results. A prediction is considered correct if the algorithm predicts the same class as the ground truth data, or a class that is more specific than the ground truth data, and the prediction has an IoU greater than 0.5. The networks are trained in the same way as explained in section 4.4 and tested on the test set. mAP is not used because it does not provide representative results as explained in section 4.5.

Measure	YOLO9000	YOLO9000-WT
oLRP ↓	0.830	0.789
Precision ↑	0.693	0.731
Recall ↑	0.488	0.556
s	0.460	0.520

Table 6: Performance of YOLO9000 with and without WordTree (WT). The best probability threshold for the prediction error oLRP, precision and recall is denoted by s. ↓ represents a metric where a lower value is better.

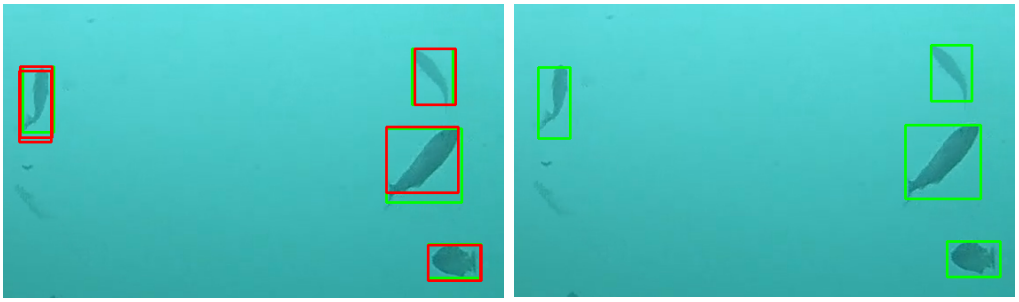
Table 6 shows an oLRP improvement of 0.041 when using WordTree. The localization of objects can be presumed to be equally well predicted, since the networks are the same. This shows that the classification is performing better as a result of the WordTree, and thereby confirms hypothesis H2.

This happens for a couple of reasons. Firstly the network is able to learn more from the data since the error is backpropagated for all classes above the predicted class in the WordTree, as explained in section 2.2.4. This results in a better trained model. Secondly there is more context at detection time. The assigned class is the class furthest down in the WordTree that has a threshold above the hierarchical threshold value specified. This means that if a class is going to be applied, both the applied class and its ancestors needs to have high probabilities. Thus increasing the confidence and reducing faulty classifications.

6.3 Modifying NMS to Account for Hierarchical Predictions

YOLO9000 produces a very large number of bounding boxes for each object. NMS is used to remove all probabilities except the one with the highest value for each bounding box. Then NMS removes the boxes that are not appropriate. This removes boxes that is the same class and has an IoU greater than 0.5, as explained in section 2.4. However, when using WordTree, softmax is applied across siblings, so it is likely for each fish to have high probabilities for multiple classes. The result of this is that NMS does not remove other bounding boxes that predict

the same fish, but at a different level in the WordTree. Figure 15a displays this phenomenon.



(a) Multiple bounding boxes per fish. (b) One box per fish.

Figure 15: Predictions from YOLO before (a) and after (b) modification of NMS. Green bounding boxes indicate correct predictions, red indicate wrong.

This problem can be solved by modifying NMS to not be class dependant. This can be done by simply removing *all* bounding boxes that have an IoU over 0.5, instead of doing it on a per class basis. Figure 15b shows this.

This experiment verifies that the modified NMS increases performance. It does this by comparing the performance of the original NMS and the modified NMS with everything else being equal.

Measure	YOLO9000-WT	YOLO9000-WT classless NMS
oLRP ↓	0.789	0.731
mAP ↑	N/A	0.728
Precision ↑	0.731	0.877
Recall ↑	0.556	0.627
s	0.520	0.550

Table 7: YOLO9000-WT performance improvement from modifying NMS.

Table 7 displays an oLRP improvement of 0.058, stemming from the removed erroneous bounding boxes, thus confirming H3. In addition it also becomes possible to accurately measure the performance with the mAP metric. Previously the precision recall curve did not produce representable results when interpolation is used. This happens because there are many erroneous predictions with high probability that breaks the interpolation used in mAP as explained in section 4.5.

6.4 Modifying YOLOv3 to Use WordTree

YOLO's WordTree algorithm is originally only implemented in the YOLO9000 architecture. However the state of the art is YOLOv3, with a significantly larger

and more complex network with 139.611 GFLOPS compared to 40.483 GFLOPS with our configurations. YOLOv3 has been shown to perform significantly better [44]. This experiment tests whether it is possible to increase performance by creating an architecture that combines WordTree with YOLOv3.

The network is changed to use softmax across sibling classes and use the loss function from YOLOv2, as explained in section 2.2.4. This means that while YOLOv3 is a multi class detector, the modified YOLOv3 no longer is multi class. This is because of changing the prediction layers, seen in purple in figure 3, from being YOLO layers to being region layers.

Measure	YOLO9000-WT	YOLOv3	YOLOv3-WT
oLRP ↓	0.731	0.655	0.556
mAP ↑	0.726	0.793	**0.899**
Precision ↑	0.887	0.800	0.936
Recall ↑	0.627	0.770	0.807
s	0.550	0.380	0.710

Table 8: Performance of YOLO9000-WT and YOLOv3 with and without WordTree.

Table 8 shows that YOLOv3 without WordTree performs better than YOLO9000 with WordTree with an improvement in oLRP of 0.076. The addition of WordTree in YOLOv3 leads to the significant improvement of 0.149 to oLRP and 17.3% to mAP. Figure 16 shows that YOLOv3-WT is able to detect close to all objects as well as producing very few false positives, even at low probabilities. This confirms hypothesis H4 since creating a new architecture that incorporates WordTree with YOLOv3 significantly improves performance.

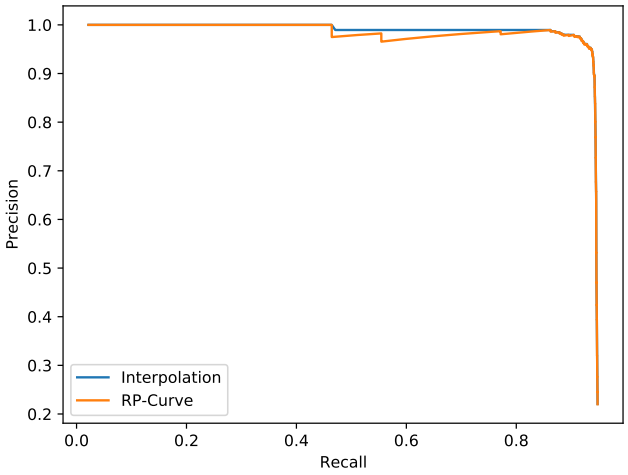


Figure 16: Precision recall curve for YOLOv3-WT that produces mAP of 0.899.

6.4.1 Further Discussion

Now that the architecture of YOLO Fish is mostly in place, we would like to discuss the results of the algorithm as a fish detector so far. This section will present some new results and discuss them in the context of hierarchical classification.

Table 9 shows that the algorithm is not very good at localization as the mAP quickly falls off as the IoU threshold for considering a detection correct is increased. Figure 17 shows that it is very good at classification. It can therefore be said to be better at localization than classification. This is in line with other research and is one of the drawbacks of using detectors that use one networks for both localization and classification, and especially YOLO as the network has large bounding box localization restrictions due to its design [43][61].

Network	$\text{mAP}_{\text{IoU=0.3}}$	$\text{mAP}_{\text{IoU=0.5}}$	$\text{mAP}_{\text{IoU=0.7}}$
YOLOv3-WT	0.925	0.899	0.664

Table 9: Performance of YOLOv3 with WordTree at different IoU thresholds.

Figure 17 shows that the algorithm manages to correctly classify every fish except one of the fish that it detects. The mistake it makes is the misclassification of *Pollachius pollachius* as *Pollachius virens*, two very similar species. It is also often able to classify fish more specifically than the specified label for Fish and Gadidae, this means that the classifier possibly is better than humans at classification. However with the dataset that is used there is no way to say whether the more specific labels are correct or not, so no definitive claim can be made.

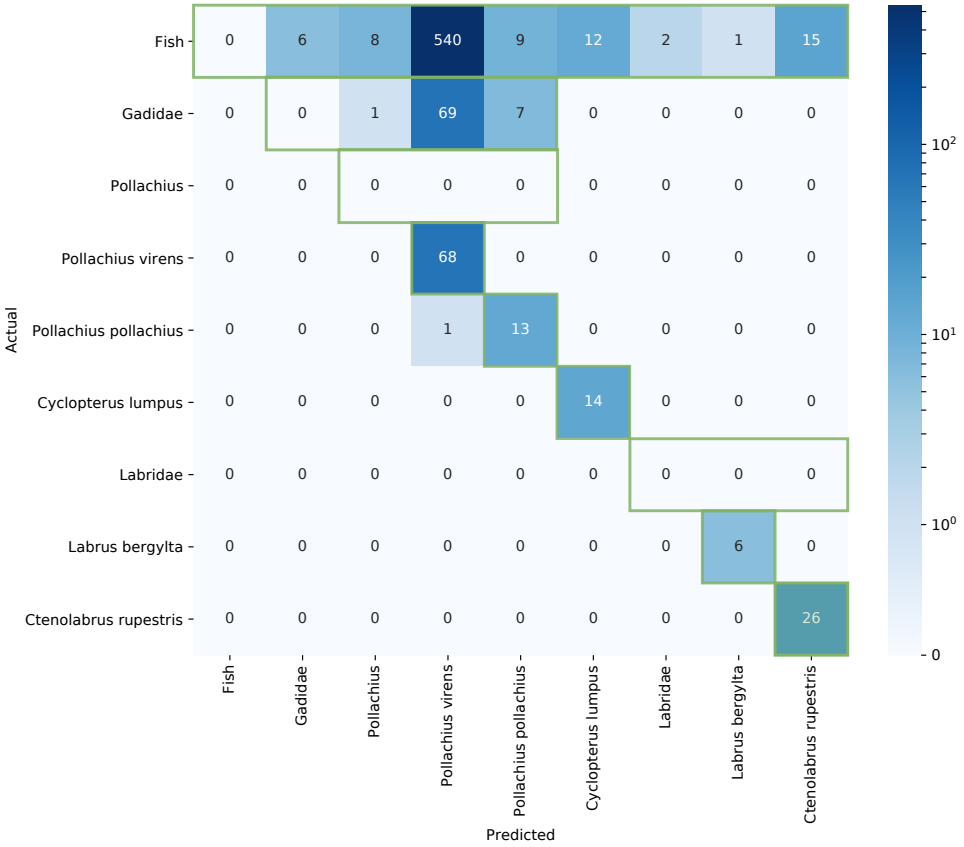


Figure 17: Confusion matrix for YOLOv3-WT at IoU 0.5. All predictions that land inside the green boxes for each class is considered correct. If the prediction is inside the green box and to the right of the diagonal the prediction is more specific than the label in the WordTree hierarchy.

Using WordTree adds an extra hyperparameter that needs to be set according to the use case of the model, a hierarchical probability threshold. This hierarchical probability threshold specifies how certain a classification must be for a specific class prediction before using the parent class. This is needed because the network will predict high probabilities for all classes in the branch of the hierarchy that the object is likely to belong to, and a cutoff value needs to be set so it can determine how deep in the tree the predictions will go. This hyperparameter is only used when predicting and not during training.

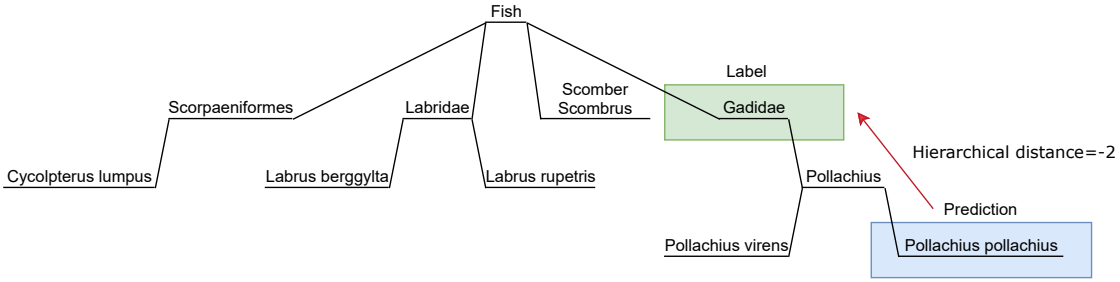


Figure 18: Calculation of hierarchical distance. A positive number means how many levels down in the tree the prediction is, compared to the ground truth. Likewise a negative number means the number of levels up in the tree. Zero means the prediction is the same as the ground truth.

Table 10 shows the average hierarchical distance from label to prediction, the hierarchical distance is calculated with the technique in figure 18. Hierarchical distance makes it possible to see how specific the network predicts species.

Furthermore, table 10 shows that the hierarchical distance is quite high when the probability threshold is 0.2 or 0.5, and becomes lower at hierarchical threshold 0.99 and 0.999. At hierarchical threshold 1 all detection are classified as fish. This illustrates that increasing the probability threshold makes it possible to be very confident in the predicted species, at the cost of how specific the prediction is. For most hierarchical probability thresholds the network is able to classify species more confidently and specifically than a human, as evident by the positive average hierarchical distance for all species. However, this cannot be verified as correct.

	Hierarchical Probability Threshold				
Species	1	0.999	0.99	0.5	0.2
Fish	0	2.20	3.36	3.89	3.92
Gadidae	-1	1.75	2.58	2.99	3.00
Cyclopterus lumpus	-1	-0.14	0.00	0.00	0.00
Labrus bergylta	-2	-0.66	-0.67	0.00	0.00
Ctenolabrus rupestris	-2	-0.42	-0.13	0.00	0.00
Pollachius virens	-3	-0.28	-0.06	0.00	0.00
Pollachius pollachius	-3	-0.08	-0.08	0.00	0.00
All	-0.53	1.74	2.70	3.15	3.18

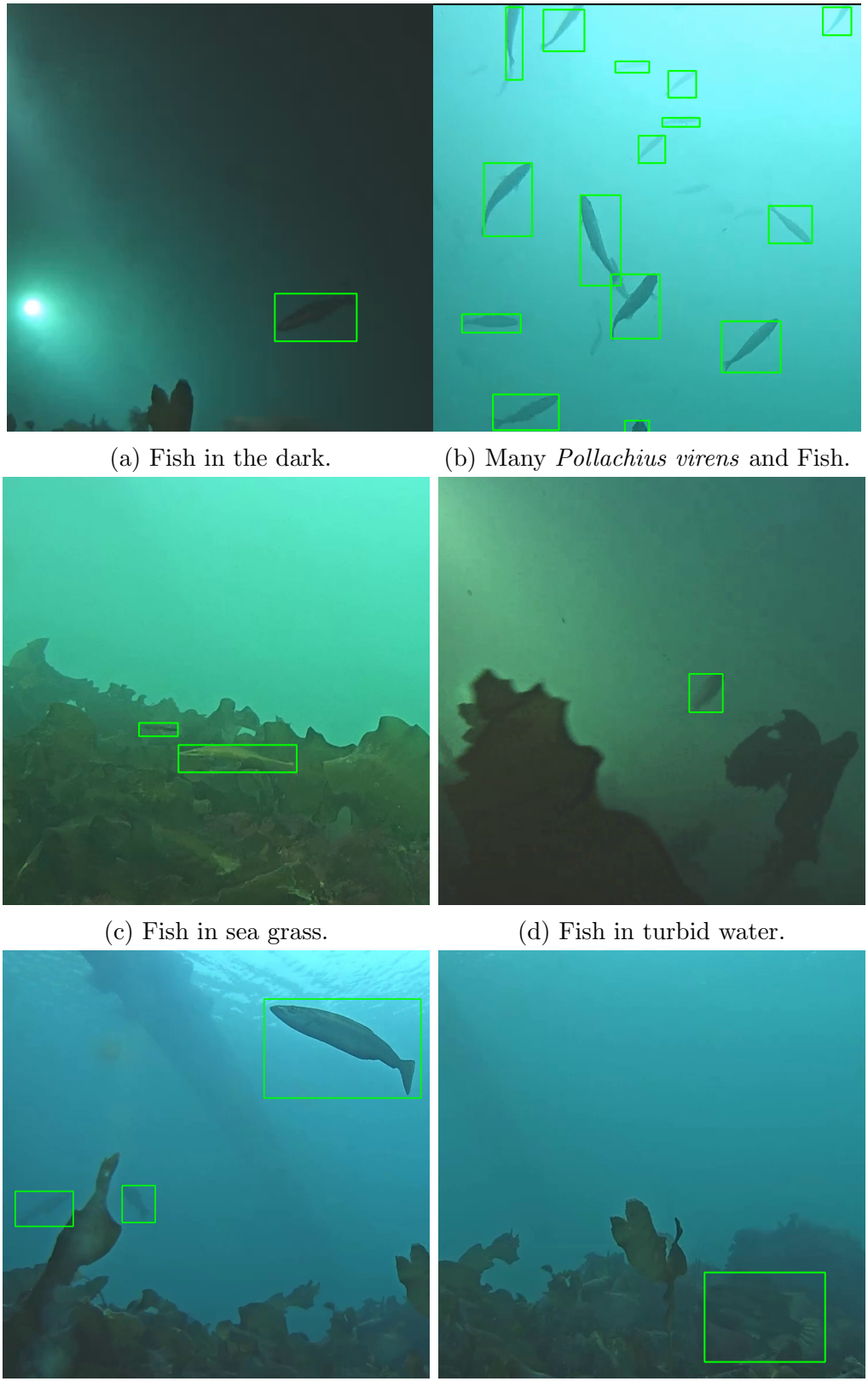
Table 10: The average hierarchical distance from label to detection for different hierarchical threshold levels for YOLOv3-WT. The table only contains correct predictions.

The ability for the end user to set a hierarchical probability threshold makes the algorithm more versatile. For instance, if it is used to collect data for determining fishing quotas there might be a need to have a very high confidence in the species

that are identified. Then a high hierarchical probability threshold might be used. Another use case might be to use it to show the public what species are in a particular area right now. The correctness in this instance might not be as important and a lower hierarchical probability threshold could be used.

So far the predictions have been discussed in a quantitative manner, but discussing them in a qualitative manner compliments this and gives insight. Examples of good and bad detections are in figure 19 and 20 respectively. Figure 19 displays how the algorithm can perform well in varying lighting conditions and water clarity. It is however prone to make mistakes if there is a lot of debris in the water as in figure 20d which was captured in a storm. It excels at detecting fish in the sea grass as in figure 19c and 19f where humans struggle. Sometimes it mistakes what clearly is seaweed for fish as in figure 20d.

The shape of the fish is heavily relied upon for classification, and as can be seen in figure 20a where a Gadidae is misclassified as a *Cyclopterus lumpus* because of its round shape when swimming away from the camera in turbid water. However, when there is clear water the algorithm performs well, and is able to detect fish even with very low contrast as seen in figure 19e.



(a) Fish in the dark.

(b) Many *Pollachius virens* and Fish.

(c) Fish in sea grass.

(d) Fish in turbid water.

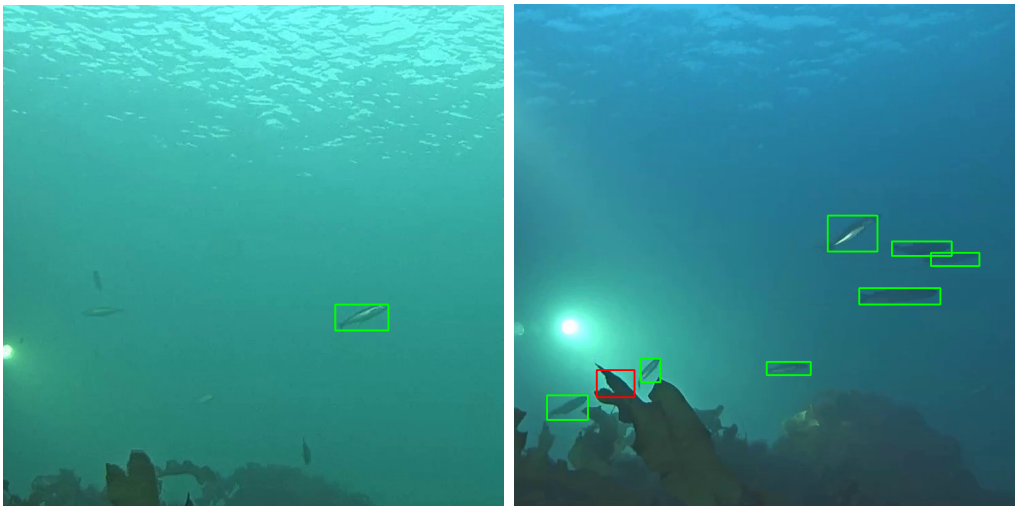
(e) Fish and Gadidae with low contrast.

(f) *Cyclopterus lumpus* in sea grass.

Figure 19: Good detection and classifications made by YOLOv3 with WordTree. Green bounding box indicates correct class and and bounding box.

(a) Detector misclassifying species and producing a bad bounding box.

(b) Turbid water where the detector mistakes the debris for a fish.



(c) Detector missing obvious fish.

(d) Detector mistaking sea weed for fish.

Figure 20: Bad detections and classifications made by YOLOv3 with WordTree.

6.5 Soft-NMS with YOLOv3

The modified NMS in section 6.3 has shown to increase confidence. A drawback of this classless NMS is however that it removes YOLO's ability to keep bounding boxes for different fish that slightly overlap. Bodla et al. [6] proposed Soft-NMS which decays the probabilities of overlapping boxes as a continuous function, as described in section 2.4, and achieves improved mAP. This experiment tests whether Soft-NMS improves the performance of YOLOv3-WT by modifying NMS and nothing else and then testing on the test set.

Measure	YOLOv3-WT	YOLOv3-WT Soft-NMS
oLRP ↓	0.556	0.556
mAP ↑	0.899	**0.918**
Precision ↑	0.936	0.936
Recall ↑	0.807	0.807
s	0.710	0.710

Table 11: Performance improvement from Soft-NMS on YOLOv3.

Table 11 shows that Soft-NMS increases the mAP with 1.9%, likley stemming from overlapping boxes for overlapping objects that are no longer removed. This is slightly higher than the improvements seen on the VOC 2007 and the COCO datasets [6]. However, no change is seen in oLRP, Precision nor Recall. This is because Soft-NMS gives overlapping bounding boxes very low probabilities, far below 0.71. As a result they are removed before calculating oLRP, Precision and Recall as the probability threshold for boxes to be included, s, removes these boxes. Boxes with lower probabilities are removed because otherwise the network would produce far too many erroneous predictions and decreasing performance.

While Soft-NMS improves the mAP score it would have no real impact on detection performance in normal use cases in this instance. Therefore it only partially confirms H5 as mAP performance is improved but no other improvements are seen.

This experiment concludes the modifications to the network itself. After all the improvements made in the previous experiments the network has become significantly different than YOLO. The object detector is now specifically tailored for fish detection and we therefore name it YOLO Fish.

6.6 Real-Time Fish Detection

Object detection of fish requires processing of large amount of data to be valuable, and data is generated as video continuously. Therefore it would be valuable to process video in real-time. One of the benefits of YOLOv3 is its ability to perform object detection on video in real-time [44]. This experiment tests whether our configuration is able to perform detection of fish in real-time. The inference time in table 12 is calculated by running through the testing set and averaging the inference time of each image.

Network	Network resolution	Average inference time (ms)	FPS
YOLO Fish	416x416	14.2	70
YOLO Fish	608x608	26.4	38

Table 12: Speed of YOLO Fish on a Tesla V100 graphics card.

Table 12 shows that YOLO Fish is fast enough to be used with real-time video on the network with input size of 608x608. A smaller network than the one used in the previous experiments can be used to achieve a higher speed, or for predictions on a less powerful computer. This however comes with the decrease in accuracy seen in table 13. These inference time numbers can be compared to the inference time numbers in table 1 however the Tesla V100 has 32% more flops than Tesla P100 and other improvements. Since YOLO Fish achieves state of the art performance on detecting Nordic fish species and achieves real-time speed this experiment confirms H6.

Measure	YOLO Fish-608	YOLO Fish-416
oLRP ↓	0.556	0.598
mAP ↑	0.918	0.908
Precision ↑	0.936	0.912
Recall ↑	0.807	0.806
s	0.710	0.670

Table 13: Performance penalty of using a network with a resolution of 416.

6.7 Detection Stability

Video object detectors' main performance metrics are mAP and FPS. Having a good average precision and inference time is very important, but it does not say everything about the quality of the detector. How stable the detections are from frame to frame is also an important factor. The stability of the detector is important for applications where objects have to be counted and also for the pleasantness if the results are shown to an end user.

The YOLO Fish detector has some inherent instability with suddenly missing an object in one frame and finding it in the next again. Also, hierarchical classification and non mutually exclusive classes leads to some class changes within the hierarchical branch. The result of this is a video that appears more choppy and uncertain than what is preferred.

The SORT object tracker explained in section 2.6 provides the ability to track objects across frames and keep objects even when the algorithm does not detect the object for a frame or two, increasing detector stability on video. Additionally, the tracking of objects makes it possible to smooth out classification by always applying the class that has most commonly been applied to that object instead of the most recent detected class. And as an added bonus, the use of object trackers makes it possible to count the number of fish that has entered the view frame in a given time period. However, if a fish enters and leaves it would be counted twice or more.

This experiment tests the detection stability of YOLO Fish and tests whether SORT and class smoothing increases detection stability. SORT was used with the parameters minimum hits 2 and maximum age 3.

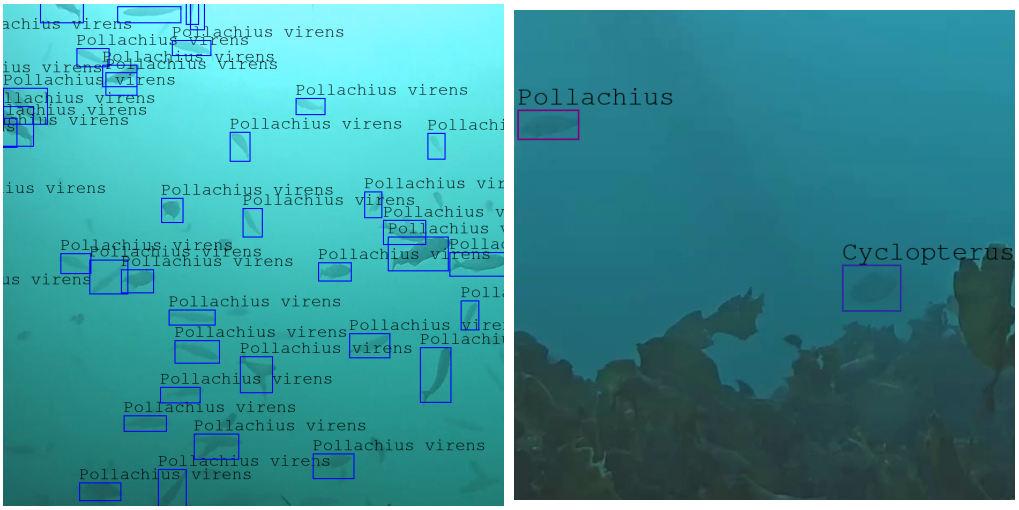
The method that is used for testing stability is counting the number of detections per class for one frame and finding the difference of detections for the same class in the next frame. This is then divided by the number of frames F that the experiment is run for, as in equation 14. B_t^i is the number of bounding boxes for class i at frame t. We call this measure Average Detection Difference (ADD). A lower ADD would indicate a more stable detector as there are fewer changes from frame to frame.

$$\text{ADD}(B) = \frac{1}{F} \sum_{t=1}^{F} \sum_{i=0}^{C} \left| B_{t-1}^i - B_t^i \right| \tag{14}$$

ADD is a simple way to measure changes in detection in video that does not require ground truth data, but has significant drawbacks. Other better metrics noted in section 3.4 that are based on error compared with ground truth data exist, but creating data for these is too laborious in this case. One drawback is that objects can leave the view port of the video, increasing the ADD but this will be negligible considering the number of frames. A second drawback is that it is not fruitful to compare different videos with ADD as number of objects in the frame and how frequent they move in and out of view significantly impact the results. It does also not say anything about how good the algorithm is because it is not a benchmark where there is a perfect score the algorithm can achieve, but rather just a measurement of change were we infer that lower is better too a point. It can however be used to compare with the baseline, which is without any processing.

Video clip	YOLO Fish	YOLO Fish with SORT
A	1.616	1.170
B	0.145	0.023

Table 14: ADD for the two video clips in figure 21.



(a) Video clip with very many *Pollachius virens*.

(b) Video clip with low number of various fish.

Figure 21: Images from the two video clips used for this experiment.

Video A seen in figure 22 has a very high number of one fish species and YOLO Fish struggles with localizing and often creates a bounding box in one frame and not the next. It also produces significant jitter in bounding boxes when many fish overlap. This leads to the high change in detections from frame to frame seen in table 14. Applying SORT and class smoothing decreases the change in number of bounding boxes from frame to frame significantly, meaning that the method manages to stabilize YOLO Fish when localization struggles.

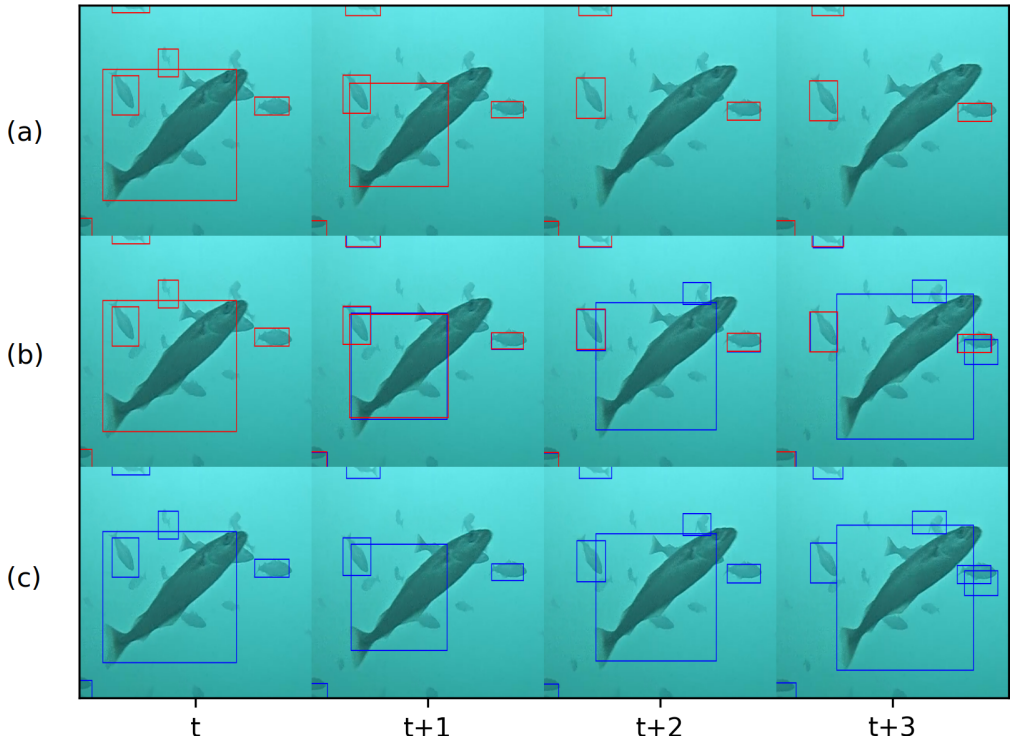


Figure 22: From video clip A. (a) is without SORT, (c) is with SORT and (b) is combined. Showing how YOLO Fish has instability in localization and how SORT reduces the instability.

Video B in figure 23 contains various species of fish and the detector has little trouble in localization. However because of bad lighting conditions and unclear water YOLO Fish jumps significantly around in the hierarchical branch leading to many class changes. In figure 23 the class goes from Fish to Gadidae to *Pollachius*, but when SORT with class smoothing is applied it does not change class just because one frame contained another species. The decrease in ADD when using SORT with class smoothing means that it manages to reduce noise in class changes.

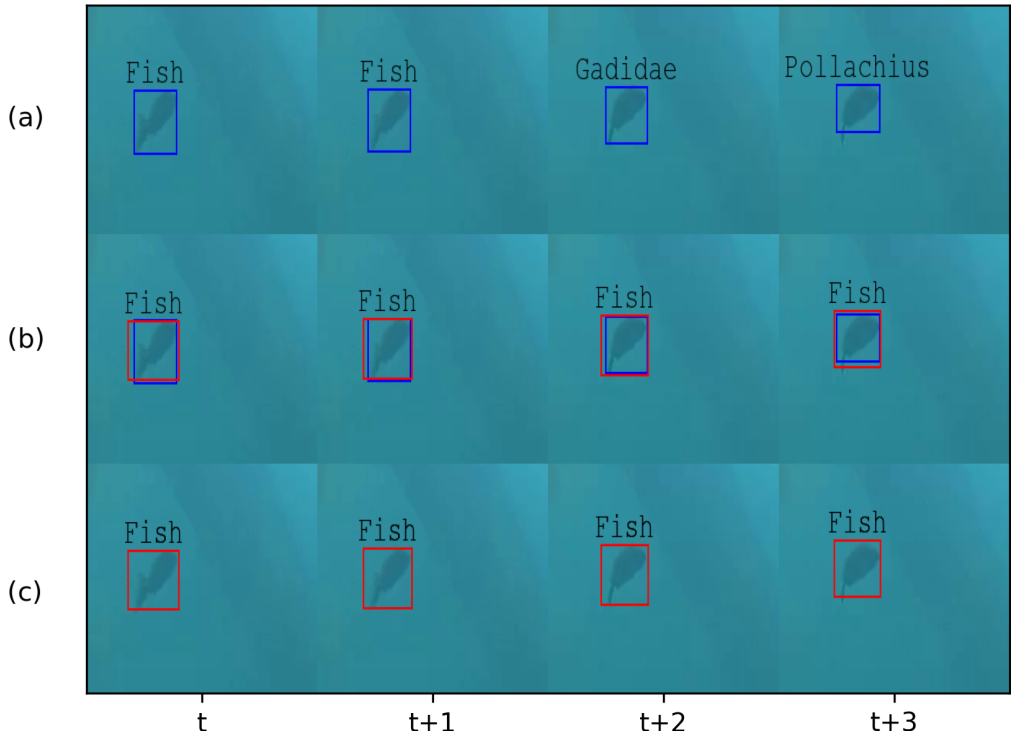


Figure 23: From video clip B. (a) is without SORT, (c) is with SORT and (b) is combined. Showing how YOLO Fish has instability in classification and how SORT with label smoothing reduces the instability.

The reduction in ADD in video A and B confirms H7 since both localization and classification on video across frames is more stable when SORT and class smoothing is done.

6.8 Summary

The experiments in section 6.1 to 6.5 gradually improves the performance of YOLO starting with YOLO 9000 and resulting in YOLO fish with the performance improvements in table 15. The improvements come as a result of first utilizing WordTree, then modifying NMS to account for the bounding box problem. After this the network is modified to use YOLOv3 with WordTree. Lastly NMS is improved again with Soft-NMS. The result of this process is the algorithm we call YOLO Fish.

	YOLO9000					YOLO Fish
WordTree?		✓	✓		✓	✓
Modified NMS?			✓		✓	✓
Yolov3 network?				✓	✓	✓
Soft-NMS?						✓
oLRP ↓	0.830	0.789	0.731	0.655	0.556	0.556
Precision ↑	0.693	0.731	0.877	0.800	0.936	0.936
Recall ↑	0.488	0.556	0.627	0.770	0.807	0.807
mAP ↑			0.728	0.793	0.899	0.918

Table 15: How the results were improved with the various new improvements.

After YOLO Fish is proven to work and has shown an improved performance, two experiments on video are performed. Section 6.6 verifies that the network is fast enough to be used on real-time video. The YOLO Fish 608 network achieves 26.4 ms inference time, enough to be used on 30 FPS video with overhead. YOLO Fish 416 achieves 14.2 ms inference time, enough to be used on 60 FPS video. Section 6.7 evaluates the quality of the resulting detection when put into a video format, and shows that the video suffers from significant label switching between frames. This is then improved by using the SORT algorithm.

7 Conclusion

Following the results of applying the techniques mentioned in the experiments YOLO Fish achieves state of the art performance at object detection of Nordic fish. YOLO Fish achieves an mAP of 91.8% on a dataset containing difficult conditions such as turbid water and nighttime pictures. It does this with inference times low enough for real-time detection.

Previous approaches often utilize a class containing unknown fish where fish species cannot easily be discerned. A problem with this is that the algorithm specifically learns that some fish are of the class 'unknown'. This gives them the ability to include fish they don't know the species of. However, it is reasonable to believe, that the detector learns that less visible fish are of class 'unknown' and not that this class can be applied when it is able to localize, but not classify. YOLO Fish however utilizes the inherent biological taxonomic tree of fish to perform hierarchical classification. This gives it the ability to apply higher hierarchical classes when it is uncertain, as it has learned what is common among all of the descendants of this higher class.

As a result of the increased context that the hierarchical classification provides, the algorithm achieves a higher mAP. For these reasons we propose that object detectors for fish should utilize hierarchical classification because of the large similarities among species and the nature of unclear underwater conditions.

So far in the conclusion, the foundation of the object detector has been discussed, but YOLO Fish has also seen improvements that make it suitable for real-time video. YOLO Fish achieves an inference time of 26.4 ms on a Tesla V100 GPU with high accuracy. However there are more aspects to video than accuracy and speed. The plain YOLO Fish algorithm suffers from largely varying detections from one frame to the next as the detections are done on a per frame basis. The SORT object tracker, which keeps information across frames, solves this. Thus YOLO Fish with SORT is a complete object detector for fish providing a pleasant and stable real-time video.

7.1 Further Work

While this thesis provides an effective fish detector, there are still further research that can be done to improve upon it. There are mainly three ideas for further research that we have identified so far. The first is to modify the algorithm to be able to count fish properly, the second is to use it for more intricate biological use cases, and the last one is to try out the proposed improvements in the recently released YOLOv4.

When it comes to fish counting, this algorithm makes it possible to count the number of fish that enters and leaves the view frame. This is done with the use of the object tracking algorithm SORT, that is mainly used to increase video stability. More advanced techniques that does object tracking and proper counting exists. For this algorithm to be effective at counting further research into object tracking and counting mechanisms is required.

As Yolo Fish handles classification using the taxonomic hierarchy, it is reasonable to believe that this could also be applied to differentiate the sex and age group of certain species. A specific example where this could be applied, has been noted in [37], where the species *S. Melops* has a male variant which disguises itself as its female counterpart. With the use of hierarchical classification one could have a 'female-like' child of the *S. Melops* and further a 'true female' and 'sneaker male'. This would make it possible to distinguish between them when it is possible and apply 'female-like' when it's not.

Bochkovskiy et al. [5] released YOLOv4 close to the end of this project. Here they employed several state of the art techniques to increase the performance of YOLOv3 and a decrease to inference time. It would be interesting to see whether the performance and speed of YOLO Fish could be improved by employing the techniques used by YOLOv4.

References

[1] Qaisar Abbas, Mostafa E. A. Ibrahim, and M. Arfan Jaffar. A comprehensive review of recent advances on deep vision systems. *Artificial Intelligence Review*, 52(1):39–76, Jun 2019. ISSN 1573-7462. doi: 10.1007/s10462-018-9633-3. URL https://doi.org/10.1007/s10462-018-9633-3.

[2] Akansha Bathija and Grishma Sharma. Visual object detection and tracking using yolo and sort, 2019.

[3] Keni Bernardin and Rainer Stiefelhagen. Evaluating multiple object tracking performance: the clear mot metrics. *EURASIP Journal on Image and Video Processing*, 2008:1–10, 2008.

[4] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 3464–3468. IEEE, 2016.

[5] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

[6] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S Davis. Soft-nms–improving object detection with one line of code. In *Proceedings of the IEEE international conference on computer vision*, pages 5561–5569, 2017.

[7] Coco Dataset. Coco 2017 object detection task, . URL http://cocodataset.org/#detection-2017. Accessed 12. April 2020.

[8] Coco Dataset. Coco detection evaluation, . URL http://cocodataset.org/#detection-eval. Accessed 12. April 2020.

[9] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387, 2016.

[10] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, page 233–240, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595933832. doi: 10.1145/1143844.1143874. URL https://doi.org/10.1145/1143844.1143874.

[11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[12] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[13] Mark Everingham, SM Ali Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International journal of computer vision*, 111(1):98–136, 2015.

[14] Cheng-Yang Fu, Wei Liu, Ananth Ranga, Ambrish Tyagi, and Alexander C Berg. Dssd: Deconvolutional single shot detector. *arXiv preprint arXiv:1701.06659*, 2017.

[15] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7036–7045, 2019.

[16] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[17] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, chapter 9. MIT Press, 2016. `http://www.deeplearningbook.org`.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015.

[21] Alan Hevner and Samir Chatterjee. Design science research in information systems. In *Design research in information systems*, pages 9–22. Springer, 2010.

[22] Phoenix X Huang, Bastiaan J Boom, and Robert B Fisher. Underwater live fish recognition using a balance-guaranteed optimized tree. In *Asian Conference on Computer Vision*, pages 422–433. Springer, 2012.

[23] Phoenix X Huang, Bastiaan J Boom, and Robert B Fisher. Gmm improves the reject option in hierarchical classification for fish recognition. In *IEEE Winter Conference on Applications of Computer Vision*, pages 371–376. IEEE, 2014.

[24] Koray Kavukcuoglu, Pierre Sermanet, Y-Lan Boureau, Karol Gregor, Michaël Mathieu, and Yann L Cun. Learning convolutional feature hierarchies for visual recognition. In *Advances in neural information processing systems*, pages 1090–1098, 2010.

[25] Kristian Muri Knausgård, Arne Wiklund, Tonje Knutsen Sørdalen, Kim Halvorsen, Alf Ring Kleiven, Lei Jiao, and Morten Goodwin. Temperate fish detection and classification: a deep learning based approach, 2020. Not yet published.

[26] Xiu Li, Youhua Tang, and Tingwei Gao. Deep but lightweight neural networks for fish detection. In *OCEANS 2017-Aberdeen*, pages 1–5. IEEE, 2017.

[27] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[28] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

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

[30] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8759–8768, 2018.

[31] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[32] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.

[33] Marcin Marszalek and Cordelia Schmid. Semantic hierarchies for visual object recognition. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–7. IEEE, 2007.

[34] Marcin Marszałek and Cordelia Schmid. Constructing category hierarchies for visual recognition. In *European conference on computer vision*, pages 479–491. Springer, 2008.

[35] David Nister and Henrik Stewenius. Scalable recognition with a vocabulary tree. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 2161–2168. Ieee, 2006.

[36] Kemal Oksuz, Baris Can Cam, Emre Akbas, and Sinan Kalkan. Localization recall precision (lrp): A new performance metric for object detection, 2018.

[37] Erlend Olsvik, Christian M. D. Trinh, Kristian Muri Knausgård, Arne Wiklund, Tonje Knutsen Sørdalen, Alf Ring Kleiven, Lei Jiao, and Morten Goodwin. Biometric fish classification of temperate species using convolutional neural network with squeeze-and-excitation, 2019.

[38] Ken Peffers, Tuure Tuunanen, Marcus A. Rothenberger, and Samir Chatterjee. A design science research methodology for information systems research. *Journal of Management Information Systems*, 24(3):45–77, 2007. doi: 10.2753/MIS0742-1222240302. URL `https://doi.org/10.2753/MIS0742-1222240302`.

[39] Diana Perry, Thomas AB Staveley, and Martin Gullström. Habitat connectivity of fish in temperate shallow-water seascapes. *Frontiers in Marine Science*, 4:440, 2018.

[40] Robert B. Fisher Phoenix X. Huang, Bastiaan B. Boom. "fish recognition ground-truth data, Sep 2013. URL `http://groups.inf.ed.ac.uk/f4k/GROUNDTRUTH/RECOG/`. Accessed 20. Jan 2020.

[41] Kazim Raza and Song Hong. Fast and accurate fish detection design with improved yolo-v3 model and transfer learning, 2020.

[42] Joseph Redmon. Darknet: Open source neural networks in c. `http://pjreddie.com/darknet/`, 2013–2016.

[43] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[44] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[45] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2015.

[46] Adrian Reithaug. Employing deep learning for fish recognition, 2018.

[47] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[48] Ahmad Salman, Ahsan Jalal, Faisal Shafait, Ajmal Mian, Mark Shortis, James Seager, and Euan Harvey. Fish species classification in unconstrained underwater environments based on deep learning: Fish classification based on deep learning. *Limnology and Oceanography: Methods*, 14, 05 2016. doi: 10.1002/lom3.10113.

[49] Zhiqiang Shen, Zhuang Liu, Jianguo Li, Yu-Gang Jiang, Yurong Chen, and Xiangyang Xue. Dsod: Learning deeply supervised object detectors from scratch. In *Proceedings of the IEEE international conference on computer vision*, pages 1919–1927, 2017.

[50] Shoaib Siddiqui, Ahmad Salman, Imran Malik, Faisal Shafait, Ajmal Mian, Mark Shortis, and Euan Harvey. Automatic fish species classification in underwater videos: Exploiting pretrained deep neural network models to compensate for limited labelled data. *ICES Journal of Marine Science*, 75, 05 2017. doi: 10.1093/icesjms/fsx109.

[51] Carlos N Silla and Alex A Freitas. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, 22(1-2):31–72, 2011.

[52] Arnold WM Smeulders, Dung M Chu, Rita Cucchiara, Simone Calderara, Afshin Dehghan, and Mubarak Shah. Visual tracking: An experimental survey. *IEEE transactions on pattern analysis and machine intelligence*, 36(7): 1442–1468, 2013.

[53] Johan Stål, Sandra Paulsen, Leif Pihl, Patrik Rönnbäck, Tore Söderqvist, and Håkan Wennhage. Coastal habitat support to fish and fisheries in sweden: Integrating ecosystem functions into fisheries management. *Ocean & Coastal Management*, 51(8-9):594–600, 2008.

[54] Chien-Yao Wang, Hong-Yuan Mark Liao, I-Hau Yeh, Yueh-Hua Wu, Ping-Yang Chen, and Jun-Wei Hsieh. Cspnet: A new backbone that can enhance learning capability of cnn. *arXiv preprint arXiv:1911.11929*, 2019.

[55] Ben G Weinstein. A computer vision for animal ecology. *Journal of Animal Ecology*, 87(3):533–545, 2018.

[56] Wikipedia. Taxonomic rank, . URL `https://en.wikipedia.org/wiki/Taxonomic_rank`. Accessed 5. march 2020.

[57] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. In *2017 IEEE international conference on image processing (ICIP)*, pages 3645–3649. IEEE, 2017.

[58] Xiu Li, Min Shang, H. Qin, and Liansheng Chen. Fast accurate fish detection and recognition of underwater images with fast r-cnn. In *OCEANS 2015 - MTS/IEEE Washington*, pages 1–5, Oct 2015. doi: 10.23919/OCEANS.2015.7404464.

[59] Fengwei Yu, Wenbo Li, Quanquan Li, Yu Liu, Xiaohua Shi, and Junjie Yan. Poi: Multiple object tracking with high performance detection and appearance feature. In *European Conference on Computer Vision*, pages 36–42. Springer, 2016.

[60] Hong Zhang and Naiyan Wang. On the stability of video detection and tracking. *arXiv preprint arXiv:1611.06467*, 2016.

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

[62] Alon Zweig and Daphna Weinshall. Exploiting object hierarchy: Combining models from different category levels. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8. IEEE, 2007.

Appendices

A - Code and dataset repository

B - Network configuration files

C - Fish hierarchy basis

Appendix A

YOLO Fish is a large project and is therefore hosted at the github repository `https://github.com/orilan93/darknet` which is based on the original work by Redmon [42] which is available at the github repository `https://github.com/pjreddie/darknet`. YOLO Fish is a Linux C program and built with make.

The repository also contains some helpful scripts. The evaluation script `calc_validation_darknet.py` which is used to calculate mAP and oLRP with hierarchical classification. Using the `fish_detector.py` a video file or stream can be processed and then displayed or streamed.

The dataset can be found at `http://dx.doi.org/10.17632/b4kcw9r32n.1` and includes the dataset, the best weights, and the meta files required by Darknet.

Appendix B

This appendix contains the configuration file for YOLO Fish which defines the entire architecture and training parameters. All of the configuration files are also found in the cfg folder in the code repository. Note that the following uses a three column page style.

```
[net]
# Testing
#batch=1
#subdivisions=1
# Training
batch=64
subdivisions=16
width=608
height=608
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=1000
max_batches = 500200
policy=steps
steps=400000,450000
scales=.1,.1

[convolutional]
batch_normalize=1
filters=32
size=3
stride=1
pad=1
activation=leaky

# Downsample

[convolutional]
batch_normalize=1
filters=64
size=3
stride=2
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=32
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=64
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

# Downsample

[convolutional]
batch_normalize=1
filters=128
size=3
stride=2
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=64
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=64
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

# Downsample

[convolutional]
batch_normalize=1
filters=256
size=3
stride=2
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
```

```
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

# Downsample

[convolutional]
batch_normalize=1
filters=512
size=3
stride=2
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
```

```
activation=linear


[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

# Downsample

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=2
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=1024
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear

######################

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=1024
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=1024
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=1024
activation=leaky

[convolutional]
size=1
stride=1
pad=1
filters=42
activation=linear
```

```
[region]
anchors =
64,52,92,111,252,208
classes=9
num=3
jitter=.3
max=90
coords=4
bias_match=1
rescore=1
softmax=1
object_scale=5
noobject_scale=1
class_scale=1
coord_scale=1
thresh = .6
absolute=1
tree=data/fish.tree

[route]
layers = -4

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[upsample]
stride=2

[route]
layers = -1, 61

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=512
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=512
activation=leaky

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
size=1
stride=1
pad=1
filters=42
activation=linear

[region]
anchors =
16,60,39,27,33,72
classes=9
num=3
jitter=.3
max=90
coords=4
bias_match=1
rescore=1
softmax=1
object_scale=5
noobject_scale=1
class_scale=1
coord_scale=1
thresh = .6
absolute=1
tree=data/fish.tree

[route]
layers = -4

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[upsample]
stride=2

[route]
layers = -1, 36

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=256
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=256
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=256
activation=leaky

[convolutional]
size=1
stride=1
pad=1
filters=42
activation=linear

[region]
anchors =
10,30,22,16,23,35
classes=9
num=3
jitter=.3
max=90
coords=4
bias_match=1
rescore=1
softmax=1
object_scale=5
noobject_scale=1
class_scale=1
coord_scale=1
thresh = .6
absolute=1
tree=data/fish.tree
```

Appendix C

This is the biological taxonomy that the hierarchical classification of YOLO Fish is based on. Fish is the root class and the next level starts on the family level and ends at the species level. To create the final hierarchy the species that were never seen and their branches are removed, furthermore classes we deemed redundant are also removed.