Faculty of Science and Technology
Department of Physics and Technology

Towards population counting of marine mammals based on drone images

Sigurd Røkenes

FYS-3941: Master's Thesis in Applied Physics and Mathematics
July 2022

Abstract

In marine science, there is a need for tools for population counting of species. Through this thesis we aim to achieve the follow three objectives: first, briefly discuss the state-of-the-art object detectors that can be used for the detection of porpoises in drone images/videos. Second, test and compare a few state-of-the-art object detectors in both quantitative and qualitative manner. Third, based on our results propose a set of suggestions that can be used for future studies associated with population counting. To answer the second question we compared three state-of-the-art object detection models, two single-stage detectors, and one two-shot detector. The models chosen were the Faster R-CNN, YOLOv4 and EfficientDet models, and they were trained and tested on a custom data-set consisting of 7300 labeled images of porpoises, where as 2300 of these were included in the test data set.

Through our experiments, we have discovered that YOLOv4 outperforms Faster R-CNN and EfficientDet D1 with detection, where YOLO achieves a recall of 97%, compared to 80% recall with EfficientDet D1 and 75% recall with Faster R-CNN. We also find the average precision $AP@50$ values of YOLOv4 to be 0.778, which is greater than EfficientDet D1 with 0.695 and Faster R-CNN with 0.686. Through both qualitative and quantitative methods we discover that both EfficientDet D1 and Faster R-CNN suffers from poor recall especially when porpoises overlap in the images. In the case of Faster R-CNN it misses nearly all detections when the porpoises overlap, but rarely non-overlapping detections. EfficientDet misses a significant portion of the overlapping detections, but also misses a few of the singular. Through examination of the COCO detection metrics, which favor bounding box accuracy, we also show that Faster R-CNN has more precise bounding boxes than YOLOv4 and EfficientDet D1 by comparing the less strict AP@50 values, with the stricter AP@75 and $AP[.50 : .05 : .95]$ values.

These results imply that a one-stage detection model in YOLOv4 could be used for object detection of porpoises from drone images. Based on the results, a few important areas for further investigation is outlined in the discussion, and a framework was developed which allows marine researches to easily perform porpoise detection from images and videos.

Acknowledgements

I would first like to thank my supervisor Puneet Sharma for excellent feedback and ideas throughout this project. In addition I express my gratitude to the UiT Machine Learning group for providing help whenever needed.

I would like to thank Marie-Anne Blanchet at the Norwegian Polar Institute for excellent information and feedback on my work.

I would also like to thank the Machine Learning ping pong group, which includes Nils, Harald, Joel, Nils, Preben, and Erland for keeping my sanity in check.

I would also like to thank my family for the moral support from the side-lines.

"Life is a journey, and I'm but a passenger" - Sigurd Røkenes (2022)

Contents

List of Figures

List of Tables

1 Introduction

1.1 Population Surveys of marine mammals

Investigating changes in biological, physiological, and demographic effects and their consequences on marine mammals is a vital issue for marine biologists [10]. Usually, marine mammal populations are monitored via surveys conducted by onboard observers (typically marine biologists) utilising ships or aircraft [10]. An example of this can be seen in figure 1.2, which shows a boat performing a line transect survey [30] in Balsfjord, Troms. Such surveys could provide imprecise estimates, as marine mammal populations are often spread over broad areas, and mammals are often submerged, i.e., they cannot be sighted [10]. Furthermore, the surveys are done repeatedly across several locations over time, making the task of conducting marine surveys quite resource intensive due to significant labour and equipment costs. These drawbacks make current population data of poor quality, as it can only detect significant population changes [28]. Access to frequent and more precise population count estimates could enable earlier enactment of measures to combat population declines.

For this thesis, we will focus on harbour porpoise detection. Harbour porpoises (figure 1.1) are a species of fully aquatic toothed whales. They are small in size, usually between 1-2m long. They need to surface to breathe and are commonly found along the coasts of the North Pacific Ocean, the North Atlantic Ocean, and the Black Sea [40].

Figure 1.1: Harbour porpoise example image. (Ecomare, Netherlands, Michael Brunvis)

Figure 1.2: Ship-based line transect survey for harbour porpoises (Balsfjord, Troms, UiT).

One disadvantage of ship surveys when counting animals such as porpoises is that they can only see the porpoises when surfacing. Depending on ocean conditions, we hypothesise that drone images can capture and spot the animals even when slightly below the surfaces, possibly leading to fewer missed detections. Figure 1.3 shows a drone image captured by the University of South Denmark in the waters around Denmark and is a part of the data set used in this project. As we can see, the conditions can be pretty challenging, with background objects making it difficult to differentiate between porpoises and background. The yellow bounding box in the image marks one of these difficult decisions, where it could be noise or a porpoise.

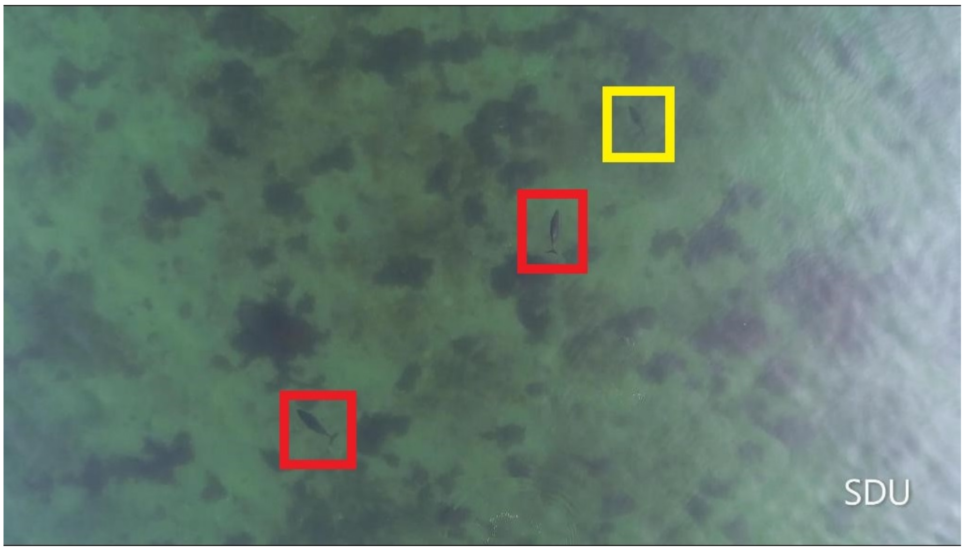


Figure 1.3: Example dataset image. The drawn boxes illustrate porpoises (red) and complex background resembling a porpoise (yellow).

1.2 Automated Object detection using Deep Learning

This thesis will explore deep learning options to utilise drone imaging to enable better and more accurate surveying of porpoises over a large area. We will attempt this by comparing the performance of three state-of-the-art object detection algorithms on drone images of porpoises collected by the University of South Denmark. We will explore three architectures: one Faster R-CNN[50] ResNet[23] variant, EfficientDet D1 [56], and YOLOv4[9]. Current publicised material rank models based on inference speed and mean average precision (*mAP*) [44]. This project will compare and contrast the three object detectors for the task of counting porpoises utilising metrics outlined in chapter 3 and section 6.1.1.

1.3 Contributions

Through this thesis, we aim to achieve the following three objectives: first, briefly discuss some state-of-the-art object detectors that can be used to detect porpoises in drone images/videos. Second, test and compare three state-of-the-art object detectors in both quantitative and qualitative manner. Third, based on our results propose a set of suggestions that can be used for future studies on population counting of marine animals.

1.4 Outline

This thesis starts by exploring the basics of convolutional neural networks and some essential techniques that these networks use. We consider it necessary to understand the theory behind the advanced architectures used later in the thesis. We then examine important terminologies and techniques for comparing object detection models in chapter 3. Chapter 4 first explains some networks used for image classification, as these are used as backbones in the three object detection networks we will discuss. We then review the three object detection methods used in this thesis: Faster R-CNN, EfficientDet and YOLO. We then explain how we processed the data set, which frameworks were used, and how the models were trained in chapter 5. Our experimental setup, results and the following performance discussion is located in chapter 6. We then propose some ideas for future research, and conclude based on our results in chapter 7. Note that some of the chapters, in particular chapters 1, 2 and 4, have been re-used from my unpublished pre-project thesis with permission from the UiT administration.

2

Convolutional Neural Networks

In this chapter, we will briefly describe some base operations behind state-of-the-art object detection models.

We can detect and recognize several objects in our environment using our visual system. Recent advances in hardware in the form of Graphical Processing Units (GPUs) have enabled complex algorithms to perform these tasks as well. The ability for computers to process images/videos enables systems to communicate and work with its surrounding in a new way. These technical improvements, along with the re-emergence of the convolution operation within image processing, have allowed significant advancements in the field.

2.1 The 2D convolution

The large advancements in image classification, and later object detection was made possible with the 2D convolution operation. The convolution of a kernel w of size $m\ x\ n$ with an image $f(x, y)$ is defined as:

$$(w * f)(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x-s, y-t). \tag{2.1}$$

Convolution is a spatial processing technique using spatial filters. These filters replace each pixel value with a function of the pixel values and their neighbors. It is apparent from equation 2.1 that all elements of w visit every pixel of f [21, p.322]. This operation drastically decreases the number of parameters of a network compared to the use of fully connected layers [7], where every feature in one layer influences every feature in the next.

Stacking layers utilising the convolutional operation is the basis of convolutional neural networks. For a networks output to yield a desirable result, we need to set the weights and biases of each separate layer. To do this we will need a performance measurement that summarizes the goals of the network.

2.2 Objective functions

Objective functions, also known as cost functions [64], measure how close a network's output is to the ground truth. The choice of cost function depends on the task at hand, like image classification, object detection, or image segmentation. In the object detection domain, it is common to use an optimized loss for combining the two tasks of classification and localization. We will cover this in section 4.

2.3 Regularization

A central problem in machine learning is how to make an algorithm that will perform well on not just training data, but also on new inputs it has never seen before. [21, p.221] The ability to perform well on previously unobserved inputs is called generalization [21, p.107]. When training a deep learning network we compute some error measure (loss function) on the training set, and end up with an optimization problem. However for the network to be generalized we also want the network to perform well on the test data. So to determine if a machine learning algorithm is good we need to evaluate both that its training error is small, and that the gap between the training and test error is small. An example of this can be seen in Figure 2.1. To the left of the green line, the network both training error and test error are high, called the underfitting zone [21, p.112], while to the right we have the overfitting zone, where the generalization gap increases with step size.

To achieve achieve a good generalized result, deep learning networks are heavily regularized through different techniques explained later in this report.

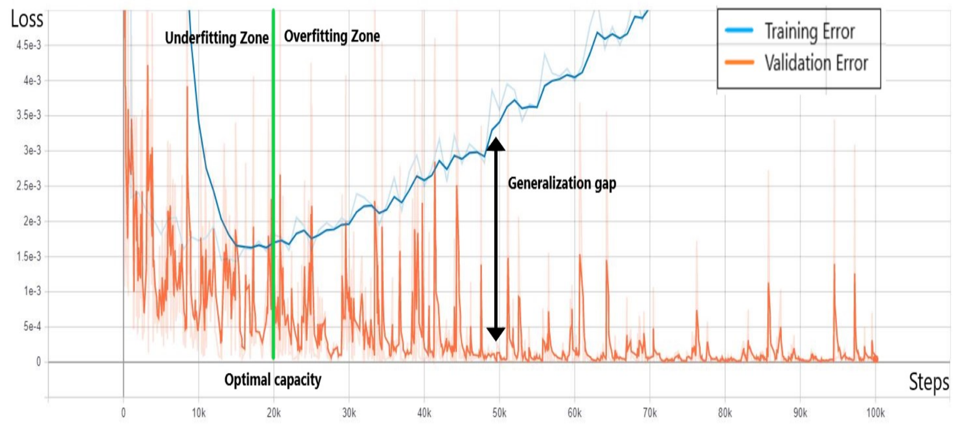


Figure 2.1: Shows an example from a typical neural network training situation.

2.4 Forward Pass

In a convolutional neural network (*CNN*), each layer will produce m feature representations of the image, where m is the number of filters in the layer. During the forward pass of the network each filter is convolved with the input volume, producing an activation map of that filter. As a result, the network will learn filters that activate when it detects some specific type of feature in that spatial position of the input. When stacking layer after layer, this becomes an approach to regularization in the network, where one could for example use smaller, and therefore less complex, filters to find patterns. [21, ch. 9]

2.5 Backpropogation

From the forward pass, each layer will contain m kernels, which will contain all m x n separate parameters. These kernels are randomly generated (or pseudo-randomly) to remove any bias from the search of optimal values for these weights. But how are these optimal weights found? CNNs is a supervised learning process. The dataset used for training is labeled with, in our case, bounding boxes containing the object of interest. This ground-truth is used to optimize the weights of each parameter, to tweak the network to predict the correct values. This process is called backpropogation, and consists of sending the gradient of the end-output of the network back through the entire network. The basic iteration step for updating the weights will be

$$\boldsymbol{w}_j^r(new) = \boldsymbol{w}_j^r + \Delta\boldsymbol{w}_j^r \tag{2.2}$$

where $\boldsymbol{w}_j^r$ is the filter matrix of the j-th neuron in the rth layer. The computation of the gradients is done through the chain rule, and from results in [58, p.166] we can show that the gradient can be written as:

$$\Delta\boldsymbol{w}_j^r = -\mu \sum_{i=1}^{N} \delta_j^r \boldsymbol{y}^{r-1}(i) \tag{2.3}$$

where: μ is the learning rate of the network, $\boldsymbol{y}^{r-1}$ is the output of the layer before layer r, and δ_j^r is the gradient of the cost function used in the network.

2.6 Activation functions

To further regularize the network, an activation function is applied to each filter output from the convolutional layers. The most commonly used activation function in modern networks is the Rectified Linear Unit (ReLU) [21, p.168], which is defined as:

$$f(x) = max(0, x) \tag{2.4}$$

One of the great advantages of ReLU has, is that it does not activate all neurons in the backward pass. All negative inputs will be set equal to zero, which will simplify and make it very computationally efficient, as much fewer neurons are activated at the same time. The gradient of the ReLU is defined as:

$$\frac{d}{dx}f(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

Equation 2.5 shows that if the gradient is negative, the weights for that neuron will not be updated in the backpropagation. One issue with this is if the randomly initialized weights are very bad, then we might end up in a situation where none of the weights update. If this happens using leaky ReLU is an option:

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0.01x & \text{if} x < 0 \end{cases} \tag{2.6}$$

In addition we have activation functions used like the sigmoid function [41]

$$f(x) = \frac{e^x}{e^x + 1}, \tag{2.7}$$

and the linear activation function

$$f(x) = ax. \tag{2.8}$$

Softmax is an activation function commonly used in the last layer (prediction), since it maps the output of all neurons to an output $x \in \{0, 1\}$, which could be interpreted as a probability for the neuron to be the correct prediction. The formula is

$$f(x) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}} \tag{2.9}$$

Some new activation functions have emerged, which we will discuss in later chapters.

2.7 Pooling Layers

The function of pooling layers is to replace the output of the network at a certain location with a summary statistic of the nearby outputs. This both decreases the computational complexity of the model by reducing the number of parameters in the subsequent layers, as well as helps make the model less invariant to small translations of the input [21, p.330].

An example of a pooling operation is the max pooling operation [72] which reports the maximum output within a rectangular neighborhood.

2.8 Batch Normalization

Batch normalization [27] is a method of adaptive reparametrization. This reparametrization significantly reduces the problem of coordinating updates across many layers, and can be applied to any input or hidden layer in a network. If we let $\boldsymbol{H}$ be a minibatch of activations of the layer to normalize, arranged as a design matrix, with the activations for each example appearing in a row of the matrix. To normalize $\boldsymbol{H}$ we replace it with

$$\boldsymbol{H}' = \frac{\boldsymbol{H} - \boldsymbol{\mu}}{\boldsymbol{\sigma}}, \tag{2.10}$$

where $\boldsymbol{\mu}$ is a vector containing the mean of each unit, and $\boldsymbol{\sigma}$ is a vector containing the standard deviation of each unit. The calculation of these depends on the model, where some calculate it across the entire mini-batch, and some use a running average that were collected during training time [21, p.309-311]

3 Evaluation Metrics

This chapter will describe the evaluation metrics used to evaluate object detection models. Object detection has two specific problems that need to be solved: localization and classification. The evaluation metrics must consider both sides of the problem, particularly the model's goals. The most common way to evaluate models is using average precision *AP* [44]. To understand AP, we first need to review some basic concepts.

3.1 Main Performance Metrics

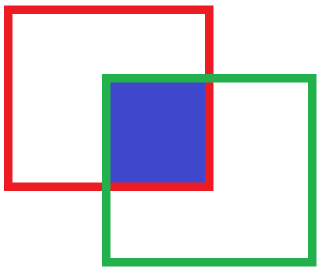
One of the core concepts of evaluating machine learning models is using a confusion matrix. A confusion matrix contains the total number of the following values:

- True Positive (*TP*): Valid detection.
- False Positive (*FP*): An incorrect detection of a nonexistent object or a misplaced detection of an existing object.
- False Negative (*FN*): Undetected ground truth bounding box.
- True Negative (*TN*): Correct misdetection.

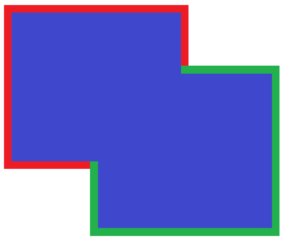
For object detection using bounding boxes, a true negative (TN) would represent all possible bounding boxes not detected, which becomes a vast number of bounding boxes within an image. For this reason, TN is not a metric used for object detection. For the other metrics in the confusion matrix, we need to establish what is defined as true and false positives. Here we introduce intersection over union (IOU). In the object detection scope, the IOU measures the overlapping area between a predicted bounding box $Bbox_p$ and the ground-truth bounding box $Bbox_{gt}$ divided by the area of union between them[44]. That is

$$IoU = \frac{\text{Area of overlap}}{\text{Area of union}} = \frac{\text{area}(gt \cap pd)}{\text{area}(gt \cup pd)} \tag{3.1}$$

where the area of overlap and area of union is illustrated in figure 3.1.



(a) Area of Overlap



(b) Area of Union

Figure 3.1: Illustrates the area of overlap and area of union where the area is marked using blue between a predicted bounding box (red) and a ground truth bounding box (red).

We can determine if the classification is correct by setting a minimum IoU threshold τ. If $IoU \geq \tau$ we label the prediction as a true positive, and if $IoU < \tau$, we consider the prediction a false positive.

Now that we have established how to sort the predictions, we can introduce the precision P and recall R concepts. These are defined as

$$P = \frac{TP}{TP + FP} = \frac{TP}{\text{all detections}} \tag{3.2}$$

$$R = \frac{TP}{TP + FN} = \frac{TP}{\text{all ground truths}} \tag{3.3}$$

Precision is a measurement of how well the classifier identifies relevant objects only. It is the percentage of correct detections. Recall is the percentage of true positives detected in all relevant ground truths, meaning the ability to find all relevant objects in an image.

An object detector's prediction contains, in addition to bounding box coordinates and class name, a confidence number that can be interpreted as a probability of a correct prediction [56]. A precision x recall curve can be used to visualize the precision and recall at different confidence values [44]. Adjusting the confidence threshold for detections to yield a higher recall commonly leads to a lower precision. Conversely, if one accepts fewer detections, the FP number decreases, causing a higher precision. A prediction x recall curve can be found by plotting the model's precision and recall for each prediction. We want a measurement that summarizes the precision x recall curve; however, they are often in a zig-zag shape, as seen by the non-interpolated precision values in figure 3.2 [46]. This poses issues with calculating values like area under the curve (*AUC*) and is commonly solved by interpolating the precision values for different recall thresholds. The new interpolated AUC values are called average precision *AP* and summarize the precision and recall in a single value.

3.2 Average Precision

To interpolate the precision x recall graph, we have two commonly used techniques: the 11-point interpolation and all-point interpolation.

In 11-point interpolation, the shape of the precision x recall curve is summarized by averaging the curve at each maximum precision value at a set of 11 equally spaced recall levels $i = [0, 0.1, \ldots, 1]$, given by

$$AP_{11} = \frac{1}{11} \sum_{R \in \{i\}} P_{\text{interp}}(R), \tag{3.4}$$

where

$$P_{\text{interp}}(R) = \max_{\tilde{R}:\tilde{R} \geq R} P(\tilde{R}). \tag{3.5}$$

Here AP is obtained by using the maximum precision $P_{\text{interp}}(R)$ whose recall value is greater than R, instead of using the precision $P(R)$ at each recall level.

In all-point interpolation, we interpolate through all points such that

$$AP_{all} = \sum_{n} (R_{n+1} - R_n) P_{\text{interp}}(R_{n+1}) \tag{3.6}$$

where

$$P_{\text{interp}}(R_{n+1}) = \max_{\tilde{R}:\tilde{R} \geq R_{n+1}} P(\tilde{R}). \tag{3.7}$$

Here, instead of the precision being observed at only 11 points, the AP is obtained by interpolating the precision at each level, taking the maximum precision whose recall value is greater or equal than R_{n+1} [43].

Since AP is calculated per class, we also need a measurement for the multi-class case. Here the mean AP (mAP) is the average AP over all classes

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i \tag{3.8}$$

with AP_i being the AP in the ith class and N is the total number of categories evaluated by the model.

The different measurements, interpolations techniques, etc., can make it challenging to compare object detection models directly. For this reason, comparing models using a standard data set and a standard metric became common.

3.3 Object detection challenges

Through the years, three object-detection challenges have been the most widely used: PASCAL VOC (2005-2012) [16], Open Images [31], and the Microsoft COCO (*COCO*) detection challenge [33]. All these challenges introduce their own data sets; however, in this thesis, we will focus on the metrics used for evaluation.

Both Open Images and PASCAL VOC [1] uses the all-point interpolated mAP value at IoU threshold 0.5 ($AP@50$) as the primary evaluation metric of the challenge.

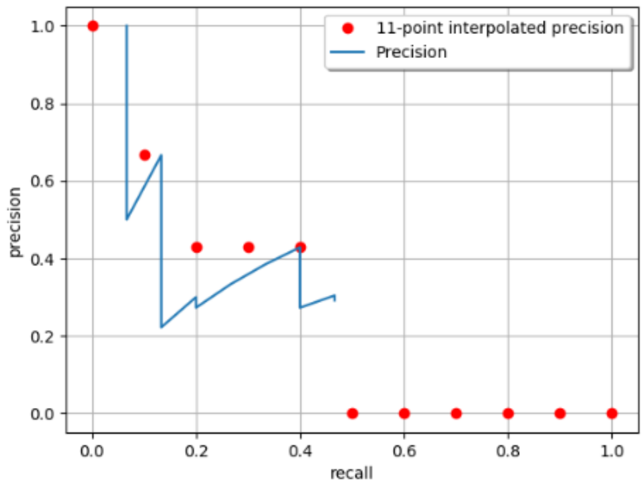
However, the Microsoft COCO detection challenge introduced new metrics used for evaluation. Here, the primary challenge metric was set to $AP[.50 : .05 : .95]$, which is the averaged AP values at the 10 IoU thresholds in $[0.50, 0.55, \dots, 0.95]$. This new metric rewards detectors with better localization [33]. In addition, COCO used 11 other metrics, for a total of 12, listed below.

- Average Precision:
  - $AP[.50 : .05 : .95]$
  - $AP@50$ (PASCAL VOC)
  - $AP@75$
- AP Across Scales:
  - $APsmall$
  - $APmedium$
  - $APlarge$

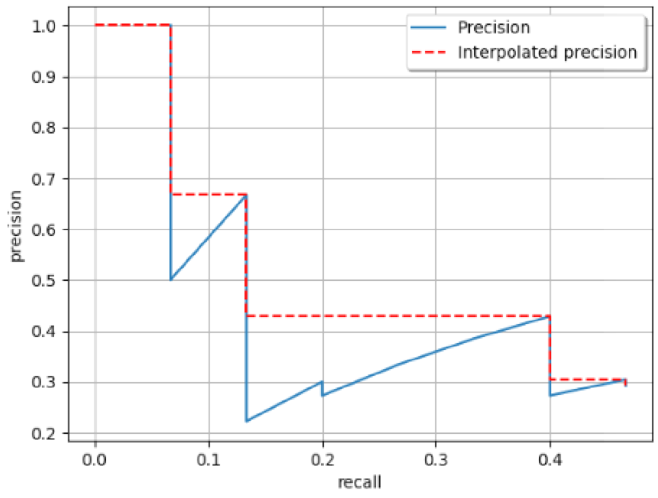
1. PASCAL VOC used 11-point interpolation to calculate AP until 2010, when it switched to all-point interpolation. In this thesis, when we mention PASCAL VOC AP, we will be referring to the post-2010 version that uses all-point interpolation.

- Average Recall (AR):
  - $AR^{max=1}$
  - $AR^{max=10}$
  - $AR^{max=100}$
- AR Across Scales:
  - $ARsmall$
  - $ARmedium$
  - $ARlarge$

Except for the $AP@50$ and $AP@75$ values, all AP and AR values are averaged across the 10 IoU thresholds $[0.50, 0.55, \ldots, 0.95]$. The AP and AR across scales values are also thresholded based on the ground truth bounding box area (measured in image pixels), whereas the categories are: $small < 32^2$, $32^2 \leq medium \leq 96^2$ and $large > 96^2$. In addition, AR is the maximum recall given a fixed number of detections per image, averaged over categories and IOUs [33].



(a) 11-point interpolation



(b) All-point interpolation

Figure 3.2: Illustration of a typical precision *x* recall curve with 11-point interpolation (**a**) and all-point interpolation (**b**). Figure by Padilla et al. [44].

4 Network Architecture

The architecture of deep convolutional networks (*DCNN*) consists of multiple chaining building blocks [6]. An ordinary object detector usually consists of a backbone network like VGG [52], ResNet [23] or CSPDarknet53 [61] followed by a head used for class prediction and bounding box generation. It is common to categorize object detectors whether they have a region of interest proposal step (two-stage) or a single network for both region proposal and object detection (one-stage) [56]. Some popular approaches to one-stage detection include the You Only Look Once *YOLO* [9], Single Shot Detector (*SSD*) [36] and EfficientDet architectures. For two-stage detectors some of the model architectures are Feature Pyramid Networks (FPN) [34] and different variants of region-based convolutional networks (R-CNN) [50]. This report will focus on YOLO, Faster R-CNN, and EfficientDet.

This chapter will first focus on the different backbone structures used and how they have been modified over the last few years. We then look at two-stage detectors, specifically how Faster R-CNN [50] was developed from R-CNN [20] and Fast R-CNN [19]. We then focus on One-stage detectors, first with the YOLO versions and then on the EfficientDet model family.

4.1 Backbone Networks

Most object detection-oriented DCNNs were initially designed for classification tasks before being adapted for object detection. These are now most commonly used as backbones for modern head structures. The backbones are used to generate features which are then sent to the head.

The backbone networks are usually pre-trained on image classification data sets (most commonly the Image Net data set, which contains millions of images of different categories). Pre-training can, in most cases, reduce training time on custom data sets by a large margin and play a major role in the performance of some architectures like Faster R-CNN, especially when the images resemble those in the dataset. [50]. Huang et al. [25] noted that as the classification accuracy of the backbone network increased on ImageNet classification tasks, the object detector performance based on those backbones also increased. However, as the best classifiers are most commonly very deep networks, the training time and inference become slower and more data-demanding. [6]

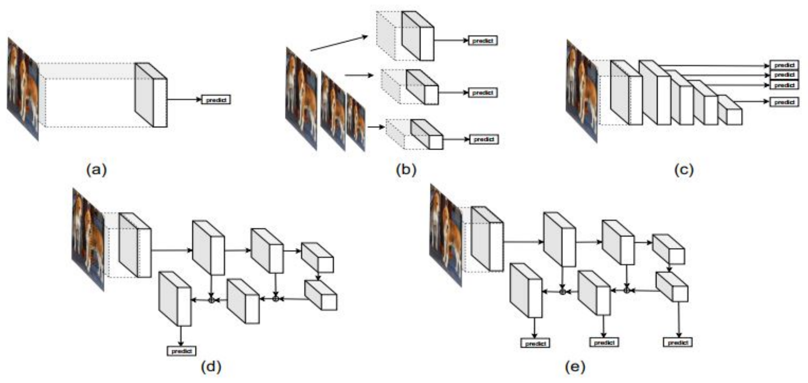


Figure 4.1: Figure 4 Illustrates how the backbones can be modified to give predictions at multiple scales and through fusion of features. (a) An unmodified backbone. (b) Predictions obtained from different scales of image. (c) Feature maps added to the backbone to get predictions at different scales. (d) A top-down network added in parallel to backbone. (e) Top-down network along with predictions at different scales. Figure from Agarwal et al.[6].

4.1.1 ResNet

Residual Nets (*ResNet*) is a variant of the HighwayNet [1] architectures, which are very deep convolutional networks with hundreds of layers. As shown in figure 4.2, the number of floating-point operations (FLOPS), especially on the deepest models, becomes very large. As the number of layers in a model increases, its training and inference time also increase. In addition to slower model performance, a degradation problem was observed with the deep models. As the network depth increases, the accuracy becomes saturated, and the accuracy starts declining rapidly. He et al. [23] showed that this degradation was not caused by overfitting since the added layers should be identity mapping, where the added layers are copied from the learned shallower layers. However, their experiments proved that this does not happen, and the training and test accuracy decrease with added layers. Residual Nets is their attempt to address this degradation problem, allowing deeper networks with higher accuracy.

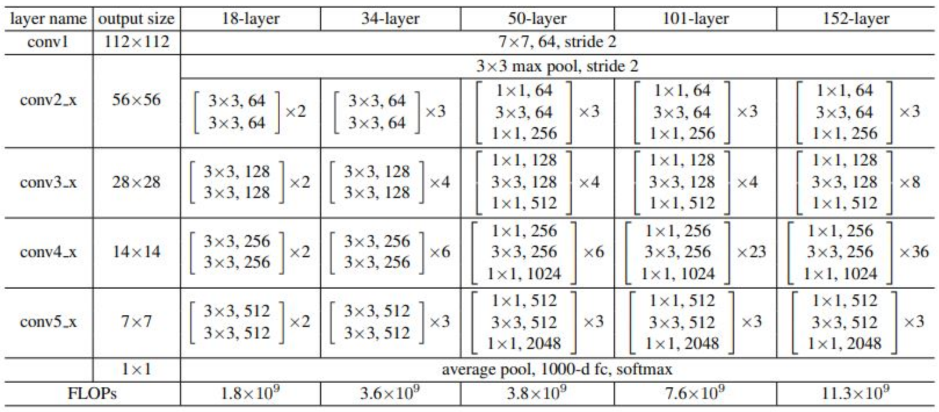
Instead of hoping each new stacked layer directly fit a desired underlying mapping $H(\boldsymbol{x})$, they let the stacked layers fit another mapping $F(\boldsymbol{x}) = H(\boldsymbol{x}) - \boldsymbol{x}$, and then let the original mapping be recast into $F(\boldsymbol{x}) + \boldsymbol{x}$[23]. These residuals can be found in a feedforward neural network using skip connections. The skip connections will perform identity mapping, and the output is added to the outputs of stacked layers [23]. The shortcut connections do not add to computational complexity and can still be trained with stochastic gradient descent (*SGD*) with back-propagation.

These results allowed ResNet to achieve convergence in much deeper networks than previous models and, in turn, improve accuracy. The architecture of the networks is based on the philosophy of VGG nets [52]: mostly 3 x 3 filters, and follows two rules:

1. For the same output feature map size, the layers have the same number of filters

2. If the feature map size is halved, the number of filters is doubled to preserve the time complexity per layer

The down-sampling happens directly by convolutional layers with stride of 2. The network ends with a global average pooling layer, followed by a 1000-neuron fully connected layer with a softmax activation function. The shortcut connections are used directly where the input and output are of the exact dimensions (identity mapping), and for the other layers, projection mapping, as defined in the study by He et al. [23] is used to increase dimensions for the residual. The number of layers depends on which model is used. The most common are 50-layer, 101-layer, and 152-layer, where speed and model complexity increase with more layers.



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 4.2: Baseline network by He et al. [23]. Downsampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2. (*Image source:* [23])

.

For the deep (50+ layers) architectures, ResNet introduces a bottleneck design to reduce training+inference times. These are explained in He et al. [23] as: "*For each residual function F, we use a stack of 3 layers instead of 2 (Fig. 4.3). The three layers are 1×1, 3×3, and 1×1 convolutions, where the 1×1 layers are responsible for reducing and then increasing (restoring) dimensions, leaving the 3×3 layer a bottleneck with smaller input/output dimensions.*".

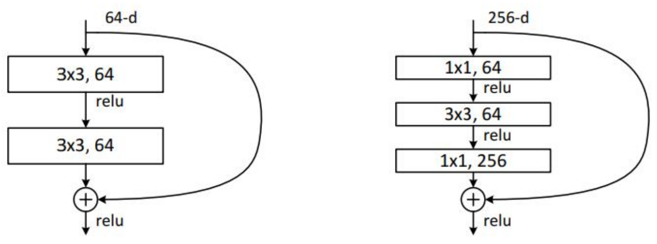


Figure 4.3: A deeper residual function *F* for ImageNet. Left: a building block (on 56x56 feature maps) (ResNet-34). Right: a "bottleneck" building block for ResNet-50/101/152. [23]

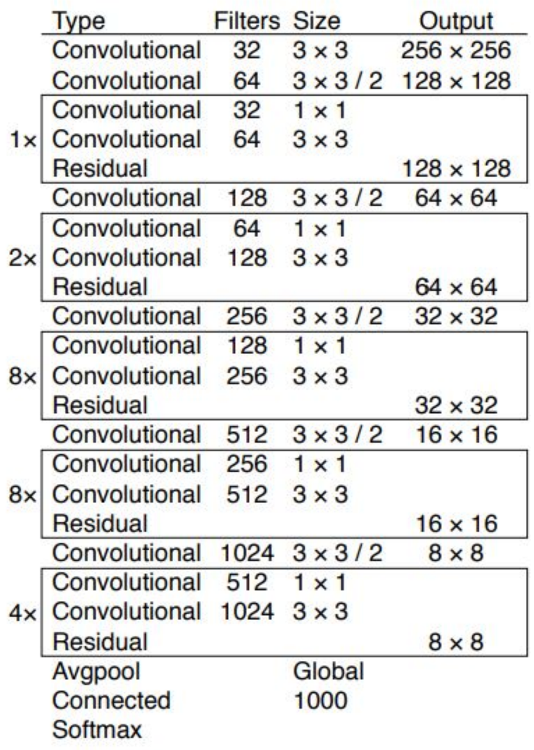
4.1.2 DarkNet53

Darknet was originally developed for YOLOv3[48] as an alternative to the ResNet architectures. This feature extractor is an attempt to keep as much of the accuracy of the ResNet architectures while increasing inference speed. The architecture of the network can be seen in figure 4.4, and as we can see it only has convolutions of $3 \; x \; 3$ and $1 \; x \; 1$, and is utilizing ResNet's[23] bottleneck skip connections mentioned in 4.1.1.

From the results in table 4.1 we see that DarkNet53 performs better than ResNet-101, equally to ResNet-152, while keeping the number of operations per second (*FLOPS/s*) much higher.

Backbone	Top-1	Top-1	Bn Ops	BFLOP/s	FPS
Darknet-19[47]	74.1	91.8	7.29	1246	**171**
ResNet-101[52]	77.1	93.7	19.7	1039	53
ResNet-152[52]	**77.6**	**93.8**	29.4	1090	37
Darknet-53[48]	77.2	**93.8**	18.7	**1457**	78

Table 4.1: Accuracy, billions of operations, billion floating operations per second, and FPS for various backbone networks on ImageNet classification tasks. Table by Redmon et al. [48].



	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 4.4: Architecture of the Darknet-53 backbone network. Figure by Redmon et al. [48].

4.1.3 EfficientNet

When choosing a model for a project, one big constraint is budget/computing power. With this in mind, Tan. et al. released the EfficientNet [54] scaleable architecture. The most common ways to increase accuracy on a DCNN was to either increase: the depth[23], the width [68], or the image-resolution [26]. The choice of scaling in these three parameters was made arbitrarily, often leading to sub-optimal results in accuracy and efficiency.

Problem formulation

A DCNN Layer i, was defined by Tan et al. [54] as a function $Y_i = \mathcal{F}_i(X_i)$, where $\mathcal{F}_i$ is the operator, Y_i is the output tensor, X_i is the input tensor with shape $\langle H_i, W_i, C_i \rangle$[1], where H_i and W_i are the spatial dimensions, and C_i is the

1. Batch dimension has been omitted for the sake of simplicity

channel dimension. Then a DCNN can be represented by the a list composed of layers: $\mathcal{N} = \mathcal{F}_k \bigodot \cdots \bigodot \mathcal{F}_1 \bigodot \mathcal{F}_1(X_1) = \bigodot_{j=1\ldots k} \mathcal{F}_j(X_1)$. Since a DCNN generally are partitioned in so called 'stages', where as the first layer of each stage performs the downsampling. As an example we can see that ResNet [23] as seen in figure 4.2 consists of 5 stages. A DCNN can then formally be defined as:

$$\mathcal{N} = \bigodot_{i=1\ldots s} \mathcal{F}_i^{L_i} \left(X_{\langle H_i, W_i, C_i \rangle} \right) \tag{4.1}$$

where $\mathcal{F}_i^{L_i}$ is the layer F_i repeated L_i times in stage i, $\langle H_i, W_i, C_i \rangle$ is the shape of input tensor X of layer i. Using this notation, maximizing model accuracy given resource constraints can be formulated as:

$$\begin{aligned} \max_{d,w,r} \quad & \text{Accuracy}(\mathcal{N}(d,w,r) \\ s.t. \quad & \mathcal{N}(d,w,r) = \bigodot \hat{\mathcal{F}}_i^{d \cdot \hat{L}_i} \left(X_{\langle r \cdot \hat{H}_i, r \cdot \hat{W}_i, w \cdot \hat{C}_i \rangle} \right) \\ & \text{Memory}(\mathcal{N}) \leq \text{Target memory} \\ & \text{FLOPS}(\mathcal{N}) \leq \text{Target FLOPS} \end{aligned} \tag{4.2}$$

where w, d, r are the scaling network parameters of width, depth and resolution, and $\hat{\mathcal{F}}_i, \hat{L}_i, \hat{H}_i, \hat{W}_i, \hat{C}_i$ are predefined parameters in a baseline network. The optimization of each of these proves to be difficult due to the parameters w, d, r being dependent on each other and the optimized value change under different resource constraints.

Scaling Dimensions

depth (d): Depth has been the traditional way of increasing accuracy on image classification tasks. Deeper networks, as explained in section 4.1.1, are said to capture more complex features and generalize well; however, large amounts of layers come with a high computational cost [23]. Figure 4.5 (middle) shows how accuracy increases with network depth and a diminishing return on this effect as depth increases.

Width(w): Scaling of network width (number of channels per layer) became especially popular in small-scale models, like MobileNet [24], MobileNetV2 [51],

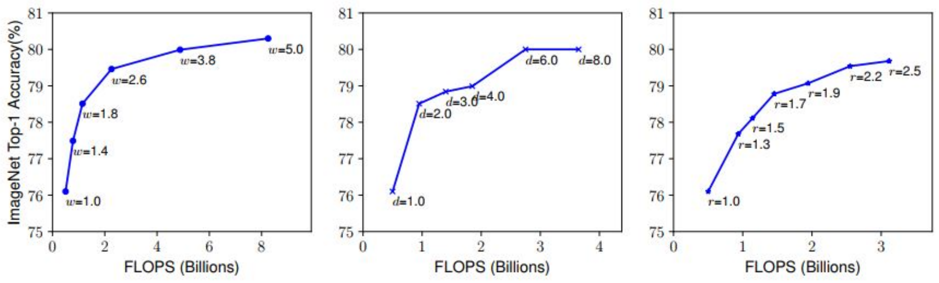


Figure 4.5: Scaling a baseline model with network width (w), depth (d) and resolution (r). This shows how accuracy quickly increases but also rapidly saturates when increasing a single parameter. *Image from Tan et al. [54]*

and MnasNet [55]. Wide networks are tend to be able to capture more fine-grained features, and are easier to train [54], but as discussed by Zagoruyko et al. [68], low-depth extremely wide networks struggles with higher level features. Figure 4.5 (left) shows the accuracy saturating with larger w.

Resolution(r): Higher resolution input images allows the capture of more fine-grained patterns [54]. As hardware, such as GPU's, come with higher memory, training of deep networks on high resolution images have also become more common [62]. Figure 4.5 shows the accuracy increasing from $r = 1$: (224 x 224) resolution, up to $r = 2.5$: (560 x 560) resolution. However we can also see here the diminishing returns effect present in the width and depth parameters.

Evident from the experiments in figure 4.5, increasing any of the three parameters does increase accuracy, however Tan et al. [54] proposed a method called *Compound Scaling* to uniformly scale the parameters simultaneously in a structured manner:

$$\begin{aligned} \text{depth: } d &= \alpha^{\phi} \\ \text{width: } w &= \beta^{\phi} \\ \text{resolution: } r &= \gamma^{\phi} \\ \text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 &\approx 2 \\ \alpha \leq 1, \beta \leq 1, \gamma &\leq 1 \end{aligned} \tag{4.3}$$

where α, β, γ are grid search determined constants. This means ϕ becomes a user-specified coefficient allowing accuracy control depending on the number of resources available. Since FLOPS of a convolution operation is proportional to d, w^2, r^2, scaling a DCNN using equation 4.3, FLOPS will approximately increase by $\left(\alpha \cdot \beta^2 \cdot \gamma^2\right)^{\phi}$.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \ x \ \hat{W}_i$	# Channels $\hat{C}_i$	# Layers $\hat{L}_i$
1	Conv3x3	$224 \ x \ 224$	32	1
2	MBConv1, k3x3	$112 \ x \ 112$	16	1
3	MBConv6, k3x3	$112 \ x \ 112$	24	2
4	MBConv6, k5x5	$56 \ x \ 56$	40	2
5	MBConv6, k3x3	$28 \ x \ 28$	80	3
6	MBConv6, k5x5	$28 \ x \ 28$	112	3
7	MBConv6, k5x5	$14 \ x \ 14$	192	4
8	MBConv6, k3x3	$7 \ x \ 7$	320	1
9	Conv1x1 & Pooling & FC	$7 \ x \ 7$	1280	1

Table 4.2: EfficientNet-B0 baseline network. Each row describes a stage i with $\hat{L}_i$ layers with input resolution $\langle \hat{H}_i, \hat{W}_i \rangle$ and output channels $\hat{C}_i$. *Table is from Tan et al. [54].*

EfficientNet Architecture

For the architecture, Tan et al. [54] developed a baseline model (table 4.2) using methods developed by Tan et al. on MnasNet [55]. This includes inverted residual blocks [51], which first widens the feature map with a $1 \ x \ 1$ convolution, then applies $3 \ x \ 3$ depthwise convolution [2] to reduce the number of parameters, before using a new $1 \ x \ 1$ convolution to reduce the number of channels so the residuals can be added. In contrast to a regular convolution, a depthwise convolution keeps each channel separate without mixing them, thus leading to fewer parameters for a full feature map.

The full architecture can be seen in table 4.2. Through experimentation Tan et al. [54] found the optimal parameters: $\alpha = 1.2, \beta = 1.1$ and $\gamma = 1.15$. These parameters were then used in equation 4.3 to obtain the EfficientNet B1-B7 models, where each new model increased FLOPS by 2^{ϕ}.

The compound scaling is proven to improve both MobileNet and ResNet, and the EfficientNet network outperforms these backbones in image classification with much fewer parameters [54].

Now that we have explained the backbone networks used for our chosen object detection models, we will start discussing two-stage detectors, specifically R-CNN.

4.2 Two-Stage Detector

The process of object detection can be divided into two parts: 1) proposing regions and 2) classifying and bounding box regression. [6]. Hence there are two parts: a proposal generator that presents the classifier with features that contain objects in the ground truth, and the classifier which assigns a class to each of the proposals and fine-tunes the coordinates of the boxes.

4.2.1 R-CNN and Fast R-CNN

The Region-based CNN (*R-CNN*) [20] and fast R-CNN [19] architectures were significant breakthroughs for object detection models, although both have been abandoned now in favor of faster models. R-CNN proposes regions using selective search [59], which proposes 2000 regions for each image, and then narrows it down. The resolution of these regions is then warped to match a CNN classifier's resolution. The output of the CNN classifier is then run through an SVM [13] to further fine-tune coordinates. This method, while not bad for accuracy, was incredibly slow compared to modern networks [19].

Fast R-CNN [19] was then built upon the previous work in the study by Girschick et al. [20]. Here both the full input image and the proposed regions (from selective search [59]) are fed into a deep CNN. These were further sent to a region of interest (*RoI*) pooling layer, which after a sequence of fully connected (*fc*) layers, extracts a fixed-length feature vector from the feature map. This feature vector is called an RoI feature vector [19], and was used as an input to two different fully connected layers: one using SoftMax (equation 2.9) to output the probability of each class, and one using a bounding box regressor to output the four bounding box coordinates. An illustration of this can be seen in figure 4.6.

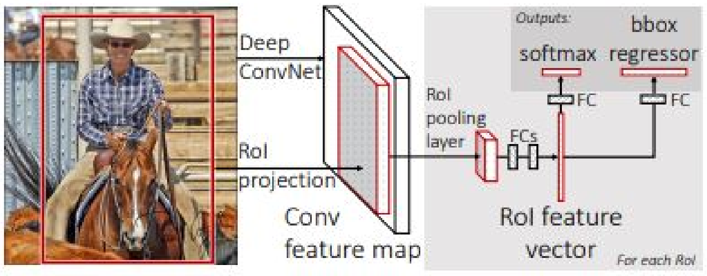


Figure 4.6: Fast R-CNN architecture. An input image and multiple regions of interest (RoIs) are input into a fully convolutional network. Each RoI is pooled into a fixed-size feature map and then mapped to a feature vector by fully connected layers (FCs). The network has two output vectors per RoI: softmax probabilities and per-class bounding-box regression offsets. The architecture is trained end-to-end with a multi-task loss. [19]

This network model also needed a new loss function for the back-propagation since it contains two different outputs. A multi-task loss L was decided on, which takes the classification loss L_{cls} and bounding-box regression loss L_{loc} into account. For each training RoI labeled with ground-truth class u and a ground-truth bounding-box regression target v, it calculates:

$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda[u \geq 1]L_{loc}(t^u, v) \tag{4.4}$$

where the classification loss is the log loss for true class u:

$$L_{cls}(p, u) = -\log p_u. \tag{4.5}$$

$[u \geq 1]$ is the Iverson bracket indicator function, which evaluates to 1 when $u \geq 1$, and 0 otherwise. The hyper-parameter λ controls the balance between the two task losses. However, it is usually left at 1. The background of the images is by convention labeled $u = 0$, meaning that for the RoIs L_{loc} will be ignored for bounding boxes only containing background. For the bounding-box regression loss Girschik et al. [50] decided to use:

$$L_{loc}(t^u, v) = \sum_{i \in \{x,y,w,h\}} \text{smooth}_{L_1}(t_i^u - v_i), \tag{4.6}$$

where

$$\text{smooth}_{L_1}(t_i^u - v_i)(x) = \begin{cases} 0.5x^2 & \text{if}|x| \leq 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}. \tag{4.7}$$

The RoI pooling layer is a special case of the spatial pyramid pooling layers used in SPPnets [22], with only one pyramid level. It works by using max-pooling to convert the features inside a region of interest into feature maps with a fixed spatial extent of $H\ x\ W$, where H and W are layer hyperparameters independent of the RoI. In Fast R-CNN [19] each RoI is defined by the tuple (r, c, h, w) that specifies coordinates of the top-left corner (r, c) and its height and width (h, w). RoI max-pooling divides the $h\ x\ w$ window into a grid of $H\ x\ W$ sub-windows of size $h/H\ x\ w/W$, then uses max-pooling on these sub-windows and applies this value to the corresponding output grid cell.

To explain the back-propagation through the RoI pooling layer, let us first let batch number $N = 1$ to simplify notations[2]. Girshick et al. then explained the results for back-propagation [19] as: "Let $x_i \in \mathbb{R}$ be the i-th activation input into the RoI pooling layer and let y_{rj} be the layer's j-th output from the r-th RoI. The RoI pooling layer computes $y_{rj} = x_{i*(r,j)}$ in which $i*(r,j) = \text{argmax}_{i' \in R(r,j)} x_{i'} \cdot R(r,j)$ is the index set of inputs in the sub-window over which the output unit y_{rj} max pools. A single x_i may be assigned to several different outputs y_{rj}." The backward-pass is then calculated by the following partial derivative of the loss with respect to input variable x_i:

$$\frac{\partial L}{\partial x_i} \sum_r \sum_j [i = i*(r,j)] \frac{\partial L}{\partial y_{rj}}. \tag{4.8}$$

This network achieves inference speeds 200 times faster than the original R-CNN and, if ignoring the feature extraction, reaches close to real-time speeds in inference, all while improving precision. However, the feature extraction is still time-consuming, hence the emergence of Faster R-CNN [50].

4.2.2 Faster R-CNN

Faster R-CNN [50] aimed to lower the inference speeds even more by improving the speeds of the region proposals. With the rise of 'attention'[12][60] mechanisms, it was proposed to use this for a convolutional region proposal network *(RPN)[50]*. This RPN takes an image as input and outputs a set of

2. Each image is treated independently in the forward pass, so this will not affect cases of $N \geq 1$.

rectangular object proposals with an objectness score. The ultimate goal was to have the RPN share computation (layer-parameters) with the fast R-CNN[19] object detection network.

The RPN feeds an input image into the backbone CNN. On the output features from the backbone, a set of 'anchors' are placed on the input image for each location on the output feature map, where each anchor represents a possible object. All these anchors are then checked by running a $3\ x\ 3$ convolution with 512 filters to the feature map. This is followed by two sibling layers, similar to those mentioned in 4.2.1, however fully convolutional instead of fully connected. The two sibling layers were:

- $1\ x\ 1$ convolution with 18 filters to output detection scores (classification branch)

- $1\ x\ 1$ convolution with 36 filters for bounding box regression (regression branch)

The 18 filters in the classification branch will output two coordinates, (H, W), that will be used to calculate the probability of whether or not each point in the backbone feature map contains an object within all anchors. The 36 filters in the regression branch will output four regression coefficients for each anchor in the backbone feature map, which is used to improve further the coordinates of the anchors that contain objects.

For each anchor, the following conditions are checked on whether the anchor should be labeled a positive sample or not:

- **Positive:**

- If the anchor has the highest IoU with a ground truth box.

- The IoU is greater than 0.7 with any ground truth box.

- **Negative:**

- If the anchor has IoU ≤ 0.3 with all ground truth boxes.

If neither condition is met, the anchor is disregarded for the training of the RPN.

During the training of the RPN, the network performs sampling of anchors to form mini-batches. Each mini-batch contains the same negative and positive samples, padding with additional negative samples when there are not enough positive samples. This process means the loss function (equation 4.4) must be modified with a normalization factor for each mini-batch N_{cls}, N_{reg}. The new loss function is then defined by Ren et al. [50] to be:

$$L(p, u, t^u, v) = \frac{1}{N_{cls}} L_{cls}(p, u) + \frac{1}{N_{loc}} \lambda [u \geq 1] L_{loc}(t^u, v). \tag{4.9}$$

The sharing of features between the RPN and Fast R-CNN [19] is done using 4-step alternating training [50].

1. The RPN is trained independently using a pre-trained backbone network. The weights will be fine-tuned for region proposals.

2. The Fast R-CNN detector is trained independently using a pre-trained backbone network and the region proposals from the RPN. The RPN weights will not be updated during this training process. Only the backbone, the RoI pooling, and the following FCN layers will be tuned for object detection.

3. The RPN is trained independently again, this time with weights from the Fast R-CNN. All weights in common between the RPN and the detector remain fixed.

4. The fast-RCNN detector is then fine-tuned, also here with the common layers being fixed.

This allows the Faster R-CNN network to share the parameters in the backbone and convolutional layers, allowing inference speeds up to $10x$ faster [50] than the Selective Search [59] method mentioned in section 4.2.1, as well as improving mAP and on the MS COCO competition dataset [50].

4.3 Single Stage Detector

In this section, we will discuss single-stage detectors. We will start with discussing the evolution of the "you only look once" *(YOLO)* models before discussing the EfficientDet model family. To fully understand all the features of YOLOv4 which we used for the experiments, the previous versions of YOLO will also be described.

4.3.1 You only look Once *(YOLO)*

YOLO is an algorithm proposed by Redmon et al. [49] as an alternative to the Double Stage Detectors. In short, YOLOv1 is a method where raw image pixels are converted to bounding-box coordinates and class probabilities by convolutional layers, which meant it could be optimized (differentiable) end-to-end directly. This allowed a proposal-free single feedforward pass when running inference, which allows great speed in the detections.

4.3.2 YOLO v1

In YOLOv1 [49] the feature extraction is done by dividing the input image into an $S\ x\ S$ grid. If the center of an object falls into a grid cell, that cell will be responsible for detecting that object. B bounding boxes and confidence scores are predicted for each grid cell. The confidence is defined as [49]:

$$\text{Confidence} = P(\text{object}) \cdot \text{IoU}_{\text{pred}}^{\text{truth}} \tag{4.10}$$

This equation should evaluate to zero if no object exists in the cell. Furthermore, it should be equal to the IoU between the prediction and the ground truth. Each bounding box contains five predictions: x, y, w, h, and confidence, where (x, y) represents the center of the bounding box relative to the bounds of the grid cell, and (w, h) the width and height of the bounding box. For each bounding box B, a score is calculated using

$$P(\text{Class}_i|\text{Object}) \cdot P(\text{Object}) \cdot \text{IoU}_{\text{pred}}^{\text{truth}} = P(\text{Class}_i) \cdot \text{IoU}_{\text{pred}}^{\text{truth}} \tag{4.11}$$

which are class-specific confidence scores that encode the probability of that class appearing in the box and how well the ground truth object box matches the predicted box.

The architecture was kept simple for YOLO(v1). The entire network only consists of 24 convolutional layers, followed by 2 FCNs. The architecture can be seen in figure 4.7. The final layer predicts both class probabilities and bounding box coordinates. The bounding box width and height are normalized by image width and height. The coordinates are also parameterized to be offsets of a particular grid cell location bounded between 0 and 1. The final layer is encoded as a $S\ x\ S\ x\ (B \cdot 5 + C)$ tensor, where C is the number of classes in the dataset.

For activation functions, the linear activation function is used in the final layer, while all other layers use the leaky rectified linear activation:

$$\phi(x) = \begin{cases} x & \text{if } x > 0 \\ 0.1x & \text{otherwise} \end{cases} \tag{4.12}$$

The loss function is set to be a multi-part loss given as:

$$\begin{aligned} L = {} & \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\ & + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\ & + \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} (C_i + \hat{C}_i)^2 \\ & + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{noobj} (C_i + \hat{C}_i)^2 \\ & + \sum_{i=0}^{S^2} 1_{i}^{obj} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \end{aligned} \tag{4.13}$$

where 1_i^{obj} denotes if an object appears in cell i and 1_{ij}^{obj} denotes that the jth bounding box predictor in cell i is responsible for that prediction. When the prediction is close to the border of multiple cells, non-maximum suppression (*NMS*) [8] is used to fix the issue where multiple prediction boxes exist for the same object.

The first version of *YOLO* was ground-breaking in speed. However, the precision was not great, especially on smaller objects that appeared in groups [47]. Each grid cell can only predict two boxes and can only have one class. It also utilizes very coarse features to predict classes, as multiple down-sampling layers are from the input image. Some of these are addressed in the newer versions of YOLO.

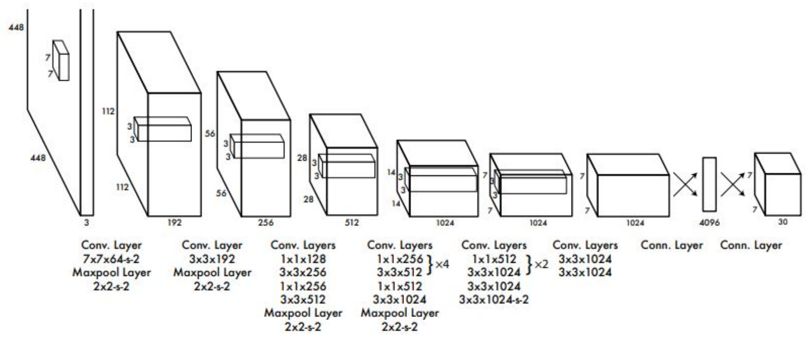


Figure 4.7: The architecture of YOLOv1. Contains a total of 24 convolutional layers, followed by two fully connected layers. Alternating between 3 x 3 and 1 x 1 layers similar to Lin et al. in [32], reduces the feature space from preceding layers which also reduces computational complexity.

4.3.3 YOLO v2

In a study by Redmond et al. [47] YOLOv2 was introduced with multiple improvements to the original structure.

Batch Normalization[27] is added to regularize the model training, allowing the Dropout layers to be removed without overfitting as easily.

Convolutional with anchor boxes. Instead of dividing the image into grids as described in section 4.3.2, YOLOv2 took two ideas from faster R-CNN [50]: replacing the fully connected layers with 1 x 1 convolutions, and using anchor boxes to predict the bounding boxes as described in section 4.2.2. However, directly implemented, the anchor boxes does not work well in the YOLO architecture. The box dimensions has to be hand-picked, which means the network might have to adjust these dimensions a lot during training. With better priors for the box dimensions, the training speed would improve. To automate choosing good priors, YOLOv2 uses k-means clustering [39] on the training set bounding boxes (named **Dimension clusters**[47]). As a distance measurement in the k-means, the formula used was [47]:

$$d(\text{box}, \text{centroid}) = 1 - \text{IoU}(\text{box}, \text{centroid} \tag{4.14}$$

which is independent of the size of the box (object), meaning larger boxes do not generate more errors than smaller boxes.

Another issue with the anchor boxes was model instability. Most of this instability came from predicting the box's (x, y) coordinates. In the RPN from section 4.2.2 the network predicts values t_x and t_y, and then the center coordinates used in YOLO is calculated as

$$
\begin{aligned}
x &= (t_x \cdot w_a) - x_a \\
y &= (t_y \cdot h_a) - y_a.
\end{aligned}
$$

This formulation allows any anchor box to end at any point in the image. With random initialization, it takes a long time for the model to stabilize and give sensible offsets. To solve this **Direct location prediction** is introduced, where instead of prediction offsets, they predict location coordinates relative to the location of the grid cell. The network predicts 5 coordinates for every bounding box t_x, t_y, t_w, t_h and t_σ. Then if the cell is offset from the top left corner (c_x, c_y) and the bounding box prior has width and height p_w, p_h, then the predictions become

$$
\begin{aligned}
b_x &= \sigma(t_x) + c_x \\
b_y &= \sigma(t_y) + c_y \\
b_w &= p_w e^{t_w} \\
b_h &= p_h e^{t_h} \\
P(\text{Object}) \cdot \text{IoU}(b, \text{Object}) &= \sigma(t_\sigma) \qquad (4.15)
\end{aligned}
$$

This will restrain the location prediction, and the parametrization is much easier to learn, stabilizing the network's training.

Further, one of the issues with the original YOLO was scaling issues. The performance on small objects was poor, and a few improvements were made. The original YOLO ran classification on a feature map of $13\ x\ 13$ resolution. Now a shortcut connection, similar to the ResNet[23] identity mappings mentioned in section 4.1.1 was added to the network, allowing the network to have access to more fine-grained features. In addition the training process was updated to use **Multi-scale training**[47]. Here for every ten batches during training, the resolution of the network was changed. As an example, if the first 10 batches were trained on the original resolution of $416\ x\ 416$, then the next 10 batches would be trained with a random resolution between the following multiples of 32: $\{320, 352, \ldots, 608\}$. This forced the network to learn to predict across various input dimensions while not affecting training speed much.

Redmond et al. [47] implemented improvements in the backbone network. The new backbone network, called Darknet-19, can be seen in table 4.3. This new backbone utilized tricks from VGG-16[52], Network In Network (*NIN*)[32], and the GoogleNet [53] architectures, to achieve high accuracy as well as speed in detection tasks.

Type	Filters	Stride	Output
Convolutional	32	3x3	224x224
Maxpool		2x2/2	112x112
Convolutional	64	3x3	112x112
Maxpool		2x2/2	56x56
Convolutional	128	3x3	56x56
Convolutional	64	1x1	56x56
Convolutional	128	3x3	56x56
Maxpool		2x2/2	28x28
Convolutional	256	3x3	28x28
Convolutional	128	1x1	28x28
Convolutional	256	3x3	28x28
Maxpool		2x2/2	14x14
Convolutional	512	3x3	14x14
Convolutional	256	1x1	14x14
Convolutional	512	3x3	14x14
Convolutional	256	1x1	14x14
Convolutional	512	3x3	14x14
Maxpool		2x2/2	7x7
Convolutional	1024	3x3	7x7
Convolutional	512	1x1	7x7
Convolutional	1024	3x3	7x7
Convolutional	512	1x1	7x7
Convolutional	1024	3x3	7x7
Convolutional	1000	1x1	7x7
Avgpool		Global	1000
Softmax			

Table 4.3: Darknet-19 Architecture.

These improvements drastically increased the precision and recall on the COCO datasets [47] compared to the original YOLO version, and the performances were closing in on the much slower double-stage detectors.

4.3.4 YOLO v3

YOLOv3 was proposed in a study by Redmon et al. [48]. The first improvement came in the form of adding an objectness score for each bounding box using logistic regression, similar to the method described in section 4.2.2, however, with a threshold of 0.5 instead of 0.7. This system only assigns one bounding box prior to each ground truth object. If a bounding box prior is not assigned to a ground truth object, it incurs no loss for coordinate or class predictions, only objectness [48]. During training, the sum of squared error loss is used. Using the equations in 4.15, the gradient of the squared error will be calculated from $\hat{t_*} - t_*$, where $\hat{t_*}$ is the coordinate ground truth, and t_* is the coordinate prediction.

Class Prediction was also generalized for use on more complex data-sets. The softmax function assumes that each box has exactly one class, which is often not the case. Instead, independent logistic classifiers (sigmoid) were used, as well as binary cross-entropy [70] loss for the class predictions. Binary cross-entropy is given as

$$L_{cls} = -\frac{1}{N_c ls} \sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i)) \tag{4.16}$$

In addition, a new backbone network was introduced with the Darknet-53, as described in section 4.1. This allowed the network to **predict across scales**. Similar to the concept of feature pyramid networks [34], in addition to the new backbone, several convolutional layers were added, with the last of these predicting a 3-d tensor encoding bounding box, objectness, and class predictions. For each scale output from the backbone, 3 boxes are predicted, meaning the tensor is of $N\ x\ N\ x\ [3 \cdot (4 + 1 + C)]$ for the 4 bounding box offsets, 1 objectness predictions, and C class predictions. To have the same dimensions during prediction, the feature map is always up-sampled $2x$ to stay at the same dimension as the largest feature map from the network (total of 3 scales). The k-means method described in 4.3.3 is still used, but the 9 clusters are evenly divided across the scales. During testing on the COCO dataset the clusters were set to $(10\ x\ 13), (16\ x\ 30), (33\ x\ 23), (30\ x\ 61), (62\ x\ 45), (59\ x\ 119), (116\ x\ 90), (156\ x\ 198), (373\ x\ 326)$.

4.3.5 YOLO v4

A study by Bochkovskiy et al. [9] offered multiple improvements to the YOLO architecture. This improved architecture was named YOLOv4. The improvements were categorized by Bochkovskiy et al. as: **Bag of Freebies** (*BoF*), and **Bag of Specials** (*BoS*) [9].

Bag of freebies consists of a collection of architectural improvements that do not affect inference speed. These methods only change the training method or are improvements that increase training cost. Bag of Specials are techniques that increase inference cost but can significantly improve the model's accuracy. This can be done by mechanisms such as enlarging receptive field, introducing attention [60] mechanisms, or strengthening the feature integration capability of the model. In YOLOv4, the following BoFs and BoSs are introduced:

Bag of Freebies used for backbone network:

- **CutMix**[67], which is a data augmentation method that combines two training samples by cutting out an object from one image and placing it close to other objects in another image.

- **Mosaic** is a new data augmentation proposed by Bochovskiy et al. in [9]. It mixes 4 different training images, thus similarly providing 4 different contexts as CutMix. Mosaic, however, also adds a calculation of batch normalization [27] statistics from 4 different images on each layer, significantly reducing the need for a large mini-batch size.

- **DropBlock regularization**[17], which is a Dropout technique designed for convolutional neural networks. Instead of dropping random weights, it will drop correlated parts of the input, forcing the detector to look elsewhere in the image for features.

- **class label smoothing** [69], which applies a weighted average between the uniform distribution and the hard label on the classification layer in classification tasks to prevent over-fitting.

Bag of freebies used for detector:

- **Complete IoU-loss**(*CIoU-loss*)[71], is a loss function for the bounding box (*BBox*) regression problem that considers the overlapping area, the distance between center points, and the aspect ratio of the predictions. CIoU can achieve better convergence speed and accuracy on the BBox regression problem.

- **CmBN** is a batch normalization technique that only collects statistics between the mini-batches within a single batch.

- **DropBlock regularization**[17].

- **Mosaic data augmentation**[9].
- **Self-adversarial training** (*SAT)*[45] is a data augmentation technique that has 2 forward backward stages. The 1st stage alters the original image instead of the network weights, which creates the deception that there is no object on the image. In the 2nd stage the network is trained on this modified image.
- **Eliminate grid sensitivity**. Grid sensitivity was a problem that arose in YOLOv2 in equation 4.15. Since c_x and c_y are always whole numbers, extremely high t_x and t_y absolute values are required for the b_x, b_y values to approach c_x and $c_y + 1$ values. This is solved by multiplying the accompanying sigmoid function with a factor larger than 1, eliminating the effect on the grid when an object is undetectable.
- **Using multiple anchors for a single ground truth** and keeping the ones passing the test IoU(truth, anchor) > IoU-threshold.
- **Cosine annealing scheduler**[37], which alters the learning rate dynamically in a sinusoidal fashion during training.
- **Random training shapes** which automatically increases the mini-batch size during small resolution training.

Bag of Specials (BoS) used for backbone network:

- **Mish activation**[38], as explained in section 2.6, helps to solve the vanishing gradient problem prevalent in YOLOv3[48].
- **Cross-stage partial connections (CSP)**[61] is a modification added to backbone networks which partitions the feature map of the base layer into two parts and then merges them through a cross-stage hierarchy. Each previous layer will undergo a transition layer and be concatenated to the output of the next layer, which increases the difference of gradient combination. In turn, accuracy, with very little cost added.

Bag of Specials (BoS) used for detector:

- **Mish activation**[38]
- **SPP-block** [22] is a pooling layer that removes the fixed-size constraint of a network. It pools the features and generates a fixed-length output for the prediction head.

- **SAM-block**, the spatial attention map proposed by Woo et al.[65], sequentially infers attention in the spatial dimension from an input feature map, to refine the next feature.

- **PAN path-aggregation block**[35] boosts information flow in from feature proposals by enhancing the hierarchy with accurate localization signals in lower layers by bottom-up path augmentation. This shortens the information path between the lower layers and the topmost feature. Additionally, adaptive feature pooling [35] is employed, which links feature grid and all feature levels to make useful information in each feature level propagate directly to following proposal sub-networks.

- **DIoU-NMS** (Distance IoU-loss with non-maximum suppression) [71] is the same as CIoU-loss without considering the aspect ratio. Non-maximum suppression is used to suppress the extra bounding boxes for the same object.

The main architecture of the YOLOv4 network then consists of a CSP-modified Darknet-53 backbone[61], with SPP[22] and PANet path-aggregation blocks[35] before an anchor based YOLOv3[48] prediction head. All these improvements to YOLOv3 increased the average precision on the MS COCO [33] dataset by over 10%, while inference speed remained around the same.

We will now discuss the features of the EfficientDet model family.

4.3.6 EfficientDet

Utilizing previous works on EfficientNet, which is described in section 4.1.3, Tan et al. proposed a new family of object detectors called EfficientDet [56]. In this section, we will explain the main features of this detector family.

EfficientDet attempts to tackle the following two challenges in object detection:

1. Efficient multi-scale feature fusion

2. Model scaling for object detection

We will start by explaining the solution to efficient multi-scale feature fusion proposed by Tan et al. [56].

To effectively represent and process the multi-scale feature representations output by the backbone remains a difficulty in object detection. Previous works often directly execute predictions based on the pyramidal feature hierarchy extracted from the backbone networks [11]. To improve efficiency compared to direct predictions on the feature maps, feature pyramid networks *(FPN)*, as proposed by Lin et al.[34], introduce top-down pathways to combine the multi-scale features as shown in figure 4.8(a). PANet [63], following this idea, introduced an extra bottom-up path aggregation network on top of FPN, as seen in figure 4.8(b). Ghiasi et al. [18] then proposed using neural architecture search [15] to automatically design a feature network topology (*NAS-FPN*).

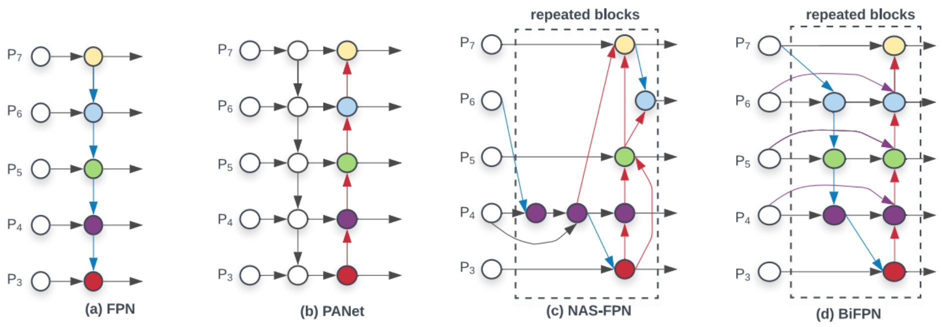


Figure 4.8: Comparison between multi-scale feature fusion methods. (a) FPN [34] shows the top-down pathway to fuse multi-scale features from level 3 to 7 $P_3 - P_7$; (b) PANet [35] has an additional bottom-up pathway; (c) NAS-FPN [18] use ; (d) BiFPN [56]. (Figure from Tan et al. [56].)

Tan et al. [56] attempt to optimize the ideas from FPN, PANet, and NAS-FPN in a weighted bi-directional feature pyramid network (*BiFPN*), which enables information flow in both the top-down and bottom-up directions. Compared with PANet, BiFPN also optimize these bottom-up connections by removing nodes with a single input edge [56]. BiFPN also uses each bi-directional path as a one-feature network layer, which repeats multiple times for higher-level feature fusion, similarily to NAS-FPN [18]; however, as neural architecture search has large hardware requirements, weighted feature fusion was introduced instead. Weighted feature fusion utilizes fast normalized fusion which allows the network to learn an additional weight; specifically, as to how important each resolution is in the output feature map. A visualization of the operation can be seen in figure 4.8(d). Fast normalized fusion is found as

$$\text{Output} = \sum_i \frac{w_i}{\epsilon + \sum_i w_j} \cdot I_i \tag{4.17}$$

where $w_j \geq 0$ is ensured by applying $ReLu$ after each w_i, and $\epsilon = 0.0001$ to avoid numerical instability. The output will, similarly to the SoftMax operation, be a value between 0 and 1.

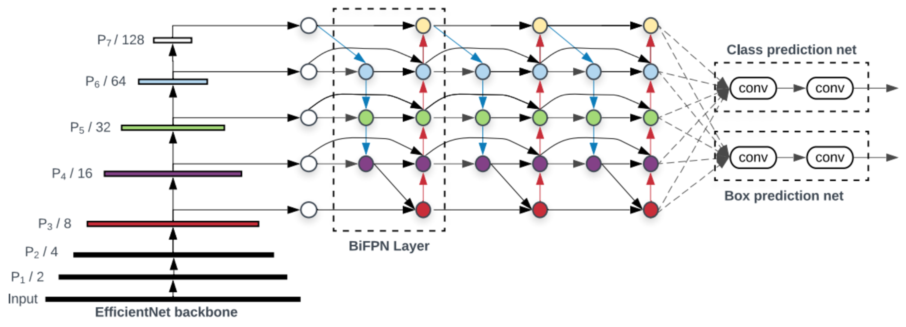


Figure 4.9: EfficientDet architecture - Utilises EfficientNet [54] as the backbone network, BiFPN as the feature network, and a shared class/box prediction network. BiFPN and class/box layers are repeated multiple times based on resource constraints shown in table 4.4. Figure by Tan et al. [56]

In addition to multi-scale feature fusion, Tan et al. [56] utilizes the idea of compound scaling, as explained in section 4.1.3, to jointly scale all dimensions of the network depending on resource constrictions. We let $\phi \in \mathcal{N}$ be the scaling factor.

BiFPN depth D_{bifpn} (number of layers) is scaled linearly, as depth needs to be a small integer. The BiFPN width W_bifpn (number of channels) was scaled exponentially, where the scaling factor was found using grid search.

$$W_{bifpn} = 64 \cdot (1.35^{\phi}), \quad D_{bifpn} = 3 + \phi \tag{4.18}$$

For the box- and class-prediction net, the width is fixed to be the same as BiFPN $W_{pred} = W_{bifpn}$, and depth is increased linearly using

$$D_{box} = D_{class} = 3 + \lfloor \phi / 3 \rfloor. \tag{4.19}$$

For the input image resolution R_{input}, the input resolution has to be dividable by $2^7 = 128$, so the resolution is also increased linearly by

$$R_{input} = 512 + \phi \cdot 128. \tag{4.20}$$

A pre-trained EfficientNet backbone uses the same scaling as explained in section 4.1.3.

Figure 4.9 shows the overall architecture of EfficientDet. An ImageNet pre-trained EfficientNet serves as a feature network, takes features at multiple resolutions into the BiFPN layers, which repeatedly (D_{bifpn} times) applies top-down and bottom-up bidirectional feature fusion. These fused features are fed to a class and box network, which uses the same weights repeated depending on D_{box} to output bounding box predictions and class.

In our own experiments in section 6.2 we will be utilizing EfficientDet with $\phi = 1$ (*EfficientDet D1*) which can be seen in table 4.4.

	Input Size	Backbone Network	BiFPN		Box/class
	R_{input}		#Channels W_{bifpn}	#layers D_{bifpn}	#layers D_class
D0	512	B0	64	3	3
D1	640	B1	88	4	3
D2	768	B2	112	5	3
D3	896	B3	160	6	4
D4	1024	B4	224	7	4
D5	1280	B5	288	7	4
D6	1280	B6	384	8	5
D7	1536	B7	384	8	5

Table 4.4: Scaling configurations for EfficientDet. ϕ is the compound coefficient that controls all the scaling dimensions according to equations 4.18, 4.19, 4.20 respectively. The backbone networks is EfficientNet B0-B7 as explained in section 4.1.3.

5 Methodology

In this chapter, we first show the key features of our data set and the pre-processing performed, then demonstrate how the object detection architectures discussed in section 4 can be used to detect the porpoises in the data set.

5.1 Data Set

5.1.1 Original Data Set

The original data provided consists of 23 compilation videos, whereas 18 videos are of resolution 1980 by 1080, and 5 videos are 1280 by 720. An expert labeled samples from two videos, one 1980 by 1080 and 1280 by 720. In addition to the video files, we have access to the log files of the imaging drone containing flight information. This includes the GPS coordinates and height of the drone every 100ms; however, this data was not used for this thesis. The videos were collected by the University of South Denmark and labeled by an expert at the Norwegian Polar Institute (*NPI*).

The two videos were sampled into 11964 labeled images and split into training, evaluation, and test data in a 60/25/15% split. This split was done randomly, which might cause some issues in assessing the models; since the images are sampled from a video, the test and validation data might contain frames very close in time-stamp to the training data, which might risk over-fitting the model.

When we started doing experiments, we noticed some issues in the labeling. In the original project proposal, the idea was to have a 2-class object detector separating porpoises and mother-calf. The mother-calf is recognized as two porpoises swimming very close, often with one right underneath the other. Although consistent within the marine sciences field, these labels are inconsistent for typical object detectors.

We also noticed that quite a few porpoises were not labeled. We believe the missing labels were from issues with the exporter app written to convert the original Matlab labels to Comma separated values (*CSV*).

Another observation from the original dataset is that the bounding boxes are not very precise and contain unnecessary background information. For this reason, the AP score, especially for the larger IoU values, would be affected. Since the labels have noise, the validation metrics could contain noise.

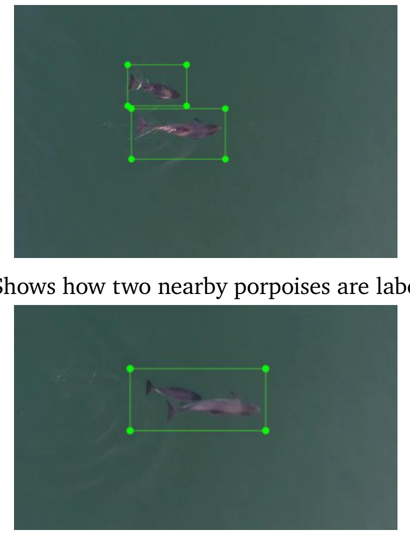
5.1.2 Pre-processing

To fix the issues mentioned in section 5.1 we re-labelled parts of the data-set using LabelImg [14] to add the missing porpoises to the fold. In total, we re-labeled 7327 out of the original 11964 images. We also tuned most bounding boxes to be more precise around the object.

To tune the mother-calf label, we had three choices: 1) Choose criteria to decide how close two porpoises should be for it to be labeled as a mother-calf and keep the second class. 2) Attempt to label each separate porpoise when they are overlapping. This proved difficult due to object detection algorithms utilizing Non-maximum Suppression (NMS) [8] to prune overlapping bounding boxes. 3) Choose criteria for the amount of overlap for us to label two porpoises as a single porpoise, and make this a 1-class problem where we only detect porpoises.

We decided to try 1), and we refined the labels so every time two porpoises touch, we label them as a mother-calf. We agreed on this threshold for consistency in the labeling, as well as to not cause any issues with the IoU thresholds in NMS. Figure 5.1 shows an example of this.

Initial experiments utilizing these labels yielded poor results, as neither YOLO nor Faster R-CNN was able to separate the two classes, leading to low precision. This led us to the decision to combine the mother-calf label and porpoise label into a single category.


(a) Shows how two nearby porpoises are labeled.

(b) Shows how two touching porpoises are labeled.

Figure 5.1: Shows how labeling was handled with nearby porpoises.

We split the data set into training/evaluation/test data. We avoided random sampling in the split since the labeled images are subsequently sampled from a video, meaning images can be very similar. This could lead to the detector having seen an almost identical shot in all three data set partitions. Instead, we choose images from different video clips as training/test/evaluation data.

5.2 Model training frameworks

5.2.1 TensorFlow models

The faster R-CNN and EfficientDet D1 models were implemented using TensorFlow's Model Garden API [66]. Due to limited amounts of memory on the GPUs used for training, as well as the reported inference speeds of the models [66] the choice of models fell the on the two following models:

1. Faster R-CNN [50] with 640 *x* 640 resolution and a ResNet-101 backbone.

2. EfficientDet D1 [56], with 640 *x* 640 resolution and the EfficientNet D1 backbone.

The Faster R-CNN model chosen reportedly achieves a 31.8 COCO mAP score with a speed of 55ms per image, whereas EfficientDet D1 achieves a 38.4 COCO mAP with a speed of 54ms per image [66]. There are models tested to be more accurate, but due to hardware limitations and a requirement of decent inference speeds, these were the considered models.

Another consideration was keeping the resolution of the networks the same to reduce the number of parameters to consider when comparing models. This allows the comparison between the models without accounting for the resolution of the training images. All three models used in this thesis were trained using a pre-trained backbone trained on ImageNet.

5.2.2 Faster-RCNN

The Faster-RCNN network was trained on the Springfield GPU cluster provided by the Machine Learning group at the University of Tromsø. We trained the model with a batch size of 4, for a total of 5000 steps[1], with 1000 warm-up steps. The validation loss converged as seen in figure 5.2. For training, the IoU-threshold for the second stage post-processing was lowered to 0.3 to match YOLOv4[9] and get comparable results. Dropout was also enabled with a value of 0.5. The rest of the hyper-parameters were left as the default values, as specified in section 8.2 of the appendix.

The weights were saved every 1000 steps, and from figure 5.2, the weights from step 3,000 were chosen as the weights in the final model, as here the evaluation loss converges. Any further training could lead to over-fitting and poor regularization. As we can see from figure 5.3, step 3000 is also when the precision and recall of the model converges, which further confirms stopping training.

Once training was completed, the model was frozen and exported as a TensorFlow .pb model for inference on the test data.

5.2.3 EfficientDet D1

The EfficientDet D1 model was trained using the same TensorFlow Model Zoo API [66], and was trained using the exact same method as described in section 5.2.2, except for different hyper-parameters.

A bug in the API led to the log files of the loss functions exceeding 100GB of

1. A common notation for training object detections models is number of steps instead of epochs. Steps is the number of images ran through the training process. In other words: 1 epoch is when number of steps is equal to the number of training images.

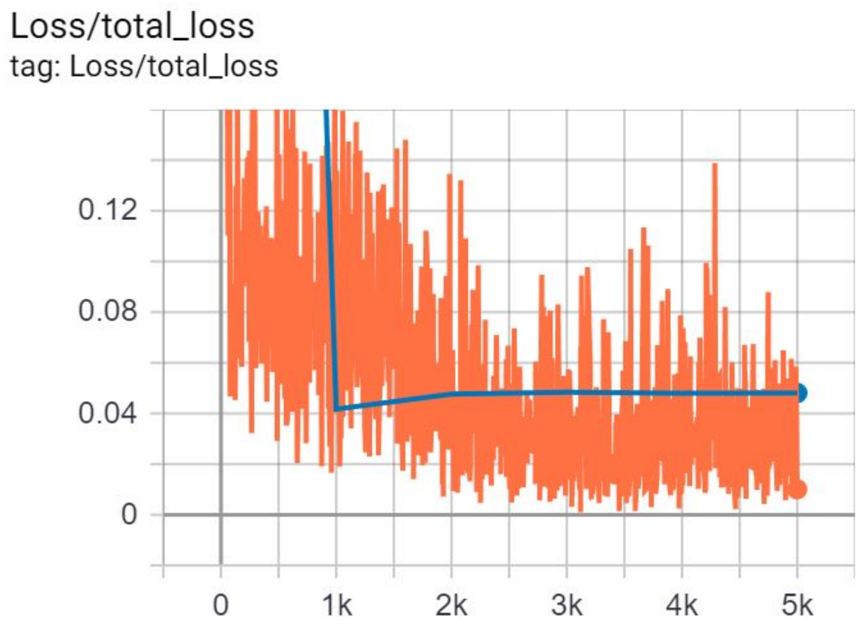
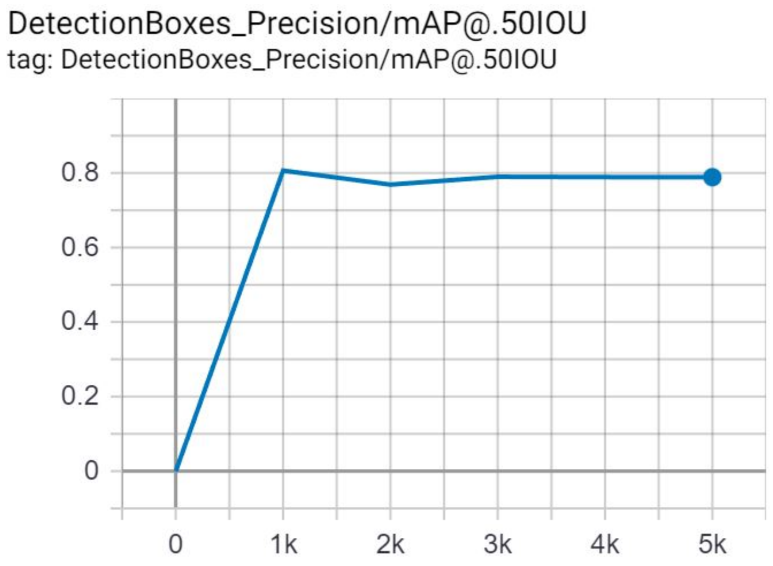


Figure 5.2: Total loss (equation 4.9) at each training step of faster R-CNN. Orange is here the training loss, while blue is the evaluation loss calculated every 1000 steps.

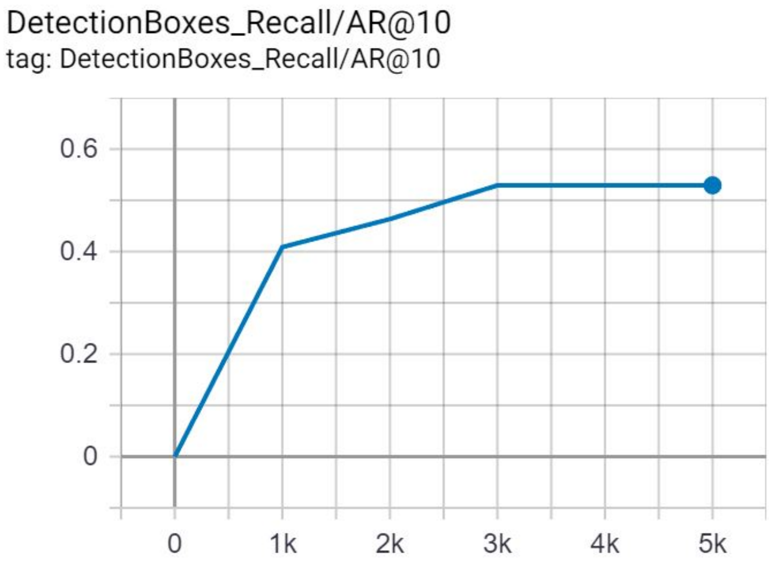
data and crashed Tensorboard. For this reason, the saving of information from the loss functions was disabled, and we used AP@50and AR$^{max=10}$ to decide on when to stop training. Due to the behavior of Faster R-CNN, where the loss and AP/AR values seem to converge around the same step, we will assume this to be ok for testing purposes.

Another issue was exploding gradients during training. The default model was configured to be trained on hardware far exceeding the available hardware. This led to a decrease in batch size from 256 to 4 and caused issues with exploding gradients. This issue was fixed by significantly reducing the learning rate, leading to the model training very slowly compared to Faster R-CNN. The problems mentioned with the source code could lead to the EfficientDet D1 results not being optimal during inference.

From figure 5.4 we see that both the precision and recall values converge around step 40000. We therefore choose to stop training here and freeze the graph to use for inference.



(a) AP@50

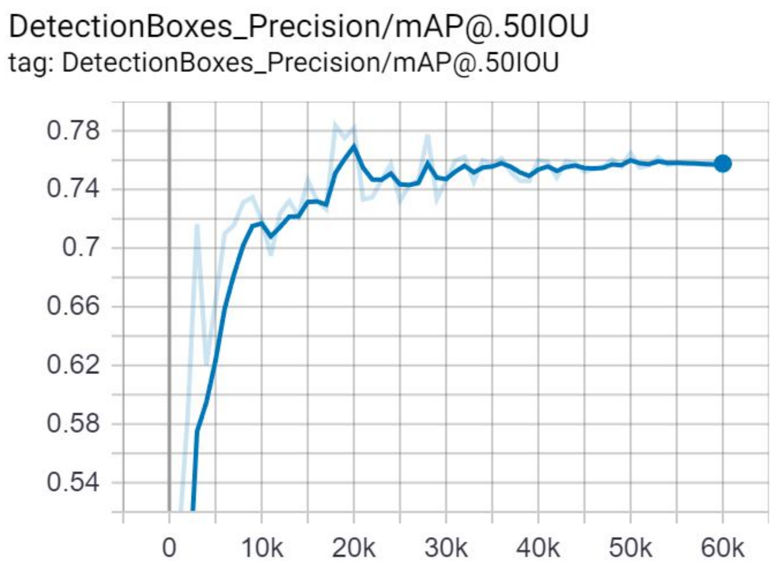


(b) AR$^{max=10}$

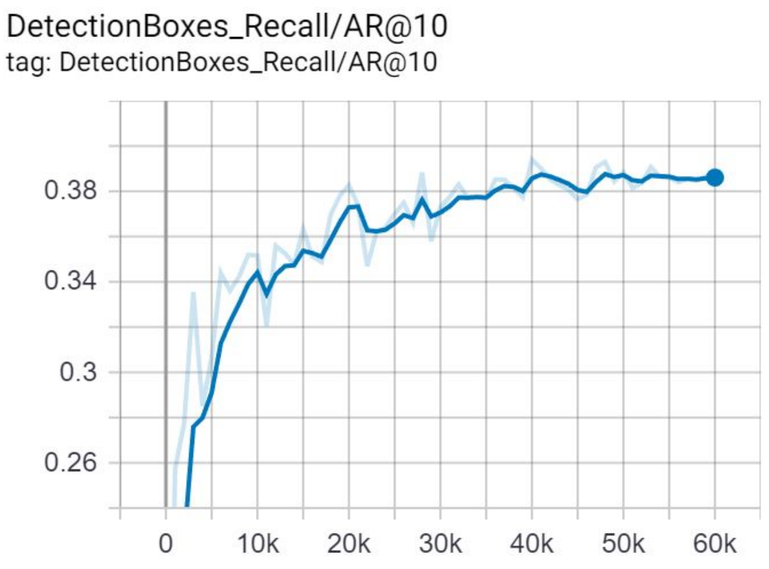
Figure 5.3: AP@50 (**a**) and AR$^{max=10}$(**b**) metric on evaluation data over number of steps during training for the Faster R-CNN model. This was calculated every 1000 steps.

5.2.4 Darknet

Darknet is a C++ library by Bochkovskiy et al.[9] developed to train and run the various YOLOv4 [9] models.



(a) AP@50



(b) AR$^{max=10}$

Figure 5.4: AP@50 (**a**) and AR$^{max=10}$(**b**) metric on evaluation data over number of steps during training for the EfficientDet D1 model. This was calculated every 1000 steps.

It was trained with a batch size of 64, with mini-batches of size 4 (subdivision 16). Network resolution was set to 640 *x* 640. All the techniques mentioned in 4.3.5 were left enabled. The number of filters in the last convolutional layer before each YOLO layer was also changed to $(C+5)\cdot 3$ to match the number of classes $C = 1$ in the YOLO prediction layers. The changes in the configuration can be seen in appendix 8.3 The weights from step 6800 were the final trained weights for this detector, as it had the best AP@50 on the validation data.

Due to issues getting Darknet to run on local hardware, this model was trained on Google Colab. However, due to time constraints on runtimes when using Colab and the training plot resetting when training was restarted, the entire training loss was not saved, as seen in figure 5.5. However, this Darknet model automatically saves the weights where it achieves the highest AP@50 score. In our case, this was at step 6800.

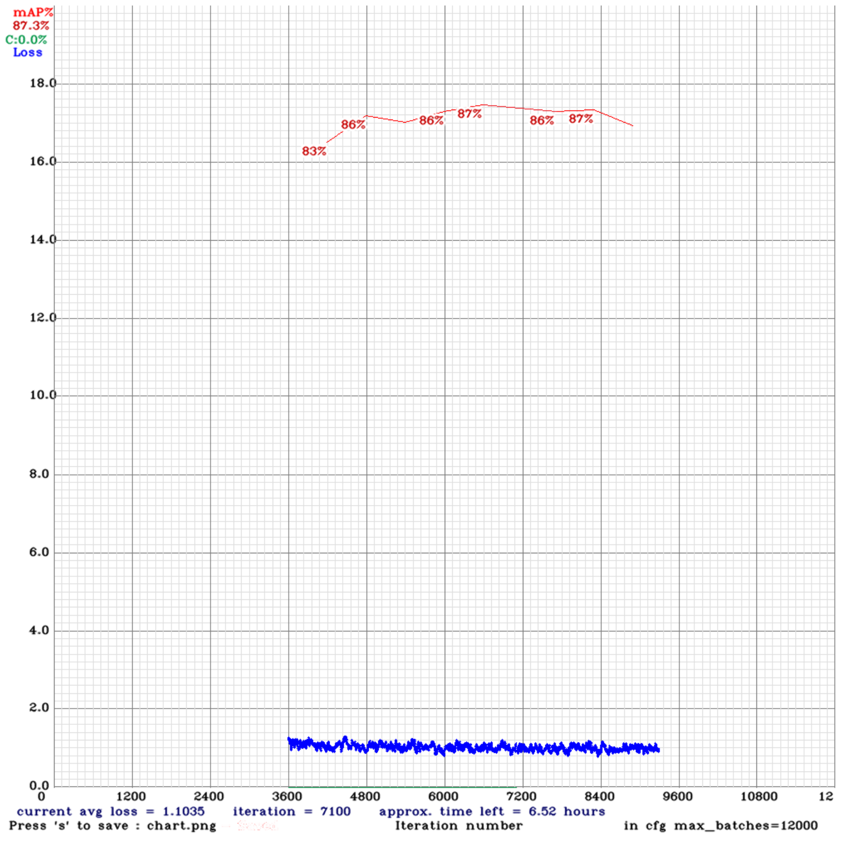


Figure 5.5: Training loss with validation mAP@50 on YOLOv4. Some data is missing due to the chart getting deleted after resuming training on Google Colab.

This model was then exported as a Tensorflow model, using theAIGuy's open source code [57] to be available in the same Tensorflow framework as used for Faster R-CNN and EfficientDet D1.

/6

Results & Discussion

In this chapter, we will, in section 6.1 discuss the experimental setup. Section 6.2 shows our results from the three object detection models on our test data set and discusses these results. Finally, in section 6.3 we show the key features of the framework developed for the NPI.

6.1 Experimental Setup

This section will explain which evaluation metrics we weigh and the hyper-parameters used.

6.1.1 Evaluation metrics

To evaluate the three models quantitatively, we need a measurable metric that best describes the end goal of our application. Previously, the most common metrics for comparing object detectors used to be the AP@50 metric, included in the PASCAL VOC challenge [16]. This metric, as explained in section 3, is **AUC**(*Area Under Curve)* of the interpolated precision-recall curve, which incorporates both Recall and precision into one metric, without favoring either, at IoU threshold 0.5.

This has now mostly been replaced by the COCO detection metrics [33], where the main metric is the interpolated average precision AP[:.5,:50,:95] (*AP*) as explained in chapter 3.3. The COCO metrics heavily favor localization [44] compared to the *AP*@50 metric. In addition, the COCO challenge introduced size-dependent metrics, which filter the AP and AR values on the size of the ground-truth bounding box, allowing comparison on different scales.

For this thesis, our primary goal is not to miss any detections, and exact bounding boxes are not our primary goal. For this reason, we will favor AP@50 as the primary metric while also favoring Recall with an IOU threshold of 0.5. However, to truly analyze the performance of the models, we will also utilize the COCO detection metrics and use the COCO bounding box size threshold on other metrics like Recall and precision. This is to do error analysis without including the IoU interpolation of COCO that has the significant localization component. In short, we use the following metrics:

1. **AP@50 (PASCAL VOC)** Average precision and average recall of all classes with at least 50% IoU overlap between ground-truth and prediction bounding boxes [16].

2. **Precision/Recall (small / medium / Large):** Precision and recall with the same IoU threshold, but also thresholded on bounding box area measured in pixels. In this case we let the size thresholds be the same as in the COCO metrics [33]: [Small $> 32^2$, $32^2 \leq$ Medium $\leq 96^2$, Large $> 96^2$].

3. COCO detection metrics (see section 3.3).

One parameter that could be tweaked is the IoU threshold for the AP@50 and precision/recall metrics. Lowering the IoU threshold reduces the demand for precision in the bounding box placements, which could have been helpful for this problem. However, for consistency and for allowing comparisons of results with previous works, we let this threshold stay on the industry standard IoU Threshold= 0.5.

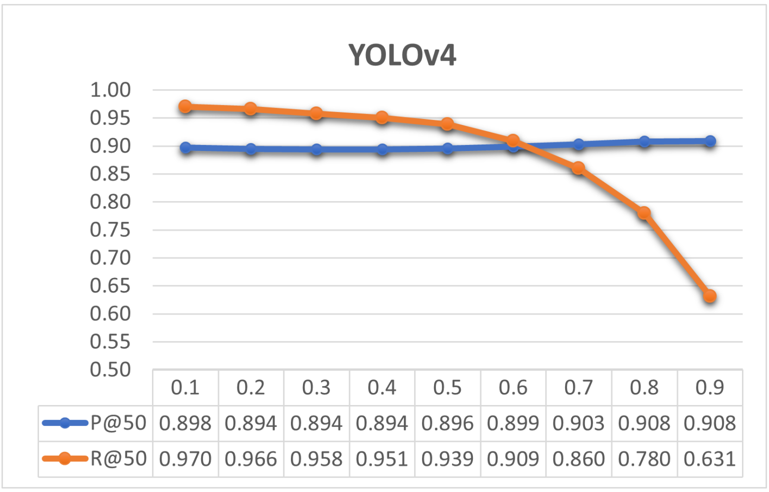
Another critical parameter when viewing results is the confidence threshold.

6.1.2 Confidence Threshold τ

Since the models have very different structures, the confidence threshold τ could massively change the result in evaluation depending on the model. This parameter filters out the detections with a lower confidence score than τ. Depending on the model, this parameter could tweak both the precision and recall of the model, whereas intuitively: increasing τ leads to increasing precision and lowering Recall while decreasing τ leads to decreasing precision and achieving higher Recall.

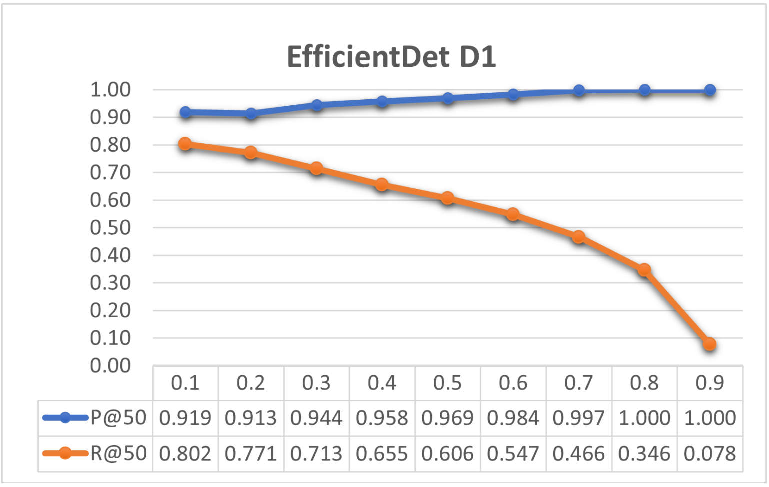
To find the best confidence threshold τ we evaluate all 3 models using $\tau = [0.1, 0.2, \ldots, 0.9]$. The results can be seen in figure 6.1. From this figure, we can see the effect mentioned in section 6.1; for both EfficientDet D1 (figure 6.1b) and YOLOv4 (figure 6.1a), the precision increases and recall decreases as the confidence threshold increases since one of the *main* goals of this model is to minimize missed detections, which leads us to set the confidence threshold by $\tau_{\text{EffDet}} tau_{\text{YOLO}} = 0.1$ for both the YOLOv4 and EfficientDet D1 models. This maximizes the Recall without too high of a penalty in precision.

Faster-RCNN behaves a bit differently compared to EfficientDet and YOLO. The confidence numbers are almost binary, with scores around 0.98 for valid detections, or 0.01 for the other proposed bounding boxes. This leads to the effect seen in figure 6.1c where both the precision and Recall are horizontal when plotted against confidence. Here the choice of threshold does not seem to matter too much, so we set it to the same as YOLO and EfficientDet $\tau_{\text{RCNN}} = 0.1$ for consistency.



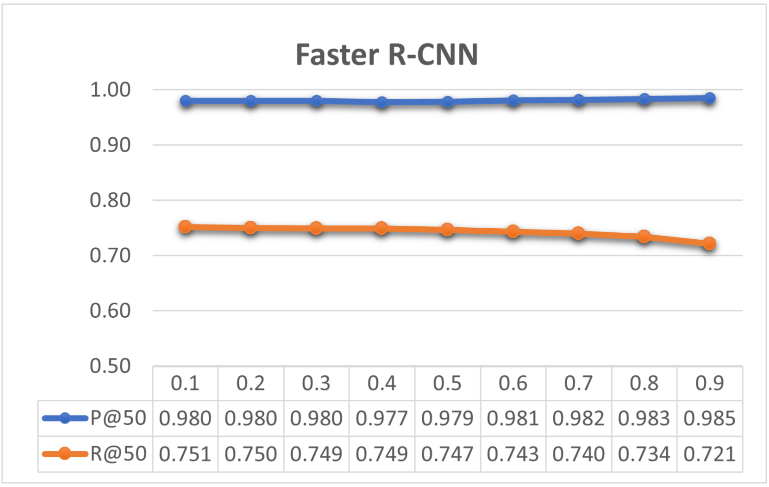
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
P@50	0.898	0.894	0.894	0.894	0.896	0.899	0.903	0.908	0.908
R@50	0.970	0.966	0.958	0.951	0.939	0.909	0.860	0.780	0.631

(a) YOLOv4



	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
P@50	0.919	0.913	0.944	0.958	0.969	0.984	0.997	1.000	1.000
R@50	0.802	0.771	0.713	0.655	0.606	0.547	0.466	0.346	0.078

(b) EfficientDet D1



	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
P@50	0.980	0.980	0.980	0.977	0.979	0.981	0.982	0.983	0.985
R@50	0.751	0.750	0.749	0.749	0.747	0.743	0.740	0.734	0.721

(c) Faster R-CNN

Figure 6.1: Confidence threshold testing for all models. Shows precision@50 (blue) and recall@50 (orange) over a range of confidence thresholds τ as mentioned in section 6.1.

6.2 Experimental Results & Discussion

In this section, we will show the different results yielded from the two detectors when run on the validation data. All results are generated using full-resolution test images.

6.2.1 Speed/accuracy trade-off

Model inference speed can, in some cases, affect choice of model. The processing time of images depends on hardware, model complexity, and image resolution [54]. Generally, more complex models achieve better results at the cost of long inference time and training times. Usually, proposal-based detectors like Faster R-CNN [50] tend to be more accurate, while single-shot detectors like YOLO [9] and EfficientDet [56] tend to be faster but less accurate.

When utilizing the chosen model here, we want to reduce the time it takes for marine biologists to perform population surveys. Currently, the NPI does not have any GPU clusters or exceptional hardware to run the largest models, which leads to the choice of models being limited to the faster models. Table 6.1 shows the inference times, frames per second (FPS), and what hardware this was run on. Nvidia's documentation [42] show that the Nvidia GTX 1080ti and Quadro P100 are very comparable in computation speeds, and we therefore assume this did not affect the results since they match the expected results of YOLOv4 being the fastest and EfficientDet D1 being faster than Faster R-CNN[66] [9] [56]. The results in table 6.1 show that YOLOv4 with 48.1 FPS is much faster than both Faster R-CNN with 48.1 FPS and EfficientDet D1 with 19.2 FPS on the test data set. However, these results also indicate that all three models are fast enough to be used for this project.

Model	Inference Speed (s)	FPS	GPU
Faster R-CNN	309.1s	7.0	Nvidia GTX 1080ti
EfficientDet D1	112.4s	19.2	Nvidia GTX 1080ti
YOLOv4	45.0s	48.1	Nvidia Quadro P100

Table 6.1: Inference speed and Frames per Second (FPS) of the three models on the test data (2164 images)

6.2.2 Model Performance

The studied models were Faster R-CNN, EfficientDet D1, and YOLOv4. Table 6.3 and 6.2 presents the main results from the experiments on these models. The experiments were run on the test data, which, as explained in section 5.1.2,

	YOLOv4	EfficientDet D1	Faster R-CNN
AP (COCO)	0.304	0.278	0.313
AP@50	0.766	0.689	0.676
AP@75	0.178	0.140	0.252
APsmall	0.141	0	0.051
APmedium	0.214	0.224	0.352
APlarge	0.349	0.303	0.298
$AR^{max=1}$	0.215	0.188	0.225
$AR^{max=10}$	0.423	0.328	0.422
ARsmall	0.15	0	0.05
ARmedium	0.331	0.273	0.473
ARlarge	0.469	0.356	0.397

Table 6.2: COCO metrics for EfficientDet D1, Faster R-CNN, and YOLOv4 at confidence threshold $\tau = 0.1$. Calculated using open-source software from Padilla et al. [43]

	YOLOv4	EfficientDet D1	Faster R-CNN
AP@50 (PASCAL VOC)	0.778	0.695	0.686
Precision	0.898	0.919	0.980
Recall	0.970	0.802	0.751
Precision Small	1.000	0.000	1.000
Precision Medium	0.850	0.897	0.971
Precision Large	0.919	0.931	0.987
Recall Small	0.500	0.000	0.250
Recall Medium	0.926	0.860	0.958
Recall Large	0.990	0.778	0.657

Table 6.3: Inference results on test data of the three trained models. All models were evaluated with IoU Threshold = 0.5 and confidence threshold $\tau = 0.1$.

consists of labeled video frames consisting of 100-300 sequential images for a total of 2164 images.

We will first analyze each model separately and then select a model for our task based on the results.

Faster R-CNN: Faster R-CNN achieves a high Precision with 98% however it has a low recall of 75%. Using the size constraints, we can see that it is especially on large objects that this model has sub-par performance. From figures 6.2a and 6.4a, we can also see that the only missed detections in these example images were situations where the porpoises are overlapping. To investigate this, the original labels were analyzed and can be seen in table 6.5. From this we can see that 1007 out of the total 2775 bounding boxes classified as 'large' were originally classified as a mother-calf. This number is 38 out of 1045 bounding boxes for the medium bounding boxes. The results from Faster R-CNN were then analyzed to find the number of FN, FP, and TP for each size, and these numbers can be found in table 6.4. As we can see in table 6.4, the number of true positives and false negatives seem very similar to the respective numbers of Porpoises and Mother-calves in the original data-set. This, along with a qualitative evaluation of the predicted images, allows us to conclude that Faster R-CNN misses close to every overlapping porpoise in the data set while performing well on singular porpoises. For details, please see Figures 6.2a and 6.4a.

Another note is that Faster R-CNN is the only model to detect the seagull in Figure 6.6a. This, combined with the poor detection rate of overlapping porpoises and the very high confidence scores for singular porpoises, could indicate poor regularization of the model. This could be caused by the combination of a lack of data augmentation built into the model and a low variance data set. The images in this data set contain almost no images containing other objects, mostly some complex background that could, for humans, look like objects.

On the other hand, the Faster R-CNN model yields more accurate localization than the other detectors used in this project. Comparing the AP@50 values in table 6.3 with the *AP* values in table 6.2, we can see that Faster R-CNN moves from the worst model to the best model. As the COCO $AP[.50 : .05 : .95]$ value and all the other interpolated values will be heavily favored by well-localizing detectors. We can assume that the bounding boxes output from the Faster R-CNN model is more precise than YOLOv4 and EfficientDet D1.

Faster R-CNN	Total	Small	Medium	Large
True Positives	3096	1	1238	1857
False Positives	62	0	37	25
False Negatives	1025	3	54	968

Table 6.4: Stochastic result analysis of test data analyzed on the Faster R-CNN model.

EfficientDet D1: EfficientDet D1 comparatively yields a better recall than Faster R-CNN. It detects some overlapping porpoises but misses quite a few of them, meaning it has some of the same issues as Faster R-CNN with these situations. This is evident in figure 6.2c, where the bounding box is very imprecise around the porpoises and has a very low confidence score of 17%. We can also see this in figure 6.4c, where it has completely missed the detection when there was another porpoise in the image. This could be due to the fact that a lot of the training images have only one porpoise and that could mean that this version

Label	Any Size	Small	Medium	Large
Porpoise	3138	4	1366	1768
Mother-calf	1045	0	38	1007
Total	4183	4	1404	2775

Table 6.5: Shows original ground-truth test-data information. The mother-calf label was later discarded in favor of porpoise, so 'Total' is the number of porpoises in the test data set.

of the trained model misses actual features pertaining to a porpoise.

Studying the medium false negatives in table 6.6 we can see that even if EfficientDet miss-classified every overlapping porpoise, it still misses some singular porpoises, meaning this model is more 'well-balanced' in that it misses some objects from different original classes compared to Faster R-CNN.

In total, this model misses around 20% of all possible detections while only achieving slightly higher precision than YOLOv4, which leads to a lower $AP@50$ value. We can also see from the COCO metrics in table 6.2 that the $AP[.50 : .05 : .95]$ and AP@75 values compared to the AP@50 value has decreased drastically, indicating imprecise bounding boxes. However, when evaluation the data qualitatively, the bounding boxes on singular porpoises (Figure 6.5c, 6.6c, 6.3c especially seem precise enough for our purposes.

EfficientDet D1	Total	Small	Medium	Large
True Positives	3136	0	1040	2096
False Positives	275	0	119	156
False Negatives	772	4	170	598

Table 6.6: Stochastic result analysis of test data analyzed on the EfficientDet D1 model.

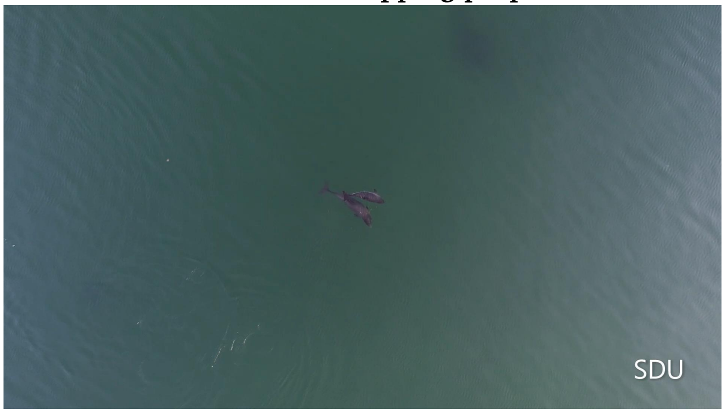
YoloV4: achieves overall better results among the object detectors used in this project. In particular, we can see the average Recall of the YOLOv4 model is much greater than Faster R-CNN and EfficientDet, at the cost of lower precision. As seen in table 6.3, YOLO achieves a recall of 97%. This would mean close to all objects will be detected, and the project goals are possible using this model.

Both EfficientDet and Faster R-CNN were having problems with the overlapping porpoises, however, YOLO v4 has no issues here. This could be due to all the data augmentation techniques described in section 4.3.5. This data set contains a lot of very similar images, and a large amount of data augmentation could have yielded a much better-regularized model.

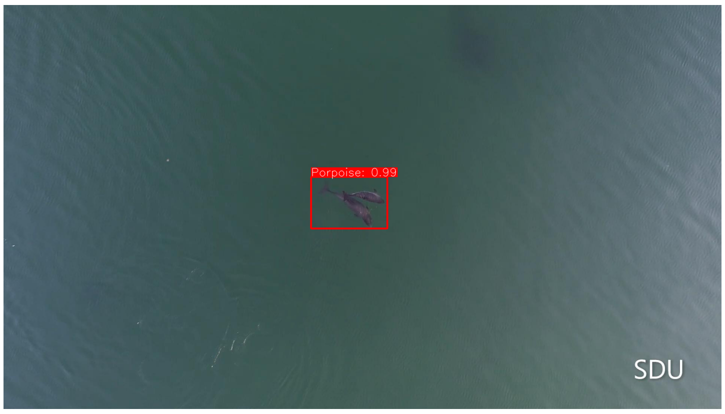
Similarly to EfficientDet, YOLOv4 has worse bounding box precision compared to Faster R-CNN. This is evident by comparing the AP@50 value, where YOLOv4 outperforms the other models, with the much stricter AP@75 and $AP[.50 : .05 : .95]$ values, where Faster R-CNN outperforms YOLOv4. However, as explained in section 6.1.1, a bounding box IOU of 50% is enough for the uses in this project.

Choosing our model: YOLOv4 achieves an average recall score much higher than both EfficientDet D1 and Faster R-CNN. Although the precision is lower than the other two models, YOLOv4 makes up for it by detecting 97% of the porpoises in the data set, and achieving the highest AP@50. All this while also having the fastest inference speed. YOLOv4's confidence scores also seem to resemble probabilities in contrast to Faster R-CNN, as evident in figure 6.1a. This could allow filtering of the predicted porpoises, where all high confidence detections are automatically flagged as a valid detection, and the lower probability images get flagged for manual review by a marine biologist. For these reasons, this model is seemingly the best fit amongst our chosen models to improve efficiency in the work of doing population surveys of porpoises.

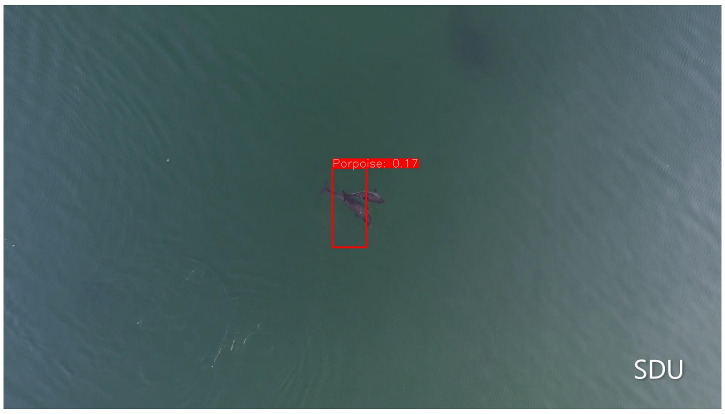
Prediction - Overlapping porpoises



(a) Faster R-CNN



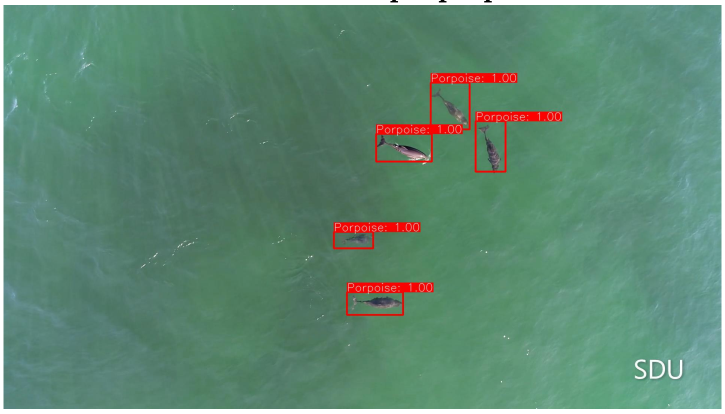
(b) YOLOv4



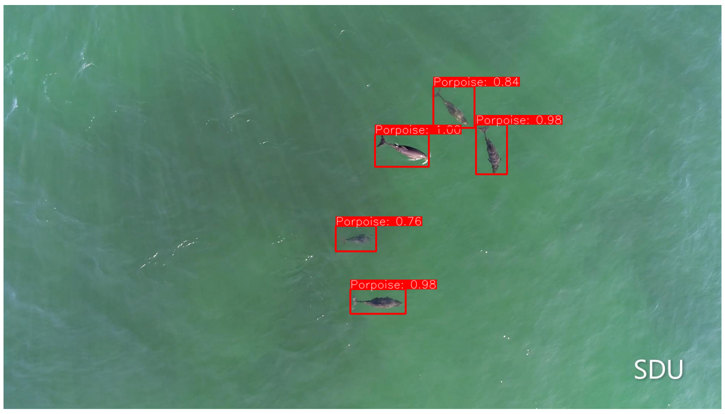
(c) EfficientDet D1

Figure 6.2: Prediction comparison of the three models on a test image with an overlapping porpoise. Here we can see Faster R-CNN missing the object completely, YOLO detecting it accurately with a 99% confidence, and EfficientDet having an in-precise bounding box with a 17% confidence.

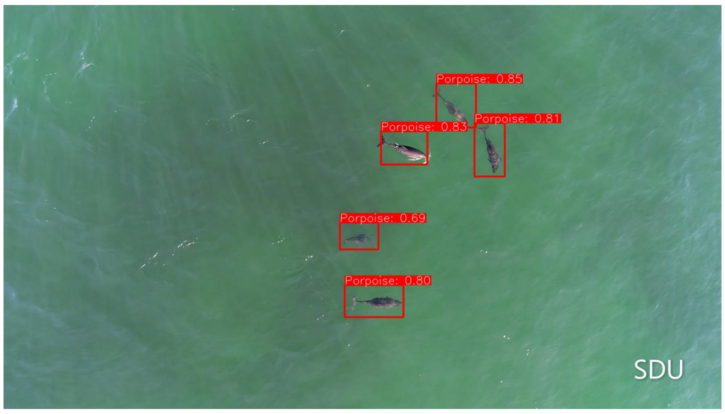
Prediction - Multiple porpoises



(a) Faster-RCNN



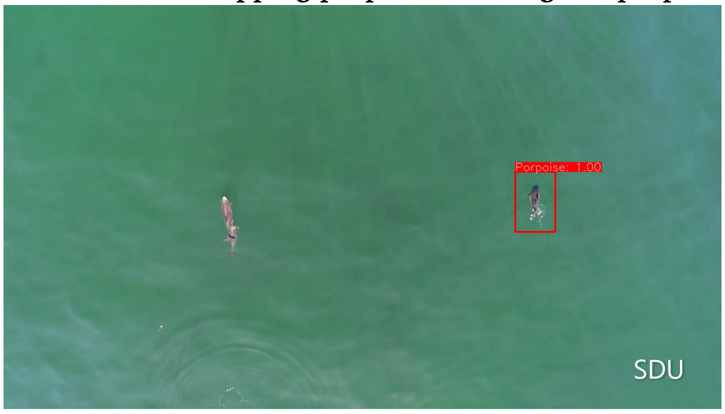
(b) YOLOv4



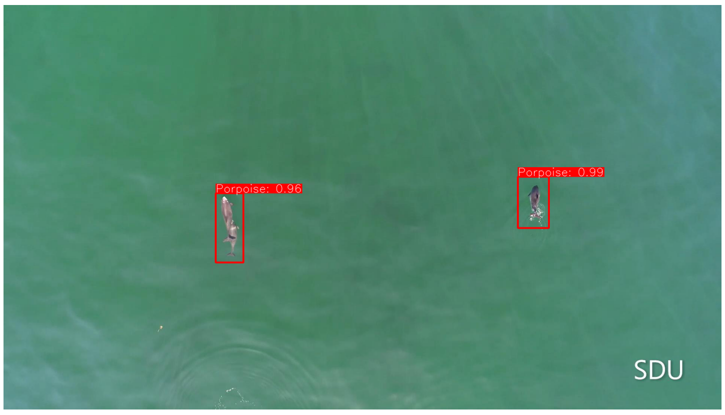
(c) EfficientDet D1

Figure 6.3: Prediction comparison of the three models on a test image with multiple porpoises. Here we can see all three models performing well and detecting accurately.

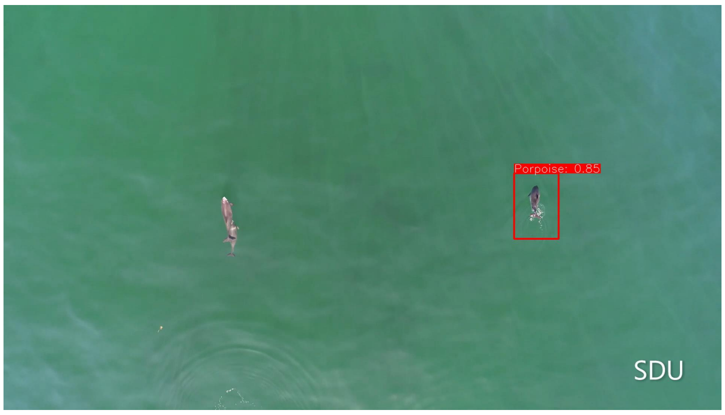
Prediction - Overlapping porpoise and singular porpoise,



(a) Faster-RCNN



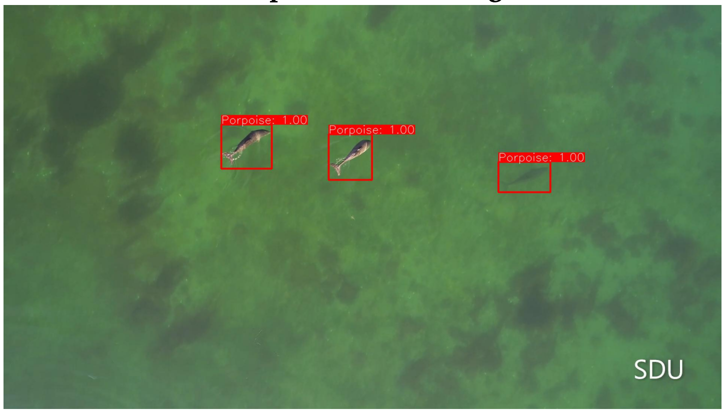
(b) YOLOv4



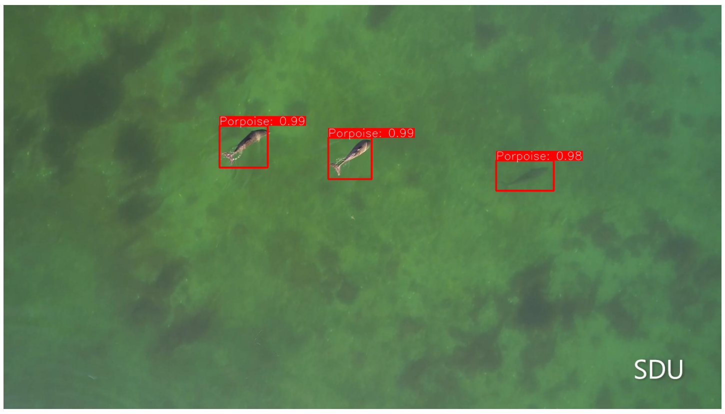
(c) EfficientDet D1

Figure 6.4: Prediction compares the three models on a test image with an overlapping porpoise and another porpoise. Here we can see both Faster R-CNN and EfficientDet D1 missing the overlapping porpoise (left) completely, while YOLO detects it accurately with a 96% confidence. The only porpoise (right) is detected accurately by all three models.

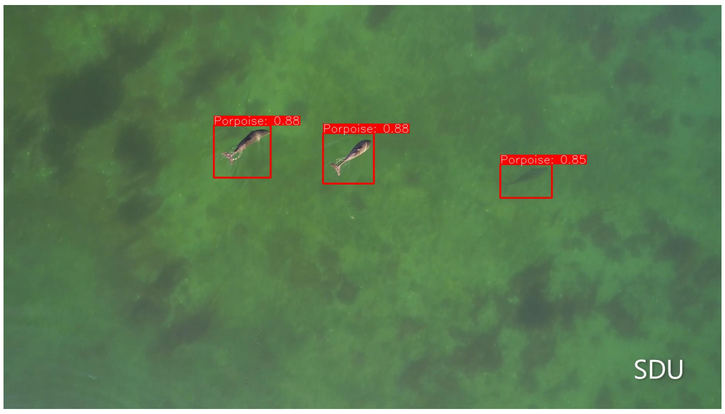
Prediction - Porpoises with background noise



(a) Faster-RCNN

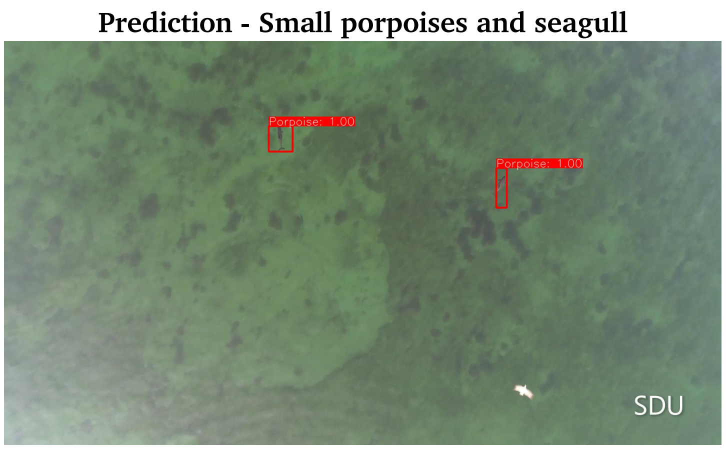


(b) YOLOv4

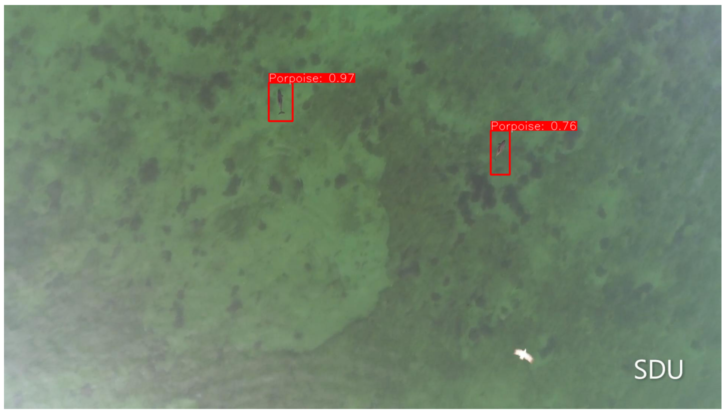


(c) EfficientDet D1

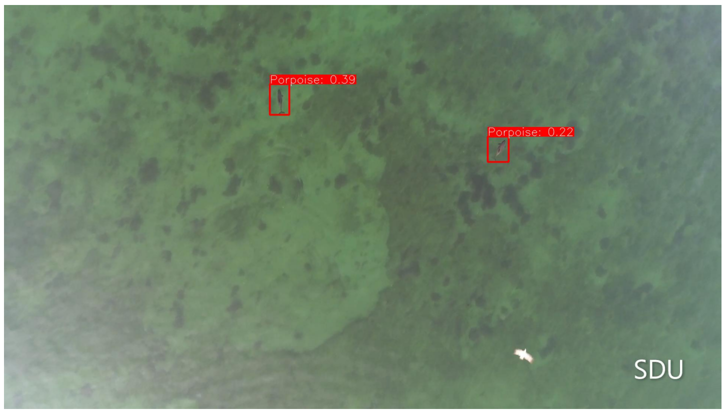
Figure 6.5: Prediction compares the three models on a test image with an overlapping porpoise and another porpoise.



(a) Faster-RCNN



(b) YOLOv4



(c) EfficientDet D1

Figure 6.6: Prediction comparison of the three models on a test image of two small porpoises and a seagull.

6.3 Application Development

To enable this model to be used in a user-friendly way for people with limited programming experience, it was decided to develop an application that automatically uses the model. The Norwegian Polar Institute representative set a few specifications for version 1 (*V1*) of the application:

1. Option to classify both images and videos.
2. A simple user interface.

V1 of the application was developed containing all these as well as:

1. Graphical user interface (GUI).
2. Option to change the IoU threshold for overlapping objects. [1]
3. Option to change confidence threshold for classified objects.
4. Batch processing. Process multiple images/videos without further input from the user.
5. Option to sample videos before classification with a customized interval.

The output for image-classification was the classified images with a bounding box, the classification confidence, and class name. For videos, all classified frames were put assembled in video format (slide-show).

1. For nearby boxes, Non-maximum Suppression (NMS) is applied, this changes how much the boxes overlap there should be before this is applied.

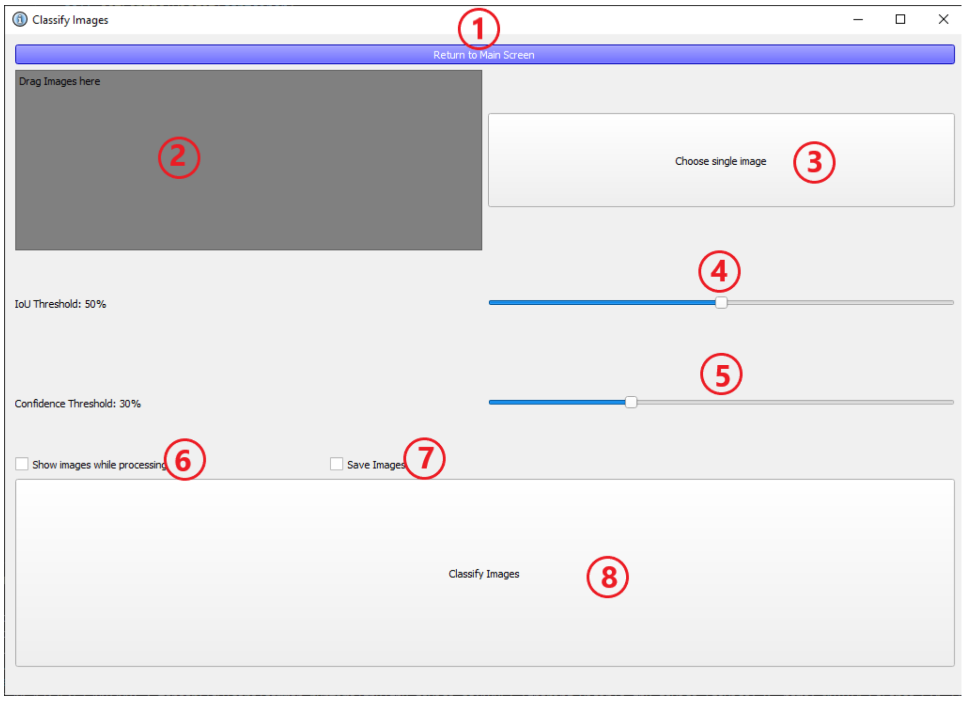
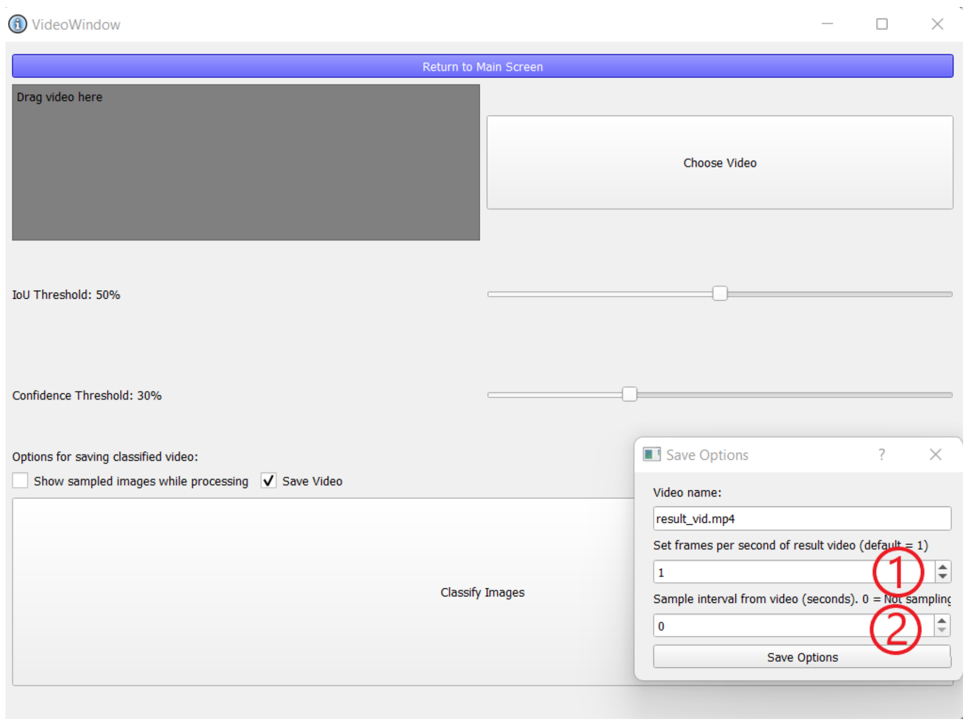


Figure 6.7: GUI developed for NPI.

Figure 6.7 shows how the UI of v1 of the application. The elements marked with a red number are explained as:

1. Return to the main screen. The model stays loaded. Allows classification of both images and videos without reloading the model each time.
2. Drag and drop images to classify to this box.
3. Option to drag and drop: Can choose single images.
4. IoU Threshold: Sensitivity to overlapping objects.
5. Confidence Threshold: Adjust the model's cut-off threshold for the probability score.
6. Option to show the classified images in real-time in a separate window while processing.
7. Option to save the images.
8. Run classification.



IoU Threshold: 50%

Confidence Threshold: 30%

Options for saving classified video:

☐ Show sampled images while processing ✓ Save Video

Video name: result_vid.mp4

Set frames per second of result video (default = 1): 1

Sample interval from video (seconds). 0 = Not samplinc: 0

Figure 6.8: Additional features of UI for classifying video.

The video classification UI as seen in figure 6.8 contains the same elements, with the addition of extra options for saving the video and sampling rate:

1. The result from this classification will be a video. This sets the number of classified images per second shown in a result video.

2. Sets the interval between each frame of the video that should be classified. The default value is zero, which classifies every frame of the video.

V1 was compiled and given to the NPI for testing. They were very impressed by the classification results, however, had some suggestions for possible features:

1. Have an output in CSV format containing the Time-stamp or Image-ID, confidence, and the number of detections.

2. Option to only run on a specified segment of a video.

3. Have the option to sort images/video frames into two different folders. One containing the images with high confidence which can be assumed

to be classified correctly, and one containing images with low confidence detections which will be flagged for manual review.

These features have been implemented, and the application has been completed. This application can also be used as a framework for the NPI. The only change needed with the addition of a new model, even with new classes, would be to exchange the model folder of the program.

7 Conclusion

7.1 Future work

Ideas for future work involving the results of this thesis include:

- Test Faster R-CNN and EfficientDet performance utilizing the same data augmentation techniques used in YOLOv4 and compare results.
- Testing robustness of the detection models, using e.g. methods proposed by Kang et al. [3].
- Test models on an expanded dataset using more species. Here a Few-Shot Object Detection via Feature Reweighting model, as proposed by Kang et al. [29], could be used. This model is said to require less training data to detect novel classes in a data-set.
- Explore explainability of the features using e.g. the Quantus toolkit [4].

Another exciting topic made possible with faster detection algorithms that produce good results could be to include a temporal dimension to the data. Instead of detecting each image one by one, you could transfer the detection information to the next frame, allowing the porpoise not just to be detected, but tracked. This would enable a system to automatically count since porpoises included in sequential frames would be kept track of and counted only once.

7.2 Conclusion

In this thesis, we have studied how three state-of-the-art object detection models could be utilized in the detection of porpoises from drone images. The challenging environment to perform detection makes it difficult for humans and object detectors alike. In this thesis we compare and contrast the performance of three object detectors, namely Faster R-CNN, EfficientDet and YOLOv4.

Through our experiments, we have discovered that YOLOv4 outperforms Faster R-CNN and EfficientDet D1 with detection, where YOLO achieves a recall of 97%, compared to 80% recall with EfficientDet D1 and 75% recall with Faster R-CNN. We also find the average precision $AP@50$ values of YOLOv4 to be greater than EfficientDet D1 and Faster R-CNN. Through both qualitative and quantitative methods we discover that both EfficientDet D1 and Faster R-CNN suffers from poor recall especially when porpoises overlap in the images. In the case of Faster R-CNN it misses nearly all detections when the porpoises overlap, but rarely non-overlapping detections. EfficientDet misses a significant portion of the overlapping detections, but also misses a few of the singular.

Through examination of the COCO detection metrics [33], which favor bounding box accuracy, we also show that Faster R-CNN has more precise bounding boxes than YOLOv4 and EfficientDet D1 by comparing the less strict AP@50 values, with the stricter AP@75 and $AP[.50 : .05 : .95]$ values. However as discussed in previous sections, the exactness of the bounding boxes is not the primary goal of the model we choose. The primary goal is to detect all porpoises in images.

In summary, we found YOLOv4 both more accurate and faster than Faster R-CNN and EfficientDet D1, which led us to utilize this model to develop a porpoise detection framework that the Norwegian Polar Institute may use for their work.

This concludes this thesis.

Links to frameworks developed:

Porpoise detection framework:

https://github.com/SigurdRokenes/Porpoise-detection

Tools used for analysis of data:

https://github.com/SigurdRokenes/inference_tools

Bibliography

[1] Highway Networks, November 2015. URL http://arxiv.org/abs/1505.00387. Number: arXiv:1505.00387 arXiv:1505.00387 [cs].

[2] Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation, August 2018. URL http://arxiv.org/abs/1802.02611. arXiv:1802.02611 [cs] version: 3.

[3] Testing Robustness Against Unforeseen Adversaries, June 2020. URL http://arxiv.org/abs/1908.08016. Number: arXiv:1908.08016 arXiv:1908.08016 [cs, stat].

[4] Quantus: An Explainable AI Toolkit for Responsible Evaluation of Neural Network Explanations, February 2022. URL http://arxiv.org/abs/2202.06861. Number: arXiv:2202.06861 arXiv:2202.06861 [cs].

[5] Model Garden for TensorFlow, May 2022. URL https://github.com/tensorflow/models/blob/5212bb9f5571a4e173a54ff393bfbeed707a52dd/research/object_detection/g3doc/tf2_detection_zoo.md. original-date: 2016-02-05T01:15:20Z.

[6] Shivang Agarwal, Jean Ogier Du Terrail, and Frédéric Jurie. Recent Advances in Object Detection in the Age of Deep Convolutional Neural Networks. *arXiv:1809.03193 [cs]*, August 2019. URL http://arxiv.org/abs/1809.03193. arXiv: 1809.03193.

[7] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, August 2017. doi: 10.1109/ICEngTechnol.2017.8308186.

[8] Matthew B. Blaschko, Juho Kannala, and Esa Rahtu. Non Maximal Suppression in Cascaded Ranking Models. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Stef-

fen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Joni-Kristian Kämäräinen, and Markus Koskela, editors, *Image Analysis*, volume 7944, pages 408–419. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-38885-9 978-3-642-38886-6. doi: 10.1007/978-3-642-38886-6_39. URL http://link.springer.com/10.1007/978-3-642-38886-6_39. Series Title: Lecture Notes in Computer Science.

[9] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. YOLOv4: Optimal Speed and Accuracy of Object Detection. *arXiv:2004.10934 [cs, eess]*, April 2020. URL http://arxiv.org/abs/2004.10934. arXiv: 2004.10934.

[10] Cormac G. Booth, Rachael R. Sinclair, and John Harwood. Methods for Monitoring for the Population Consequences of Disturbance in Marine Mammals: A Review. *Frontiers in Marine Science*, 7:115, 2020. ISSN 2296-7745. doi: 10.3389/fmars.2020.00115. URL https://www.frontiersin.org/article/10.3389/fmars.2020.00115.

[11] Zhaowei Cai, Quanfu Fan, Rogerio S. Feris, and Nuno Vasconcelos. A Unified Multi-scale Deep Convolutional Neural Network for Fast Object Detection. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, volume 9908, pages 354–370. Springer International Publishing, Cham, 2016. ISBN 978-3-319-46492-3 978-3-319-46493-0. doi: 10.1007/978-3-319-46493-0_22. URL http://link.springer.com/10.1007/978-3-319-46493-0_22. Series Title: Lecture Notes in Computer Science.

[12] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-Based Models for Speech Recognition. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://papers.nips.cc/paper/2015/hash/1068c6e4c8051cfd4e9ea8072e3189e2-Abstract.html.

[13] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, September 1995. ISSN 1573-0565. doi: 10.1007/BF00994018. URL https://doi.org/10.1007/BF00994018.

[14] darrenl. LabelImg, May 2022. URL https://github.com/tzutalin/labelImg. original-date: 2015-09-17T01:33:59Z.

[15] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural Architecture Search: A Survey. page 21.

[16] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010. ISSN 1573-1405. doi: 10.1007/s11263-009-0275-4. URL https://doi.org/10.1007/s11263-009-0275-4.

[17] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V. Le. DropBlock: A regularization method for convolutional networks. *arXiv:1810.12890 [cs]*, October 2018. URL http://arxiv.org/abs/1810.12890. arXiv: 1810.12890.

[18] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V. Le. NAS-FPN: Learning Scalable Feature Pyramid Architecture for Object Detection. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7029–7038, Long Beach, CA, USA, June 2019. IEEE. ISBN 978-1-72813-293-8. doi: 10.1109/CVPR.2019.00720. URL https://ieeexplore.ieee.org/document/8954436/.

[19] Ross Girshick. Fast R-CNN. *arXiv:1504.08083 [cs]*, September 2015. URL http://arxiv.org/abs/1504.08083. arXiv: 1504.08083.

[20] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *arXiv:1311.2524 [cs]*, October 2014. URL http://arxiv.org/abs/1311.2524. arXiv: 1311.2524.

[21] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. *arXiv:1406.4729 [cs]*, 8691:346–361, 2014. doi: 10.1007/978-3-319-10578-9_23. URL http://arxiv.org/abs/1406.4729. arXiv: 1406.4729.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, December 2015. URL http://arxiv.org/abs/1512.03385. arXiv: 1512.03385.

[24] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. Technical Report arXiv:1704.04861, arXiv, April 2017. URL http://arxiv.org/abs/1704.04861. arXiv:1704.04861 [cs] type: article.

[25] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Ko-

rattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. *arXiv:1611.10012 [cs]*, April 2017. URL http://arxiv.org/abs/1611.10012. arXiv: 1611.10012.

[26] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, and zhifeng Chen. GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/hash/093f65e080a295f8076b1c5722a46aa2-Abstract.html.

[27] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167 [cs]*, March 2015. URL http://arxiv.org/abs/1502.03167. arXiv: 1502.03167.

[28] R. Jewell, L. Thomas, C. M. Harris, K. Kaschner, R. Wiff, P. S. Hammond, and N. J. Quick. Global analysis of cetacean line-transect surveys: detecting trends in cetacean density. *Marine Ecology Progress Series*, 453: 227–240, May 2012. ISSN 0171-8630, 1616-1599. doi: 10.3354/meps09636. URL https://www.int-res.com/abstracts/meps/v453/p227-240/.

[29] Bingyi Kang, Zhuang Liu, Xin Wang, Fisher Yu, Jiashi Feng, and Trevor Darrell. Few-Shot Object Detection via Feature Reweighting. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 8419–8428, Seoul, Korea (South), October 2019. IEEE. ISBN 978-1-72814-803-8. doi: 10.1109/ICCV.2019.00851. URL https://ieeexplore.ieee.org/document/9010944/.

[30] Douglas Kinzey, Paula Olson, and Tim Gerrodette. Marine Mammal Data Collection Procedures on Research Ship Line-Transect Surveys by the Southwest Fisheries Science Center. page 35.

[31] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander Kolesnikov, Tom Duerig, and Vittorio Ferrari. The Open Images Dataset V4. *International Journal of Computer Vision*, 128(7):1956–1981, July 2020. ISSN 1573-1405. doi: 10.1007/s11263-020-01316-z. URL https://doi.org/10.1007/s11263-020-01316-z.

[32] Min Lin, Qiang Chen, and Shuicheng Yan. Network In Network. *arXiv:1312.4400 [cs]*, March 2014. URL http://arxiv.org/abs/1312.4400.

arXiv: 1312.4400.

[33] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft COCO: Common Objects in Context. *arXiv:1405.0312 [cs]*, February 2015. URL http://arxiv.org/abs/1405.0312. arXiv: 1405.0312.

[34] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature Pyramid Networks for Object Detection. *arXiv:1612.03144 [cs]*, April 2017. URL http://arxiv.org/abs/1612.03144. arXiv: 1612.03144.

[35] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path Aggregation Network for Instance Segmentation. *arXiv:1803.01534 [cs]*, September 2018. URL http://arxiv.org/abs/1803.01534. arXiv: 1803.01534.

[36] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single Shot MultiBox Detector. *arXiv:1512.02325 [cs]*, 9905:21–37, 2016. doi: 10.1007/978-3-319-46448-0_2. URL http://arxiv.org/abs/1512.02325. arXiv: 1512.02325.

[37] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic Gradient Descent with Warm Restarts. *arXiv:1608.03983 [cs, math]*, May 2017. URL http://arxiv.org/abs/1608.03983. arXiv: 1608.03983.

[38] Diganta Misra. Mish: A Self Regularized Non-Monotonic Activation Function. *arXiv:1908.08681 [cs, stat]*, August 2020. URL http://arxiv.org/abs/1908.08681. arXiv: 1908.08681.

[39] Laurence Morissette and Sylvain Chartier. The k-means clustering technique: General considerations and implementation in Mathematica. *Tutorials in Quantitative Methods for Psychology*, 9:15–24, February 2013. doi: 10.20982/tqmp.09.1.p015.

[40] Paul E. Nachtigall. Encyclopedia of Marine Mammals. 3rd edition. B. Würsig, J. G. M. Thewissen and K. M. Kovacs, eds. ISBN 13: 978-0-12-804327-1. Academic Press, London, U.K. 2018. 1,157 pp. Cloth US$170.00. *Marine Mammal Science*, 34(2):553–555, 2018. ISSN 1748-7692. doi: 10.1111/mms.12499. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/mms.12499. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/mms.12499.

[41] Sridhar Narayan. The generalized sigmoid activation function: Com-

petitive supervised learning. *Information Sciences*, 99(1-2):69–82, June 1997. ISSN 00200255. doi: 10.1016/S0020-0255(96)00200-9. URL https://linkinghub.elsevier.com/retrieve/pii/S0020025596002009.

[42] NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. CUDA, release: 10.2.89, 2020. URL https://developer.nvidia.com/cuda-toolkit.

[43] Rafael Padilla. Open-Source Visual Interface for Object Detection Metrics, June 2022. URL https://github.com/rafaelpadilla/review_object_detection_metrics. original-date: 2020-11-11T23:58:38Z.

[44] Rafael Padilla, Sergio L. Netto, and Eduardo A. B. da Silva. A Survey on Performance Metrics for Object-Detection Algorithms. In *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pages 237–242, July 2020. doi: 10.1109/IWSSIP48289.2020.9145130. ISSN: 2157-8702.

[45] Tianyu Pang, Xiao Yang, Yinpeng Dong, Hang Su, and Jun Zhu. Bag of Tricks for Adversarial Training. *arXiv:2010.00467 [cs, stat]*, March 2021. URL http://arxiv.org/abs/2010.00467. arXiv: 2010.00467.

[46] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830, 2011.

[47] Joseph Redmon and Ali Farhadi. YOLO9000: Better, Faster, Stronger. *arXiv:1612.08242 [cs]*, December 2016. URL http://arxiv.org/abs/1612.08242. arXiv: 1612.08242.

[48] Joseph Redmon and Ali Farhadi. YOLOv3: An Incremental Improvement. *arXiv:1804.02767 [cs]*, April 2018. URL http://arxiv.org/abs/1804.02767. arXiv: 1804.02767 version: 1.

[49] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, Las Vegas, NV, USA, June 2016. IEEE. ISBN 978-1-4673-8851-1. doi: 10.1109/CVPR.2016.91. URL http://ieeexplore.ieee.org/document/7780460/.

[50] 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*, volume 28. Curran As-

sociates, Inc., 2015. URL https://proceedings.neurips.cc/paper/2015/hash/14bfa6bb14875e45bba028a21ed38046-Abstract.html.

[51] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. pages 4510–4520, 2018. URL https://openaccess.thecvf.com/content_cvpr_2018/html/Sandler_MobileNetV2_Inverted_Residuals_CVPR_2018_paper.html.

[52] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556 [cs]*, April 2015. URL http://arxiv.org/abs/1409.1556. arXiv: 1409.1556.

[53] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions. *arXiv:1409.4842 [cs]*, September 2014. URL http://arxiv.org/abs/1409.4842. arXiv: 1409.4842.

[54] Mingxing Tan and Quoc Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning*, pages 6105–6114. PMLR, May 2019. URL https://proceedings.mlr.press/v97/tan19a.html. ISSN: 2640-3498.

[55] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. MnasNet: Platform-Aware Neural Architecture Search for Mobile. pages 2820–2828, 2019. URL https://openaccess.thecvf.com/content_CVPR_2019/html/Tan_MnasNet_Platform-Aware_Neural_Architecture_Search_for_Mobile_CVPR_2019_paper.

[56] Mingxing Tan, Ruoming Pang, and Quoc V. Le. EfficientDet: Scalable and Efficient Object Detection. *arXiv:1911.09070 [cs, eess]*, July 2020. URL http://arxiv.org/abs/1911.09070. arXiv: 1911.09070.

[57] theAIGuy. tensorflow-yolov4-tflite, June 2022. URL https://github.com/theAIGuysCode/tensorflow-yolov4-tflite. original-date: 2020-07-08T19:41:10Z.

[58] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition, Fouth Edition*. Academic Press, Inc., 4th edition, 2008. ISBN 1-59749-272-8.

[59] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. Selective Search for Object Recognition. *International Journal of Computer Vision*, 104(2):154–171, September 2013. ISSN 1573-1405. doi: 10.1007/

s11263-013-0620-5. URL https://doi.org/10.1007/s11263-013-0620-5.

[60] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *arXiv:1706.03762 [cs]*, December 2017. URL http://arxiv.org/abs/1706.03762. arXiv: 1706.03762.

[61] 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:1911.11929 [cs]*, November 2019. URL http://arxiv.org/abs/1911.11929. arXiv: 1911.11929.

[62] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Scaled-YOLOv4: Scaling Cross Stage Partial Network. pages 13029–13038, 2021. URL https://openaccess.thecvf.com/content/CVPR2021/html/Wang_Scaled-YOLOv4_Scaling_Cross_Stage_Partial_Network_CVPR_2021_paper.html?ref=https://githubhelp.com.

[63] Kaixin Wang, Jun Hao Liew, Yingtian Zou, Daquan Zhou, and Jiashi Feng. PANet: Few-Shot Image Semantic Segmentation With Prototype Alignment. pages 9197–9206, 2019. URL https://openaccess.thecvf.com/content_ICCV_2019/html/Wang_PANet_Few-Shot_Image_Semantic_Segmentation_With_Prototype_Alignment_ICCV_2019_paper.html.

[64] Joe H. Ward. Hierarchical Grouping to Optimize an Objective Function. *Journal of the American Statistical Association*, 58(301):236–244, 1963. ISSN 0162-1459. doi: 10.2307/2282967. URL https://www.jstor.org/stable/2282967. Publisher: [American Statistical Association, Taylor & Francis, Ltd.].

[65] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. CBAM: Convolutional Block Attention Module. *arXiv:1807.06521 [cs]*, July 2018. URL http://arxiv.org/abs/1807.06521. arXiv: 1807.06521.

[66] Hongkun Yu, Chen Chen, Xianzhi Du, Yeqing Li, Abdullah Rashwan, Le Hou, Pengchong Jin, Fan Yang, Frederick Liu, Jaeyoun Kim, and Jing Li. TensorFlow Model Garden, 2020. URL https://github.com/tensorflow/models.

[67] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features. *arXiv:1905.04899 [cs]*, August 2019. URL http://arxiv.org/abs/1905.04899. arXiv: 1905.04899.

[68] Sergey Zagoruyko and Nikos Komodakis. Wide Residual Networks. Technical Report arXiv:1605.07146, arXiv, June 2017. URL `http://arxiv.org/abs/1605.07146`. arXiv:1605.07146 [cs] type: article.

[69] Chang-Bin Zhang, Peng-Tao Jiang, Qibin Hou, Yunchao Wei, Qi Han, Zhen Li, and Ming-Ming Cheng. Delving Deep into Label Smoothing. *arXiv:2011.12562 [cs]*, July 2021. doi: 10.1109/TIP.2021.3089942. URL `http://arxiv.org/abs/2011.12562`. arXiv: 2011.12562 version: 2.

[70] Zhilu Zhang and Mert Sabuncu. Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL `https://proceedings.neurips.cc/paper/2018/hash/f2925f97bc13ad2852a7a551802feea0-Abstract.html`.

[71] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren. Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34 (07):12993–13000, April 2020. ISSN 2374-3468. doi: 10.1609/aaai.v34i07.6999. URL `https://ojs.aaai.org/index.php/AAAI/article/view/6999`. Number: 07.

[72] Zhou and Chellappa. Computation of optical flow using a neural network. In *IEEE 1988 International Conference on Neural Networks*, pages 71–78 vol.2, 1988. doi: 10.1109/ICNN.1988.23914.

8 Appendix

8.1 EfficientDetD1.config

Shows the configuration used for training the EfficientDet D1 model utilizing Tensorflow Model Zoo [5].

```
model {

  ssd {
    num_classes: 1
    image_resizer {
      keep_aspect_ratio_resizer {
        min_dimension: 640
        max_dimension: 640
        pad_to_max_dimension: true
      }
    }
    feature_extractor {
      type: "ssd_efficientnet-b1_bifpn_keras"
      conv_hyperparams {
        regularizer {
          l2_regularizer {
            weight: 3.9999998989515007e-05
          }
        }
        initializer {
          truncated_normal_initializer {
            mean: 0.0
            stddev: 0.029999999329447746
          }
        }
        activation: SWISH
        batch_norm {
          decay: 0.9900000095367432
          scale: true
          epsilon: 0.0010000000474974513
        }
        force_use_bias: true
      }
      bifpn {
        min_level: 3
        max_level: 7
        num_iterations: 4
        num_filters: 88
      }
    }
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 1.0
        x_scale: 1.0
        height_scale: 1.0
        width_scale: 1.0
      }
    }
    matcher {
      argmax_matcher {
        matched_threshold: 0.5
        unmatched_threshold: 0.5
        ignore_thresholds: false
        negatives_lower_than_unmatched: true
        force_match_for_each_row: true
        use_matmul_gather: true
      }
    }
    similarity_calculator {
      iou_similarity {
      }
    }
```

```
box_predictor {
  weight_shared_convolutional_box_predictor {
    conv_hyperparams {
      regularizer {
        l2_regularizer {
          weight: 3.9999998989515007e-05
        }
      }
      initializer {
        random_normal_initializer {
          mean: 0.0
          stddev: 0.009999999776482582
        }
      }
      activation: SWISH
      batch_norm {
        decay: 0.9900000095367432
        scale: true
        epsilon: 0.0010000000474974513
      }
      force_use_bias: true
    }
    depth: 88
    num_layers_before_predictor: 3
    kernel_size: 3
    class_prediction_bias_init: -4.599999904632568
    use_depthwise: true
  }
}
anchor_generator {
  multiscale_anchor_generator {
    min_level: 3
    max_level: 7
    anchor_scale: 4.0
    aspect_ratios: 1.0
    aspect_ratios: 2.0
    aspect_ratios: 0.5
    scales_per_octave: 3
  }
}
post_processing {
  batch_non_max_suppression {
    score_threshold: 9.99999993922529e-09
```

```
        iou_threshold: 0.3
        max_detections_per_class: 100
        max_total_detections: 100
      }
      score_converter: SIGMOID
    }
    normalize_loss_by_num_matches: true
    loss {
      localization_loss {
        weighted_smooth_l1 {
        }
      }
      classification_loss {
        weighted_sigmoid_focal {
          gamma: 1.5
          alpha: 0.25
        }
      }
      classification_weight: 1.0
      localization_weight: 1.0
    }
    encode_background_as_zeros: true
    normalize_loc_loss_by_codesize: true
    inplace_batchnorm_update: true
    freeze_batchnorm: false
    add_background_class: false
  }
}
train_config {
  batch_size: 6
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    random_scale_crop_and_pad_to_square {
      output_size: 640
      scale_min: 0.1
      scale_max: 2.0
    }
  }
  sync_replicas: true
  optimizer {
    momentum_optimizer {
      learning_rate {
        cosine_decay_learning_rate {
          learning_rate_base: 0.0003
          total_steps: 60000
          warmup_learning_rate: 0.000001
          warmup_steps: 500
        }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
  fine_tune_checkpoint: "workspace/pre-trained-models/
     efficientdet_d1_coco17_tpu-32/checkpoint/ckpt-0"
  num_steps: 60000
  startup_delay_steps: 0.0
  replicas_to_aggregate: 8
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
  fine_tune_checkpoint_type: "detection"
  use_bfloat16: true
  fine_tune_checkpoint_version: V2
}
train_input_reader: {
  label_map_path: "workspace/annotations/label_map.
     pbtxt"
  tf_record_input_reader {
    input_path: "workspace/annotations/porpoise_train.
       tfrecord"
  }
}

eval_config: {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
  batch_size: 1;
}

eval_input_reader: {
  label_map_path: "workspace/annotations/label_map.
     pbtxt"
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "workspace/annotations/porpoise_test.
       tfrecord"
  }
}
```

8.2 Faster-RCNN.config

Shows the configuration used for training the Faster R-CNN model utilizing Tensorflow Model Zoo [5].

```
# Faster R-CNN with Resnet-50 (v1)
# Trained on COCO, initialized from Imagenet
   classification checkpoint

# Achieves -- mAP on COCO14 minival dataset.

# This config is TPU compatible.

model {
  faster_rcnn {
    num_classes: 1
    image_resizer {
      keep_aspect_ratio_resizer {
        min_dimension: 640
        max_dimension: 640
        pad_to_max_dimension: true
      }
    }
    feature_extractor {
      type: 'faster_rcnn_resnet101_keras'
      batch_norm_trainable: true
    }
    first_stage_anchor_generator {
      grid_anchor_generator {
        scales: [0.25, 0.5, 1.0, 2.0]
        aspect_ratios: [0.5, 1.0, 2.0]
        height_stride: 16
        width_stride: 16
    }
  }
  first_stage_box_predictor_conv_hyperparams {
    op: CONV
    regularizer {
      l2_regularizer {
        weight: 0.0
      }
    }
    initializer {
      truncated_normal_initializer {
        stddev: 0.01
      }
    }
  }
  first_stage_nms_score_threshold: 0.0
  first_stage_nms_iou_threshold: 0.7
  first_stage_max_proposals: 300
  first_stage_localization_loss_weight: 2.0
  first_stage_objectness_loss_weight: 1.0
  initial_crop_size: 14
  maxpool_kernel_size: 2
  maxpool_stride: 2
  second_stage_box_predictor {
    mask_rcnn_box_predictor {
      use_dropout: false
      dropout_keep_probability: 1.0
      fc_hyperparams {
        op: FC
        regularizer {
          l2_regularizer {
            weight: 0.0
          }
        }
        initializer {
          variance_scaling_initializer {
            factor: 1.0
            uniform: true
            mode: FAN_AVG
          }
        }
      }
      share_box_across_classes: true
      }
    }
    second_stage_post_processing {
      batch_non_max_suppression {
        score_threshold: 0.0
        iou_threshold: 0.3
        max_detections_per_class: 100
        max_total_detections: 300
      }
      score_converter: SOFTMAX
    }
    second_stage_localization_loss_weight: 2.0
    second_stage_classification_loss_weight: 1.0
    use_static_shapes: true
    use_matmul_crop_and_resize: true
    clip_anchors_to_image: true
    use_static_balanced_label_sampler: true
    use_matmul_gather_in_matcher: true
  }
}

train_config: {
  batch_size: 4
  sync_replicas: true
  startup_delay_steps: 0
  replicas_to_aggregate: 8
  num_steps: 100000
  optimizer {
    momentum_optimizer: {
      learning_rate: {
        cosine_decay_learning_rate {
          learning_rate_base: .04
          total_steps: 100000
          warmup_learning_rate: .013333
          warmup_steps: 5000
        }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
  fine_tune_checkpoint_version: V2
  fine_tune_checkpoint: "workspace/pre-trained-models/
      faster_rcnn_resnet101_v1_640x640_coco17_tpu-8/
      checkpoint/ckpt-0"
  fine_tune_checkpoint_type: "detection"
  data_augmentation_options {
    random_horizontal_flip {
    }
  }

  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
  use_bfloat16: true  # works only on TPUs
}

train_input_reader: {
  label_map_path: "workspace/annotations/label_map.
      pbtxt"
  tf_record_input_reader {
    input_path: "workspace/annotations/porpoise_train.
        tfrecord"
  }
}

eval_config: {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
  batch_size: 1;
}

eval_input_reader: {
  label_map_path: "workspace/annotations/label_map.
      pbtxt"
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "workspace/annotations/porpoise_test.
        tfrecord"
  }
}
```

8.3 YOLOv4.config

Shows the hyperparameters used during training of the YOLOv4 on Darknet [9].

```
yolov4-custom.cfg

[net]
# Testing
#batch=1
#subdivisions=1
# Training
batch=64
subdivisions=16
width=640
height=640
channels=3
momentum=0.949
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=1000
max_batches = 6000
policy=steps
steps=4800,5400
scales=.1,.1

#cutmix=1
mosaic=1
```