SMARTPHONE BASED OBJECT DETECTION FOR SHARK SPOTTING

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Darrick Oliver

October 2023

COMMITTEE MEMBERSHIP

TITLE: Smartphone Based Object Detection for Shark Spotting

AUTHOR: Darrick Oliver

DATE SUBMITTED: October 2023

COMMITTEE CHAIR: Franz Kurfess, Ph.D.
Professor of Computer Science

COMMITTEE MEMBER: Sumona Mukhopadhyay, Ph.D.
Professor of Computer Science

COMMITTEE MEMBER: Paul Anderson, Ph.D.
Professor of Computer Science

ABSTRACT

Smartphone Based Object Detection for Shark Spotting

Darrick Oliver

Given concern over shark attacks in coastal regions, the recent use of unmanned aerial vehicles (UAVs), or drones, has increased to ensure the safety of beachgoers. However, much of city officials' process remains manual, with drone operation and review of footage still playing a significant role. In pursuit of a more automated solution, researchers have turned to the usage of neural networks to perform detection of sharks and other marine life. For on-device solutions, this has historically required assembling individual hardware components to form an embedded system to utilize the machine learning model. This means that the camera, neural processing unit, and central processing unit are purchased and assembled separately, requiring specific drivers and involves a lengthy setup process. Addressing these issues, we look to the usage of smartphones as a novel integrated solution for shark detection. This paper looks at using an iPhone 14 Pro as the driving force for a YOLOv5 based model, and comparing our results to previous literature in shark-based object detection. We find that our system outperforms previous methods at both higher throughput and increased accuracy.

ACKNOWLEDGMENTS

Thanks to:

- My family for supporting me every step of the way, with special regard to my father for inspiring me every day to grow and expand my knowledge.
- My friends, especially Shivam Asija for his continual persuasion and persistence to pursue higher education with me.
- Dr. Franz Kurfess for exciting me into joining and exploring the realm of sharks, and for his patience through the learning process.
- The CSU Long Beach Shark Lab for providing the dataset that made this thesis possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Motivation

The counting and monitoring of marine animals is historically a tedious yet crucial process for obtaining data on population, determining behavioral trends, and for ensuring the safety of nearby humans [10]. The manual nature of this process has led to issues of efficiency, as it requires substantial time and labor to manually track marine wildlife, especially over long periods of time. In response to this, many researchers have already turned to the usage of unmanned aerial vehicles (UAVs), or drones, to monitor these animals in their natural habitat [12].

While this technological advancement has improved the counting process over ground searching, it still requires manual observance of footage. Especially in situations where immediate response may be necessary, this is a significant drawback to the current system. On top of this, researchers must also review recorded footage to ensure nothing is missed during flight time [35]. The process of manually reviewing and observing footage is not only time consuming, but also requires specialized skills to perform well [1].

However, further complexity exists beyond just observation. Conducting manual reviews also introduces the potential for significant human error, as inherit limitations and variability in judgement can affect the decision making process. One of the ways this can be seen impacting productivity the most is through pilot fatigue. Operating a drone for multiple hours can lead to diminishing accuracy in measurements over the typical 6 hour blocks that UAV operators perform their duties [10, 11].

To further highlight the urgency of addressing these limitations, there was a recent incident involving a shark attack at Rockaway Beach [34]. Following this attack, city officials increased beach surveillance using drones, but these drones were to be manually operated from 9 am to dusk daily until the end of the summer. The amount of labor required for this is not sustainable over a long period of time, emphasizing the need for a more autonomous solution.

1.2 Constraints and Limitations

In pursuit of such solutions, researchers have sought to train neural networks to make numerical predictions of population estimates [12, 52]. Many of these studies have been conducted over the counting of livestock, leaving shark detection and tracking a relatively uncharted domain. Because the models proposed by researchers are used in combination with UAVs, they typically require "edge computing" devices to process data. Edge computing devices live in the physical world and can either generate predictions on their local hardware ("edge-only"), or make use of cloud-based GPUs ("edge-cloud"). The processing power of these devices is generally limited, as their form factor plays a significant role in their real-world viability.

One of the major issues when considering an edge-cloud monitoring system is the cost associated with cloud computing. While the processing capabilities of devices hosted on the cloud are much greater than the those of onboard hardware, they come with a large, ongoing price tag. Another major consideration when considering the usage of cloud devices is the availability of networks. This is important when observing ocean life, as the devices will be farther away from civilization. Even if the network is available, processing important data over an unstable network can lead to data loss. Recent work done by the Sharkeye team have relied on custom expanded network

infrastructure to facilitate the usage of cloud-based solutions [18]. However, the requirement of additional infrastructure presents a fiscal trade-off between investing in hardware and infrastructure.

Of edge-only options, the most common solution is to use custom hardware with neural accelerators to efficiently run object detection models on the drone itself. However, this approach requires the integration of multiple components into a single system, posing a challenge of both hardware and software by developing an embedded device. This not only risks more potential points of failure, but also increases the overall complexity of the system. Alternatively, custom object detection models can also be installed on existing drone hardware. However, this comes with its own set challenges. Commercial drones usually are closed systems, which make developing and integrating code a much greater endeavour. On top of this, the hardware may not be suitable or powerful enough to run complex machine learning models.

The approach we choose to investigate is the usage of smartphones as a potential alternative to embedded hardware. Phones are both readily available and contain a variety of modern hardware components, including neural accelerators, high quality cameras, and even LiDAR. Their widespread availability and the consolidation of hardware into a single device eliminates the need for integration of chips from different manufacturers, presenting a simplified option from a development standpoint. Not only are they a more simple option, but modern smartphones rival hardware of some laptops.

1.3 Goals and Objectives

The overarching goal of this thesis is to explore and develop an autonomous solution for the monitoring and counting of marine animals using unmanned aerial vehicles

(UAVs) with neural networks and edge computing techniques. Ultimately this system should support multi-classification of both people and sharks, being able to detect both accurately. This endeavor is guided by the following specific objectives:

1. **Smartphone Feasibility Study:** To investigate the feasibility of utilizing phones as an integrated and readily available solution, examining the viability of their utilization in the context of shark detection from a UAV.

2. **Prototype Development:** To develop an open-source prototype of an autonomous UAV monitoring system on a mobile device that utilizes onboard neural accelerators, aiming to reduce human error and streamline the monitoring process of sharks.

3. **System Testing and Evaluation:** To conduct a series of tests and evaluations of the developed system, assessing its performance in terms of accuracy, throughput, and resource utilization.

Through these objectives, this thesis aims to provide a novel mobile solution for a more efficient, accurate, and sustainable system to observe marine animals, ensure the safety of people in proximity to sharks, and reduce inherent human error. As far as we are aware, there has been no prior investigation by other researchers into the usage of mobile phones for shark detection.

Chapter 2

BACKGROUND

2.1 Object Detection

The concept of object detection is comprised of two parts: the process of locating objects, and the process of identifying them from an image or a sequence of images. In the context of object detection, these processes are called "localization" and "classification" tasks respectively.

As humans, we perform a natural form of object detection effortlessly; our brains continuously identify and locate objects within our field of vision. For computing devices, object detection algorithms aim to replicate this process through the usage of neural networks. Modern object detection frameworks utilize convolutional neural networks (CNNs) to extract features from the inputted images, acting as a "brain" to interpret what the "eyes" see.

2.2 R-CNN (Region-Based Convolutional Neural Networks)

Many modern object detection methods have been derived from a framework developed by Girshick et al. called "R-CNN," which they define as "Regions with CNN Features" [17]. The general flow of this model involves first processing an inputted image into regions of interest, performing feature extraction on those regions, and finally classifying the objects it finds [17]. Figure 2.1 shows the overview of the complete object detection system.

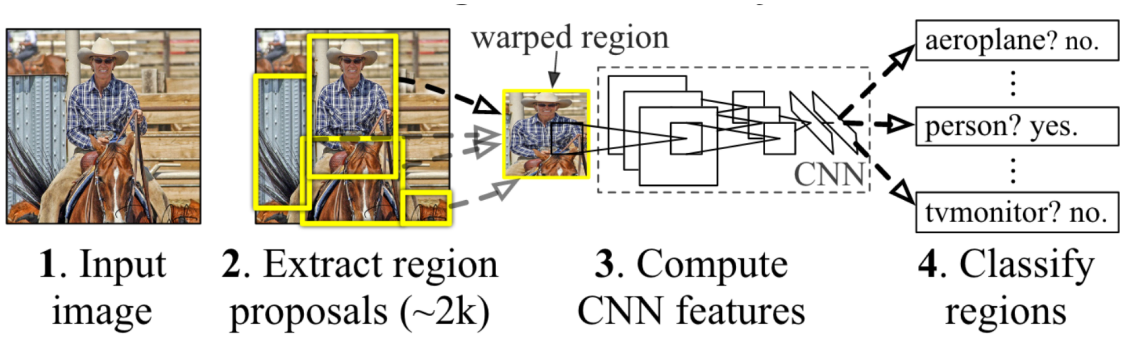


Figure 2.1: R-CNN System Overview [17, Figure 1]

To determine regions of interest, R-CNN uses a region proposal algorithm that identifies regions of interest in the input image. Once all proposed regions have been selected, they undergo feature extraction through a CNN. The CNN then transforms each region into a fixed-size feature vector. The final stage involves classifying these feature vectors using class-specific linear Support Vector Machines (SVMs) to determine the objects present in the regions [17].

At the time, R-CNN was most notable for enhancing the mean average precision (mAP) by over 30% compared to the previous best result on the VOC 2012 dataset. Furthermore, R-CNN saw a performance increase on the ILSVRC2013 detection dataset, outperforming previous leading object detection methods. The open-source nature of R-CNN, with its source code being publicly available [17], has facilitated even more research and development in the field, creating a new generation of object detection methodologies derived from the R-CNN framework.

For edge computing devices, however, performance increases are high priority. When Ren et al. developed and published the model framework Faster R-CNN, they showed that latency from the R-CNN model could be drastically improved by using an additional neural network to find regions of interest before performing classification [41]. This allowed for more practical use cases for R-CNN on more devices, as images could be processed in "near real-time" [41].

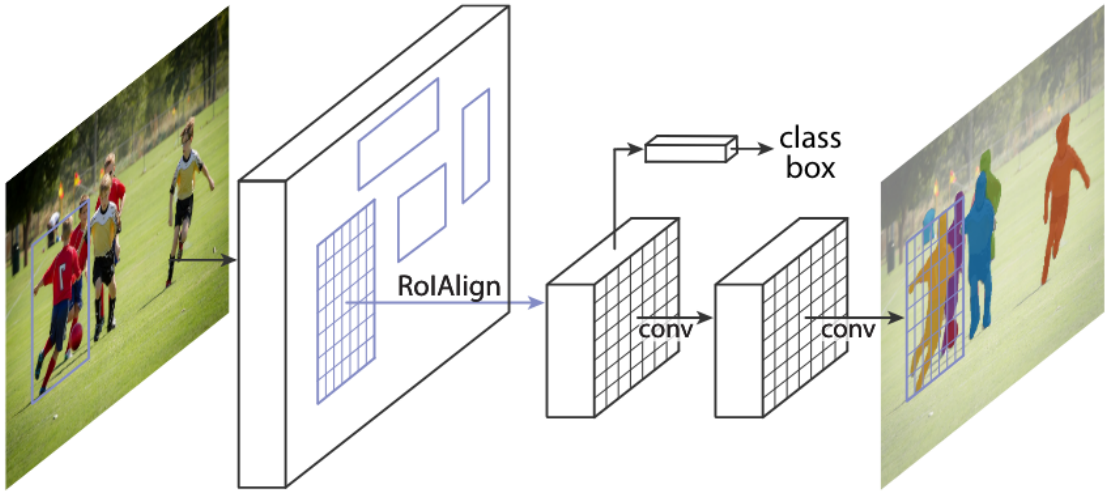


Figure 2.2: Mask R-CNN Framework [19, Figure 1]

The following year, He et al. further extended the research done by Ren et al. to improve on Faster R-CNN. Rather than using bounding boxes to identify an object, they proposed the creation of pixel by pixel "masks" over an object for higher precision estimates. This method uses a similar architecture to Faster R-CNN, but an additional branch predicts whether or not each individual pixel in a region of interest would be a part of the object [19]. This branch runs in parallel to the classification model, as seen in Figure 2.2, increasing the overhead only slightly [19].

2.3 YOLO (You Only Look Once)

Shortly after Ren. et al released their paper on Faster R-CNN, Joseph Redmon et al. introduced a new technique that used only a single large neural network. YOLO, standing for "You Only Look Once," revolutionized object detection by introducing a framework that was capable of real-time object detection [39]. Developed by Joseph Redmon et al., this approach diverges significantly from the region proposal-based methodologies like R-CNN and its derivatives. Rather than taking multiple passes over an image to perform region of interest identification into recognition, YOLO

is considered a "one-stage" detector that only performs one pass over the image to produce bounding boxes with classification labels [39].

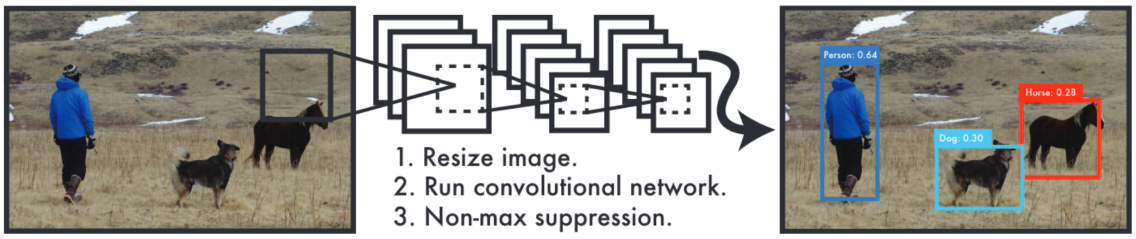


Figure 2.3: YOLO System Overview [39, Figure 1]

The YOLO framework operates through a single convolutional neural network [39, 41] that applies to the entire image and divides it into a grid. Each cell in the grid is responsible for detecting objects in it's region. These cells will predict a fixed number of bounding boxes along with confidence scores for the boxes and class probabilities. These predictions are then used to determine the class labels and bounding boxes for objects in the image. The primary goal of this research was to create a real-time detector, with their framework able to process images in real-time at 45 frames per second, and at even faster rates with a reduced version of the network [39].

2.4 SSD (Single Shot MultiBox Detector)

The theory behind the Single Shot MultiBox Detector was first introduced by Liu et al. in 2016. This paper presented a method for detecting objects in images using a single deep neural network. The proposed process involves converting the possible shapes and sizes of an object (bounding boxes) into a standardized set of boxes with different proportions and sizes for each part of the image being analyzed [30]. The network then generates scores based on the presence of each object category in each default box, and makes adjustments to the box to more closely match the

object shape. Liu et al. found that the SSD framework is simple, fast, and accurate compared to methods that require object proposals, like R-CNN.

Liu et al. reported that SSD512 (SSD with a 512x512 input size) outperformed Fast R-CNN, Faster R-CNN, and YOLO in precision on the PASCAL VOC2007 dataset [30]. In terms of speed, it achieved slightly worse results than YOLO, but saw a boost in performance when compared to both Fast and Faster R-CNN. The SSD framework brought forward a balance between speed and accuracy, carving out a significant space in real-time object detection research.

2.5 MobileNet

Following the research done by Liu et al., Google researcher Andrew Howard and his team proposed a new architecture for computer vision in 2017, specifically designed for mobile devices, referred to as a "MobileNet" model. This proposed method introduces the usage of depthwise separable convolutions as an efficiency improvement over standard convolutions [22].

Typically when a neural network processes information, it does so in a single step. This can be slow and use an abundance of resources. With depthwise separable convolutions, the network splits the process into two parts: filtering and combining. The filtering step applies a filter to each input channel, while the combining step uses a 1×1 convolution to put the filtered outputs back together [22].

When compared to leading architectures like Inception V2 and deeplab-VGG, MobileNet displayed a marginal dip in precision but used significantly less resources. This architecture was concluded to be a viable option on a SSD after the researchers found that using MobileNet on a SSD achieved greater accuracy scores when com-

pared to Faster-RCNN. This was further substantiated when MobileNetV2 integrated with SSD300 outperformed YOLOv2 in both precision and performance [42].

2.6 EfficientNet and EfficientDet

In 2019, Mingxing Tan and Quoc V. Le developed and put forward a novel CNN architecture dubbed "EfficientNet". This approach centered on a compound scaling method, a strategy that simultaneously scales the depth, width, and resolution of the network layers, optimizing the model's performance [48]. The EfficientNet architecture demonstrated superior performance compared to leading methods, while maintaining a reduced parameter count, effectively creating a more efficient model with little compromise on accuracy [48].

The principles of EfficientNet were then used to develop EfficientDet, an object detection model that leverages the aforementioned principles to enhance execution speed [49]. EfficientDet uses an ImageNet-pretrained EfficientNet as the backbone network, and a BiFPN as the feature network, which are used to achieve state-of-the-art performance while being much more efficient than previous detectors [49]

Chapter 3

RELATED WORK

3.1 Animal Object Detection

The concept of studying animals to track both their physical health and location has been a common research focus. The goals of such research vary, some using it as a method to ensure safety of animals and others for individual animal identification [6]. In some cases, researchers use object detection on UAVs to monitor threatened or invasive species, allowing researchers to observe animals that are of conservation concern [20, 44, 13].

R-CNN is a well-established object detection framework, and has historically been the standard for completing object detection related tasks. The creation of descendant frameworks of R-CNN that have iteratively improved on speed concerns has led to a more widespread usage of the R-CNN family of detectors. R-CNN detectors have been utilized in many research oriented papers involving livestock animals, ranging in use from sheep [43], to cattle [3], and even pigs [14].

In 2019, Cowton et al. published a research paper in which they investigated the usage of Faster R-CNN to monitor and identify pigs. Due to the quality of their RGB camera, Cowton et al. were only able to record and process images at an average of 4 frames per second (FPS). This ended up being a bottleneck for other components of their system, limiting the number of frames that could be captured. Despite this limitation, they were reportedly able to achieve 0.901 mean average precision (mAP) at an intersection over union (IOU) of 0.6 [14].

Another group of researchers utilized a hybrid model of Faster R-CNN and YOLO to observe an endangered species of koalas located in Australia at around the same time [13]. This method of identifying and counting animals yielded precision rates higher than that of manual counters, a reported 60% score by the system compared to 34% using manual counting methods. The combination of the two object detection models brought a significant increase in precision when compared to previous studies [13]. However, the overhead required for such a task meant that it performed detection at the same speed as manual validators.

More recently, Xu et al. looked into the usage of a Mask R-CNN model to count cows from a drone. They explored the usage of this model because it had been demonstrated to provide greater precision than both SSD and YOLO models, especially on lower resolutions. Mask R-CNN improves on difficulties experienced with models like YOLO by making pixel-mapped "masks", or outlines, on only the parts of classified objects that are visible [19].

Prior research in this area reported issues of occlusions as a pain point when developing models for animal detection [52]. Most of the time, these occlusions occurred when animals would walk in front of each other. Xu et al. found that Mask R-CNN outperforms these leading algorithms in both counting accuracy and average precision (AP) on image datasets that included overlapping cattle. This is an especially important point when considering livestock, or other animals that live in herds, as there are frequently visual blockages due to their close proximity to one another. The reported APs were 0.96, 0.92, and 0.94 for full detection, head detection in pastures, and detection in feedlot respectively, measured at an IOU threshold of 0.5.

It is important to note that reported values must be understood in context, as results given by different papers tend to use different metrics. Some papers grade their

algorithms on higher or lower IOU thresholds, so values of precision and recall cannot be compared one-to-one unless they are performed under the same conditions.

3.2 Marine Animal Object Detection

The domain of object detection has seen significant expanse into a variety of applications beyond simply livestock, including to the monitoring of marine animals. Automation of this process has been a necessary advancement, as aerial monitoring has been undertaken manually for decades [45]. The marine environment presents unique challenges for object detection models, as occlusions and reflections are frequent, making accurate identification and tracking of marine animals exceedingly complex.

Luis Mejias et al. explored deep learning approaches to marine mammal detection back in 2013. They aimed to automate the process of counting the population of dugongs by utilizing a model built to understand aerial imagery. The research they performed investigated detecting dugongs on a custom payload attached to a drone.

Of approaches they tested, they were able to achieve a maximum precision of 4.97% and a maximum recall of 51.4%, on an average processing time of one image every 9.14 seconds [33]. The proportion of retrieved detections that were truly dugongs was very low, but this was primarily due to a large number of false detections.

Following this research, improvements were made by segmenting inputted images into sectors for classification [31]. Similar to R-CNN, these researchers created a pipeline that utilized a region proposal network to perform clustering on relevant spaces. Their results found a significant increase in recall scores, with slightly better precision scores compared to previous methods.

In 2018, Sharma et al. investigated the effectiveness of object detection on sharks, addressing a notable gap in research concerning marine animal detection, which was lacking most in the context of shark identification [45]. The dataset that they used featured a range of marine creatures, as well as human swimmers and boats, to ensure a general understanding of objects encountered in the ocean was feasible. These researchers ran into challenges regarding the lack of ground truth information for some of the larger animals, leading to less specific identifiers for these creatures.

Ultimately, their model was able to achieve a mean average precision (mAP) of 0.901 at 50% IOU with a speed of 0.13 seconds per image [45]. Taking into account the lack of distinction between the larger marine animals, these results were satisfactory to the researchers. They suggested that future work should be done to evaluate if using R-CNN for animal detection is beneficial on edge devices.

Advancing this technology further, an autonomous alert system for sharks called "Sharkeye" was published. The system developed by Gorkin et al. features a YOLO-like detector to process images, which are captured through the usage of a high resolution camera mounted to a blimp. Contrary to most other shark detectors, Sharkeye's system processes video on the cloud rather than on an edge device [18]. Alerts would be sent to the researchers' mobile devices if a shark was spotted from the cloud detector. To achieve a reliable internet connection over the ocean, they built a nano-gateway consisting of a LoRa Gateway module and Raspberry Pi. This system was attached to the blimp, allowing the team to extend the network beyond its typical capabilities. While this is a clever solution, it requires additional investment in hardware components and infrastructure to get running. The cost of maintaining a network like this not only is limited to initial setup, but also incurs ongoing costs associated with monitoring, maintenance, and even upgrades to ensure the system is reliable.

Even more recently, researchers studying the Great Barrier Reef designed and constructed a system around the NVIDIA Jetson Xavier, a neural accelerator built for embedded computing [28]. These researchers used the Crown-of-Thorn Starfish (COTS) as the primary target of detection, as they are an invasive reef species that targets coral. Using YOLOv4 as the primary detection model, they developed a real-time data collection and curation system on edge devices for COTS monitoring. The results of their study propose further emphasis on resource efficiency, as their model obtained a maximum throughput of 22 FPS [28].

3.3 Object Detection on the Edge

Object recognition and detection is utilized today on a wide variety of applications, including the usage of object recognition in edge computing devices. Using edge computing devices with object recognition naturally comes with a variety of considerations, such as processing power, battery life, and sometimes network connectivity. Processing power is typically a crucial factor when it comes to object recognition on edge devices as it determines how quickly and accurately the system can identify and categorize objects within its environment. A more powerful processor can handle complex algorithms and larger datasets, enabling more accurate recognition and faster processing times. However, this increased processing capability often comes at the cost of higher energy consumption, which can be a significant concern, especially in battery-powered devices [29].

Performing object detection from a UAV will typically feature edge computing devices to generate predictions. Recent research into real-time detection has brought general consensus that single stage detectors, such as YOLO and SSD, are the most effective detectors for embedded systems [24, 2, 9]. Primarily this is due to their excellent

trade-off between speed and accuracy, allowing for real-time detection on edge devices. Some researchers have looked into improving detection speed further by using pruned versions of YOLO, effectively removing unimportant connections with weights close to zero [29].

Most importantly, object detection on edge devices is an ongoing area of research that has seen great improvements in recent years [54]. Despite leaps in this area, there has been a lack of insight into the effectiveness of mobile devices running object detection algorithms. Prior research in this area has typically regarded novel object detection systems, such as ThunderNet [38] and Pelee [51], specifically designed for mobile devices. However, comparisons to other object detectors often are between older versions of YOLO and SSD.

3.4 Object Detection With Small Objects

Small object detection has been a challenge for object detection frameworks for a long time, especially for one-stage detectors [54]. This challenge is often worsened by the inherit limitations of detection frameworks, leading to decisions regarding trade-offs.

The YOLO framework, for instance, has notoriously been known to struggle with detecting smaller objects. This is especially relevant when comparing its performance to that of leading two-stage detectors. The architecture of YOLO is inherently designed for high speed with lower regard for accuracy, but more recent versions of YOLO [40, 7, 50] have aimed at addressing this issue with more optimized structures for small objects. YOLOv3 utilized a "backbone network," responsible for initial feature extraction from the input images, using ResNet for improved accuracy on small objects [40]. More recently, improvements to YOLOv7 for small object detection have

included the use of a human-like attention mechanism, using a special loss function and structure, and using new convolution and attention techniques [27].

Chapter 4

EVALUATION CRITERIA

4.1 Model Criteria

4.1.1 Concepts

Before examining the different standards used for validating object detection models, we will briefly touch on common principles they share. The most basic concepts to understand are defined below:

1. **True Positive (TP):** Correctly identified object presence.
2. **False Positive (FP):** Incorrectly detected an object.
3. **False Negative (FN):** Missed detection of an actual object.

Note that true negative (TN) is not present, as it does not apply in object detection. There are infinitely many bounding boxes that shouldn't be detected.

From these definitions, it is clear we need a way to classify what a "correct" vs "incorrect" detection looks like in order to obtain these metrics. The most common way of doing so is by using a measurement called the intersection over union (IOU) [37]. IOU is derived from a coefficient of similarity called the Jaccard Index [23].

Determining IOU requires first calculating the intersection and union of the predicted bounding box B_p with the ground truth bounding box B_{gt}. For a more visual understanding of how these areas are determined, see Figure 4.1. Taking the ratio of these values gives the intersection over union, as shown in Equation 4.1.

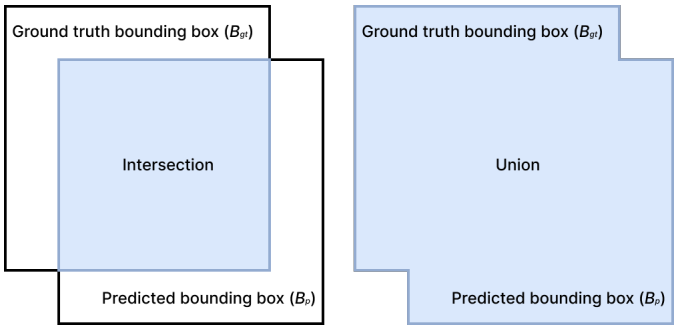


Figure 4.1: Area of Intersection and Union on Bounding Boxes

$$IOU = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})} \tag{4.1}$$

The resulting score of similarity is then compared to an IOU threshold t, which is set to determine at what point the detection is considered correct [36]. If $IOU \geq t$, the detection is considered correct and recorded as a true positive. Conversely, if $IOU < t$ the detection is recorded as incorrect. The larger this threshold is, the higher the standard is for determining correctness.

4.1.2 Precision

Precision is a widely accepted metric in computer vision to evaluate the performance of models. It is a metric that evaluates a model on its ability to make correct assumptions when it predicts the presence of an object. In essence, it is the percentage of correctly made positive predictions. Precision is calculated by taking the ratio of true positives to the sum of true positives and false positives, as shown in Equation 4.2.

$$Pr = Precision = \frac{TP}{TP + FP} \tag{4.2}$$

To maximize precision, the number of false positives recorded must be zero. In doing so, there may be many positives missed, potentially leading to a greater number of false negatives. Therefore, precision is not the only factor that needs to be considered when determining what qualifies as an acceptable model.

4.1.3 Recall

Recall, also known as sensitivity or true positive rate, is another important metric in computer vision for assessing model performance. While precision focuses on the accuracy of positive predictions, recall looks at the model's ability to identify relevant cases. In other words, it measures the percentage of correctly identified positives over all ground truths. The method to calculate recall is shown in Equation 4.3.

$$Rc = Recall = \frac{TP}{TP + FN} \tag{4.3}$$

A perfect recall would mean that there are no missed positives, resulting in zero false negatives. However, achieving a high recall might come at the expense of precision, as the model might also increase its number of false positives.

4.1.4 Average Precision and Mean Average Precision

Confidence threshold, τ, is a slightly different value than the aforementioned IOU threshold (t). While t establishes a boundary between correct and incorrect predictions, τ determines the level of confidence required for a prediction to be considered. If our model makes predictions below this confidence level, they are ignored and do not contribute to model metrics. For example, if our confidence threshold is set to 0.5 (50%), then any predictions with a confidence of less than 0.5 are discarded. Predic-

tions with higher confidences such as 0.5, 0.6, etc. are considered and will be factored into our metrics.

This is an important concept for understanding how a precision-recall curve is created, as we can iteratively modify this confidence threshold over N points to observe how precision and recall changes [36]. The resulting precision-recall curve can then be plotted with the resulting values, similar to what is shown in Figure 4.2. To determine average precision (AP) at a given IOU threshold, it's typical to either take an N-point interpolation or an all-point interpolation [37].

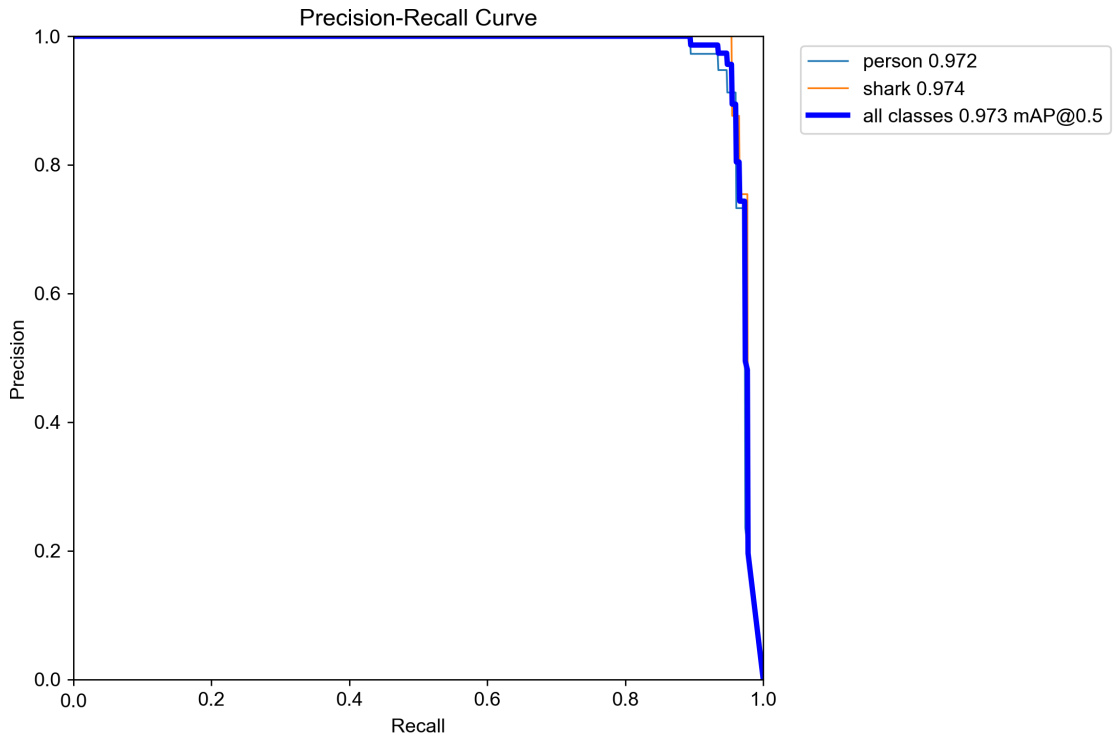


Figure 4.2: Precision-Recall Curve at $t = 0.5$

In both interpolations, the precision-recall curve is described using the precision values at some number of recall levels. While the N-point interpolation only interpolates over N evenly spaced recall points, the all-point interpolation interpolates through all points. The higher the number of points used, the more accurate the estimate for area under the curve (AUC) will be. For AP, this equation defines K as the number of recall

values from the set $R_r(k)$ [36]. Additionally, $Pr_{interp}(R)$ is defined as the maximum precision $Pr(\tau(k))$ for which the corresponding recall value $Rc(\tau(k))$ is greater than or equal to R, a real recall value on the interval [0,1] [36]. This ensures the precision-recall curve is monotonic after interpolation. The formula for determining AP is shown in Equation 4.4.

$$AP_{all} = \sum_{k=0}^{K} (R_r(k) - R_r(k+1)) Pr_{interp}(R_r(k)) \tag{4.4}$$

While this may be helpful for understanding precision on a single class, most research requires a method to represent the precision of detections over all classes. This is where mean average precision (mAP) can be utilized, computing the average AP over all classes. This is achieved through Equation 4.5, where C represents the number of classes, and AP_i is the AP of the ith class.

$$mAP = \frac{1}{C} \sum_{i=1}^{C} AP_i \tag{4.5}$$

For a better understanding of model behavior, it's also common to vary the IOU over mAP by taking an interval of IOU values and measuring the mAP at each. This is often denoted as either mAP@[i:j] or mAP@[i,j], where i is the lower bound and j is the upper bound of IOU values. For this research, mAP is taken as the average for $IOU \in [0.5 : 0.05 : 0.95]$, written simply as mAP@[0.5:0.95].

4.1.5 Average Recall and Mean Average Recall

In 2015, Jan Hosang et al. proposed a novel metric, average recall (AR) [21]. The method of calculating AR is different from AP because the estimated detection is

not taken into account, effectively taking $\tau = 0$. AR takes into account recall values at different IOU thresholds along the interval $[0.5, 1]$. The formula for AR is then written:

$$AR = \frac{2}{n} \sum_{i=1}^{n} max(IOU(gt_i) - 0.5, 0) \tag{4.6}$$

$IOU(gt_i)$ represents the IOU between the given ground truth, gt_i, and the best detection. n represents the total amount of ground truths.

Mean average recall (mAR) can then similarly be calculated for each individual class using Equation 4.7.

$$mAR = \frac{1}{C} \sum_{i=1}^{C} AR_i \tag{4.7}$$

4.2 Hardware Evaluation Criteria

4.2.1 Throughput

One of the primary evaluation criteria to consider when performing a test on edge devices is throughput (denoted X), measured in frames per second (FPS). This metric is used to evaluate both models and hardware, deriving the number of times that it is possible to execute a model on a supplied dataset over a period of time. Throughput can be calculated as a total number of frames f_{tot} over a given time interval, t_{tot}. This methods is shown below, in Equation 4.8.

$$X_{avg} = \frac{f_{tot}}{t_{tot}} \tag{4.8}$$

30 FPS is generally regarded as the threshold for what is considered "real-time" object detection, as per Wang et al. in their work on YOLOv7 [50]. Therefore, for our system to be considered operating in real-time, it is important to measure and observe performance at a minimum of 30 FPS.

This metric is particularly useful for comparison between using hardware present in a phone and specialized hardware. Ultimately this comparison leads to an important objective: determining if the device can operate at the level of or surpassing hardware specifically designed for machine learning on edge devices. Through this comparison we also gain insight into the cost-performance trade-offs for either choice.

4.2.2 Resource Utilization

Not only is it important to evaluate the throughput of the entire system, but also being able to identify and resolve bottlenecks is crucial for optimizing overall performance. The critical resources identified for utilization monitoring on mobile devices are memory, central processing unit (CPU), the Apple Neural Engine (ANE), and graphics processing unit (GPU) utilization. Resource utilization is often displayed as a percentage, representing the proportion of resource capacity that is being used to execute a task or application. Struggling to either maintain consistent readings or showing consistently large utilization indicates an inefficient usage of onboard resources. In addition, resource utilization has been shown to have a linear relationship with energy consumption [46].

The following formula is a general guideline to calculate the utilization of any resource, where t_{exec} is the time spent by the resource executing the workload and t_{total} is the total uptime of the resource [5]

$$R_{util} = \frac{t_{exec}}{t_{total}} \times 100\% \tag{4.9}$$

Additionally, the mean utilization of a resource can be determined by taking the average resource utilization over many test runs, denoted as $R_{util,avg}$.

Memory utilization is a key factor for edge devices, which typically have limited memory resources compared to centralized cloud servers. High memory utilization can greatly affect performance and even functionality of the application on a mobile device. Memory is typically measured in megabytes (MB) or gigabytes (GB), and can be evaluated both at rest (the base memory usage when no inference is being performed) and during inference (the additional memory required to perform inference).

Similar to memory utilization, CPU utilization is another important metric for evaluating performance on edge devices. High CPU utilization can lead to increased power consumption and heat generation, which in turn can affect the device's battery life and longevity. CPU utilization is usually expressed as a percentage of the total CPU capacity and can be evaluated both at rest and during inference.

The ANE utilization is an additional metric that will be thoroughly investigated, as plays a key role in machine learning tasks. The ANE is a type of neural processing unit, a hardware component designed for accelerating neural network operations like matrix multiplications and convolutions. Running Core ML model inferences on Apple hardware will automatically offload any processes to the ANE, which means that our device will be utilizing this component more than 30 times every second.

Similar to the aforementioned metrics, this utilization is expressed as a percentage of time executing to total time running. It's important to note that performance metrics from the usage of this chip will apply only to Apple mobile devices and not smartphones in general. There are similar neural accelerator architectures in other devices, but the complexities that arise from differences in hardware means that results obtained from the ANE do not translate one-to-one to all mobile neural accelerators.

The final metric that will be used for evaluation is the GPU utilization. Similar to the ANE, GPUs are a specialized hardware component designed for accelerating certain operations. But rather than allowing for faster neural network related operations, these chips are specifically created to accelerate graphics operations.

These metrics complete a comprehensive framework for evaluating the efficiency of machine learning tasks on mobile devices, allowing for comparisons to specialized edge hardware.

Chapter 5

DESIGN AND METHODOLOGY

5.1 Data Collection

The dataset used for training the models in this thesis was provided by CSU Long Beach, and all data collection occurred prior to the creation of this project. The number of images totaled to 3037 overall, with resolutions of 4096px by 2160px (4096x2160), gathered from a range of coastal locations. Figure 5.1 shows the distribution of label instances, along with distributions of coordinates and bounding box size.

During the annotation phase, human validators were tasked with reviewing each image and assigning bounding boxes around discernible objects. Validators assigned labels to humans, boats, sharks, and other marine life. While this was a very manual task, it was made simpler through the usage of the online application RoboFlow. RoboFlow provided a visual method of assigning coordinates to bounding boxes and labels to each corresponding bounding box. These annotations were vital to model training, as they provided ground truths for us to compare the given model outputs to.

The resulting dataset was then exported into splits of training, testing, and validation images. The images were randomly separated into 70% training, 10% testing, and 20% validation using RoboFlow's built-in training split software. The training and testing images would be used during the training of the object detection model, while the validation set would be held out to evaluate the models' generalization capabilities. The training of models involved "transfer learning," fine-tuning the model to this dataset on top of pre-trained weights. The process of transfer learning

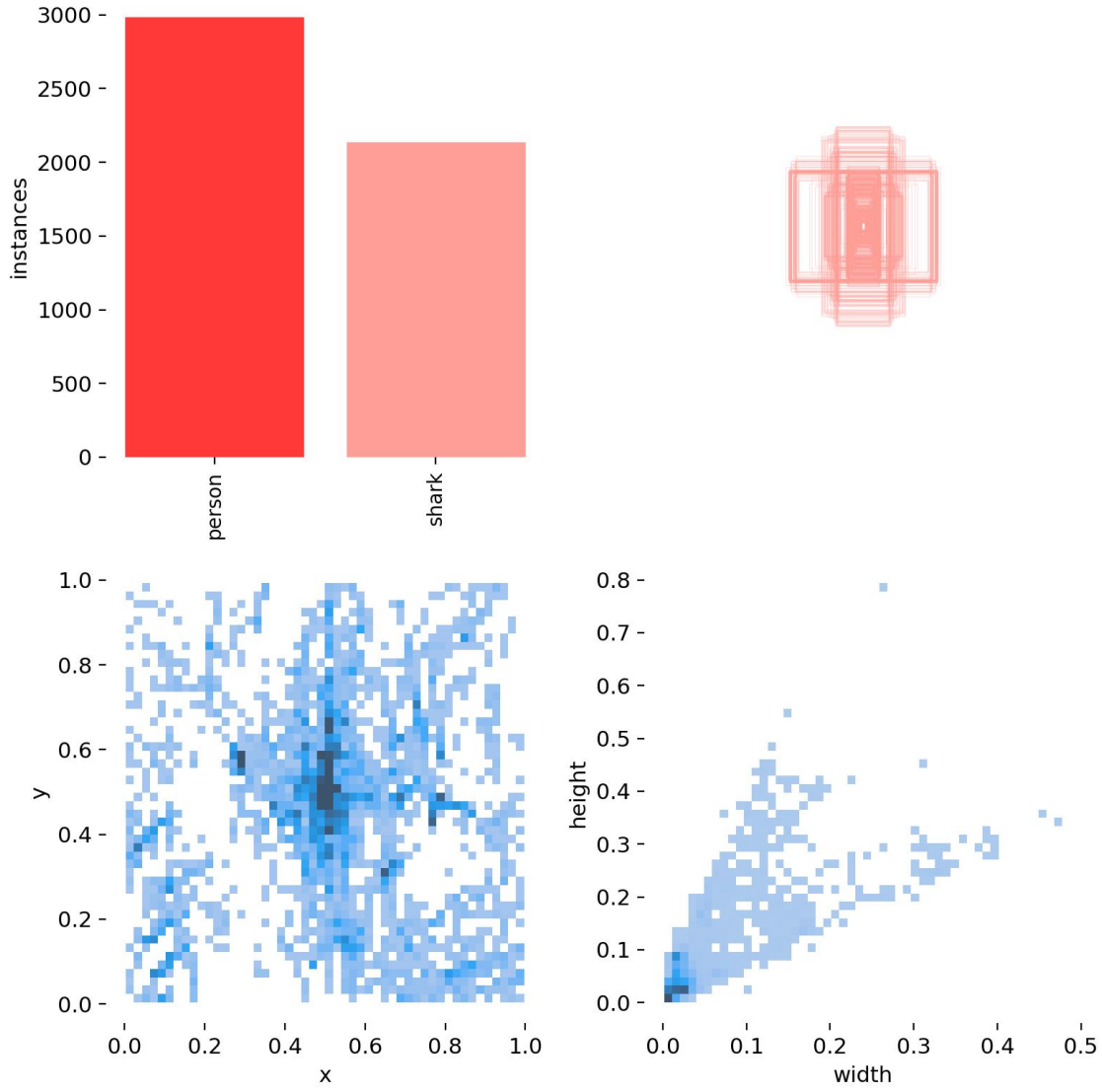


Figure 5.1: Dataset Distribution

involves utilizing a general source of knowledge, and applying it to a more specific use case. In this scenario, the pre-trained weights already have learned features and patterns from previous researchers training them on much larger datasets. These features are then fine-tuned on this shark dataset, theoretically allowing for more accurate predictions in this context.

5.2 Models Under Consideration

The models selected for evaluation were:

- Faster R-CNN with ResNet50
- YOLOv5
- YOLOv8
- SSD MobileNetV2
- EfficientDet D0

The models themselves were stored locally and validated using a Jupyter Notebook to apply the best weights to the given model. Both accuracy and speed tests were performed on each to obtain an average performance vector.

The models selected for testing were obtained through publicly available sources. Tensorflow's Detection Model Zoo [53] was used to obtain and train the models that required TensorFlow 2, while the YOLO models were obtained in PyTorch from the YOLOv5 [25] and YOLOv8 [26] repositories maintained by Ultralytics.

5.3 Model Selection

The comparison of each model is shown in Table 5.1, where results are sorted in descending order by mAP. As mentioned previously, mAP stands for mean average precision, mAR is mean average recall, and X_{avg} is average throughput in frames per second. When viewing this table, it's important to note that greater mAP, mAR, and throughput are more desirable. The varying image sizes listed here represent the

Table 5.1: Results

Model	Batch Size	Image Size	mAP	mAR	X_{avg}
YOLOv5n	16	640x640	0.624	0.662	31.1
YOLOv8n	16	640x640	0.610	0.648	24.0
YOLOv5n	16	416x416	0.591	0.624	31.0
SSD MobileNetV2	32	320x320	0.560	0.657	23.6
YOLOv8n	16	416x416	0.534	0.573	29.2
EfficientDet D0	4	512x512	0.530	0.597	8.95
Faster R-CNN	1	640x640	0.307	0.424	9.56

input image size to the model, which were chosen as default values provided from the respective sources.

Each model was evaluated on an NVIDIA GeForce RTX 3060 GPU with 6144 MiB of VRAM. The evaluation device contains 16 GB of RAM, which became a bottleneck for some of the models while training, notably when determining batch sizes and learning rates for training.

From these results it was observed that the YOLO models exhibited much greater precision and recall scores on average than EfficientDet and Faster R-CNN on this dataset. SSD MobileNetV2 showed more impressive inference times than EfficientDet and Faster R-CNN, but regardless didn't get close to the speed of the faster YOLO models. Figure 5.2 shows the comparison of mAP and execution time for each model, where models in the top left corner exhibit faster inference times and are exceedingly more accurate. The YOLO models not only reported quicker inference times than all other models on this dataset, but they also displayed much more accurate scores on the validation set. In Figure 5.3, the mAR is compared with execution time. There is not a clear difference between precision and recall on YOLO models, unlike the other models that exhibited between 11.8% and 32.0% differences.

While it may seem like the mAP and mAR values are lower than those recorded by researchers mentioned in the related works section, this is due to a stricter requirement

Table 5.2: Comparable Results

Model	Image Size	mAP@0.5	mAR@0.5
YOLOv5n	640x640	0.973	0.924
YOLOv8n	640x640	0.973	0.935
YOLOv5n	416x416	0.965	0.921
YOLOv8n	416x416	0.920	0.844
SSD MobileNetv2	320x320	0.861	0.657
EfficientDet D0	512x512	0.84	0.595
Faster R-CNN	640x640	0.645	0.423

taken for these metrics. As mentioned previously, mAP@[0.5:0.95] implies that the average over $IOU \in [0.5 : 0.05 : 0.95]$ is taken for precision, while mAP@0.5 is the precision over all classes at an IOU of 0.5. For the sake of a direct comparison to these models, the mAP@0.5 and mAR@0.5 for each is shown in Table 5.2. These values will be examined in further detail to compare to previous work in the results section (Table 7.2). From this point onwards, mAP is taken to mean mAP@[0.5:0.95] and mAR implies mAR@[0.5:1].

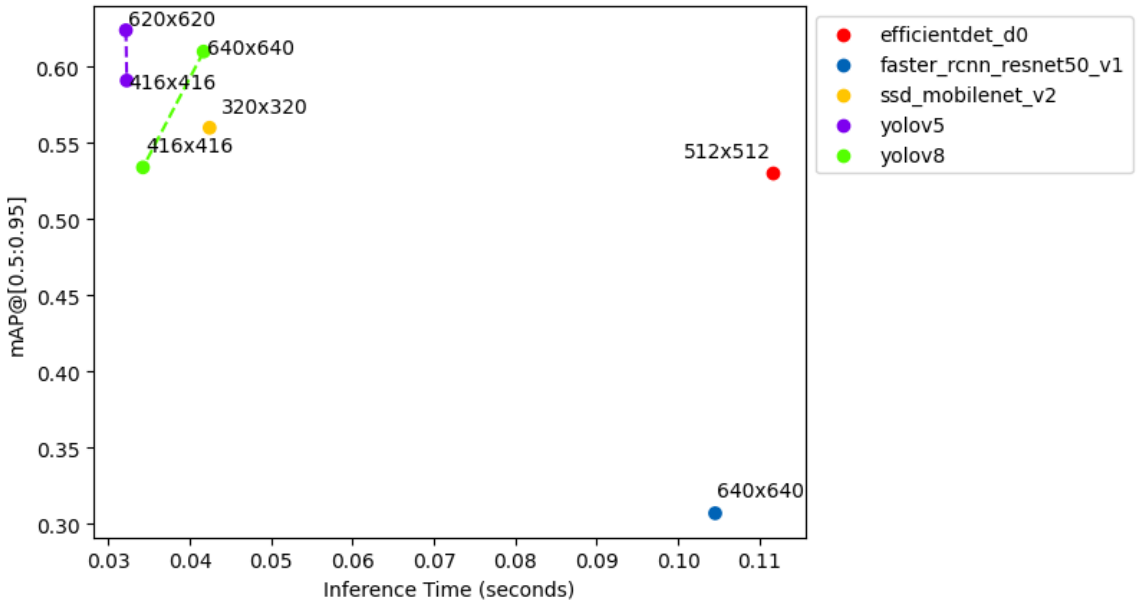


Figure 5.2: mAP versus Inference Time

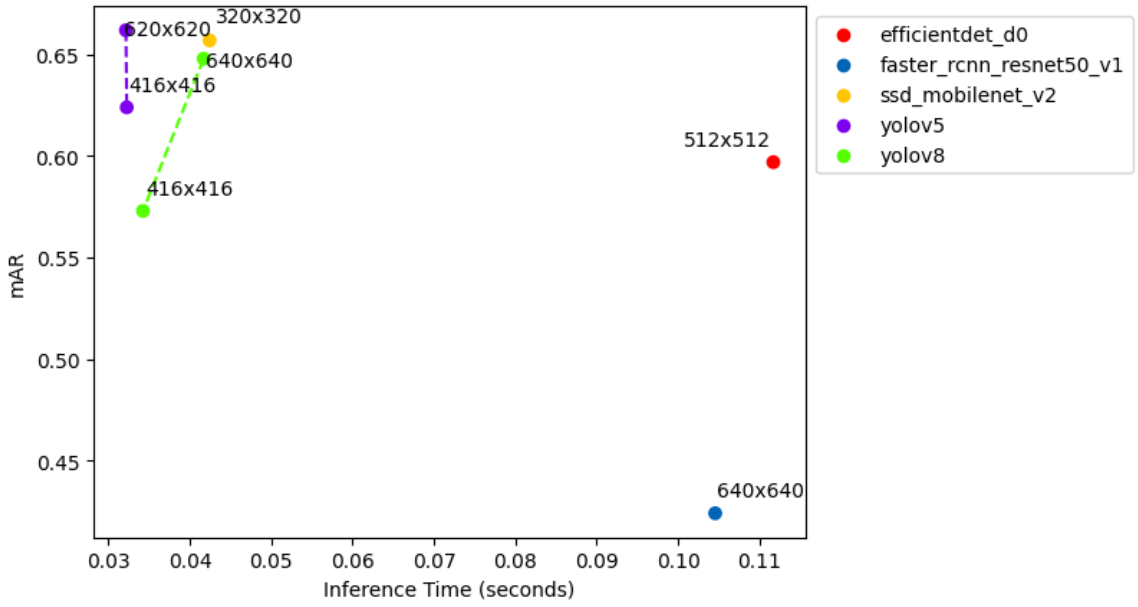


Figure 5.3: mAR versus Inference Time

Of the YOLO models tested, YOLOv5 outperformed equivalent YOLOv8 marginally in terms of mAP, mAR, and inference time. These findings reinforce YOLOv5 as the most appropriate for real-time detection on a drone. YOLOv5 not only offers greater mAP compared to YOLOv8, but also process images with the greatest throughput, meeting the essential criteria identified previously for real-time detection. For an example of predictions made on the validation set using YOLOv5, see Figure 5.4. Additionally, Figure 5.5 shows the confusion matrix with the percentage of predicted classes.

5.4 System Design

The overall design of the system was built with the intention of running on an edge device, enabling real-time object detection directly on a mobile platform without the need for constant communication with a cloud server. This approach not only

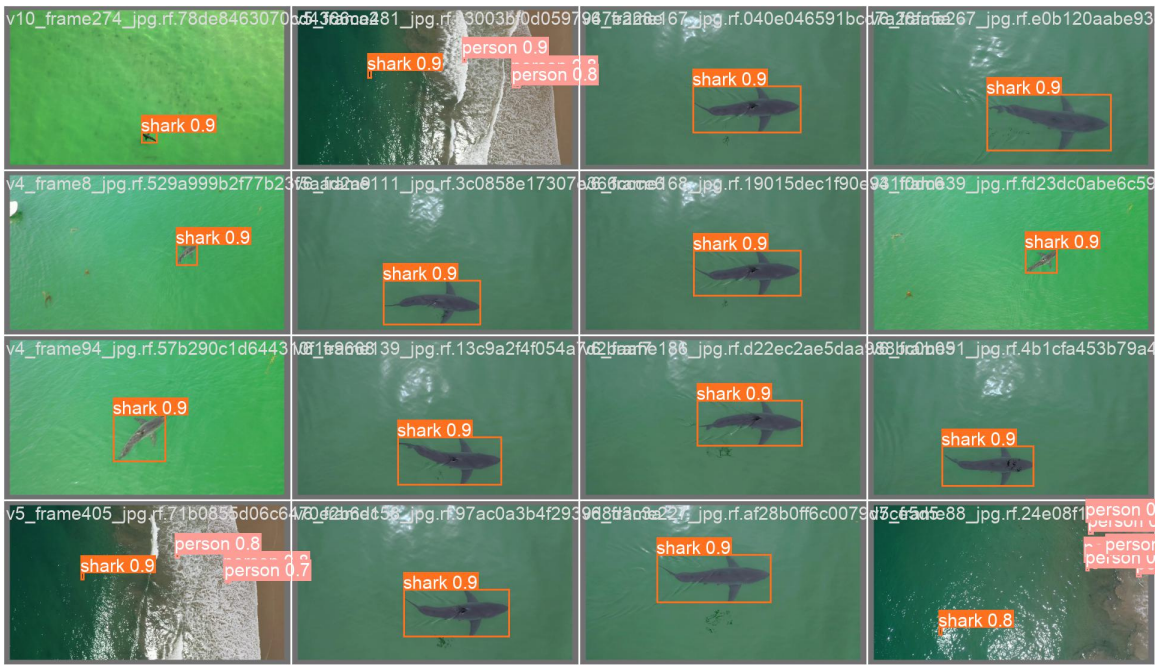


Figure 5.4: YOLOv5 Predictions on Validation Subset

optimizes for a lower latency but also aligns with the broader goal of utilizing the phone's neural accelerators for real-time on-device processing.

Due to constraints with memory and CPU utilization, metric monitoring was added to the system to provide insight into resource consumption post-flight. This allows for retroactive analysis of resource management, ensuring that the system operates within reasonable boundaries. Because this framework will be open-sourced, this type of feature will be useful in identifying potential bottlenecks, and can serve as the benchmark for future optimizations to enhance its real-time capabilities. The remaining metrics identified previously do not have an accessible API through iOS, so they had to be recorded using the Instruments application within XCode. This application allows for the monitoring of additional hardware components that aren't accessible through an open API.

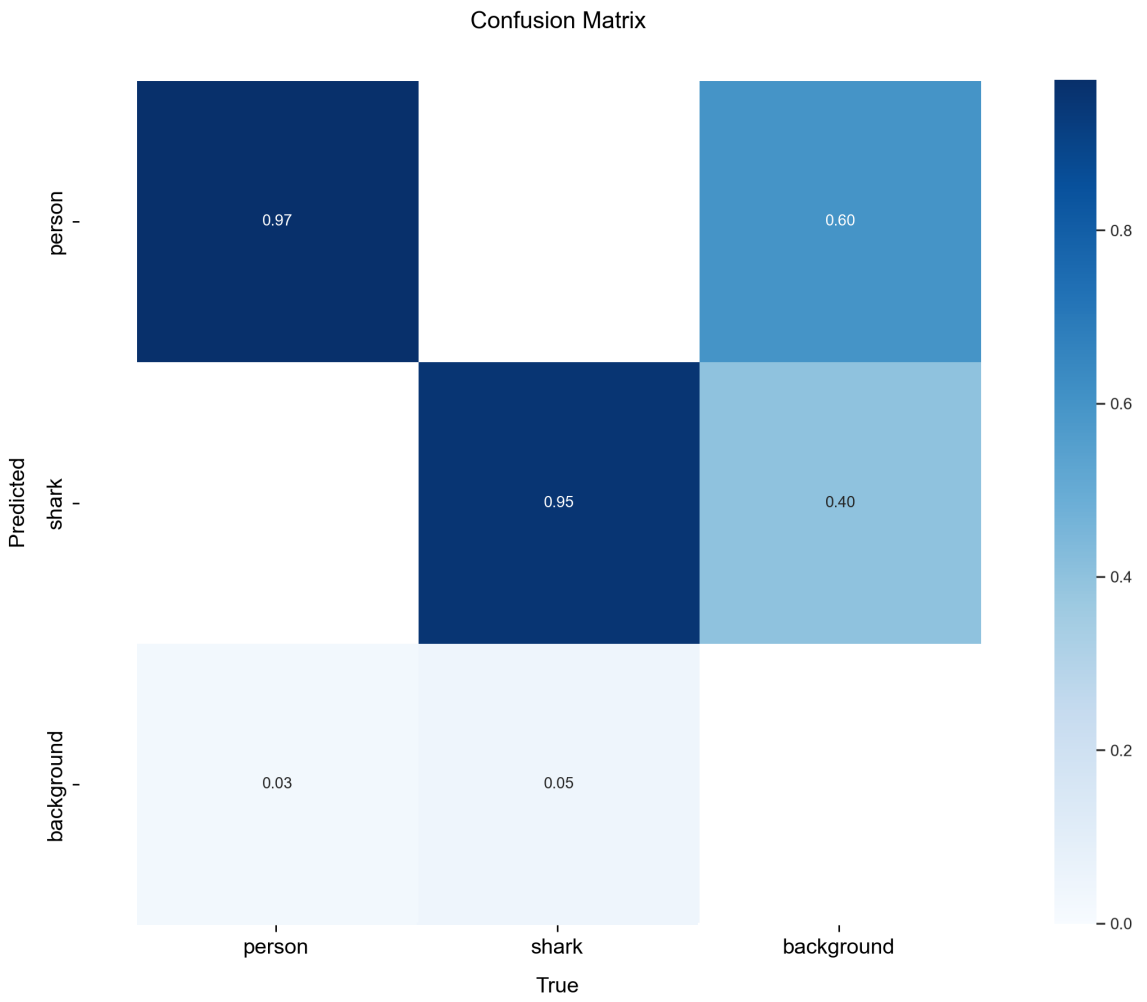


Figure 5.5: YOLOv5 640x640 Confusion Matrix

5.4.1 Requirements

The design of this framework has been guided by a set of requirements, used to ensure that it both meets overall goals and performs optimally on mobile hardware. The following sections outline the functional and non-functional requirements identified as crucial elements of the overall system design.

5.4.1.1 Functional Requirements

As mentioned in previous sections, the expectation for a real-time system is a minimum throughput of 30 FPS. Therefore, our system must be designed in such a way that allows for high rates of image processing and still can display resulting data to the screen.

Additionally, the user interaction element is essential because this system will be used by individuals not familiar with machine learning or programming in general. Therefore, the system should provide an intuitive interface that allows users to easily interact and receive feedback from the application.

5.4.1.2 Non-Functional Requirements

Several non-functional requirements were also selected as a basis for the construction of this framework. Namely, they are high performance and efficiency, stable metric monitoring, code reusability, and ease of maintenance.

Performance and efficiency are fundamental to handling real-time processing accurately, therefore extensive testing must be conducted in order to conclude if the performance of the application implies that it is viable for shark spotting.

Part of what makes this possible is being able to monitor the performance of the system using metric logging. It will be essential to record these metrics to the device for post-flight analysis, providing valuable insights into the performance of the system's components. Additionally, this is especially critical for discovering any hardware bottlenecks that may prevent the system from achieving desired performance.

Re-usability and ease of maintenance are two additional points that are critical for a complete system. The primary concern that this addresses is that this codebase will be reused by the Cal Poly Shark Spotting project. This not only means that the model should be hot-swappable, but also that the code must be easy to use and interpret.

5.4.2 High-level Design

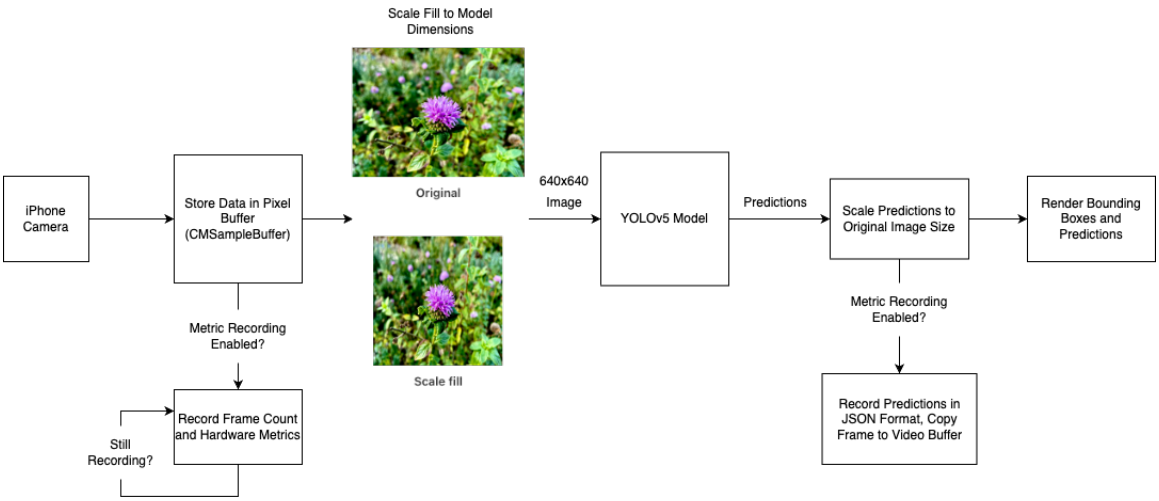


Figure 5.6: System Flowchart

The flowchart in Figure 5.6 shows a high level overview of how the final system was designed. There are two different modes of operation, standard use and metric logging.

Standard mode simply displays predictions and bounding boxes to the screen without the added overhead required for capturing a live video feed and recording metrics. This was designed with the aforementioned requirement of high performance in mind, with the possibility of a live alert system as potential future work.

Metric logging mode was an important addition to this system, as it addressed the need to monitor hardware metrics that was outlined in the non-functional require-

ments. Through the usage of this feature, it will be possible to conclude if both expected model performance and throughput meet expectations.

In addition, the user interface for this application was designed for simplicity. As seen in Figure 5.7, the top of the application is reserved for metric logging visible to the user while a button resides at the bottom of the screen to switch between modes of operation. When in use, the background will display real-time footage captured from the camera and render bounding boxes with their respective predictions to the screen.

5.4.3 Component Descriptions

As mentioned in the non-functional requirements, being able to swap a model with ease was an important feature for this system. The way this was achieved was by allowing for modification simply by adding the model file to the repository, and passing in the file name as a parameter to the detector class. This allows for ease of use for future developers, not only for larger models but also for a variety of different models.

Following any modifications to the selected model, the application handles the complete setup of the camera for regular usage, including making an access request, setting video resolution, and setting the maximum video framerate. In addition, the application handles the capturing of video, allowing for the starting and stopping of capture at any moment.

Once the application and model are fully loaded onto the phone, a subroutine executes on each frame stored in the pixel buffer. Before captured frames enter the pixel buffer, however, each must be scaled to fill the model's dimensions (640px by 640px for the aforementioned YOLOv5 model). Additionally, any hardware metrics are logged before the aforementioned subroutine is called and recorded to disk. The pixel buffer

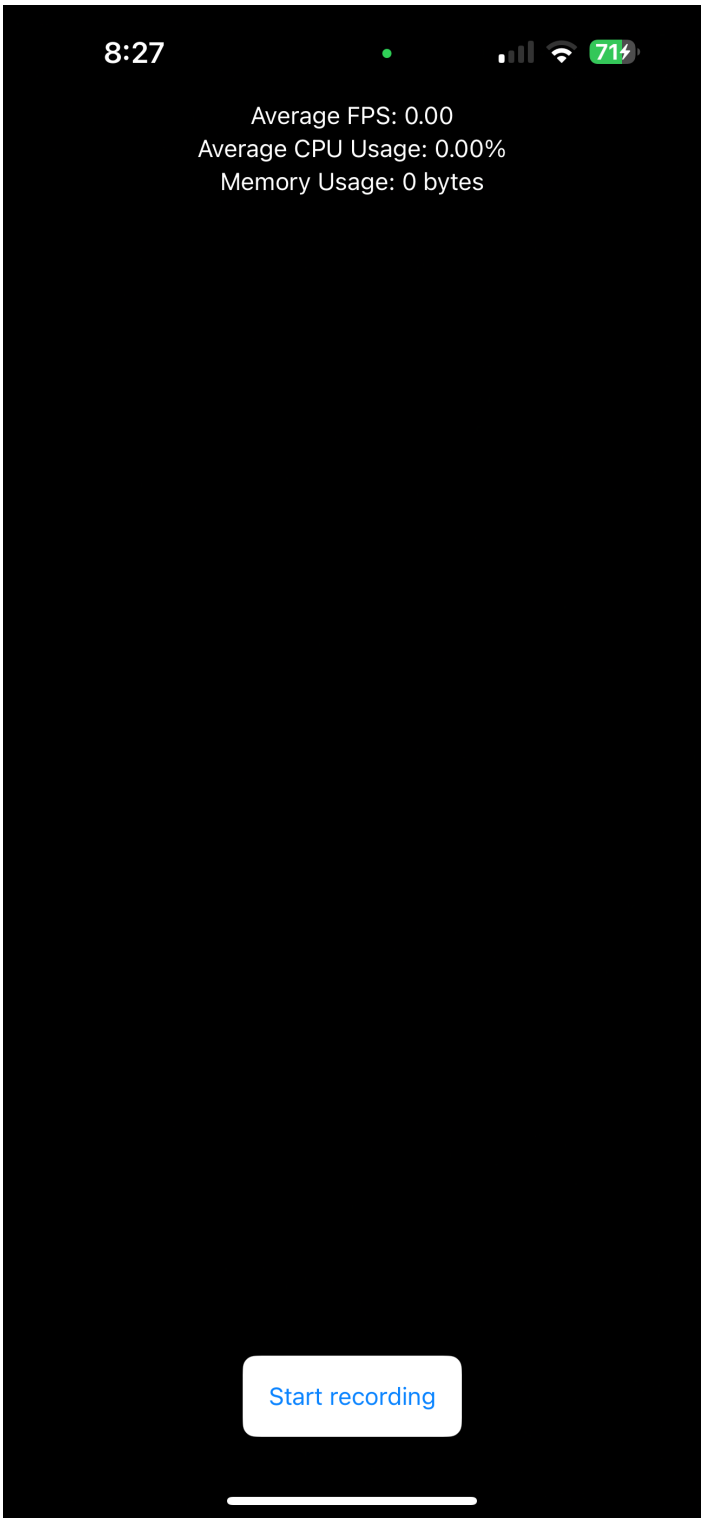


Figure 5.7: Application Interface

subroutine is responsible for inputting the resized image to the loaded object detection model on every frame, recording any results to disk if in metric logging mode. Any detections that are made must be scaled to the screen's dimensions, resulting in a labelled bounding box rendered to the device.

The detector is initialized within the app's main component, and the video feed from the camera is output to the device's screen using a video preview layer that automatically resizes the output to fit the screen's dimensions. Any hardware metrics that get recorded are displayed to the UI on a single second interval. The aim of this is to be non-obtrusive to the performance of the model, as all data points are recorded regardless for post-flight analysis.

Finally, the interface of this application was designed with the SwiftUI framework, making it simpler to design, iterate, and test the interface as quickly as possible. This choice of framework promotes code reusability, as SwiftUI is the modern choice over Storyboard to create Swift applications.

Chapter 6

IMPLEMENTATION

6.1 Technology Stack

The technology stack chosen for this project is a reflection of the current best practices in the field, aiming for efficient model training, quick and easy implementation to a mobile device, and modern application development.

The initial stage of this project involved training models to recognize and classify both people and sharks. For this purpose, the Jupyter Notebook was used due to its simple computing environment which allows for repeatable training, testing, and validation of models. The inherent support for iterative and interactive testing comes in the form of executable blocks of Python code, providing a method of re-running snippets of code without having to run the entire program again. This in turn allowed for much faster development, especially since machine learning tasks take significantly more time to execute than typical Python applications.

Subsequently, the PyTorch framework was utilized for training of the YOLOv5 based model due to its dynamic computational graph and ease of use. While TensorFlow has been the primary framework for machine learning and object detection for most research, recently there has been a shift to PyTorch based models.

The trained models were then converted to the Core ML model format for implementation, which is a the model standard of the Core ML framework created by Apple. Utilizing this framework allows for the deployment of the final model to any iOS device seamlessly, and allows for access to the built-in neural accelerator. Conversion

to a Core ML model was achieved through the usage of Apple's coremltools library built for Python, allowing for the translation of PyTorch models to Core ML models [4]. Part of this conversion required specification of inputs and outputs to ensure the model performed exactly the same as expected.

Taking advantage of the hardware capabilities of modern iOS devices was vital to ensuring the model could run in real-time on the phone. This both enhances the performance of the object detection tasks, and ensures a smoother experience by offloading these computations to dedicated hardware. In essence, this allows for other resources to be used in parallel with the neural accelerator while reducing the load on these same resources.

The development phase of the application was carried out using Swift on the XCode platform. A conscious choice was made to use SwiftUI for the user interface development instead of the traditional Storyboard. SwiftUI, being a modern UI framework developed by Apple, provides a simpler set of tools for building more complex user interfaces. Despite being a simpler UI, developing in Swift still allows for complex behavior behind-the-scenes.

To carry out real-time object detection, the Camera API AVCam was used to access the device's camera. This integration allows for immediate capture and processing of the live feed into an object detection model, enabling the primary functionality for this application.

The final important piece of technology to be selected was the smartphone. For this thesis, the iPhone 14 Pro was used for experimentation and final results. The choice of this model was guided by factors that align with the overarching project goals, including their modern hardware capabilities and advanced built-in camera system.

It also allows for future improvements to be made by leveraging additional hardware, such as the LiDAR sensor.

6.2 Setup of Experiment

To perform efficient validation testing on this system, a structured testing plan was designed. For these tests, three different configurations were tested using the YOLOv5n model identified previously. These three configurations were each based on the maximum throughput allowed by the system, 30 FPS, 60 FPS, and an uncapped framerate. Each configuration was then loaded to the device under test (DUT) and ran three trials over the course of ten minute intervals. From these tests, average metrics were recorded and averaged overall to get the resulting values.

The ANE, GPU, and battery usage metrics had to be recorded using external tools outside of the constructed application as no viable API exists to track these. The ANE and GPU could be recorded using XCode's Instruments software, designed for usage over direct connection between the computer and smartphone. Battery percentages had to be recorded and averaged over the ten minute runs, then extrapolated to determine an average percentage of battery loss per minute, as there were no other viable options within Instruments or an iOS API.

Chapter 7

RESULTS AND VALIDATION

7.1 Results

The performance metrics for the three different system configurations were evaluated and are summarized in Table 7.1.

These finding suggest that the system's utilization of the ANE depends on the maximum throughput of the system. There is a large increase in ANE utilization seen when transitioning from a frame rate of 30 FPS to 60 FPS, nearly doubling (a 114.0% increase) in usage. When the frame rate is uncapped, the ANE experiences a moderate increase in utilization of 14.5% from the 60 FPS system, which similarly aligns with the 12.7% increase in throughput observed between these configurations.

Furthermore, the contrasting behaviors in GPU and ANE utilization with respect to throughput show that there are still some complexities at play in managing the system's computational tasks. These complexities might stem from architectural designs or the distribution of computational workloads between the GPU and ANE. As the frame rate increases, the ANE appears to take on a larger share of the computational

Table 7.1: Metric Averages

Metric	$\text{system}_{\text{30fps}}$	$\text{system}_{\text{60fps}}$	$\text{system}_{\text{uncapped}}$
ANE Utilization (%)	14.92	31.93	36.56
GPU Utilization (%)	13.98	17.84	14.22
Memory Utilization (%)	1.05	1.07	1.06
CPU Utilization (%)	12.32	17.24	18.07
Throughput (FPS)	29.94	59.98	67.65
Battery Usage (%/min)	-0.23	-0.38	-0.43

burden, while the GPU's utilization trend is less straightforward, indicating a possible redistribution of tasks or a difference in workload handling capabilities between these components.

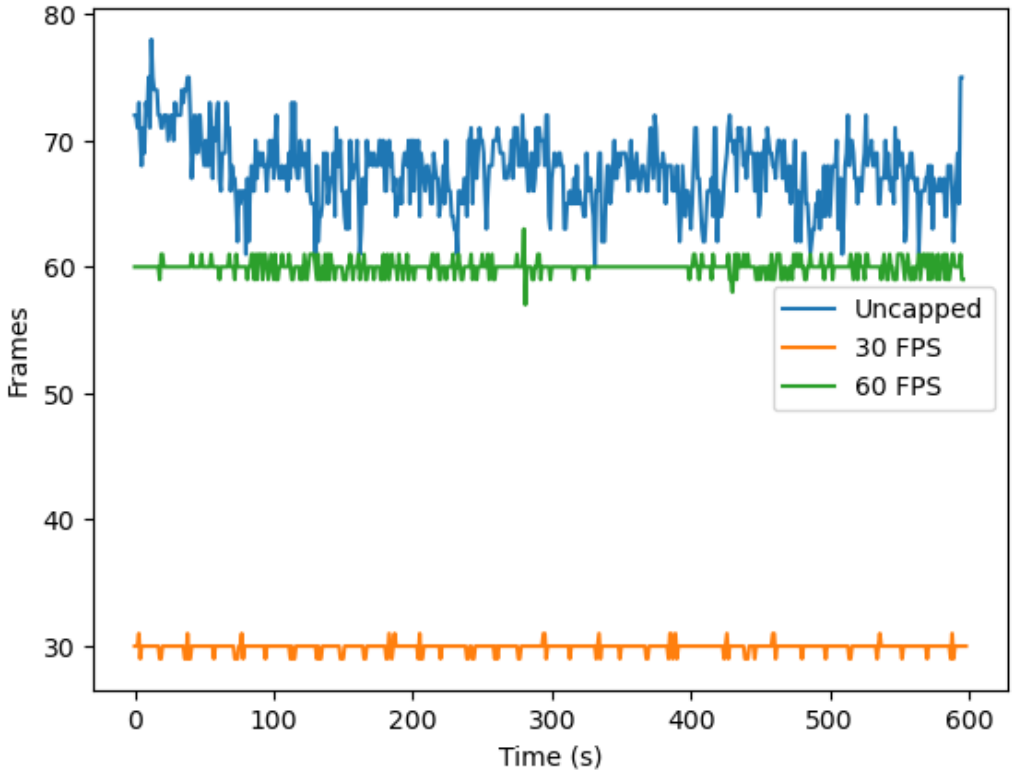
Contrasting this behavior, the CPU utilization follows a trend that more closely resembles that of the ANE. However, the percent increases between CPU utilizations do not match the throughputs measured at each configuration. This likely implies that the ANE and CPU share the computational burden of processing detection tasks as throughput increases.

The values shown in Table 7.1 reflect the extrapolated battery percentage loss per minute on average. As expected, these values become more negative as throughput increases. More negative values imply a greater velocity in battery loss, leading to lower final battery percentages than a less negative value.

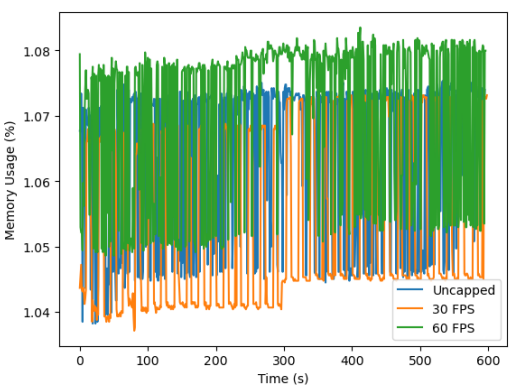
For metrics that were able to be captured during runtime, plots were created to display metric counts over the course of one second intervals.

The number of frames captured every second was recorded for each run and is shown in Figure 7.1a. The throughput measured for both the 30 FPS and 60 FPS configurations remains relatively stable around their respective framerates. When uncapped, this graph becomes significantly more noisy, ranging from lows of 60 to highs of 78 frames processed. Because of this uncertainty in throughput it is difficult to record footage at a specific framerate. The resulting footage may appear to be slightly sped up or slowed down due to this, but all frames will still be captured in the output.

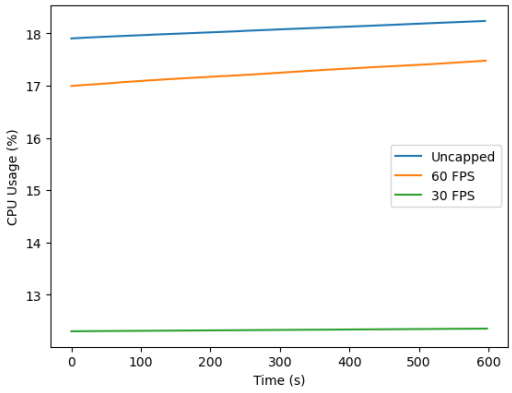
Figure 7.1b shows the memory utilization over time for each system. It is clearer to see here how noisy this signal is, and how close they are in values. The bytes of memory used for each run on average was 63.20 MB, 64.20 MB, and 63.86 MB for the 30 FPS, 60 FPS, and uncapped configurations respectively.



(a) Throughput



(b) Memory Utilization



(c) CPU Utilization

Figure 7.1: Metrics Over Time by Configuration

The CPU utilization by configuration over time is shown is shown in Figure 7.1c. This graph shows the utilizations of both the uncapped and 60 FPS configurations increase sharply initially, and taper off towards the end of the tests. The 30 FPS configuration, however, shows a steady utilization that doesn't increase by much over the course of the ten minute examination window.

7.2 Discussion

In terms of memory utilization, it is noteworthy that it remains relatively stable across each configuration. One reason for this low utilization could be because predictions are not stored in memory by our system, but rather dumped to a file as they are generated. Additionally, multiple tests were run to stress test the system for any memory leaks, with the results showing that the program does not leak any memory. This indicates that the memory subsystem is not a bottleneck for this system, despite varying framerates.

The method used to gauge battery usage, while necessary due to the lack of a more precise measurement tool, introduces some uncertainty into the data. A different approach involving more accurate measurements of thermals and usage could provide more insight into the actual usage across each configuration. Despite this, the trend of an increased battery drain on higher frame rates is intuitive and aligns with the other observed trends. Extrapolating this data, we predict that the system has an overall lifespan of 7.24, 4.39, and 3.88 hours for the 30 FPS, 60 FPS, and uncapped systems respectively. These times long outlast the life of drone batteries, like on the Skydio 2+, which can last up to 27 minutes on a full charge [47].

The 30 FPS configuration demonstrates a stable and predictable utilization of system resources, offering the lowest battery impact among the three configurations. It

achieves this at the expense of a lower throughput than the other tested configurations. Contrasting this, the system capped at 60 FPS provides some balance to resource utilization and throughput. The near doubling of ANE utilization suggests a linear scaling of detection tasks with frame rate, leading to a higher impact on battery life.

The final uncapped configuration was used as a stress test, to find the limits of the device under test. Ultimately, this configuration should not be needed or viable in its current state, as real-time devices will not require this boost in frame rate. Additionally, some performance hits were taken as the device experienced thermal throttling under longer tests, capping the throughput at an average of close to 30 FPS when overheated.

Between the remaining two configurations, the 30 FPS option appears to be the most sensible to pursue for UAV detection. Not only does it demonstrate minimal battery impact, but it satisfies the 30 FPS throughput minimum defined earlier for a real-time system. On top of this, the stability of its performance over many runs inspires confidence that it will operate within its known parameters. Additionally, the lower resource utilization exhibited allows for further exploration and expansion into additional computing tasks. Further opportunities for expansion on this research is outlined in the future work chapter.

7.2.1 Literature Comparison

Comparing these results to the ones obtained by prior researchers in shark detection contexts, we see a massive increase in maximum throughput to a system. Not only is our system able to consistently perform detection at over 60 FPS, but it performs at an accuracy slightly better than that of previous models.

Author	Year	Model	mAP	mAR	X_{sys} (FPS)
Our system	**2023**	**YOLOv5**	**0.973**	**0.924**	**67.65**
Gorkin et al. [18]	2020	YOLO-based	0.908[1]	NA	NA[2]
Sharma et al. [45]	2018	R-CNN	0.901	NA	7.69
Dujon et al. [16]	2021	Custom CNN-based	0.417	0.827	NA
Moore [35]	2021	SSD 300	0.296	0.385	NA
Mejias et al. [33]	2013	Custom Convolution	0.0497	0.514	0.109

Table 7.2: Comparison to UAV Marine Animal Detection Models

Additionally, comparing model accuracy to previous reported values, we see marginal increases over these other methods. The comparison of our model to other UAV-based shark spotting models is shown in Table 7.2.

From this table, it is clear that using a smartphone for onboard object detection with sharks is a viable alternative to previous methods, outperforming these in both accuracy and throughput metrics. The closest comparison to our model's accuracy is the YOLO-based model developed by Sharkeye. This provides a degree of confirmation that YOLO models are a competitive option for edge detection tasks in both accuracy and inference time.

Although many of these models appear to be very accurate, some with recorded precisions over 90%, they all will lack generalization on other locations and conditions due to limited training data available for sharks [10]. A potential solution to this issue is through "active learning," [10] discussed as future work. Another option that has been expressed in prior work is collaboration between researchers to share and combine data [35]. However, this may prove to be a more challenging approach because only a handful of organizations are currently building shark detection systems.

[2]Sharkeye's precision is based on the reported accuracies of 3 classes on a training set: sharks, stingrays, and surfers

[2]Reported as operating in real-time [18]

Chapter 8

FUTURE WORK

The results obtained provides ground for further development and research into the usage of smartphones for onboard object detection. This section outlines the possible avenues for future work to continue this research.

8.1 Model Improvements

A logical next step in the continuation of this research is further integration and testing of more finely tuned and larger models. For this thesis, the YOLOv5 nano model was trained and tested, showing that it can significantly outperform equivalent embedded systems. Using a model with a greater parameter count would greatly benefit this system, allowing for increasingly more accurate predictions. Using feature pruning on minimal impact nodes, as suggested by Liu et al., could provide even greater speeds on such larger models.

Looking into the viability of other object detection models and frameworks is yet another vital area of research for mobile detection. Expanding beyond simply one and two stage detectors, further research could include looking into the implementation of an RNN based model on iPhone hardware. Overall, exploring a broader spectrum of models offers increased versatility to the system, and will provide greater insight into speed and accuracy balances on mobile devices.

8.2 Occlusions

Dealing with complex object detection scenarios, such as those involving occlusions or three-dimensional interactions, can be difficult for traditional models that only produce bounding boxes. Methods for overcoming these challenges are currently being researched and developed, with some solutions readily available for integration in this system today.

The two major framework solutions that should be investigated are Mask R-CNN and Instance Segmentation in YOLOv8. Both of these models aim to solve the issue of occlusions by performing pixel-by-pixel segmentation of recognized objects [19]. Rather than simply drawing a bounding box over the region of interest, a predictor is used to determine which pixels are a part of the identified object. The resulting output is a "mask" ideally outlining the object that was detected. One of the major challenges of this endeavor will be the creation of a viable dataset, as none currently exist in the context of shark detection.

8.3 System Improvements

Taking advantage of all aspects of the smartphone is key in maximizing the potential usage of a mobile device for object detection. One hardware feature in the iPhone 14 Pro that wasn't used in this thesis is the LiDAR (Light Detection and Ranging) sensor. LiDAR is generally used for object distance, as it can emit a beam of light into the real world and detect its return to the sensor. With this added sense of distance, the system can be improved to estimate both the distance and size of detected sharks. This adds another dimension to threat detection, as smaller sharks may not pose as great of a threat to surfers as larger ones.

Additionally, taking full advantage of the iPhone's camera system would be helpful for creating a more accurate system. Currently only a 640x640 image is processed per frame, while the image resolution is far greater. Improvements to both the system and model to handle larger resolutions would theoretically provide greater accuracy measurements. On top of this, using the telephoto lens on the iPhone 14 Pro would allow drones to fly at a higher distance from the animals to be detected, or to zoom in on smaller objects for better detection results.

8.4 Object Tracking

Experimenting on the practicality of object tracking on mobile devices is a natural extension of this thesis, allowing for further research into techniques possible on smartphones. The primary difference between object detection and object tracking is the permanence of the object between detections. When identifying entities using classical object detection techniques, each processed frame is assumed to be individual with no relation between it and other frames. On the other hand, object tracking takes into account previous frames to track a specific object's position.

An object tracking model can be better suited for a variety of different tasks relating to shark spotting. For example, an object tracking model can enable continuous monitoring of specific shark movements near popular beaches. This could allow for better understanding of sharks' intentions by treating them as individual entities, enabling greater public safety as we further understand shark movement.

This expansion allows for further exploration into behavioral analysis of sharks through continuous monitoring of their patterns over time. This could include tracking their proximity to shore, interactions with other marine life, or response to differing envi-

ronmental conditions. Through this type of analysis, local authorities may be more prepared to inform beachgoers of potential safety hazards before they occur.

8.5 Active Learning

One of the largest limitations that has been a longstanding issue for researchers in this field is the lack of sufficient training data for sharks. A strategy proposed to mitigate this issue is the employment of "active learning." Active learning enables real-time feedback to iteratively improve an onboard model based on human confirmation or rejection of the generated predictions. This process can aid in reducing the initial amount of data required, as it will continue to learn as it is deployed in real-world settings.

Creating a system to support active learning on the phone is a challenge in itself, but recent research has shown promising results. For example, Brust et al. proposed a novel method of active learning for object detection, which combines incremental learning with uncertainty-based active learning metrics. Using the YOLO family of detectors, their approach showed improved performance over passive supervised learning methods on the PASCAL VOC 2012 dataset [8]. With similar techniques, it may be possible to develop a system for active learning on a mobile device that can improve overall accuracy with limited training data.

8.6 Vision Transformers

Vision transformers, or ViTs, are the latest evolution in computer vision in an attempt to move away from conventional CNN architectures. ViTs were first proposed in 2021, after transformers saw great success in natural language processing (NLP) tasks [15].

Rather than using a CNN to process the entire image at once, transformers are used to process sequences of flattened 2D image patches, where each patch is treated as a token. This approach has shown promising results, especially when pre-trained on large datasets and fine-tuned on smaller datasets [15].

Additionally, recent research into ViTs for mobile vision tasks has examined the viability of the architecture in an embedded setting. Apple researchers published a paper investigating such models in 2022, calling their system "MobileViT" [32]. Because ViT-based networks are not nearly as light-weight as their CNN counterparts, researchers used a hybrid model using both CNNs and ViTs for convolutions. This model is worth investigating because it reportedly can run in real-time on a mobile device, albeit worse than CNN-based models. Mehta et al. believe that inference speeds on MobileViT will improve further with dedicated device-level operations for transformers in the future [32].

Chapter 9

CONCLUSION

Among the many systems tried for shark detection, smartphone-based methods show significant potential based on these results. Not only do modern object detection frameworks match and exceed the capabilities of prior studies models', they also run at a much greater throughput on mobile devices. Through experimentation, we found that our model could perform object detection at 0.624 mAP@[0.5:0.95] (0.973 mAP@0.5), 0.662 mAR@[0.5:0.95] (0.924 mAR@0.5), at 67.65 FPS on an iPhone 14 Pro. In the worst possible scenario, we extrapolate from battery usage that our system would last 3.88 hours on a full charge. This means that our proposed detection system will outlast current leading drone batteries.

Through the work accomplished in this thesis, we conclude that:

1. Mobile devices are a viable alternative to traditional embedded systems, evident through the hardware efficiency and model accuracy studies.

2. A prototype was successfully developed that emphasized reusable code, efficient use of hardware, and an intuitive interface. On top of this, the code was made open-source for greater collaboration on this endeavour in the future.

3. The overall system was appropriately evaluated on a number of hardware and software metrics, giving a holistic understanding of its operating power.

9.1 Limitations

One of the primary limitations of performing object detection on this system is that it still has accuracy and recall scores on the lower end, especially for smaller objects. This is likely happening for a number of reasons, namely that this is a longstanding issue with YOLO, and our input image size is on the smaller end. This issue is most evident when dealing with detections performed at a farther distance. Longer flight distances could occur due to restrictions with flying drones over endangered species, for example. As discussed in the future work section, this can be mitigated by looking into higher input resolution or taking advantage of the telephoto lens on the iPhone 14 Pro's camera system.

Another limitation of this system is that it does not have enough training data on varying conditions to make it viable in all real-world scenarios. The dataset used contains images of sharks during the day and under a single environmental color palette. Elevating our model to perform better under differing conditions may look like re-training frequently with new data, or investing in the creation of an active learning model that iteratively improves itself during flight.

9.2 Concluding Remarks

While there is still room for improvement in small object detection, occlusions, and dataset expansion, having an open-source framework for object detection on a smartphone opens up a novel avenue for coastal safety and public awareness. The ability to harness computing power on an easily accessible and integrated device allows for further use in areas that may not have access to specialized hardware or equipment, or do not have the resources to afford the infrastructure setup. Furthermore, the

open-source nature of this system allows for future collaboration and continuous improvement from specialists. Future researchers, developers, and marine biologists can contribute to refine the system's capabilities. Ultimately, as smartphone technology continues to advance, so will the efficiency and accuracy of mobile detection systems.

BIBLIOGRAPHY

[1] K. Adams, A. Broad, D. Ruiz-García, and A. R. Davis. Continuous Wildlife Monitoring Using Blimps as an Aerial Platform: A Case Study Observing Marine Megafauna. *Australian Zoologist*, 40(3):407–415, May 2020. https://doi.org/10.7882/AZ.2020.004.

[2] A. Ahmadinia and J. Shah. An Edge-based Real-Time Object Detection. In *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, page 465–470, December 2022. https://doi.org/10.1109/ICMLA55696.2022.00075.

[3] W. Andrew, C. Greatwood, and T. Burghardt. Visual Localisation and Individual Identification of Holstein Friesian Cattle via Deep Learning. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, page 2850–2859, 2017. https://openaccess.thecvf.com/content_ICCV_2017_workshops/w41/html/Andrew_Visual_Localisation_and_ICCV_2017_paper.html.

[4] Apple. Core ML Tools, 2023. https://github.com/apple/coremltools.

[5] M. S. Aslanpour, S. S. Gill, and A. N. Toosi. Performance Evaluation Metrics for Cloud, Fog and Edge Computing: A Review, Taxonomy, Benchmarks and Standards for Future Research. *Internet of Things*, 12:100273, December 2020. https://doi.org/10.1016/j.iot.2020.100273.

[6] J. Bao and Q. Xie. Artificial Intelligence in Animal Farming: A Systematic Literature Review. *Journal of Cleaner Production*, 331:129956, January 2022. https://doi.org/10.1016/j.jclepro.2021.129956.

[7] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao. YOLOv4: Optimal Speed and Accuracy of Object Detection. *arXiv:2004.10934 [cs.CV]*, 2020. http://arxiv.org/abs/2004.10934.

[8] C.-A. Brust, C. Käding, and J. Denzler. Active Learning for Deep Object Detection. *arXiv:1809.09875 [cs.CV]*, 2018. http://arxiv.org/abs/1809.09875.

[9] A. Bulut, F. Ozdemir, Y. S. Bostanci, and M. Soyturk. Performance Evaluation of Recent Object Detection Models for Traffic Safety Applications on Edge. In *Proceedings of the 2023 5th International Conference on Image Processing and Machine Vision*, IPMV '23, page 1–6, New York, NY, USA, March 2023. Association for Computing Machinery. https://dl.acm.org/doi/10.1145/3582177.3582178.

[10] P. A. Butcher et al. The Drone Revolution of Shark Science: A Review. *Drones*, 5(11):8, March 2021. https://doi.org/10.3390/drones5010008.

[11] A. B. Carlisle and R. M. Starr. Habitat Use, Residency, and Seasonal Distribution of Female Leopard Sharks Triakis Semifasciata in Elkhorn Slough, California. *Marine Ecology Progress Series*, 380:213–228, April 2009. https://doi.org/10.3354/meps07907.

[12] P. Chamoso, W. Raveane, V. Parra, and A. González. UAVs Applied to the Counting and Monitoring of Animals. In *Ambient Intelligence - Software and Applications*, Advances in Intelligent Systems and Computing, page 71–80, Cham, 2014. Springer International Publishing. https://doi.org/10.1007/978-3-319-07596-9_8.

[13] E. Corcoran et al. Automated Detection of Koalas Using Low-level Aerial Surveillance and Machine Learning. *Scientific Reports*, 9(11):3208, March 2019. https://doi.org/10.1038/s41598-019-39917-5.

[14] J. Cowton, I. Kyriazakis, and J. Bacardit. Automated Individual Pig Localisation, Tracking and Behaviour Metric Extraction Using Deep Learning. *IEEE Access*, 7:108049–108060, 2019. https://doi.org/10.1109/ACCESS.2019.2933060.

[15] A. Dosovitskiy et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *arXiv:2010.11929 [cs.CV]*, 2021. http://arxiv.org/abs/2010.11929.

[16] A. M. Dujon et al. Machine Learning To Detect Marine Animals in UAV Imagery: Effect of Morphology, Spacing, Behaviour and Habitat. *Remote Sensing in Ecology and Conservation*, 7(3):341–354, 2021. https://doi.org/10.1002/rse2.205.

[17] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, page 580–587, 2014. https://openaccess.thecvf.com/content_cvpr_2014/html/Girshick_Rich_Feature_Hierarchies_2014_CVPR_paper.html.

[18] R. Gorkin et al. Sharkeye: Real-Time Autonomous Personal Shark Alerting via Aerial Surveillance. *Drones*, 4(22):18, June 2020. https://doi.org/10.3390/drones4020018.

[19] K. He, G. Gkioxari, P. Dollar, and R. Girshick. Mask R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision*, page

2961–2969, 2017. https://openaccess.thecvf.com/content_iccv_2017/html/He_Mask_R-CNN_ICCV_2017_paper.html.

[20] T. Hollings et al. How Do You Find the Green Sheep? A Critical Review of the Use of Remotely Sensed Imagery To Detect and Count Animals. *Methods in Ecology and Evolution*, 9(4):881–892, 2018. https://doi.org/10.1111/2041-210X.12973.

[21] J. Hosang, R. Benenson, P. Dollár, and B. Schiele. What Makes for Effective Detection Proposals? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(4):814–830, April 2016. https://doi.org/10.1109/TPAMI.2015.2465908.

[22] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv:1704.04861 [cs.CV]*, 2017. http://arxiv.org/abs/1704.04861.

[23] P. Jaccard. Étude Comparative de la Distribution Florale Dans une Portion des Alpes et des Jura. *Bull Soc Vaudoise Sci Nat*, 37:547–579, 1901. https://doi.org/10.5169/seals-266450.

[24] Z. Jiang, L. Zhao, S. Li, and Y. Jia. Real-time Object Detection Method Based on Improved YOLOv4-tiny. *arXiv:2011.04244 [cs.CV]*, 2020. http://arxiv.org/abs/2011.04244.

[25] G. Jocher. YOLOv5 by Ultralytics, May 2020. https://github.com/ultralytics/yolov5. Accessed 2022-8-30.

[26] G. Jocher, A. Chaurasia, and J. Qiu. YOLO by Ultralytics, January 2023. https://github.com/ultralytics/ultralytics. Accessed 2022-8-30.

[27] K. Li, Y. Wang, and Z. Hu. Improved YOLOv7 for Small Object Detection Algorithm Based on Attention and Dynamic Convolution. *Applied Sciences*, 13(1616):9316, January 2023. https://doi.org/10.3390/app13169316.

[28] Y. Li et al. A Real-Time Edge-AI System for Reef Surveys. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, MobiCom '22, page 903–906, New York, NY, USA, October 2022. Association for Computing Machinery. https://dl.acm.org/doi/10.1145/3495243.3558278.

[29] H. Liu, K. Fan, Q. Ouyang, and N. Li. Real-Time Small Drones Detection Based on Pruned YOLOv4. *Sensors*, 21(1010):3374, January 2021. https://doi.org/10.3390/s21103374.

[30] W. Liu et al. SSD: Single Shot MultiBox Detector. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *Computer Vision – ECCV 2016*, Lecture Notes in Computer Science, page 21–37, Cham, 2016. Springer International Publishing. https://doi.org/10.1007/978-3-319-46448-0_2.

[31] F. Maire, L. M. Alvarez, and A. Hodgson. Automating Marine Mammal Detection in Aerial Images Captured During Wildlife Surveys: A Deep Learning Approach. In B. Pfahringer and J. Renz, editors, *AI 2015: Advances in Artificial Intelligence*, Lecture Notes in Computer Science, page 379–385, Cham, 2015. Springer International Publishing. https://doi.org/10.1007/978-3-319-26350-2_33.

[32] S. Mehta and M. Rastegari. MobileViT: Light-weight, General-purpose, and Mobile-friendly Vision Transformer. *arXiv:2110.02178 [cs.CV]*, 2022. http://arxiv.org/abs/2110.02178.

[33] L. Mejias, G. Duclos, A. Hodgson, and F. Maire. Automated Marine Mammal Detection From Aerial Imagery. In *2013 OCEANS - San Diego*, page 1–5, September 2013. https://ieeexplore.ieee.org/abstract/document/6741088.

[34] H. Meko. After Queens Shark Bite, Drones Buzz Over Unfazed Beachgoers. *The New York Times*, August 2023. https://www.nytimes.com/2023/08/13/nyregion/rockaway-beach-shark-bite.html. Accessed 2022-8-30.

[35] D. Moore. Towards On-Device Detection of Sharks with Drones. Master's thesis, California Polytechnic State University, December 2021. https://digitalcommons.calpoly.edu/theses/2370/.

[36] R. Padilla et al. A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit. *Electronics*, 10(33):279, January 2021. https://doi.org/10.3390/electronics10030279.

[37] R. Padilla, S. L. Netto, and E. A. B. da Silva. A Survey on Performance Metrics for Object-Detection Algorithms. In *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, page 237–242, 2020. https://ieeexplore-ieee-org.calpoly.idm.oclc.org/abstract/document/9145130.

[38] Z. Qin et al. Thundernet: Towards Real-Time Generic Object Detection on Mobile Devices. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, page 6718–6727, 2019. https://openaccess.thecvf.com/content_ICCV_2019/html/Qin_ThunderNet_Towards_Real-Time_Generic_Object_Detection_on_Mobile_Devices_ICCV_2019_paper.html.

[39] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, page 779–788, 2016. https://www.cv-foundation.org/openaccess/content_cvpr_2016/html/Redmon_You_Only_Look_CVPR_2016_paper.html.

[40] J. Redmon and A. Farhadi. YOLOv3: An Incremental Improvement. *arXiv:1804.02767 [cs.CV]*, 2018. http://arxiv.org/abs/1804.02767.

[41] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. https://proceedings.neurips.cc/paper_files/paper/2015/hash/14bfa6bb14875e45bba028a21ed38046-Abstract.html.

[42] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018. https://openaccess.thecvf.com/content_cvpr_2018/html/Sandler_MobileNetV2_Inverted_Residuals_CVPR_2018_paper.html.

[43] F. Sarwar et al. Detecting and Counting Sheep with a Convolutional Neural Network. In *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, page 1–6, November 2018. https://ieeexplore.ieee.org/abstract/document/8639306.

[44] A. C. Seymour et al. Automated Detection and Enumeration of Marine Wildlife Using Unmanned Aircraft Systems (UAS) and Thermal Imagery. *Scientific Reports*, 7(11):45127, March 2017. https://doi.org/10.1038/srep45127.

[45] N. Sharma, P. Scully-Power, and M. Blumenstein. Shark Detection from Aerial Imagery Using Region-Based CNN, a Study. In T. Mitrovic, B. Xue, and X. Li, editors, *AI 2018: Advances in Artificial Intelligence*, Lecture Notes in Computer Science, page 224–236, Cham, 2018. Springer International Publishing. https://doi.org/10.1007/978-3-030-03991-2_23.

[46] S. Singh, I. Chana, M. Singh, and R. Buyya. SOCCER: Self-Optimization of Energy-efficient Cloud Resources. *Cluster Computing*, 19(4):1787–1800, December 2016. https://doi.org/10.1007/s10586-016-0623-4.

[47] Skydio. Skydio 2+ Battery, 2023. https://shop.skydio.com/products/skydio-2-plus-battery. Accessed 2022-8-30.

[48] M. Tan and Q. Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning*, page 6105–6114. PMLR, May 2019. https://proceedings.mlr.press/v97/tan19a.html.

[49] M. Tan, R. Pang, and Q. V. Le. EfficientDet: Scalable and Efficient Object Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, page 10781–10790, 2020. https://openaccess.thecvf.com/content_CVPR_2020/html/Tan_EfficientDet_Scalable_and_Efficient_Object_Detection_CVPR_2020_paper.html.

[50] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao. YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, page 7464–7475, 2023. https://openaccess.thecvf.com/content/CVPR2023/html/Wang_

YOLOv7_Trainable_Bag-of-Freebies_Sets_New_State-of-the-Art_for_Real-Time_Object_Detectors_CVPR_2023_paper.html.

[51] R. J. Wang, X. Li, and C. X. Ling. Pelee: A Real-Time Object Detection System on Mobile Devices. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. https://proceedings.neurips.cc/paper_files/paper/2018/hash/9908279ebbf1f9b250ba689db6a0222b-Abstract.html.

[52] B. Xu et al. Automated Cattle Counting Using Mask R-CNN in Quadcopter Vision System. *Computers and Electronics in Agriculture*, 171:105300, April 2020. https://doi.org/10.1016/j.compag.2020.105300.

[53] H. Yu et al. TensorFlow Model Garden, 2020. https://github.com/tensorflow/models. Accessed 2022-8-30.

[54] Z. Zou et al. Object Detection in 20 Years: A Survey. *Proceedings of the IEEE*, 111(3):257–276, March 2023. https://doi.org/10.1109/JPROC.2023.3238524.