# TOWARDS ON-DEVICE DETECTION OF SHARKS WITH DRONES

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

Daniel Moore

December 2021

COMMITTEE MEMBERSHIP

TITLE: Towards On-Device Detection of Sharks with Drones

AUTHOR: Daniel Moore

DATE SUBMITTED: December 2021

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

COMMITTEE MEMBER: Maria Pantoja, Ph.D.
Professor of Computer Science

COMMITTEE MEMBER: Jonathan Ventura, Ph.D.
Professor of Computer Science

ABSTRACT

Towards On-Device Detection of Sharks with Drones

Daniel Moore

Recent years have seen several projects across the globe using drones to detect sharks, including several high profile projects around alerting beach authorities to keep people safe. However, so far many of these attempts have used cloud-based machine learning solutions for the detection component, which complicates setup and limits their use geographically to areas with internet connection. An on-device (or on-controller) shark detector would offer greater freedom for researchers searching for and tracking sharks in the field, but such a detector would need to operate under reduced resource constraints. To this end we look at SSD MobileNet, a popular object detection architecture that targets edge devices by sacrificing some accuracy. We look at the results of SSD MobileNet in detecting sharks from a data set of aerial images created by a collaboration between Cal Poly and CSU Long Beach's Shark Lab. We conclude that SSD MobileNet does suffer from some accuracy issues with smaller objects in particular, and we note the importance of customized anchor box configuration.

# ACKNOWLEDGMENTS

Thanks to Dr. Franz Kurfess for his patience in my thesis journey. Thanks to Michael Devin, who also takes everything apart. And thanks to Lucas Moore, who changed the way I look at an almond. Your support has meant a lot to me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

# Chapter 1

# INTRODUCTION

Shark populations off the California coast have increased in recent years [30][7], a successful result of protective measures. However, this brings with it increased potential for human-shark interactions, including shark bites. Protecting beach-goers in an ecologically-friendly way is an ongoing challenge for countries across the world that are home to shark populations. However, modern UAV technology may offer a less destructive alternative to more traditional culling strategies.

UAVs (unmanned aerial vehicles)—or drones as they are colloquially referred to—are revolutionizing the ability of beach communities with sharks to maintain safe beaches through the implementation of real-time shark notification systems that alert lifeguards when sharks are present in the water [8]. Though still an area of research, these systems hold promise for aiding beach authorities in identifying shark species and discriminating between sharks that may pose danger to humans and those that do not [8]. Not only have drones become useful for real-time shark beach alerts, they also have greatly expanded opportunities for shark research. While traditional methods such as autonomous underwater vehicles (AUVs), remotely operated vehicles (ROVs), shark cages, and observation from boats offer some visibility into shark activity, many of these methods of data collection involve close proximity with sharks in a way that is known to alter their behavior [8]. Drones offer an ideal birds-eye-viewpoint for observing sharks, and their small size and low noise output make them unlikely to influence shark activity [8].

While the task of detecting sharks for beach safety can be done by patrolling a fixed area of water surrounding a beach, using a drone to find and study sharks for research is more exploratory in nature. Depending on the drone and controller, this may involve searching for sharks on a small screen embedded in the controller or attached tablet while flying. Sometimes sharks are missed during flight but later noticed during viewing of the recorded video on a larger screen [34]. Though drone flight times are on the order of 12-40 minutes (which can limit tracking) [12], multiple drones or a battery charger can enable several hours worth of flying [8]. On a small controller screen, after several hours of searching, pilot fatigue may affect their ability to locate sharks for study [8]. In this scenario, real-time shark detection could also be useful in helping scientist drone pilots find sharks in the field. To be useful to a pilot, and because shark research observations may take place at remote locations far from an internet connection, shark detection processing would need to take place on the drone itself or on the controller/tablet. Such a system, even if operating at limited accuracy, could aid shark researchers in their search for sharks.

However, detection in these scenarios would require a machine learning model capable of performing reasonably well with limited computing resources. In this work we look at the performance of SSD MobileNet [29][22], a lightweight object detection model, as a candidate for on-device shark detection.

# Chapter 2

# RELATED WORK

## 2.1 Drones and Shark Research

Marine biologists use a variety of tools for studying sharks in the wild including satellite tags, recoverable biologgers, underwater video systems, acoustic telemetry, animal-borne cameras and aerial video capture to name a few [10]. Aerial methods, including manned aircraft, drones, and balloons are useful for doing population surveys over a large area or for studying animal behavior from above [10]. Though sharks spend much of their time in deep water not visible from the air, aerial studies are still useful for analyzing behavior of larger species in coastal or shallower ecosystems [10]. Additionally, there have been a few attempts to identify via aerial study the intersection of areas of high shark activity with areas of high human activity in order to better protect both sharks and humans from mutually harmful interactions [10].

As drones have become more popular, they offer a cheaper alternative to manned aircraft surveys. UAVs are also safer than manned aircraft, something made poignant by the death of at least 11 marine mammal researchers in aerial survey airplane crashes over the last several decades [19]. UAVs come in a wide variety of capacities from expensive military-grade drones with a large range and a flight time of a couple days to consumer-grade devices with localized ranges and flight times less than an hour [10]. Larger drones have the same benefits of manned aircraft—larger area coverage and ability to fly further offshore. Rotored drones have the ability to fly in a stationary position or to follow slow-moving animals for behavioral analysis, enabling aerial observation of feeding activities, socializing, interspecies interaction,

and other behaviors in a way that is impossible for manned surveys by plane. Drones are also likely to be substantially less noisy than manned aircraft, with a potentially lower impact on the behavior of the animals under study [10]. UAV data can also be combined with data from other tracking sources to supplement more traditional means of gathering data. For an exploration of the advantages of using drones to study sharks, see Butcher et al. [8].

In aerial and other types of studies, video may be analyzed in real-time or saved for later analysis [8]. When analyzing after-the-fact, the task of manually counting hundreds of sharks in video footage is a daunting one, as well as going through large time periods of footage to identify segments worthy of attention by researchers. To speed up these efforts, researchers have turned to computer vision techniques [10].

## 2.2 Deep Learning and Drones

Since the the Deep Convolutional Neural Network (DCNN) AlexNet [24] emerged as victor in the Large Scale Visual Recognition Challenge (ILSRVC) [35] in 2012, the computer vision community has shifted focus from handcrafted feature extraction towards deep learning [28]. DCNN architectures have not only the advantage of improved accuracy, but they also require less domain-specific knowledge. As computer vision research has evolved, research into new applications of domain-specific features has been largely supplanted by inquiry into new network architectures and new network training regimens [28].

In the last 5 years or so, three of the most popular computer vision architecture families that have emerged are Faster-RCNN (Faster Regions with Convolutional Neural Net features) [33], YOLO (You Only Look Once) [32], and SSD (Single Shot MultiBox Detector) [29]. Faster-RCNN represents a more accurate two-stage family

**Table 2.1: Comparison by Hossain et al. [21] of the speed of various object detection architectures on UAVs equipped with different NVIDIA Jetson modules and one UAV streaming to a machine on the ground**

| | TX1 | TX2 | Xavier AGX | GPU-Based Ground Station: Gtx 1080 |
|---|---|---|---|---|
| YOLOv2 | 2.9 FPS | 7 FPS | 26~30 FPS | 28 FPS |
| YOLOv2 tiny voc | 6~7 FPS | 15~16 FPS | 29 FPS | 30+ FPS |
| YOLOv3 | — | 3 FPS | 16~18 FPS | 15.6 FPS |
| YOLOv3 tiny | 9~10 FPS | 12 FPS | 30 FPS | 30+ FPS |
| SSD | 8 FPS | 11~12 FPS | 35~48 FPS | 32 FPS |
| DeepLab-v3 Semantic Segmentation | – | 2.2~2.5 FPS | 10 FPS | 15~16 FPS |
| Faster R-CNN | | 0.9 FPS | 1.3 FPS | – |
| Mask R-CNN | – | – | – | 2~3 FPS |
| YOLOv3 + Deep SORT | – | 2.20 FPS | 10 FPS | 13 FPS |

of detectors that first proposes candidate regions and then refines them. YOLO and SSD represent the family of "single shot" detectors, which generate detections in a dense manner across an entire image in a single stage. Faster-RCNN is a common "go-to" solution for computer vision tasks, while YOLO and SSD are often used in scenarios requiring near-real-time speed at the expense of some accuracy. These characteristics make YOLO and SSD likely candidates for running on-board a UAV device.

Using deep learning object detection with a drone in real-time can be done either on the UAV device itself or on a ground station computer receiving a video stream from the drone. Hossain et al. [21] compared the performance of various computer vision architectures on several UAV configurations, including UAVs with 1) on-device detection with a NVIDIA Jetson embedded GPU, 2) on-device detection with a non-GPU processor (such as a Raspberry Pi) augmented with an Intel Neural Compute Stick (NCS), and 3) off-device detection in which data is streamed to a ground station where detection is performed with a full video card. Table 2.1 on page 5and Table 2.2 on page 6show the results of their comparison. Note that SSD performs at the highest speed in both tables.

**Table 2.2: Comparison by Hossain et al. [21] of the speed of YOLO and SSD MobileNet on various GPU-less UAVs, mostly augmented by a Movidius Neural Compute Stick (NCS). For info on the difference between SSD MobileNet and plain SSD see Background section.**

| Systems | YOLO | SSD Mobile Net |
|---|---|---|
| Movidius NCS + Raspberry Pi | 1 FPS | 5 FPS |
| Movidius NCS + Latte Panda | 1.8 FPS | 5.5~ 5.7 FPS |
| Movidius NCS + Odroid | 2.10 FPS | 7~8 FPS |
| Just Odroid without Movidius NCS | – | 1.4 FPS |

### 2.3 Deep Learning and Sharks

As both drones and deep learning have gained popularity, there have been several applications of deep learning computer vision architectures to the detection of sharks. Saqib and Sharma et al. compared results of Faster-RCNN object detection of aerial shark images with various base networks in two studies [36][37]. More recently, the SharkEye project—a joint effort between SalesForce, the Benioff Ocean Initiative from the University of California Santa Barbara, and San Diego State University—used drones with regular flight paths to collect video data and stream to SalesForce's Einstein Vision system in order to report the number of sharks in real-time to beachgoers and scientists [4]. Another project by the University of Wollongong in Australia, also named Sharkeye, similarly provides real-time detection to smart watches and phones of beachgoers and lifeguards [17]. This system works by streaming footage from a blimp and drones to a beach laptop ground station, which then sends the data to a YOLO-based [32] machine learning pipeline in the cloud. A computer monitor at the ground station with a view of the blimp's camera feed was noted as giving lifeguards an increased ability to respond to people in trouble even without the object detection [17]. Besides the above, other projects include SharkSpotter,

a partnership between Ripper Corp. and abovementioned Dr. Nabin Sharma at the University of Technology Sydney [5].Finally, New South Wales' Department of Primary Industries' SharkSmart program [3] is worth mentioning because of their use of drones to monitor sharks, but we were unable to find evidence of machine learning use on their website.

Most of the shark detection solutions so far have run the machine learning portion of the solution on a server or in the cloud as opposed to on the drone itself [4][17]. This is obviously optimal from a computing perspective and also avoids modifying the drone (which can have implications on FAA compliance), taking advantage of the drone's built in streaming capabilities. After their trials however, Gorkin et al. expressed a desire to "explore where it makes sense to do the process computing 'at the edge' (on the beach) sending only essential data or notifications via the cloud, given the bandwidth constraints (that may vary from beach to beach) as well as operating costs" [17]. Bandwidth issues are even more relevant to the shark researcher who may travel to exotic locations far from wireless infrastructure. While a ground station with a good video card may fit this need, a more streamlined option would be to run the model on the drone itself or on a tablet connected to a controller. A simplified configuration like this might also enable participation by citizen scientists, thus expanding the number of drones collecting data. Such a configuration would need a model conforming to the more restricted computing resources available.

Starting with some student class projects and continuing with the summer SURP program, Cal Poly has recently partnered with the Shark Lab at CSU Long Beach to create a data set of shark aerial footage of over 4,000 labelled images from 60 videos including multiple days, drone heights, and lighting conditions [25]. Various student groups have experimented with architectures such as Mask RCNN, YOLO, and Faster-RCNN, including a field test of Faster-RCNN against a live video stream

from a drone via YouTube. On-board deployment of the object detector has also been discussed for the future. SSD MobileNet is one of the fastest state-of-the-art object detectors, designed for object detection on edge devices, making it a good candidate for use in on-device detection with drones. We look at SSD MobileNet's performance on shark data and compare our results with those from Faster-RCNN.

Chapter 3

# BACKGROUND

## 3.1 Object Detection

The general problem of *object recognition*—that is, identifying the category and location of all the objects present in an image—is usually organized into several related problems. *Object classification* or *object categorization* is the identification of objects present in an image without determining their locations. *Object detection* involves identifying both the object classes in an image as well as their locations, usually denoted by bounding boxes. Object detection can be either *specific*, for detecting a single type of object (faces, pedestrians, etc.) or *generic*, aiming to determine the class and location of objects from a variety of object categories. *Semantic segmentation* assigns a category (but not a specific object) to each pixel in an image, while *object instance segmentation* determines which object instance each pixel belongs to [28].

## 3.2 SSD MobileNet

Modern object detection architectures can be divided into two categories: single and two-stage detectors. Two-stage detectors like Faster-RCNN [33] perform a region selection in the first stage to select candidate regions from the image, then refine and categorize those regions in the second stage. Single-stage detectors skip the region selection stage and instead perform a large number of object detections over the whole image. Single-stage detectors have traditionally been faster, but have lagged behind the accuracy of two-stage detectors [28].

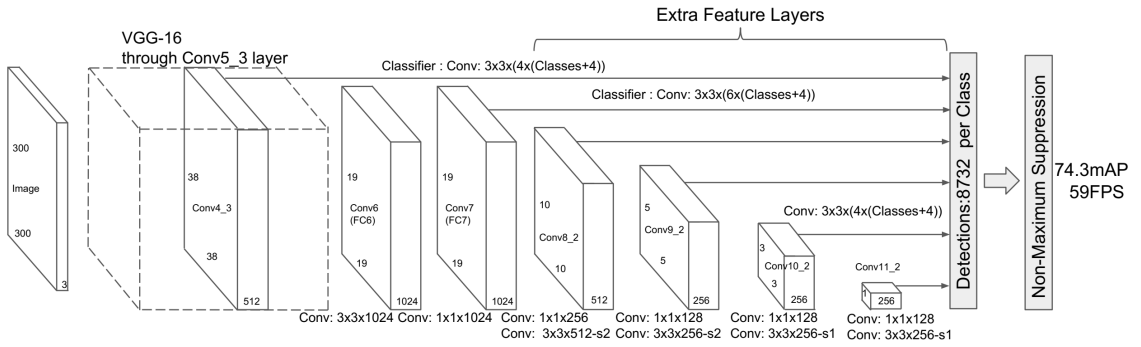


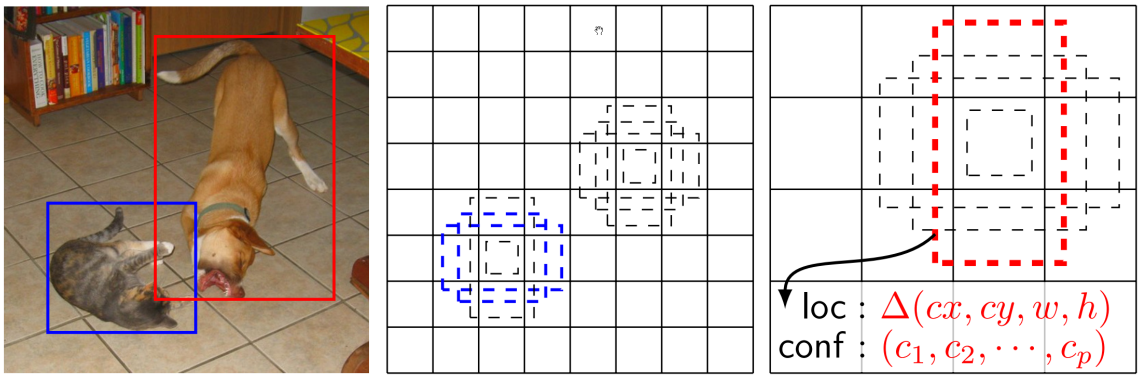

**Figure 3.1: The SSD original architecture. A VGG-16 base network is followed by a tapering series of feature maps, allowing detections to be made at 6 different resolutions. At each resolution detections are made across the layer using anchor boxes, producing for each detection a per class probability and a box position. The *tensorflow/models* implementation we used used MobileNets v1 as a base network instead of VGG-16. Adapted from the original SSD paper [29].**

SSD, standing for "single shot detector", is a single-stage detector which has gained popularity because of its speed and simplicity [29]. SSD is composed of a base network followed by a tapering series of detection layers of diminishing height and width (see Figure 3.1 on page 10). The original SSD [29] used VGG-16 [39] as a base network, but different base networks may be used and varying results have been examined [23] using base networks like MobileNet [22] and ResNet [18].

MobileNet [22] is a convolutional neural network which trades some accuracy for performance by replacing convolution operations with the less computationally expensive *depthwise separable convolutions* introduced by Sifre [38]. It includes hyperparameters for adjusting the tradeoff between speed and accuracy. As the name implies, MobileNet targets lower-performance environments like embedded and mobile devices. SSD with MobileNet as a base network is a popular combination, combining two architectures which both aim to increase speed while retaining reasonable accuracy.

After the base network, SSD adds successively smaller convolutional feature layers. From each of these layers SSD performs detection using object detectors in a grid pattern across the feature layer, with the grid size scaled according to the layer's size. The fact that different layers of varying size each have their own set of detectors has the effect of applying object detection at a variety of image resolutions and object sizes. The earlier feature layers are larger, corresponding to finer resolution and smaller object detection. For each layer's grid, at each point in the grid detection occurs at several varying aspect ratios such as 1:1, 1:3, 3:1, 1:2, 2:1, etc. Each of these aspect ratio positions on a grid in a layer constitutes a *default box* or *anchor box*, which predicts a confidence per category and a box location as a relative offset from the anchor box. In contrast with 2-stage architectures containing a separate region proposal stage, SSD's strategy is to cover most of the image with anchor boxes of varying size/resolution and aspect ratio, thereby generating a large and fixed number of detections corresponding to each layer/grid position/aspect ratio. During training, this large number of anchor box detections is then matched with ground truth boxes by first matching each ground truth box to the detection box with the highest IoU, and then matching remaining detection boxes to any ground box with an IoU of over 0.5. IoU stands for *intersection over union* and is a common measure of how much two boxes overlap. Given two bounding boxes, the IoU is calculated as the area of their intersection divided by the area of their union.

After creating a large number of detections driven by the dense placement of anchor boxes, SSD uses a *non-maximum suppression* step. In general, *non-maximum suppression* is an algorithm for removing duplicate overlapping detection boxes detecting the same object. First, detected boxes with a confidence less than a set threshold are discarded (0.01 in the original SSD), eliminating most of the detections. Then the detected box with the *maximum* confidence is chosen and all other detected boxes overlapping with an IoU more than 0.5 are *suppressed*, i.e. discarded. Then the next




(a) Image with GT boxes (b) $8 \times 8$ feature map (c) $4 \times 4$ feature map

**Figure 3.2: An example of anchor boxes at two levels of resolution, (b) and (c), to detect a dog and a cat (a). The cat is detected by two anchor boxes in the 8x8 layer (b) while the larger dog is detected by one anchor box in the lower-resolution 4x4 layer (c). In this example each point of the grid of each layer has 4 anchor boxes with different aspect ratios, which tend to detect objects close to that shape. *loc* indicates the location output by the detector as an offset from the anchor box. *conf* indicates confidences output by the detector for each of the $p$ categories. From the original SSD paper [29].**

remaining detected box with the maximum confidence is chosen, etc., until there are no remaining candidate detected boxes.

Because the matching and non-maximum suppression steps occur outside of the neural network, general neural network performance optimizations like neural acceleration hardware do not apply, and some work has been done on improving performance at this part of the architecture in other ways. In particular, the choice of anchor box configuration–aspect ratios, sizes, and positions–has a direct effect on the number of candidate detected boxes that need to be processed for each iteration. Several popular object detection architectures including YOLO [32] and Faster-RCNN [33] use variations on the anchor box concept and so improvements in this area benefit multiple architectures. Strategies such as *anchor pruning* [6] show that often a significant number of anchor boxes can be removed to decrease total computations while retaining accuracy. In addition to performance considerations, anchor boxes

may be tailored to a particular data set. YOLO, for example, uses k-means of a data set's bounding boxes to choose anchor box aspect ratios [31, 32]. Besides improving performance, these automated configuration efforts also have the benefit of removing anchor box configuration as a hyperparameter that requires tuning.

While detection of small objects is an ongoing challenge for object detection systems in general, SSD in particular performs poorly on small objects [29] even compared with other state-of-the-art architectures [23].

The original SSD included SSD300 and SSD512, corresponding to the resolutions of the input images, $300^2$ and $512^2$ respectively. A higher resolution leads to higher accuracy, but can also correspond with a larger number of configured anchor boxes and slower frames-per-second at inference time.

The TensorFlow Model Garden project on GitHub [43] (from here on referred to as TFMG) is a collection of cutting-edge machine learning models implemented in TensorFlow and maintained by Google. It includes models on problems including image classification, object detection, image segmentation, natural language processing, reinforcement learning, recommendation systems, and more. The implementations are created based on recent machine learning papers chosen by the TFMG team and the community. Some of the implementations are considered "research" quality, subject to change and experimentation. This includes their object detection API.

TensorFlow Model Garden's object detection API is a framework built to facilitate experimentation with different object detection architectures. Notably, it makes use of a protobuf configuration file system that includes abstractions of common object detection devices, including things like input resizing, anchor box configurations, batch normalization, loss functions, matching algorithms, hard example mining, non-max suppression, data augmentation, and many other concepts. This configuration

system makes it easy to make common alterations in your model and run experiments. Also provided are a folder of sample configurations for different models and a number of pre-trained models for transfer learning.

In addition to the object detection API implementation at TFMG, Google Research also did a broad analysis of various configurations of popular object detection architecture and base network combinations [23]. Figures 3.3-3.5 and Table 3.1 on pages 14-16 show some of the results demonstrating characteristics of SSD MobileNet compared to other architectures and configurations. We have highlighted SSD MobileNet in each.

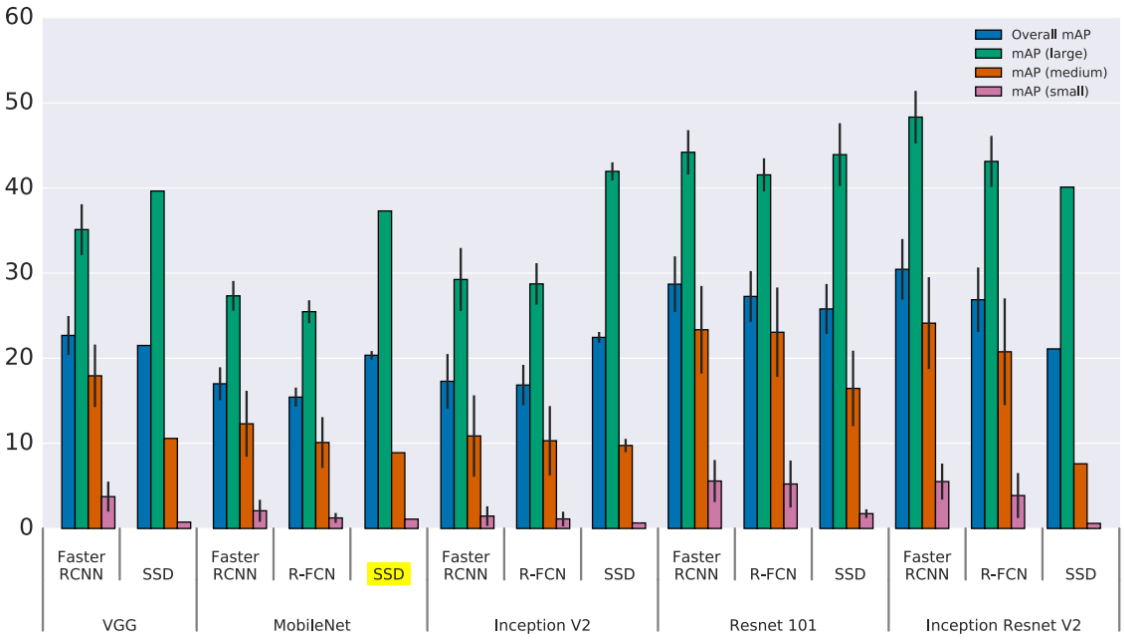



**Figure 3.3: Accuracy for various model/base network combinations with an image resolution of $300^2$. We have highlighted SSD MobileNet, which has a poorer accuracy for small/medium objects, but a more competitive accuracy for larger ones. Taken from Huang et al. [23].**

Table 3.1: Accuracy and number of parameters of various base networks. Top-1 accuracy is the classification accuracy on ImageNet. Note that MobileNet has significantly fewer parameters (ostensibly better performance) while remaining reasonably competitive in accuracy compared with the other base networks. Taken from Huang et al. [23].

| Model | Top-1 accuracy | Num. Params. |
|---|---|---|
| VGG-16 | 71.0 | 14,714,688 |
| MobileNet | 71.1 | 3,191,072 |
| Inception V2 | 73.9 | 10,173,112 |
| ResNet-101 | 76.4 | 42,605,504 |
| Inception V3 | 78.0 | 21,802,784 |
| Inception Resnet V2 | 80.4 | 54,336,736 |

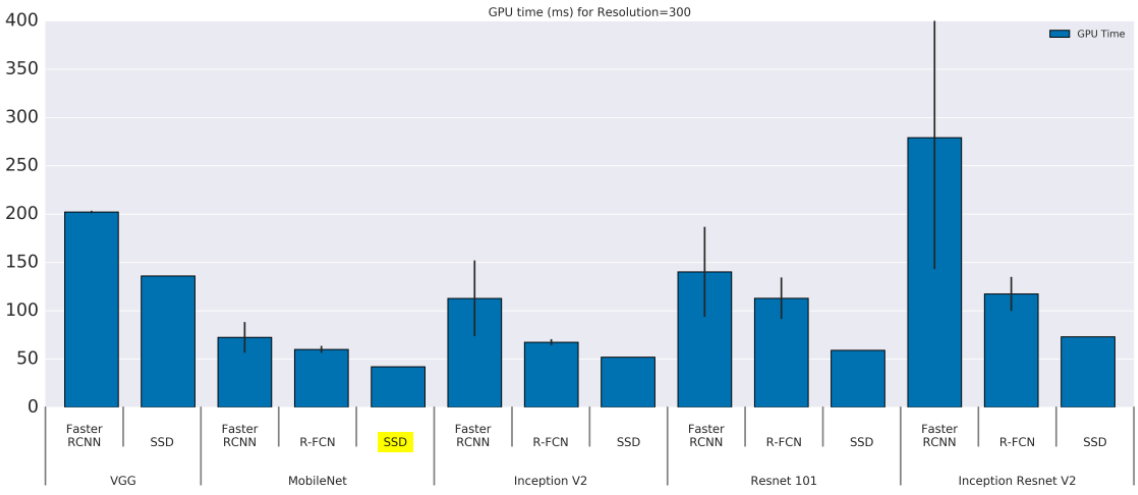



Figure 3.4: GPU time (milliseconds) for various model/base network combinations with an image resolution of $300^2$. We have highlighted SSD MobileNet. Taken from Huang et al. [23].

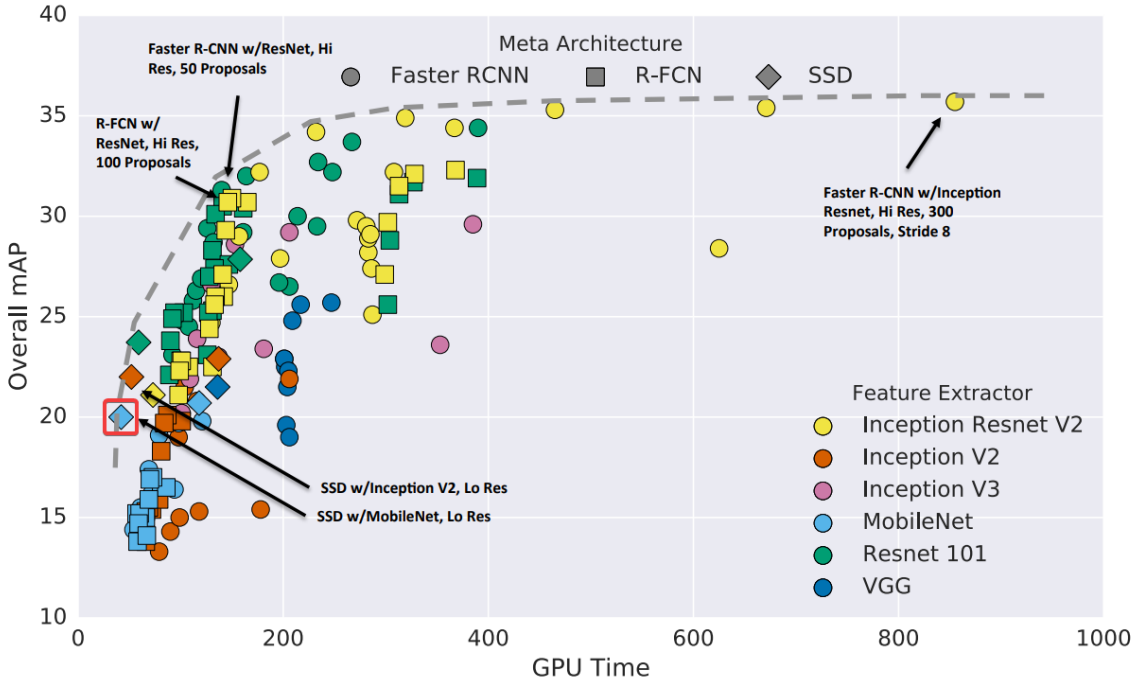



**Figure 3.5: Accuracy vs time, with marker shapes according to architecture family (meta-architecture) and colors indicating base network (feature extractor). SSD MobileNet with input of $300^2$ resolution has a red box around it. A given architecture/base network combination can occur more than once because of different input sizes, stride, etc. The dotted line represents an "optimality frontier" of the accuracy/speed tradeoff. The arrows indicate the fastest, the most accurate, and those at what they term as the "Sweet Spot". Note that Faster-RCNN may be configured for high accuracy at the right, or for higher speed in the "Sweet Spot". Taken from Huang et al. [23].**

# Chapter 4

# DESIGN/METHODOLOGY

One of the technical goals of the Cal Poly shark group has been the ability to detect sharks in UAV footage in real-time. Running a machine learning model to detect sharks from live video in an embedded environment such as on board a UAV requires a model that can perform at a high rate of FPS with few resources. To that end, we've focused on SSD MobileNet, identified in the study by the TensorFlow Model Garden's Huang et al. [23] as both the fastest model in their study and one of the models using the least memory. We use the implementation in the TensorFlow Model Garden.

## 4.1 Anchor Box Configuration

While many modern object-detection architectures perform poorly on smaller objects, SSD is among the worst [29][23]. This is possibly due in part to an "out-of-the-box" anchor box configuration that favors larger objects. Figure 4.1 (Top left) on page 20 shows the default TensorFlow Model Garden anchor box aspect ratio and size configuration. Each dot represents a group of anchor boxes of a certain aspect ratio spread in a grid over their layer, and each dot's color represents the feature map layer of a certain resolution to which they belong. Smaller anchor boxes are used on higher-resolution layers and applied in grids of smaller cells, requiring more anchor boxes to cover the layer. Because of this the highest-resolution layer only has 3 anchor box aspect ratios per cell for the sake of performance. The dots overlay a two-dimensional histogram of the widths and heights of the objects in our data set.

It can be noted that most of the bounding boxes in the data set are much smaller than the anchor boxes. The right side of the figure shows a new configuration with two of the smallest anchor boxes shrunk to a size closer to that of the objects. This new configuration does not change the resolution of the convolutional feature map layers that the anchor boxes operate on, it just changes two of the anchor box sizes used to calculate the object detection location values at the highest-resolution layer.

## 4.2 Image Resolution and Sampling

The original SSD input image sizes are $300^2$ and $512^2$, while our images are on the order of 2,000 x 4,000 pixels. TFMG's SSD provides for online preprocessing to resize them to $300^2$, which we use for most of our experiments. However, $300^2$ is challenging for even humans to pick out some of the objects. We run an experiment with $512^2$ for comparison, noting the run time as well as the accuracy results. The default resize method uses bilinear interpolation, which can leave objects blurry. We experiment with using the provided nearest neighbor resizing interpolation setting to reduce the blurring of object edges.

## 4.3 Measures for Mitigating Object Versus Background Imbalance

The architecture of SSD and other related single-stage detectors has also been shown to suffer from a large class imbalance that negatively effects the ability of the network to learn effectively. Because SSD creates a detection for each aspect ratio/anchor box at each grid point in several grids covering different image resolutions, this results in a large number of negative background "easy" matches. These high-confidence background matches can overwhelm the loss function and have detrimental effects on the accuracy [26]. One solution to this is *hard example mining*, where high-confidence

negative (background) detections are thrown out in favor of "harder" detections in order to achieve a more favorable ratio of background-to-object detections during training. SSD—and by default the implementation we used—uses hard example mining during training to enforce a ratio of at most 3 negative (background) detections per positive (object) [29].

A more effective solution to the problem of too many background detections during training of single-stage object detectors was suggested by Lin et al.'s RetinaNet using a modified loss function, Focal Loss [26]. Focal Loss adds an extra term to the cross entropy loss function to give less weight to high confidence background examples. Using Focal Loss, the one-stage RetinaNet is able to achieve accuracy comparable to two-stage detectors like Faster-RCNN. We apply Focal Loss to SSD using the builtin implementation.

### 4.4 Comparison With Faster-RCNN

Faster-RCNN is also considered one of the state-of-the-art object detection frameworks, and was used in previous work by the Cal Poly shark group. Though it is in general more accurate than SSD, it is also slower, though some configurations can be made to run at a competitive speed [23]. We run an experiment with Faster-RCNN as a comparison.

We use the default data augmentation during training, consisting of a random horizontal flip and a random crop in the method described in the SSD paper [29].

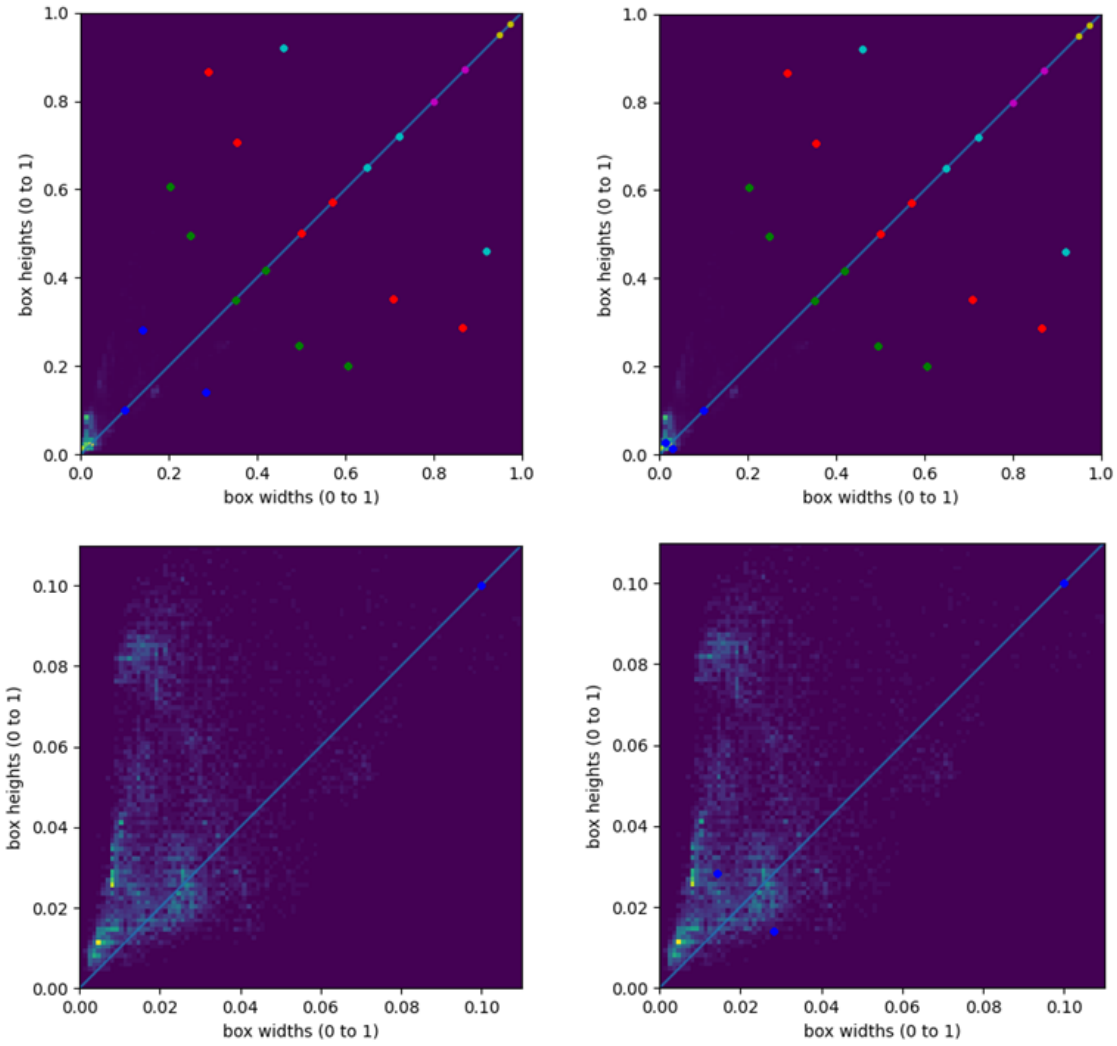



**Figure 4.1: (Top left) Shows the size and aspect ratios of the orignal-SSD-inspired anchor boxes used in most of the TFMG's sample SSD configurations. A dot shows an aspect ratio used by one or usually more anchor boxes. The dot colors group anchor boxes of the same feature map layer, with smaller boxes corresponding with higher resolution layers. Smaller boxes will also be more numerous to cover their layer. The anchor box dots overlay a histogram of the sizes of objects in our 4030 image shark data set. The concentration of object sizes is at a much smaller size than the smallest anchor boxes. (Bottom left) Shows a zoomed in version of the graph above it. The prevalence of tall objects over wide ones in the histogram is at least in part due to the horizontal squashing of landscape-aspect images during preprocessing, when they are resized to $300^2$ (Top right) We move two anchor boxes (in blue) to a smaller size more appropriate for our objects. (Bottom right) Shows a zoomed in version of the graph above it.**

# Chapter 5

# IMPLEMENTATION

Our data was obtained from CSU Long Beach's Shark Lab and consisted of videos taken from aerial UAV footage. Images were extracted from the videos and then uploaded to Labelbox [1], a SAAS labeling solution[1]. Twelve team members from CSU Long Beach and Cal Poly labelled objects in the images with a bounding box and one of five categories: {"boat", "person", "dolphin", "sealion", or "shark"} [25]. Subcategories were labelled as well but were not used in this paper. Images with no objects were discarded. Images were then shuffled using data from RANDOM.ORG [2] and split into a training group (2824 images), a validation group (706 images), and a final test group (500 images), for a total of 4030 images. The final test group was not used except for during analyses treating the entire data set such as Figure 4.1 on page 20.

Jobs were performed on one of several Intel Xeon E5 2695 2.30GHz machines, with 28 CPUs and 31GB of memory running Linux. TensorFlow 1.15.2 was used in CPU mode and small modifications were made to the TFMG codebase to fix a bug regarding the loading of models for transfer learning.

Transfer learning was applied according to the instructions in the TFMG pets tutorial [41]. Weights from a Microsoft COCO-trained model were used before training, available from TFMG under the name "ssd_mobilenet_v1_coco_2018_01_28". An initial training job was run for several days and based on its results an early stopping point of 200,000 steps was chosen for future jobs, which was also the number of steps

[1]The code to download and transform the data from Labelbox's format was adapted from work by Casey Daley and is available at `https://github.com/caseydaly/LabelboxToTFRecord`

used by the pets tutorial. All the SSD jobs also use the same constant learning rate of 0.004 as the pets tutorial, which is low to preserve the value of the transferred weights. The batch size was set to 24 for all SSD jobs.

Following one of the TFGM SSD focal loss configuration examples, the focal loss experiment used $\alpha = 0.25$ and $\gamma = 2.0$ for the loss function hyperparameter values, which are also the values that produced the highest accuracy using RetinaNet in the ablation experiments by Lin et al. [26].

The Faster-RCNN experiment was run on a machine with the same specifications, also in CPU mode. Its configuration differed but was also mostly a use of the corresponding pets example configuration. Instead of resizing to a square $300^2$, it was configured to rescale each image, preserving aspect ratio. The rescaling applies first to reduce the image's smaller dimension to 600px. If at this point the image's larger dimension is larger than 1024px, the size is further reduced so that the larger dimension is equal to 1024px. The learning rate was a constant 0.0003 and a batch size of 1 was used. ResNet 101 was used as a base network and TFMG's provided COCO-trained model was used for transfer learning. Data augmentation consisted of a randomly applied horizontal flip. Training was run for 200,000 steps, the same number used for the SSD runs.

Chapter 6

# RESULTS/VALIDATION

## 6.1 Results

Results can be seen in Table 6.1 on page 23. To reach 200,000 steps (a little over 70 epochs), all of the SSD 300 training runs took around 5 days. The two runs working with higher resolutions took longer, with Faster-RCNN taking 7.3 days, and SSD 512 taking the longest at 14.6 days. For measuring accuracy we use the normal Microsoft COCO metrics [27].

**Table 6.1: Duration, mean average precision (mAP), and average recall (AR) for several configurations of SSD 300, one of SSD 512 and one of Faster-RCNN. All jobs were run to 200,000 steps.**

| | Duration (days) | mAP | mAP | | |
|---|---|---|---|---|---|
| | | | S | M | L |
| SSD 300 [29] Baseline | 5.5 | 0.275 | 0.00282 | 0.119 | 0.468 |
| Focal Loss [26] | 5.0 | 0.267 | 0.00332 | 0.0989 | 0.459 |
| NN Resizing | 5.4 | 0.274 | 0.0015 | 0.138 | 0.456 |
| Modified anchors | 5.5 | 0.296 | 0.002 | 0.181 | 0.473 |
| SSD 512 [29] | 14.6 | 0.306 | 0.004 | 0.219 | 0.509 |
| Faster-RCNN [33] | 7.3 | 0.426 | 0.109 | 0.314 | 0.619 |

| | AR | AR | | |
|---|---|---|---|---|
| | | S | M | L |
| SSD 300 [29] Baseline | 0.363 | 0.0112 | 0.202 | 0.567 |
| Focal Loss [26] | 0.347 | 0.0072 | 0.149 | 0.568 |
| NN Resizing | 0.361 | 0.0064 | 0.25 | 0.56 |
| Modified anchors | 0.385 | 0.008 | 0.28 | 0.583 |
| SSD 512 [29] | 0.404 | 0.0168 | 0.307 | 0.608 |
| Faster-RCNN [33] | 0.501 | 0.143 | 0.406 | 0.71 |

None of the SSD 300 configurations noticeably improved the the overall mAP over the baseline of 0.275, with the exception of modifying the anchors, which improved it by 2.1 percentage points. This increase in the overall mAP is due to increased performance of the medium category, discussed below. However, while modest increases may indicate higher performance, small improvements (or decreases) should be taken with a grain of salt. Runs appeared to show some moderate instability in the reported metrics over the course of a run, and small mAP differences between runs may possibly be accounted for by these fluctuations. With this in mind, the only runs that appeared to significantly increase the overall mAP over that of the SSD 300 baseline were the higher resolution runs, SSD 512 and Faster-RCNN, which showed results of 0.306 (3.1 percentage points higher) and 0.426 (15.1 percentage points higher) respectively. A higher-resolution SSD leading to better results is not surprising, and Faster-RCNN is known to have better accuracy than SSD at the cost of speed.

The large (L) mAP category was also mostly the same across all SSD 300 configurations, with focal loss and nearest-neighbor interpolation-based resizing resulting in lower mAP by 0.9 and 1.2 percentage points. These decreases may be due to training instability. Again SSD 512 performed better than the baseline, this time by 4.1 percentage points, with Faster-RCNN performing the best by a significant margin of 11 percentage points.

The medium (M) mAP category may be the most beneficial target of improvement because it contains roughly half of the objects in our data set and yet has a much poorer baseline performance of 0.119, a staggering 34.9 points below the large (L) category's 0.468. Our SSD 300 experiments show modest improvements in this category, with nearest-neighbor resizing showing a result of 0.138, 1.9 percentage points over the baseline of 0.119. If this is not the result of convergence instability, then

it makes sense intuitively that nearest neighbor would be more effective for smaller objects, since they would likely benefit the most from sharper edges. Of the SSD 300 runs, the anchor box modification shows the largest improvement, resulting in 0.181, 6.2 over baseline. This anchor box change is targeted specifically towards improving smaller objects detection, as seen by Figure 4.1 on page 20, so an improvement in the medium category confirms the effectiveness of anchor box configuration changes to this end. Of the SSD 300 results, this is the strongest. For the rest of the medium category, SSD 512 performed 10 percentage points over the baseline and Faster-RCNN performed 19.5 percentage points over.

The AR results follow the mAP results, with the slight exception of nearest neighbor resizing in the medium category. This configuration shows a slight increase of 1.9 percentage points over the baseline medium mAP, and a larger medium AR increase of 4.8 percentage points over the baseline. This difference could be due to training instability, or it could indicate that nearest-neighbor resizing, perhaps through a hardening of object edges, results in more detections partially through the inclusion of more background objects such as seaweed or sand colorations, which can be very similar to sharks.

All SSD runs performed extremely poorly (less than 0.01 mAP) in the small category. Even Faster-RCNN, the best result, was only 0.109. This reinforces past findings of poor SSD performance on small objects [29][23], but it is also likely amplified by the paucity of small objects in our data set. Additionally, it is difficult for even humans to detect some objects of this size, especially after images are resized, introducing some interpolation blur.

### 6.2 Discussion

The SSD 512 results are consistently better than SSD 300, and the Faster-RCNN results are the best, which is in accordance with the literature [29][23]. Also, all of the configurations performed best on large objects, which is again an established result [29][23]. Faster-RCNN is more accurate and in fact, as of the writing of Liu et al. [28], is a common starting point for winning entries of object detection competitions. SSD MobileNet's main benefit is that it is fast, and the speed comes at the cost of accuracy.

In a comparison between F-RCNN and SSD 300 baseline in the medium and large categories, the gap between detectors for medium is only 4.4 percentage points more than for the large category. This suggests that the weakness of SSD 300 is more pronounced for medium objects than large ones, but only moderately so. Nevertheless, because of its low baseline, the medium category may still be a good target of efforts to improve. In particular, the anchor box modification result demonstrates that out-of-the-box detector configurations may be configured towards larger objects than appropriate for our data set. Our anchor box configuration change shows that improvement is possible. The anchor box configuration change we made is minimal and there is much room for more experimentation in this area to improve both accuracy and speed [6].

The nearest neighbor configuration results appear inconclusive, but may be worth keeping in mind for a very small potential accuracy boost on medium objects. However, its effects may be limited to low-resolution configurations.

The focal loss results did not seem to show any improvement, which may be because the original focal loss [26] was applied to a different CNN architecture, RetinaNet,

and because we used the default hyperparameter values without any ablation experimentation. Additionally, SSD's hard example mining [29] has a purpose similar to that of focal loss, though the results of Lin et al. with RetinaNet indicate that focal loss should be more effective.

# Chapter 7

# FUTURE WORK

The challenge of shark object detection can be split into real-time and non-real-time, on-device and off-device. We have focused on SSD MobileNet as a model for use in real-time on-device detection. However, future work may focus on other shark-detection scenarios. We see the following as possible next steps particular to the Cal Poly Shark group.

## 7.1 Collaboration

Since the field is new, there appear to be only a handful of research organizations building prototypes of shark detection solutions at the moment. While some of the more commercial offerings may be less willing to share proprietary information, there may be opportunities for Cal Poly and CSU Long Beach to collaborate with the more academic groups from around the world. In particular, sharing and combining data would likely benefit all, as the effort of collection and labelling is so high, and we are not the only group to express a need for more data [17]. A larger number of training images could increase accuracy, and images from different locations across the world with different weather, water/sand coloring, etc. may improve the diversity of the data set. Additionally, with the exception of Saqib and Sharma et al. [36][37], there have not been many instances of shark detection results being published in a raw, comparable form using standard metrics like COCO. Another advantage of combining the data sets between groups into a common set (with common performance metrics) is that machine learning results between groups and publications would be more

directly comparable, like it is between work using crafted public data sets such as COCO, Pascal VOC, etc. While the shark data need not be formalized and made public to provide similar value between the handful of shark detection research groups, publishing the data may be a way to attract interest and help from non-shark machine learning practitioners. Besides data sharing, other knowledge sharing between shark research groups in high-skill areas such as machine learning and drone development could also be beneficial.

## 7.2 Model and Hardware Selection

The appeal of SSD MobileNet is its reduced number of computations and its faster speed for use on edge devices such as a drone. However, to increase accuracy, other models may be more desirable. Valuable to this end is the work of Huang et al. [23], which shows the result of speed and accuracy tests across a broad number of object detection architectures and configurations (though regrettably not YOLO). As stated in the paper, their work is intended as a guide to help people choose object detection models at a scale and accuracy level appropriate for their applications. Though a few years old now, we believe it to still be useful in selecting object detection models and their base networks, though newer models and base networks should be considered as well. One notable result of the study is that Faster-RCNN can be configured to run at speeds somewhat approaching that of SSD MobileNet but with better accuracy. Also relevant is the comparison of performance of object detection architectures on different drone configurations performed by Hossain et al. [21]. Though the range of models compared was more limited than in the Huang analysis, SSD MobileNet and variations of YOLO were the fastest. This comparison also looks at on-device vs off-device detection as well as performance with different on-device processing hardware,

including some low-cost hardware similar to devices already available to the Cal Poly Shark group.

Faster-RCNN is the obvious choice for cloud solutions, from our limited experiments, and past team members have had success with Faster-RCNN as well. But object-detection is still an unresolved problem and new architectures are coming out each year. Perhaps future work should expect to need to change models every few years to keep up with state-of-the-art. TFMG's configuration DSL offers a way to do this, and TFMG already contains some newer models that we were unable to try (mentioned also below). Additionally, YOLO [32] is one of the leading architectures which we did not examine because it is not implemented in TFMG, but would probably be worth exploring, especially because of its performance in the on-device drone tests performed by Hossain et al. [21].

### 7.3 Where to Perform Object Detection

The decision of where to run the model (cloud, local ground station, drone, tablet/-controller, etc.) is central because it can have a large impact to how useful the solution will be to CSU Long Beach research and because it defines constraints around model and programming framework choice which are not easy to change down the line without starting over. Model training and research into domain-specific model improvements, for example, may not be transferable across model architectures, and programming work done for an iPad will not be very relevant to a solution in the cloud. Knowing the exact specifications of the target environment (like the incoming resolution) would also help with model training and design [25]. A usability study of CSU Long Beach shark researchers might help inform this decision to insure that solutions were useful to researchers.

Regarding the challenges of installing an object detection model on an actual device (FAA regulations, closed platforms, etc.), we believe an attractive alternative would be to run the model on a tablet device connected to a controller. DJI, a popular drone manufacturer, offers a Mobile SDK that appears to be compatible with some of the drones used by CSU Long Beach. There is a similar Onboard SDK whose documentation even includes a tutorial on object detection with YOLO [13], but it appears to only be compatible with a small subset of drones marketed and priced for industrial use [14].

On the other hand, performing real-time object detection in the cloud on a live stream from a drone allows for more computational resources for the object detector and doesn't require drone modification. However, this introduces the potential for connectivity challenges and latency, which may be more relevant to scenarios of shark research than the more stationary beach alert systems.

The target environment also affects the testing that can be performed, but we encourage future students to explore testing setups that mimic the target environment as close as reasonably possible without waiting for access to an actual drone. For example, if the video quality of a drone's live stream was known, a cloud/drone streaming solution could be tested by playing a video with the same quality instead of training against the quality of our existing test data. Or, to test an on-drone solution, a drone flight or even a flight-worthy drone may not be necessary if an embedded device was available that approximated the drone's on-board computing constraints. This requires investigation into the technical specifications of the devices used by CSU Long Beach shark researchers.

### 7.4 Improving Accuracy

Within the scope of improving SSD MobileNet there are several unexplored areas. The most straightforward is to replace MobileNet with a different base network such as ResNet [18]. TFMG contains other transfer learning models with varying accuracy including several for ResNet. Regarding other configuration changes, our results showed some modest improvements in the medium category due to anchor box changes. The changes we made were minimal and could very likely be expanded to improve accuracy further. YOLO for example uses k-means clustering to determine a good set of anchor box aspect ratios [31]. One thing that we did not try was anchor box modifications on the more accurate SSD 512 model, which we expect would yield good results. It is also likely that some of the overly-large anchor boxes could be removed to improve performance [6]. Finally, our changes did not change the actual layer sizes that the anchors were connected to, and it may be that the default SSD layers could be better sized for our data set.

SSD and Faster-RCNN are both generic DCNN object detectors in the sense described by Liu et al. [28]. One approach for increasing accuracy could be to focus on creating hand-crafted features or image preprocessing techniques that would be helpful specifically in the domain of aerial marine data, though determining how best to integrate these techniques with out-of-the-box frameworks may be challenging. For example, fluid lensing [11] is a technique for removing water distortions that has been used to increase the quality of marine drone footage of underwater objects. Similarly, there may be domain-specific data augmentation techniques that could be explored, as we only used the default TFMG SSD data augmentation.

The difficulty in understanding the meaning of neural network output is well-known. One thing that may help is to go beyond reported COCO metrics and perform an

analysis of the evaluation results, looking at where false positive bounding boxes were detected or where false negatives were missed. This could help determine where a model is weak and perhaps identify targets of the domain-specific techniques mentioned above. Hoiem [20] has written such an analysis and has made available some MATLAB tools that create charts from annotated Pascal VOC data showing the prevalence of different types of false positives. We considered such an analysis, but it requires manually inspecting all of the evaluation images and annotating them with the type of false positive.

### 7.5 Real-Time Video Detection

While this work and much of the literature have focused on the detection of objects within individual images, the best application of an image detector to video is still a matter of investigation. The naive strategy of running the detector on every video frame is computationally expensive to do in real-time, though like Gorkin et al. [17] detection may be performed every $n$ frames to help performance. However, doing a full object detection for each frame even when sampling the frames may be unnecessary. Once an object is detected in a frame, the object is likely to be at a nearby location in the next frame. We have not investigated object detection using contextual information from previous video frames to track objects across frames but we believe it may be a promising line of inquiry that could reduce the computational resources needed.

### 7.6 Small Object Detection

Small object detection is an ongoing area of computer vision research, as our results demonstrate clearly. This may make it difficult for our group to appreciably increase

accuracy for small objects beyond the default for a given model, though domain-specific tricks may help. Small objects may be particularly troublesome because when researchers are looking for sharks, they may fly as high as possible in order to see the most area at once and save battery life. Flying higher makes the sharks appear smaller, making it more difficult for an object detection system. However, object detection may still be useful once sharks are found and drone height is lowered for automated photogrammetry. Ultimately, as with the overall imperfect detection accuracy, it remains to be seen what level of accuracy would be required for a solution to be useful to researchers in the field.

One way to increase the size of objects under detection would be to cut the images into tiles before inital image resizing, and then resize each tile to $300^2$ or $512^2$ to feed individually to the detector. In fact, though it is a little unclear from the description of Gorkin et al. [17], this appears to be the strategy adopted by the Australian Sharkeye project with their YOLO-based model.

### 7.7 Drone Height Data

Our data set did not contain drone heights, but height is reported by the drones at flight time, and may be able to be incorporated into future data sets. Height data as an extra input feature might be useful for improving object detection, as it has a direct relationship with object size. How to incorporate this extra feature into an existing model like SSD or Faster-RCNN would need to be explored.

### 7.8 Usefulness in Shark Research

While we have focused on real-time on-device object detection, value can also be had from offline detection. The current workflow of shark researchers at CSULB involves manually going through footage post-flight to find instances of sharks to study, a tedious process. A tool that could automate this search through footage could save researchers time, and may not require 100% accuracy. Similarly, tools to automate shark photogrammetry (taking measurements of the sharks' sizes, distances from other people, body bending angles, etc.) could aid shark research as well. For example, a tool might identify timestamps in a video where a shark was less than 10 feet from a person. Additionally, the ability to automatically count sharks could be useful in population sampling studies. Whether in offline or real-time scenarios, success in object detection may be able to be extended to related computer vision problems such as object tracking, enabling researchers track and study the movements of individual sharks within large groups. Finally, shark species identification may be useful in beach safety scenarios to help beach authorities without in-depth shark species knowledge to know which shark detection events necessitate a beach closure.

### 7.9 Data Set

Though this work has touched upon it, further interesting work could include a more in depth analysis of the existing shark data set. In particular, comparisons with how this data set compares with popular test sets like Microsoft COCO [27], Pascal VOC [15] or iNaturalist [42] could bring interesting insights into both the performance of this model as well that of other models. Different data sets have different difficulty levels for object detection due to things like the size of the objects and the number of categories. Moreover, there may be domain-specific characteristics to our data set

that could be exploited for better accuracy that do not apply to other data sets. To name a couple (hypothetical) examples, marine aerial data may have distinct qualities like a characteristic background color spectrum or a narrow range of object sizes per image based on drone height and ground sample distance. One observation we have made is that in some images, sharks very similar in color to water and sand features are distinct to the human eye because they are positioned at a different angle from the grain of the sand. Such qualities may be exploitable to improve our domain-specific results.

Additionally, it would probably help accuracy to increase the number of training images. Besides further manual labelling on our part, there may be other shark researchers willing to share, or existing data sets online that could be combined with ours. One such example is the recent aerial data set created by Gasienica-Józkowy et al. [16]. Additionally, a transfer learning data set consisting of data more similar to ours than COCO may help.

### 7.10 Computation Resources for Training

Our group has taken two approaches so far for training environments. One approach is to use a Google Colab notebook, the other is to use a machine from Cal Poly's Massively Parallel Accelerated Computing (MPAC) lab. The Colab notebook is run from a web browser against a GPU, and can have problems of reliability with long-running jobs. The MPAC machines, which our experiments used, contain 28 CPUs each, but from our experiments are significantly slower than the Colab notebooks. We believe this to be due to the CPU/GPU difference. This would seem to indicate that an on-campus dedicated machine running in GPU mode could combine the best of both worlds. Additionally, TensorFlow has built-in ability for clustering training

jobs, and it may be feasible to configure more than one machine to create a cluster on-campus for faster training.

It would be remiss to go without mentioning cloud platforms as an option for training. Cal Poly ITS has formed a relationship with Amazon in moving its infrastructure to AWS within the last several years, and in the creation of its Digital Transformation Hub. In fact there has already been effort by another group at Cal Poly doing marine object recognition on AWS through the Digital Transformation Hub [9]. Additionally, TensorFlow comes with built in integration with Google Cloud, and it is fairly trivial to run the TFMG pets tutorial on Google Cloud. We experimented with Google Cloud but our new user free credits were only sufficient to run two successful runs. From these runs alone it is difficult to compare the speed of the Google Cloud cluster settings we used with the MPAC lab setup, but a rough comparison appears to show this particular Google Cloud configuration to be on the order of seven times faster than our single-machine MPAC setup.

The challenge of the cloud across platforms seems to be that the cost of resources is often difficult to predict—perhaps meant to be absorbed on the scale of a business budget and not that of a student. Additionally we could not find any safeguards on Google Cloud for preventing long jobs or mistaken configurations from running their full course and resulting in large bills.

One possible mitigating strategy to this problem could be to establish a shared group pipeline in place of separate student projects. If student experiments were run on the same code base and cloud pipeline, repeated cloud runs would in theory have comparable cost, though consideration would still need to be made for configurations that greatly changed training time (such as model changes or input resolution changes). With a large enough team, this could be done in a controlled manner such as at the same time once week, perhaps even with a code review for new experiments as a

gateway to protect against accidental large and expensive job runs. It may also be possible to create the pipeline in such a way that students could run smaller jobs locally to test changes out before submitting them to the large shared group pipeline. However, all this would require continued infrastructure work and administration.

Another factor that could aid in improving the computing story is to decrease the resources used by the code through configuration changes (like removing anchor boxes) or by trying newer and more efficient models. Determining which models is a topic for more research, but TFMG has some newer models in their TensorFlow 2 section that we were not able to try (see below).

The challenge of adequate computing capacity is very relevant for students working on machine learning projects at Cal Poly. In our work, the length of time to run each job was part of what caused us to forego ablation experiments in place of trying a handful of strategies. Solving this problem would greatly aid future student experiments.

### 7.11 TensorFlow Model Garden

If future students choose to use TensorFlow Model Garden, a few things are worth taking note of. Object detection in TFMG is a research-quality project which may frequently be undocumented, misdocumented, or have documented features that have been broken for long periods of time. Our work required tracking down an open GitHub issue to apply a brief 3rd-party suggested code patch to fix the transfer learning feature. We also spent time fixing the image export feature for TensorFlow 1 with the goal of performing an analysis similar to that of Hoiem [20]. However, we did not use this fix. The TFMG codebase is complicated and was designed very generically we believe to facilitate the broad cross-model testing performed by Huang et al. [23]. A simpler framework may be more helpful for future students wanting to

try only one or two models. Finally, TFMG is a fast-moving repository that receives a large number of commits, and it may be useful to pin to a particular release. Since our work began, TFMG has released official pip packages, which may make things easier to install but more difficult to alter the underlying code.

While we started with and decided to stay with TensorFlow 1, there are several promising new models available from TFMG only for TensorFlow 2 which we have not experimented with, including RetinaNet [26], CenterNet [44], and EfficientDet [40]. There are also a variety of new transfer learning models for the abovementioned as well as SSD and Faster-RCNN with higher resolutions and better base networks. We would strongly recommend trying some of these new TensorFlow 2 models.

Though this is not specific to TFMG, we have found it a good practice to pin specific library versions using conda and pip. This can even be done after the fact by looking to see what version was installed and freezing to a particular version. This has enabled us to reproduce our set up quickly on multiple machines. In contrast, there was an old group Colab notebook we had difficulty running which may have been helped by pinning library versions.

Finally, the TFMG code base appears to have undergone a large reorganization since our work, and many of the above comments may or may not still hold true.

# Chapter 8

# CONCLUSION

On-drone and on-controller shark detection provide a simpler setup than cloud solutions, the advantage of not relying on internet connectivity, and the potential for low-latency flight-time detection to aid drone pilots. As object detection improves, things like single and multi object tracking may enable drones to automatically follow sharks, allowing more flight automation.

SSD MobileNet shows promise for on-device detection because of its speed, but it does perform substantially worse than Faster-RCNN on our shark data when it comes to accuracy. It performs moderately on large objects, but struggles on medium ones (COCO size categories). While the speed-accuracy tradeoff may be acceptable for some use cases, users of SSD should be sure to configure anchor boxes according to the underlying data since the default anchor box sizes do not appear to be ideal for small-to-medium objects. Moreover, increasing the input image size to SSD beyond the default $300^2$—to the extent possible on an embedded device—may be an effective means of improving performance, as well as trying more accurate (and compute-intensive) base networks. Finally, it remains to be seen what level of accuracy a model requires to be of use to shark researchers in the field, which is an important consideration for making decisions trading speed and accuracy.

# BIBLIOGRAPHY


[1] Labelbox. https://labelbox.com. Accessed: 2021-10-16.

[2] RANDOM.ORG: True random number service. https://www.random.org. Accessed: 2021-10-23.

[3] Drones. https://www.sharksmart.nsw.gov.au/technology-trials-and-research/drones, Sept. 2019. Accessed: 2021-12-03.

[4] How Salesforce Einstein is helping track (and protect) great white sharks in the wild - Salesforce news. https://www.salesforce.com/news/stories/how-salesforce-einstein-is-helping-track-and-protect-great-white-sharks-in-the-wild/, Oct. 2019. Accessed: 2021-8-24.

[5] SharkSpotter: a world first in shark detection. https://www.uts.edu.au/news/tech-design/sharkspotter-world-first-shark-detection, Jan. 2020. Accessed: 2021-12-03.

[6] M. Bonnaerens, M. Freiberger, and J. Dambre. Anchor pruning for object detection. *arXiv:2104.00432v1 [cs.CV]*, 2021. https://arxiv.org/abs/2104.00432v1.

[7] G. H. Burgess, B. D. Bruce, G. M. Cailliet, K. J. Goldman, R. D. Grubbs, C. G. Lowe, M. A. MacNeil, H. F. Mollet, K. C. Weng, and J. B. O'Sullivan. A re-evaluation of the size of the white shark (carcharodon carcharias) population off California, USA. *PLoS One*, 9(6), June 2014. https://doi.org/10.1371/journal.pone.0098078.

[8] P. A. Butcher, A. P. Colefax, R. A. Gorkin, S. M. Kajiura, N. A. López, J. Mourier, C. R. Purcell, G. B. Skomal, J. P. Tucker, A. J. Walsh, J. E. Williamson, and V. Raoult. The drone revolution of shark science: A review. *Drones*, 5(1), Jan. 2021. https://doi.org/10.3390/drones5010008.

[9] Cal Poly DxHub. Using deep sea animal recognition to measure the impact of offshore wind energy. https://dxhub.calpoly.edu/challenges/mbari/. Accessed: 2021-7-22.

[10] J. C. Carrier, M. R. Heithaus, and C. A. Simpfendorfer, editors. *Shark Research : Emerging Technologies and Applications for the Field and Laboratory*. CRC Press, 2018. https://doi.org/10.1201/b21842.

[11] V. Chirayath and R. Instrella. Fluid lensing and machine learning for centimeter-resolution airborne assessment of coral reefs in American Samoa. *Remote Sensing of Environment*, 235:111475, Dec. 2019. https://doi.org/10.1016/j.rse.2019.111475.

[12] A. P. Colefax, P. A. Butcher, and B. P. Kelaher. The potential for unmanned aerial vehicles (UAVs) to conduct marine fauna surveys in place of manned aircraft. *ICES Journal of Marine Science*, 75(1):1–8, Jan. 2018. https://doi.org/10.1093/icesjms/fsx100.

[13] DJI. Advanced sensing - object detection sample - DJI onboard SDK documentation. https://developer.dji.com/onboard-sdk/documentation/sample-doc/advanced-sensing-object-detection.html, 2017-12-19. Accessed: 2021-9-6.

[14] DJI. Onboard SDK. https://developer.dji.com/onboard-sdk/. Accessed: 2021-9-6.

[15] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (VOC) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2009. https://doi.org/10.1007/s11263-009-0275-4.

[16] J. Gasienica-Józkowy, M. Knapik, and B. Cyganek. An ensemble deep learning method with optimized weights for drone-based water rescue and surveillance. *Integrated Computer-Aided Engineering*, 28(3):221–235, 2021. https://content.iospress.com/articles/integrated-computer-aided-engineering/ica210649.

[17] R. Gorkin, K. Adams, M. J. Berryman, S. Aubin, W. Li, A. R. Davis, and J. Barthelemy. Sharkeye: Real-Time autonomous personal shark alerting via aerial surveillance. *Drones*, 4(2):18, May 2020. https://doi.org/10.3390/drones4020018.

[18] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition CVPR*, pages 770–778, 2016. https://doi.org/10.1109/CVPR.2016.90.

[19] A. Hodgson, N. Kelly, and D. Peel. Unmanned aerial vehicles (UAVs) for surveying marine fauna: A dugong case study. *PLOS ONE*, 8(11), 2013. https://doi.org/10.1371/journal.pone.0079556.

[20] D. Hoiem, Y. Chodpathumwan, and Q. Dai. Diagnosing error in object detectors. *Computer Vision, ECCV 2012 - 12th European Conference on*

*Computer Vision, Proceedings*, 7574 LNCS:340–353, 2012. https://doi.org/10.1007/978-3-642-33712-3_25.

[21] S. Hossain and D.-J. Lee. Deep Learning-Based Real-Time Multiple-Object detection and tracking from aerial imagery via a flying robot with GPU-Based embedded devices. *Sensors*, 19(15), July 2019. https://doi.org/10.3390/s19153371.

[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] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. In *2017 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3296–3297. IEEE Computer Society, 2017.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html.

[25] F. J. Kurfess, G. Nolan, K. Gounder, C. Skae, C. Daly, and D. Tan. Deep Learning at a Distance: Remotely Working to Surveil Sharks. In *ASEE 2021*, 2021. https://www.asee.org/annual-conference/2021.

[26] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine*

*Intelligence*, 42(2):318–327, 2020.
https://doi.org/10.1109/TPAMI.2018.2858826.

[27] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, and L. Zitnick. Microsoft COCO: Common objects in context. *European Conference on Computer Vision*, 2014.
https://www.microsoft.com/en-us/research/publication/microsoft-coco-common-objects-in-context/.

[28] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen. Deep learning for generic object detection: A survey. *International Journal of Computer Vision*, 128(2):261–318, 2020.
https://doi.org/10.1007/s11263-019-01247-4.

[29] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *Computer Vision – ECCV 2016*, pages 21–37. Springer International Publishing, 2016.
https://doi.org/10.1007/978-3-319-46448-0_2.

[30] C. G. Lowe, M. Blasius, E. J. Mason, and J. O'Sullivan. Historic fishery interactions with white sharks in the southern california bight. In *Global Perspectives on the Biology and Life History of the White Shark*, pages 169–186. Feb. 2012. https://www.routledge.com/Global-Perspectives-on-the-Biology-and-Life-History-of-the-White-Shark/Domeier/p/book/9781439848401.

[31] J. Redmon and A. Farhadi. YOLO9000: Better, faster, stronger. *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.
https://doi.org/10.1109/CVPR.2017.690.

[32] J. Redmon and A. Farhadi. YOLOv3: An incremental improvement. *arXiv:1804.02767 [cs.CV]*, 2018. https://arxiv.org/abs/1804.02767.

[33] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017.

[34] P. Rex. personal communication, 2021-08-26.

[35] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. https://doi.org/10.1007/s11263-015-0816-y.

[36] M. Saqib, S. Daud Khan, N. Sharma, P. Scully-Power, P. Butcher, A. Colefax, and M. Blumenstein. Real-Time drone surveillance and population estimation of marine animals from aerial imagery. In *2018 International Conference on Image and Vision Computing New Zealand (IVCNZ)*, pages 1–6, Nov. 2018. https://doi.org/10.1109/IVCNZ.2018.8634661.

[37] N. Sharma, P. Scully-Power, and M. Blumenstein. Shark detection from aerial imagery using Region-Based CNN, a study. In *AI 2018: Advances in Artificial Intelligence*, pages 224–236. Springer International Publishing, 2018. https://doi.org/10.1007/978-3-030-03991-2_23.

[38] L. Sifre. *Rigid-motion scattering for image classification.* PhD thesis, Ecole Polytechnique, Oct. 2014. http://www.cmapx.polytechnique.fr/~sifre/research/phd_sifre.pdf.

[39] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations*, 2015. https://arxiv.org/abs/1409.1556.

[40] M. Tan, R. Pang, and Q. V. Le. Efficientdet: Scalable and efficient object detection. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10778–10787, 2020. https://doi.org/10.1109/CVPR42600.2020.01079.

[41] TensorFlow Model Garden. Quick start: Distributed training on the Oxford-IIIT pets dataset on Google Cloud. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/running_pets.md. Accessed: 2021-5-31.

[42] G. Van Horn, O. Mac Aodha, Y. Song, Y. Cui, C. Sun, A. Shepard, H. Adam, P. Perona, and S. Belongie. The iNaturalist species classification and detection dataset. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8769–8778, 2018. https://openaccess.thecvf.com/content_cvpr_2018/html/Van_Horn_The_INaturalist_Species_CVPR_2018_paper.html.

[43] H. Yu, C. Chen, X. Du, Y. Li, A. Rashwan, L. Hou, P. Jin, F. Yang, F. Liu, J. Kim, and J. Li. TensorFlow Model Garden. https://github.com/tensorflow/models.

[44] X. Zhou, D. Wang, and P. Krähenbühl. Objects as points. *arXiv:1904.07850 [cs.CV]*, 2019. http://arxiv.org/abs/1904.07850.