Winter 2020

Automating the Boring Stuff: A Deep Learning and Computer Vision Workflow for Coral Reef Habitat Mapping

Jordan Patrick Pierce
University of New Hampshire, Durham

This page was intentionally left blank.

AUTOMATING THE BORING STUFF: A DEEP LEARNING AND COMPUTER VISION WORKFLOW FOR CORAL REEF HABITAT MAPPING

BY

JORDAN PATRICK PIERCE

B.S. Geography/GIS, Texas A&M University, 2016

MASTER'S THESIS

Submitted to the University of New Hampshire

in Partial Fulfillment of

the Requirements for the Degree of

Master of Science

In

Oceanography

December, 2020

Jordan Patrick Pierce

This thesis has been examined and approved in partial fulfillment of the requirements for the degree of Master of Science in Oceanography by:

Thesis Director, Dr. Jennifer Dijkstra, Associate Research Professor, Center for Coastal and Ocean Mapping, University of New Hampshire

Dr. Yuri Rzhanov, Research Professor, Center for Coastal and Ocean Mapping, University of New Hampshire

Dr. Kim Lowell, Research Scientist, Center for Coastal and Ocean Mapping, University of New Hampshire

On October 29th, 2020

Original approval signatures are on file with the University of New Hampshire Graduate School.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisors Jennifer Dijkstra, Yuri Rzhanov, and Kim Lowell for their input, their intellectual guidance, and of course, their general awesomeness throughout this entire project. Yuri, your expertise in optics and familiarity with computer vision algorithms helped guide the direction of this project from its inception; without your help I am absolutely positive that I would not have been able to complete all of what I did here. Whether our conversations were over Local Binary Patterns or foreign cinema, I always learned something from you and thoroughly enjoyed your consul. Kim, because of your background in machine learning and analytics our discussions often provided insight into aspects of my own project that I never would have considered. You really helped round out this work by approaching it from all sides, and by providing (some much needed, I might add) recommendations on how it should be documented. You both have become role models by demonstrating that the desire to learn and solve problems never needs to end.

Jenn, you helped legitimize this work by keeping it on track and focused within the domain of benthic ecology; without you this project would have never made it off the ground. As an advisor, you provided the mental and emotional support that every student needs and others would do well to emulate you. The ceaseless encouragement and abundance of opportunities that you presented are the most telling indicators of how much you genuinely care about your students and the desire for them to succeed in life. Being the first of your students to finish I say with unfaltering hesitation that you are doing an excellent job and you need to keep doing what you are doing. Together the four of us formed a perfectly well-balanced team that I for one would not have changed for anything. Again, thank you for everything that you have done.

I also want to acknowledge a broader base of people who directly helped with this project including members of CCOM, the Dijkstra Lab (Anne, Brandon, Kristen, Kaitlyn and Matt), and of course Dr. Mark Butler and his lab (especially Nick, Samantha and Emily). Finally, I want acknowledge those individuals who, now that I look back, I see played a key role in pivotal life moments such as Rita Sperry and Mr. Young, who ignited my joy for programming, Dr. Oliver Frauenfeld and Dr. Chris Houser, who introduced and encouraged participation in undergraduate research, and finally Dr. David Baker (and his eclectic lab of amazing individuals) for helping me find my way in coral reef ecology.

Oh, and Dr. Robert Ballard of course (he’s just awesome).

DEDICATION

To my friends, who've always included me.

To my teachers, who've encouraged my curiosity.

To my family, who've created the foundation from which I built my life on.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

ABSTRACT

AUTOMATING THE BORING STUFF: A DEEP LEARNING AND COMPUTER VISION WORKFLOW FOR CORAL REEF HABITAT MAPPING

By

Jordan Patrick Pierce

University of New Hampshire, December 2020

High-resolution underwater imagery provides a detailed view of coral reefs and facilitates insight into important ecological metrics concerning their health. In recent years, anthropogenic stressors, including those related to climate change, have altered the community composition of coral reef habitats around the world. Currently the most common method of quantifying the composition of these communities is through benthic quadrat surveys and image analysis. This requires manual annotation of images that is a time-consuming task that does not scale well for large studies. Patch-based image classification using Convolutional Neural Networks (CNNs) can automate this task and provide sparse labels, but they remain computationally inefficient. This work extended the idea of automatic image annotation by using Fully Convolutional Networks (FCNs) to provide dense labels through semantic segmentation. Presented here is an improved version of Multilevel Superpixel Segmentation (MSS), an existing algorithm that repurposes the sparse labels provided to an image by automatically converting them into the dense labels necessary for training a FCN. This improved implementation—Fast-MSS—is demonstrated to perform considerably faster than the original without sacrificing accuracy. To showcase the applicability to benthic ecologists, this algorithm was independently validated by converting the sparse labels provided with the Moorea Labeled Coral (MLC) dataset into dense labels using Fast-MSS. FCNs were then trained and evaluated by comparing their predictions on the test images with the corresponding ground-truth sparse labels, setting the baseline scores for the task of semantic segmentation. Lastly, this study outlined a workflow using the methods previously described in combination with Structure-from-Motion (SfM) photogrammetry to classify the individual elements that make up a 3-D reconstructed model to their respective semantic groups. The contributions of this thesis help move the field of benthic ecology towards more efficient monitoring of coral reefs through entirely automated processes by making it easier to compute the changes in community composition using 2-D benthic habitat images and 3-D models.

GENERAL INTRODUCTION

Coral reefs provide a number of ecosystem services including a high biodiversity comparable to the Amazon Rainforest [1], a habitat to one-quarter of all marine life [2], and are of cultural and economic significance to millions of people around the world; globally, coral reefs have been estimated to provide $30 B/yr in various goods and services that include tourism, coastal protection and fisheries [3]. Unfortunately, through climate change and other anthropogenic means, a number of stressors are threatening the health of coral reefs around the world. Ocean acidification, increasing sea-surface temperatures, polluted river runoff from agricultural centers, sedimentation from nearby construction projects and overfishing are a few of the stressors affecting reef systems that can cause difficulties for coral polyps to perform their primary and secondary functions such as reef building, potentially resulting in habitat loss for other organisms [2].

To rapidly assess the response of coral reefs to changing environmental conditions, a number of remote sensing methods are used. One of the most common is benthic habitat surveys where researchers collect underwater images of a coral reef using randomly placed quadrats [4]. These images are then uploaded into an annotation software tool such as Coral Point Count (CPCe), which randomly projects a number of points onto each image and tasks the user with manually labeling the class category that each point is superimposed on [5]. Coverage statistics such as relative abundance, mean, standard deviation and standard error for each annotated species can then be estimated for each image, or for the entire research area. Such point-based annotation software and analysis tools are a standard method of calculating metrics allowing habitat changes to be tracked across space and time. Nonetheless, they are expensive and time-consuming as the user must manually annotate each image. Recently, Convolutional Neural Networks (CNNs) have been adopted to automate the annotation of images, drastically reducing the amount of time and effort required by the user. The 'patch-based' image classification technique has been demonstrated as a method for assigning labels to different taxa automatically [6, 7, 8]. However, like the manual method this technique can only provide sparse labels. Hence typically less than one percent of all an image's pixels are actually provided with a label, potentially resulting in misleading coverage statistics. Ideally, coverage statistics would be calculated using dense labels (i.e., pixel-wise labels); unfortunately, this style of annotation is typically not used by benthic ecologists.

While calculating percent coverage statistics within a 2-D quadrat is the most common coral monitoring method, it fails to assess the changes in community composition as a 3-dimensional system. Coral reefs are structurally complex and facilitate diverse assemblages of organisms largely due to the niche habitats that they provide. Although they are highly intricate, advancements in computer vision have made it possible to model the structure of a reef through Structure-from-Motion (SfM) algorithms, which utilize the images collected from various viewpoints to form an accurate 3-D reconstruction. SfM gives researchers the ability to non-invasively capture the geometry of a reef structure with a high level of precision that can then be analyzed in far greater detail than with more traditional methods. However, currently there are few efficient methods for denoting which portions of the 3-D model belong to a particular class category or functional group. This means that researchers are able to model the structure of the entire habitat and observe how it changes as a whole, but are unable to record which class categories are actually responsible for causing changes in community and structure.

To help move the field of benthic ecology towards more efficient monitoring of coral reefs, this study investigated how dense semantic labels could be obtained for both 2-D images and 3-D reconstructed models through semantic segmentation with the use of Fully Convolutional Networks (FCNs). Like all deep learning algorithms, a FCN requires a non-trivial amount of labeled samples to learn from, which can often

be a significant hurdle for many studies due to the amount of time and resources that are required to create pixel-wise labels for each image. But because sparse labels are already ubiquitous within the field of benthic ecology, this thesis demonstrates how they can be repurposed with Multilevel Superpixel Segmentation (MSS) [9], which can convert them into the format necessary for training a FCN.

Thus, this thesis consists of three individual components, each of which builds off the previous. In Chapter 1, an improved version of the MSS algorithm that performs significantly faster and with classification scores that are comparable—if not better—than the current start-of-the-art is demonstrated through a comparison using the CamVid semantic segmentation benchmark data. In Chapter 2, the performance of the improved implementation—Fast-MSS—was independently validated by creating dense labels for the images of the Moorea Labeled Coral (MLC) dataset, a notoriously difficult benchmark dataset that includes three classification experiments created to test the performance of computer vision algorithms with real benthic habitat survey images. The provided ground-truth sparse labels associated with each image in the training sets were used with Fast-MSS to create dense labels that a FCN could learn from, which was then used to perform semantic segmentation on the images in the test sets. Lastly, Chapter 3 utilized the methods described in the previous two chapters and combined them with the standard SfM procedure typically used to create 3-D models of coral reefs. Beginning with unannotated images, this study outlined a workflow that results in a classified point cloud and 3-D model that could be imported into other spatial modeling or GIS software for further analysis.

CHAPTER 1: IMPROVEMENTS TO THE MULTILEVEL SUPERPIXEL SEGMENTATION ALGORITHM

INTRODUCTION

Point annotations are labels that are provided to individual pixels within an image denoting the semantic category for which they are thought to belong. Point annotations are commonly referred to as 'sparse' when only a small percentage of the total number of pixels within the image are provided with labels, and 'dense' when all of the pixels are provided with a label (i.e., pixel-wise labels). Mentioned previously, sparse labels are typically provided to images through point-based annotation software, which randomly projects numerous points on to each image and tasks the user with providing labels to the pixel each point is superimposed on. Because of the randomness in which the points are projected onto an image, the labels can be used to estimate various coverage statistics.

However, when compared to other forms of annotation, creating sparse labels can be an expensive and time-consuming process for the human annotator that results in very few pixels within an image actually being provided with labels. Using dense labels instead would ensure higher accuracies. Unfortunately, this style of annotation is not commonly used by benthic ecologists.

To provide a method for creating dense labels for images of coral reefs, in 2018 [10] developed the annotation tool *Deep Segments*, which used the Simple Linear Iterative Clustering (SLIC) over-segmentation algorithm to aggregate pixels of an image into visually homogenous regions of similar size called 'superpixels' [11]. Users then only need to provide a single class label to each superpixel, which is propagated to all of the pixels associated with it, thus reducing annotation times (Fig. 1).

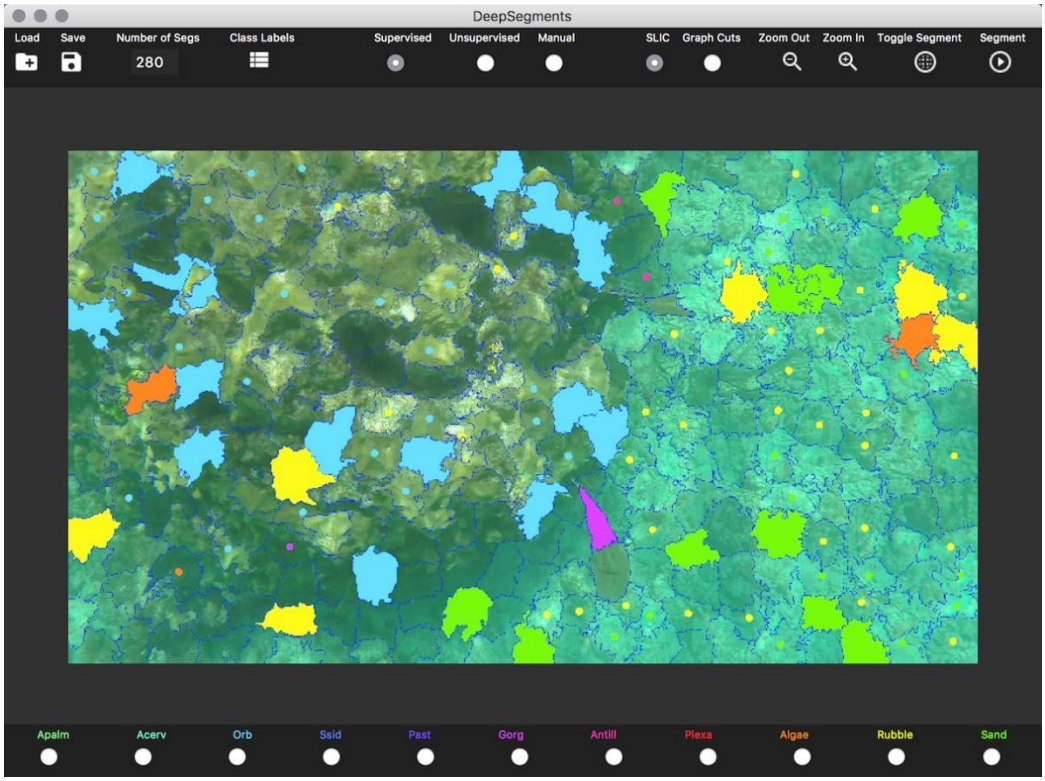

Fig. 1 – Dense labeling using Deep Segment; here the image is segmented by SLIC into 280 superpixels each composed of individual pixels that share similarities in location and color components, as indicated by the blue lines [10]. Each class label provided to a superpixel by the user was propagated to all pixels associated with it.

Alonso *et al.* [9] and [12] also explored the use of an over-segmentation algorithm for a similar purpose, but used it to propagate the labels of existing sparse labels for an image to adjacent pixels in an attempt to create dense labels automatically. Originally [12] segmented the image into a pre-defined number of superpixels, and then propagated the class label of any sparse labels that happened to have an X, Y location that lay within the boundaries of a superpixel to the associated pixels. However, the major drawback to this method was determining how many superpixels should be formed; as discussed in [9], having a large number of superpixels allows for the contours of objects to be a better fit, but it also increases the number of superpixels that are left without a label. This trade-off was later addressed in the MSS algorithm, which, as the name implies, used not one but multiple iterations of the over-segmentation algorithm.

This study makes improvements to the original MSS algorithm making it perform significantly faster and with classification scores that are comparable—if not better. Specifically, these improvements are:

1. Re-writing the algorithm and providing it with a user-friendly application program interface;
2. The use of an alternative over-segmentation algorithm, a variant called Fast-SLIC [14];
3. The method in which labels from each iteration are combined together to form a set of dense labels for the image.

The next section provides a detailed explanation of how the improved implementation—Fast-MSS—generates dense labels for an image using the existing ground-truth sparse labels. First, an overview of the Fast-SLIC over-segmentation is reviewed, followed by a description of how Fast-MSS joins the labels from each iteration of Fast-SLIC into a set of dense labels for the image. Finally, a comparison between the Fast-MSS implementation and the original is performed using the CamVid semantic segmentation benchmark dataset.

METHODOLOGY

Fast-SLIC

Fast-SLIC's methodology is based on the K-Means classification algorithm that groups data points into *K* clusters based on their relative location to one another in feature space through an iterative process in an attempt to minimize the total intra-cluster variance [13]. In the case of Fast-SLIC, the data points are the pixels that make up the image where the five features considered are their X, Y locations in image coordinate space and their three color components (i.e., *l*, *a*, and *b*) represented in CIELAB color space [14].

With Fast-SLIC, the number of clusters specified by the user is not an indication of the number of class categories present within the image, but the number of superpixels an image is to be partitioned into (Fig. 2). Because the image contains a finite number of pixels, there is an inverse relationship between the number of superpixels/clusters formed and their size. Decreasing the number of clusters results in superpixels becoming larger and thus less homogenous, whereas if the number of clusters is equal to the number of pixels in an image, then each superpixel is just a single pixel.

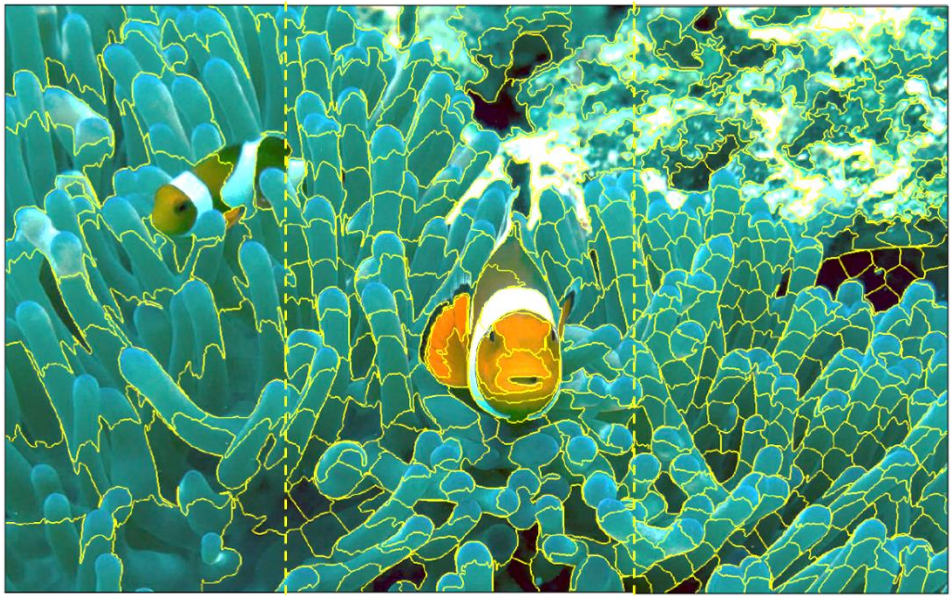

Fig. 2 – An image segmented by Fast-SLIC into approximately 100 (left), 500 (center), and 1000 (right) superpixels, each composed of individual pixels that share similarities in location and color components [14]. The relative size of each superpixel is determined by the total number of superpixels formed, whereas the compactness is controlled by a weighted function that reconsiders the importance of the color components and relative distance to neighboring pixels.

Unlike K-Means clustering, which scales exponentially with increasingly large datasets, Fast-SLIC becomes tractable even with images composed of millions of pixels by limiting the considered search space to a fewer number of potential clusters all located within closer proximity [14]. This search space *S*, is the radius in which a pixel must be for a potential cluster to consider it for inclusion (1). Let the user defined number of clusters that are formed for an image with a height and width be equal to *K*, *H*, and *W*, respectively:

$$S = \sqrt{\frac{W * H}{K}} \tag{1}$$

where the distance between a pixel and each potential cluster's centroid D_s is determined by a weighted function between the Euclidian distance in CIELAB color space d_{lab}, and image coordinate space d_{xy} as defined in (2).

$$d_{lab} = \sqrt{(l_k - l_i)^2 + (a_k - a_i)^2 + (b_k - b_i)^2}$$

$$d_{xy} = \sqrt{(x_k - x_i)^2 + (y_k - y_i)^2}$$

$$D_s = d_{lab} + \frac{m}{S} * d_{xy}$$

(2)

The weighting of importance between the color value and the relative location for pixel *i*, when being considered by a potential cluster *k*, is determined by the parameter *m*, which is provided by the user to control the level of compactness of the superpixels to be formed. For higher values of *m*, each pixel's assignment to a potential cluster is largely dependent on its proximity to the centroid in image space, resulting in a superpixel that is more compact and congruent in shape. Alternatively, lower values for *m* increase the importance of similarity in color value allowing a superpixel to be less compact and congruent in shape. A pixel's final assignment to a superpixel is determined by the minimal distance between it and all of the potential clusters considered.

Fast-SLIC follows the same methodology as the original SLIC implementation but includes optimization techniques such as color quantization, subsampling, parallelization and integer-based arithmetic. These optimizations allow the algorithm to be run on an off-the-shelf CPU with reduced latency that is comparable to implementations made to be run on a GPU [14].

Fast Multilevel Superpixel Segmentation (Fast-MSS)

The first iteration of Fast-MSS starts by using Fast-SLIC to segment the image into a relatively large number of superpixels so that each one is small enough to capture the finer details between bordering semantic groups. Then for each successive iteration, the image is segmented into a *fewer* number of superpixels making each one larger and as a result, encompassing more pixels. The number of superpixels that form during each iteration is calculated in the same way as described in [9]. Shown in (3), the number of superpixels to form N_{sp}, for any given iteration *i*, is computed by:

$$N_{sp_{(i)}} = First_{N_{sp}} \left[\left(\frac{Last_{N_{sp}}}{First_{N_{sp}}} \right)^{\frac{i}{NI}} \right] \tag{3}$$

where the number of superpixels to partition the image into during the first ($First_{Nsp}$) and last ($Last_{Nsp}$) iteration as well as the total number of iterations (*NI*) are parameters provided by the user. For example, if the user-provided values for $First_{Nsp}$, $Last_{Nsp}$, and *NI*, are equal to 1000, 10, and 10 (respectively), the following sequence N_{sp}, represents the number of superpixels formed during each iteration:

$$N_{sp} = \{1000, 599, 359, 215, 129, 77, 46, 27, 16, 10\}$$

but because (3) is an exponential equation, decreasing the number of iterations *NI*, to 5 while holding the other two parameters constant results in the sequence N_{sp}, being equal to:

$$N_{sp} = \{1000, 316, 100, 31, 10\}$$

Within a single iteration, each superpixel formed is provided with a unique identifier that is shared with all of the individual pixels that are associated with it. If an existing sparse label associated with the image happens to have an X, Y location that lies within the boundaries of a superpixel, then that label is propagated to all of the pixels associated with the superpixel. If a superpixel contains multiple sparse labels that represent different class categories, then the class category that makes up the majority is used to label all the pixels associated with it. If there are no sparse labels located within the boundaries of a superpixel, then all of its pixels associated with it are labeled as a null class. At the end of the iteration, the class labels that were propagated to adjacent pixels are stored in a 2-dimensional array with dimensions that are equal to the height *H*, and width *W*, of the original image, where each index contains the potential class label for the corresponding pixel found within the image (i.e., segmentation map).

The entire process described in the previous passage is repeated for each iteration resulting in an additional 2-dimensional array for each of the iterations. Collectively, these 2-dimensional arrays create a 3-dimensional data structure or 'stack', with the shape (*H* x *W* x *I*), where *I* is equal to the number of iterations. The original MSS implementation joined each of the 2-dimensional arrays in the stack starting with the one made during the first iteration so that the smaller superpixels that captured the finer details are not overwritten by superpixels from subsequent iterations. In the Fast-MSS implementation, the dense labels were made by calculating the statistical mode of class labels across the 3rd dimension of the stack.

As mentioned previously, partitioning an image into a larger number of superpixels results in each one being rather small, which, depending on the number of existing annotations could lead to many pixels being assigned with the null class label. If this occurs for the same pixel index for the majority of the iterations, then that pixel index will also hold the null class label in the resulting dense labels. To avoid this, when calculating the mode during the final step, in the scenario where the most common class label is the null class, it is replaced with the second most common class label instead.

Comparison using the CamVid dataset

To highlight the differences between the Fast-MSS implementation and the original, a comparison was performed using the CamVid Road Scenes dataset, a semantic segmentation benchmark used within the domain of autonomous vehicles. This version of the CamVid dataset contains 600 images with the same dimensions (360 pixels x 480 pixels) depicting eleven different class categories (e.g., car, building, road), and includes corresponding dense labels for each image (Fig. 3, [15]).

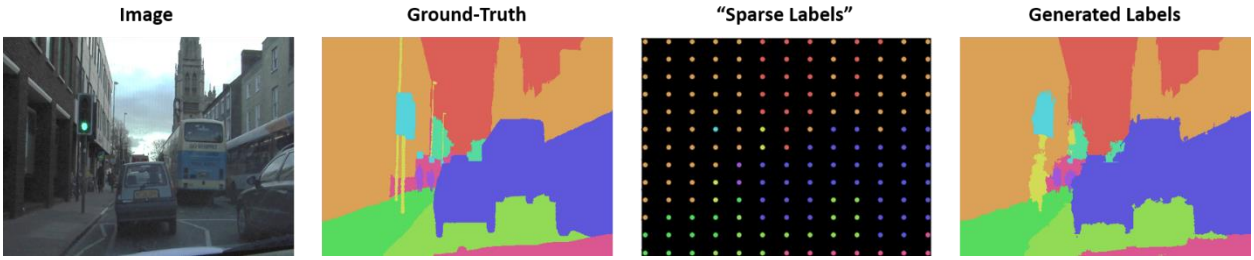

Fig. 3 – Generating dense labels using Fast-MSS. From left to right: an image from the CamVid dataset, the corresponding ground-truth dense labels, the synthesized sparse labels, and the dense labels generated

from those sparse labels using Fast-MSS. Note that labels are color-coded based on class category, and that sparse points are enlarged for display purposes.

Three trials were conducted in which sparse labels were synthesized for each image by uniformly sampling a different percentage of the ground-truth dense labels following a grid formation (see Table 1). From these sparse labels, Fast-MSS and the current state-of-the-art were used to generate dense labels that were then compared with the original dense labels (ground-truth).

The metrics used to quantify the differences between the resulting dense labels and the ground-truth data were pixel accuracy (PA), mean pixel accuracy (m-PA) and mean Intersection-over-Union (m-IoU), calculated using (4), (5), and (7), respectively:

$$PA = \frac{TP + TN}{TP + FP + TN + FP} \tag{4}$$

$$m\text{–}PA = \frac{\sum_{i=1}^{N} PA_i}{N} \tag{5}$$

$$IoU = \frac{TP}{TP + FP + FN} \tag{6}$$

$$m\text{–}IoU = \frac{\sum_{i=1}^{N} IoU_i}{N} \tag{7}$$

For each equation *TP*, *TN*, *FP*, and *FN* represents the True Positive, True Negative, False Positive and False Negative rates, respectively. PA represents the number of correct instances over the total number of instances, whereas m-PA represents the PA per class *N*, averaged together unweighted. Intersection-over-Union (IoU) is calculated by (6), with m-IoU being the IoU per class *N*, averaged together unweighted. Also included in the comparison is the amount of time required to generate dense labels for all 600 of the images in the dataset, and an approximation of the amount of time required to process each image.

The metrics and the original MSS algorithm were implemented using the code published in [9] with the recommendations of 1500 and 50 for the initial and final number of superpixels across 15 iterations [12]; the Fast-MSS implementation used 7500 and 80 for the initial and final number of superpixels across 20 iterations.

RESULTS

CamVid Classification Scores

Table 1 shows the results from the comparison between Fast-MSS and the original implementation using the CamVid dataset. Alongside each metric is reported the percentage and actual number of pixels that were sampled for synthesizing the sparse labels. Scores are colored red, yellow, or green if they are lower, the same, or higher than the other implementation's score, respectively

Table 1 – Comparison between Fast-MSS and the original implementation using CamVid.

Implementation	% of total pixels	Number of pixels	PA	m-PA	m-IoU	Time (seconds)	Time Per Image (seconds)
[12]	0.1	180	0.87	0.70	0.57	9613	16.02
	0.5	814	0.91	0.77	0.65	9787	16.31
	1.0	1664	0.91	0.79	0.67	9862	16.43
Fast-MSS	0.1	180	0.87	0.72	0.56	984	1.64
	0.5	814	0.91	0.82	0.67	1338	2.23
	1.0	1664	0.91	0.84	0.68	1386	2.31

Note: All trials were conducted on the same PC with an Intel i7-8700 processor. Scores are colored red, yellow, or green if they are lower, the same, or higher than the other implementation's score, respectively.

Abbreviations: PA, pixel accuracy; m-PA, mean pixel accuracy; m-IoU, mean Intersection-over-Union. For each metric 1.0 represents a perfect score.

Discussion

For each of the three trials Fast-MSS was comparable to—if not better than—the current state-of-the-art with regards to m-PA and m-IoU, and by using Fast-SLIC the amount of time needed to produce dense labels was drastically reduced.

Empirically it was found that Fast-MSS does well with the previously mentioned parameter values for this particular dataset, but it is recommended that users try different parameters for other datasets as these values depend on the size of the image and the number of sparse labels provided.

The improved implementation of the MSS algorithm demonstrates how a dataset that contains only sparse labels can quickly and easily be converted to dense labels that are accurate enough to be used for calculating coverage statistics, or train a deep learning semantic segmentation algorithm as is shown in the following chapters.

The improvements to the MSS algorithm (i.e., Fast-MSS) were made both publicly available and easy to use following a simple application-program interface (API) written entirely in Python. The code and examples for its use can be found at github.com/JordanMakesMaps/Fast-Multilevel-Superpixel-Segmentation

CHAPTER 2: SEMANTIC SEGMENTATION OF THE MOOREA LABELED CORAL DATASET

INTRODUCTION

As environmental conditions for coral reefs continue to change, it is critical that researchers are able to regularly assess these habitats and as quickly as possible. Some of the more common methods for observing these habitats include satellites, unmanned aerial vehicles (UAVs), diver-towed sleds, remotely operated vehicles (ROVs), and SCUBA. With these tools researchers are able to obtain a plethora of high-resolution imagery data, but often at a pace that far exceeds the rate in which a human can annotate them. Quite often studies will allocate a considerable amount of resources to acquire data only to have them stored in a database for extended periods of time, sometimes unutilized because of expensive or inefficient methods of processing.

In an attempt to overcome this bottleneck, researchers have begun looking into techniques for automating the annotation of coral reef imagery using deep learning and computer vision algorithms. However, deep learning algorithms require a non-trivial amount of expertly labeled data to learn from; thus, to aid researchers in their development of these recognition algorithms, in 2012 Beijbom *et al.* published the Moorea Labeled Coral (MLC) dataset to serve as the first large scale benchmark to gauge the progress of coral reef image classification algorithms [6]. The dataset is composed of 2,055 images taken of the same sites across three years (2008-2010) with approximately 400,000 manually annotated labels. Outlined with it are three image classification experiments that use the nine most abundant class categories to test an algorithm's ability to generalize across time.

Unlike other image classification benchmark datasets that assign a single label to an entire image, the MLC dataset provides roughly 200 sparse labels with each image to assist in the advancement of patch-based image classifiers. When trained, these classifiers should be able to provide sparse labels to novel images automatically, and ideally, drastically reduce the amount of effort required to annotate data collected during future studies.

Beijbom *et al.* set the baseline scores for the three patch-based image classification experiments by using handcrafted feature descriptors that take into account both color and texture by using a Maximum Response (MR) filter bank with the Bag of Visual Words (BoVW) algorithm [6]. They found that representing each image in CIELAB color space and using color channel stretch yielded the best results as a pre-processing method, and that combining features from various scales increased the classification accuracy further (see Table 2).

In 2015, Mahmood *et al*. [7] surpassed the results published by [6] by using features extracted from VGGNet [16], a CNN previously trained on the ImageNet dataset. They incorporated information at multiple scales by using what they termed the 'Local-Spatial Pyramid Pooling' technique, which extracted multiple patches of different sizes all centered on the same annotated point, later combining them into a single feature descriptor using a max pooling operation [7].

The current state-of-the-art for patch-based image classification was created in 2018 by [8]. They used a custom CNN called the Multipatch Dense Network (MDNet) that learned class categories at multiple scales and adopted the use of densely connected convolutional layers to reduce overfitting. MDNet extracted features from image-patches of different sizes in parallel, later concatenating them together to create a final descriptor for each annotated point. This technique allowed them to train the CNN end-to-end to learn information at different scales without having to perform costly resizing operations on each patch.

Table 2 – Global classification accuracies from previous and current state-of-the-art methods for each of the three experiments outlined in the MLC benchmark dataset.

MLC Benchmark	***Experiment 1***	***Experiment 2***	***Experiment 3***
Beijbom *et al.* [6]	74.3%	67.3%	83.1%
Mahmood *et al.* [7]	77.9%	70.1%	84.5%
Modasshir *et al.* [8]	**83.4%**	**80.1%**	**85.2%**

To go beyond patch-based image classification and to push the field towards a more useful form of annotation for coral reef imagery, the present study demonstrated how a dataset with ground-truth sparse labels could be used to perform semantic segmentation on previously unannotated images automatically. Using the MLC dataset, sparse labels associated with each image in the training sets were used with Fast-MSS to create dense labels that a deep learning model could learn from, and then be used to perform semantic segmentation on the images in the test sets. A thorough literature review suggests that this dataset has only been used to assess the accuracy for patch-based image classification algorithms, making this work the first to adapt it for the purposes of semantic segmentation.

In the next section an adaptation to the MLC experiments for the purpose of semantic segmentation is explained, followed by a discussion of the role of the patch-based image classifier technique and how it was used to provide additional sparse labels to each image. Next the specifications of deep learning models explored and the training procedure are outlined, followed by the results for the three experiments.

METHODOLOGY

Defining the Benthic Quadrat

This study performed the same three experiments as originally outlined in [6], but included the 'Off' class category signifying the location of the metal quadrat frame within each image. As was mentioned in [17] there are inconsistences in how annotators chose to label these points. Depending on the situation, some annotators would label points superimposed on the quadrat or the transect tape as 'Off', whereas others would label the points on the quadrat as 'Off' but ignore the presence of the tape; however, most often points clearly superimposed on the quadrat or transect tape were labeled with the class category that was assumed to be underneath them.

Although this class category is not one of those that is included in the original experiments, through preliminary analysis it was found that by allowing the deep learning model to learn the difference between the quadrat and all of the other classes, the overall quality of the predictions improved. However, because of the inconsistences in how the original 'Off' points were labeled, this class category was redefined. The original sparse labels belonging to 'Off' were discarded and replaced by providing the pixels along the perimeter of each image with 'Off' sparse labels instead (see the pink colored points in Fig. 4), while the transect tape was ignored entirely. These artificial sparse labels placed along the edge of each image worked

well to help generate dense labels for an 'Off' class category that the deep learning models could then learn from, but they were not included in any of the experiments or used when calculating any of the metrics.

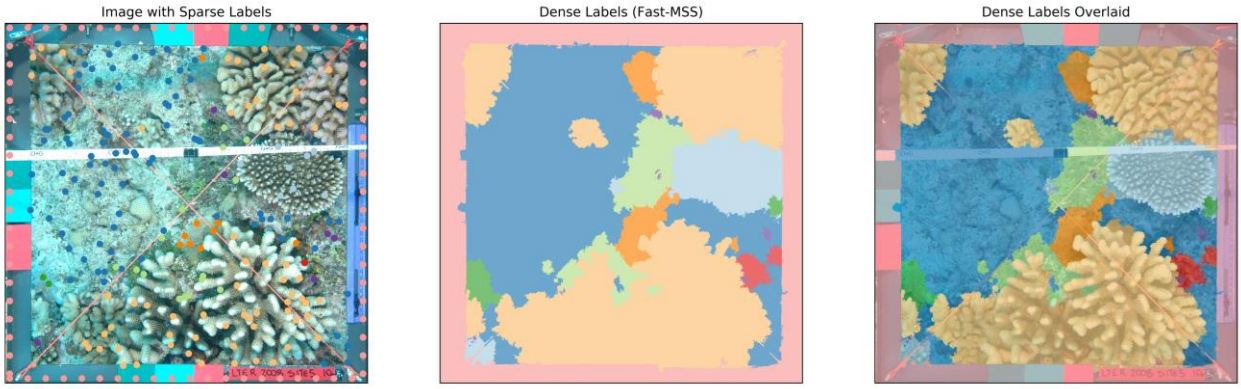

Fig. 4 – A side-by-side comparison between an image in the MLC dataset with its original sparse labels (left), the dense labels created using Fast-MSS when supplied with additional sparse labels (center), and those same dense labels overlaid on the original image (right). Artificially created 'Off' points were placed along the perimeter of the image (left) to help generate dense labels representing the metal quadrat frame, but not the transect tape. Note that labels, both sparse and dense, are color-coded based on class category.

Creating Dense Labels from Sparse Ground-Truth

During the comparison using the CamVid dataset in Chapter 1, up to 0.1% of the total amount of pixels in an image were sampled and used to mimic the presence of sparse labels; however, the MLC dataset has far fewer sparse labels available (~0.005%). Therefore, this study investigated if a patch-based image classifier could be used as a reliable method for adding additional sparse labels to each image automatically, and if doing so helped increase the classification scores of the resulting dense labels.

Thus, two sets of dense labels were made for each image: one that was supplied with additional sparse labels using a patch-based image classifier, and the other using only the original ground-truth sparse labels that were provided with the MLC dataset. These two sets of sparse labels were converted into dense using Fast-MSS and used to train two sets of FCNs whose classification scores on the test set for the three experiments were used to validate this method (Fig. 5).

To avoid data contamination and biasing the FCNs, three different patch-based image classifiers were created: one for each of the MLC experiments. Classifiers were only trained on patches extracted from images that belonged to the same experiment to which they would later provide additional labels, and classifiers were only trained on patches that were extracted from images within the experiment's training set and not the testing set. Following the method outlined in [6], [7], and [8], image classifiers were trained on patches centered on each of the original ground-truth sparse labels associated with every image in each experiment's training set. A preliminary analysis showed that classifiers trained on patches with dimensions of 112 x 112 pixels performed better than those trained on smaller sized patches.

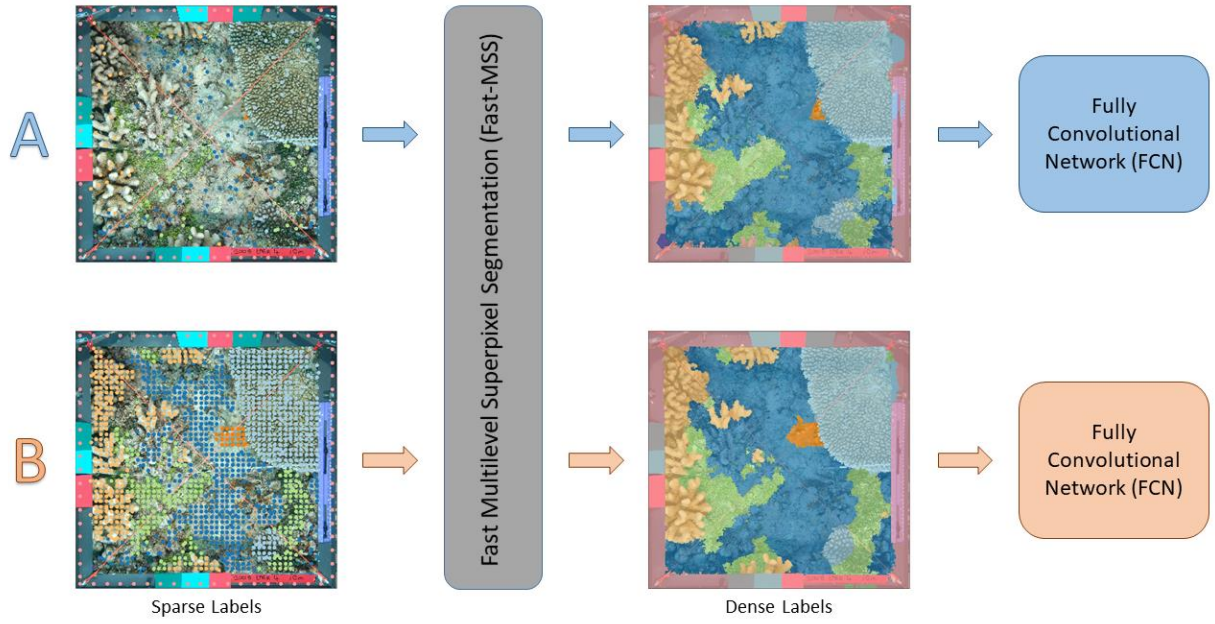

Fig. 5 – A flow chart showing how the original sparse labels (A) and those created by a patch-based image classifier (B) were used with Fast-MSS to generate two separate sets of dense labels for each training image of an experiment. These were then used to train two separate FCNs whose accuracy on the test set for the same experiment provided validation for the use of the CNN.

Providing additional labels involved first uniformly extracting patches of 112 x 112 pixels from each image in the training set following a grid formation (Fig. 6). In total, approximately 2000 patches were sampled from each training image, representing potentially 2000 additional labels, or roughly 0.05% of the total number of pixels in the image. These extracted patches were then passed to a classifier as input. The output for each was a corresponding vector representing the probability distribution of class categories to which the center-most pixel of the patch likely belonged. For each patch the extracted location, the presumed class label, and the difference between the two highest probability distributions (i.e., top-1 and top-2 choices) were recorded. This difference in "top 2" probabilities is the confidence level of the classifier when making the prediction. If the difference was small, the classifier is less confident about its top-1 choice (i.e., the presumed class label).

The difference in "top 2" probabilities were used to filter out sparse labels that were more likely to have been misclassified. By setting a confidence threshold value of 0.5, approximately 15% of those additional labels were removed from each image. A second filter removed any sparse labels with class categories that were not already recorded in the image by the human annotator. Any additional labels that remained through this filtering process were concatenated to the original ground-truth sparse labels associated with the image to create the second set of sparse labels; the first set used only the original ground-truth sparse labels. Both sets were then provided with the points labeled 'Off' as explained in the previous section.

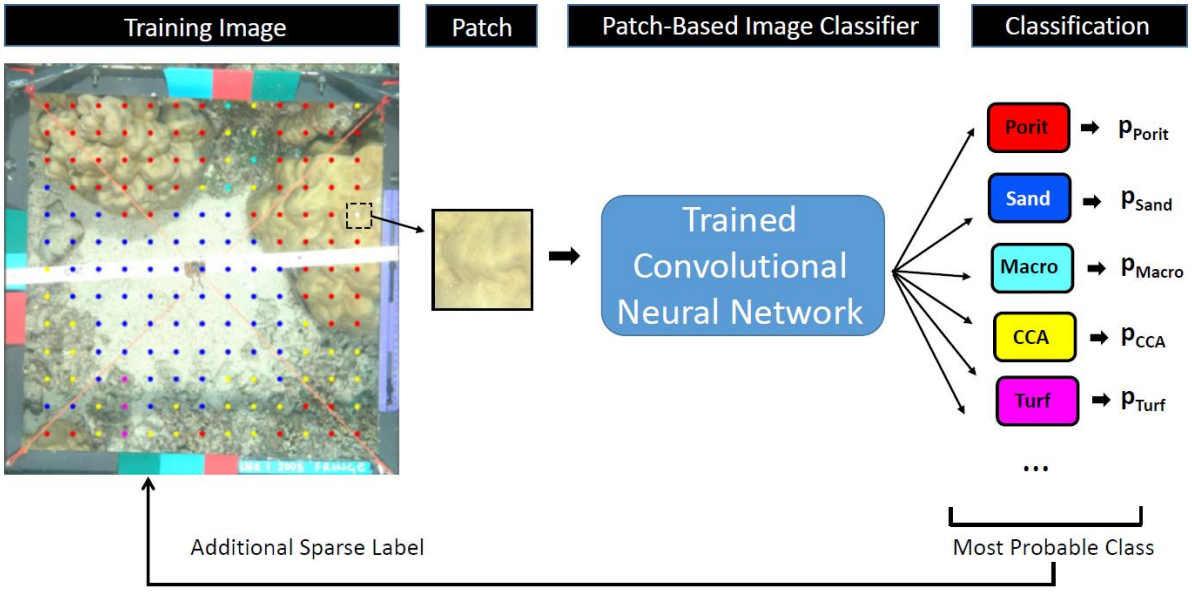

Fig. 6 – A diagram illustrating how a patch-based classifier was used to provide additional sparse labels to each image in an experiment's training set. After the classifier was trained, 112 x 112 pixel patches were uniformly sampled from the training images following a grid formation and passed to the classifier as input. If specific criteria were met, then the presumed class label was provided to the center-most pixel where the patch was extracted. Note that for display purposes, the number of sparse labels provided to the training image is significantly less that what was done in the study.

These two sets of sparse labels were then converted into two sets of dense labels using Fast-MSS. When generating dense labels for either sets, the number of superpixels to form during the first and last iteration were 2000 and 100, respectively, and iterations was set to 20.

The accuracy of these dense labels could not be validated quantitatively as the MLC dataset does not provide any ground-truth dense labels, just sparse. Because Fast-MSS used these same sparse labels to generate the dense labels, relying on them for validation purposes would result in deceivingly high accuracies. Instead the dense labels were evaluated by using them as training data for multiple FCNs, which were then compared based on their classification scores using the original ground-truth sparse labels within the test set for each of the experiments just as was done for the benchmark studies cited earlier and whose results are presented in Table 2.

Experiments

This study used the same experimental setup to split data as outlined in [6]: for experiment one, K-fold cross-validation was used to split the 2008 data into three folds, two of which were used for training and the remaining was used for testing; this was done three times so that each fold was used for both training and testing, and accuracy scores were later averaged. Experiment two used all of the data from 2008 for training and tested on data from 2009, and experiment three used all of the data from 2008 and 2009 for training, and tested on data from 2010. This same setup was used for splitting the data for both the patch-based image classifiers and the semantic segmentation models.

Metrics for each of the three experiments were calculated by using the original sparse labels within the test sets as ground-truth, compared against what the deep learning model predicted for the corresponding image. However, because the ground-truth was in the form of sparse labels and the deep learning model produced dense labels, only the pixel indices that were provided with labels by the original MLC annotators could be used to validate the deep learning model's predictions (Fig. 7). Furthermore, because the 'Off' class category is not included in the original experiments, any of the dense labels that were predicted as 'Off' (approximately 1%) were replaced with the top-2 choice label instead.

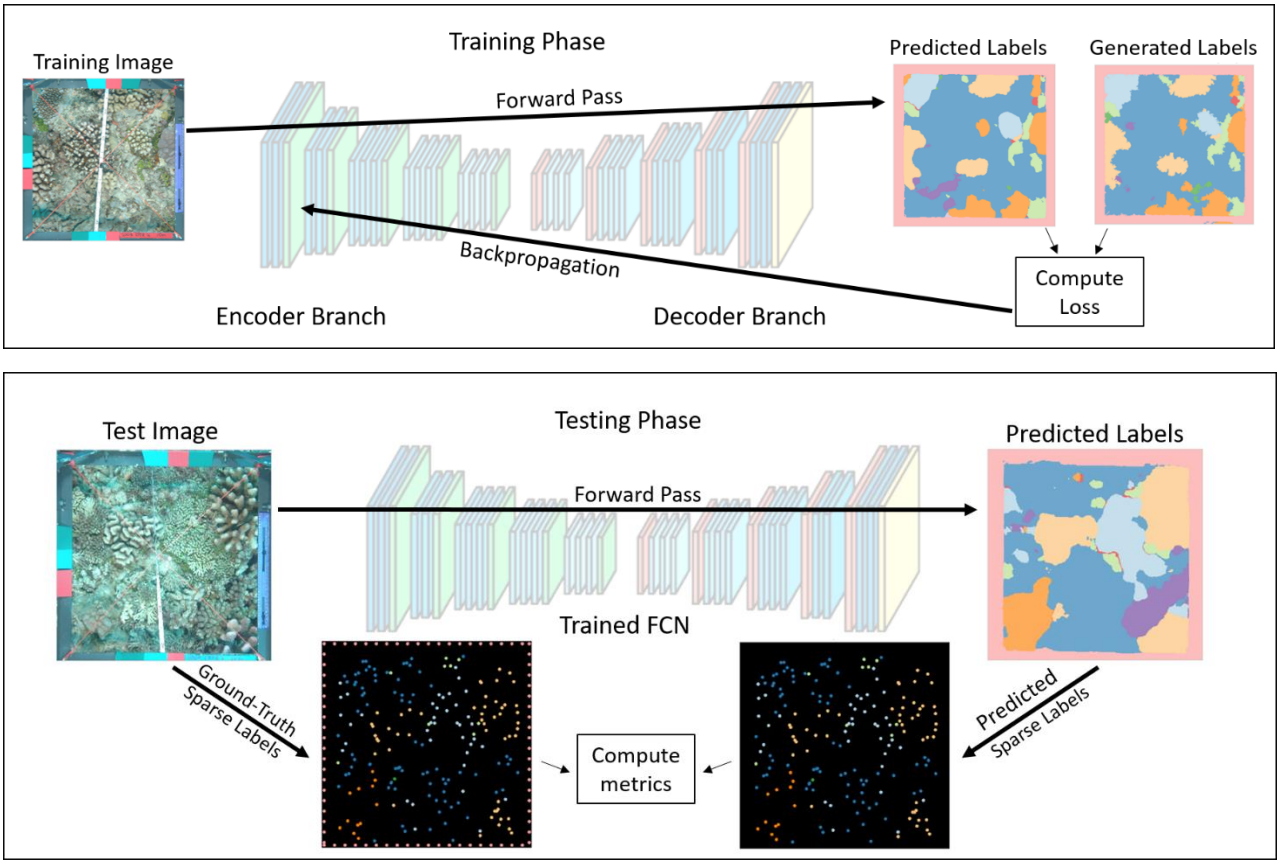

Fig. 7 – A diagram inspired by a figure in [9] illustrating the use of a FCN architecture on the MLC dataset for semantic segmentation. Models were trained with images and dense labels generated by Fast-MSS, and metrics were calculated by comparing the ground-truth sparse labels for each image in the test set against the corresponding pixel indices within the predicted dense labels for the same image.

For the sake of consistency, the same metrics used in [6], [7], and [8] were reported; these include classification accuracy, precision, and recall, which were calculated using (8), (9), and (10), respectively.

$$\underset{Accuracy}{Classification} = \frac{TP + TN}{TP + FP + TN + FN} \tag{8}$$

$$Precision = \frac{TP}{TP + FP} \tag{9}$$

$$Recall = \frac{TP}{TP + FN} \tag{10}$$

Classification accuracy was computed by calculating the subset "accuracy", which requires that the predicted label match exactly the ground-truth label. Precision and recall were calculated by computing a confusion matrix that incorporated all of predictions and ground-truth samples from the test set for an experiment, calculating the metric for each individual class, and then averaging them together to obtain the final score (i.e., macro-averaged). It should be noted that the MLC dataset is heavily imbalanced (see Fig. 8) and these metrics do not take into consideration the frequency of each class category.

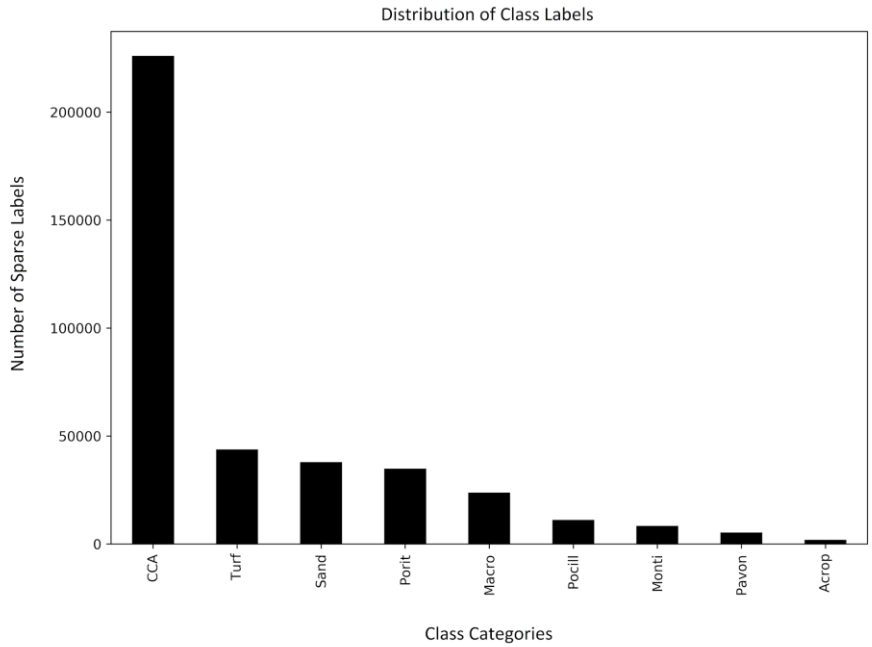

Fig. 8 – The distribution of the nine most abundant class categories used in the three patch-based image classification experiments associated with the MLC dataset, making up approximately 94% of all sparse labels. Note the class category 'Off' is not included as it is not used in the experiments or when calculating metrics.

Model Training

For patch-based image classification, after preliminary analysis the NASNet architecture [18] was chosen to train three different classifiers, one for each of the experiments. Models were initialized with weights pre-trained on ImageNet and consisted of the encoder followed by a max pooling operation, a dropout layer (80%) and then finally a single fully connected layer with 10 output nodes. Patches that were extracted from the original images with dimensions of 112 pixels x 112 pixels were resized to 224 pixels x 224 pixels to match the input size requirement of each encoder, normalized between 0 and 1 using patch-specific max-min normalization, and then heavily augmented (e.g., adding noise, blurring, sharpening, altering contrast) using *ImgAug* [19] to reduce overfitting.

Categorical-cross entropy was used as the loss function along with the optimizer Adam with an initial learning rate of 10^{-3} using the *ReduceLROnPlateau* callback to reduce the learning rate by a factor of 0.5 for every three epochs in which the validation loss failed to decrease. Because the problem is multi-categorical classification, the activation function used was softmax; models were trained for 50 epochs with a batch size of 32 as this represented the maximum amount of memory that could be allocated during training by the GPU being used.

For semantic segmentation the U-Net architecture was used, which, unlike the original FCN architecture outlined in [20], gradually upsamples feature maps by using transposed-convolutional layers and skip connections to increase the resolution of the model's prediction [21]. Because determining the optimal encoder for a given dataset is often heuristic, the following eight encoders were experimented with to obtain a range of performances: DenseNet-201 [22], EfficientNet-b0 and EfficientNet-b4 [23], InceptionV3 [24], ResNet-34, ResNet-50, ResNet-101 [25], and VGGNet-19 [16]. All segmentation models were implemented in Python using the *Segmentation Models* library provided by [26].

Each of the encoders were initialized with pre-trained weights from the ImageNet dataset and were left frozen (i.e., immutable) for the entire process; only the weights in the decoders were updated during training. Images were pre-processed using the methodology recommended for each encoder, and dense labels were converted into one-hot-encoded form with a shape of (B x H x W x C) where B and C represent the batch size and the number of class categories, respectively. Augmentations were randomly performed on each sample using *ImgAug* in the form of simple affine transformations (flips, flops, rotations) and a channel shuffle operation that randomly swaps the location of each channel in the image.

Preliminary analysis indicated that when training with larger images, the resulting models produced better predictions, but due to differing computational requirements for each of the encoders and the amount of memory that could be allocated by the GPU, images were reduced in size to 736 x 736 during training and testing for all encoders. Consequently, this resulted in the batch size having to be equal to one (i.e., a single image).

Soft-Jaccard was used as the loss function, which is a differentiable proxy that attempts to maximize the Intersection-over-Union metric [27]. The optimizer employed was Adam, with an initial learning rate of 10^{-3} along with the *ReduceLROnPlateau* callback using the same settings as described before. The activation function was softmax, and models were trained for 20 epochs.

All model training was performed on a PC equipped with a NVIDIA GTX 1080 Ti GPU and an Intel i7-8700 CPU, using the Keras deep learning framework and the Tensorflow numerical computational library.

RESULTS

Classification Performance

The results of the trained FCNs on the three MLC experiments can be seen in Table 3 and 4. Each of the encoders were compared, as well as the encoders trained using only the MLC sparse labels against their counterparts that were trained with the MLC plus the patch-based sparse labels. As seen in Table 3, the general trend shows that models using the DenseNet and EfficientNet encoders performed with a higher classification accuracy than the others.

More interesting is the difference in classification accuracy between models that were trained with additional sparse labels against their counterparts that were trained without them. Models that were trained with the additional sparse labels provided by the patch-based classifier saw an increase in accuracy by approximately 3%, on average.

Table 3 – Classification accuracies for each model on all three experiments, trained with and without additional sparse labels.

Encoder	Accuracy		
	Exp 1	Exp 2	Exp 3
MLC Sparse Labels			
DenseNet-201	0.716	0.626	0.802
EfficientNet-b0	0.709	0.620	0.797
EfficientNet-b4	0.703	0.613	0.827
InceptionV3	0.662	0.580	0.795
ResNet-34	0.676	0.630	0.805
ResNet-50	0.668	0.612	0.787
ResNet-101	0.672	0.612	0.771
VGGNet-19	0.618	0.571	0.771
MLC + Patch-based Sparse Labels			
DenseNet-201	**0.754**	0.614	**0.839**
EfficientNet-b0	0.737	**0.649**	0.824
EfficientNet-b4	0.737	0.645	0.836
InceptionV3	0.673	0.570	0.811
ResNet-34	0.714	0.642	0.814
ResNet-50	0.696	0.595	0.809
ResNet-101	0.686	0.617	0.785
VGGNet-19	0.648	0.606	0.773

Note: Encoder scores are colored red, yellow, or green if they are lower, the same, or higher than the score of its counterpart for the same experiment, respectively. Bold numbers show the best performing encoder over all trials, with 1.0 representing a perfect score.

The same trend among models using the DenseNet and EfficientNet encoders can also be seen with regards to precision and recall (Table 4). Moreover, the average increase for both of these metrics for models trained with additional sparse labels was approximately 7% compared to those trained without.

Table 4 – The mean precision and mean recall for each model on all three experiments, trained with and without additional labels.

Encoder	Precision			Recall		
	Exp 1	Exp 2	Exp 3	Exp 1	Exp 2	Exp 3
MLC Sparse Labels						
DenseNet-201	0.598	0.476	0.497	0.549	0.473	0.564
EfficientNet-b0	0.574	0.517	0.493	0.531	0.483	0.521
EfficientNet-b4	0.565	0.523	0.538	0.517	0.485	0.575
InceptionV3	0.500	0.435	0.481	0.457	0.451	0.471
ResNet-34	0.567	0.519	0.478	0.520	0.508	0.472
ResNet-50	0.476	0.505	0.528	0.498	0.503	0.533
ResNet-101	0.533	0.469	0.476	0.491	0.484	0.504
VGGNet-19	0.481	0.402	0.425	0.391	0.401	0.363
MLC + Patch-based Sparse Labels						
DenseNet-201	**0.632**	0.517	0.584	**0.593**	**0.602**	0.604
EfficientNet-b0	0.607	0.561	0.565	0.559	0.494	0.580
EfficientNet-b4	0.631	**0.626**	**0.597**	0.563	0.554	**0.627**
InceptionV3	0.541	0.453	0.535	0.502	0.474	0.494
ResNet-34	0.524	0.547	0.494	0.518	0.475	0.594
ResNet-50	0.540	0.487	0.513	0.520	0.461	0.538
ResNet-101	0.553	0.531	0.560	0.521	0.505	0.499
VGGNet-19	0.518	0.466	0.461	0.397	0.363	0.392

Note: Encoder scores are colored red, yellow, or green if they are lower, the same, or higher than the score of its counterpart for the same experiment, respectively. Bold numbers show the best performing encoder over all trials, with 1.0 representing a perfect score.

Discussion

The increase in classification scores between models trained using only the MLC sparse labels against their counterparts that were trained with the MLC plus the patch-based sparse labels is in agreement with what was observed from the comparison using the CamVid dataset in Chapter 1: additional sparse labels can positively affect the quality of the resulting dense labels, and deep learning models trained on them are also likely to achieve gains in classification scores. This validated the use of the patch-based image classifier in this study and also provides evidence for its use in future studies, which (as suggested by results presented in Table 1) may save researchers a significant amount of time and resources by automating the task of sparse image annotation and improve coral reef monitoring and assessment.

The top scoring FCNs are suitable for many benthic ecology applications, and would be expected to increase in performance even further if provided with additional images to learn from. Typically, the predicted segmentation maps (i.e., dense labels) are validated by comparing them to ground-truth segmentation maps,

but because the MLC only has sparse labels these were used instead. This form of validation does provide some indication of performance, although it is not the most beneficial format as it does not take into account the other 99.5% of labels that were predicted by the deep learning model. Looking at a randomly sampled segmentation map produced by a DenseNet model, the predicted sparse labels that tend to be misclassified are those that are located along the borders of different semantic groups (Fig. 9). This is not unexpected as the transition between neighboring class categories is often not sharp in contrast, but instead is usually fuzzy and complex.

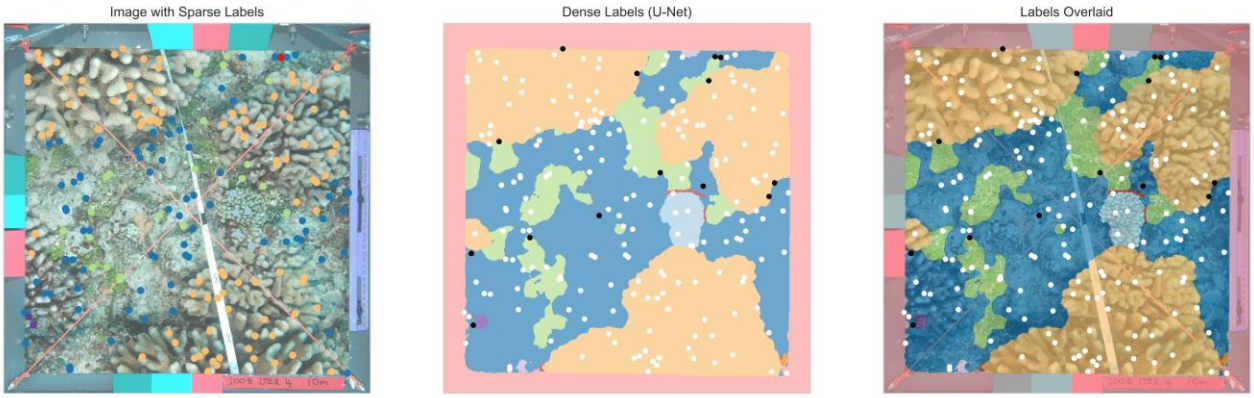

Fig. 9 – A side-by-side comparison between an image and its labels. From left to right: the original image with ground-truth sparse labels superimposed, the dense labels predicted by a deep learning model, and those same dense labels overlaid on top of the original image. The sparse labels in the last two columns are colored white if predicted correctly, or colored black if incorrectly; note that most of the incorrect predictions appear along the borders of semantic groups.

However, some of the misclassified predictions made by the FCNs could also be attributed to incorrect ground-truth labels, which were created in error for the same exact reason. It has already been established in [17] that 'Off' points were labeled inconsistently, where some that were clearly on the quadrat were provided with labels of different class categories that were nearby; the same is also likely true for labels of other class categories. Unfortunately, without properly annotated segmentation maps to serve as ground-truth these questions cannot be completely addressed.

CHAPTER 3: SEMANTIC SEGMENTATION WORKFLOW FOR THE CLASSIFICATION OF 3-D RECONSTRUCTED CORAL REEFS

INTRODUCTION

Coral reefs are complex 3-dimensional structures that promote the assemblage of diverse groups of organisms by providing niche habitats for prey seeking refuge from predation. Although images collected through benthic quadrat surveys are routinely used to evaluate community composition, they fail to capture the changes that occur to a coral reef when considering the 3-dimensional structure, arguably one of its most important attributes.

Fortunately, the new standard for obtaining 3-D measurements has emerged in the form of Structure-from-Motion (SfM) algorithms, which utilize images collected from multiple angles to estimate depth and provide the ability to reconstruct 3-D models of coral colonies, or even entire reefs [28, 29, 30]. Because of the relative ease and the accuracy of the models it can produce, SfM has opened new opportunities for exploring how the physical structure of a reef changes across space and time at unmatched levels of precision. However, one drawback of SfM is it lacks an inherent mechanism for denoting which portions of the reconstructed model belong to a particular class category or functional group. This means that (1), 3-D coverage statistics relating to the composition of species cannot be calculated and (2), any metric that describes the structure of a reef can only be resolved at the model scale. This inability severely hinders the potential to understand any connections that may exist between changes in habitat structure and its community composition, such as those that occur during coral-algal phase shifts.

Currently, there is only one other known technique that can be used to classify the 3-D reconstruction of a coral reef. Published in 2020, Hopkinson *et al.* [31] demonstrated how a CNN can be trained on, and used to classify the images that are used in the SfM reconstruction. Their methodology involved using the camera transformation matrices that are created during the camera alignment phase of the reconstruction process to identify all of the images that correspond to each of the elements that make up the 3-D model. Then, after training a CNN on a representative subset of those images, it was used to classify all of the remaining images that are associated with an element; because multiple viewpoints correspond to every element, a majority-voting scheme was used to determine the final semantic label for each one. Conceptually, this technique is not unlike a 3-D version of classifying each individual pixel within an image one-by-one, and can be computationally demanding especially for high resolution models made up of millions of elements, each of which may be associated with 10+ images.

This study demonstrated a more efficient method that used the pixel-wise labels (i.e., dense labels) for each of the images used in the reconstruction process to classify a 3-D reconstructed model. Because each image only needs to be provided with a corresponding set of labels once, this method scales linearly and can be used to provide semantic labels to a 3-D model regardless of its size or resolution.

Although providing dense labels to thousands of high-resolution images usually requires a significant amount of time, this study drastically reduced the amount of effort needed by developing a workflow that used the deep learning and computer vision algorithms that were described in the previous two chapters.

The next section describes how the dataset used in this study was initially collected, and the process in which the 3-D model was reconstructed using SfM photogrammetry software. Next, a comprehensive walkthrough for each step of the workflow is provided, followed by an overview of the class categories that were defined for this study, and then an explanation of how the deep learning models from the workflow

were trained. Finally, how the 3-D models were classified is described, followed by an analysis, the results and lastly, a discussion.

METHODOLOGY

Image acquisition

Video data were collected of a coral patch reef located near Cheeca Rocks (24.9041°N, 80.6168°W) in the Florida Keys National Marine Sanctuary (Fig. 10) using a custom frame equipped with two GoPro Hero 7 Black video cameras mounted approximately one meter apart, both encased in waterproof housing with a flat-view port. Videos were collected by SCUBA divers who swam 1-4 m above the patch reef in a boustrophedonic (i.e., lawnmower) pattern with cameras angled towards nadir followed by a second pass with cameras angled at approximately 45 degrees to obtain oblique views. Finally, divers were instructed to swim freely at various depths and distances from the patch reef, completely encircling it in an attempt to acquire footage of any occluded areas on the reef. Videos were recorded in 4K HD (2160 pixels x 3840 pixels) and at 24 frames per second (fps) in wide field-of-view mode with *HyperSmooth* stabilization set to active. Twenty-three coded targets were strategically placed on and around the site to assist in estimating camera locations and the calibration coefficients during the reconstruction of the 3-D model.

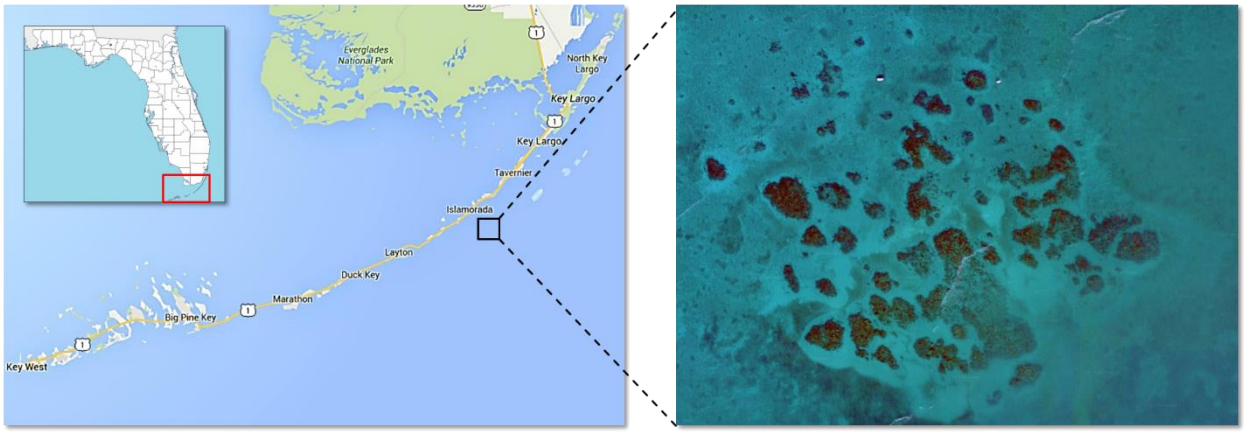

Fig. 10 – A Google Map of the Florida Keys (left) and satellite imagery obtained from Google Earth showing Cheeca Rocks (right), which is located approximately one mile southeast of the Upper Matecumbe Key within the Florida Keys National Marine Sanctuary.

The video survey covered an area of approximately 5 m x 5m with divers swimming between 8 ± 2 m deep, was conducted in July of 2019 while water visibility was greater than 35ft, and used only ambient light. Post-capture, 2180 images were extracted from the video footage by sequentially sampling one in every eight frames, allowing for enough forward overlap (> 60%) between successive images.

Structure-from-Motion Photogrammetry (SfM)

The 3-D model in this study was created using the SfM photogrammetry software (Agisoft Metashape Pro 1.6, previously Photoscan) following a similar methodology outlined by [30], with a few additional steps that were found to enhance model quality [34]. The patch reef, as seen in Fig. 11, was reconstructed using

all of the 2180 still images that were extracted from the video footage. The ‘Camera Calibration’ profile was set to ‘Fisheye lens’ to help account for the refraction caused by the GoPro’s wide-angle lens, and the ‘Detect Markers’ tool was used to automatically create control points for each coded target found within the image drastically reducing much of the manual work needed; any of the coded targets not detected were marked manually.

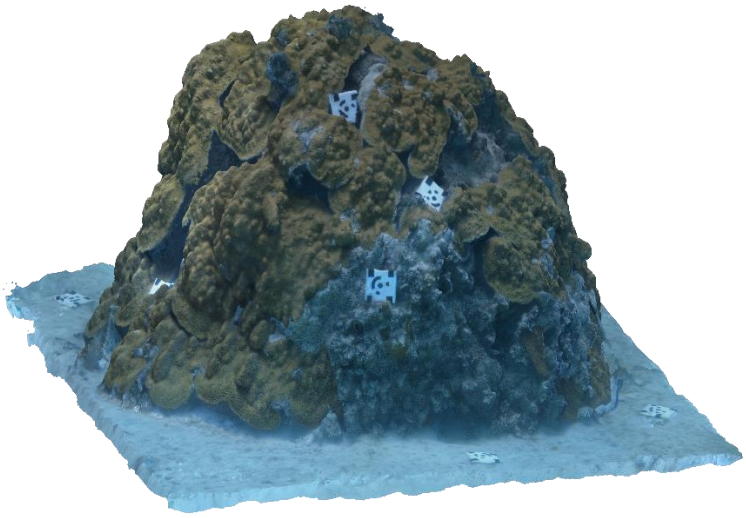

Fig. 11 – A textured mesh representing the example coral patch—which is roughly 1.5 m in diameter and 3 m in height—was reconstructed from still images extracted from video footage using Agisoft Metashape SfM photogrammetry software. The mesh consisted of 10 million faces, and had an estimated accumulative error of 1.4 mm after providing absolute scale using the real world dimensions of the coded targets.

The remainder of the reconstruction process followed the standard procedure of (1) photo alignment, (2) densification, (3) building a mesh and then (4) texturizing it. All quality settings were set to ‘Medium’ with the exception of photo alignment, which was set to ‘Highest’ resulting in 95% of images being aligned. The reconstructed model consisted of roughly 10 million triangular faces that approximated the surface of the patch reef. The model was estimated to have a ground resolution of 0.278 mm/pixel and a reprojection error (i.e., root-mean square error) equal to 1.6 pixels. Absolute scale was provided to the model in Metashape by creating scale bars along the length and width of seven coded targets found within the model, and supplying them with the corresponding real world dimensions (4 ¼ inches x 4 ¼ inches); the estimated accumulative error was reported to be approximately 1.4 mm.

A Deep Learning and Computer Vision Workflow

The still images used to reconstruct the 3-D model were the same ones used to train a deep learning semantic segmentation algorithm. However, before they could be used as training data they needed to be provided with the appropriate annotations. For semantic segmentation every pixel in the image needs to be provided with a label denoting the class category it belongs to (i.e., dense labels), which is a time-consuming and expensive process. Even when using commercial image annotation software, providing pixel-wise labels can cost the annotator 20+ minutes per image.

Thus, to reduce the burden of having to manually perform pixel-level annotations for thousands of images, this study designed a workflow that provided every still image in the dataset with dense labels while also minimizing the amount of work needed to be performed by the user. The workflow is summarized in Fig. 12, which first required the user to manually create a dataset that could be used to train a patch-based image classifier. This classifier provided numerous sparse labels to each still image automatically, and then using Fast-MSS, they were converted into dense labels. These dense labels and their corresponding images formed a dataset that were then used to train a Fully Convolutional Network (FCN) capable of performing pixel-level classifications on unannotated images.

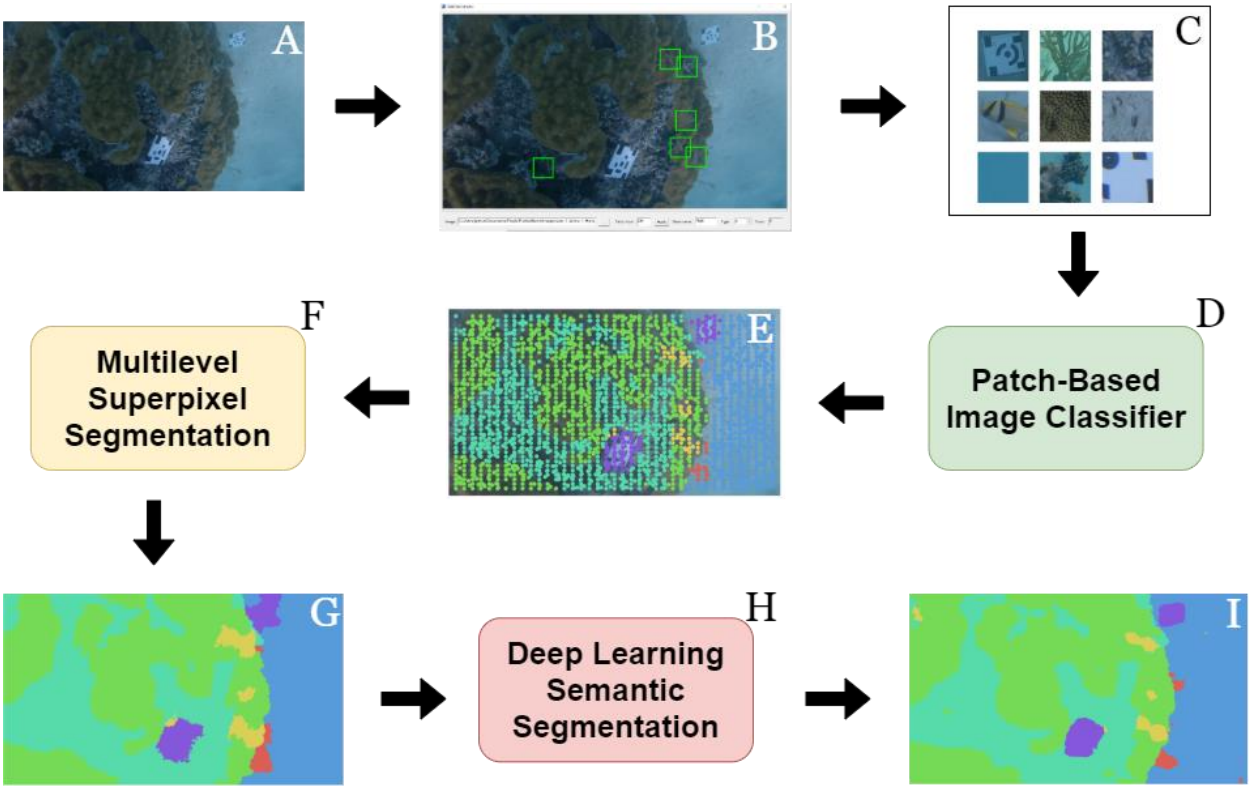

Fig. 12 – A diagram illustrating the workflow used to obtain dense labels for each image. Still images were extracted from the video footage (A) and imported into Rzhanov's patch-extraction tool (B) where patches for each class category of interest were extracted (C). These patches and their corresponding labels were used to train a patch-based image classifier (D) that then provided numerous sparse labels to each image in the dataset (E). Using Fast-MSS (F), the sparse labels were converted into dense (G) and used as the pixel-wise labels necessary for training a deep learning semantic segmentation algorithm. (H). With a trained FCN, novel images collected from the same or similar habitats could be provided with dense labels automatically (I) and without having to perform any of the previous steps (B-G).

Although there were multiple steps involved in this workflow, only the first step required manual effort from the user; the remainder of the steps were completed automatically using deep learning and computer vision algorithms. Thus, this workflow showcased that training data created through almost entirely automatic processes (as opposed to being done manually) could still produce a deep learning model that performs with acceptable classification scores to be used in other applications. To evaluate how well these deep learning models perform, 50 images were first randomly sampled *with replacement* from the dataset

and given an additional set of ground-truth dense labels that were created by hand using the commercial image annotation software *LabelBox* [32]. These ground-truth dense labels served as a testing set to gauge the performance of the patch-based image classifier, the dense labels created by Fast-MSS, and the predictions made by the FCNs.

1. Creating an Image-Patch Dataset

Beginning the workflow, the first step involved creating a dataset that a patch-based image classifier could learn from. Unlike a normal image classifier, a patch-based image classifier is trained on sub-images commonly referred to as 'patches' that are cropped on individual class categories. A common method for creating an image-patch dataset is outlined in [6], where patches are extracted centered on top of the existing sparse labels that were created manually by a user with a point-based annotation software tool like Coral Point Count (CPCe).

However, instead of going through the time-consuming process of creating CPCe annotations for each image, this study used the Center for Coastal and Ocean Mapping (Durham, NH, USA)'s in-house annotation software tool made by Dr. Yuri Rzhanov specifically for the purpose of extracting patches from still images. This patch-extraction tool is fast and provides an intuitive interface that allows the user to easily sample any part of the image, while archiving the location of extraction and assigned class label. Given the freedom to extract patches using a mouse or trackpad, a user can quickly create a highly representative dataset. Using this tool, roughly10,000 patches with dimensions of 112 pixels by 112 pixels were extracted from the still images in the dataset, averaging approximately 50 patch extractions per minute.

2. Training a Classifier to Provide Sparse Labels

This newly created dataset consisting of patches and their corresponding labels served as the training data for the patch-based image classifier; as in Chapter 2, the classifier was first trained and then used to provide sparse labels to each still image automatically.

Providing additional labels involved first uniformly extracting patches with dimensions of 112 pixels x 112 pixels from an image following a grid formation. In total, approximately 2800 patches were sampled from each image in the dataset, representing potentially 2800 additional labels per image, or roughly .035% of the total number of pixels in the image. Extracted patches were then passed to the classifier as input. The output for each was a corresponding vector representing the probability distribution of class categories to which the center-most pixel of the patch likely belonged. For each patch the extracted location, the presumed class label, and the difference between the two highest probability distributions (i.e., top-1 and top-2 choices) were recorded.

Again, the difference in probabilities were used to filter out sparse labels that were more likely to have been misclassified. Determining the ideal threshold involved trying different values and comparing the classification scores of the sparse labels predicted for the test images against the labels in the corresponding pixel indices of the ground-truth. As discussed in the results section, the final threshold value that was chosen balanced the tradeoff between the number of labels that were accepted and their classifications scores.

With regards to efficiency, the patch-based image classifier assigned roughly 200 sparse labels to an image per second, as opposed to the one annotation every six seconds that it cost users who used a point-based annotation software tool [17].

3. Converting Sparse Labels to Dense using Fast-MSS

The next step converted the accepted sparse labels that were assigned to each image into dense using Fast-MSS. For this dataset, the first and last number of superpixels to partition each image into was 5000 and 300, respectively, and across 30 iterations. Each image was downsized by reducing the height and width by a factor of six after confirming that a reduction in the input image's dimensions could decrease the time required to create the dense labels without negatively affecting the classification scores (Table 5). Dense labels were then upsized using nearest neighbor interpolation so they matched the image's original dimensions, a requirement for deep learning model training.

Table 5 – The effect of reducing an input image's dimensions on the output of Fast-MSS

Reduction Factor	Dimensions (pixels)	PA	m-PA	w-IoU	w-Dice	Time (Seconds)
1	2160 x 3840	0.8852	0.8051	0.8199	0.8938	260.45
2	1080 x 1920	0.8854	0.8049	0.8197	0.8937	64.42
3	720 x 1280	0.8856	0.8050	0.8195	0.8936	22.21
4	540 x 960	0.8853	0.8053	0.8196	0.8936	13.85
5	432 x 768	0.8853	0.8054	0.8195	0.8933	9.98
6	360 x 640	0.8850	0.8051	0.8192	0.8930	7.79

Abbreviations: PA, pixel accuracy; MPA, mean pixel accuracy; MIoU, mean intersection over union. Cells are colored red, yellow, or green if they are lower, the same, or higher compared to other dimensions, with 1.0 representing a perfect score for classification metrics.

Note: All trials were conducted on the same PC with an Intel i7-8700 processor; dense labels were resized using nearest neighbor interpolation before compared to ground-truth.

4. In the Future, Automate the Boring Stuff

Although the dense labels created could have been used to classify the 3-D reconstructed model directly, they were instead used as training data to train a deep learning semantic segmentation algorithm. The major advantage of a FCN is its ability to generalize to images collected from domains that are similar to those that it was trained on. A researcher could obtain dense labels for images collected from the same or similar habitats that the FCN was previously trained on without having to perform any of the previous steps in the workflow. Thus, the objective of this workflow was not just to obtain a set of dense labels for every still image, but rather it was a means of acquiring dense labels for datasets collected in the future more efficiently.

This study experimented with five FCNs, all of which used encoders from the EfficientNet series (B0 – B4) as those were shown to perform with the highest classification scores in the previous study. Each FCN was used to create an additional set of dense labels for every image in the dataset; these and the set created by Fast-MSS were validated and compared against the ground-truth dense labels within the testing set.

Class Categories

Of the different organisms, substrate types, and objects present in the video footage data, seven class categories were formed. Four of these were biological ("Branching", "Fish", "Massive Coral" and "Algae") and consisted of multiple species, one encompassed all of the potential substrate types ("Substrate"), another was used to denote the coded targets ("Target"), and lastly was the class to represent the background ("Water", Fig 13.) The first five class categories served as functional groups to demonstrate the ability to calculate community composition in both 2-D images and 3-D models, but alternative functional groups could be chosen for different purposes.

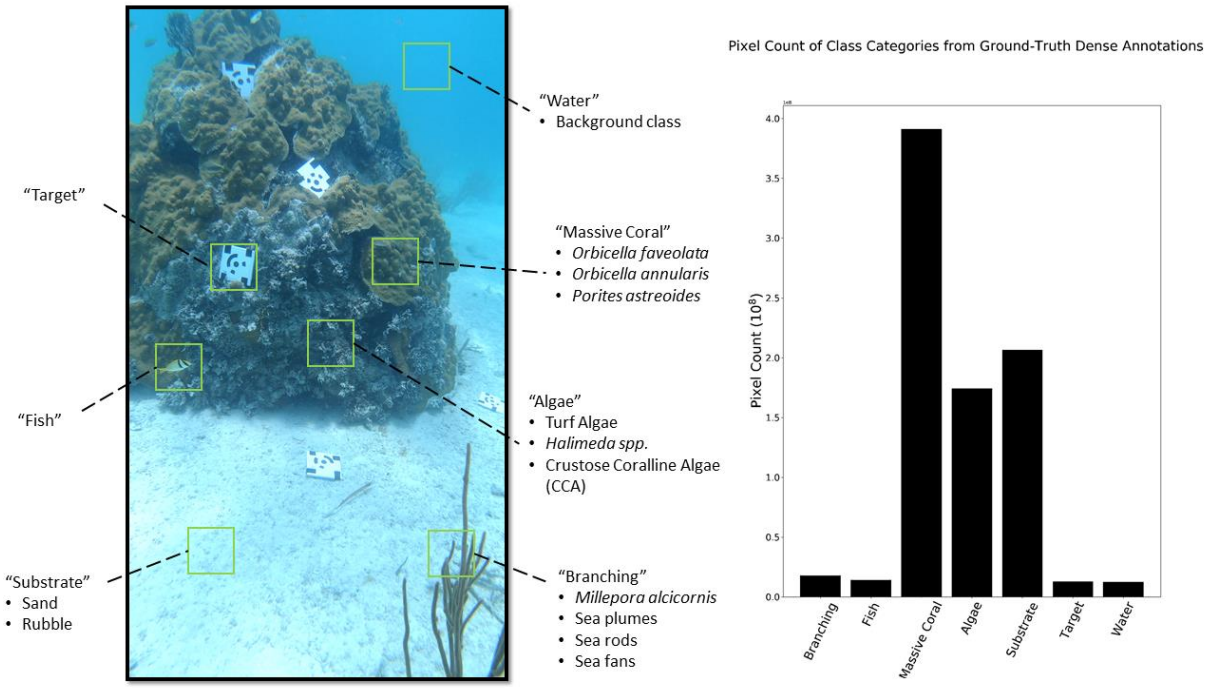

Fig. 13 – A still image (2160 pixels x 3840 pixels) extracted from the video footage showcasing the class categories used in this study (left), and the distribution of each class category based on their pixel count calculated from the 50 ground-truth dense labels within the testing set (right).

The majority of the still images in the dataset were made up of pixels that belonged to massive corals (*Oorbicella faveolata*, *Orbicella annularis* and *Porites astreoides*), followed by different types of substrate (sand, rubble). The third most represented class category was "Algae", which contained some crustose coralline algae (CCA) and filamentous turf algae, but primarily *Halimeda spp.*, which was found in abundance in crevices between coral colonies. The "Branching" class was comprised of fire coral (*Millepora alcicornis*) and various other types of octocorals that included sea plumes, sea rods, and sea fans; the "Fish" class category incorporated all individuals with no distinction made between genus or species. To ensure that the coded targets would not be assumed to be associated with one of the functional groups a class was created for it. Lastly L "Water" served as the background class meant to represent the pixels in an image where there was nothing as visible as a result of light attenuation through the water column.

These seven class categories could be found in the still images, but only “Branching”, “Massive Coral”, “Algae”, “Substrate” and “Target” were included in the 3-D model because SfM photogrammetry is only capable of reconstructing objects that are static within the source images. Thus “Fish” and “Water” would be excluded.

Model Training

The patch-based image classifier that was used to provide sparse labels to each image as described in the workflow used the EfficientNet-B0 architecture. Instead of using the typical ‘ImageNet’ weights, the classifier was initialized with the ‘Noisy-Student’ weights, which were learned using a semi-supervised training scheme that was demonstrated to outperform the former [33]. The encoder was followed by a max pooling operation, a dropout layer (80%), and finally a single fully connected layer with seven output nodes (one for each of the class categories). Patches were resized to 224 pixels x 224 pixels and fed to the model as training data after heavy augmentation techniques were applied using the *ImgAug* library, and normalized to have pixel values between 0 and 1.

Because the task was multi-categorical classification, softmax was chosen as the activation function for the network, and the batch size was set to 32 as this was the largest amount possible given the network architecture, the size of the image patches, and the amount of memory that could be allocated by the GPU being used. The model was trained on 10,000000 image patches that were randomly split into a training (90%) and validation (10%) set for 25 epochs.

During training the error between the actual and predicted output was calculated using the categorical-cross entropy loss function. Parameters throughout the network were adjusted using the Adam optimizer with an initial learning rate of 10^{-4}. Using the *ReduceLROnPlateau* callback, the learning rate was reduced by a factor of 0.5 for every three epochs in which the validation loss failed to decrease, and the weights from the epoch with the lowest validation loss were archived.

Based on the results from Chapter 2, this study experimented with five different FCNs, all of which used the U-Net architecture and were equipped with one of the five smallest encoders within the EfficientNet family (i.e., B0 through B4). Again, all deep learning semantic segmentation models were implemented in Python using the *Segmentation Models* library provided by [24].

Each of the EfficientNet encoders was initialized with ‘Noisy-Student’ weights, but was left frozen (i.e., immutable) for the entire training process, meaning only the weights within the decoder of the FCN were updated. Images were pre-processed in the same way as the images were when the original encoders were trained on the ImageNet dataset, while dense labels were converted into one-hot-encoded vectors forming a shape of (B x H x W x C) where B and C represent the batch size and the number of class categories, respectively. During preliminary analysis it was found that heavier augmentation techniques (e.g., adding noise, blurring, sharpening, altering contrast) resulted in lower classification accuracies; instead only augmentations in the form of simple affine transformations (flips, flops, rotations) were applied to each sample.

Each successive encoder within the EfficientNet family required an additional amount of memory to train due to their increasing architectural size and number of parameters. To accommodate the memory requirements of each of the encoders, all images were reduced in height and width by a factor of three resulting in dimensions of 736 pixels x 1280 pixels; this was the largest an image could be to work with all of the encoders, and consequently resulted in the batch size having to be equal to one (i.e., a single image).

Models were trained for 25 epochs on all 2180 images, which were randomly split into a training (90%) and validation (10%) set.

During training of the FCNs the error was calculated using the soft-Jaccard loss function, which acts as a differentiable proxy that attempts to maximize the Intersection-over-Union metric [25]. Parameters were updated via backpropagation using the Adam optimizer, which was set with an initial learning rate of 10^{-4} and used the *ReduceLROnPlateau* callback with the same settings as described before; after 20 epochs, only the weights from the epoch with the lowest validation loss were archived.

All deep learning models were trained on a PC equipped with an NVIDIA GTX 1080 Ti GPU and an Intel i7-8700 CPU, using the Keras deep learning framework and the Tensorflow numerical computational library.

3-D Model Classification

Following the training process, Fast-MSS and the five FCNs were each used to create a set of dense labels for each image in the dataset. With each respective set of dense of labels, a separate classified 3-D model was created, thus allowing the comparison between the five FCN encoders (i.e., EfficientNet B0 – B4) and Fast-MSS. The technique to assign semantic labels to the 3-D model was straight forward and was done almost entirely in Agisoft Metashape; the instructions for how this was done are explained below (Fig. 14).

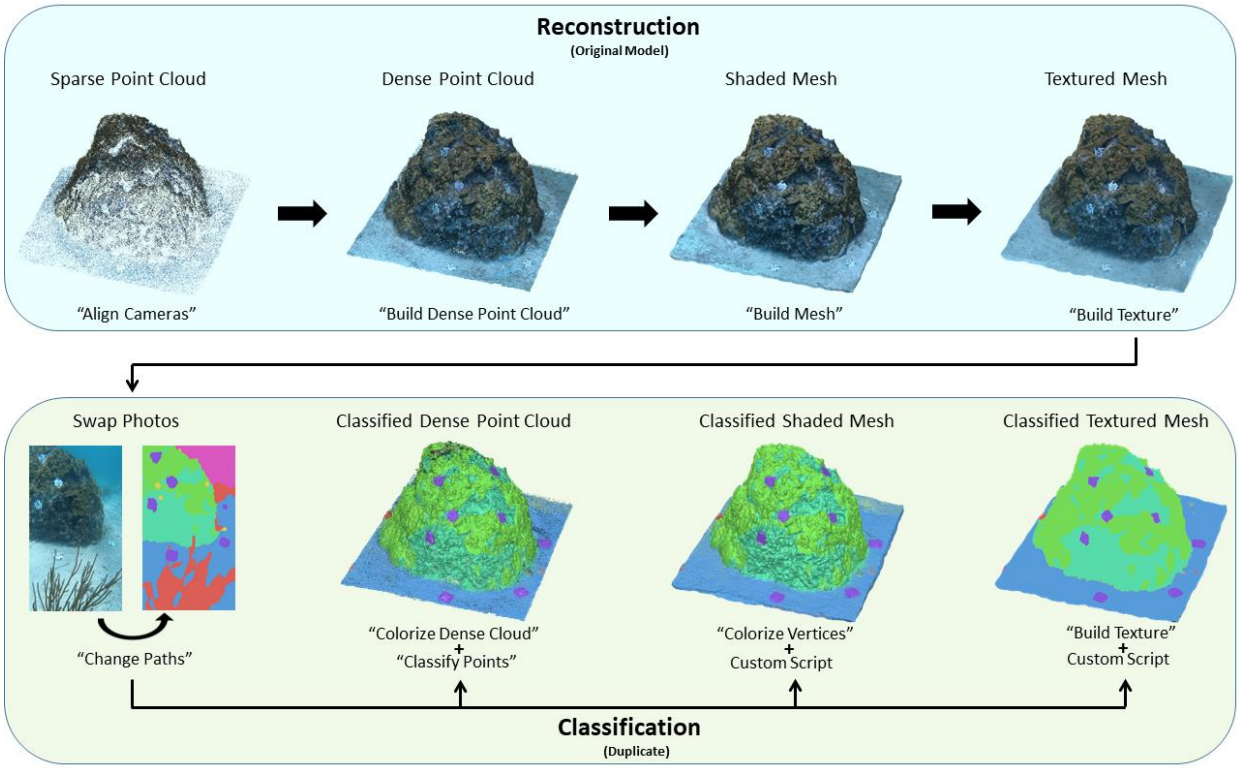

Fig. 14 – A diagram showing step-by-step which tools in Metashape were used to reconstruct the 3-D model, followed by how semantic labels were provided to it. Once the images are swapped with their corresponding dense labels, the classified point cloud, shaded mesh, and textured mesh can be created independently of one another and is not a sequential process like the reconstruction.

Once the textured mesh for the 3-D model was created, the entire project was duplicated using the 'Duplicate Chunk' action. Within the duplicated project, all information that was created during the construction of the original model (e.g., the mapping of pixels from image to 3-D space, color components and UV coordinates) was also copied to the project folder. Following the duplication, images that were used in the reconstruction process were swapped with their corresponding dense labels using the 'Change Paths' tool; because the dimensions of the images and the dense labels were identical, and they both shared the same filenames, the swap executed without error.

Next the 'Build Texture' tool was used to create another textured mesh but using the dense labels as the source images instead. By default, this tool reused the existing UV coordinates that were copied over during the duplication. The 'Texture Type' was set to diffuse, and 'Blending' was disabled to ensure that the discrete categorical values representing each class in the dense labels would not accidently be averaged along the borders of neighboring semantic groups in the resulting classified textured mesh (e.g., seamlines). Alternatively, it was found that if the blending mode was set to 'average' or 'mosaic'—as is recommended by Agisoft—the model could be corrected using a custom post-processing script, which is explained in a later passage.

Once completed, the classified textured mesh was identical to the original in appearance, but with textures that were mapped from the respective set of dense labels that were used as source images instead of the still images (Fig. 15). A classified shaded mesh and dense point cloud were then created using the 'Colorize Vertices' and 'Colorize Dense Cloud' tools, respectively. These tools worked similarly, mapping the color components from the pixel indices found in the source images (i.e., dense labels) to their corresponding elements or points within the shaded mesh or dense cloud.

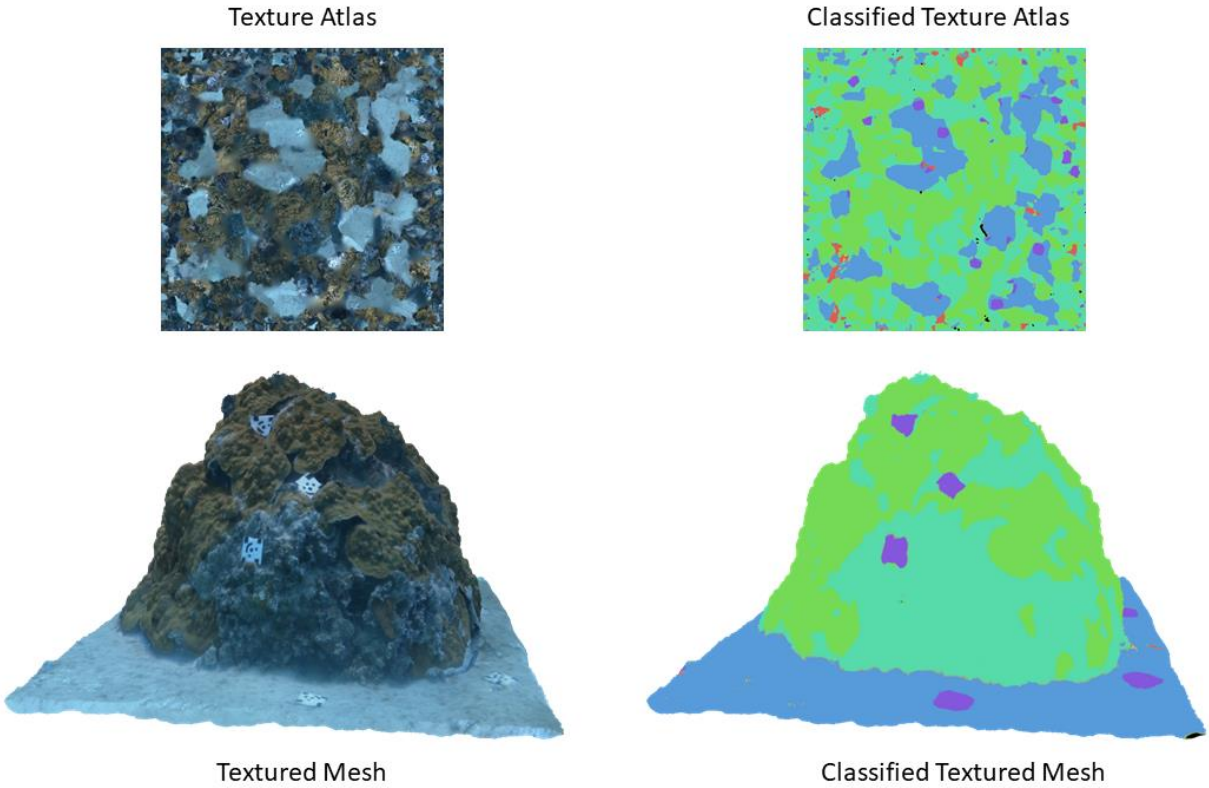

Fig. 15 – Comparison between the textured mesh and its corresponding atlas (left) against the classified version of the mesh and its corresponding atlas after being corrected using the custom post-processing script (right). Validation for the classified model was obtained by comparing the classified atlas with a manually annotated texture atlas (not shown) that served as ground-truth.

However, unlike in the 'Build Texture' tool, the blending mode could not be disabled in either 'Colorize Vertices' or 'Colorize Dense Cloud'. This resulted in some of the elements or points having color components with values that are not within the set of discrete values that denoted the class categories, which could potentially pose a problem for those attempting to perform spatial analysis using the classified models in future studies.

Fortunately, Metashape provided a tool called 'Classify Points', which selects and then classifies points based on an individual, or range of color components. After this was done for each class category, the dense cloud and its corrected classifications were exported as conventional point cloud formats including LAS and XYZ to confirm that they could be used in other spatial analysis software.

However, the current version of Metashape does not offer the ability to classify the vertices of a mesh based on color components; instead, this study used a custom script written in Python that performed this task outside of Metashape, demonstrating that it could be done if needed. After the mesh was colorized using 'Colorize Vertices', it was exported as an OBJ file in ASCII format that stored the 3-dimensional coordinates of each vertex and its color components in an easily parsable format. When provided with the set of discrete color components that denote each of the class categories, the script was made to first check if each vertex had one of the correct color components; if the values were not within the set, they were changed to the color components to which they are closest in RGB color space as measured by their Euclidean distance. Because the script parsed the file line-by-line, even large models could be corrected this way without having memory allocation errors. This same script could also be used to adjust each of the pixel indices in the classified textured atlas if the blending mode of the 'Build Texture' tool had been set to either 'mosaic' or 'average' instead of being disabled.

Although the classifications were provided to the shaded mesh and dense point cloud within Agisoft Metashape, there was no tool that could be used to evaluate their accuracies. Instead this was done outside of Agisoft Metashape, and by using the classification scores of the classified textured mesh as a proxy for the scores of the classified shaded mesh and dense point cloud. From Agisoft Metashape both the original and the classified textured mesh were exported as 2-D images (i.e., texture atlases), and then the former was made into a 'ground-truth texture atlas' by manually providing it with semantic labels using the image annotation software tool *LabelBox*. As is done when annotating a typical 2-D image, the pixel indices in the original texture atlas were assigned labels denoting the class category they were thought to belong to by a trained annotator.

Unfortunately, not all of the textures were discernable to the annotator as some were either too small, or simply did not resemble any of the class categories when represented in the texture atlas. In an attempt to provide an accurate form of ground-truth, annotators only assigned labels to the pixel indices that they were confidently able to identify, resulting in a ground-truth texture atlas (4096 pixels by 4096 pixels) where 88% of the pixels were provided with semantic labels.

Experiments

For the analysis, this study validated the results of the patch-based image classifier and its ability to produce sparse labels, the dense labels that were created by Fast-MSS, the predictions made by the five FCNs experimented with, and the classification accuracy of the 3-D classified models.

To calculate classification scores, the sparse labels predicted by the patch-based image classifier for each image in the test set were compared to the labels in the corresponding pixel indices of the ground-truth. Similarly, the dense labels created by Fast-MSS and the FCNs for each image in the test set were compared to the ground-truth dense labels. Lastly, each classified 3-D model was evaluated following the process described in the previous section, where each 3-D model was exported from Agisoft Metashape as a 2-D image and its semantic labels were compared to the ground-truth labels that were provided by the annotator.

As seen in the bar-chart of Fig. 10, the distribution between class categories was not uniform, which likely caused predictive models to learn features that favor over-represented classes, at the expense of under-represented classes. However, because this study did not value one class over any other, two of the metrics used to evaluate the classification scores were calculated as a weighted average based on the frequency (i.e., total number of pixels) of each class.

The metrics used include pixel accuracy (PA), mean pixel accuracy (m-PA), weighted Intersection-over-Union (w-IoU), and weighted Dice coefficient (w-Dice). PA was computed by globally calculating the ratio of correctly classified pixels to the total number of pixels; this is identical to the overall classification accuracy and does not take into consideration class imbalances. The m-PA calculates the global accuracy of each class individually and then averages them together so that each class contributes to the final score equally, regardless of class imbalances. Last are IoU and Dice (i.e., Jaccard index and F1-Score, respectively), which are similarity coefficients commonly used for quantifying classification scores of semantic segmentation tasks. The weighted average for these two metrics were calculated using (10) and (12), respectively:

$$w\text{-}IoU = \frac{\sum_{i=1}^{N} IoU_i * w_i}{\sum_{i=1}^{N} w_i} \tag{10}$$

$$Dice = \frac{2 * TP}{TP + FP + TN + FN} \tag{11}$$

$$w\text{-}Dice = \frac{\sum_{i=1}^{N} Dice_i * w_i}{\sum_{i=1}^{N} w_i} \tag{12}$$

where the weight for each class w_i, was calculated as ratio of pixels per class over the total number of pixels in the test set.

RESULTS

Classification Scores

First are the results of the patch-based image classifier and how its performance changed as a function of the confidence threshold value used. Mentioned in the methods section, the confidence score was used to filter sparse labels that were more likely to have been misclassified; a higher threshold value usually represents more confidence in a prediction.

Table 6 – Classification scores for the patch-based image classifier compared against ground-truth.

Threshold	Accepted	PA	m-PA	w-IoU	w-Dice
0.0	100%	0.833	0.786	0.739	0.844
0.25	94%	0.855	0.815	0.769	0.864
0.50	89%	0.875	0.835	0.796	0.882
0.75	83%	0.896	0.857	0.827	0.902
0.90	76%	0.914	0.874	0.855	0.919
0.99	61%	0.941	0.902	0.899	0.944

Abbreviations: PA, pixel accuracy; m-PA, mean pixel accuracy; w-IoU, weighted Intersection-over-Union; w-Dice, weighted Dice coefficient.

Note: For classification metrics, 1.0 represents a perfect score.

As expected, Table 6 shows that there is an inverse relationship between the amount of points accepted and the overall classification scores, which can readily be seen in Fig. 16. Based on these results, 0.50 was chosen as the confidence threshold value for the remainder of the workflow as it was deemed to produce results that balanced this tradeoff.

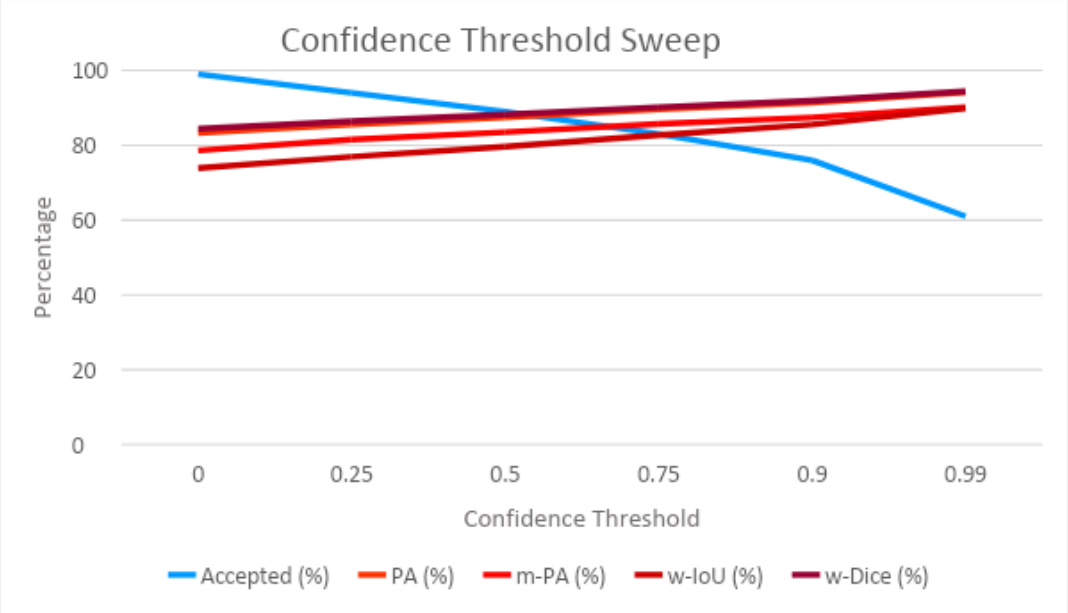

Fig. 16 – A line-graph displaying the inverse relationship between the confidence threshold value and the classification scores of the sparse labels accepted. As the threshold value becomes more conservative along the x-axis, more of the sparse labels the classifier is unsure about are rejected causing the classification scores of the remaining labels to increase as a result.

Table 7 shows that the dense labels that were produced by Fast-MSS produced classification scores that were slightly less than those created by any of the FCNs, except for B2, which produced the lowest scores; among the FCNs, the differences in performance were marginal. With regards to speed, all FCNs performed substantially faster than Fast-MSS, whose recorded time also included the time required by the patch-based image classifier to first predict sparse labels for the input image. However, even when the input image was reduced in dimensions by a factor of 6, the patch-based image classifier and Fast-MSS combo produced a result in 22.6 seconds, which is still 10x slower than the slowest FCN.

Table 7 – Classification scores of each method for producing dense labels compared against ground-truth.

Method	PA	m-PA	w-IoU	w-Dice		Time (seconds)
Fast-MSS	0.885	0.805	0.819	0.893		37.06
EfficientNet-B0	0.895	0.809	0.826	0.899		0.99
EfficientNet-B1	**0.900**	0.811	**0.833**	**0.903**		1.34
EfficientNet-B2	0.870	0.797	0.793	0.878		1.78
EfficientNet-B3	0.897	**0.817**	0.830	0.901		2.14
EfficientNet-B4	0.897	0.811	0.830	0.901		2.31

Abbreviations: PA, pixel accuracy; m-PA, mean pixel accuracy; w-IoU, weighted Intersection-over-Union; w-Dice, weighted Dice coefficient.

Note: Times to perform are based on input images with dimensions of 736 pixels by 1280 pixels. Scores are colored red, yellow, or green to represent the worst, the intermediate, and the best methods, respectively, for classification scores and speed. Bold numbers highlight the best performing method for each metric, with 1.0 representing a perfect score.

Last are the results for the classified 3-D model (Table 8). Overall the classification scores followed the same general trend that can be seen in Table 7. The classified texture atlas that used the dense labels produced by Fast-MSS as the source images had scores for PA, w-IoU and w-Dice that were slightly less than those created by any of the FCNs; the FCNs were equally good with no clear indication that one outperformed another.

Table 8 – Classification scores of 3-D models represented as texture atlases compared against ground-truth.

Method	PA	m-PA	w-IoU	w-Dice
Fast-MSS	0.896	0.775	0.823	0.899
EfficientNet-B0	0.905	0.762	0.836	0.907
EfficientNet-B1	0.910	0.766	0.843	0.911
EfficientNet-B2	0.908	0.781	0.842	0.910
EfficientNet-B3	0.907	0.781	0.840	0.910
EfficientNet-B4	**0.913**	0.775	**0.850**	**0.915**

Abbreviations: PA, pixel accuracy; m-PA, mean pixel accuracy; w-IoU, weighted Intersection-over-Union; w-Dice, weighted Dice coefficient.

Note: Due to the inability to discern the class category of all the pixels in the ground-truth texture atlas, only those that could be provided with labels (~88%) were used in the comparison. Scores are colored red, yellow, or green if they are lower, the same, or higher than the other method's score, respectively. Bold numbers highlight the best performing method for each metric with 1.0 representing a perfect score.

Discussion

Table 6 shows the inverse relationship between the confidence threshold value chosen and the percentage of sparse labels accepted: as the threshold value became more conservative (i.e., increases) more of the labels that the model was not confident about were rejected. This also created a direct relationship between the threshold value and the classification scores, because again, as more of the labels the model was not confident about were rejected, the overall classification accuracy of the remaining labels was likely to increase as a result.

In Chapter 1 it was shown that Fast-MSS produced dense labels with higher classification scores when it was provided a greater number of sparse labels. While this remains true, Table 6 shows that supplying more sparse labels is not necessarily beneficial. By decreasing the confidence threshold, there is an increase in the number of misclassified labels, whose error would only be compounded when Fast-MSS propagated the class label to the adjacent pixel indices when creating dense labels. Thus, when using a patch-based image classifier in conjunction with Fast-MSS there exists a balance between the number of sparse labels that should be accepted and the resulting classification scores; this can also be used as an indicator to determine whether or not the classifier requires further training. Table 6 also highlights why developing a comprehensive and well-representative dataset from the very beginning is important, as the classifier's performance has a significant effect on the classification scores of the remaining portions of the workflow (as seen in Table 7 and 8).

A key takeaway from Table 7 is that even though the FCNs were trained on the dense labels produced by Fast-MSS, all but B2 achieved higher classification scores. This suggests that as a deep learning algorithm, a FCN has the potential to develop a better understanding of which features are associated with each class category by learning from all of the images collectively throughout the training process. This is in contrast with Fast-MSS, which, although it performed well, is limited by the fact that it can only propagate the label that it is provided with outwards to neighboring pixel and does not contain a mechanism for learning

whether or not those labels are in fact correct. This is not to say that the patch-based image classifier and Fast-MSS do not serve a valuable role within the context of the workflow, but rather that it would be preferable to use a trained FCN as the primary method for producing dense labels for novel images collected in the future. This, combined with the fact that the FCNs performed substantially faster highlight why researchers should be moving towards the use of deep learning algorithms for the annotation of coral reef imagery.

Table 8 shows the classification scores for the 3-D classified models that were made from the different sets of dense labels shown in Table 7. Although the difference in scores between each 3-D model is not substantial, the fact that they closely resemble the scores in Table 7 suggests three things at a minimum. The first is that Agisoft Metashape is able to map the textures from the dense labels to create a 3-D classified model with a high level of accuracy. Secondly, the classification scores of the 3-D models appear to be largely dependent on the classification scores of the dense labels that were used as source images; this reinforces what was already assumed to be true and also provides positive validation for this technique of creating 3-D classified models. Finally, the results suggest that the non-conventional ground-truth texture atlas that was created is of similar quality when compared to the more conventional ground-truth dense labels that were created for the images in the test set. This provides validation for this method of evaluating the classification scores of the 3-D model directly, which could prove useful in future studies.

Although the scores between Table 7 and 8 are similar, there is a pattern of a 1 to 2-point increase for PA, w-IoU and w-Dice, which may be caused by the blending of color components that occurs during the 'Build Texture' function. For each individual element that comprises the 3-D mesh, there are multiple pixels found within different source images that all correspond to it, but from different vantage points. When creating the textured mesh with the blending mode set to either 'mosaic' or 'average', each element is assigned a color based on the weighted average of the color components from the pixels that it corresponds to [34]. Thus, by using either of these modes, the blending of source images—in this case, the dense labels—may serve as a weighted average ensemble that contributes to slightly higher classification scores. However, Table 8 shows that m-PA drops by approximately 3 to 4-points for each method, but this may be explainable by the following.

SfM algorithms make the assumption that all parts of the scene are static meaning anything dynamic will not accurately be incorporated in the reconstruction. For this reason, the two class categories 'Fish' and 'Water' that were defined for this study cannot be represented in the 3-D model nor the ground-truth texture atlas. However, the 'Build Texture' tool will map the semantic labels from the source images to the 3-D model regardless of which class category they belong. Because each method still has the potential to misclassify some pixels in the source image (as seen by their lack of perfect scores in Table 7), their misclassifications *can* make their way into the classified texture atlas.

Secondly, by comparing the per-class accuracy between the two top scoring models from tables 7 and 8, it can be seen that the scores decrease for all of the classes in the classified texture atlas except for 'Massive', the most represented category, whereas 'Branching' and 'Target' drop in score considerably, which are the two smallest and least represented class categories that can be found in the 3-D model (Table 9).

Table 9 – Comparing the Relative Abundance and Per-Class Accuracy of the Still Images against the Texture Atlas.

	Relative Abundance		**Per-Class Accuracy**	
Class Categories	Still Images	Texture Atlas	Still Images	Texture Atlas
Branching	0.02	0.01	0.713	0.406
Fish	0.01	NA	0.773	NA
Massive Coral	0.45	0.50	0.865	0.939
Algae	0.23	0.26	0.965	0.911
Substrate	0.26	0.22	0.911	0.857
Target	0.02	0.01	0.938	0.698
Water	0.01	NA	0.837	NA

Note: the 'Fish' and 'Water' class categories are not included in the texture atlas due to an inability to reconstruct dynamic objects within a photogrammetric model. Classification scores for still images and the texture atlas came from models with B1 and B4 as encoders, respectively.

These two reasons suggest that the decrease in m-PA is not necessarily reflective of an issue inherent in the 'Build Texture' tool but rather the difficulties in manually providing semantic labels to the ground-truth texture atlas—especially for classes that are both relatively small in size and less frequent—as well as how the metric is calculated, which does not take into account the imbalances in class distribution and weighs each per-class accuracy equally.

In conclusion, these results provide evidence that the 'Build Texture' tool is a method to accurately assign semantic labels from source images to a 3-D model, and that resulting classification accuracy of the classified texture model is a function of the reconstruction error of the original model, as well as the classification scores of the method used to produce dense labels. Although the classified textured mesh is not typically used in spatial analyses, this study showed that it can serve as a useful proxy for validating the accuracy of the classified shaded mesh and dense point cloud, which often are. Because the elements that make up the textured mesh store both the texture coordinates and the color components, it stands to reason that all three model types share similar classification scores (Fig. 17).

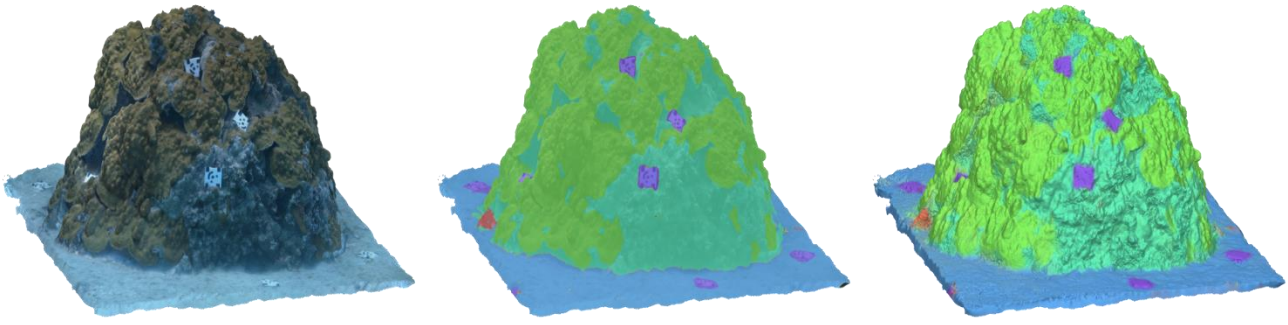

Fig. 17 – A side-by-side comparison between the textured mesh (left), the classified textured mesh with 40% transparency (center), and the classified shaded mesh (right). The classified textured mesh was used as a method for validating the classification results of the classified shaded mesh and dense point cloud (not shown), which can be used in spatial analyses.

GENERAL CONCLUSION

The underlying theme of this thesis was to investigate techniques that can easily be adopted by ecologists to assist them in their ability to monitor the changes that occur in benthic habitats. Point-based annotations created by software tools like Coral Point Count (CPCe) are already ubiquitous within this scientific community as a method for calculating coverage statistics, and help to assess a reef's general health both spatially and temporally. Unfortunately, the task of manually providing annotations to each image collected during a benthic habitat quadrat survey is tedious, time-consuming and prohibitive with regards to cost and project scale. With computer vision and deep learning algorithms, this thesis demonstrated how an existing set of sparse labels for an image could be converted into pixel-wise labels, allowing for the calculation of more robust coverage statistics. By adding improvements to the multilevel superpixel segmentation (MSS) algorithm, the first chapter of this thesis demonstrated through a comparison using the CamVid semantic segmentation benchmark dataset that the enhanced implementation (i.e., Fast-MSS) performs significantly faster and with classification scores that exceed those created by the original.

Chapter 2 further validated the results of Fast-MSS by using it as a method to create dense labels for each image in the Moorea Labeled Coral (MLC) dataset, a rigorous benchmark for testing computer vision algorithms in coral reef image recognition. Following the same experimental setup first outlined in [6], this study trained multiple Fully Convolutional Networks (FCNs) and used them to set the baseline scores for the task of semantic segmentation. Furthermore, classification scores were shown to increase when additional sparse labels were provided to each image using a patch-based image classifier. These results demonstrate the effectiveness of the technique and illustrate how ecologists may be able to augment their existing datasets through entirely automated processes.

Finally, the methods demonstrated in the first two chapters of this thesis were coupled with Structure-from-Motion (SfM) photogrammetry to demonstrate how the same techniques applied to 2-D images could also be applied to 3-D photogrammetric models. The same images that were extracted from video footage and used to create a 3-D model were provided with dense labels following a workflow designed to minimize the amount of manual work required by the user. In Agisoft's Metashape, the source images used in the reconstruction were swapped with their corresponding dense labels and used to classify the model, which was post-processed using a custom script written in Python. Classification scores were validated by comparing the 2-D texture atlas of the classified 3-D model against one that was provided with annotations manually using an annotation software tool. Overall the results indicate that this method can be used to classify 3-D models and would be suitable for many ecological applications including calculating coverage statistics for a reef as a 3-D system, which could provide a more accurate assessment of coral reef cover as opposed to just using 2-D images.

In the future there are plans to incorporate Fast-MSS into an annotation software tool equipped with a graphical user interface (GUI) making it accessible to all users regardless of their proficiency in Python or command-line interfaces, and to disseminate it freely for public use. As demonstrated in the comparison using the CamVid dataset, this algorithm is not specific to images of coral reefs and instead can be applied to produce dense labels for images from any domain. The same is also true for the method for classifying 3-D reconstructed models.

In conclusion, this thesis represents a step in the direction towards fully automated assessments and monitoring systems for coral reefs, and it is hoped that the techniques outlined here can provide at least some assistance in understanding how the changes that are occurring are affecting the habitat.

LIST OF REFERENCES

[1] B. Groombridge, "Coral Reefs," Global Biodiversity, pp. 307–323, 1992.

[2] "UNEP Report – 'Reefs at risk revisited,'" Management of Environmental Quality: An International Journal, vol. 22, no. 4, 2011.

[3] H. Cesar, L. Burke, and L. Pet-Soede, "The Economics of Worldwide Coral Reef Degradation," International Coral Reef Action Network, Jan. 2003.

[4] P. L. Jokiel, K. S. Rodgers, E. K. Brown, J. C. Kenyon, G. Aeby, W. R. Smith, and F. Farrell, "Comparison of methods used to estimate coral cover in the Hawaiian Islands," PeerJ, vol. 3, Dec. 2015.

[5] K. E. Kohler and S. M. Gill, "Coral Point Count with Excel extensions (CPCe): A Visual Basic program for the determination of coral and substrate coverage using random point count methodology," Computers & Geosciences, vol. 32, no. 9, pp. 1259–1269, 2006.

[6] O. Beijbom, P. J. Edmunds, D. I. Kline, B. G. Mitchell, and D. Kriegman, "Automated annotation of coral reef survey images," 2012 IEEE Conference on Computer Vision and Pattern Recognition, 2012.

[7] M. Mahmood, M. Bennamoun, S. An, F. Sohel, F. Boussaid, R. Hovey, G. Kendrick, and R. B. Fisher, "Coral classification with hybrid feature representations," 2016 IEEE International Conference on Image Processing (ICIP), 2016.

[8] M. Modasshir, A. Q. Li, and I. Rekleitis, "MDNet: Multi-Patch Dense Network for Coral Classification," OCEANS 2018 MTS/IEEE Charleston, 2018.

[9] Alonso, M. Yuval, G. Eyal, T. Treibitz, and A. C. Murillo, "CoralSeg: Learning coral segmentation from sparse annotations," Journal of Field Robotics, vol. 36, no. 8, pp. 1456–1477, Oct. 2019.

[10] King, S. M. Bhandarkar, and B. M. Hopkinson, "A Comparison of Deep Learning Methods for Semantic Segmentation of Coral Reef Survey Images," 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2018.

[11] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, "SLIC Superpixels Compared to State-of-the-Art Superpixel Methods," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 34, no. 11, pp. 2274–2282, 2012.

[12] Alonso, A. Cambra, A. Munoz, T. Treibitz, and A. C. Murillo, "Coral-Segmentation: Training Dense Labeling Models with Sparse Ground Truth," 2017 IEEE International Conference on Computer Vision Workshops (ICCVW), 2017.

[13] Mac Queen, J. Some Methods for Classification and Analysis of Multivariate Observations; University of California: Berkeley, CA, USA, 1967; Volume 1, pp. 281–297.

[14] Alchan, Fast-SLIC, (2019), GitHub repository, https://github.com/Algy/fast-slic

[15] V. Badrinarayanan, A. Kendall, and R. Cipolla, “SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 12, pp. 2481–2495, Jan. 2017.

[16] Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” CoRR, vol. abs/1409.1556, 2014.

[17] O. Beijbom, P. J. Edmunds, C. Roelfsema, J. Smith, D. I. Kline, B. P. Neal, M. J. Dunlap, V. Moriarty, T.-Y. Fan, C.-J. Tan, S. Chan, T. Treibitz, A. Gamst, B. G. Mitchell, and D. Kriegman, “Towards Automated Annotation of Benthic Survey Images: Variability of Human Experts and Operational Modes of Automation,” Plos One, vol. 10, no. 7, Aug. 2015.

[18] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning Transferable Architectures for Scalable Image Recognition,” 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018.

[19] Jung, ImgAug, (2019), GitHub Repository, https://github.com/aleju/imgaug

[20] Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.

[21] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” Lecture Notes in Computer Science Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015, pp. 234–241, 2015.

[22] G. Huang, Z. Liu, L. V. D. Maaten, and K. Q. Weinberger, “Densely Connected Convolutional Networks,” 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[23] Mingxing Tan and Quoc V Le. EfficientNet: Rethinking model scaling for convolutional neural networks. International Conference on Machine Learning, 2019.

[24] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the Inception Architecture for Computer Vision.” 2015.

[25] He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[26] P. Yakubovskiy, Segmentation Models, (2019), GitHub Repository, https://github.com/qubvel/segmentation_models

[27] Berman, A. R. Triki, and M. B. Blaschko, “The Lovasz-Softmax Loss: A Tractable Surrogate for the Optimization of the Intersection-Over-Union Measure in Neural Networks,” 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018.

[28] J. Burns, D. Delparte, L. Kapono, M. Belt, R. Gates, and M. Takabayashi, “Assessing the impact of acute disturbances on the structure and composition of a coral community using innovative 3D reconstruction techniques,” Methods in Oceanography, vol. 15-16, pp. 49–59, 2016.

[29] R. Harborne, P. J. Mumby, and R. Ferrari, “The effectiveness of different meso-scale rugosity metrics for predicting intra-habitat variation in coral-reef fish assemblages,” Environmental Biology of Fishes, vol. 94, no. 2, pp. 431–442, 2011.

[30] G. C. Young, S. Dey, A. D. Rogers, and D. Exton, “Correction: Cost and time-effective method for multi-scale measures of rugosity, fractal dimension, and vector dispersion from coral reef 3D models,” Nos 15, no. 3, 2020.

[31] B. M. Hopkinson, A. C. King, D. P. Owen, M. Johnson-Roberson, M. H. Long, and S. M. Bhandarkar, “Automated classification of three-dimensional reconstructions of coral reefs using convolutional neural networks,” Plos One, vol. 15, no. 3, 2020.

[32] Labelbox, "Labelbox," Online, 2020. [Online]. Available: https://labelbox.com

[33] Q. Xie, E. H. Hovy, M.-T. Luong, and Q. V. Le, “Self-training with Noisy Student improves ImageNet classification.,” CoRR, vol. abs/1911.04252, 2019.

[34] AgiSoft MetaShape Professional (Version 1.6) (Software). (2020*). Retrieved from http://www.agisoft.com/downloads/installer/

APPENDIX

Structure-from-Motion Photogrammetry (SfM)

As Structure-from-Motion photogrammetry (SfM) is used to reconstruct the 3-D model and also plays a crucial role in how semantic labels are assigned, this section serves as an overview of the reconstruction process and is meant to provide context to some of the more important details. SfM uses the fundamental principal of motion parallax to obtain some estimation of depth of an object or a scene captured from multiple overlapping images. By measuring the angle from the multiple viewpoints to the object while also estimating the distance between each viewpoint, the distance to the object can be calculated using basic trigonometry. Although not all SfM algorithms are identical, many use the same general principles that are described below.

A. Feature Detection

The first step in a typical SfM algorithm is feature extraction, which is used to detect specific parts within the object that can also be found in some of the other images. Key points represent local neighborhoods of pixel groupings in areas of an image with large changes in intensity in all directions (e.g., corners), and ideally are distinct and can be located within other images regardless of changes in scale, rotation and brightness. Once detected, information about those key points including a unique identifier, and their location in image space are stored in a file that is associated with the image that they were found in. Finally, an algorithm is used to match each of those key points with their corresponding points that were also found within other images.

B. Camera Alignment

The next step uses the key points to estimate the location of the camera at the time each image was taken. This process is sequential and starts by finding the two images that contain the most co-registered key points. Given the X, Y locations in image space of each key point and by assuming that all viewing rays to the optical sensor of the camera were straight and intersected at the time the image was taken, the Z-location for each key point can be estimated using trigonometry; consequently, this also provides an approximate location of the camera at the time the second image was taken relative to the first. This process is repeated for each additional image, estimating the location of the camera for subsequent images relative to those preceding it. However, due to refraction, and imprecise key point localization and camera calibration techniques, an error accumulates for each additional camera; camera locations are refined with a bundle adjustment algorithm, which uses projection matrices to simultaneously optimize camera and 3-D point locations.

C. Sparse Point Cloud

Key points are projected into 3-D space to form a sparse point cloud, which primarily serves as an indication of how well cameras were aligned (Fig. 18). Further refinements can be made to the point cloud by removing any points that are considered to be outliers as determined by their reconstruction uncertainty, re-projection error, and projection accuracy.

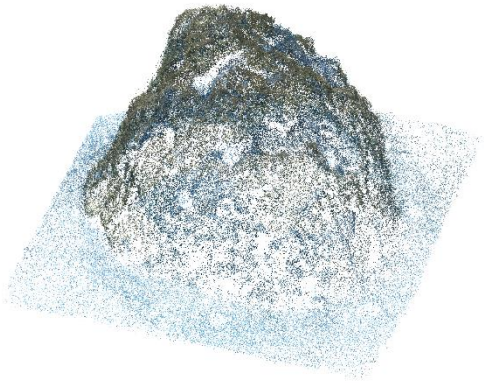

Fig. 18 – The sparse point cloud generated from the 2180 aligned images. Each key point represents a corresponding location found within two or more images (i.e., co-registered), which is then projected into 3-D space. The sparse point cloud serves as an indicator as to how well the images were aligned, but it is not used directly to create the dense point cloud in the following step.

D. Dense Point Cloud

This point cloud is then densified by creating depth maps for every pair of images, which determines the location each pixel should be in 3-dimensional space (Fig. 19). Each point in this dense cloud is assigned with an X, Y and Z location, as well the color components (i.e., RGB values) averaged from the pixels in the images that it originated from.

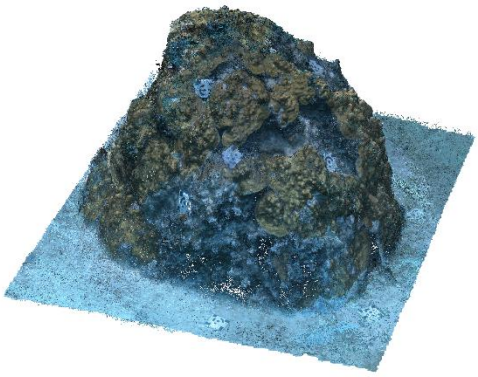

Fig. 19 – The dense point cloud generated from the depth maps created as a result of the images being aligned. By estimating the relative location of the camera when each image was taken, trigonometry can be used to create depth maps for pairs of images, thus giving the pixels within each a location in 3-D space.

E. Shaded and Textured Mesh

From this point cloud a triangular mesh is created using the Poisson surface reconstruction algorithm (or some similar variant) to approximate the surface of the object being modeled (Fig. 20). This mesh is made up of many elements (vertices, edges, faces, etc.) that also act as data structures storing the associated attributes such as location, color components, normal vectors, light reflectance values, and texture coordinates.

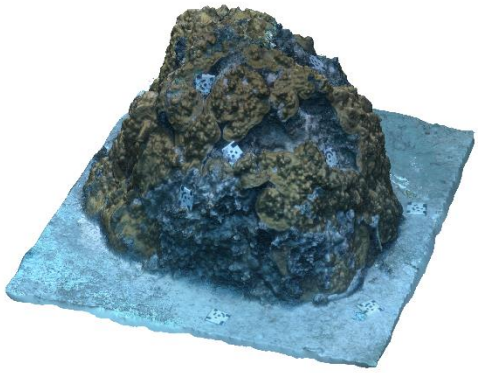

Fig. 20 – The shaded mesh created from the dense point cloud. Color is provided to the mesh by assigning the color component values found in the pixel indices to the corresponding elements making up the mesh.

The texture or, "UV" coordinates represent the mesh in two dimensions, which conceptually can be thought of as flattening or unfolding the 3-D model to form a 2-D image. Through UV mapping, portions from the source images that were used in the reconstruction process are mapped onto the 2-D representation of the model creating what is referred to as a 'texture atlas'; note that these textures are different from the color components and instead consist of groups of pixels in the shape of a triangle that are grafted from the source image onto the atlas. When the texture atlas is applied to the mesh and represented in 3-D it forms a textured mesh and is often used for display purposes (Fig. 21); typically, the underlying mesh and the dense point cloud that were created in the previous steps are what are used for making precise measurements during various spatial analyses.

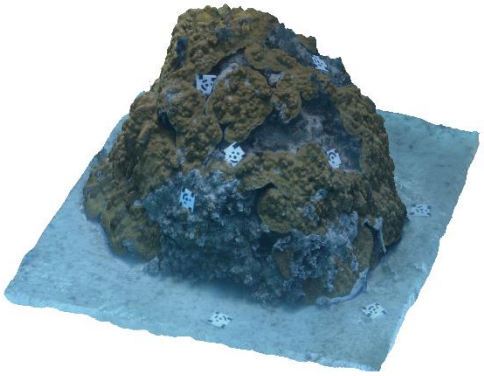

Fig. 21 – The textured mesh created from the images that were used in the reconstruction process. Unlike the point clouds and the shaded mesh, the appearance of the textured mesh does not come directly from the color component values of the pixel indices. Instead, groups of pixels that are thought to best represent an area of the 3-D model are taken from the images and grafted onto it.