AUTOMATIC ESTIMATION OF EELGRASS COVER USING SEAFLOOR IMAGES

by

Paras Mehta

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
June 2024

I would like to dedicate this thesis to my family who has been a pillar of support throughout this journey.

Table of Contents

List of Tables

List of Figures

Abstract

This research assesses various methods to monitor eelgrass populations along Canada's eastern coasts. This is done by providing image data to deep learning models to estimate the percentage cover of eelgrass from these models. The dataset comprises ocean floor images obtained through kayak and diver surveys. Human-estimated percentage covers of eelgrass in these images are used to evaluate our models. Image classification and image segmentation (pixel-wise classification) approaches are evaluated in this research. Image classification determines the percentage cover estimates by discretizing these estimates into 6 classes representing eelgrass cover, and image segmentation does this by generating segmentation masks and extracting percentage cover information from the pixels identified as eelgrass. The models were either pre-trained on the BenthicNet dataset or used with random weight initialization, together with various pre-processing techniques for image segmentation. Two separate datasets were used to compare model performances on unseen data, where the first dataset corresponds to same-domain images because they are collected from locations nearby to the training data collection sites, ensuring similar characteristics, whereas the second dataset corresponds to different-domain images because they are collected from randomly distributed locations, providing a diverse set of characteristics. We found that the segmentation models underestimated eelgrass percentage cover, and the classification models overestimated. All the segmentation models used, performed equivalently, except the **V**GG**16** encoder pre-trained on **I**mageNet dataset with a **UNET** decoder (V16I-UNet). All classification models had similar but worse results than the segmentation models.

List of Abbreviations and Symbols Used

SSL	**S**elf-**S**upervised **L**earning
BIIGLE	**B**io-**I**mage **I**ndexing and **G**raphical **L**abelling **E**nvironment
MAIA	**M**achine-learning **A**ssisted **I**mage **A**nnotation method
CNN	**C**onvolutional **N**eural **N**etwork
MLP	**M**ulti-**L**ayer **P**erceptron
R50B-CNN	**R**esNet**50** encoder pre-trained on **B**enthicNet dataset with a simple **CNN** decoder
V16I-UNet	**V**GG**16** encoder pre-trained on **I**mageNet dataset with a **UNET** decoder
V16B-UNet	**V**GG**16** encoder pre-trained on **B**enthicNet dataset with a **UNET** decoder
V16-MLP	**V**GG**16** encoder with a **M**ulti-**L**ayer **P**erceptron
V16B-MLP	**V**GG**16** encoder pre-trained on **B**enthicNet dataset with a **M**ulti-**L**ayer **P**erceptron
R50-MLP	**R**esNet**50** encoder with a **M**ulti-**L**ayer **P**erceptron
R50B-MLP	**R**esNet**50** encoder pre-trained on **B**enthicNet dataset with a **M**ulti-**L**ayer **P**erceptron
VS-MLP	**V**iT-**S**mall encoder with a **M**ulti-**L**ayer **P**erceptron
VB-MLP	**V**iT-**B**ase encoder with a **M**ulti-**L**ayer **P**erceptron
VSB-MLP	**V**iT-**S**mall encoder pre-trained on **B**enthicNet dataset with a **M**ulti-**L**ayer **P**erceptron
VBB-MLP	**V**iT-**B**ase encoder pre-trained on **B**enthicNet dataset with a **M**ulti-**L**ayer **P**erceptron
.	Scalar-Scalar Multiplication
X	Matrix Multiplication

Acknowledgements

I am profoundly grateful to my thesis supervisor, Dr. Thomas Trappenberg, for his support and inspiration throughout this journey. His ability to keep me focused on the ultimate goal, coupled with strong and constructive feedback at all stages, has been invaluable. His mentorship has enhanced the quality of this research and fostered my growth as a researcher.

I also want to express my deepest gratitude to Dr. Benjamin Misiuk for his invaluable support and guidance throughout this project. His unique expertise in Marine Biology and Computer Science has been instrumental in shaping the direction and success of this research. His insightful feedback and encouragement have been crucial at every stage of this work.

Lastly, I would like to thank my lab members, Martin Gillis and Isaac Xu, fellow PhD students at the Hierarchical Anticipatory Learning (HAL) lab, at Dalhousie University, for their invaluable assistance with coding challenges and for suggesting innovative approaches to the problems I encountered. Their support and insights have been greatly appreciated.

Chapter 1

Introduction

Eelgrass (*Zostera marina*) is a plant that visually resembles grass and grows in or around the sea. Some benefits of having eelgrass are stabilizing sediments [1], carbon sequestration [2], and reducing the force of waves [3]. It also provides food for different marine organisms and acts as a habitat for some fish and animals [4]. However, many factors are causing a decline in the eelgrass population. Rising sea levels [5] and ocean warming [6] are among these factors. Hence, conservation of eelgrass has become an important task. The first step towards conservation is monitoring. Obtaining geographical and quantitative information about eelgrass would be helpful in monitoring.

Analyzing images from the ocean floor to determine the percentage of eelgrass cover can help monitor the overall coverage in a geographical region. This process involves collecting images from the ocean floor within a specific area. Each image is then assessed to estimate the percentage of eelgrass cover. These individual estimates are averaged to provide an overall approximation of eelgrass cover in the region.

The focus of this study is to assess images to determine the percentage of eelgrass cover. Manually performing this task would be time-consuming and labour-intensive. Therefore, we explore automatic assessment methods using deep learning models. Chapter 3 highlights multiple studies that have employed similar methods. Deep learning models are designed to recognize and distinguish various objects and elements within an image. During this process, the network is trained on a dataset of human-labelled images. The labels provide information to the model concerning the percentage cover of eelgrass. The model learns to detect features such as edges, textures, shapes, and more complex structures at multiple layers of abstraction. The model then uses these learned features to identify eelgrass covers within these images. This automated feature extraction process expedites the process of eelgrass cover estimation.

This study uses image classification and image segmentation using various deep-learning models to map eelgrass coverage. For experiments with image classification, images are categorized into discrete classes based on the percentage cover of eelgrass. Image classification then assigns an image to one of these categories. Experiments with image segmentation take a pixel-wise approach to classify each pixel as either eelgrass or background. By summing the pixels identified as eelgrass, we determine the overall percentage cover.

We explore five distinct techniques for pre-processing training and validation data. The first technique employs unaltered images and masks, serving as a baseline approach with no modifications. The second technique uses sharpened images to enhance edge definition, while the masks remain unchanged. The third approach keeps the images in their original state but modifies the masks to improve annotation accuracy. The fourth technique involves both sharpening the images and adjusting the masks. Finally, the fifth technique utilizes a reduced dataset, excluding low-quality images and masks, to focus on higher-quality data. We also compared models initialized with random weights and models initialized with BenthicNet pre-trained weights in image classification.

The study aims to test three primary hypotheses. The first hypothesis is that if edge detection and clustering are applied to segmentation masks before using them for training an image segmentation model, then it will aid the model in producing better segmentation masks. Edge detection highlights the boundaries of eelgrass within the segmentation masks, making the transitions between eelgrass and non-eelgrass regions more distinct than in human-annotated segmentation masks. This clarity helps the model to learn precise boundaries. The second hypothesis is that if low-quality images are removed from the training dataset of the model, then the model performs better at producing segmentation masks and more robust predictions are generated. Low-quality images usually contain noise, which can obscure important features and make it harder for the model to learn useful patterns. Removing these images, makes the dataset clearer, allowing the model to focus on learning from high-quality, informative examples. The third hypothesis is that both image classification and image segmentation models that are pre-trained on a similar dataset, will perform better than the models with random weight initialization. Models pre-trained on

similar datasets have already learned to extract useful features relevant to the new task.

The structure of this thesis is organized to explore and address the challenges in automating eelgrass monitoring using deep learning techniques and answer the hypotheses comprehensively. Following this introduction, Chapter 2 delves into the background that provides context on the research topic. Chapter 3 reviews the existing literature on determining percentage cover estimates, and eelgrass monitoring using different techniques. Chapter 4 details the methodology of conducting experiments, including data collection, data pre-processing, model configurations, and the experiments. Chapter 5 shows the results of the experiments, providing a comparative analysis of the models, the impact of different pre-processing techniques on segmentation, and the effects of different initializations on classification. A detailed analysis of segmentation masks and the hypotheses test results are also included in this chapter. Finally, Chapter 6 gives suggestions for future research directions and key takeaways from this study.

Chapter 2

Background

This chapter introduces terminologies essential to understanding the contents of the following chapters. The first section introduces self-supervised learning. This section also introduces the basic working of the MoCov3 model used for pre-training the vision transformer encoders and the Barlow Twins model used for pre-training the VGG16 and ResNet50 encoders. This is followed by a section containing a detailed explanation of the model architectures used in this study. Next, an explanation of the use of evaluation metrics used in this study is delineated. Lastly, the superpixels method used by a study in Chapter 3 is explained briefly.

2.1 Self-Supervised Learning

Self-**S**upervised **L**earning (SSL) [7] is a type of machine learning algorithm where the model learns to predict part of its input from other parts, essentially creating its labels from the data. This method does not require manually labelled data like supervised learning. Instead, it uses structures and patterns in the data to learn representations.

In SSL, there are two primary types of tasks, which are pretext [8] and downstream tasks [9, 10]. In pretext tasks, SSL trains models to grasp important representations of unstructured data. These representations are then used in downstream tasks like image segmentation. Transfer learning [11, 12] is the process where models trained on pretext tasks are then used to train the model on a downstream task because the learned representations act as a "starting point"[13] for the model on the new dataset.

Contrastive learning introduced in [14] is a pretext task in SSL, where the model learns to distinguish between similar and dissimilar examples. Models like SimCLR introduced in [15] and MoCo introduced in [16] use contrastive learning to increase similarities between augmentations of the same image, while decreasing similarities

between augmentations of distinct images. Augmentation [17] refers to applying various transformations to the existing data. Different augmentations include geometric transformations, colour modifications and noise addition. Two examples of SSL models used for transfer learning here are MoCov3 and Barlow Twins.

2.1.1 Momentum Contrast V3

Momentum Contrast (MoCo) v3 introduced in [18] is an evolution of MoCo, which stands for Momentum Contrast. It is a family of self-supervised learning algorithms that aim to learn visual representations by contrastive learning. The following paragraphs briefly explain how MoCo v3 works.

Two encoders are used in the architecture: a query encoder (f_q) and a key encoder (f_k). These encoders share the same architecture but have different parameters during training. For every input image x in a batch, two augmented views are generated. These views are referred to as x_q and x_k. Other images in the batch act as negative pairs. x_q and x_k are fed into the query encoder and key encoder, respectively, to produce $query = f_q(x_q)$ and $key = f_k(x_k)$. MoCo maintains a queue of encoded keys. This queue is a dynamically updated memory bank that stores feature representations from previous batches. The current batch's keys are enqueued to this memory bank, and the oldest entries are dequeued to maintain a fixed size. A contrastive loss (introduced in [19]) is used to train the model. The objective is to cluster the embeddings of the same image (positive pairs) while separating the embeddings of different images (negative pairs). For each query q, the corresponding key k from the same image is considered a positive example, while the other keys in the queue serve as negative examples.

$$\theta_k = m \cdot \theta_k + (1 - m) \cdot \theta_q \tag{2.1}$$

Equation 2.1 taken from [18] shows the momentum update rule where m is the momentum coefficient which regulates the update rate of the key encoder, θ_k are the key encoder parameters, and θ_q are the query encoder parameters. This update helps ensure that the key encoder evolves more smoothly over time. The entire process is repeated for each batch of images during training. The query and key encoders process the images, the queue is updated, and the contrastive loss is minimized to

improve the feature representations.

2.1.2 Barlow Twins

In Barlow Twins introduced in [20], the task is to make the representations of different augmentations of the same image as similar as possible. The idea is to reduce the redundancy between parts of the learned representations. The following paragraphs explore the steps involved in processing images using Barlow twins.

Initially, a series of augmentations is applied to each image to create two different views of the same image. Then, a backbone neural network is used to extract features from the augmented images. This can be any network that identifies important features from the input. Both augmentations are passed through the same backbone network. This results in two sets of embeddings for each image. Finally, the cross-correlation matrix C between the embedding vectors of the two augmentations across a batch of images is computed.

$$C_{ij} = \frac{\sum_b \left(z^A_{b,i} \times z^B_{b,j}\right)}{\sqrt{\sum_b (z^A_{b,i})^2} \cdot \sqrt{\sum_b (z^B_{b,j})^2}} \tag{2.2}$$

Equation 2.2 taken from [20] depicts the cross-correlation matrix where b traverses through the whole batch of size b, and i and j represent the dimensions of the networks' outputs. z^A and z^B show the two embeddings obtained after applying different augmentations to the image. Both these embeddings have dimensions $[b \cdot D]$ where D is the embedding dimension. z^A is transposed before multiplying with z^B to produce a numerator of dimensions $[b \cdot b]$. C is a square matrix with size the dimensionality of the network's output, with values comprised between -1 showing perfect anti-correlation and 1 showing perfect correlation. The loss function encourages the diagonal elements of the cross-correlation matrix C_{ij} to be close to 1, which ensures that each feature dimension has similar values for the two augmentations from the same image. However, the off-diagonal elements C_{ij} (where $i! = j$) are encouraged to be close to 0, which reduces redundancy by ensuring that different feature dimensions are correlated.

$$\mathcal{L} = \underbrace{\sum_i (1 - C_{ii})^2}_{\text{invariance term}} + \lambda \cdot \underbrace{\sum_i \sum_{j \neq i} C_{ij}^2}_{\text{redundancy reduction term}} \tag{2.3}$$

The loss function based on the cross-correlation matrix is shown in Equation 2.3 taken from [20] where λ is a hyperparameter that balances the two terms. It is evident from Equation 2.3 that as the correlation between two augmentations of the same image (diagonal elements) increases, the loss decreases, and as the correlation between the augmentations of different images (off-diagonal elements) decreases, the loss decreases. The process of applying augmentations, extracting features, computing the cross-correlation matrix, calculating the loss, and updating the network parameters continues until the model converges.

2.2 Models Used for Image Segmentation

In this section, we present the models utilized in this study to address the task of image segmentation. They are Segformer, **V**GG**16** encoder pre-trained on **I**mageNet dataset with a **UNET** decoder (V16I-UNet), V16B-UNet and **R**esNet**50** encoder pre-trained on **B**enthicNet dataset with a simple **CNN** decoder (R50B-CNN). They are explained in the following sections.

2.2.1 Segformer

Segformer introduced in [21] is an image segmentation model that comprises an encoder-decoder architecture. Segformer uses transformers [22] to capture contextual information in the image and produce segmentation masks. Transformers were originally used in natural language processing tasks like text summarization [23], text generation [23], and text-to-speech [24]. Later, they became popular with image data [25]. The segformer architecture used in this study is taken from the GitHub repository by Phil Wang [26]. The following sections explain this architecture in detail.

Segformer Encoder

The input images are fed into the encoder for training. The encoder contains three main mechanisms: overlapping patch construction, patch embedding generation, and transformer block operation. Each of these three mechanisms is implemented four times, with each iteration referred to as a component. The output from each of these components is stored to be used by the decoder. These stored outputs are referred

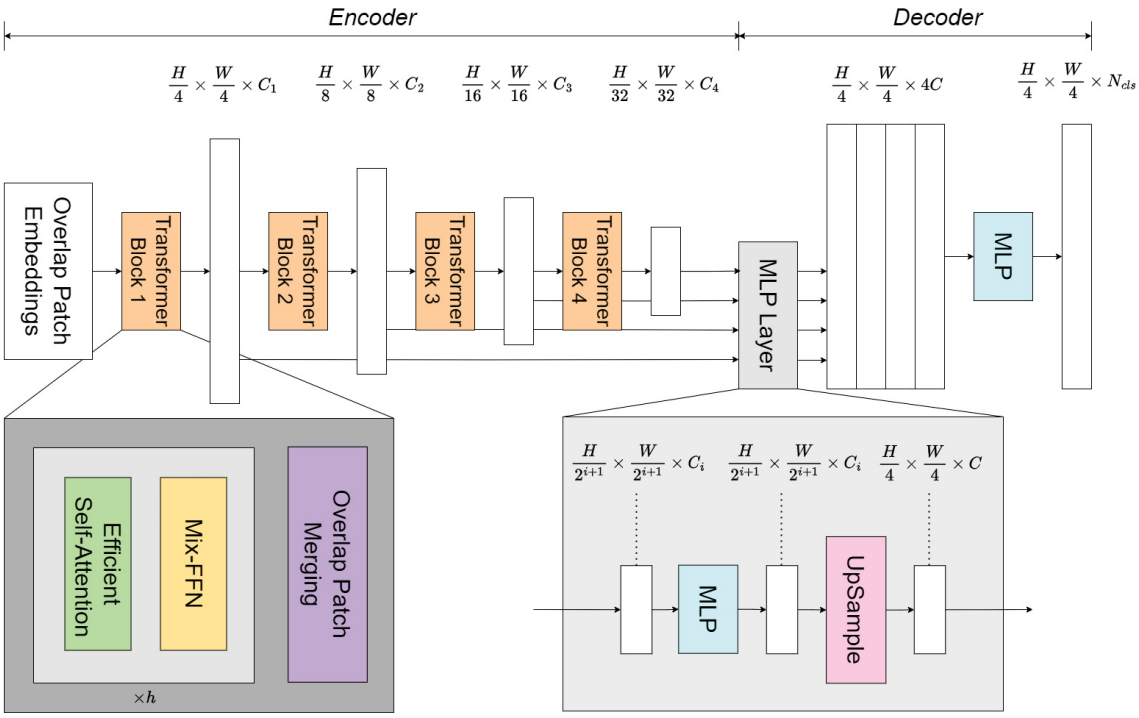


Figure 2.1: Encoder-decoder architecture of the Segformer. Here, the input images of dimension $[H, W, C]$ are passed to the overlap patch embeddings and the segmentation masks are obtained from the MLP layer with output dimensions $\left[\frac{H}{4}, \frac{W}{4}, N_{cls}\right]$ (original image from [21]).

to as skip connections.

Subsequent sections in the encoder are explained taking into account the first component of the encoder. The remaining follow the same operations with a change only in three sets of parameters. The first parameter is the number of transformer heads h which is 1, 2, 5, and 8 respectively in the four transformer blocks. The second set of parameters is the kernel K, stride S, and padding P (K, S, P) used in the overlapping patch construction process. They are $(7, 4, 3)$ for the first component and $(3, 2, 1)$ for the remaining three components. Lastly, the channels in the outputs of the transformer blocks are 64, 128, 320, and 512 in the four components respectively, which are referred to as C_1, C_2, C_3, and C_4 in Figure 2.1. The internal workings of the three mechanisms in the first component are explained below.

The first mechanism is the overlapping patches mechanism. Here, the input image of size $[H, W, C]$ (height, width, and channels) is divided into overlapping patches by passing it through a convolutional layer with a kernel size of 7, a stride of 4, and a

padding of 3. Every patch will have dimensions $[7, 7, C]$ where the C channels account for the original number of channels in the image. The dimensions of the output generated by this convolutional operation will be $\left[\frac{H}{4}, \frac{W}{4}, K \cdot K \cdot C\right]$. Equation 2.4 shows the number of patches n generated after the overlapping patches mechanism.

$$n = \frac{H \cdot W}{49} \tag{2.4}$$

The second mechanism embeds the overlapped patches obtained from the overlapping patches mechanism using an embedding dimension $d_{model} = 64$. A convolutional operation does this with a kernel size of 1, and using 64 filters. The patch embeddings produced from this operation are of dimensions $\left[\frac{H}{4}, \frac{W}{4}, d_{model}\right]$.

The third mechanism contains the transformer block consisting of two main components — efficient self-attention and mixed feed-forward network, both repeated h times where h is the number of heads of the transformer block. Every head of the block captures different context information from the image. The outputs from all the heads are then concatenated and passed on to the mix feed-forward network. Inside a transformer head, patch embeddings are passed to the self-attention block as input. Patch embeddings are split into three parts namely query Q, key K, and value V by multiplying them with learnable weight matrices W_Q, W_K, and W_V with dimensions $[d_{model},\ d_{model}/h]$. The query Q contains the representations of the areas within the image that the model is currently analyzing. The key K serves as a comprehensive set of criteria from the whole image against which the query vectors are compared. The value V represents the information extracted from the regions of the image, guided by the attention mechanism computed by query and key. Self-attention is computed using Equation 2.5 as used in [25].

$$Attention(Q, K, V) = softmax\left(\frac{Q \times K^T}{\sqrt{d_{model}/h}}\right) \times V \tag{2.5}$$

The attention map identifies the correlation of every patch with every other patch. The dimensions of the attention map generated from one head are $\left[\frac{H}{4}, \frac{W}{4}, \frac{d_{model}}{h}\right]$. Attention outputs from all the heads are concatenated along the channel dimension to form the output of the self-attention block.

$$MultiHead(Q, K, V) = Concat(head_1, head_2, ..., head_h) \times W^O \tag{2.6}$$

(Kernel, Stride, Padding)	Input channels	Output channels
(1, 1, 0)	d_{model}	hidden_dim
(3, 1, 1)	hidden_dim	hidden_dim
(1, 1, 0)	hidden_dim	hidden_dim
(1, 1, 0)	hidden_dim	d_{model}

Table 2.1: Shows the kernel, stride, and padding of the MLP inside the encoder of Segformer's transformer block.

Equation 2.6 shows the concatenation and projection of this concatenated output to d_{model} dimension. In this equation, W^O is a weight matrix that projects the concatenated output back to d_{model}. This concatenated output is passed on to the mix feed-forward network which gets an input dimension of $\left[\frac{H}{4}, \frac{W}{4}, d_{model}\right]$. The Mix-FFN layer consists of a few convolutional layers that aim to capture more details in the images by expanding and contracting the inputs obtained from the multi-head self-attention block. These channel-mixing operations allow the model to integrate information across different dimensions effectively. This block uses a GeLU activation function [27] that has a smooth gradient transition facilitating training. The configurations for the convolutional layers are shown in Table 2.1. The output dimensions except the channel dimension remain the same as the input dimensions during all the convolutional operations. The outputs of the four transformer blocks are passed on to the decoder. The dimensions of these outputs are $\left[\frac{H}{4}, \frac{W}{4}, C_1\right]$, $\left[\frac{H}{8}, \frac{W}{8}, C_2\right]$, $\left[\frac{H}{16}, \frac{W}{16}, C_3\right]$, and $\left[\frac{H}{32}, \frac{W}{32}, C_4\right]$ as shown in Figure 2.1.

Segformer Decoder

There are primarily two components in the decoder — a multi-layer perceptron and an upsampling layer. The multi-layer perceptron contains a convolutional layer that embeds the concatenated transformer head outputs to a uniform dimension $dim_{decoder} = 256$. This is done through a convolutional layer containing a kernel size of 1. The upsampling layer upsamples the height and width of these outputs to a uniform size. The operations are shown in Table 2.2.

After the upsampling, all the outputs are concatenated along the channel dimension which produces an output of dimension $\left[\frac{H}{4}, \frac{W}{4}, dim_{decoder} \cdot 4\right]$. This output is passed on to the final two convolutional layers having kernel size 1, which convert the

$$
\begin{array}{lll}
\left[\frac{H}{4}, \frac{W}{4}, C_1\right] \xrightarrow{Conv} \left[\frac{H}{4}, \frac{W}{4}, dim_{decoder}\right] & \Rightarrow & \left[\frac{H}{4}, \frac{W}{4}, dim_{decoder}\right] \\
\left[\frac{H}{8}, \frac{W}{8}, C_2\right] \xrightarrow{Conv} \left[\frac{H}{8}, \frac{W}{8}, dim_{decoder}\right] & \xrightarrow{Upsample} & \left[\frac{H}{4}, \frac{W}{4}, dim_{decoder}\right] \\
\left[\frac{H}{16}, \frac{W}{16}, C_3\right] \xrightarrow{Conv} \left[\frac{H}{16}, \frac{W}{16}, dim_{decoder}\right] & \xrightarrow{Upsample} & \left[\frac{H}{4}, \frac{W}{4}, dim_{decoder}\right] \\
\left[\frac{H}{32}, \frac{W}{32}, C_4\right] \xrightarrow{Conv} \left[\frac{H}{32}, \frac{W}{32}, dim_{decoder}\right] & \xrightarrow{Upsample} & \left[\frac{H}{4}, \frac{W}{4}, dim_{decoder}\right]
\end{array}
$$

Table 2.2: Operations performed by the Segformer decoder on encoder outputs.

channels to the number of classes required in the segmentation masks. The first layer converts the $dim_{decoder} \cdot 4$ to $dim_{decoder}$ and the second layer converts the $dim_{decoder}$ to N_{cls} as shown in Figure 2.1

Interpolation

The segmentation masks generated from Segformer are one-fourth the size of the original images given as input. Hence, they need to be interpolated to the original size. This interpolating is done with the help of bilinear interpolation from [28]. This is a resampling method used in image processing to resize images. It uses the four nearest pixel values around the target pixel to compute a weighted average, producing a smoother image as compared to nearest-neighbour interpolation.

2.2.2 VGG16 Encoder With a UNet Decoder

The VGG16 encoder with a UNet decoder is another model used for segmentation here. UNet was first developed for biomedical image segmentation [29] and then popularized for various image segmentation tasks. This is an encoder-decoder architecture like the Segformer. The choice of encoder in a UNet architecture significantly impacts the model's performance for specific tasks and datasets. Different types of encoders can be used in the UNet architecture. The encoder used for experiments in this study is the VGG16 encoder architecture. VGG16 [30] is a CNN architecture that was proposed by the Visual Geometry Group (VGG) at the University of Oxford. It has a simple and uniform design, using only $[3, 3]$ convolutional layers stacked on top of each other with a fixed stride and padding, followed by $[2, 2]$ max-pooling layers. This combination of the UNet decoder and VGG16 encoder was first introduced by Pravitasari et al. in [31] for brain tumor segmentation, and the implementation of

this architecture used in this study is taken from the GitHub repository by Zheng Zhou [32].

Two differently initialized versions of the VGG16 encoder were used in this study. The first comprises pre-trained weights from the ImageNet dataset [33] and the second contains pre-trained weights from the BenthicNet dataset [34]. The former model is referred to as V16I-UNet and the latter model is referred to as V16B-UNet. The pre-trained weights are obtained using the MoCov3 model [18] to train the VGG16 encoder on the BenthicNet dataset. This process of obtaining the pre-trained model is described briefly in section 2.1.1. The BenthicNet dataset is explained briefly in section 4.1.1.

VGG16 Encoder

The encoder extracts essential features from the input and compresses it into a lower-dimensional representational space. To recover fine-grained details in the predictions, encoder outputs are stored after a certain number of layers for the decoder to use during the reconstruction process. The following paragraphs explain the process of an image going through the encoder.

The input image is a grid of pixel values. This image of dimensions $[H, W, C]$ is passed through block 1 (Figure 2.2). Initially, the image is passed through a convolutional layer with 64 filters. These filters detect basic features like edges, corners, and textures. The result of this operation is a set of feature maps of dimensions $[H, W, 64]$, where each slice along the depth (64) represents a feature map produced by one filter. The outputs of the convolutional layer are passed to the batch normalization layer, which normalizes the outputs from the previous layer. The ReLU activation function [35, 36] is applied to introduce non-linearity. The feature maps stay the same size $[H, W, 64]$, but now they have non-linear properties. The feature maps are passed through another set of filters. This layer has more filters to detect more complex features. It detects more complex patterns built upon the basic features identified by the first layer. This layer uses 64 filters, and the output dimensions are $[H, W, 64]$. The outputs are normalized again using the batch normalization layer. ReLU is then applied to introduce non-linearity. All these processes were performed inside block 1 shown in Figure 2.2 inside the VGG16 Encoder. The output of block 1 is stored to be

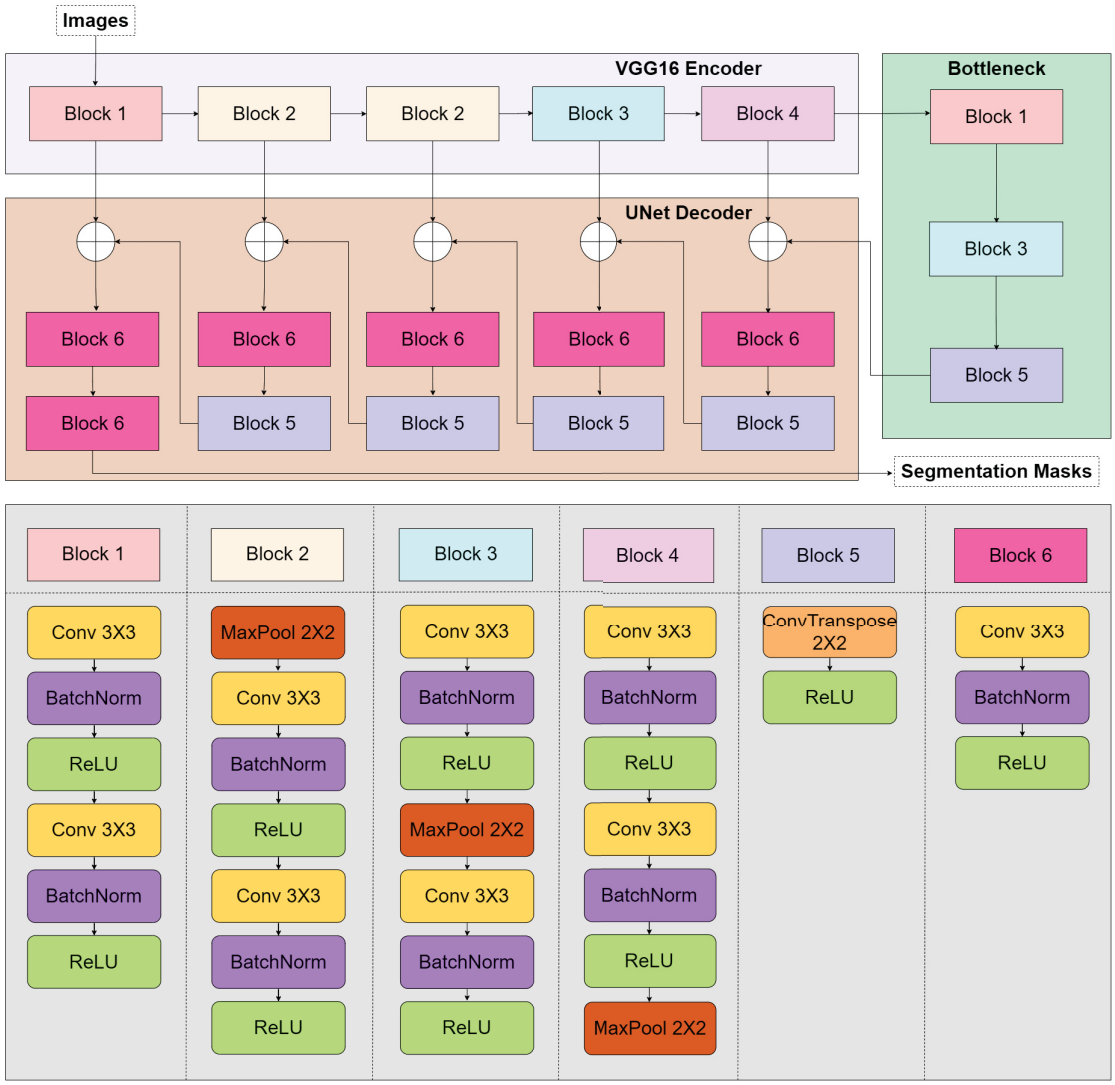


Figure 2.2: Encoder-decoder architecture of UNet containing a VGG16 encoder which contains convolutional, max-pooling and transpose convolutional layers with skip connections in between to remember encoding information efficiently. The repeated blocks have the same architecture and are not the same layers being reused.

used later by the decoder. The feature maps from block 1 are passed to block 2 which has the same architecture as block 1, except it has a preceding max pooling layer. This layer takes the maximum value in each window (e.g., $[2, 2]$ window) from the feature maps. For a $[2, 2]$ max pooling operation, the output size is $\left[\frac{H}{2}, \frac{W}{2}, 64\right]$. The output from block 2 after the convolution, normalization, and activation functions are applied again, is stored. This process of convolution, batch norm, activation, and

pooling is repeated multiple times. Each subsequent layer detects increasingly complex and abstract features. As the layers go deeper, the spatial dimensions continue to decrease (e.g., $\left[\frac{H}{4}, \frac{W}{4}\right]$, $\left[\frac{H}{8}, \frac{W}{8}\right]$, while the number of feature maps (filters) usually increases. The final output which is a compressed representation of the original image is a $\left[\frac{H}{16}, \frac{W}{16}, 512\right]$ feature map, representing very high-level features.

Bottleneck

The bottleneck layer is a connecting layer between the encoder and decoder which consists of blocks 1, 3, and 5 containing convolutional, normalization, activation, and max pooling layers as shown in Figure 2.2. It takes a $\left[\frac{H}{16}, \frac{W}{16}, 512\right]$ dimensional feature map as input from the encoder and gives out a $\left[\frac{H}{32}, \frac{W}{32}, 1024\right]$ dimensional output to the decoder.

UNet Decoder

The decoder takes the compressed representation and reconstructs it back to a higher-dimensional space. It generates segmentation masks as output. The outputs from blocks 1, 2, 3, and 4 of the encoder, and from the bottleneck layer are passed to the decoder for processing.

The decoder begins by adding the outputs from block 5 of the bottleneck layer and block 4 of the encoder. This is passed to blocks 6 and 5 in succession. The process starts by passing the added output to convolution, normalization, and activation layers. This output is then upsampled by the transpose convolution layer to increase its spatial dimensions. Upsampling reverses the downsampling process performed by the encoder, gradually reconstructing the spatial resolution. The output is a $\left[\frac{H}{16}, \frac{W}{16}, 512\right]$ dimensional feature map as the upsampling doubles each dimension except the channel dimension. The ReLU activation function enhances non-linear properties. The output of the activation function is added again with the encoder's intermediate output from block 3 to aid the construction of the segmentation mask. This combines the encoder's $\left[\frac{H}{16}, \frac{W}{16}, 512\right]$ intermediate output with the decoder's $\left[\frac{H}{16}, \frac{W}{16}, 512\right]$ feature map to form a $\left[\frac{H}{16}, \frac{W}{16}, 512\right]$ output. After upsampling, a convolutional layer is applied to refine the feature map and add more detail. This layer helps to refine the upsampled features and make them more representative of the original image content. This

process of adding intermediate outputs from the encoder and upsampling continues until the output produced becomes the same size as the input image. Each stage increases the spatial resolution and refines the features. The feature map sizes gradually increase (e.g., $\left[\frac{H}{4}, \frac{W}{4}\right]$, $\left[\frac{H}{2}, \frac{W}{2}\right]$, etc.) while the number of feature maps decreases (e.g., 128, 64, etc.). A final convolutional layer reduces the number of feature maps to match the number of channels in the segmentation mask (e.g., 2 for two classes). The final output is a $[H, W, N_{cls}]$ array, representing the segmentation mask containing N_{cls} classes to be distinguished.

2.2.3 ResNet50 Encoder With an Upsampling Convolutional Neural Network

ResNet50, introduced in [37], is a deep convolutional neural network architecture that belongs to the family of Residual Networks (ResNets). ResNets introduced residual blocks that implement skip connections. These connections are used to pass information from multiple previous layers to later layers, ensuring that information from earlier layers is not lost and the vanishing gradient problem is addressed.

The ResNet50 encoder with an upsampling convolutional neural network combines a ResNet50 encoder and a simplified small CNN decoder and is referred to as R50B-CNN in further sections. The encoder is pre-trained using the BenthicNet dataset. For pre-training, the ResNet50 encoder is inserted into the Barlow Twins architecture explained briefly in section 2.1.2, and it is trained as a pretext task using self-supervised learning to produce the weights used for the downstream segmentation task.

ResNet50 Encoder

Four primary stages in the ResNet50 encoder contain multiple convolutional layers, batch normalization layers, and activation functions with one max-pooling layer.

A given image of size $[H, W, C]$ is first passed through a convolutional layer of 64 filters with a kernel size of 7, a stride of 2, and a padding of 3, which converts the input image to the dimensions $[H, W, 64]$. This is followed by a batch normalization and ReLU layer that normalizes and introduces non-linearity in the input.

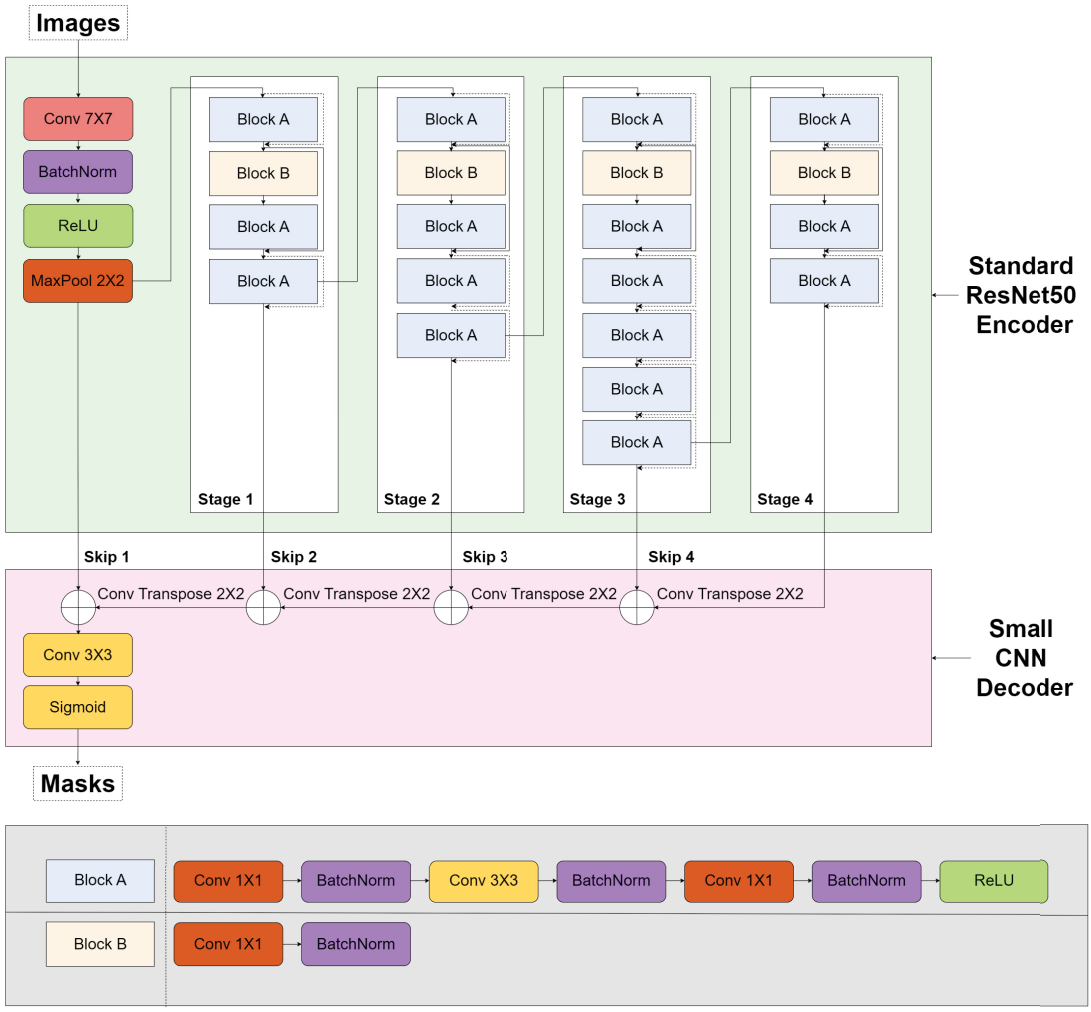


Figure 2.3: ResNet50 Encoder With An Upsampling Convolutional Neural Network model for image segmentation which contains a popular ResNet50 architecture as the encoder and a smaller decoder compared to other well-known segmentation models.

A max-pooling layer then captures high-level features from these non-linearized outputs producing an output of dimensions $\left[\frac{H}{2}, \frac{W}{2}, 64\right]$. This output is stored as skip 1 as shown in the Figure 2.3. Further stages contain multiple convolution, normalization, and activation functions that contain residual connections in between. There are two types of residual connections used in the architecture. The dashed arrows in Figure 2.3 represent the connections that add the previous layer outputs to the current layer outputs, and the solid arrows indicate the connections that concatenate the previous layer outputs to the current layer outputs. Stages 1, 2, and 3 produce

skip connections skip 2, skip 3, and skip 4 shown in Figure 2.3 that are stored for the decoder to facilitate learning via residual connections. Finally, stage 4 produces an output of dimensions $\left[\frac{H}{32}, \frac{W}{32}, 2048\right]$. Each subsequent layer detects increasingly complex and abstract features. As the layers go deeper, the spatial dimensions continue to decrease, while the number of feature maps (filters) increases. These outputs are then passed on to the simplified decoder.

CNN Decoder

The decoder consists of transposed convolutional layers that perform the reverse operations of the convolutions in the encoder and add the outputs of the skip connections along the architecture. This architecture is different from the usual decoder architectures which contain multiple activation functions and normalization layers. Only upsampling layers are included in this decoder making it suitable for resource-constrained environments.

The output of stage 4 of the encoder shown in Figure 2.3 containing dimensions $\left[\frac{H}{32}, \frac{W}{32}, 2048\right]$, is passed through a transposed convolutional layer to upsample and produce an output of dimensions $\left[\frac{H}{16}, \frac{W}{16}, 1024\right]$. This output is added to skip 4 from the encoder which also has the dimensions $\left[\frac{H}{16}, \frac{W}{16}, 1024\right]$. This is followed by three more upsampling and adding layers that produce outputs of dimensions similar to the skip connection matrices. Each transposed convolutional layer increases the spatial resolution and refines the features. The feature map sizes gradually increase while the number of feature maps decreases. The output is then passed through a final convolutional layer that converts the output to the required number of channels in the segmentation mask which is equal to N_{cls}. The output dimensions are $[H, W, N_{cls}]$. A sigmoid activation function is then used to convert the logits into a range of 0 to 1.

2.3 Models Used for Image Classification

2.3.1 Vision Transformer

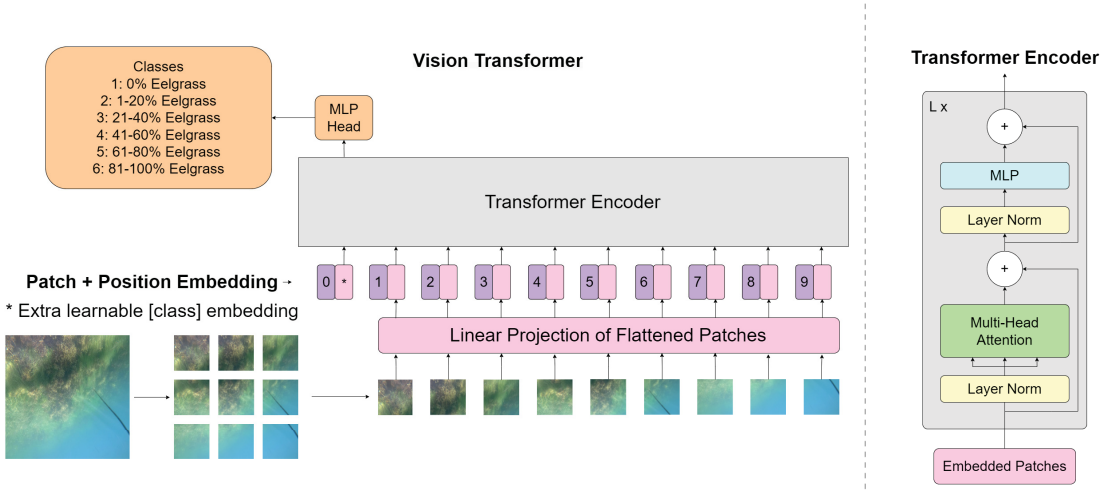


Figure 2.4: Architecture of the vision transformer model along with the internal mechanisms of the transformer encoder (adapted from [25]).

Vision Transformer (ViT) introduced in [25] employs a stack of transformer encoder layers to process image patches and extract hierarchical features. By iteratively refining features through transformer encoders, the vision transformer achieves high-dimensional representations that produce accurate classifications. The implementations of the vision transformer were taken from a GitHub repository by Victor Turrisi [38]. There are four different versions of the vision transformer used for this study. They are VS-MLP (**V**iT-**S**mall encoder with a **M**ulti-**L**ayer **P**erceptron), VB-MLP (**V**iT-**B**ase encoder with a **M**ulti-**L**ayer **P**erceptron), VSB-MLP (**V**iT-**S**mall encoder pre-trained on **B**enthicNet dataset with a **M**ulti-**L**ayer **P**erceptron), and VBB-MLP (**V**iT-**B**ase encoder pre-trained on **B**enthicNet dataset with a **M**ulti-**L**ayer **P**erceptron). ViT-Small has a smaller number of parameters compared to ViT-Base as it is designed to be more lightweight and computationally efficient. Whereas, the ViT-Base has higher capacity and potentially better performance, especially on larger and more complex datasets. The only difference between them is the embedding dimension d_{model} used for the models. d_{model} is 384 for ViT-Small and 768 for ViT-Base.

A given input image of size $[H, W, C]$ is converted into small non-overlapping patches of size $[16, 16, C]$ in the process of forming patches. Equation 2.7 shows the total number of patches (n) formed in this process.

$$n = \frac{H \cdot W}{16 \cdot 16} \tag{2.7}$$

These patches are linearized into dimensions $[N, 16{\cdot}16{\cdot}C]$. After this process, there are three main components through which the image passes in a vision transformer architecture. They are patch and position embedding, a transformer encoder, and an MLP head for classification. These components are explained in further sections from the perspective of the ViT-Small model.

Patch and Position Embedding

The linearized patches are embedded using an embedding dimension d_{model} of 384, which converts the output dimensions to $[N, d_{model}]$. This is done by multiplying the linearized patches with a learnable embedding matrix of dimensions $[16 \cdot 16 \cdot C, d_{model}]$ and it is called a linear projection of flattened patches. A learnable class token parameter is added to remember class information during the training of the model. This token aims to create a distinct representation for each class in the classification task. This makes the dimensions of the output $[N + 1, d_{model}]$. Since transformers are permutation invariant (i.e., they do not inherently capture the order of the image patches), we need to add positional information to the patch embeddings to retain the spatial structure of the image. This is done by adding a positional encoding to each patch embedding. Positional encodings can be learned during training. The output after adding positional embeddings is given as input to the transformer encoder.

Transformer Encoder

The transformer encoder primarily contains the multi-head attention block and the multi-layer perceptron, along with two layer normalization layers [39].

Patch embeddings are split into three parts namely query Q, key K, and value V matrices by multiplying them with learnable weight matrices W_Q, W_K, and W_V with dimensions $[d_{model}, \frac{d_{model}}{12}]$ as in the Segformer architecture. These matrices are passed to the normalization layer. The normalization layer used by the encoder is

the layer norm. Used before the multi-head attention layer, this layer ensures that the query, key, and value matrices are normalized which helps in maintaining stable gradients and improving training. These matrices are passed on to the multi-head attention block.

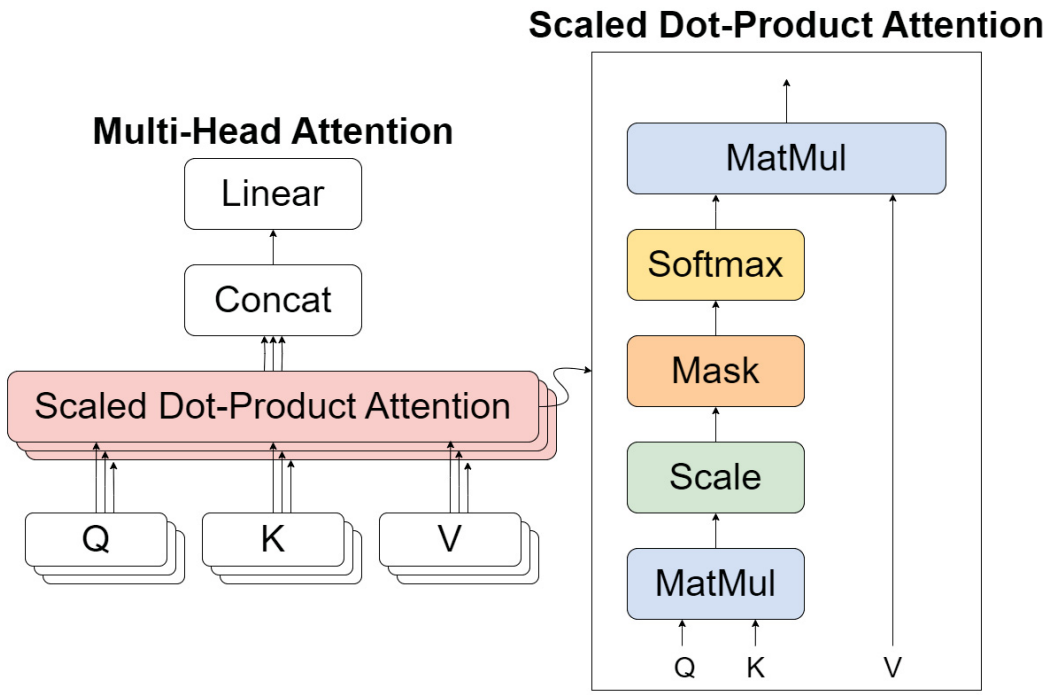


Figure 2.5: Depiction of the internal mechanism of the multi-head attention block (original image from [40]).

The transformer encoder consists of 12 heads that capture different contextual information about the images. The outputs from these heads are concatenated to form the output of the transformer encoder. Each head contains a self-attention block and a few multi-layer perceptrons. Self-attention is computed using Equation 2.5 where $h = 12$. The attention map identifies the correlation of every patch with every other patch. The attention maps are of dimensions $\left[N+1, \frac{d_{model}}{12}\right]$. Attention outputs from all the heads are concatenated along the channel dimension to form the output of the self-attention block shown in Equation 2.6. This concatenated output of dimensions $[N+1, d_{model}]$ is normalized again using layer norm and passed on to the MLP network inside the transformer block. The MLP consists of a few dense layers that aim to capture more details in the images by expanding and contracting

the inputs obtained from the multi-head self-attention block. These channel-mixing operations allow the model to integrate information across different dimensions more effectively. This is performed by four dense layers with d_{model}, d_{model}, $d_{model} \cdot 4$, d_{model} nodes in these dense layers successively. The class token is extracted from the output of the MLP inside the transformer block. This is done by selecting $[1, d_{model}]$ out of $[N+1, d_{model}]$ dimensions from the output of MLP inside the transformer block.

MLP Classifier

The class token is then passed through three dense layers containing 384, 128, and 6 nodes which successively decrease the dimensions of the input and bring them to the number of classes (i.e., 6) at the end. The output dimension is $[1, N_{cls}]$ where $N_{cls} = 6$.

2.3.2 ResNet50 Encoder With a Multilayer Perceptron

The ResNet50 encoder with a simple multi-layer perceptron classifier shown in Figure 2.6 is another classification model used in this study. From the explanation of the ResNet50 encoder in section 2.2.3, an input image of dimensions $[H, W, C]$ will produce an output of $\left[\frac{H}{32}, \frac{W}{32}, 2048\right]$. This output passes through an adaptive average pooling layer [41]. This pooling layer reduces the spatial dimensions of the image to a single number by taking an average across the height and width dimensions. This produces an output of 2048 dimensions. This is then passed through the multi-layer perceptron (MLP) for further processing. The MLP reduces the output dimensions from 2048 to 512 to 128 to 6 which is the number of classes in the classification task. After the MLP, a softmax operation is performed to extract the class probabilities and the class with the maximum probability is selected as the model prediction.

Two versions of this model were used. The first version uses random weights initialization and the second version uses pre-trained weights obtained in the same way as in section 2.2.3. The former is referred to as R50-MLP (**R**esNet**50** encoder with a **M**ulti-**L**ayer **P**erceptron), and the latter is referred to as R50B-MLP (**R**esNet**50** encoder pre-trained on **B**enthicNet dataset with a **M**ulti-**L**ayer **P**erceptron) going further.

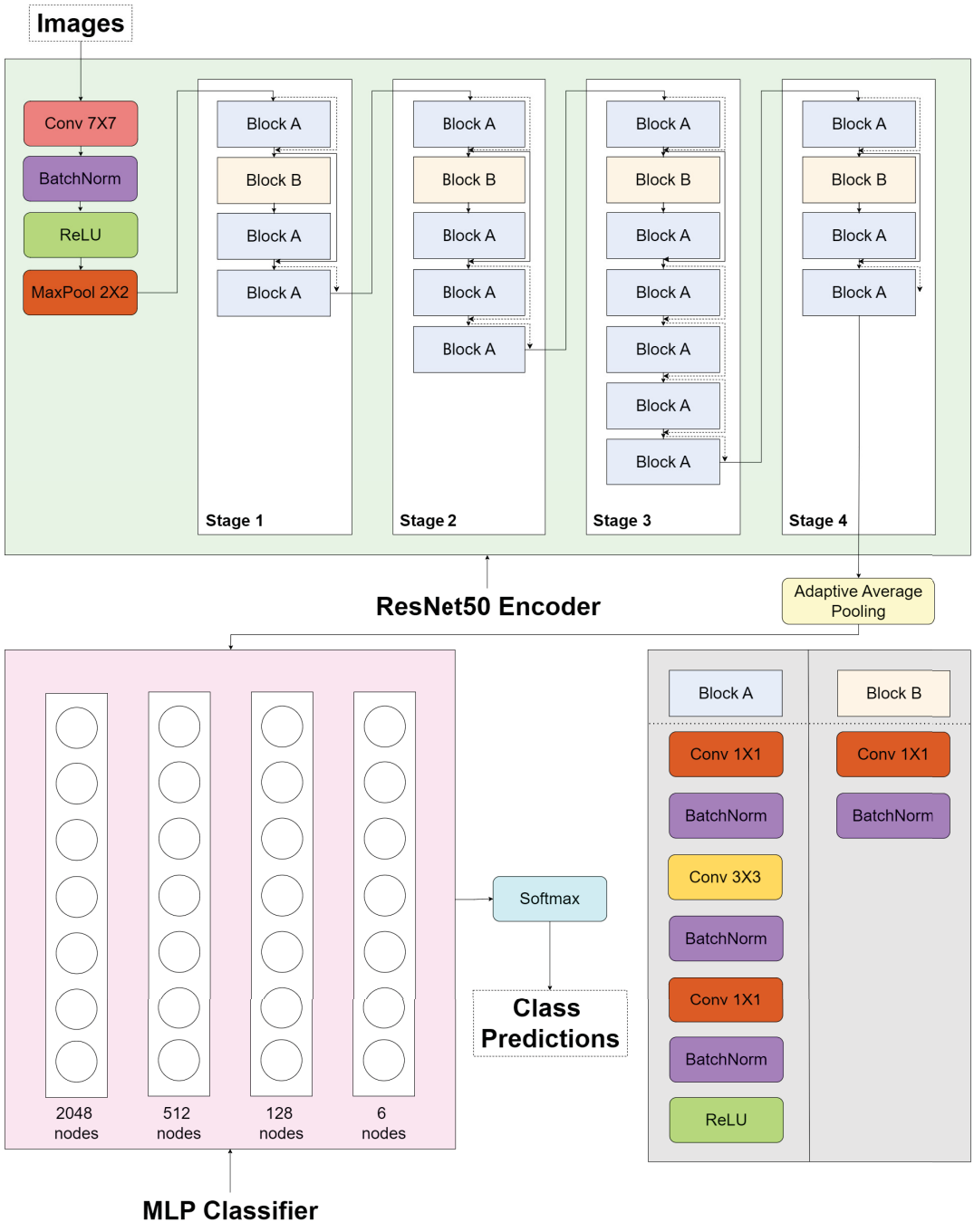


Figure 2.6: ResNet50 encoder with a multi-layer perceptron containing 6 classes depicting eelgrass cover in an image

2.3.3 VGG16 Encoder With a Multilayer Perceptron

The VGG16 encoder with a multi-layer perceptron classification model shown in Figure 2.7 combines the VGG16 encoder and a simple MLP to classify eelgrass percentages within an image. From the explanation of the VGG16 encoder in section 2.2.2, an input image of dimensions $[H, W, C]$ will produce an output of $[7, 7, 512]$. This output is flattened into a 25088-dimensional vector. This is then passed through the MLP for further processing. The MLP reduces the output dimensions from 25088 to 2048 to 128 to 6 which is the number of classes in the classification task. After the MLP, a softmax operation is performed to extract the class probabilities and the class with the maximum probability is selected as the model prediction.

Two versions of this model were used. The first version uses random weights initialization and the second version uses pre-trained weights obtained in the same way as in section 2.2.2. The former model is referred to as V16-MLP (**VGG16** encoder with a **M**ulti-**L**ayer **P**erceptron), and the latter model is referred to as V16B-MLP (**VGG16** encoder pre-trained on **B**enthicNet dataset with a **M**ulti-**L**ayer **P**erceptron) going further.

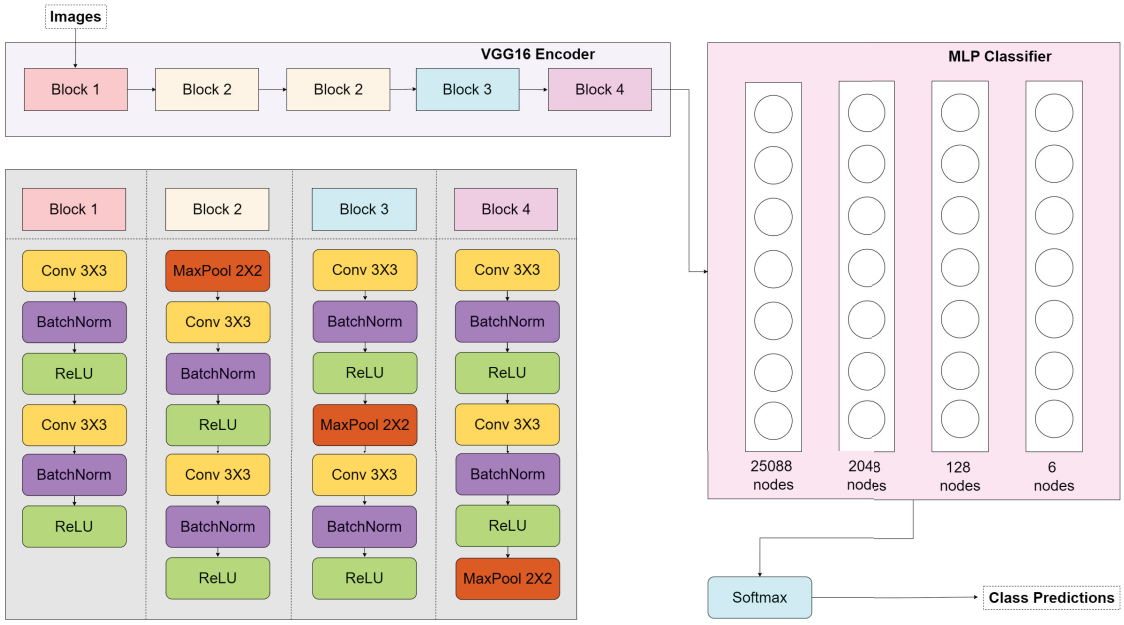


Figure 2.7: VGG16 encoder with a multi-layer perceptron containing 6 classes depicting eelgrass cover in an image.

2.4 Evaluation Metrics

This study uses human-estimated target segmentation masks along with human-estimated percentage covers to evaluate the performance of image segmentation models. The classification models are evaluated only on the basis of the human-estimated percentage covers. The target segmentation masks and the model predictions are flattened out for computing the pixel-wise F1 scores (originally introduced in [42]) for image segmentation. Pixel-wise F1 scores have been used popularly by researchers for image segmentation [43, 44]. The human-estimated percentage covers are compared with the model-predicted classes for computing F1 scores for image classification. By considering both the precision and recall of the results, the F1 score pays attention to false positives and negatives. Another metric for evaluating image segmentation performance is the IoU score (originally termed as "...ratio of verification" in [45], also known as the Jaccard Index). It penalizes false positives and false negatives more heavily than the dice score which is equivalent to F1 score for segmentation (originally developed by botanists Lee Raymond Dice [46], and Thorvald Sørensen [47]), making it a stricter metric. It is widely used in various segmentation challenges and competitions [48], providing a standard benchmark for comparing different models. The segmentation masks generated from the segmentation model are converted to percentage cover estimates by taking a sum of all pixels corresponding to eelgrass and compared with the human-estimated covers to calculate the R^2 score (introduced in [49]). This shows whether the model explains the variability in the data in terms of predicting the cover estimates. The dataset used does not encompass eelgrass distributions on a global scale. However, the validation data can be utilized as a proxy for this broader distribution. Consequently, accuracy is employed as a metric for evaluation to monitor the overall efficiency of the models.

Overall, the pixel-wise F1 score and IoU score assess the effectiveness of segmentation models on a per-pixel basis, while the R^2 score evaluates performance at the per-image level. For classification models, macro F1 scores and traditional accuracy scores without class balancing are used.

2.5 Superpixels

Superpixels introduced in [51] are an image processing technique used to group pixels into meaningful regions, which facilitates the processing of images or video frames by reducing the complexity of the data. Instead of dealing with individual pixels, superpixels aggregate pixels into larger, more informative regions. Superpixels are generated using various algorithms that aim to partition an image into visually coherent regions. Common algorithms include SLIC (Simple Linear Iterative Clustering) [52], SEEDS [53], and Felzenszwalb-Huttenlocher's algorithm [54].

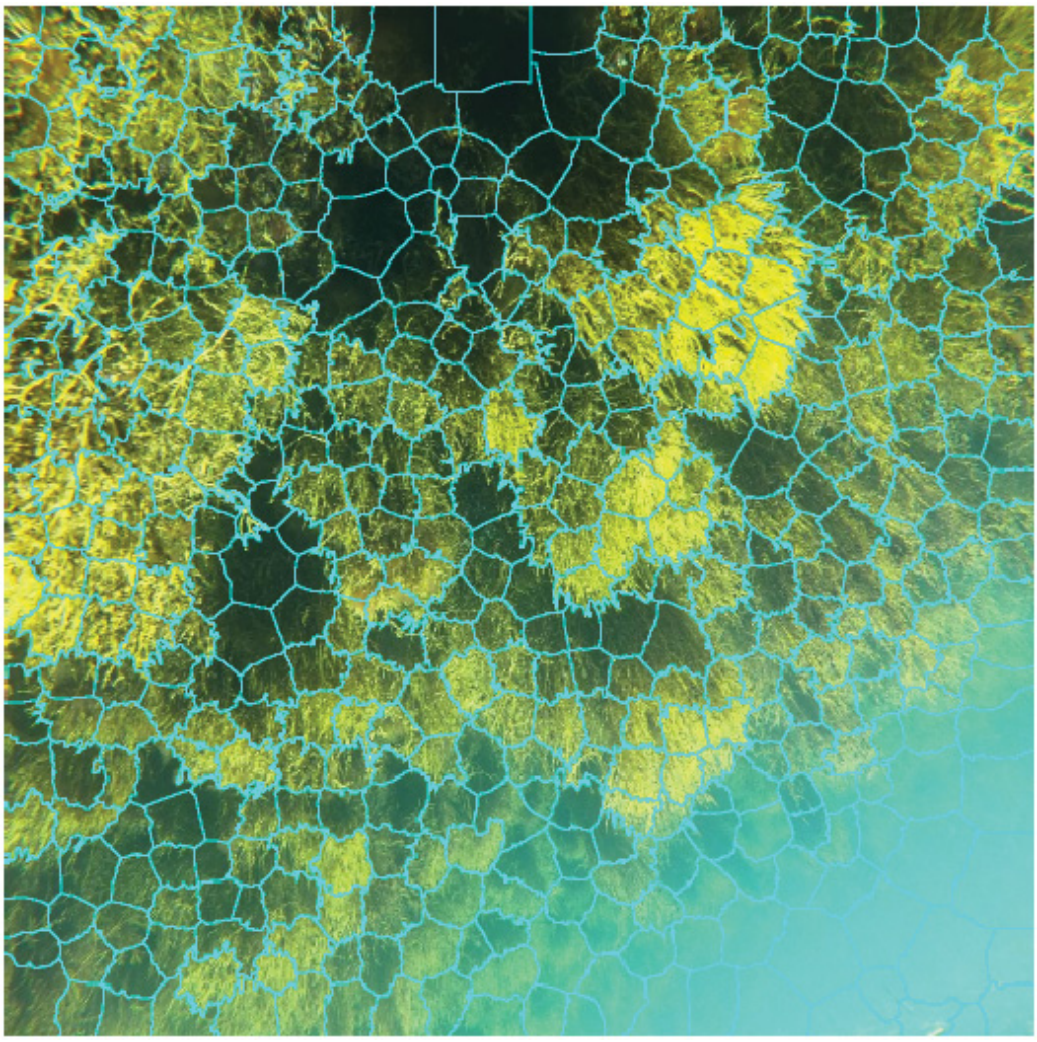


Figure 2.8: An example of superpixels generation, where pixels are grouped into larger informative regions to make processing faster (code adapted from [50]).

Chapter 3

Related Work

This section explores some research works that have been done around obtaining percentage cover estimates of objects from images. It includes analyzing image segmentation and image classification methods on images from different locations and taken using different cameras.

In 2003, a study [55] used the OTSU intraclass variance [56] to compute the segmentation masks of eelgrass from ocean floor images. The images used in this study were collected from ROVs and the background was easily distinguishable from eelgrass. OTSU is a global thresholding technique, and it might struggle to separate eelgrass from complex backgrounds accurately, which is the case with the dataset used in our study. To verify the same, this method was tried on the dataset used in this study and it performed very poorly because the images contained eelgrass blades having different brightness levels. The predictions generated by this method for eelgrass images are shown in figures A.1 and A.2.

In a 2018 study [57], researchers employed a novel method to segment eelgrass in underwater images. This technique involved calculating patches of equal size from superpixels of video frames captured around the Island of Murter in Croatia. This was followed by feature extraction using Convolutional Neural Networks (CNNs). This approach achieved a pixel accuracy of 0.94 and an Intersection over Union (IoU) score of 0.81, indicating robust performance in segmenting eelgrass from underwater imagery. Building on this work, a 2019 study [58] improved the performance using the same dataset by training a Fully Convolutional Network (FCN) and a DeepLabV3+ network [59]. The DeepLabV3+ network yielded the best results, achieving a mean IoU score of 0.88 and a pixel accuracy of 0.96. These improvements underscore the efficacy of advanced neural network architectures in enhancing segmentation accuracy. In these studies, data was collected from a single location and using a single camera, which harms the model's generalizability. Our study captures data from multiple

locations using multiple cameras aiming to improve model generalizability.

A research conducted by Mohammed Asaad Ghazal et. al. [60] was conducted in the city of Louisville in the state of Kentucky in the United States in 2019. CNNs were used for vegetation cover estimates in this study. A deep 3-D convolutional neural network was used, which was given images from four different cameras (RGB, NIR, NDVI, and red) as input. The segmentation masks were compared with ground truth masks and this model gave a mean dice score of 0.97 for the different image types. This study benefited from images where vegetation was easily distinguishable due to its distinct colour. In contrast, eelgrass cover estimation poses a challenge due to the presence of non-eelgrass objects with similar colours. The code corresponding to the CNN used here was not found, hence we constructed a CNN for segmentation and tried it for eelgrass segmentation. This yielded a dice score of 0.81 ± 0.01 which shows that a more complex model would be beneficial to this study.

Chapter 4

Methodology

4.1 Data Collection

The dataset used for training and validation for this study consists of 2257 images concentrated along the Atlantic coast of the Canadian province of Nova Scotia through diver and kayak surveys. This is referred to as the primary dataset in further sections. The images are collected from Halifax and several smaller coastal communities or towns south of Halifax along the shoreline. Additionally, a cluster of data was collected from around Cape Breton Island, specifically near Sydney. The dataset also includes samples from the Charlottetown area of Prince Edward Island. These diverse locations provide a comprehensive representation of eelgrass habitats in these regions. The dataset consists of images taken from different depths, angles, brightness, cameras, directions and resolutions. This ensures that the models are trained on a vast domain of images, and it ensures model generalizability to unseen images. The labelling of the primary dataset is required to generate segmentation masks for the segmentation models and percentage cover estimates for the classification models to train on. The masks were generated by using a web service called **B**io-**I**mage **I**ndexing and **G**raphical **L**abelling **E**nvironment (BIIGLE) [61]. This tool is specifically designed to annotate benthic fauna in marine image datasets. Along with the segmentation masks, human-estimated percentage covers and quality measurements are available. The percentage covers are between a range of 0 to 100% indicating the quantity of eelgrass present in the given image. The quality measurements indicate the clarity of these images where quality 3 (1330 images) indicates completely clear images, quality 2 (675 images) indicates partially clear images where part of the image is clear (this part may or may not contain eelgrass), and a quality of 1 (252 images) indicates completely hazy/unclear images where it is very difficult to make out the existence of eelgrass if any. The distribution of qualities is shown in Figure 4.1. Volunteers from the Dalhousie University Marine Biology Department

generated the segmentation masks, human-estimated percentage covers, and quality measurements.

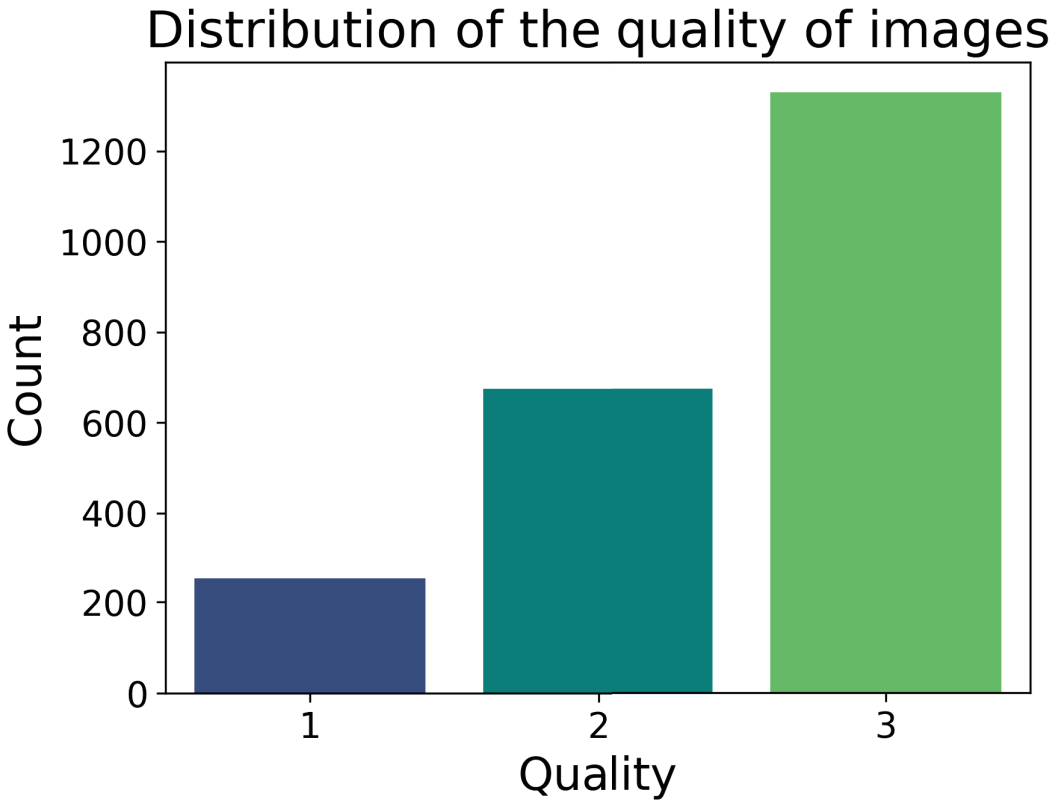


Figure 4.1: Quality distribution of the images used for training and validation of image segmentation and classification models. Quality 1 indicates hazy/unclear images where delineating eelgrass is difficult, quality 2 indicates partially clear images where part of the image is hazy/unclear, and quality 3 images are completely clear.

Two datasets were used to test the models developed further. The first dataset (Medway-test dataset) comprises 60 quadrat images from the Medway River basin via snorkelling [62]. This dataset corresponds to same-domain images because they are collected from locations nearby to the training data collection sites, ensuring similar characteristics. The second dataset (SGS-test) is taken from the Seagrass Spotter website which was developed as a part of Project Seagrass analyzed in [63]. This dataset comprises 60 images taken from different locations around the world. This dataset corresponds to different-domain images, because they are collected from randomly distributed locations, providing a diverse set of characteristics. Both these

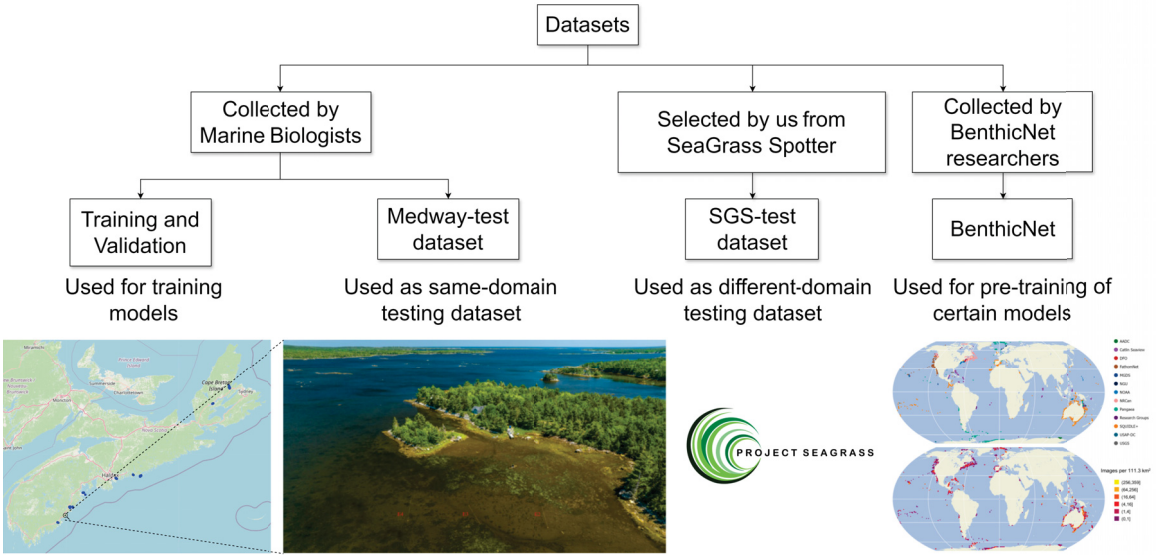


Figure 4.2: Different datasets selected for this study.

datasets contain images taken from a variety of depths, camera angles, brightness, and quality to test the robustness of models.

4.1.1 BenthicNet Dataset

BenthicNet [34] is a compilation of ocean floor images that are created to "support the training and evaluation of large-scale image recognition models" as mentioned in [34]. The total number of images in BenthicNet amounts to 11.4 million, out of which 188 thousand images are labelled as corresponding to different ocean elements. A subset of 1.3M unlabelled images was used for the training of **S**elf-**S**upervised **L**earning (SSL) models. The encoders from these SSL models were then used by certain models in this study. These models were trained using SSL (pretext task — explained briefly in 2.1) and used for the downstream tasks of segmentation and classification.

4.2 Data Pre-processing

As the images are collected from different locations, they do not have uniform sizes. For training the classification and segmentation models, they need to be brought to a uniform size. All the images lie in the range of 980–1100 pixels in height and width. Hence, initially, they are converted to a size of $[1024, 1024]$ and the masks are further converted to grayscale. This is done using the 'nearest' interpolation mode

available from the torchvision/transforms library in the vision repository available on GitHub [64]. Nearest neighbour interpolation for high-resolution image interpolation is a method for resizing images [65]. It selects the value of the nearest pixel to the target location in the output image and copies the nearest pixel's value to the new location. It does not create new pixel values but replicates the nearest pixel values.

Above mentioned pre-processing is performed for both image classification and segmentation models. The following sections describe the pre-processing done individually to train image segmentation and image classification models.

4.2.1 Image Segmentation

This segment describes the two types of labelling methods tried for generating segmentation masks. One of them is automatic labelling and the other is manual labelling. Due to some flaws in the automatic labelling, the segmentation masks generated by manual labelling were used in the experiments performed.

Automatic Labelling

The generation of segmentation masks to train the models is a time-consuming task if the labelling is done manually. Hence, an automatic labelling method provided by BIIGLE [61] was tried in this study. This method is called the **M**achine-learning **A**ssisted **I**mage **A**nnotation method (MAIA) introduced in [66]. It is used to annotate huge image datasets faster than doing it manually. The four stages of MAIA are described briefly below.

The first stage tries to capture objects that are distinguishable from a fairly uniform background. These objects are then given as input to the second stage. This stage contains human intervention where only the objects important to the task are kept and the remaining objects are filtered out. The third stage involves an auto-encoder that learns features about the filtered objects. Finally, in the fourth stage, the objects are again filtered manually and transformed into annotations.

This method produced annotations concentrated on regions of interest other than eelgrass, even after the manual refinement process. Also, it produced annotations with a certain well-defined shape and because eelgrass does not have a well-defined shape (such as a circle as shown in Figure 4.3), this method was not used for labelling

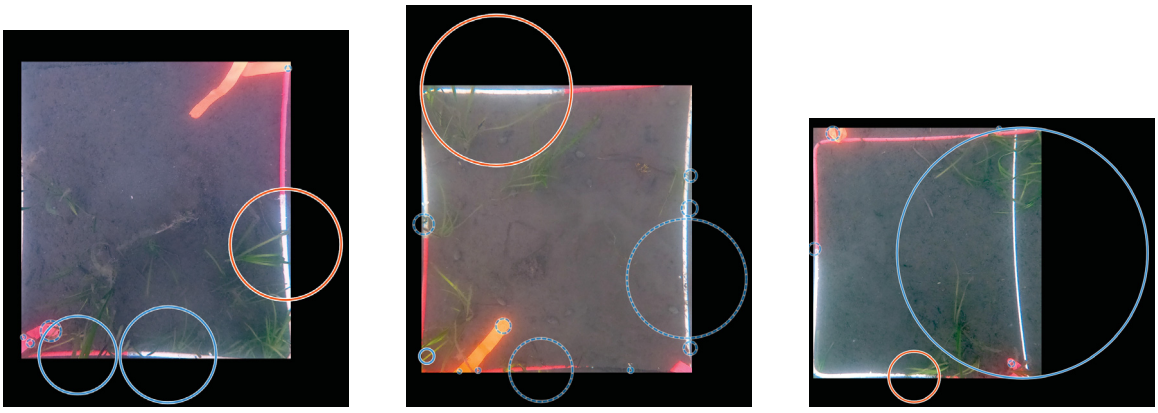


Figure 4.3: Three example annotations generated by MAIA from BIIGLE. These annotations were discarded and the human annotations were used in further experiments.

the images. Instead, the human-annotated segmentation masks generated through manual labelling were preferred.

Manual Labelling

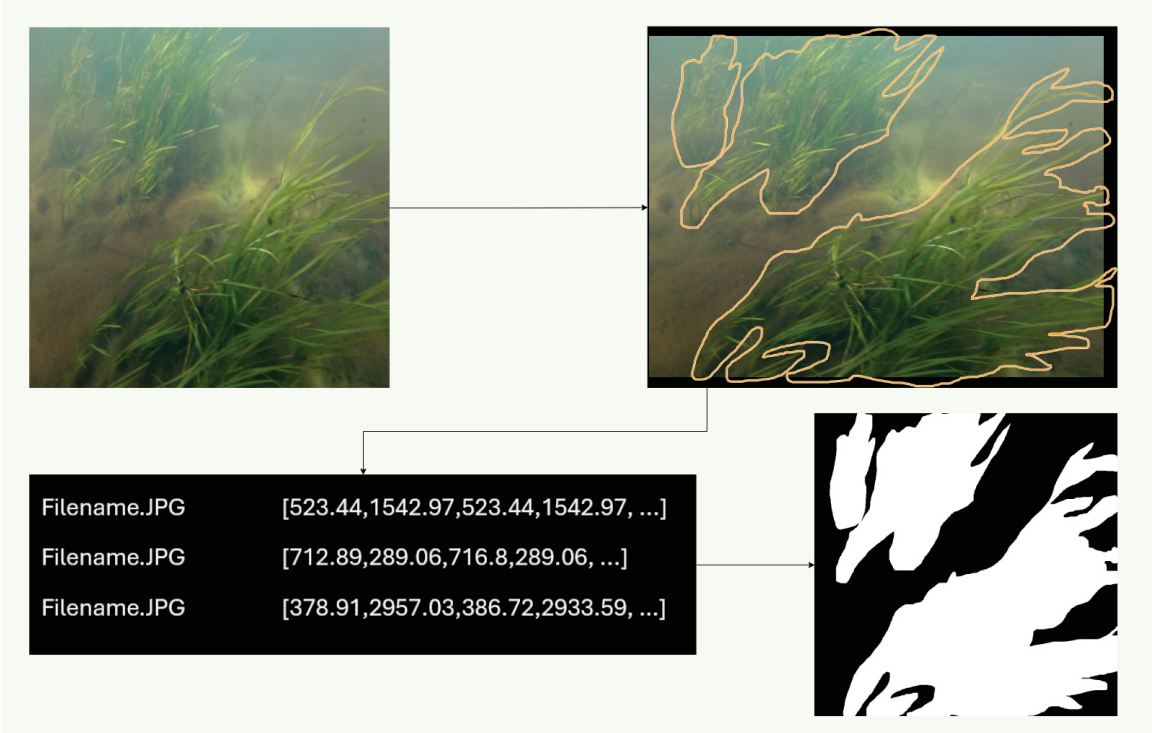


Figure 4.4: Manual annotation of eelgrass using the polygon tool from BIIGLE.

For manual labelling, the polygon generation tool from BIIGLE [61] was used. A polygon consists of three or more coordinates enclosing a specific area on the image. Multiple polygons can be drawn on the image where each polygon represents a patch of eelgrass. These polygons are then converted to annotation arrays depicting the coordinates of these polygons within the image. These annotation arrays are then converted to segmentation masks using a simple Python code described in the GitHub repository in [67].

4.2.2 Image Classification

The images and their human-estimated percentage covers are used for training the classification models. The percentage covers originally represent numbers between 0 and 100 and they are converted to 6 different classes by separating them in ranges. Class 1 indicates 0% eelgrass cover in the given image, class 2 indicates 1–20% eelgrass cover, class 3 indicates 20–40% cover and so on where class 6 indicates 80–100% cover. Class 1 is the dominant class with 1018 images, classes 2 to 6 contain 189, 200, 215, 219, and 416 images respectively as shown in Figure 4.5.

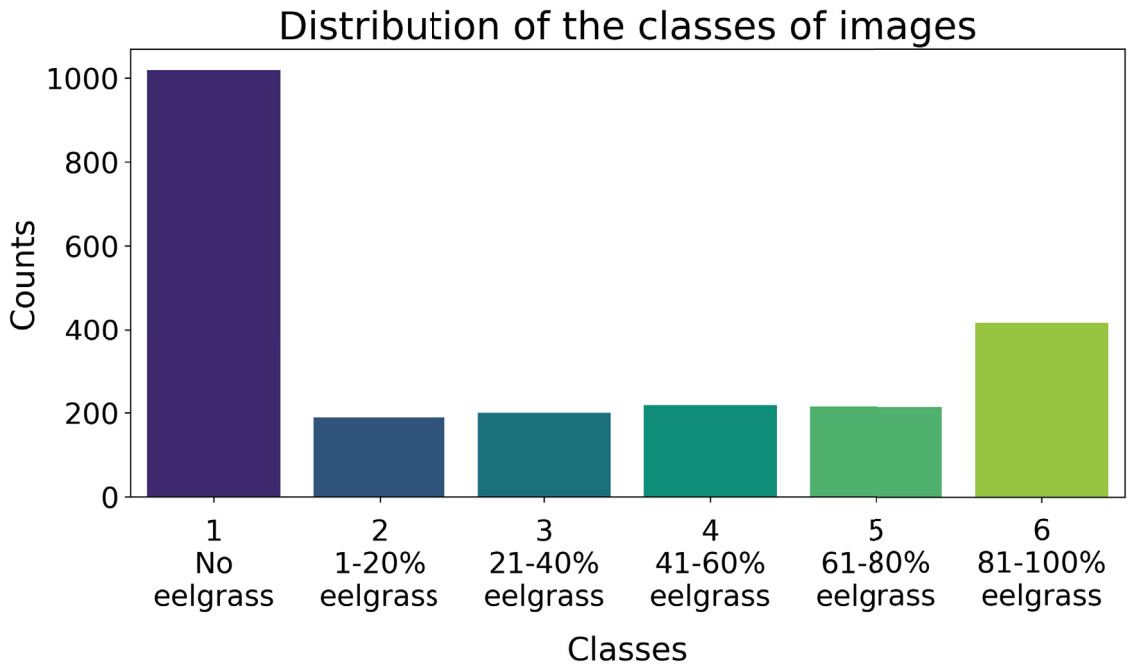


Figure 4.5: Class distributions for the image classification task.

4.3 Model Configurations

4.3.1 Loss Function

For image segmentation and image classification, we used a cross-entropy loss function with a minor difference in how it is used.

For segmentation, cross-entropy loss is applied on a per-pixel basis, treating each pixel as a separate multi-class classification problem. The overall loss is the average cross-entropy loss over all pixels. The cross-entropy loss for the entire image is described in Equation 4.1. In this equation, $N = 1024 \cdot 1024$ is the number of pixels in the image, $C = 2$ is the number of classes, y_{ij} is the true label for the j^{th} class of the i^{th} pixel, and $\hat{y}_{ij}$ is the predicted probability for the j^{th} class of the i^{th} pixel.

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij} \cdot log(\hat{y}_{ij}) \tag{4.1}$$

Each pixel is assigned a class label, and the model's output is a probability distribution over both classes for each pixel. Cross-entropy loss penalizes deviations from the true pixel-wise class distributions. In the eelgrass dataset, class imbalance is common, where the background class is over-represented. Cross-entropy loss naturally handles class imbalance by emphasizing the contribution of less frequent classes to the loss function, ensuring that they are not overwhelmed by more dominant classes.

For classification, cross-entropy loss measures the dissimilarity between the predicted probability distribution P, a 6-dimensional vector containing predicted probabilities for 6 classes, and the true probability distribution y, a one-hot encoded vector. The cross-entropy loss is defined in Equation 4.2, where $C = 6$ is the number of classes, y_i and P_i are the true and predicted probability distributions respectively.

$$L = -\sum_{i=1}^{C} y_i \cdot log(P_i) \tag{4.2}$$

In classification, every image is assigned a label, and the model outputs are probability distributions over all classes for an image. Out of the 6 classes represented in this research, the classes where there is no eelgrass present is a dominant class, and cross-entropy loss handles class imbalance effectively.

4.3.2 Optimizer

Stochastic Gradient Descent (SGD) is used here as an optimization algorithm to minimize the loss function by iteratively updating the model parameters. SGD is often used to train deep learning models such as Convolutional Neural Networks (CNNs) and transformer-based models like Segformer. It tends to generalize better and provides more stable training when compared to the Adam optimizer, especially when dealing with small datasets like the eelgrass dataset or models prone to overfitting.

Root Mean Square Propagation (RMSProp) is an adaptive learning rate optimization algorithm widely used for image classification tasks. It addresses some of the challenges encountered with traditional gradient descent and improves convergence by adjusting the learning rate for each parameter dynamically.

4.3.3 Learning Rate

The learning rate controls the size of the steps the algorithm takes when adjusting the model's parameters in response to the gradients computed during backpropagation. If the learning rate is too high, the algorithm may overshoot the optimal solution, causing the loss function to diverge or oscillate rather than converge to a minimum. If the learning rate is too low, the algorithm makes slow progress towards the minimum, leading to a long training process and potentially getting stuck in local minima. Learning rates used for image segmentation and image classification are shown in tables 4.1 and 4.2 respectively.

Models \ Experiments	1	2	3	4	5
Segformer	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}
V16I-UNet	10^{-6}	10^{-4}	10^{-4}	10^{-4}	10^{-4}
V16B-UNet	10^{-2}	10^{-2}	10^{-2}	10^{-2}	10^{-2}
R50B-CNN	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}

Table 4.1: Learning rates used for the 5 experiments in image segmentation. The five experiments shown here are described in section 4.4.1.

Models	Learning Rates
VS-MLP	10^{-5}
VSB-MLP	10^{-5}
VB-MLP	10^{-5}
VBB-MLP	10^{-5}
R50-MLP	10^{-6}
R50B-MLP	10^{-6}
V16-MLP	10^{-6}
V16B-MLP	10^{-6}

Table 4.2: Learning rates used for the eight models in image classification.

4.3.4 Epochs

An epoch is a full iteration over the entire dataset, meaning that every training example has been seen once during that epoch. Repeated exposure to the training data allows the model to gradually reduce the training loss and improve its performance. Training for too many epochs can lead to overfitting, where the model performs well on the training data but poorly on unseen validation or test data. The models were tried on 50 and 100 epochs, but the performance did not improve significantly after 50 epochs, hence, all the models were trained until 50 epochs.

4.3.5 Cross-Validation

K-fold cross-validation is a robust method for assessing the performance of a machine-learning model. Cross-validation allows for evaluating the robustness of a model across different data splits, providing insights into its stability and reliability. If a model consistently performs well across multiple cross-validation folds, it is likely to be more robust and less sensitive to variations in the training data. It also provides a reliable estimate of how well a model will generalize to unseen data. The value of K is chosen to be 5 for further experiments. For performing 5-fold cross-validation, the dataset is first divided into 5 nearly equal-sized subsets or folds. The model is trained 5 times, each time using a different fold as the validation set and the remaining 4 folds as the training set. The performance metrics are averaged over the 5 folds to produce a performance estimate mean and a standard deviation. This gives a more reliable measure of the model's ability to generalize to unseen data. Stratified K-fold cross-validation is a variation of the standard K-fold cross-validation technique,

which ensures that each fold of the dataset has the same proportion of instances for each class label as the original dataset. This is particularly useful for imbalanced datasets where some classes are underrepresented. For performing stratified 5-fold cross-validation, the dataset is divided into 5 folds ensuring that each fold has approximately the same percentage of samples of each target class as the complete set. The rest of the steps are the same as in 5-fold cross-validation without stratification. All segmentation and classification experiments were performed using unstratified cross-validation. Additionally, stratified cross-validation was tried for classification experiments.

4.3.6 Batch Size

The batch size defines the number of training examples used in one epoch to update the model's parameters. Small batch size provides noisier estimates of the gradient, which can help in escaping local minima and finding a better overall minimum. However, it can lead to unstable convergence. The large batch size provides more accurate estimates of the gradient, leading to more stable and smoother convergence. However, it might get stuck in local minima and converge more slowly. A batch size of 4 was used for R50B-CNN and a batch size of 2 was used for the rest of the segmentation models. For classification models, a batch size of 16 was used for all the models.

4.4 Experiments With Image Segmentation

4.4.1 Data Preparation

Initially, the images and masks are converted to a uniform size of $[1024, 1024]$, and the masks are converted to grayscale. The data preparation after this is different for 5 experiments. These experiments pre-process data in different ways. Deep networks are end-to-end networks where pre-processing is usually not required, but in a relatively small dataset, pre-processing the data before training is expected to aid the models in producing better segmentation masks. Table 4.1 refers to the learning rates used for these experiments.

Experiment 1 uses the training and validation dataset of 2257 images and corresponding human-annotated segmentation masks. No pre-processing was performed in this

experiment. Experiment 2 uses a sharpening filter on the dataset of 2257 images and uses those images with corresponding human-annotated segmentation masks. The sharpening of images was performed using a simple convolutional filter on the original images to aid the model in identifying faint eelgrass patches from the images. The convolutional filter used for this process is shown in Equation 4.3.

$$Sharpening\_Filter = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{4.3}$$

Experiment 3 uses the dataset of 2257 images and a corresponding modified version of segmentation masks. The modified version of the segmentation masks was constructed by following three steps . The first step involves applying a canny edge detection filter introduced in [68] implemented using OpenCV [69], with a threshold of 8, an aperture size of 3, and an L2 gradient. This is done because human-annotated masks do not capture every single blade of eelgrass and edge-detection can aid with the same. Different settings of thresholds, aperture sizes, and L2 gradient were tried and the ones mentioned above provided the best segmentation masks. The second step involves passing the edges from step 1 to a connected components algorithm (implemented as connectedComponentsWithStats in [69]) which takes 8 nearby pixels and clumps them together. This ensures that the pixels inside the eelgrass blades are labelled as eelgrass because the edge detection only picks up on the edges of the blades and the pixels inside the blades are not labelled as eelgrass. In the final step, the connected components image is overlayed with the human-annotated masks. This is done because edge-detection will detect every single object in the image and we need to classify only eelgrass. Hence, applying a binary 'and' to these connected components will ensure that the masks contain edges corresponding to eelgrass only. Experiment 4 uses a sharpening filter on the dataset of 2257 images as in Experiment 2 and uses them with a corresponding modified version of segmentation masks as in Experiment 3. Experiment 5 uses a reduced version of the dataset by removing low-quality images which amounts to a total of 2008 images. These images with corresponding human-annotated masks are used to train the model.

The PyTorch library [70] was used for all the experiments in this research. The pairs of images and masks are then loaded into a PyTorch dataset object by converting

them to PyTorch tensors. The datasets have a seed value of 10 which is uniform throughout all the experiments. This ensures that the dataset is collected from the respective folders in the same order every time it is accessed to ensure reproducibility. After this, for every fold, the dataset is split into a training and a validation dataset using the KFold function provided by the scikit-learn library [71]. This function also uses a seed value of 10 to ensure that the dataset is split the same way as it was during training, which ensures reproducibility. For experiment 5, the low-quality images were removed first, followed by using the remaining images for training and validation. The split datasets are then loaded into a PyTorch dataloader object with the corresponding batch size. The training and validation dataloaders are then passed through a trainer function which learns the masks using a standard training loop.

4.4.2 Training and Loss Calculation

Inside the training loop, the images and corresponding target segmentation masks are loaded from the training dataloader, and the model is set to training mode. When the model is set to training mode, PyTorch enables features like dropout and batch normalization, typically used during training but not during inference. The training set of images is passed through the model and a set of model predictions is generated. The model predictions are compared with the target segmentation masks to calculate the losses. The optimizer is then used to update the weights of the model based on the batch gradients. This process repeats for every batch in every epoch. The training and validation losses are calculated after the training is completed in a particular epoch. This ensures that the loss computed over the entire epoch is less affected by noise or fluctuations compared to losses computed after processing individual batches. Once a fold is completed, the model weights are saved.

4.4.3 Evaluation

The evaluation of segmentation models is performed using two constructs. The first construct evaluates validation data from the 5 folds in cross-validation using pixel-wise F1 scores, pixel-wise accuracy scores, R^2 scores, and IoU scores. The second construct uses two test datasets to observe model performance on unseen data using R^2 scores.

The segmentation masks from the model outputs are compared with the target segmentation masks to produce an IoU score for validation data of each fold. These masks are flattened out to calculate the pixel-wise F1 scores for validation data for each of the 5 folds. Then, the percentage cover predictions are extracted from segmentation masks by summing up all the pixels representing eelgrass and dividing by the total number of pixels in the images. For calculating the R^2 scores, initially, a linear regression model from scikit-learn (defined as LinearRegression in [71]) is fit to the human percentage cover estimates (true labels) and model predictions. The true labels are then passed to this regression model to obtain the predicted values. These predicted values and true labels are then used to compute the R^2 score. These experiments provide a wide variety of evaluations which help us decide which model is suitable for predicting eelgrass percentage cover. The model is then tested on the two test datasets. The mean and standard deviation of the predictions from 5 models obtained from cross-validation are recorded. The standard deviations give an estimate of the uncertainty of the models.

Model Predictions / **Original Pixels**	**All Same**	**One Different**	**Two Different**
0	0	0.2	0.4
1	1	0.8	0.6

Table 4.3: Weighted pixel values used for calculating percentage cover estimations based on the outputs of the 5 models obtained from cross-validation. The rows represent the pixel values obtained from the majority vote between these models. The columns show the agreement between these models.

Another method was tried to improve the percentage cover estimates based on the results obtained from cross-validation. Originally, to calculate the percentage cover estimate of eelgrass in an image, 5 segmentation masks generated by 5 models in cross-validation were examined. The pixel values in the final segmentation mask were generated by taking the majority vote between these 5 models. The sum of all the pixels containing eelgrass in this resultant segmentation mask was then used to determine the percentage cover estimate of eelgrass in the entire image.

In another approach, the 5 segmentation masks generated by the 5 models obtained from cross-validation were examined as in the earlier approach. However, the

pixel values in the final segmentation mask were not generated by taking a majority vote between these 5 models, but by weighing the pixels according to model outputs as shown in Table 4.3. The table shows that if all the models provide the same output for a pixel as in column 2, the weighted pixel values do not change in the resultant segmentation mask. Furthermore, if one or two models provide a different output than the other models, the resultant pixel values change as shown in columns 3 and 4. These pixel values in the resultant segmentation mask are then summed up to calculate the percentage cover estimate of eelgrass.

The predictions using this approach were compared with the predictions using the original approach using R^2 scores. An R^2 score of 0.99 was obtained. This indicates that pixel-weighting would cause an unnecessary overhead while computing percentage cover estimates and hence, was not used in further calculations.

4.4.4 Inference

The best-performing model was V16B-UNet. The code used for inference from this model is posted on GitHub [72]. This code takes in a list of images from a user-specified folder and provides two outputs. The first output is a CSV file that contains the model-predicted percentage covers along with the filenames, and the second output is a user-specified folder that contains the predicted binary segmentation masks. The flowchart for obtaining the percentage cover estimates is shown in the README.md file of the GitHub repository [72]. This code takes 71 and 84 seconds to run inference on Medway-test and SGS-test datasets respectively using an NVIDIA GeForce RTX 3060 laptop GPU.

4.5 Experiments With Image Classification

In image classification, the model is trained to distinguish between 6 classes. Each image is entirely classified into one of these classes instead of pixel-wise identification in image segmentation. The model produces a label as output. All the experiments in image classification follow the steps listed below.

4.5.1 Data Preparation

Initially, the images are converted to a uniform size of [224, 224]. This is contrary to the images used in image segmentation which used images of size [1024, 1024]. This is because the vision transformers used in image classification are pre-trained on the BenthicNet dataset, and the images used for the pre-training were [224, 224] sized images. Hence, using this size ensures that features learned by the pre-trained encoders do not get distorted. The human-estimated percentage covers are divided into 5 classes between 0 and 100. Class 1 represents images containing 0% eelgrass, class 2 represents images containing 1–20% eelgrass and in this way, classes 3, 4, 5, and 6 represent 21–40%, 41–60%, 61–80%, and 81–100% eelgrass. The pairs of images and percentage cover classes are then loaded into a PyTorch dataset object by converting them to Pytorch tensors. The datasets have a seed value of 10 which is uniform throughout all the experiments to ensure reproducibility. After this, for every fold, the dataset is split into training and validation datasets. The split is performed using K-fold cross-validation. Stratified K-fold cross-validation was tried where the split ensures that every class has a nearly equal number of samples. This split did not work as expected because of the class imbalances. Class 1 contained around 800 samples out of the 1800 samples in the training dataset, while the other classes contained 150–200 samples. Using under-sampling to address class imbalances works in many cases when there is enough data available for training, but in a small dataset, it reduces the number of available examples even further, some of which might be valuable samples, due to which the model misses necessary patterns due to limited data. Over-sampling techniques lead to overfitting in small datasets, causing the model to memorize a few available examples instead of learning generalizable patterns. Methods like SMOTE for over-sampling can inadvertently amplify noise in the minority class, leading to a model that learns from noisy data. The split datasets are then loaded into a PyTorch dataloader object with the corresponding batch size. The training and validation loaders are then passed through a trainer function which learns the classes using a standard training loop.

4.5.2 Training and Loss Calculation

Inside the training loop, the images and corresponding target class vectors are loaded from the training dataloader, and the model is set to training mode. For the first experiment, all the models were initialized using random weights, and for the second experiment, BenthicNet weights were used for initialization. The training set of images is passed through the model and a set of model predictions is generated. The model predictions are compared with the target classes to calculate the losses. The optimizer is then used to update the weights of the model based on the batch gradients. This process repeats for every batch in every epoch. The training and validation losses are calculated after the training is completed in a particular epoch. Once a fold is completed, the model weights are saved.

4.5.3 Evaluation

For evaluation, the model predictions from validation data of the 5 folds are compared with the target human-estimated percentage covers using accuracy, and F1 score shown in Figure 5.8. Additionally, the confusion matrices are generated to analyze which classes the model does better on. These experiments provide evaluations that help us decide which model is suitable for predicting eelgrass percentage cover. For testing, the model generates predictions on the two test datasets. The human-estimated percentage covers are converted into discrete classes and compared with the model predictions.

4.5.4 Inference

The best-performing model was VBB-MLP. The code used for inference from this model is posted on GitHub [73]. This code takes in a list of images from a user-specified folder and provides a CSV file that contains the model-predicted percentage cover classes along with the filenames. The flowchart for obtaining the percentage cover estimates is shown in the README.md file of the GitHub repository [73]. This code takes 21 and 19 seconds to run inference on the Medway-test and SGS-test datasets respectively using an NVIDIA GeForce RTX 3060 laptop GPU.

Chapter 5

Results

5.1 Image Segmentation

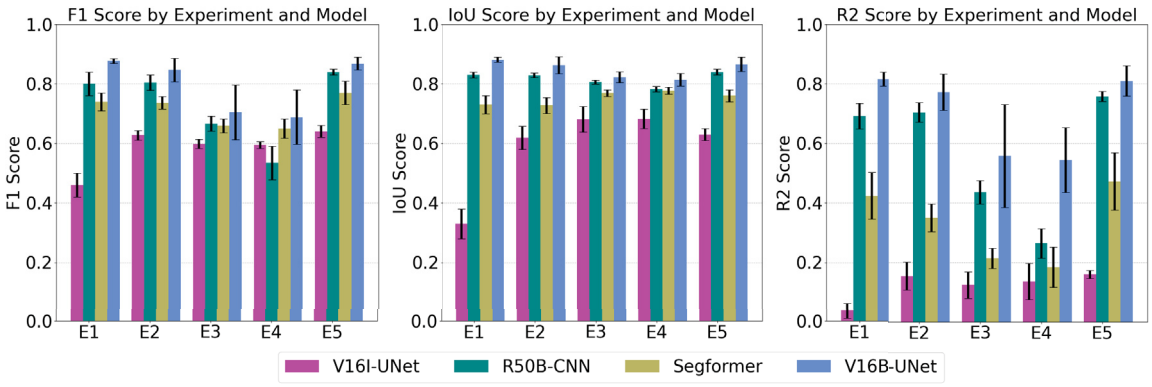


Figure 5.1: Comparison between F1 scores (left), IoU scores (middle), and R^2 scores of four models for five experiments across a 5-fold cross-validation is shown in this diagram. Experiment 1 (E1) refers to using unaltered images and masks for training. Experiment 2 (E2) uses sharpened images and unaltered masks. Experiment 3 (E3) uses unaltered images and modified masks. Experiment 4 (E4) uses sharpened images and modified masks. Experiment 5 (E5) uses a quality-refined dataset with unaltered images and masks. Details of the experiments are enlisted in section 4.4.1. The tabular representation for this image is provided in Table 5.1.

The segmentation results in Figure 5.1 show that all the models except the V16I-UNet model perform nearly the same. As one standard deviation represents around 66% samples assuming a Gaussian distribution, we cannot say that one of these models is the best model based on the validation results. The V16B-UNet performs better than the V16I-UNet which reinforces the idea that transfer learning on a similar dataset (with more comprehensive ocean floor images) improves model performance on unseen data.

Comparing Experiments 1 and 2 in Figure 5.1, we observe that when the images are sharpened before training, the performance of V16B-UNet (blue bars) remains similar and V16I-UNet (pink bars) improves. Models pre-trained on a similar dataset

Models	Experiments				
	E1	**E2**	**E3**	**E4**	**E5**
			F1 scores		
V16I-UNet	0.46 ± 0.04	0.63 ± 0.02	0.60 ± 0.02	0.59 ± 0.01	0.64 ± 0.02
R50B-CNN	0.80 ± 0.04	0.80 ± 0.03	0.67 ± 0.03	0.53 ± 0.06	0.84 ± 0.01
Segformer	0.74 ± 0.03	0.74 ± 0.02	0.66 ± 0.02	0.65 ± 0.03	0.77 ± 0.04
V16B-UNet	0.88 ± 0.01	0.85 ± 0.04	0.70 ± 0.09	0.69 ± 0.09	0.87 ± 0.02
			IoU scores		
V16I-UNet	0.33 ± 0.05	0.62 ± 0.04	0.68 ± 0.04	0.68 ± 0.03	0.63 ± 0.02
R50B-CNN	0.83 ± 0.01	0.83 ± 0.01	0.81 ± 0.01	0.78 ± 0.01	0.84 ± 0.01
Segformer	0.73 ± 0.03	0.73 ± 0.03	0.77 ± 0.01	0.78 ± 0.01	0.76 ± 0.02
V16B-UNet	0.88 ± 0.01	0.86 ± 0.03	0.82 ± 0.02	0.81 ± 0.02	0.87 ± 0.02
			R^2 scores		
V16I-UNet	0.04 ± 0.03	0.15 ± 0.05	0.12 ± 0.04	0.14 ± 0.06	0.16 ± 0.01
R50B-CNN	0.69 ± 0.04	0.7 ± 0.03	0.44 ± 0.04	0.26 ± 0.05	0.76 ± 0.02
Segformer	0.42 ± 0.08	0.35 ± 0.05	0.21 ± 0.03	0.18 ± 0.07	0.47 ± 0.1
V16B-UNet	0.82 ± 0.02	0.77 ± 0.06	0.56 ± 0.17	0.54 ± 0.11	0.81 ± 0.05

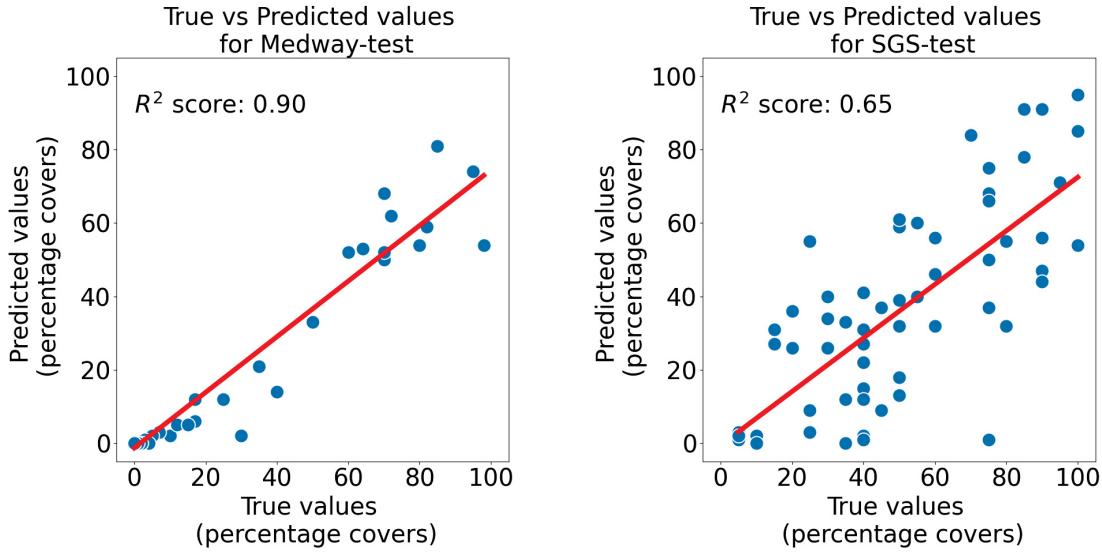
Table 5.1: F1, IoU, and R^2 scores presented in Figure 5.1

might rely on subtle textures that are altered by sharpening, while the models pretrained on a more generalized dataset can adapt to use the emphasized edges. Further, when the segmentation masks are altered using edge detection and clustering as explained in section 4.4.1, the models struggle to capture robust features, because of the discontinuity introduced by edge detection. The IoU scores (middle figure in 5.1) for all the models except the V16I-UNet remain fairly consistent across all experiments. The R^2 scores shown in Figure 5.1 follow nearly the same trend as the F1 scores. This similarity in trends suggests that the models' performance is relatively consistent across different scales — from individual pixels to larger areas. This indicates that the features the models are learning are effective at both fine-grained and broader scales.

A true test of these models is provided by the performance on the test datasets as shown in Figure 5.2. The negative R^2 scores in this table indicate that the models

Models	Experiments				
	E1	**E2**	**E3**	**E4**	**E5**
Medway test dataset					
V16I-UNet	−4.60	0.23	0.26	0.60	−0.35
R50B-CNN	0.85	0.36	0.38	−0.39	0.92
Segformer	−0.13	−0.11	−0.40	−0.37	−0.31
V16B-UNet	0.91	−0.18	0.01	−0.39	0.93
SGS test dataset					
V16I-UNet	−0.92	−0.02	0.1	−1.15	0.61
R50B-CNN	−0.88	−1.54	−2.47	−3.52	−0.47
Segformer	0.11	0.25	−3.05	−2.94	0.30
V16B-UNet	0.64	−1.51	−2.12	−3.18	0.60

Table 5.2: R^2 scores for different models evaluated on the test datasets.



(a) Nearby shorelines

(b) Further shorelines

Figure 5.2: R^2 scores comparing model predictions on Medway-test dataset and SGS-test dataset with human estimated percentage covers.

perform worse than a horizontal line fit to the data. Scores near to 1 indicate near-perfect predictions. The results on the Medway test dataset which is collected from locations near the training and validation dataset collection sites and the SGS test dataset which is a collection of images from the world indicate that the V16B-UNet performs the best using unaltered images and masks (experiment E1) as well as using the reduced set of images and masks (experiment E5). The scatterplots in Figure 5.2 compare the true and predicted percentage cover estimates when V16B-UNet is used for inference on the two test datasets. The R^2 scores indicate that the model generalizes well to data with similar distributions.

The first hypothesis stated that if edge detection and clustering are applied to the segmentation masks before training, it helps the model to discern boundaries between eelgrass and non-eelgrass regions and in turn improve the model performance. As seen in Figure 5.1, the hypothesis is true for models that are trained from scratch (random initialization of weights). However, pre-trained models did not show a significant change in performance. Edge detection and clustering, while helpful for highlighting certain features, inevitably result in some information loss for the eelgrass dataset. Pre-trained models seem to be more sensitive to this loss, as they have been optimized to use more of the original input information. While edge detection and clustering provide a helpful starting point for non-pre-trained models by emphasizing important structural information, these processes interfere with the feature extraction capabilities already present in pre-trained models. Hence, for pre-trained models, it's advisable to provide raw or minimally processed input data to leverage their pre-learned features and representations.

The model performance for V16I-UNet improves when the low-quality data is removed as seen by comparing experiments 1 and 5 in Figure 5.1. All the other models do not show a significant change after the low-quality images are removed. This is contrary to the second hypothesis stating that the model performance improves when low-quality data is removed. An interesting result was observed in this scenario. It is that the robustness of V16B-UNet decreases and R50B-CNN increases when the low-quality images are removed. Robustness is measured in terms of the standard deviation, where a higher standard deviation indicates lower robustness and vice versa. The removal of low-quality images improved the quality of features learned

during fine-tuning without losing the robustness gained during initial training for R50B-CNN, which increases its robustness. On the contrary, removing low-quality images reduces data diversity, which acted as a regularizer for V16B-UNet, which overfit the remaining high-quality data, reducing its robustness. It can be said that due to its simpler architecture, the V16B-UNet model relies more on consistent data quality, while R50B-CNN benefits from learning from diverse data, which helps in generalization.

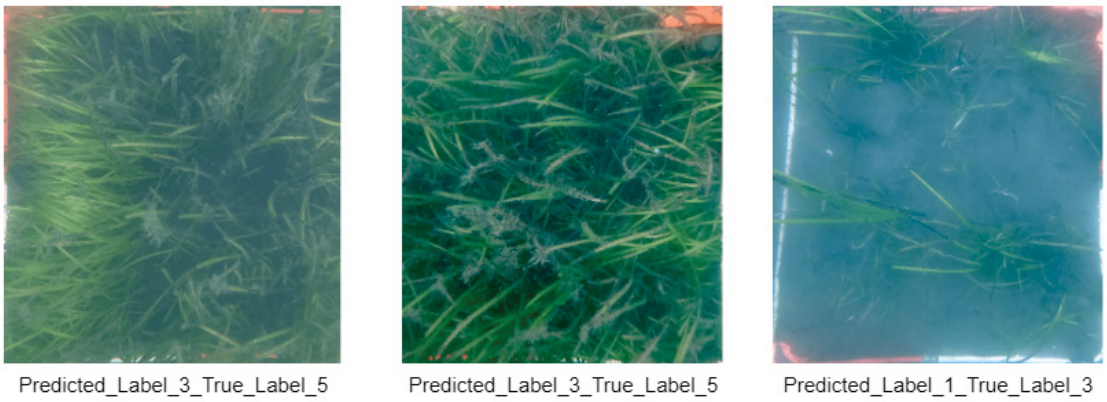


Figure 5.3: Examples of predictions where the ground truth is two or more than two classes away from the actual class from the segmentation model predictions for test set 1. The image labels show the model predictions followed by the true class label.

The segmentation masks generated by V16B-UNet were converted to percentage cover estimates and then discretized as explained in section 4.5.1. These outputs are compared with the human-estimated covers and the predictions that were further than two classes from the true predictions were noted in figures 5.3 and 5.4. These figures show that V16B-UNet underestimated eelgrass in all the predictions where the estimations were two classes further than true labels. For the Medway-test dataset, the model failed to recognize eelgrass blades where an organism had grown on top of the blades, along with blades that were tightly joined together which possibly made it difficult for the model to distinguish from the background because of analogous colours. For the SGS-test dataset, blurry images were predominantly the ones not classified properly (i.e., 6 out of 11). Others included images having very dark blades of eelgrass which were not present in the training set, and images where the tips of the blades were recognized, but not the blades themselves. The latter is a surprising

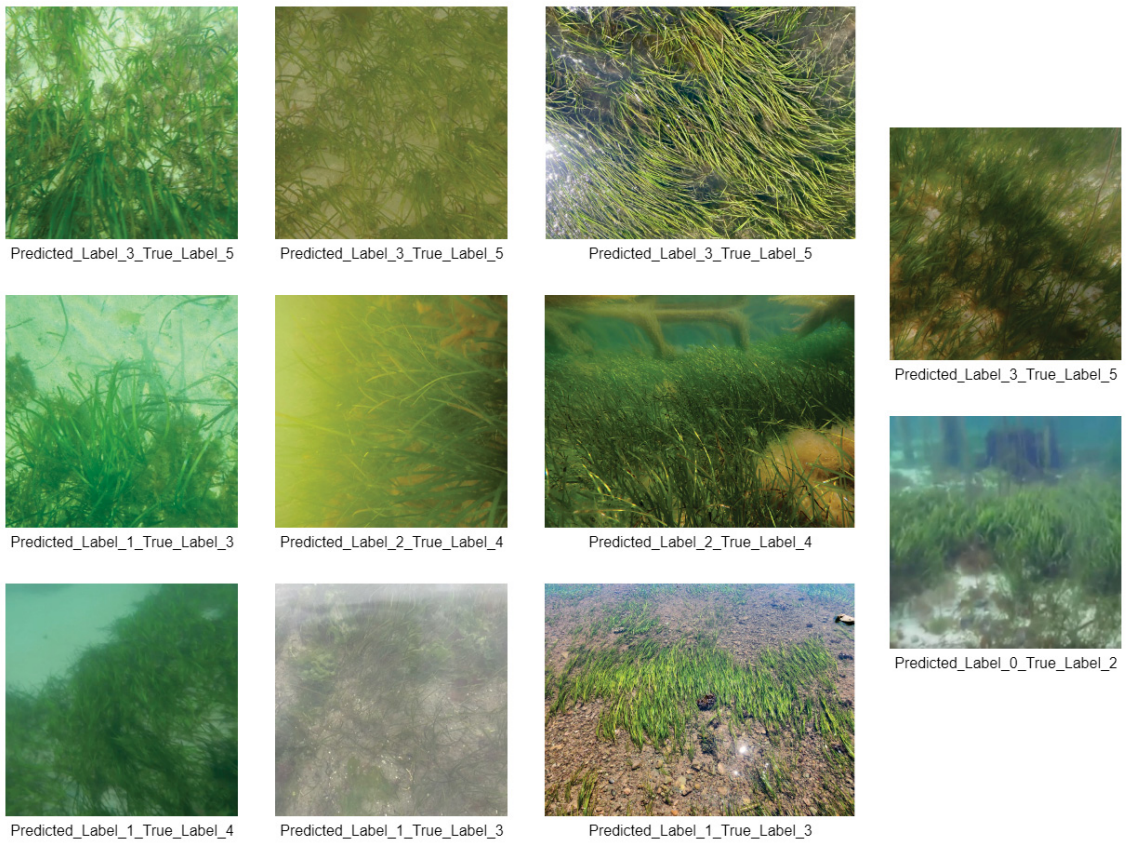


Figure 5.4: Examples of predictions where the ground truth is two or more than two classes away from the actual class from the segmentation model predictions for test set 2. The image labels show the model predictions followed by the true class label.

discovery, evident in both the test sets.

The analysis performed here uses the V16B-UNet. The model was able to identify eelgrass blades that were at a certain angle to each other as shown in Figure 5.5. This included long and short blades, including blades with shades of brown. Eelgrass blades were detected despite them having dense organisms grown on top of them. If the image was blurred, only thin separate blades were detected shown in Figure 5.6. If the image was blurred, except for thin blades, eelgrass was not detected. The model primarily struggled to detect small patches of eelgrass consisting of only 2 or 3 blades. When these blades were clustered together, detection was easier compared to when the blades were spread out in different directions. Additionally, very thin blades were more likely to be detected. Dark green blades that closely matched the background colour were not detected. In some images, brown vegetation, which is not

eelgrass, was incorrectly classified as eelgrass. Long, thin, separate blades and long blades were also not identified as eelgrass. Additionally, blades sparsely covered with other organisms were not recognized as eelgrass. These observations are illustrated in Figure 5.7.

Figure 5.5: Overlay of predictions where eelgrass blades on a different angle from each other have been detected. The light mask shows the presence predictions.

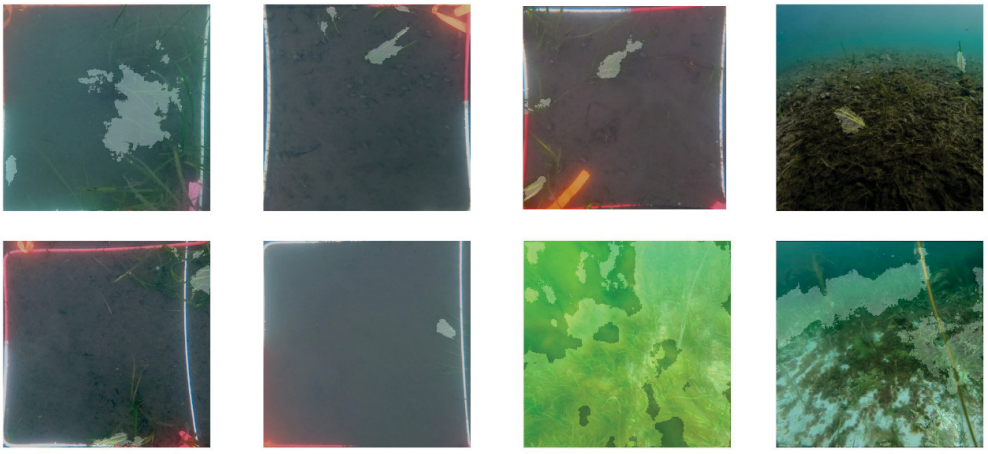


Figure 5.6: Overlay of predictions where thin eelgrass blades were detected in blurred images. The light mask shows the presence predictions.

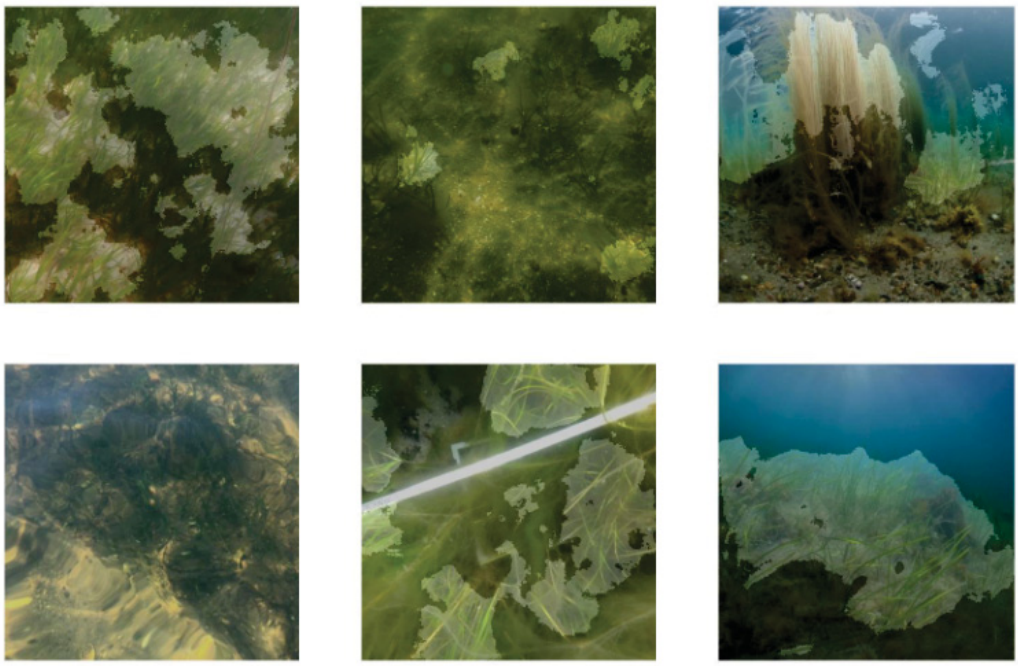


Figure 5.7: Overlay of predictions where dark eelgrass blades, and ones analogous to the background were not detected. The light mask shows the presence predictions.

5.2 Image Classification

The results in Figure 5.8 overall show that all the pre-trained models perform better than the non-pre-trained models. The classification models were trained on both random splits of the dataset and stratified splits. The F1 scores and accuracies were not statistically significantly different after stratification. The statistical significance of models is evaluated using standard deviations. Specifically, if the upper bound of the standard deviation of a model trained on a random split is within the lower bound of the standard deviation of a model trained on a stratified split, the models are not considered to be statistically significantly different. This criterion ensures that any observed differences in model performance are not due to random chance but are instead indicative of genuine variance in the data splits. As the stratified split does not result in a statistically significant difference, the metrics shown in Figure 5.8 represent the results obtained from a random split of the dataset. The pre-trained models perform better than their non-pre-trained counterparts for all the types of

models evaluated in image classification as shown in Figure 5.8. The BenthicNet dataset on which the models are pre-trained, consists of ocean-floor images from all over the world and the pre-trained model starts with weights that already capture useful information, helping to avoid poor local minima in the loss landscape. It also needs less labelled data as compared to training a model from scratch. This is particularly beneficial in the context of marine image datasets, where obtaining labelled data can be time-consuming and costly. This proves the third hypothesis stating that models pre-trained on a similar dataset, perform better than the models with random weight initialization.

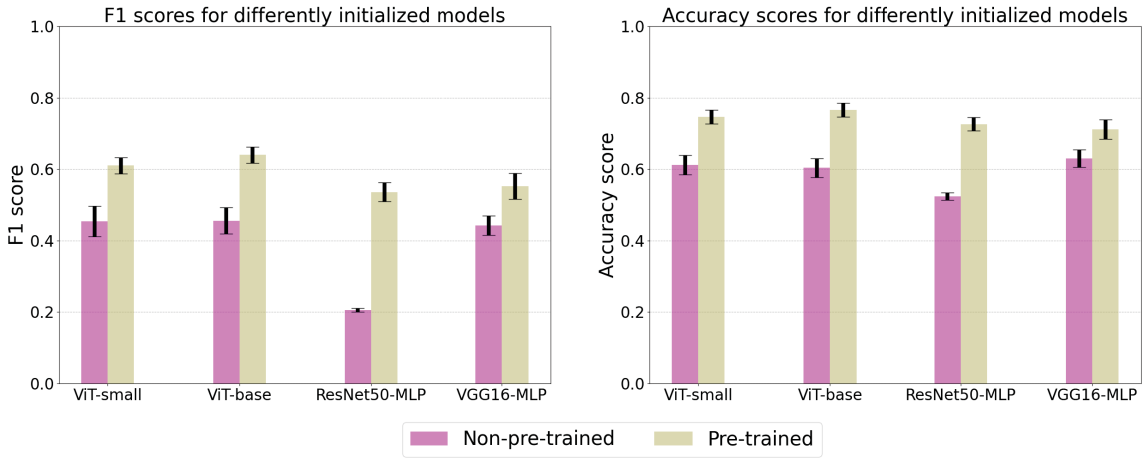


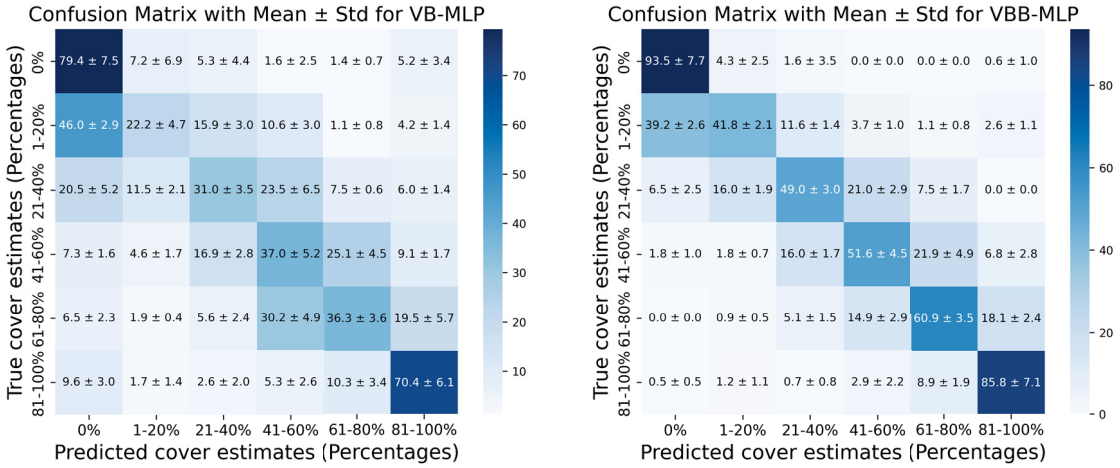
Figure 5.8: Comparison between the F1 and accuracy scores obtained from the two experiments performed in image classification for the pre-trained and non-pre-trained versions of the four models used. The blue bars show results from experiment 1 which uses random weights to initialize the models, and the orange bars show results from experiment 2 which uses BenthicNet pre-trained weights to initialize the models. The tabular representation for this image is provided in Table 5.3.

The confusion matrices from the ViT-Base model initialized using random weights (Subfigure (a) in 5.9) and pre-trained weights (Subfigure (b) in 5.9) are shown. Both models tend to misclassify into adjacent classes more often than distant classes. As the classes represent continuous ranges of eelgrass coverage, this ordinal nature of the classes causes misclassifications between adjacent classes. Pre-training addresses this problem significantly for this dataset. The pre-trained model also shows lower standard deviations, indicating more consistent performance across runs.

Class predictions were generated from the ViT-base model with pre-trained weights from BenthicNet, and trained using a random split on the dataset. These predictions

Initialization	Models			
	ViT-small	**ViT-base**	**ResNet50-MLP**	**VGG16-MLP**
		F1 scores		
Non-pre-trained	0.45 ± 0.04	0.46 ± 0.03	0.21 ± 0.00	0.44 ± 0.03
Pre-trained	0.61 ± 0.02	0.64 ± 0.02	0.54 ± 0.02	0.55 ± 0.04
		Accuracy scores		
Non-pre-trained	0.61 ± 0.03	0.60 ± 0.0	0.52 ± 0.01	0.63 ± 0.02
Pre-trained	0.75 ± 0.02	0.77 ± 0.02	0.73 ± 0.02	0.71 ± 0.03

Table 5.3: F1 and accuracy scores presented in Figure 5.8



(a) Non-pre-trained

(b) Pre-trained

Figure 5.9: Confusion matrices for the ViT-Base model with randomly initialized weights (a) and BenthicNet pre-trained weights (b). The entries represent the mean percentage of correct predictions in that cell ± standard deviation.

were compared with the human-estimated percentage covers. The figures 5.10 and 5.11 show the images for which the predictions were two classes further than the true classes. These diagrams show that the classification model overestimated the quantity of eelgrass in many of the images from both the test datasets. In the other images, where the model underestimated, it was not able to recognize any of the eelgrass. This is speculated due to blurred images and very tiny blades.

Figure 5.10: Examples of predictions where the ground truth is two or more than two classes away from the actual class from the classification model predictions for test set 1. The image labels show the model predictions followed by the true class label.

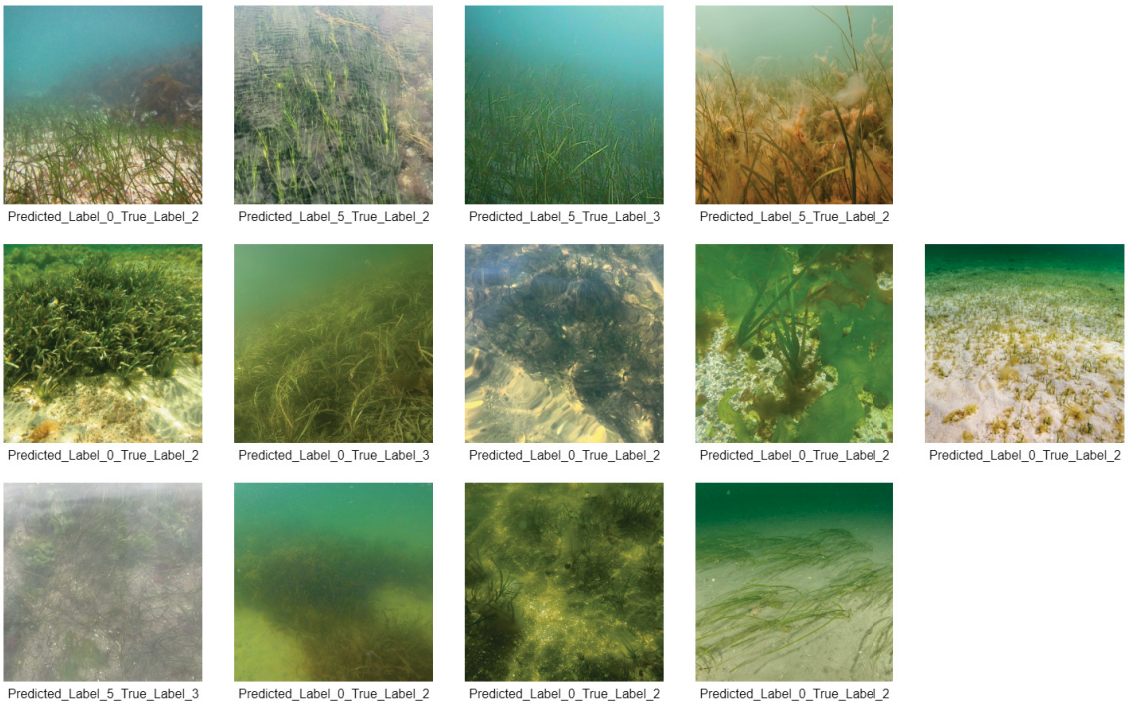


Figure 5.11: Examples of predictions where the ground truth is two or more than two classes away from the actual class from the classification model predictions for test set 2. The image labels show the model predictions followed by the true class label.

5.3 Comparing Image Segmentation And Image Classification Models

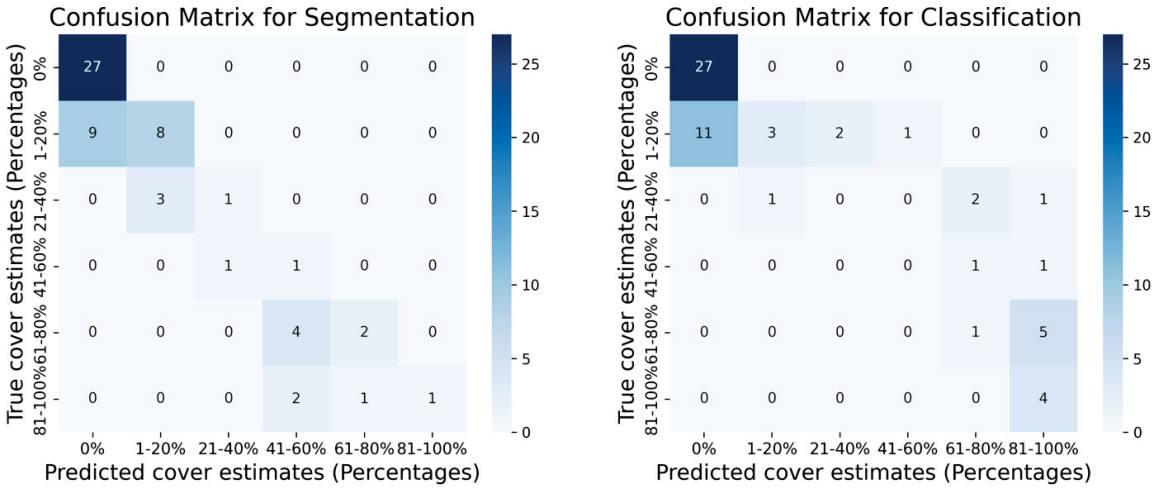


Figure 5.12: Examples of predictions where the ground truth is two or more than two classes away from the actual class from the classification model predictions for test set 2. The image labels show the model predictions followed by the true class label.

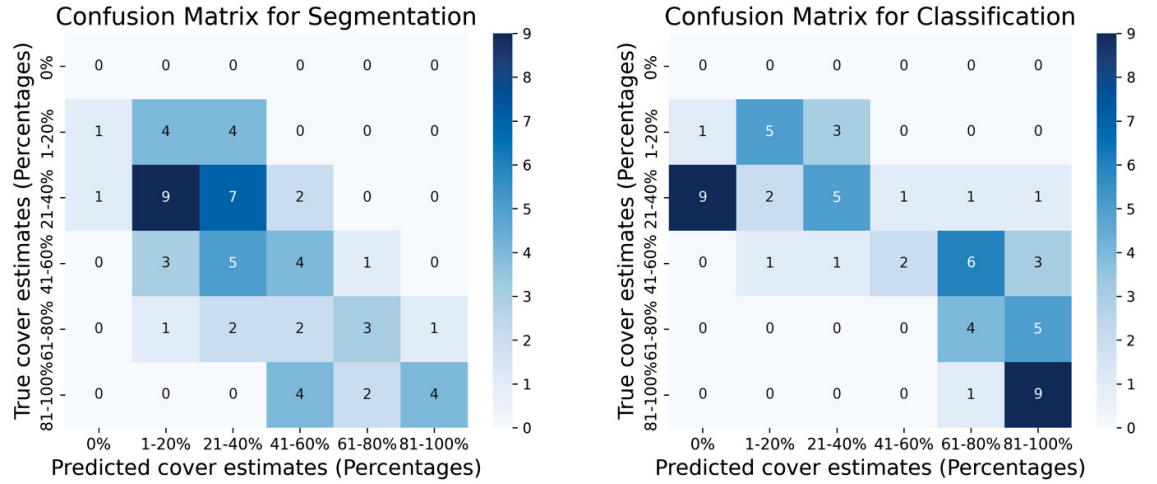


Figure 5.13: Examples of predictions where the ground truth is two or more than two classes away from the actual class from the classification model predictions for test set 2. The image labels show the model predictions followed by the true class label.

A comparison between the segmentation and classification models on test datasets can be used to juxtapose their effectiveness. The results from the segmentation models are continuous and need to be converted to discrete classes as in image classification to compare them. The predictions from the V16B-UNet segmentation model are

converted to discrete classes and compared to predictions from the VBB-MLP classification model for both the Medway-test dataset in Figure 5.12 and the SGS-test dataset in Figure 5.13. Analyzing Figure 5.12, the segmentation model is better at distinguishing between cover percentage ranges and the classification model tends to underestimate cover, often predicting 0% when there is actually some cover present. Classification shows more extreme misclassifications (e.g., predicting 21-40% for true 81-100%), while segmentation's errors are generally closer to the true values looking at Figure 5.13.

Chapter 6

Conclusion

In this study, we tried different methods to estimate the percentage cover of eelgrass from a given image. These included image segmentation and image classification approaches using several models. Various pre-processing techniques were experimented with for image segmentation, and two different initializations were tried on image classification models.

To perform image classification, the human-estimated percentage covers were converted to 6 discrete classes where class 1 represented 0% eelgrass, class 2 represented 1–20% eelgrass, class 3 represented 21–40% eelgrass, and so on where class 6 represented 81–100% eelgrass cover in a given image. All the classification models struggled to accurately distinguish between classes with similar eelgrass coverage percentages. All the classification models produced analogous results. While the pre-trained ViT models, particularly ViT-Base with an F1 score of 0.64 ± 0.02 and an accuracy score of 0.77 ± 0.02, show a trend towards higher performance, the differences between models are not always statistically significant given the reported standard deviations. Pre-training consistently improves performance across all models. The choice of the best model depends on factors beyond these metrics, including computational resources and specific application requirements. To use the VBB-MLP classification model, we have developed a GitHub repository [73]. The code takes in a set of images from a user-specified folder, runs inference on these images, and creates a CSV file containing the filenames and model predictions. The flowchart explaining this is shown in the README.md file of the GitHub repository [73].

For image segmentation, human-annotated segmentation masks were used for training the models. All the segmentation models except the V16I-UNet performed very similar to each other. For applications with non-preprocessed data, the V16B-UNet model with a pixel-wise F1 score of 0.88 ± 0.01, an IoU score of 0.88 ± 0.01,

and an R^2 score of 0.82 ± 0.02 would be a preferable choice. Segformer is more consistent across different pre-processing conditions. While the raw performance metrics provide valuable insights, measuring effect size offers a deeper understanding of the significance of the differences observed between models and across experimental conditions. Effect size analysis, using measures such as Cohen's d [74], can quantify the magnitude of differences beyond mere statistical significance. To use the V16B-UNet segmentation model, we have developed a GitHub repository [72]. The code takes in a set of images from a user-specified folder, runs inference on these images, and creates a CSV file containing the filenames and model predictions, along with the binary segmentation masks. The flowchart explaining this is shown in the README.md file of the GitHub repository [72].

The segmentation model V16B-UNet overall performs better than the other segmentation models as well as the classification models for the test datasets. This model comfortably detects eelgrass blades that are situated at a certain angle to each other. This includes long and short blades along with brown blades. If the image provided is blurry, the model can only detect individual thin blades. Apart from this observation, dark-green blades which are analogous to the background colour were not detected.

Pre-processing techniques like sharpening images, and applying edge detection to refine image features for image segmentation models proved to be helpful for non-pre-trained models as they emphasized important structural information, while these pre-processing techniques interfered with the feature extraction capabilities already present in pre-trained models. Hence, providing raw or minimally processed input data to pre-trained models proved to be more beneficial. Removing low-quality images from the training dataset did not affect the performance of the segmentation models significantly which is contrary to the second hypothesis which stated that removing low-quality images would improve the model performances. Also, unseen data is expected to contain low-quality images. Hence, the model needs to train on these images to improve its generalizability. Pre-trained models with simpler architectures relied more on consistent data quality, while complex architecture models benefited from diverse data, improving generalization. All the models pre-trained on a similar domain dataset proved to perform better than their randomly initialized counterparts.

Another study on the test datasets shown in Table 5.2 suggests that the segmentation models are more suitable for Canadian coasts than other coasts. The reason is the varying performance of this model on the two datasets. On the Medway-test dataset, which is collected from the Medway River basin, the model performs much better than the SGS-test dataset, which is collected randomly from different shorelines across the world, as shown in Figure 5.2.

Future Improvements

Some improvements can be experimented with for effectively detecting eelgrass. The first improvement can be ensuring that the dataset contains a reasonably balanced amount of images for all depths, camera angles, and brightness to make the models more generalizable and robust to identify eelgrass in any ocean-floor environment. Secondly, a combination of classification and segmentation can be used to determine percentage cover estimates, where the classification model predicts whether there is eelgrass in an image. If there is eelgrass, then the segmentation model can be used to produce segmentation masks. If there is no eelgrass, segmentation is not needed, and the percentage cover is zero. Although this approach did not work on the classification models tried in this study, using bigger image sizes was not experimented with and can be explored further. Bayesian models [75, 76, 77] which take into account the model uncertainty can be used to better identify faulty predictions. Bayesian models take a distribution of weights instead of a single weight which can perform well for estimating eelgrass percentage covers. Practically, these models are compute-intensive and hence cannot be used in resource-constrained environments. Models can be developed which can be fine-tuned to a specific dataset by looking only at a few images of the new dataset. This is another application of transfer learning.

Bibliography

[1] M. S. Fonseca, W. J. Kenworthy, and G. W. Thayer, "A low cost transplanting procedure for sediment stabilization and habitat development using eelgrass (zosteramarina)," *Wetlands*, vol. 2, no. 1, pp. 138–151, 1982. [Online]. Available: https://doi.org/10.1007/BF03160551

[2] M. E. Röhr, M. Holmer, J. K. Baum, M. Björk, K. Boyer, D. Chin, L. Chalifour, S. Cimon, M. Cusson, M. Dahl, D. Deyanova, J. E. Duffy, J. S. Eklöf, J. K. Geyer, J. N. Griffin, M. Gullström, C. M. Hereu, M. Hori, K. A. Hovel, A. R. Hughes, P. Jorgensen, S. Kiriakopolos, P.-O. Moksnes, M. Nakaoka, M. I. O'Connor, B. Peterson, K. Reiss, P. L. Reynolds, F. Rossi, J. Ruesink, R. Santos, J. J. Stachowicz, F. Tomas, K.-S. Lee, R. K. F. Unsworth, and C. Boström, "Blue carbon storage capacity of temperate eelgrass () meadows," *Global Biogeochemical Cycles*, vol. 32, no. 10, pp. 1457–1475, 2018. [Online]. Available: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2018GB005941

[3] M. Luhar, E. Infantes, and H. Nepf, "Seagrass blade motion under waves and its impact on wave decay," *Journal of Geophysical Research: Oceans*, vol. 122, no. 5, pp. 3736–3752, 2017. [Online]. Available: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1002/2017JC012731

[4] E. L. Jackson, A. A. Rowden, M. J. Attrill, S. J. Bossey, and M. B. Jones, "The importance of seagrass beds as a habitat for fishery species," *Oceanography and marine biology*, vol. 39, pp. 269–304, 2001.

[5] K. A. Capistrant-Fossa and K. H. Dunton, "Rapid sea level rise causes loss of seagrass meadows," *Communications Earth Environment*, vol. 5, no. 1, p. 87, 2024. [Online]. Available: https://doi.org/10.1038/s43247-024-01236-7

[6] K. A. Moore, E. C. Shields, and D. B. Parrish, "Impacts of varying estuarine temperature and light conditions on zostera marina (eelgrass) and its interactions with ruppia maritima (widgeongrass)," vol. 37, no. 1, pp. 20–30. [Online]. Available: https://doi.org/10.1007/s12237-013-9667-3

[7] L. Ericsson, H. Gouk, C. C. Loy, and T. M. Hospedales, "Self-supervised representation learning: Introduction, advances, and challenges," *IEEE Signal Processing Magazine*, vol. 39, no. 3, pp. 42–62, 2022. [Online]. Available: https://doi.org/10.1109/MSP.2021.3134634

[8] C. Doersch, A. Gupta, and A. A. Efros, "Unsupervised visual representation learning by context prediction," in *2015 IEEE International Conference on Computer Vision (ICCV)*. Los Alamitos, CA, USA: IEEE Computer Society,

dec 2015, pp. 1422–1430. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICCV.2015.167

[9] A. Jaiswal, A. R. Babu, M. Z. Zadeh, D. Banerjee, and F. Makedon, "A survey on contrastive self-supervised learning," *Technologies*, vol. 9, no. 1, 2021. [Online]. Available: https://doi.org/10.3390/technologies9010002

[10] L. Jing and Y. Tian, "Self-supervised visual feature learning with deep neural networks: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, pp. 4037–4058, 2019. [Online]. Available: https://doi.org/10.1109/TPAMI.2020.2992393

[11] S. Bozinovski and A. Fulgosi, "The influence of pattern similarity and transfer of learning upon training of a base perceptron b2," in *Proceedings of the Symposium Informatica*, Bled, 1976, pp. 3–121–5, original in Croatian: Utjecaj slicnosti likova i transfera ucenja na obucavanje baznog perceptrona B2.

[12] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2021. [Online]. Available: https://doi.org/10.1109/JPROC.2020.3004555

[13] R. Ribani and M. Marengoni, "A survey of transfer learning for convolutional neural networks," in *2019 32nd SIBGRAPI Conference on Graphics, Patterns and Images Tutorials (SIBGRAPI-T)*, 2019, pp. 47–57. [Online]. Available: https://doi.org/10.1109/SIBGRAPI-T.2019.00010

[14] S. Becker and G. E. Hinton, "Self-organizing neural network that discovers surfaces in random-dot stereograms," *Nature*, vol. 355, no. 6356, pp. 161–163, 1992. [Online]. Available: https://doi.org/10.1038/355161a0

[15] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 1597–1607. [Online]. Available: https://proceedings.mlr.press/v119/chen20j.html

[16] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. [Online]. Available: https://doi.org/10.1109/CVPR42600.2020.00975

[17] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, p. 60, 2019. [Online]. Available: https://doi.org/10.1186/s40537-019-0197-0

[18] X. Chen, S. Xie, and K. He, "An empirical study of training self-supervised vision transformers," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2021, pp. 9640–9649. [Online]. Available: https://doi.org/10.1109/ICCV48922.2021.00950

[19] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, 2005, pp. 539–546 vol. 1. [Online]. Available: https://doi.org/10.1109/CVPR.2005.202

[20] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny, "Barlow Twins: Self-Supervised Learning via Redundancy Reduction," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 12 310–12 320. [Online]. Available: https://proceedings.mlr.press/v139/zbontar21a.html

[21] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo, "Segformer: Simple and efficient design for semantic segmentation with transformers," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 12 077–12 090. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2021/file/64f1f27bf1b4ec22924fd0acb550c235-Paper.pdf

[22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[23] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault, Eds. Online: Association for Computational Linguistics, Jul. 2020, pp. 7871–7880. [Online]. Available: https://aclanthology.org/2020.acl-main.703

[24] Y. Ren, Y. Ruan, X. Tan, T. Qin, S. Zhao, Z. Zhao, and T.-Y. Liu, "Fastspeech: Fast, robust and controllable text to speech," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/f63f65b503e22cb970527f23c9ad7db1-Paper.pdf

[25] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://doi.org/10.48550/arXiv.2010.11929

[26] P. Wang, "lucidrains: segformer-pytorch," GitHub repository, 2024. [Online]. Available: https://github.com/lucidrains/segformer-pytorch

[27] D. Hendrycks and K. Gimpel, "Bridging nonlinearities and stochastic regularizers with gaussian error linear units," *CoRR*, vol. abs/1606.08415, 2016. [Online]. Available: https://doi.org/10.48550/arXiv.1606.08415

[28] T. maintainers and contributors, "TorchVision: PyTorch's Computer Vision library," Nov. 2016. [Online]. Available: https://github.com/pytorch/pytorch/blob/main/torch/nn/functional.py

[29] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds. Cham: Springer International Publishing, 2015, pp. 234–241. [Online]. Available: https://doi.org/10.1007/978-3-319-24574-4_28

[30] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: https://doi.org/10.48550/arXiv.1409.1556

[31] A. Pravitasari, N. Iriawan, M. Almuhayar, T. Azmi, I. Irhamah, K. Fithriasari, S. Purnami, and W. Ferriastuti, "Unet-vgg16 with transfer learning for mri-based brain tumor segmentation," *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, vol. 18, p. 1310, 06 2020. [Online]. Available: http://doi.org/10.12928/telkomnika.v18i3.14753

[32] Z. Zhou, "zhoudaxia233: Pytorch-unet," GitHub repository, 2024. [Online]. Available: https://github.com/zhoudaxia233/PyTorch-Unet

[33] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *CoRR*, vol. abs/1409.0575, 2014. [Online]. Available: https://doi.org/10.48550/arXiv.1409.0575

[34] S. C. Lowe, B. Misiuk, I. Xu, S. Abdulazizov, A. R. Baroi, A. C. Bastos, M. Best, V. Ferrini, A. Friedman, D. Hart, O. Hoegh-Guldberg, D. Ierodiaconou, J. Mackin-McLaughlin, K. Markey, P. S. Menandro, J. Monk, S. Nemani, J. O'Brien, E. Oh, L. Y. Reshitnyk, K. Robert, C. M. Roelfsema, J. A.

Sameoto, A. C. G. Schimel, J. A. Thomson, B. R. Wilson, M. C. Wong, C. J. Brown, and T. Trappenberg, "Benthicnet: A global compilation of seafloor images for deep learning applications," 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2405.05241

[35] K. Fukushima, "Cognitron: A self-organizing multilayered neural network," *Biological Cybernetics*, vol. 20, pp. 121–136, 1975. [Online]. Available: https://doi.org/10.1007/BF00342633

[36] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML'10. Madison, WI, USA: Omnipress, 2010, p. 807–814. [Online]. Available: https://dl.acm.org/doi/10.5555/3104322.3104425

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, jun 2016, pp. 770–778. [Online]. Available: https://doi.org/10.1109/CVPR.2016.90

[38] V. Turrisi, "vturrisi: solo-learn," GitHub repository, 2024. [Online]. Available: https://github.com/vturrisi/solo-learn/blob/main/solo/backbones/vit/vit_mocov3.py

[39] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *stat*, vol. 1050, p. 21, 2016. [Online]. Available: https://doi.org/10.48550/arXiv.1607.06450

[40] S. Nerella, S. Bandyopadhyay, J. Zhang, M. Contreras, S. Siegel, A. Bumin, B. Silva, J. Sena, B. Shickel, A. Bihorac, K. Khezeli, and P. Rashidi, "Transformers in healthcare: A survey," 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2307.00067

[41] Y.-H. Tsai, O. C. Hamsici, and M.-H. Yang, "Adaptive region pooling for object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015. [Online]. Available: https://doi.org/10.1109/CVPR.2015.7298673

[42] C. Van Rijsbergen, *Information Retrieval*. Butterworths, 1979. [Online]. Available: https://books.google.ca/books?id=t-pTAAAAMAAJ

[43] T. Wiesner-Hanks, H. Wu, E. Stewart, C. DeChant, N. Kaczmar, H. Lipson, M. A. Gore, and R. J. Nelson, "Millimeter-level plant disease detection from aerial photographs via deep learning and crowdsourced data," *Frontiers in Plant Science*, vol. 10, 2019. [Online]. Available: https://www.frontiersin.org/journals/plant-science/articles/10.3389/fpls.2019.01550

[44] R. A. Arun, S. Umamaheswari, and A. V. Jain, "Reduced u-net architecture for classifying crop and weed using pixel-wise segmentation," in *2020 IEEE International Conference for Innovation in Technology (INOCON)*, 2020, pp. 1–6. [Online]. Available: https://doi.org/10.1109/INOCON50539.2020.9298209

[45] A. H. Murphy, "The Finley Affair: A Signal Event in the History of Forecast Verification," *Weather and Forecasting*, vol. 11, no. 1, pp. 3–20, Mar. 1996. [Online]. Available: https://doi.org/10.1175/1520-0434(1996)011%3C0003:TFAASE%3E2.0.CO;2

[46] L. R. Dice, "Measures of the amount of ecologic association between species," *Ecology*, vol. 26, no. 3, pp. 297–302, 1945. [Online]. Available: https://doi.org/10.2307/1932409

[47] T. Sørensen, *A Method of Establishing Groups of Equal Amplitude in Plant Sociology Based on Similarity of Species Content and Its Application to Analyses of the Vegetation on Danish Commons*, ser. Biologiske skrifter. Munksgaard in Komm., 1948. [Online]. Available: https://books.google.ca/books?id=rpS8GAAACAAJ

[48] U. Koc, E. Sezer, Y. Ozkaya, Y. Yarbay, O. Taydas, V. Ayyildiz, H. Kiziloglu, U. Kesimal, I. Cankaya, M. Beşler, E. Karakas, F. Karademir, N. Sebik, M. Bahadir, O. Sezer, B. Yeşilyurt, S. Varli, E. Akdogan, M. Ulgu, and S. Birinci, "Artificial intelligence in healthcare competition (teknofest-2021): Stroke data set," *The Eurasian Journal of Medicine*, vol. 54, 07 2022. [Online]. Available: https://doi.org/10.5152/eurasianjmed.2022.22096

[49] S. Wright, "Correlation and causation," *Journal of agricultural research*, vol. 20, no. 7, p. 557, 1921. [Online]. Available: https://books.google.sn/books?id=lNNdIV_qpwIC

[50] MathWorks, "Superpixels," https://www.mathworks.com/help/images/ref/superpixels.html, accessed: July 11, 2024.

[51] J. Malik and X. Ren, "Learning a classification model for segmentation," in *Computer Vision, IEEE International Conference on*, vol. 2. Los Alamitos, CA, USA: IEEE Computer Society, oct 2003, p. 10. [Online]. Available: https://doi.org/10.1109/ICCV.2003.1238308

[52] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, "Slic superpixels," EPFL, Tech. Rep. 149300, June 2010. [Online]. Available: https://www.epfl.ch/labs/ivrl/research/slic-superpixels/

[53] M. Van den Bergh, X. Boix, G. Roig, B. de Capitani, and L. Van Gool, "Seeds: Superpixels extracted via energy-driven sampling," in *Computer Vision – ECCV 2012*, A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 13–26. [Online]. Available: https://doi.org/10.48550/arXiv.1309.3848

[54] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *International Journal of Computer Vision*, vol. 59, no. 2, pp. 167–181, 2004. [Online]. Available: https://doi.org/10.1023/B:VISI.0000022288.19776.77

[55] M. Yamamuro, K. Nishimura, K. Kishimoto, K. Nozaki, K. Kato, A. Negishi, K. Otani, H. Shimizu, T. Hayashibara, M. Sano, M. Tamaki, and K. Fukuoka, *Mapping tropical seagrass beds with an underwater remotely operated vehicle (ROV)*. Japan International Marine Science and Technology Federation, 04 2003. [Online]. Available: https://www.researchgate.net/publication/242533299_Mapping_tropical_seagrass_beds_with_an_underwater_remotely_operated_vehicle_ROV

[56] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979. [Online]. Available: https://doi.org/10.1109/TSMC.1979.4310076

[57] G. Reus, T. Möller, J. Jäger, S. T. Schultz, C. Kruschel, J. Hasenauer, V. Wolff, and K. Fricke-Neuderth, "Looking for seagrass: Deep learning for visual coverage estimation," in *2018 OCEANS - MTS/IEEE Kobe Techno-Oceans (OTO)*, 2018, pp. 1–6. [Online]. Available: https://doi.org/10.1109/OCEANSKOBE.2018.8559302

[58] F. Weidmann, J. Jäger, G. Reus, S. T. Schultz, C. Kruschel, V. Wolff, and K. Fricke-Neuderth, "A closer look at seagrass meadows: Semantic segmentation for visual coverage estimation," in *OCEANS 2019 - Marseille*, 2019, pp. 1–6. [Online]. Available: https://doi.org/10.1109/OCEANSE.2019.8867064

[59] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018. [Online]. Available: https://doi.org/10.1007/978-3-030-01234-2_49

[60] M. A. Ghazal, A. Mahmoud, A. Aslantas, A. Soliman, A. Shalaby, J. A. Benediktsson, and A. El-Baz, "Vegetation cover estimation using convolutional neural networks," *IEEE Access*, vol. 7, pp. 132 563–132 576, 2019. [Online]. Available: https://doi.org/10.1109/ACCESS.2019.2941441

[61] D. Langenkämper, M. Zurowietz, T. Schoening, and T. W. Nattkemper, "Corrigendum: Biigle 2.0 - browsing and annotating large marine image collections," *Frontiers in Marine Science*, vol. 7, 2020. [Online]. Available: https://doi.org/10.3389/fmars.2017.00083

[62] H. Bordin, "An assessment of manual and machine learning methods for analyzing seagrass restoration success," in *Dalhousie University's 37th Annual Cameron Conference*, Dalhousie University, Halifax, NS, Canada, February 2023, poster

presentation, supervised by Derek Tittensor, Biology Dept., Dalhousie University.

[63] L. L. Strachan, R. J. Lilley, and S. J. Hennige, "A regional and international framework for evaluating seagrass management and conservation," *Marine Policy*, vol. 146, p. 105306, 2022. [Online]. Available: https://doi.org/10.1016/j.marpol.2022.105306

[64] T. maintainers and contributors, "TorchVision: PyTorch's Computer Vision library," Nov. 2016. [Online]. Available: https://github.com/pytorch/vision/torchvision/transforms/transforms.py

[65] O. Rukundo and H. Cao, "Nearest Neighbor Value Interpolation," *International Journal of Advanced Computer Science and Applications*, vol. 3, no. 4, 2012. [Online]. Available: https://dx.doi.org/10.14569/IJACSA.2012.030405

[66] M. Zurowietz, D. Langenkämper, B. Hosking, H. A. Ruhl, and T. W. Nattkemper, "MAIA—A machine learning assisted image annotation method for environmental monitoring and exploration," *PLOS ONE*, vol. 13, no. 11, pp. 1–18, 11 2018. [Online]. Available: https://doi.org/10.1371/journal.pone.0207498

[67] P. Mehta, "parashirenmehta: Metadata_PreProcessing," GitHub repository, 2024. [Online]. Available: https://github.com/parashirenmehta/Metadata_Preprocessing/masks_from_csv.py

[68] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis amp; Machine Intelligence*, vol. 8, no. 06, pp. 679–698, nov 1986. [Online]. Available: https://doi.org/10.1109/TPAMI.1986.4767851

[69] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000. [Online]. Available: https://docs.opencv.org/4.x/d1/dfb/intro.html

[70] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf

[71] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and Édouard Duchesnay, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011. [Online]. Available: http://jmlr.org/papers/v12/pedregosa11a.html

[72] P. Mehta, "parashirenmehta: Folder_inference_segmentation," GitHub repository, 2024. [Online]. Available: https://github.com/parashirenmehta/Folder_inference_segmentation/

[73] ——, "parashirenmehta: Folder_inference_classification," GitHub repository, 2024. [Online]. Available: https://github.com/parashirenmehta/Folder_inference_classification/

[74] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*, 2nd ed. New York: Routledge, 1988. [Online]. Available: https://doi.org/10.4324/9780203771587

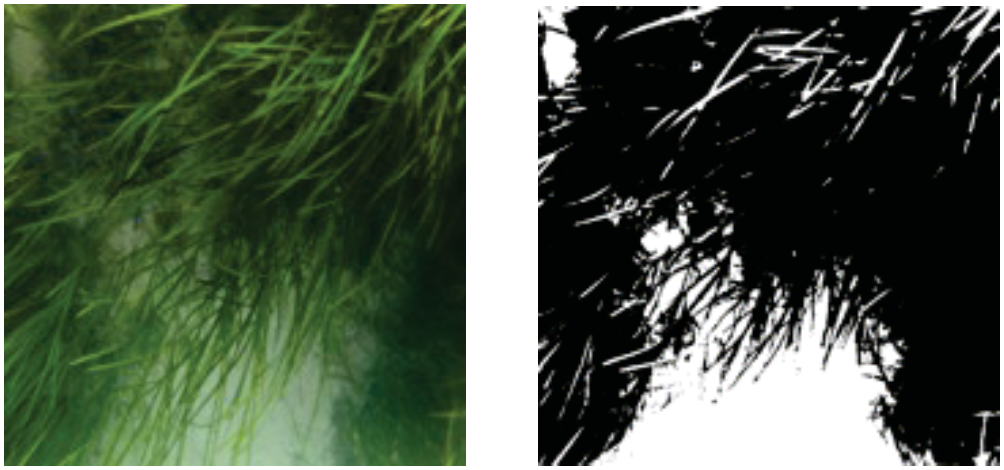
[75] D. J. C. MacKay, "Probable networks and plausible predictions-a review of practical bayesian methods for supervised neural networks," *Network: Computation in Neural Systems*, vol. 6, no. 3, p. 469, aug 1995. [Online]. Available: https://dx.doi.org/10.1088/0954-898X/6/3/011

[76] J. Lampinen and A. Vehtari, "Bayesian approach for neural networks—review and case studies," *Neural Networks*, vol. 14, no. 3, pp. 257–274, 2001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608000000988

[77] H. Wang and D.-Y. Yeung, "A survey on bayesian deep learning," 2021. [Online]. Available: https://arxiv.org/abs/1604.01662

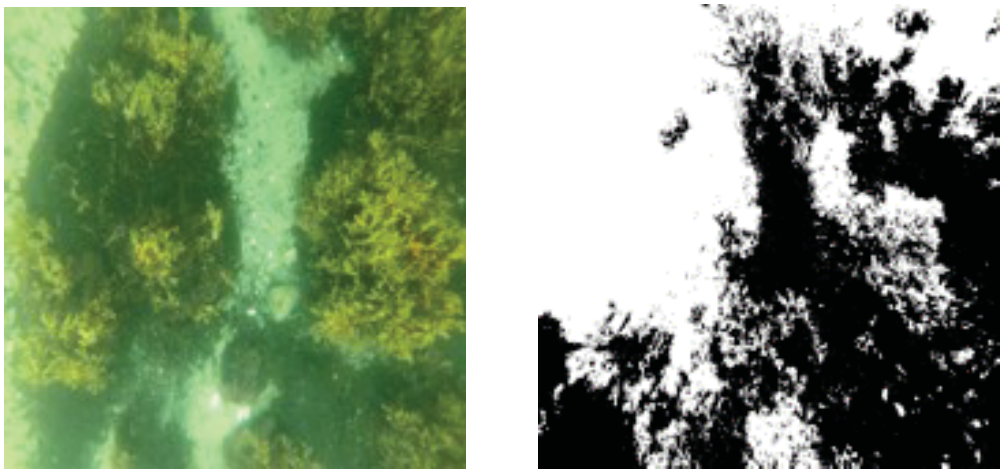
Appendix A

Related Tables and Diagrams



(a) Image (b) Prediction

Figure A.1: Image (a) and prediction (b) using OTSU for eelgrass segmentation.



(a) Image (b) Prediction

Figure A.2: Image (a) and prediction (b) using OTSU for eelgrass segmentation.