2022

Measuring Vegetation Density in Marsh Grass Photographs Using Convolutional Neural Networks

Lucas W. Welch
University of North Florida, lucaswwelch@gmail.com

MEASURING VEGETATION DENSITY IN MARSH GRASS PHOTOGRAPHS USING CONVOLUTIONAL NEURAL NETWORKS

by

Lucas Welch

A thesis submitted to the
School of Computing
in partial fulfillment of the requirements for the degree of

Master of Science in Computer and Information Sciences

UNIVERSITY OF NORTH FLORIDA
SCHOOL OF COMPUTING

May, 2022

The thesis "Measuring Vegetation Density in Marsh Grass Photographs Using Convolutional Neural Networks" submitted by Lucas Welch in partial fulfillment of the requirements for the degree of Master of Science in Computing and Information Sciences has been

Approved by the thesis committee: Date:

Dr. Xudong Liu
Thesis Advisor and Committee Chairperson

Dr. Indika Kahanda
Committee Member

Dr. Sandeep Reddivari
Committee Member

Dr. Karthikeyan Umapathy
Committee Member

ACKNOWLEGEMENTS

I would like to extend my thanks to several people from the School of Computing, the most important of whom is Dr. Xudong Liu who has supervised this project from the very beginning. Thank you for your guidence and for your support over the last two years—especially during some recent times of personal and family crisis. These thanks also extend to the other members of my thesis committee, Dr. Indika, Dr. Reddivari, and Dr. Umapathy. These individuals have provided the needed professional opinions to create a robust project and to get this project published several times. Thank you for your continued input and proofreading.

A special thanks also goes out to Anoushka Kabra who has programmed a large portion of our framework. Without her help, finishing this thesis would have been much more challenging.

I'd also like to thank my husband for his constant encouragement and assistance. Just being there for me during this rigorous program has been one of the biggest sources of encouragement.

CONTENTS

FIGURES

TABLES

ABSTRACT

A marsh is a wetland dominated by various species of grasses rather than trees. It supports an important ecosystem, providing habitats for many kinds of animals and posing a crucial impact on coastal climates. Marshes are rapidly changing, and it is vital for scientists to track these changes to understand the health of them. To do so, biologists perform *vegetation monitoring* to estimate the coverage of vegetation in an area of marsh. This task often calls for extensive human labor carefully examining pixels in photos of marsh sites to calculate *vegetation density* for a variety of marsh grass species. This is a very time-consuming process.

In this thesis work, I designed a computational framework that automatically predicts the vegetation density using deep learning models, in particular, convolutional neural networks (CNNs). Next, to obtain data for the purpose of training and testing of the predictive models, I developed a labeling software and used it to collect labeled image snippets from photoquadrats provided by biologists at Guana Tolomato Matanzas Estuarine Research Reserve (GTMNERR). With a set of university volunteers, 77,530 labeled images were collected. Using this collected dataset, I experimented with CNNs, such as LeNet-5, AlexNet, and VGG-16, to train effective models for predicting whether an input snippet is (1) unvegetated or not, (2) one of the five considered marsh grass species, or (3) either unvegetated or one of the five vegetated classes for a total of six classes. These models were applied in two variant frameworks: a single model framework with only the six-class CNN model, and a chained classifier that first classifies a point as vegetated or unvegetated and then classifies it as belonging to a specific species if it is vege-

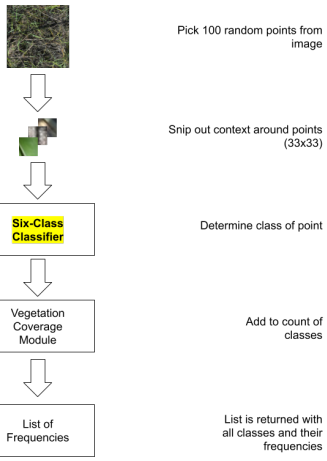
tated. We found the chained VGG-16 classifier to be the most accurate with a test accuracy of 84%. Finally, integrating the pre-trained CNN models in the back-end, I implemented *MarshCover*, a web-based system that automates the entire process of monitoring vegetation density.

CHAPTER 1

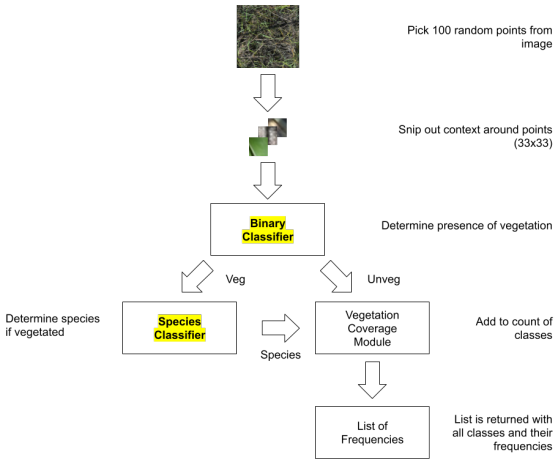
Introduction

Often in the ecological sciences, the gathering and processing of large datasets is essential for researchers to know how the Earth's habitats are changing over time. One such habitat is the inter-tidal marsh at Guana Tolomato Matanzas Estuarine Research Reserve (GTMNERR). These datasets have to be built up over time with manual laborers, which can be very intensive. **Given a set of high resolution images from GTMNERR, we set out to automate the process of estimating these densities using a set of convolutional neural networks.** This automated system would ideally remove volunteers from their vegetation monitoring program, allowing for this additional labor to engage in further conservation work at GTMNERR.

In order to automate this process we propose the creation of a framework with which to estimate the density of each species in the marsh present in an image. Because this was a novel problem that does not appear in the literature to our knowledge, there was no available dataset to start with. As such, using a series of images obtained from GTMNERR, **we generated a number of points as well as a labeling program for a set of volunteers to create our dataset**. From there, using the newly generated dataset, **we tested a number of convolutional neural network (CNN) models in a series of experiments including learning curve creation, 10-fold cross validation, and a comparison of the two pipelines we had devised for the task using the best models generated from the previous 10-fold cross validation**. After models and a pipeline were chosen, **a website was developed for use by GTMNERR and other re-**



(a) Single model framework.



(b) Two model framework.

Figure 1.1: A flowchart representing the major components of the vegetation density frameworks.

searchers with this task that takes an image, chooses 100 random pixels within the image, extracts the sub-image surrounding the pixel (33x33 sub-image), and passes the sub-images into a pipeline of classification modules. The results are then given from there. An illustration of this framework is given in Figure 1.1. There are two different possible frameworks we devised—a single model framework and a two model framework.

CHAPTER 2

Background and Existing Research

In order to automate the vegetation monitoring process, we first must find a way for a computer to identify points in a way similar to a human. Convolutional Neural Networks (CNNs) are one such method to do so. Below are three different neural networks we test on our vegetation monitoring task, followed by several related papers.

2.1 Convolutional Neural Networks

For this thesis, we will primarily be focusing on one primary type of neural network: **Convolutional Neural Networks (CNNs)**. These models are designed with one thing in mind: images. Modeled off of the mammalian visual cortex, CNNs use features such as lines, curves, and certain color patterns in an image in order to classify them into certain classes. These features are not set, however—they must be learned by the model through training on a training set. These features are expressed as values for groups of shared weights. These weights can be imagined as a window of a certain size that slides along the image. This window looks for the aforementioned features within the receptive field. Because this receptive field is replicated across the whole image, the models are able to detect these features anywhere in the image. This is one of the major advantages of CNNs: because weights are shared, we can train models using datasets where the object being detected can be anywhere in the image. This is ideal for image classification, as essential features to identify a class can be located anywhere within the image and still be detected by the model—in order for this to work with a more traditional

fully connected network, we would need those features to be present in every part of the image at least somewhere in the training set, making for a much less reliable system. The layer that looks for these patterns is known as a **convolutional layer**. Convolutional layers work by taking all the values within the receptive field and multiplying each by the value of the corresponding weight in the field; all these values are then added together, and the result is applied to the activation function for the layer, the result being stored in a single neuron in the next layer. Several different sets of distinct shared weights are used for each convolutional layer. These different sets of weights create several different "feature maps" from the input, which are used in further convolutional layers, allowing even more complex features to be detected. When looking for these more complex features, it is often important to reduce the number of smaller features that are being examined. This is achieved through another vital layer for CNNs known as the **pooling layer**. These layers come in several varieties such as max and average pooling. In these layers, all of the values of one feature map within a receptive field of a given size are considered, and some value (maximum, average, minimum, etc) is calculated from them.

2.1.1 LeNet-5

LeNet-5 is a simple CNN that was designed and implemented for optical character recognition on the MNIST dataset of images of 32x32x1[3]. Because our images are of a different size, we adopt a slight variation of the original LeNet-5 model. The details of the LeNet-5 model we adapt are included in Table 2.1. There are two differences between our model and the original model. One is the filter size used in the first convolutional layer (C1), which is 4x4, instead of 5x5 in standard LeNet-5. Because of this tweak in the first convolutional layer, it produces the output of size 28x28, same with the standard model, to ensure the rest of the hidden layers are the same as well. The other difference is the size of the output layer, for which

we have 2 values for the binary classification task, as opposed to 10 values for the single digits. From Table 2.1, we calculate the number of parameters to be learned is 59,964.

Table 2.1: Structure and number of parameters of each layer in LeNet-5 model for 31x31 image. Parameters have been tweaked to accommodate the different size of the input.

Layer Type	Size	Feature Maps	Kernel Size	Stride	Activation	Parameters
Input	31x31	3	—	—	—	—
Convolution	28x28	6	4x4	1	ReLU	294
Avg Pooling	14x14	6	2x2	2	ReLU	—
Convolution	10x10	16	5x5	2	ReLU	1216
Avg Pooling	5x5	16	2x2	2	ReLU	—
Convolution	1x1	120	5x5	1	ReLU	48120
Fully Connected	84	—	—	—	ReLU	10164
Output	2	—	—	—	Softmax	170
Total						59964

2.1.2 AlexNet

Similar to LeNet-5, AlexNet also is a convolutional neural network with numerous convolution and pooling layers[1]. It is a more complex model than LeNet-5, primarily attributed to the fact that AlexNet has more convolutional layers with more feature maps. However, the size of the input image originally used for AlexNet was 244x244 with three color channels [1]. Because our input image is only 31x31, we had to make adjustments to the stride and receptive field sizes of the layers in the network in order to fit the model to the size of the image. As such, while the number of feature maps and number of units in fully connected layers were kept the same, we adjusted kernel size and stride to work with the image size we have. The hidden layers, much like LeNet-5, consist of an alternating pattern of convolutions and pooling layers. However, for AlexNet, max pooling is used instead of average pooling [1]. As with our LeNet-5 variant, we resort to a different implementation of AlexNet because of the different input size. Table 2.2 provides the details of our adopted AlexNet model, where we see the differences are not only in the first

convolutional layer and output layer, but also in the intermediate layers so as to fit our input size. From the parameters provided in Table 2.2, we calculate the number of parameters to be 24,271,968. Clearly, there are more parameters for AlexNet than for LeNet-5; this is caused mostly by the much deeper feature space of AlexNet compared to its counterpart, potentially enabling AlexNet to solve more difficult learning problems.

Due to this sheer large number of trainable parameters, we observe extensive training time shown later in our experiment sections. Consequently, we also propose to experiment another model, a modified variant of AlexNet, for which we call *mAlexNet*, with the two fully connected layers of 100 neurons instead, resulting in about 3.4 million parameters, a very considerable decrease from AlexNet. We include such differences between AlexNet and mAlexNet in Table 2.2.

Table 2.2: Structure and number of parameters of each layer in AlexNet model for 31x31 image. Parameters have been tweaked to accommodate the different size of the input. Included are the number of trainable parameters for our downsized AlexNet model as well, separated by a "/"

Layer Type	Size	Feature Maps	Kernel Size	Stride	Activation	Parameters
Input	31x31	3	—	—	—	—
Convolution	28x28	96	4x4	1	ReLU	4704
Max Pooling	13x13	96	3x3	2	ReLU	—
Convolution	11x11	256	3x3	1	ReLU	221,440
Max Pooling	9x9	256	3x3	1	ReLU	—
Convolution	7x7	384	3x3	1	ReLU	885,120
Convolution	5x5	384	3x3	1	ReLU	1,327,488
Convolution	3x3	254	3x3	1	ReLU	878,078
Max Pooling	2x2	254	2x2	1	ReLU	—
Fully Connected	4096/100	—	—	—	ReLU	4,165,632/101,700
Fully Connected	4096/100	—	—	—	ReLU	16,781,312/10,100
Output	2	—	—	—	Softmax	8194/202
Total						24,271,968/3,428,832

2.1.3 VGG-16

VGG-16—so named for its 16 convolutional layers—continues our trend of deeper and more complex networks. VGG-16 was designed as a more generalized model. Made originally for the Google ImageNet challenge, it has been widely used in

various applications [5]. This is attributable to its depth and relative simplicity as well as its effectiveness. As such, it is often used to quickly train a familiar model on new image datasets for image identification problems [5]. While its architecture is much deeper than the previously discussed models, it is very similar in that it is composed of a series of alternating convolutional and max pooling layers. There is one such pooling layer every three convolutions, and the model concludes with a series of three fully connected layers. The model is shown for its original input size below in Figure 2.1. Because of its deeper nature, it is hoped that the VGG-16 model will be able to identify more complex features in the images, helping in identification.

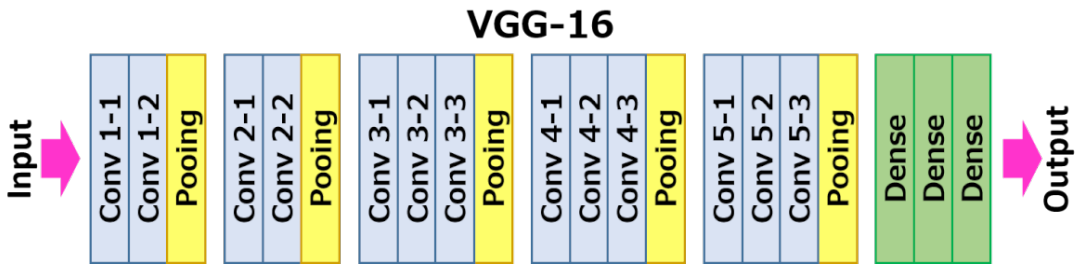


Figure 2.1: Diagram of original VGG-16 network. Note the 16 convolutional layers.

Because a VGG-16 model is included in the Keras applications library, instead of implementing the model ourselves, we have decided to work with the pre-included version of VGG-16 for simplicity's sake.

2.2 Related Work

Because of the high volume of images and the reliance on slow methodologies, this novel application could help reduce labor hours for essential scientific research. This application, however, is not without precedent, as there have been several similar applications published in the literature. These range from using semantic segmentation to measure vegetation density to using image recognition to determine coral density. These applications serve as not only inspiration, but also a

basis that helps us establish the applicability of the current technologies to our problem.

2.2.1 Vegetation Estimation in the Wild

Unlike rooted plants, plants like Spanish moss (known as epiphites) grow atop other plants, using them as a substrate on which to grow and feed. Because they are contained on the surfaces of other plants, it can be quite hard to estimate their population and biomass. One such plant, Spanish moss, is of particular concern here in the Southeast because of its importance to our environment. Found hanging from the branches of many trees, it has been very difficult to get accurate estimates of just how much Spanish moss is in an area—an important piece of information for understanding an area's habitat. To this end, the researchers in this paper want to use a convolutional neural network in order to perform **semantic segmentation** in which the model actually labels every pixel in the image. In this instance, the two class labels are "Spanish moss" and "other." Such a system allows for a very atomized approach to labeling. Instead of labeling an entire image or boxed segments, the model actually labels each individual pixel. This is particularly good for finding Spanish moss as we see because of the sparse and spread-out nature of the plant. [4]

Because each pixel must be labeled, it is a very time-consuming process, and the researchers had to use a special program in order to label multiple pixels at once. This labeling process requires many thousands of images of moss in order to properly train a model to segment out the plant. As such, due to the fact that we have only 811 photoquadrats provided by GTMNERR from which to obtain our data for training and testing, semantic segmentation is not a viable option for our problem. However, when attempting to estimate density of moss coverage, it is a system that allows for maximum accuracy.

The model for the task operates on a basic **encoder-decoder** principle in which a series of convolutional layers is used to reduce the image into a smaller coded set of feature maps before the decoder portion of the network expands the image back to its original size with each pixel assigned a class. After training, the model has a decent accuracy, able to label 87.4% of pixels correctly. This allows for a more automated way of estimating Spanish moss density in forestry imagery [4]

2.2.2 Counting and Describing Animals in Camera Traps

In order to count animals and track them in a non-obtrusive manner, many areas have begun to deploy camera traps, which are special cameras fitted with motion sensors. These sensors trigger the camera to take a picture whenever a creature moves in its field of view. Because of the large volume of images these traps produce, the labor-intensity of this process, like the previous one, prevents us from processing large amounts of data.

To solve this problem, a team of researchers created a classification model that is able to classify images taken in the Serengeti camera trap dataset [6]. Their approach is a simplistic one, using an image classification algorithm to both identify animals, count them, and determine their behavior as opposed to using object detection. These researchers used a variety of networks including many found in this proposal such as AlexNet, ResNet50, VGG16, and GoogLeNet. The use of pre-made models allows for rapid prototyping, which results in more models being tested. [6]

One interesting aspect of their labeling program was the use of **confidence intervals** to include human labelers—when the confidence of the model is too low when labeling an image, it sends the image to a user to manually label. This helps prevent the less confident results from negatively impacting the data; instead, the

machine and the user work together to process more data than the user alone, the hybrid approach giving the system the "best of both worlds" so to speak. [6]

2.2.3 More Recent Research

Since 2019, the interest in using machine learning to monitor changes in ecosystems and land use has been on the rise with several different papers being published since the beginning of this thesis project. Below are two of the most relevant of these recent publications, which both use drone footage to determine species composition or the presence/absence of vegetation.

In Yang *et al.* 2020, authors wanted to use deep learning in order to assess drone images of rice paddies for damage after storms or other disasters. [7] In order to do this, they used semantic image segmentation to identify rice paddies and, further, identify whether a rice paddy is exhibiting lodging (a phenomenon in which a plant has fallen over to one side, reducing its yield and making it more vulnerable to pests). Using two different encoder-decoder networks for semantic image segmentation, the researchers were able to identify paddies with rice lodging with an F-1 score of .78 for their validation data and at least 10 times faster than if it was done with the human eye. [7]

In Zhou & Klein 2020, the researchers use a dataset of forest imagery taken by a fleet of drones in order to train a deep learning classifier to identify patches of ground (represented as 32x32 snippets of larger drone images) as one of several classes (conifer, hardwood, shrub, reforested, or barren).[8] The authors were able to generate models that classified snippets with an accuracy of 94%. The resulting model was used in a function that split new drone images into a mosaic of non-overlapping 32x32 pixel snippets, which were each classified by the deep learning model. An image coverage for each class is then calculated by dividing the number of snippets classified as that class by the total number of snippets. This is a method

very similar to the one used later in this thesis document. [8]

CHAPTER 3

Dataset

To the best of our knowledge, our proposed approach to the marsh vegetation coverage problem is novel, and there was no data available to train our CNN models, the central predictive component in our method to solve the problem. As a result, we collected and created our own dataset.

3.1 Dataset Description and Construction

The images used for the creation of the dataset were provided by GTMNERR in St. Augustine, Florida. The image set from which our data was obtained consists of 811 one-square-meter photographs of marsh grass known as "photoquadrats." These photoquadrats, each of which is of dimensions ranging from 1653x1666 to 3268x2830, are used to obtain data in a per-square-meter fashion that allows for easy interpretation (such as density of a species per square meter). Because the program currently used by GTMNERR does not record the coordinates for the random pixels chosen, we were not able to use the work previously done by GTMNERR in the creation of our dataset. As such, we had to create our own set of points for labeling. To do this, we chose 100 random pixels from each image that fit the criteria of being at least 50 pixels away from the edge of the image, so, when passing an image into our algorithms, we would not have to pad the image in order for it to fit our input vector for the neural network—even when we begin to try different snippet sizes.

The data set consists of a set of 811 different photoquadrat images and a list of 77,630 points with the image the point is found in, the coordinates for the point,

and a label representing the class to which the point belongs.

After generating a set of pixels to use for the dataset, we then had to label each one for use as our ground truth. This was done with a group of volunteers who are trained university students. These students were chosen from the School of Computing and given a training course on the species in the marsh and how to operate the labeling program. The dataset consists of points that are classified into six different categories: *bare* or *unvegetated*; *Spartina*, which is the dominant species in the marsh; *Juncus*; *Batis*; *Avicennia*; and *Sarcocornia*. The latter four are less common than *Spartina*, and, as such, are less represented in the data.

An additional problem could come from the fact that, while four of the species have a single part of the plant that will be visible to the model (a blade/leaf), one class, *Avicennia*, being a tree, has several different plant parts that can be classified as Avicennia such as leaves, branches, and air roots—special roots sticking out of the ground, so the tree can breathe at low tide. This might impact the accuracy when trying to classify points from an image with Avicennia. As such, it might prove necessary to eventually split the Avicennia class into multiple smaller classes that would all be counted as Avicennia by the wrapper program.

When analyzing our data, we see the unvegetated class is the most prevalent with over 65.07% of the images, compared to 34.93% being vegetated. This is, however, to be expected, for every site that was sampled, while it never contained all species, contained bare areas. We also see the lack of data for *Sarcocornia*, which composes only 1.19% of the dataset. This is drastic compared to *Spartina*, *Batis*, and *Avicennia*, which have frequencies of 17.01%, 5.12%, and 7.60% respectively. Additionally, *Juncus* has a frequency of 4.00% This is a fairly even distribution. However, a lack of *Sarcocornia* could be problematic later on.

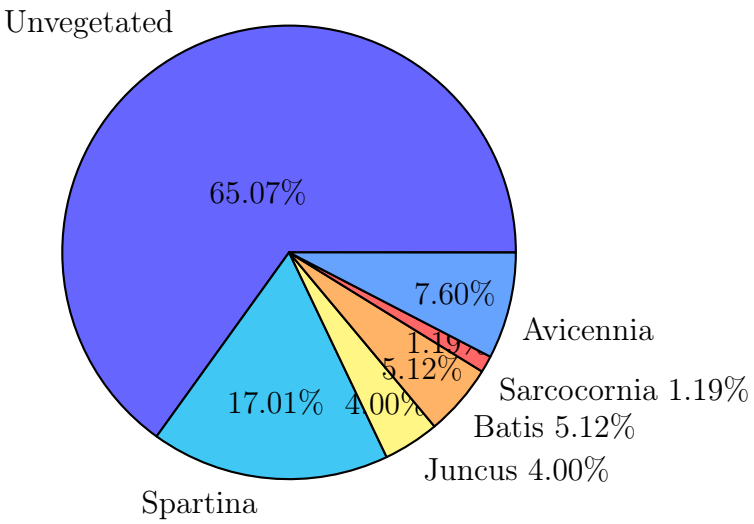


Figure 3.1: Distribution of the 77,630 points labeled for the MarshCover dataset.

3.2 Labeler Contributions

Because labelers joined at different times or only took part in certain phases of the labeling process, all four labelers submitted a different number of points. These relative contributions are given below in Figure 3.2. As can be seen, Labeler3 had the biggest contribution, contributing 30,000 points or 38.64% of the total dataset while Labeler2 had the smallest contribution with 6,564 points or only 8.46% of the dataset. These values will be critical later for determining the overall reliability of the dataset and its labelers.

3.3 Inter-Rater Reliability

While we now have a novel dataset labeled with six classes, we are unsure how accurate it is–accuracy in this context referring to the number of labels a person (or a model) labels correctly divided by the total number of labels. This accuracy compared to the expertly labeled data is known as the **inter-rater reliability**. In order to gauge this dataset's inter-rater reliability, we sent out a total of 1000 data points to each labeler. While there were a total of four labelers, only two of

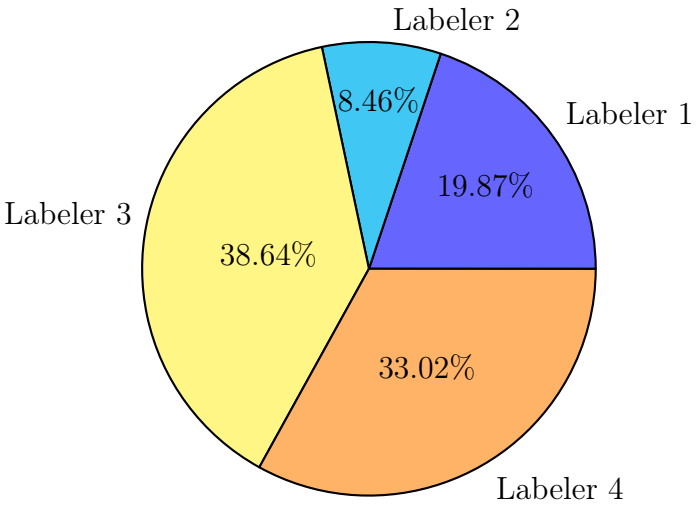


Figure 3.2: Graph depicting the number of labeled each labeler contributed to the dataset. Contributions ranged from 30,000 points to 6,564. The point contributions are 15,429; 6,564; 30,000; and 25,637 for labelers 1, 2, 3, and 4 respectively.

them returned any sort of labeled inter-rater reliability. The dataset was also sent to an expert at GTMNERR, who would act as the expert labeler against which all others will be compared. This expert was only used for this inter-rater reliability dataset. Note that, while the original plan was to use 1000 points, the expert labeler's dataset only has 657, limiting our scope.

Table 3.1: Table depicting the inter-rater accuracy for the two labelers who have inter-rater reliability data. Accuracies are broken down into class with the overall accuracy at the bottom The far right column lists the number of each class included in dataset according to the expert labeler.

Class	Labeler 1	Labeler 2	Total
Bare	85.96%	95.44%	285
Spartina	67.81%	63.01%	146
Juncus	—	—	0
Batis	94.74%	0.0%	152
Sarcocornia	5.88%	76.47%	17
Avicennia	96.49%	89.47%	57
Total	82.80%	65.14%	657

According to Table 3.1, there is a large discrepancy between our two respondents' accuracies. With a 17% difference between overall accuracy, we wanted to

look deeper. When looking at accuracies by class, Bare and *Spartina* have similar accuracies, not deviating by more than 10%; the same thing happens with *Avicennia*. However, for Labeler 1, their accuracy when labeling *Sarcocornia* is only 5.88% while *Batis* has an accuracy of 94.74%. However, with Labeler 2, this pattern is reversed and even more dramatic—*Batis* has an accuracy of 0% while *Sarcocornia* is labeled correctly 76.47% of the time. These results cause us to suspect that the labelers are confusing these two classes. This suspicion is confirmed by post-analysis interview with the labelers who both claimed that they often could not tell the two apart. However, there are different ways one can calculate inter-rater reliability.

3.3.1 Cohen's Kappa

The major problem with using simple rate of agreement for inter-rater reliability is that it cannot take into account the random aspect of labeling. For instance, given two different labelers, one might have a bias toward a more common class than one labeler (for instance, having a bias toward labeling a point as unvegetated). This could make it appear as though that labeler is more accurate while the other is the labeler who actually can discern between these different classes. As such, a measure needs to be used that can account for such bias and randomness.

This can be accounted for via **Cohen's Kappa**, which is a value that assesses the reliability of a labeler, taking into account those biases. The value is easily calculated as:

$$\kappa = 1 - \frac{1 - p_o}{1 - p_e}$$

where p_0 is equal to the rate of agreement:

$$p_0 = \frac{\text{number of points both labelers labeled the same}}{\text{total number of points}}$$

and p_e is equal to:

$$p_e = \frac{1}{N^2} \sum_k n_{k1} n_{k2}$$

where N is equal to the number of labels in the dataset, k is the number of classes in the dataset, and n_{ki}, where i is either 1 or 2, is the number of labels of the given class k for the first or second labeler.

Table 3.2: Table showing the Cohen's Kappas between all combinations of labeler and expert. All values are between –1 and 1.

	Labeler 1	Labeler 2	Expert
Labeler 1	1.00	0.4866	0.7516
Labeler 2	0.4866	1.00	0.5019
Expert	0.7516	0.5019	1.00

Applying the calculations above, we obtain the kappa values presented in Table 3.2.

There are several points of interest in this table. For instance, we see that the kappa value between our two labelers is a low value of 0.4866, which is a rate of agreement that is considered to be "moderate." [2] This moderate rate of agreement, however, is the lowest value in the table with both labelers being in better agreement with the expert than with one another. While Labeler2 may only have "moderate" agreement with the expert, Labeler1 is much better with a kappa value of 0.7516, which is considered "substantial." This is great news considering that, according to figure 3.2, Labeler1 contributes a large portion (19.87%) of the dataset while Labeler2 only contributes 8.46% of the dataset. By taking the relative contributions to the dataset of the two labelers, we can calculate an average for the kappa value weighted with respect to each labeler's given contribution to

the dataset. This measure was devised by us as a way to take assessed labelers' contributions to the dataset into account to better estimate the agreement of the dataset used for training. We determined this by summing the total number of points our two labelers who returned inter-rater reliability data contributed to the dataset (21,993 points) and calculating each labeler's percent contribution to this smaller set of points. These calculated percent contributions are then multiplied by the kappa value between the respective labeler and the expert before being summed together.

The data collection process has taken 18 months. Despite the above difficulties, our models eventually produced acceptable prediction results.

CHAPTER 4

Experimental Setup

For these experiments, we used an online tool provided by Google called Colab. Colab is a cloud-based Jupyter Notebook that is connected to GPU and TPU (tensor processing unit) servers owned by Google that can be used for training machine learning models.

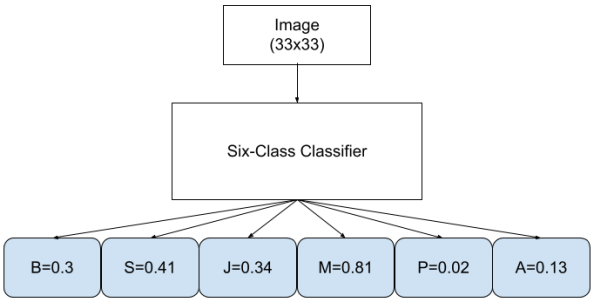
Due to limitations of Google Colab (limited run-time duration of 24 hours being the most challenging), the run-times had to be reset every day. Additionally, because the resources provided by Google Colab are dynamically allocated, it is very difficult to determine the exact resources used to train any given models. This fact also imparts several more challenges. For instance, due to the ease with which available computing resources can change, training times and execution times can vary wildly from day to day; this becomes especially apparent when an account has been executing continuously for more than one run-time duration, as the site will gradually reduce the amount of computing power available to you.

Before Colab was used for the below experiments, model training was performed using the School of Computing (SoC) GPU servers while we only had around 55k points and were only working on binary classifiers. While these allowed for the use of GPUs for model training, they were still very limited in their abilities. As such, as the size of the dataset and the number of total models that needed to be trained increased, we found the SoC servers were not able to deliver results in a timely fashion. This is why Google Colab, even with its problems, along with the fact that all saved models are saved to Google Drive, was used for all of the below experiments.

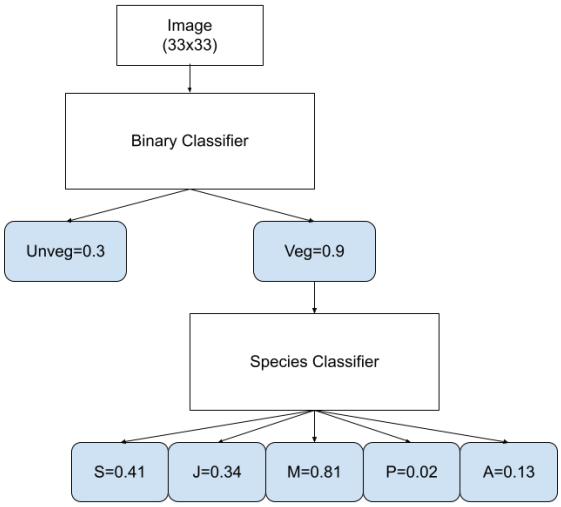
We ran a total of three different types of experiments on the models: **learning curves** to establish the behavior of the models given the dataset, **10-fold cross-validations** to find a model to insert into the model pipelines, and **top-n evaluation** of the single classifier and chained classifier pipelines in order to establish which pipeline to utilize in the framework. The **single classifier** pipeline involves a single model with an input of a snippet and an output of 6 classes. The **chained classifier**, on the other hand, is a grouping of two models into a decision tree-like structure with each non-leaf node being one of two models. A depiction of these two model pipelines is shown below in Figure 4.1. When the snippet is first inserted into the pipeline, it is passed into a binary classifier that determines the presence or absence of vegetation. At this point, if it is determined that there is no vegetation, the result is recorded—if the point at the center of the snippet is considered to be vegetated, however, it is passed into a second model that classifies the species, and the result is recorded.

Additionally, while we explored the potential of using five different models (LeNet-5, AlexNet, VGG-16, ResNet-50, and GoogLeNet), due to the limited computing resources, we decided to stick to the three smallest models (LeNet-5, AlexNet, and VGG-16).

All models are trained over 120 epochs with a learning rate of 0.0001 using stochastic gradient descent and a batch size of one. The learning-curve was determined for a small grid-search of potential learning rates $10^{-1}, 10^{-2}, ...10^{-5}$ on the LeNet model and was found to give the best result. The number of epochs was also settled via a grid search on 30, 60, 90, and 120 epochs, and 120 epochs was chosen, as it was found to be the maximum number of epochs for a model's loss to converge.



(a) Single classifier model.



(b) Chained classifier model.

Figure 4.1: Diagram of our two different model pipelines—single classifier and chained classifier. Letters in output correspond to labels (B=Unvegetated, S=Spartina, J=Juncus, M=Batis, P=Sarcocornia, A=Avicennia). The numbers in the cells represent a potential output.

4.1 Learning Curves

In order to evaluate the behavior of the models being trained, a series of learning curves was generated for LeNet-5 and AlexNet models on the binary classification problem.

4.1.1 Splitting the Data for Learning Curves

The setup for these curves is very simple. After loading in the data, a stratified train-test split is performed on all the data from all four labelers (77,530 snippets) to obtain a training and test set. A seed was used in this and all further splitting processes in order to ensure the split is the same every time. For this initial split, the training set received 80% of the total data while the test set received the remaining 20%. This test set is then set aside, as it is unused in the learning curves. However, taking the training set, we further split the set (in a stratified manner) into 100 different buckets of similar size.

Because the data contains 6 classes while the models being trained contain only 2 output classes (vegetated and unvegetated), all labels other than bare had to be combined into another vegetated label. For simplicity's sake, these classes are called vegetated and unvegetated. In order to keep the training and test sets the same for all experiments, the five vegetation classes are not combined until after all splitting of the dataset is complete.

4.1.2 Running the Learning Curves

With the buckets prepared, the training and validation sets can be constructed. The sets were constructed by taking the buckets and assigning them to either training or validation set. There were a total of 27 models trained for each learning curve with the discerning factor being the number of buckets used for the training

set.

The validation set is constructed using a method know as **holdout**. In holdout, we take all buckets that were not used to create the training set and use them to create the validation set.

Each of the 27 models has a different number of buckets being used for training set. The number of buckets used start at one with the other 99 being used for after which the number of buckets being used for training increments by one until 20 buckets are being used for training. At this point, the number of buckets used for training is incremented by 10 until we are using 90 buckets for training and 10 for validation. This process is repeated either five or ten times.

4.2 10-Fold Cross Validations

In order to find the best model or models to use in our framework, we executed a total of nine 10-fold cross validations (CVs), one for each combination of topology (LeNet-5, AlexNet, and VGG) and number of output classes (binary, species, and six-class). In order to do this, the dataset was split in a stratified manner on all six classes into a training and test set. The training and test set are allotted 80% and 20% of the dataset respectively. After obtaining the training set, we then split it into 10 buckets, each containing 10% of the training set. In order to perform the 10-fold CV from here, the 10 buckets from the training set are iterated through, each one being used as a validation set for the model being trained, the other nine buckets being combined into a training set. After training a fresh model for each bucket, the results are compared, and the model with the best performance on its validation set is chosen for each of the nine cross-validations.

4.3 Single Classifier vs. Chained Classifier

Now that the best models from the 10-fold CVs have been identified, we must find the best models for the framework. We have two different options for how the framework will operate: a **single classifier**, which is a single model with six output classes, and a **chained classifier**, in which a snippet is first passed into a binary classifier and then, if the binary classifies the snippet as vegetated, it is passed into a species classifier. In order to evaluate these two different pipelines, we obtained the top-n accuracy of each of the six produced pipelines (three single classifiers and three chained classifiers).

4.3.1 Construction of Chained Classifier Vector

In order to calculate the top-n accuracy, we first need an output vector on which to run the computations. Because the single classifier pipeline produces a single vector of six classes, it is already able to be used to calculate the top-n accuracy. However, being a two model pipeline, the output is different for the chained classifier. As such, we must construct a vector for the chained classifier pipeline to calculate the top-n.

For a chained classifier model M, let s be a snippet fed to M, $< u_s, v_s >$ the binary output vector produced by the binary classifier module, $< s, j, b, c, a >$ the multi-ary output vector by the species module. Clearly, the probability of s being unvegetated by model M is $p_M(s = Unv) = \frac{u}{u+v}$, and the probability of s being vegetated by model M is $p_M(s = Veg) = \frac{v}{u+v}$. Now, we define the probabilities of s being one of the five species, Spartina, Juncus, Batis, Sarcocornia, and Avicennia, to be $p_M(s = Spa) = p_M(s = Veg) * \frac{s}{s+j+b+c+a}$, $p_M(s = Jun) = p_M(s = Veg) * \frac{j}{s+j+b+c+a}$, $p_M(s = Bat) = p_M(s = Veg) * \frac{b}{s+j+b+c+a}$, $p_M(s = Sarc) = p_M(s = Veg) * \frac{c}{s+j+b+c+a}$, and $p_M(s = Avi) = p_M(s = Veg) * \frac{a}{s+j+b+c+a}$,

respectively.

Therefore, for any snippet s sent to a chained classifier model M, we obtain a probability distribution over the six labels of Unvegetated, Spartina, Juncus, Batis, Sarcocornia, and Avicennia, and such distribution, denoted by $P_M(s)$, is given by the vector $< p_M(s = Unv), p_M(s = Spa), p_M(s = Jun), p_M(s = Bat), p_M(s = Sarc), p_M(s = Avi) >$. Such probability distribution $P_M(s)$ is straightforward for the single classifier models.

4.3.2 Top-n

Now that $P_M(s)$ has been defined (for the single classifier pipeline with the model output and the chained classifier pipeline with the above process), the top-n must be obtained for all six pipelines. The determination of top-n is fairly simple.

Given a test set, T, of n snippets $\{s_1, ..., s_n\}$, each s_i labeled by a ground-truth g_i, and a model (single or chained classifier), M. We first obtain, for snippet s_i, a vector $\{l_{i1}, ..., l_{i6}\}$ of the six labels ranked by their corresponding probabilities obtained in P. We can then define the top-n accuracy, $TopAccuracy(M, T, n)$, where $1 \leq n \leq 6$, as:

$$\frac{|\{s_i \in T : g_i \in \{l_{i1}, ..., l_{in}\}\}|}{|T|} \tag{4.1}$$

Top-n accuracy is not usually a concern for the end-user, as they only see the most model's top-1 result (the class with the highest confidence). However, top-n accuracy is often used in machine learning competitions to rank models—this allows for models to be ranked in the unlikely occurrence of a tie between two models in top-1 accuracy or for comparing models with a very large number of output classes.

These top-n results will finally be used to determine which model pipeline will be used in the final program.

Note that, unless otherwise specified, any reference to "accuracy" in this paper

is referring to top-1 accuracy.

CHAPTER 5

Results and Analysis

Before we can implement a final product with our models, we must first identify which models perform best and implement them. Once the models are identified, they will be inserted into the framework.

Also note that, due to their complexity and the limited computing power available to us, ResNet-50 and GoogLeNet were not used for these experiments. Additionally, the size of the fully connected layers was reduced from 4096 to 100, drastically cutting training time for AlexNet as well.

Training time for all models varied widely due to the aforementioned dynamic resource allocation. However, on average, when training on the full training set (made up 90% of the 80% split of the full dataset), we find that an epoch of training for LeNet-5 takes 137 seconds (totaling 16,440 seconds per model or approximately 4.5 hours). For AlexNet, this takes 307 seconds on average (36,840 seconds or 10.2 hours per model) and 558 seconds for VGG-16 (66,960 seconds or 18.6 hours per model).

5.1 Learning Curves

In Figure 5.1 we show the learning curves for LeNet-5 and AlexNet models, where we see, for both models, an overall trend of training and validation accuracies increasing as more experiences are given to the learning algorithms.

For both CNNs, it is clear that the training subsets and the validation subsets are *representative* for our binary classification tasks. This is so because all four accuracies only show a few noisy fluctuations, which could be mitigated with more

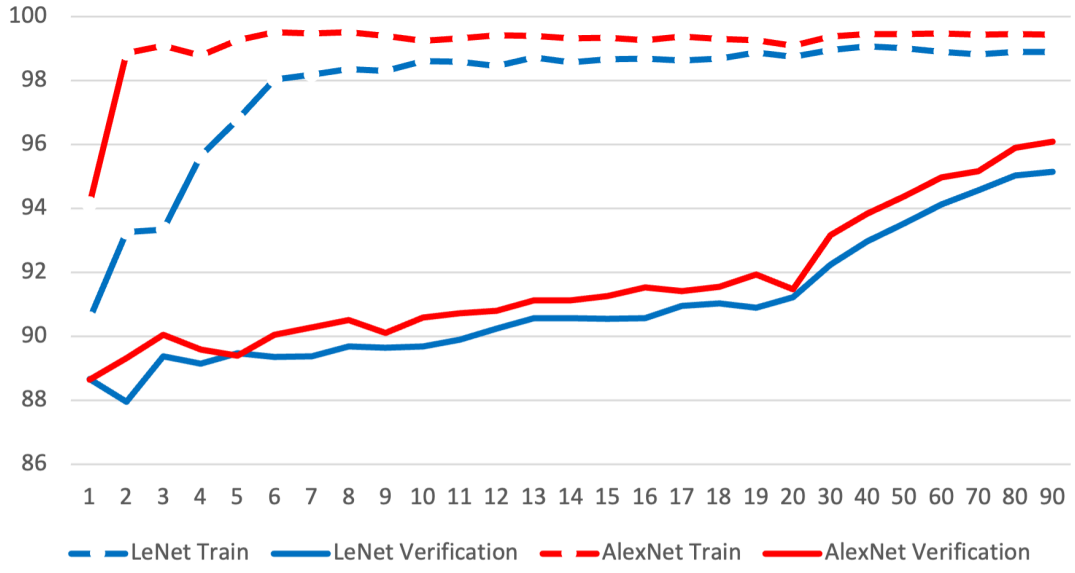


Figure 5.1: Learning curves for LeNet-5 and AlexNet

iterations in the experiments. Another reason is that, towards the end of the curves, the gap between the training and validation accuracies for both models is getting reduced with more training instances used for learning.

For LeNet-5, when only 1% of training set is used to train a model, the accuracy on such training subset is 90.54% and that on the remaining 99% validation subset is 88.66%. When the training set makes up 90% of the buckets, the model has a training accuracy of 98.89% while the validation set has an accuracy of 95.14%. While the training set quickly grows to over 98%, plateauing before reaching 10 buckets in the training set, the validation set continues to increase in accuracy up to the end of the curve, indicating that more accuracy can be obtained from the model with additional data.

Let us turn to the results of the more complex model AlexNet, which we first see *higher* accuracies for all but one size of training subset, 2%. It starts with 94.14% and 88.65%, and finishes with 99.44% and 96.10%, on training and validation, respectively. Notably, at 90% size of training subset, AlexNet training and validation are 0.55% and 0.96% better than LeNet-5 respectively. Throughout the

learning, it shows the more complex model of AlexNet pays off in the accuracies. Furthermore, the steady improvement of AlexNet validation accuracy, especially near the end of the learning curve, indicates the model is not yet overfitting and more training examples highly likely could revamp it further.

Because of the continued underfitting of the AlexNet models, we decided to increase the complexity of the models further by using VGG-16 in the 10-fold cross validations. With the inclusion of VGG, we are hoping to see a continued improvement in the accuracies.

5.2 10-Fold Cross Validations

In order to find the best model or models to use in MarshCover, we executed a total of nine 10-fold cross validations (CVs). We executed 10-fold CVs for each combination of topology (LeNet-5, AlexNet, and VGG) and number of output classes (binary, species, and six-class). In order to do this, the dataset was split in a stratified manner on all six classes into a training and test set. The training and test set are allotted 80% and 20% of the dataset respectively. After obtaining the training set, we then split it into 10 buckets, each containing 10% of the training set. In order to perform the 10-fold CV from here, the 10 buckets from the training set are iterated through, each one being used as a validation set for the model being trained, the other nine buckets being combined into a training set. After training a fresh model for each bucket, the results are compared, and the model with the best performance on its validation set is chosen for each of the nine cross-validations. A summary of the results for this are contained in Table 5.1. Because of the large volume of data, these results are a summary, including only the best results from each cross-validation. **As was expected, the addition of the added complexity in VGG-16 compared to LeNet-5 and AlexNet improved binary and species classifiers by 0.8% for binary and almost**

3% for species classification while there was no change for six-class.

Table 5.1: Summary table of the nine 10-fold cross-validations. Below are the training, validation, and test results for the best model from each 10-fold CV. Bolded results are the models with the highest accuracy for each output type based on test set. All models were evaluated using top-1 accuracy.

	LeNet-5			AlexNet			VGG-16		
	Training	Validation	Test	Training	Validation	Test	Training	Validation	Test
Binary	98.18	90.05	89.3	98.94	90.37	90.52	**99.21**	**91.14**	**90.76**
Species	76.98	68.85	67.31	96.04	72.84	71.87	**89.09**	**75.81**	**75.37**
Six-Class	**84.5**	**82.58**	**82.35**	96.25	82.9	82.31	97.87	82.05	81.32

According to these results, the VGG models are usually the best with validation accuracies surpassing LeNet-5 and AlexNet for all output types other than the six-class where AlexNet is best.Looking at the results from the best 10-fold models, it becomes clear that the LeNet and AlexNet models were underfitting the data for binary classification, as the validation and training accuracy continue to increase with the addition of more complex models (increasing depth between LeNet-5, AlexNet, and VGG-16). This pattern continues with the species classifier, the validation and training accuracy increasing between each model, the test accuracy even rising by almost a full 4% between AlexNet and VGG-16. This pattern, however, does not hold up when we observe the six-class models. This is evident from LeNet-5's position as the best-performing six-class model. The differences, however, are fairly small. While training accuracy continues to increase with model complexity, the test set accuracy seems to be stuck around 82%. We believe this is likely due to the high proportion of unvegetated data points. As such, this would likely be corrected with the addition of more points of the other five classes—either through data augmentation such as a GAN or through the recruitment of more labelers to continue to expand the dataset.

5.3 Single Classifier vs. Chained Classifier

Now that the best models have been identified, we must compare them in their respective pipelines to one another in the manner discussed in the previous chapter in order to figure out which pipeline to utilize in the final product.

The top-n results are listed in Table 5.2. As can be seen, VGG has the best top-1 accuracy when comparing chained classifier models. However, AlexNet has a better test set accuracy when comparing the single classifier models. These patterns continue beyond top-1 accuracy to encompass all top-n values (1-5). Additionally, VGG's chained classifier pipeline has the best top-n accuracy when comparing all six pipelines, including LeNet-5's single-shot model. As such, **VGG chained classifier will be the pipeline to be used in the web-based MarshCover prototype, which is discussed in detail in the next chapter.**

Table 5.2: Top-n results for each of the best pipelines. Single classifier pipelines consist of a single model with 6 output classes while the chained classifier pipelines consist of a binary classifier and species classifier chained together.

	LeNet-5		AlexNet		VGG-16	
n	Single	Chained	Single	Chained	Single	Chained
1	82.35	80.49	82.31	82.86	81.32	**84**
2	92.52	90.94	91.65	92.87	90.55	93.48
3	96.66	95.25	95.3	96.16	94.72	96.45
4	98.69	97.22	97.4	97.87	97.33	97.8
5	99.57	98.18	98.6	99.05	98.67	98.76
6	100	100	100	100	100	100

Despite its larger size and potentially longer run-time, with an increase of over 1.5%, we believe it to be worth it to use the deeper VGG models in a chained fashion than use the smaller LeNet-5 single classifier model.

CHAPTER 6

MashCover: A Web-Based Platform for Vegetation Density Calculation

In this section, we discuss MarshCover, the web-based system that we implemented to measure vegetation coverage in marsh grass photographs. In Figure 6.1 we show an overview of MarshCover that includes front-end and back-end and how data flow between the two.

The website was created using the Django framework. In addition to Django, we used Tensorflow with Keras to create, load, and use all deep learning models. Additionally, Scikit-Image was used to load and crop the images, and matplot-lib was used to generate the pie-chart that is presented after all snippets have been analyzed.

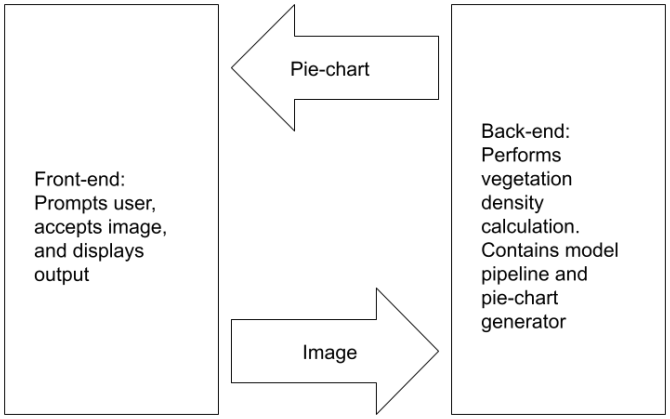


Figure 6.1: Diagram depicting the interaction between the front-end and back-end.

6.1 Front-End

The front-end for this website is very simple. When accessing the website, we find a homepage, an about section, a contact section, and the webapp. We will focus on the webapp here.

When accessing the webapp, the user is presented with a prompt to upload an image. This page is shown below in Figure 6.2. In order to prevent the server from being overwhelmed, the user is only able to upload one image at a time. After the image is uploaded, the image is sent to the back-end, which will be discussed in the next section. In addition to the image, the user also enters the number of random snippets they want trimmed from the image.

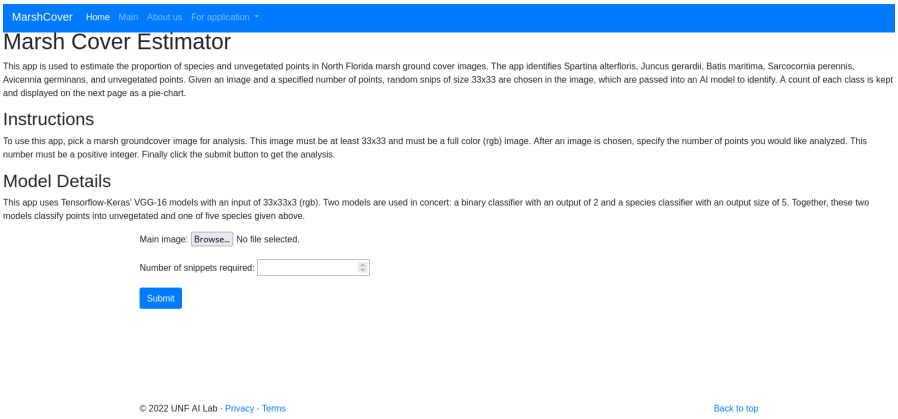


Figure 6.2: Screenshot of opening page of app where user uploads image for evaluation.

After the image has been analyzed in the back-end, a pie-chart is returned to the user and displayed. This pie-chart records the number of each class that was found when analyzing the image. Additionally, the user will be able to return to the beginning of the app to upload another image. A screenshot of the page is included in Figure 6.3.

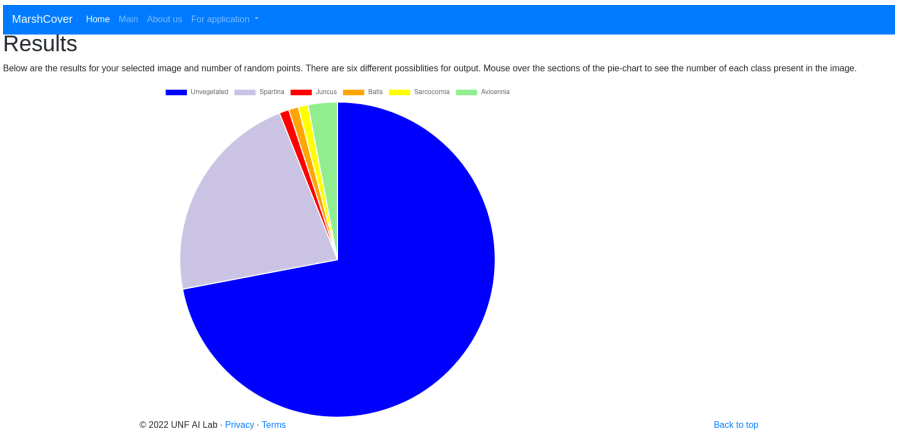


Figure 6.3: Screenshot of output showing outputted pie-chart for an image.

6.2 Back-End

The back-end for the MarshCover webapp is fairly straightforward and illustrated previously in Figure 1.1.

The image that was uploaded by the user is first saved to the server before the length and width is taken. Once the length and width are taken, 100 random pixels are selected within a 16x16 pixel border (exclusive). After the pixels of interest are selected, 33x33 pixel snippets are taken with these pixels in the center. At this point, each snippet is passed into the binary classifier. if they are classified as unvegetated, it is recorded as such. However, if the snippet is classified as vegetated, it is passed into the second model, which classifies it into a species before recording it. After all 100 snippets have been classified, a pie-chart is generated with matplot-lib before being returned to the front.

CHAPTER 7

Conclusions and Future Work

For this thesis, we set out to automate the marsh vegetation monitoring program at GTMNERR. To do this, we began by creating a dataset from already acquired marsh grass photoquadrats. This dataset was labeled using an in-house labeling program made in Java. After the dataset had been labeled, it was then used to generate a number of models using 10-fold cross validation. These models were tested in their various pipelines to determine which would be best for the final product. We found the chained classifier VGG-16 pipeline to be the best suited for the framework. We have now produced a final webapp using this pipeline that takes an image and estimates the percent cover using the same method as GTMNERR.

While a lot of work has been done to automate this process, many things could be done to improve the framework and its components.

7.1 Future Work

In order to improve performance of the produced models, it would be prudent to continue to expand the dataset. Because *Sarcocornia* still has less than 1000 total points, images containing this class should be the main priority for continued expansion of the dataset. Because *Juncus* is also below 5% of the dataset, it could be valuable to focus on images containing Juncus as well.

Though the first draft of the MarshCover webapp has been completed, there are still several more things that can be done to improve it. To start, the website currently is local to the developers, but will be made available to the general public.

Also, there has not yet been a usability study to assess its ease of use. Additionally, at the moment, only one image can be passed into the app at a time. While this has originally been done as a way to prevent the user from overwhelming the server, it would speed up the process quite a bit for the GTMNERR team. Additionally, adding the ability to upload multiple images at once would allow us to provide an output file in an economical way—because of the need to analyze one image at a time, each image would have its own output file, making for a cumbersome process for workers to append them all together. However, with the ability to upload multiple files, an output file could be created for download.

Recently, GTMNERR has begun to propose the gradual transition away from traditional photoquadrat vegetation monitoring toward the use of drones to take groundcover images. While this transition will greatly increase the efficiency of collecting images, the use of a new camera with different resolution at a different scale could very well impact the accuracy of the image classification. As such, a new set of classifiers might need to be trained with these new images in mind. However, because we have trained these models already on a similar set of images, we can use **transfer learning** to retrain the models on the new set of images, using the current models as a starting point instead of a new random initialization of weights. This will allow us to "tweak" the models instead of having to train them again from scratch. This would cut down on the total amount of data needed for the retraining task, which would make dataset creation a much less labor-intensive task for an additional drone-based dataset.

This is, however, not the only image processing task at GTMNERR. For instance, there are many other habitats where this image processing task could be put to use. As such, while this task has been automated, there is still much left to be done for the reserve.

REFERENCES

[1] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (2012), pp. 1097–1105.

[2] Landis, J. R., and Koch, G. G. The measurement of observer agreement for categorical data. biometrics (1977), 159–174.

[3] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. Proceedings of the IEEE 86, 11 (1998), 2278–2324.

[4] Mihail, R. P., Cook, W. I., Griffin, B. M., Uyeno, T. A., and Anderson, C. D. Vegetation density estimation in the wild. In Proceedings of the ACMSE 2018 Conference (2018), ACM, p. 9.

[5] Simonyan, K., and Zisserman, A. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014).

[6] Tabak, M. A., Norouzzadeh, M. S., Wolfson, D. W., Sweeney, S. J., VerCauteren, K. C., Snow, N. P., Halseth, J. M., Di Salvo, P. A., Lewis, J. S., White, M. D., et al. Machine learning to classify animal species in camera trap images: Applications in ecology. Methods in Ecology and Evolution 10, 4 (2019), 585–590.

[7] Yang, M.-D., Tseng, H.-H., Hsu, Y.-C., and Tsai, H. P. Semantic segmentation using deep learning with vegetation indices for rice lodging identification in multi-date uav visible images. Remote Sensing 12, 4 (2020), 633.

[8] Zhou, W., and Klein, L. Monitoring the impact of wildfires on tree species with deep learning. arXiv preprint arXiv:2011.02514 (2020).