8-2024

Hyperspectral Image Classification of Bacteria Using a Deep Convolutional Neural Network

Bruce S. Vogelsberg Jr
Clemson University, bvogels@clemson.edu

8-2024

Hyperspectral Image Classification of Bacteria Using a Deep Convolutional Neural Network

Bruce S. Vogelsberg Jr
Clemson University, bvogels@clemson.edu

HYPERSPECTRAL IMAGE CLASSIFICATION OF BACTERIA USING A DEEP CONVOLUTIONAL NEURAL NETWORK

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Bioengineering

---

by
Bruce Stewart Vogelsberg
August 2024

Accepted by:
Dr. Bruce Z. Gao, Committee Chair
Dr. David Karig
Dr. Lucas Schmidt
Dr. Tong Ye

ABSTRACT

Hyperspectral imaging is a non-invasive imaging method capable of collecting both spatial and spectral information. However, because of the large volume of data collected, much of it is redundant or not useful for classification. Deep learning is a subset of machine learning that uses artificial neurons in a multilayered structure to learn representations from data. One of the main advantages of deep learning is the powerful feature extraction capabilities, which allow the model to learn both high- and low-level features. Convolutional neural networks are a type of deep learning model that have alternating convolutional and pooling layers capable of extracting features while also downsampling the input image to remove unimportant features, retain important features and reduce the computational load of the deep learning model.

CNNs can be combined with hyperspectral imaging to rapidly process the large amounts of data while removing unnecessary features. Thus, CNNs can be used to classify different hyperspectral images of bacteria and determine which bacteria are present in the image based on their spectral responses. From the output of the network, false color images can be generated from the input hyperspectral images to enhance visualization and provide spatial information.

In this thesis, we explained the goal of this project to design a deep learning model capable of processing hyperspectral images of bacteria and outputting classifications with high accuracy. We described a detailed procedure of the development of a three-layer convolutional neural network capable of generating predictions with

approximately 96% accuracy and producing false color images of the hyperspectral inputs.

DEDICATION

I would like to dedicate this work to my family and friends who have supported me throughout my undergraduate and graduate career. Everyone has done so much to support me and guide me through this phase of my life. My mother, Margot, has been the biggest supporter of me not only through college, but throughout my entire life. Nothing I could say could express how grateful I am to have you in my life. My father, Bruce, has loved and supported me throughout this program and I will always be grateful for him being there. My stepdad, Bill, has not only supported me since the day he met my mother, but has provided help and support in ways that most people would not expect from a stepparent, and I will always be thankful that he met my mother. My bonus sisters, Megan and Erin, have been such a wonderful addition to my life and have always been excited to hear about everything I was working on during my graduate degree. The both of you are the best sisters that someone could ask for. My grandmother and grandfather, Gram and Bill, I know that if both of you could be here to see me finish this degree, both of you would and you would both be so proud of me. My grandparents, aunts, uncles, and cousins who are too numerous to list here have always been supportive and excited to learn about what I was working on because they all realized how much bioengineers can really help people once I began my program during my undergraduate career. And finally, to all my friends I met during my time at Clemson, I will always considered myself lucky to have met such a great group of people that have always pushed me to be a better person.

ACKNOWLEDGMENTS

I would first like to thank my advisor, Dr. Bruce Gao, for his help and support during my time in the graduate program. He gave me new opportunities that I never dreamed of before joining his lab and I will always be grateful that I made the choice to have him as my advisor. I would also like to thank my committee members Dr. Lucas Schmidt, Dr. David Karig, and Dr. Tong Ye.

I would also like to thank my lab members Reece Fratus, Adam Baker, Dr. Carol Zhang, Lidadi Agbomi, Taylor Seawell, Wesley Nichols, Shantanu Kore, and Calvin Chernyatinskiy for their support, guidance, and friendship throughout these past few years.

Finally, I would like to thank the Bioengineering department for helping me develop my skills as a bioengineer and prepare me to join the work force. I would like to give a special thanks to Dr. Laura Tam, Dr. Hobey Tam, Dr. Jordon Gilmore, and Dr. Delphine Dean for providing guidance and helping me with problems regarding my research and life in general.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

List of Figures (Continued)

Figure

Page

CHAPTER ONE

INTRODUCTION AND BACKGROUND

Unmanned underwater vehicles are submerged underwater for extended periods of time. Once a substrate or structure is exposed to seawater, a biofouling process immediately starts at a micro-scale and expands to a macro-scale over time [1]. This biofouling process results in waterborne organisms, such as barnacles, beginning to grow which can result in unwanted effects such as drag occurring [2]. This unwanted effect requires the unmanned underwater vehicles to be brought back to be cleaned more often than they should need to be which undermines the idea of them being considered unmanned. A potential solution to this problem is growing a single bacterium or a combination of bacteria on the outside of these unmanned underwater vehicles to create a smooth, stable biofilm comparable to a hydraulically smooth surface. However, this solution requires being able to determine the presence of bacteria in a flow setup with conditions comparable to what the unmanned underwater vehicle would typically experience. Using hyperspectral imaging and deep learning, this can be achieved.

1.1 Hyperspectral Imaging

1.1.1 Hyperspectral Imaging Introduction

Hyperspectral imaging is a noninvasive imaging modality that images a target sample over a range of wavelengths to collect a 3-dimensional dataset containing both spatial (X and Y) and spectral (λ) information to produce a dataset with dimensions of (X, Y, λ) which typically is referred to as a data cube or hypercube [3, 4]. As seen in

Figure 1.1, this can be compared to other imaging methods such as microscopy, optical

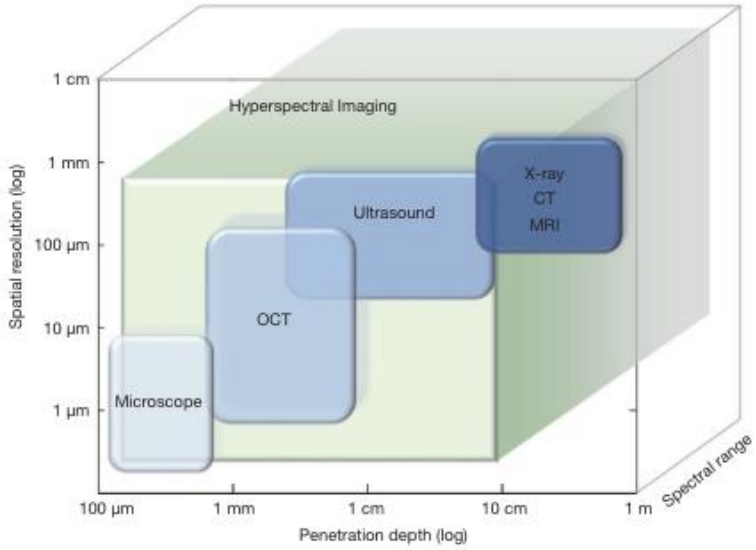

Figure 1.1 Diagram from Y. Zhang et al. [4] illustrating the differences between different imaging modalities characteristics.

coherence tomography (OCT), ultrasound, x-ray, computed tomography (CT), and magnetic resonance imaging (MRI). Hyperspectral imaging, however, comes with a unique dimension compared to the other imaging methods. While standard imaging methods tend to focus on penetration depth and spatial resolution, hyperspectral imaging also deals with spectral range.

1.2 Deep Learning

Deep learning is a subset of AI that loosely mimics human neurons that consists of interconnected neurons or nodes in a multilayered structure to learn representations from data such as video or images to predict what the output is based on the input [5-7]. Deep learning uses powerful feature extraction methods by transforming the input at a low level (starting with the raw input) to a higher, more abstract level [8]. These higher layers of representation are types of patterns that a human would be unable to notice such

as specific combinations of shapes, patterns, edges, and/or colors. These predictions could be predicting gene expression, predicting the activation of drug molecules, or the prediction of a single bacterium or combination of bacteria based on reflected wavelengths [8].

Deep learning is loosely based on the biological neuron. The neuron's dendrites accept some weighted input, the nucleus decides if the information is relevant, and will pass it down through the axon if it is onto the next neuron through the synapse in which the process repeats [9]. A mathematical model of the biological neuron was created Warren McCulloch and Walter Pitts called a perceptron or McCulloch-Pitts neuron as seen in Figure 1.2. This model works similarly to the biological as it accepts some

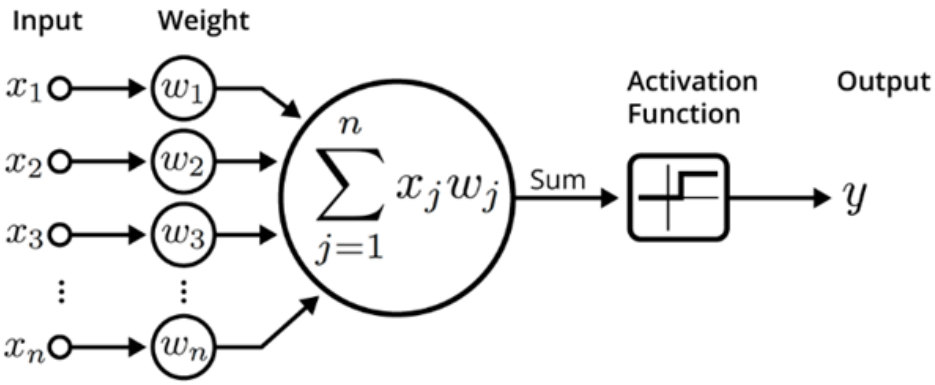

Figure 1.2 Illustration from [9] showing the mathematical perceptron model.

weighted inputs, performs a weighted sum of the inputs, sees if the weighted sum crosses the threshold of the activation function, and then produces an output depending on if it crosses the threshold. Convolutional neural networks contain these artificial neurons along with kernels or filters that represent different receptors that respond to different features, activation functions that require a signal that exceeds a certain threshold to pass

on to the next neuron, and loss functions and optimizers to allow for the model to learn [10].

Deep learning can be broken up into a few types of learning: supervised learning, unsupervised learning, semi-supervised learning, reinforcement learning, self-supervised learning, and transfer learning with unsupervised and supervised learning being the more well-known types.

1.2.1 Unsupervised Learning

Unsupervised deep learning is a learning method where a model is designed and only includes inputs [11]. This method involves unlabeled data being used and the model learns how to best classify the output bases on patterns that it learns during training. There are no defined output labels at the beginning of training and validating the model as the model needs to undergo the process to determine what the outputs are. This can be done through clustering data based on similarity, detecting anomalies in the data, or learning patterns in the data [11].

1.2.2 Supervised Learning

Supervised deep learning is a learning method where a model is designed based on a dataset containing labeled inputs. In this method, the input data is input into the model and an output is provided. During the training process, this output would be cross referenced with the ground truth label and a loss will be calculated. A lower loss value corresponds to a more accurate model. The model will then go through a process of updating different learnable parameters (referred to as weights and biases) within the model to better minimize the calculated loss vale to increase the accuracy of predictions.

A common type of supervised learning methods used for image classification are convolutional neural networks.

1.2.3 Convolutional Neural Networks

Convolutional neural networks are a deep learning model that tend to be a supervised deep learning model. They are widely used for image classification because of their ability to process large amounts of data while also producing accurate predictions [12]. As previously mentioned, hyperspectral image can generate lots of useful data to help us differentiate different types of bacteria from one another, however it will also generate a large amount of redundant or useless data. Convolutional neural networks are a feedforward type of deep learning neural network that uses a combination of convolutional layers and pooling layers to learn the differences between the different input images to generate a probabilistic output for each input image.

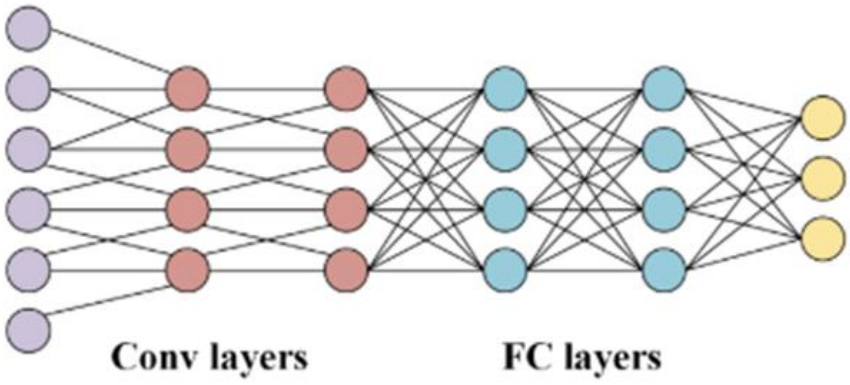

Figure 1.3 Diagram from Z. Li et al. [13] of convolutional layers and fully connected layers.

As seen in Figure 1.3, the convolutional neural network model is a multilayered model consisting of convolutional layers that accepts the input and performs feature extraction along with fully connected layers where the model begins to make predictions

which is then shown at the output layer where the classification is made. There are a few important parts of a convolutional neural network: the input layer, the hidden layers, and the output layers. The input layer is where the model accepts the input, which would be hyperspectral images in our case. The hidden layers are where the convolutional layers, pooling layers, and fully connected layers are located. This is where the bulk of the feature extraction occurs. The convolutional layer is where a kernel of a predetermined size is placed over the input as some weight matrix. In the case of image classification, the inputs are pixels, which means that each location in the weight matrix is placed over each point in the input image. An element-wise multiplication followed by summation is performed which is where the feature map begins to be formed. Information can be lost on the border with this convolution, so padding is used to make the input slightly larger with zero value. Some stride of a preset size is then used to move the convolution kernel

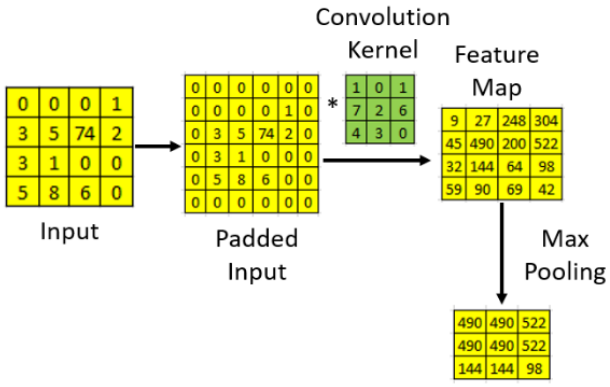

Figure 1.4 Diagram demonstrating a basic convolution that would happen in a convolutional neural network. An input matrix followed by padding is convolved with a kernel of size 3 and a stride of 1. Two numerical examples showing the generation of the feature map are shown underneath. After the feature map is generated, max pooling is applied with a size of 2 and a stride of 1 until the final feature map is output.

once the convolution is performed. This is done over the entire input until the feature map is generated. This will then be followed by a pooling function to downsample the data while making sure to keep track of important features. Figure 1.4 demonstrates a basic example of a padded input matrix undergoing a convolutional to generate a feature map followed by max pooling.

This process of convolutions and pooling is continued based on the depth of the model until the information reaches the fully connected layer which is where predictions are made. The output is then shown at the output layer which is where the classification is made. Figure 1.5 demonstrates a typical convolutional neural network workflow starting from the input and ending at the output classification based on the input.

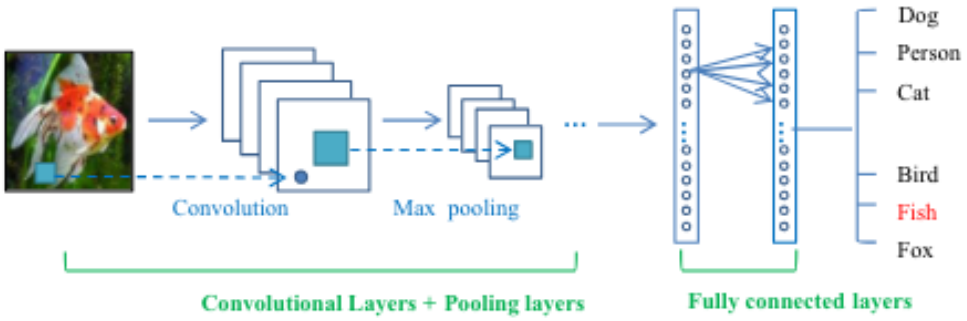

Figure 1.5 Illustration from Guo et al. [14] demonstrating a typical convolutional neural network workflow.

1.3 Deep Learning Optimization

The goal of a typical deep learning model is to minimize some loss function, and this is done as the model is learning. Convolutional neural networks consist of both learnable parameters and non-learnable parameters.

1.3.1 Learnable Parameters

The learnable parameters in the model are the randomly initialized weights and biases that are updated during backpropagation. Backpropagation is performed during training of the model and occurs after the forward pass. The forward pass is where the input image is input into the model, the information is passed through the convolutional layers, fully connected layers, and a class prediction is output along with a loss value. Backpropagation is where the gradient of the loss function is calculated with respect to each weight [15]. The gradients that are calculated are computed with the use of the chain rule and is performed layer by layer. The gradient will tell the user how much the error will change when a certain neuron is changed. The gradients are scaled by the learning rate of the optimizer function with higher learning rates meaning that the model takes larger steps to converge at ideal error values but could potentially overshoot. A smaller learning rate can remove the risk of overshooting but may get stuck at local minima. The originally randomly initialized weights and biases will then be updated based on the loss value with respect to the calculated gradient until the information has successfully made it all the way through the network. The model will then enter a validation mode and use the validation data to see how the model performed with the updates to the network. After this, the model will reenter the training mode and this process will repeat for the remaining epochs of training and validation.

1.3.2 Non-learnable Parameters

The non-learnable parameters are referred to as hyperparameters and the performance of Convolutional neural networks and other deep learning methods depend

on the configurations of these hyperparameters [16]. However, many published studies perform a grid search method to determine optimal hyperparameters, or they arbitrarily try different hyperparameters to determine the best options [16-18]. Depending on the number of hyperparameters you want to adjust, size of the dataset, depth of the model, width of the model, or the variability of the dataset, arbitrarily choosing hyperparameters or performing a grid search may be optimal. However, this may not always be the case based on how many hyperparameters you are looking to tune between runs and how fast the model trains and validates. Hyperparameters may also be optimized for a specific project such as detecting lung cancer but may need to be changed entirely for another project such as detecting bronchus cancer.

CHAPTER TWO

PROJECT APPROACH

2.1 Project Overview

Preventing the growth of biofouling does not only apply to unmanned underwater vehicles, but this can be extended to any object that encounters sea water such as ships, docks, oil rigs, etc. While it may not be necessary for something like docks to experience minimal drag because they are not moving, it can help ensure that the structural integrity stays intact over time. The goal of this project was to use deep learning combined with hyperspectral imaging to help create a smooth, stable biofilm on the outside of these unmanned underwater vehicles to prevent biofouling such as barnacles from growing on the outside and producing drag. This includes determining if a single bacterium or multiple bacteria produces the optimal results.

2.1 Project Justification

Hyperspectral imaging is a unique imaging method that combines typical imaging considerations (image depth and image resolution) while adding a new consideration (spectral range). This new consideration provides new and useful information; however, it also provides useless and redundant information. Convolutional neural networks are used widely for image classification because of their ability or process large amounts of data while also providing accurate predictions [12]. Using a convolutional neural network with hyperspectral imaging will allow for the use of hyperspectral imaging while minimizing the downsides that come with it. Combining these two methods should allow for the analysis of the different spectral responses of bacteria to be analyzed and help us

best determine the single bacteria or multiple bacteria that results in a smooth, stable biofilm to prevent biofouling from occurring on the outside of unmanned underwater vehicles.

2.1 Project Aims

The first aim of this project is to design a deep learning model able to process and classify hyperspectral images of different types of bacteria. This deep learning model should be able to differentiate between the different types of input bacteria with an accuracy rate of at least 80%. The model should be able to accept inputs of greyscale hyperspectral images of size 2048x4096 pixels at least based on the input file format. To achieve an accuracy of at least 80%, we will need a sufficient dataset size to train and validate the model on. We will be using a lab-built hyperspectral imaging system to collect the data samples. Deep learning can be used in combination with HSI to process the large amounts of data and learn which features are important for determining the type of bacteria present in the image if there is any or if there is just empty space present.

The second aim of this project is to optimize the model and analyze the data to yield spatial information and false color images to provide morphological and topological information. Deep learning can be used to process the large amount of data present in hyperspectral images. While the first aim deals with designing a model able to process and classify hyperspectral images, we are looking to obtain spatial information so that we know where the individual bacterium or multiple bacteria are present on the culture plate. With this information, we can then generate a false color image of the input to show

where the individual bacterium or multiple bacteria are present on the culture plate to provide morphological and topological information.

CHAPTER THREE

LITERATURE REVIEW

3.1 Hyperspectral Imaging Review

Hyperspectral imaging is a unique imaging method because of the spectral data acquired from it. Depending on the hyperspectral imaging setup, a sample can be imaged from ultraviolet light to longwave infrared light, or from below 400 nm to fourteen μm [19]. However, hyperspectral imaging has several issues that must be addressed while using it: 1) there is a large number of spectral channels (often referred to as the curse of dimensionality), 2): the large spatial variability of the spectral signatures, and 3): the limited number of training samples [20, 21].

Utilizing hyperspectral imaging to help determine the presence of a single bacterium or multiple bacteria is important to creating the smooth, stable biofilm for the unmanned underwater vehicle. The reason that it is important is that hyperspectral imaging works on the principle that everything reflects light differently based on the chemical makeup of the target. This means that different bacterium will reflect light differently based on the type of bacterium. However, while standard imaging techniques can be used to tell individual bacterium apart or even a combination of bacterium apart, this depends on the type being looked at. While two different bacteria may look red (620-750 nm), one may have peak reflection at 647 nm and the other may have peak reflection at 649 nm. Hyperspectral imaging can capture the peak reflectance of both along with

subtle differences as seen in Figure 3.1 and deep learning can then be used to process and classify these images.

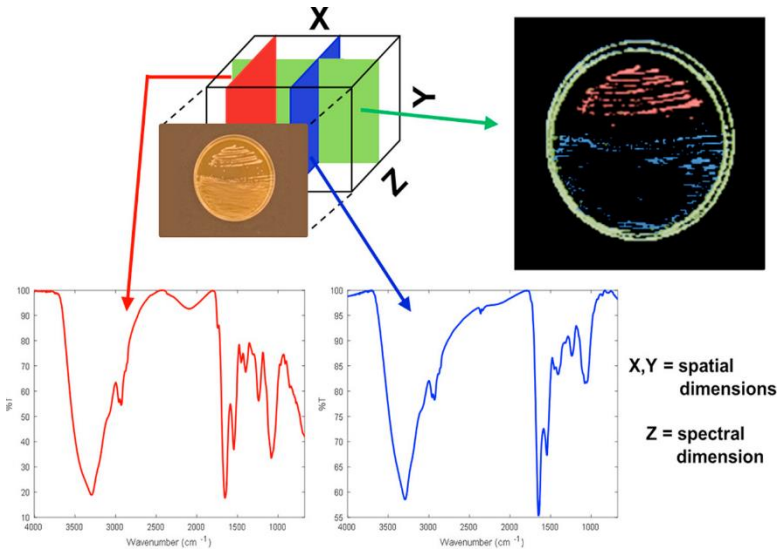

Figure 3.1 Illustration from G. Foca et al. [22] of how hyperspectral imaging can be used to capture subtle differences in peak reflectance in the spectral dimension. While the two different bacteria have similar reflectance peaks, hyperspectral imaging shows that there are differences between the two example bacteria.

While hyperspectral imaging is a great tool for capturing the subtle differences between bacteria that reflect light at similar wavelengths, it can be hard for humans to interpret this data without aid from other tools. For example, hyperspectral images captured with a linear CCD camera can be presented as a greyscale 1-dimensional image (1xλ) while it is still a 3-dimensional dataset (1x1xλ) where λ is the range of wavelengths captured by the camera. A human will be unable to view this image and gain any information as it is only a greyscale image one unit (i.e., one pixel) in height.

3.1.1 Hyperspectral Imaging Modes

Hyperspectral imaging setups can be designed with a few different approaches in mind: snapshot, staring, pushbroom, and whiskbroom.

The snapshot imaging mode, also referred to as a single-shot method, is a setup that is used to acquire spatial and spectral information all at once. This acquires the entire hypercube at once but comes at the cost of low spatial and spectral information. This is because the method images the dispersed image zones and remapping onto a CCD at the same time but is limited by the number of voxels as the number of voxels cannot exceed the number of pixels [23].

The staring imaging mode, also referred to as a band sequential method, is a setup that acquires a single wavelength image with full spatial information at once [23]. This means that the entire area of the object is scanned with a single wavelength at a time and is followed by the wavelengths changing for each subsequent image. This allows for the user to be more specific with the spectral data being acquired as they determine each wavelength image and has a short data acquisition time, but this twenty-three is generally more high cost compared to the other methods and has a low throughput [24].

The pushbroom imaging mode, also referred to as a line-scan method, is a setup that involves simultaneously collecting spectral information along with a slit of spatial information which would be a (X, λ) or (Y, λ) image [23]. This can acquire more light than the whiskbroom method as it can stay in an area for longer, however the frame acquisition rate of the detector will need to be synchronized with the camera or object moving as one line is acquired at a time [23]. If this is not considered, it is possible that

the user will not obtain a smooth image. This method is faster than the whiskbroom method, but you do not obtain as much data per point.

The whiskbroom imaging mode, also referred to as a point-scan method, is a setup that involves a single point being scanned in the two spatial dimensions (X and Y) with the reflected light being dispersed by a prism and then using a linear array detector to record the information [23]. This is the most time-consuming mode as it requires a point-by-point scan of the data, but results in the most amount of data being generated for the user. A comparison chart of snapshot, staring, pushbroom, and whiskbroom can be seen in Figure 3.2. Each of these hyperspectral imaging modes can acquire unique data

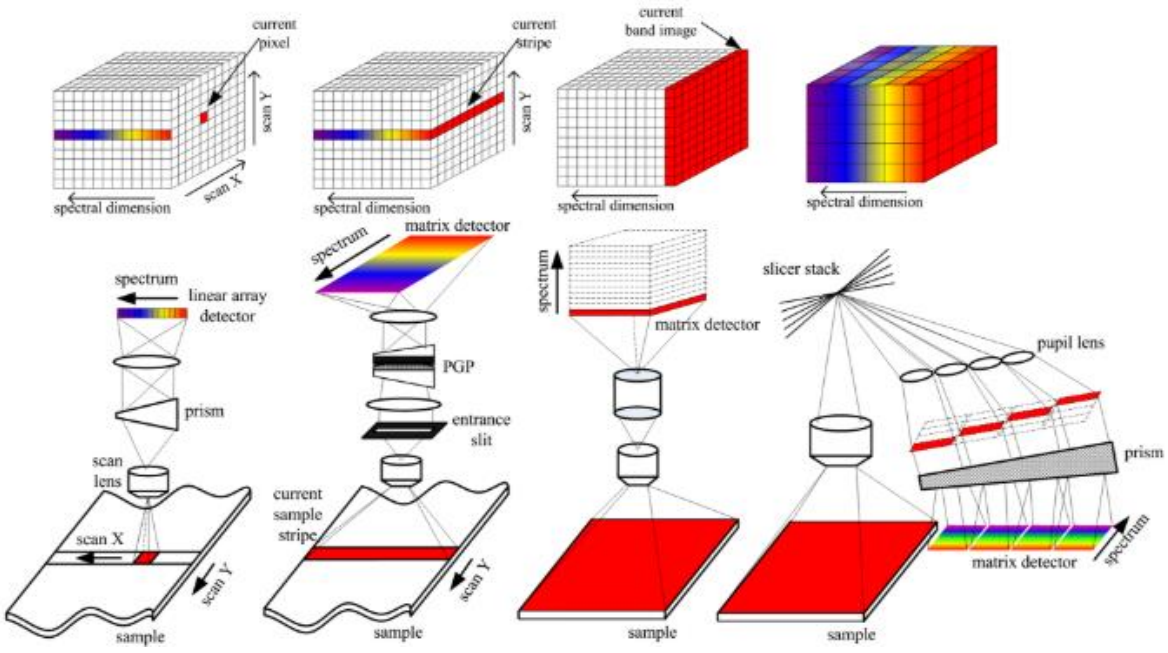

Figure 3.2 Illustration from Q. Li et al. [23] comparing the differences in imaging between the whiskbroom, pushbroom, staring, and snapshot hyperspectral imaging methods.

based on the method used which can be processed using a deep learning algorithm.

3.2 Deep Learning Review

Deep learning is a subset of machine learning that has made many advancements in the last decade that have helped solve problems that have been challenging the artificial intelligence community for years. Deep learning has turned out to perform very well at extracting and discovering different patterns and structures in high-dimensional data. This has resulted in deep learning being applied to not only just image classification, image recognition, and speech recognition, but many other types of applications such as analyzing particle accelerator data, determining the activity of drug molecules, and even reconstructing brain circuits [8, 24-34].

Year	CNN	Developed by	Place	Top-5 error rate	No. of parameters
1998	LeNet(8)	Yann LeCun et al			60 thousand
2012	AlexNet(7)	Alex Krizhevsky, Geoffrey Hinton, Ilya Sutskever	1st	15.3%	60 million
2013	ZFNet()	Matthew Zeiler and Rob Fergus	1st	14.8%	
2014	GoogLeNet(19)	Google	1st	6.67%	4 million
2014	VGG Net(16)	Simonyan, Zisserman	2nd	7.3%	138 million
2015	ResNet(152)	Kaiming He	1st	3.6%	

Table 3.1 Summary table from [35] comparing different convolutional neural networks on the ImageNet challenge.

While there are many different applications for deep learning models, an important thing to note is that no model is the same. This means that the architecture for each model being different depending on the project such as having a model with seven

layers versus three layers, having a different amount of feature maps generated for each layer, different padding values for the images, different kernel sizes for each layer, etc. Table 3.1 shows the difference between popular image classification models tested on the popular ImageNet Large Scale Visual Recognition Challenge. ImageNet is a popular image database used primarily for object recognition software and has become very popular within the deep learning community. There are more than fourteen million images that have had at least one million of them hand annotated. This database also contains more than 20,000 categories with typical categories containing a few hundred images. The challenge uses a smaller list with only 1,000 total classes being prevalent. In 2015, AlexNet was outperformed by Microsoft's deep network with one hundred layers [36].

It is important to note that just because a model has more parameters than other does not necessarily mean that the model will perform better overall. VGG Net(16) placed second in the competition with 138 million parameters in 2014 compared to GoogLeNet(19) which placed first with only four million parameters. While more parameters, more layers, etc. can result in more features being extracted, this can lead to the model learning patterns that are not there which results in overfitting.

When designing a from scratch deep learning model, there are many things that need to be taken into consideration. Some things that need to be thought of, for example, include what is the input, what does the input image look like (image height, image width, number of color channels), what is the output layer expected to produce, what

information should be extracted and stored for either the model to use or for personal use, and hyperparameters.

3.2.1 HyperOPT

As previously mentioned, hyperparameters are non-learnable parameters that are defined before the model is defined, trained, and validated. These parameters will not be updated during training and validation and can only be changed between these runs of training and validation. When dealing with more than a couple hyperparameters, standard grid searches or arbitrary choices may not work well. It is possible that even random searches may be competitive with experts [37]. HyperOPT is a Python library that conducts hyperparameter optimization through providing an optimization interface [38]. This interface distinguishes between an evaluation function and a configuration space. This evaluation function will assign loss values to points within the defined configuration space [38].

Three algorithms are currently provided with HyperOPT – simulated annealing, random search, and Tree-of-Parzen-Estimators (TPE). Random search is the simplest method and is where a random set of values for the hyperparameters are picked from the specified hyperparameter space. While this method is simple, it does not use previous evaluation results to make informative choices on future searches. Simulated annealing starts at a random point in the defined space. This will then make small and random changes to the hyperparameters. This method is ideal for escaping from a local minimum by being able to accept worse changes, but this is also its biggest problem as it can make some worse overall changes. Finally, TPE is an approach that allows for the algorithm to

distinguish between good and bad choices. This method builds a density function for hyperparameters that result in optimal outcomes and one density function for the rest. When the next set of hyperparameters is suggested for evaluation, this method will choose the value or values that maximize the ratio between the densities of the two functions. This method is more efficient than methods such as random search but depends on how well the model is designed and can require a large amount of trials to outperform the other methods [39].

The user can define a configuration space of hyperparameters they are looking to change with the model training. There are a few examples of the expressions recognized by the optimization algorithm which are hp.uniform, hp.choice, and hp.loguniform. The first argument accepted is the label, or hyperparameter, the user wants optimized and must be input as a string. The second argument depends on the expression being used. The hp.uniform function will draw uniformly between two values: a low value and a high value. The hp.loguniform function will draw between a low value and a high value, but the interval that it is constrained to is [log(low value), log(high value)] rather than [low value, high value] that is seen in the hp.uniform function. The final function shown is hp.choice which will use a value shown in the list.

F1 score is the harmonic mean of precision and recall where precision is the ratio of true positives to the sum of true positives and false positives and recall is the ratio of true positives to the sum of true positives and false negatives. When performing a binary classification, true positives, false positives, true negatives, and false negatives are

important when it comes to evaluating the experiment as we can see why the model is performing poorly or performing well.

3.2.2 F_1 Score for Optimization

It is important to note that when evaluating a deep learning model, evaluation in the training phase is not the same as evaluating the final model [40]. When a model is being trained, a dataset is usually divided into a training subset and a validation subset with a common split being an 80/20 training/validation split where 80% of the dataset is used to train and the remaining 20% is used to validate. Once training has been completed, the next step is to evaluate the model on a new dataset to see how the model performs on entirely unseen data.

The goal of a deep learning image classification algorithm is to minimize some loss function or objective. While a model can output training loss and validation loss, there are other metrics that can provide information on how the model is performing. Examples of evaluation metrics include accuracy, Area Under the Receiver Operating Characteristics (ROC) curve (AUC), and Precision/Recall (PRC) plots [40]. When choosing an evaluation metric to include with training and validation loss, it is important to note what type of model is being evaluated. A common model for image classification are binary classifiers. Binary classifiers are models that divide a dataset into two groups: positive and negative [40]. When the classifier makes the prediction, it can produce one of four outcomes: true positive, false positive, true negative, or false negative. A table can be developed showing the four outputs are referred to as a confusion matrix.

True positives, true negatives, false positives, and false negatives are the basis for the basic evaluation measures of binary classifiers [40]. Common measures for binary classifier performance include error rate and accuracy rate with sensitivity and specificity being popular [41, 42]. Another type of evaluation metric is the F_β where β is 0.1, 1, or 2 [40]. The F1 score is the harmonic mean between precision and recall, or

$$F1 = \frac{2*Precision*Recall}{Precision+Recall} \quad [40]$$

where precision is defined as

$$Precision = \frac{True\ Positives}{True\ Positives+False\ Positives} \quad [40]$$

and recall is defined as

$$Recall = \frac{True\ Positives}{True\ Positives+False\ Negatives} \quad [40]$$

This value will be between 0 or 1 with one meaning perfect precision and recall and zero meaning poor precision and recall.

Each metric has their own advantages and disadvantages, and their usage needs to be determined on a case-by-case basis as each metric behaves differently on balanced datasets compared to imbalanced datasets. F_1 scores are a synthetic indicator used to measure how well a model is performing [43]. It combines both precision and recall providing a value between 0 and 1 to show how well a model is performing and tends to be used more often on imbalanced datasets. Using training loss, validation loss, confusion matrices, and F_1 scores, an image classification convolutional neural network can be evaluated properly when it comes to classifying images on an imbalanced dataset.

3.3 Visualization

Deep learning models are generally considered a black box because of the amount of matrix math used in addition to the high-level features that a model uses to learn. It is possible to visualize what the deep learning model is using to learn as well as visualize the output. Gradient-weighted Class Activation Mappings (Grad-CAMs) can be used to visualize what the model believes to be important during training and validation and false color images can be generated based on the input and can be beneficial if the user is working with greyscale input data [44, 45].

3.3.1 CAMs

Convolutional neural networks are popular for image classification because of their feature extraction capabilities and ability to produce accurate predictions. However, an issue with deep learning models, especially from scratch models, is that they can fail with no notice [46]. It is imperative that a model is designed so that perform well and can

Figure 3.3 Illustration from Zhou et al. [44] showing an input image of brushing teeth and the corresponding CAM indicating the area of interest for the convolutional neural network.

also be explainable so that the user can know why it worked or why it failed. Class Activation Mapping (CAM) is a process that allows for the user to see why certain images were classified as one output class compared to a different class [46]. Figure 3.3 demonstrates an example of a CAM indicating the areas of interest for a model able to indicate brushing teeth.

When generating CAMs for data, it is important to note that CAMs can look different for different dimensions of input data. While the image in Figure 3.3 shows a CAM for a 2-dimensional input image as a heat map centered around the toothbrush, a CAM will look different for a 1-dimensional input image or a 3-dimensional input image. For a 1-dimensional input image, it will correspond to pixel intesnity assuming that is the information of interest with pixels of higher importance corresponding to a higher activation intensity. CAMs are a great tool for helping visualize and explain what a deep learning model is viewing as important, but CAMs are just one method to help visualize when it come to deep learning.

3.3.2 False Color Images

The final thing to consider when designing a deep learning image classification model is what should the output produce besides a prediction of the output based on the input. When dealing with greyscale input images, it can be hard to differentiate between different objects or structures. For example, looking at an MRI image and trying to determine what the image is showing can be hard and generally requires someone specialized in interpreting MRI images to be able to accurately identify different structures and produce a diagnosis.

False color images are images that use a specific color mapping to map different objects or structures to certain colors and then reconstruct the original input with this new color scheme. If looking at an MRI image of the brain, the user could map the frontal lobe to red, the hypothalamus to blue, and a tumor on the hypothalamus to yellow. Once the MRI image goes through the deep learning model, the model would make predictions on what the different structures are (such as the frontal lobe, hypothalamus, etc.) and where a tumor is located (such as on the hypothalamus). The information used for the output classifications could then be used to reconstruct the input MRI image with the new color mapping to help better differentiate between the areas of the brain and where the tumor is located. This can be beneficial as it would allow for the user to obtain margins of the tumor separated by color and help with forming a plan of action.

This can be extended to other areas such as reconstructing hyperspectral images of different types of bacteria. While a hyperspectral imaging system may image from 500 nm through 850 nm with 1 nm bandwidth, the output image may not necessarily be a file with 350 bands. It could be a greyscale image, or one band, with the spectral data corresponding to pixel index on the x-axis and the wavelength increasing across the image from left to right. This may not be useful to the user when attempting to differentiate between multiple bacteria in the same image, so mapping each bacterium to a color can be important as it helps the user differentiate between different bacteria in the same file or same image. This can then help with providing morphological and topological information in an easy to visualize and understand format.

CHAPTER FOUR

DESIGN OF CONVOLUTIONAL NEURAL NETWORK

4.1 Design Needs

4.1.1 Hyperspectral Imaging Design Needs

Hyperspectral imaging setups are designed to acquire a 3-dimensional dataset with two spatial dimensions (X and Y) and one spatial dimension (λ). While the primary focus is designing a from scratch deep learning model capable of analyzing hyperspectral images of bacteria, the design of the hyperspectral imaging setup influences how the model is designed. The hyperspectral imaging setup needs to be able to image a sample from 550 nm to 850 nm. This imaging range is necessary for the setup for three reasons: 1) the filter we have does not go past 850 nm meaning there is an upper limit of 850 nm, 2) as we approach ultraviolet light (<400 nm), samples can become damaged, and 3) research on biofilms indicate peak spectral responses are usually around 660-680 nm, so having the range go slightly below 660 nm and being unable to exceed 850 nm should provide enough spectral information [47-50]. The design should also have a way to indicate where the hyperspectral image is obtained from. The input images will need to be obtained from a 0.25 mm by 0.25 mm field of view.

4.1.2 Deep Learning Design Needs

Deep learning models tend to be referred to as a black box due to the amount of heavy computation that is performed to help the model train and learn. This becomes even more complicated by the fact that each deep learning model created depends on the task the user is looking to perform. A convolutional neural network is a deep learning

model that consists of alternating convolutional and pooling layers able to perform feature extraction on high level features while still producing accurate classifications. In order to design a convolutional neural network that can properly process hyperspectral images and be properly optimized, there are several design requirements that must be taken into consideration.

The model must be able to accept input hyperspectral images obtained from the hyperspectral imaging system. This means that the model must be able to accept 1-dimensional greyscale files with a pixel size of 2048x4096 or 1024x4096 as that is how the imaging system formats them. The model must be able to process files with the .bmp or .TIF file type. The model must be able to deconstruct the input files into the respective x-scans acquired by the imaging system. The model must be able to normalize the images to prevent cases where a vanishing or exploding gradient could occur during backpropagation. This will allow for pixel intensities to be set between values of 0 and 1 across the entire dataset. One-hot encoded vectors must be used for the multi-label classification being performed by the model which means a label dictionary must be created to work with the input images. The model must be able to split the training data into a randomly shuffled 80/20 train/validation split.

Once the model has been defined and has been able to preprocess the data, there are also several needs for the model to perform training and validation as well as store specific information regarding the data. The model must be able to store and print training loss, validation loss, F_1 scores for each class, and total training time. The model must be able to store what file the input image came from, what row in the file the input

image came from, and what the prediction for the input image was. The model must be able to store the ground truth label for the input image during the training and validation stage. With this information, the model should be able to also keep track of the true positives, true negatives, false positives, and false negatives for each class during training and validation to produce confusion matrices for each class. The model should also be able to log the training loss, validation loss, and F_1 scores for each class.

Finally, the code has a few more needs to provide practical information and so that the model can be explained properly. The code should be able to convert the input images into pixel intensity graphs so that the user is able to determine which wavelengths used for imaging are the most important for each class. The pixel intensity graphs will need to be summed together for each class for ease of reading. Wavelength will then be determined so that we can see what increase in wavelength occurs for each pixel. This information will be applied to the pixel intensity graphs. Grad-CAMs must also be able to be produced for each convolutional layer so that the user can see which wavelengths the model uses to determine the difference between the different bacteria classes. Lastly, a color mapping must be defined to map different predictions to different colors. As the images are 1-dimensional greyscale inputs, this means that each bacteria will look visually the same in each x-scan to the user. Mapping different bacteria to different colors and then using that information to reconstruct the images as a false color image will allow for the user to easily distinguish between different types of bacteria in each image.

4.2 Design Phases

When designing something new, there are many things to consider. After the

Amount of Data				
Bacteria Name	Number of 2048x4096 Files	Number of 1024x4096 Files	Total Number of Files	Number of Images
ATL 1	75	18	93	172,032
ATL 1 and ATL 3	21	21	42	64,512
ATL 1 and ATL 28	21	0	21	43,008
ATL 1 and B 27	21	0	21	43,008
ATL 3	78	21	99	181,248
ATL 3 and ATL 28	21	24	45	67,584
ATL 3 and B 27	21	0	21	43,008
ATL 28	78	18	96	178,176
ATL 28 and B 27	21	21	42	64,512
B 27	86	21	107	197,632
Total	443	144	587	1,054,720

Table 4.1 Table showing the data used for training and validation. This table indicates the class names (both individual and combinations), the amount of 2048x4096 and 1024x4096 data for each class, and the total amount of images for each class.

project has been determined and the objective has been defined, this will usually be followed by a literature review and background research. This is done to determine what is currently available and where the gaps in the research are. For deep learning, the gaps can be easier to determine as deep learning models are all very specific to their respective projects. This will then be followed by determining how to design the new model, how the model works, and what the model should be returning to the user. Feedback will be provided, and updates can be made based on the feedback. This will then return the project to the research and literature review phase as the feedback may be information that was not previously considered.

4.2.1 Literature Review and Background Research

Extensive literature review was performed to determine the objective of the project and the gaps in literature regarding the use of convolutional neural networks

combined with hyperspectral imaging. From the review, it was determined that a model must be designed to: 1) accept the specific inputs based on the file type, 2) have enough layers to properly learn the features, but not too many as to avoid overfitting, and 3) proper metrics for dealing with binary classification.

The first necessary portion of the design is to create a Python class that has access to the path of the directory that contained the dataset. It must also be able to break down each file of size 1024x4096 or 2048x4096 pixels into their respective 1x4096 images, store the location information for each image so that their original location could be determined (including folder name, file name, and the row index in the file), list the bacteria classes to be determined, and process the files stored as .bmp or .TIF. This class is given the name BacteriaData.

The second necessary portion was to define a convolutional neural network capable of accepting the greyscale 1-dimensional inputs obtained from the BacteriaData class. The convolutional neural network requires the number of input bands from the

Amount of Data				
Bacteria Name	Number of 2048x4096 Files	Number of 1024x4096 Files	Total Number of Files	Number of Images
ATL 1	75	18	93	172,032
ATL 1 and ATL 3	21	21	42	64,512
ATL 1 and ATL 28	21	0	21	43,008
ATL 1 and B 27	21	0	21	43,008
ATL 3	78	21	99	181,248
ATL 3 and ATL 28	21	24	45	67,584
ATL 3 and B 27	21	0	21	43,008
ATL 28	78	18	96	178,176
ATL 28 and B 27	21	21	42	64,512
B 27	86	21	107	197,632
Total	443	144	587	1,054,720

Table 4.1 Table showing the data used for training and validation. This table indicates the class names (both individual and combinations), the amount of 2048x4096 and 1024x4096 data for each class, and the total amount of images for each class.

input image to be specified. Our goal was to have this able to accept our input data, seen in Table 4.1, process the information through feature extraction, and capable of predicting one of the ten classes. It was decided to use three convolutional layers and one fully connected layer.

The final necessary portion was determining proper metrics for binary classification. As previously mentioned, F1 scores were chosen for this deep learning model due to the binary classification nature of this project. The ratio of true positives, false positives, true negatives, and false negatives were important as they allow for us to determine how the model is classifying, but they also allow for easy cross checking when generating false color images. This is because if a specific bacterium is misidentified as another type of bacteria for the false color images, we can still tell what the bacteria most likely is as we can look at the surrounding bacteria as well as what was imaged and make inferences based on the information.

4.2.2 Model Design

With the understanding of the basics of setting up a convolutional neural network and how to evaluate the model, we were able to design a convolutional neural network

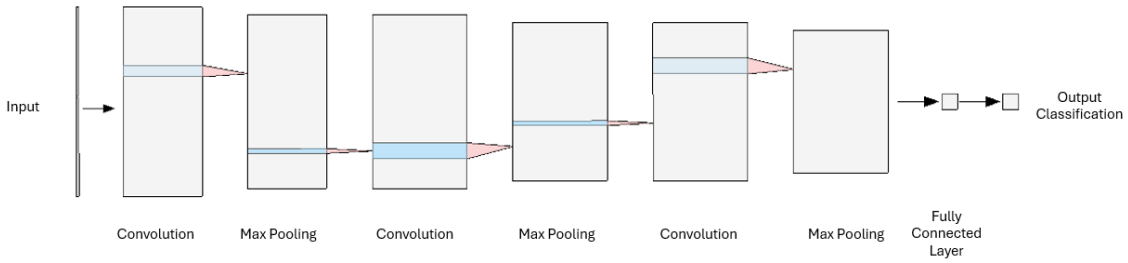

Figure 4.1 Basic diagram illustrating the design of the convolutional neural network using software from [51].

capable of accepting the individual hyperspectral input images and processing them through the network. The basic design flow can be seen in Figure 4.1.

This model design would use three convolutional layers followed by max pooling to reduce the dimensionality of the images. The output of the final pooling layer would be flattened into a 1-dimensional vector. This would then be followed by the output classification layer where the predictions are made.

4.3.1 Model Design Specifics

Using the basic design flow as seen in Figure 4.1, we designed a convolutional neural network that can be used to analyze hyperspectral images of bacteria. We needed to define a convolutional neural network capable of accepting the greyscale 1-dimensional inputs obtained from the BacteriaData class previously mentioned. The model required the number of bands to be specified. As the images were greyscale, the number of input bands was set to 1. The input width and height had to be specified which was 4096 and 2048, respectively. Most of the data files were 2048x4096, but the 2048 image height was able to work with data that was 1024 pixels in height. The number of labels/classes had to be specified. As seen in Table 4.1, some data consists of both 1024x4096 and 2048x4096 data, so it was important the model could work with both file sizes. Because the data would be downsized through pooling layers, the new height and width after pooling had to be defined for use by the model to manage the decreased dimensions.

A convolutional neural network class was defined with the name myBacteria that inherits from the nn.Module which is the base class for all PyTorch neural networks. This

class then had the layers initialized which consisted of two convolutional layers followed by a fully connected layer. The convolutional layers consisted of the nn.Conv1d( ) convolutional function, nn.LeakyReLU( ) activation function, nn.MaxPool1d( ) pooling function, and the nn.Dropout( ) dropout function.

The nn.Conv1d( ) function performs a 1-dimensional convolution and is necessary due to the 1-dimensional input data. This function accepts the number of bands for the input image and then allows for the defining of the output channels, kernel size, and padding [52]. For the first convolutional layer, the number of output channels (also referred to a feature maps or activation maps) was set to 32. The kernel size was set to 5 with a stride of one which means each randomly initialized kernel matrix would have a size of 1x5 because of the 1-dimensional nature of the data. Padding was added around the input images with a size of two. The output from this function is 256x4096x32 with 256 representing the batch size, 4096 representing the length of the output, and thirty-two representing the number of output channels.

The nn.LeakyReLU( ) assigns the Leaky ReLU activation function to this layer. The Leaky ReLU activation function is an activation function based on the ReLU activation function. The ReLU function is one of the most used activation functions in neural networks and is popular because of its fast convergence rate, linear operation, and calculation speed [53]. However, ReLU comes with a downsize called the dead neuron because for the ReLU function, any negative value as an input is always set to zero. This results in the first derivative being zero which does not allow for parameters to update properly. The Leaky ReLU was designed to work like the ReLU activation function but

was designed to help overcome the downside of the ReLU function [54, 55]. Any input value above 0 would be output as whatever the input value was, but any value below zero would be set to a value of -0.1 multiplied by the input value as to introduce a small slope to the negative input to help avoid the dead neuron problem [53]. Figure 4.2 shows the graphical difference between the two activation functions.

The nn.MaxPool1d( ) function is used to pool the output feature maps and follows a convolution. This works by first accepting an argument stating the pooling kernel size which is set to a value of 2 [56]. Like the kernel size, this is a 1x2 kernel size as the input information is 1-dimensional. This works by sliding the 1x2 kernel over the information output to the feature maps following the convolutional and choosing the maximum of the two values. This maximum is output to the final feature map of the first layer. This reduces the dimensionality to increase the computation speed, but also reduces the amount of redundant or irrelevant information. Following pooling, the final feature map

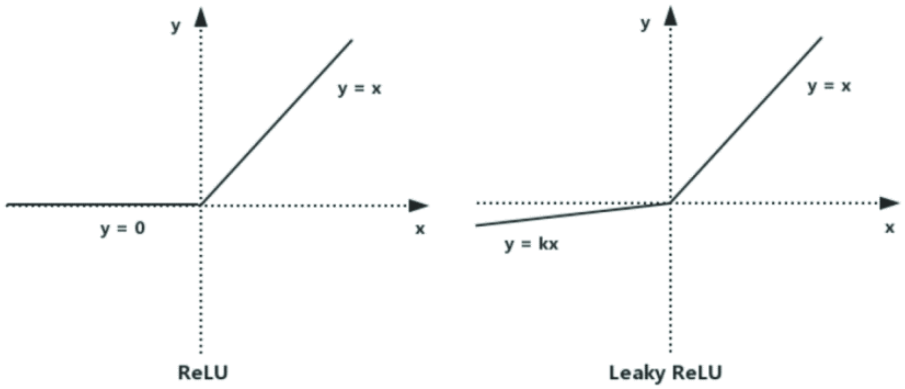

Figure 4.2 Illustration from Nair and Hinton [53] demonstrating the difference graphically between ReLU and LeakyReLU where k is a leak correction set to -0.1.

size for this layer is 256x2048x32.

The nn.Dropout( ) function is used to add a regularization technique to the model to prevent overfitting from occurring and is only present during the training portion of training and validation [57]. This function accepts a numerical probability argument between a value of 0 and 1 and is set to 0.25179056283341855 for this project. This works by randomly zeroing out elements during the training process at each update step to help prevent the model from relying on specific inputs for classification. This allows for the model to become more robust and still provide good outputs with potentially suboptimal inputs. The amount of elements zeroed out during the update steps is based on the numerical probability input and is viewed as a percentage. This means that a value of 0.25179056283341855 will zero out 25.179056283341855% of the elements during each training step. The value chosen depends on the input data and could be significantly larger or smaller than the chosen value depending on the project or even the layer the function is located in.

Following the first convolutional layer, the output from the nn.MaxPool1d( ) function is fed into the input of the second convolutional layer. This has a similar setup to the first layer, but with a few notable changes. The first difference is that the input for this layer is set to accept thirty-two channels as the previous layer output thirty-two feature maps. This layer will use a kernel size of seven with a stride of one. This layer contains the same nn.Conv1d( ), nn.LeakyReLU( ), nn.MaxPool1d( ), and nn.Dropout functions as the previous layer. The convolution for this layer will happen slightly different from the first convolution layer. The kernel will still start at the left side of the input; however, it will perform a convolution over the entire input at the same time. This

is because the input is thirty-two feature maps stacked on top of each other giving the input a shape of 256x2048x32. The kernel of size seven will do an element wise multiplication and summation of all thirty-two channels at the same time for the first seven elements. This layer will output sixty-four feature maps with size 256x1024x64 after pooling.

The third convolutional layer is set up the same as the second convolutional layer. The main difference between this layer and the previous layer is that the input will be of size 256x1024x64 rather than 256x2048x32. This layer will output sixty-four feature maps as well, but with a size of 256x512x64 after pooling.

Following the third convolutional layer is what is known as the fully connected layer. This layer is where the output 256x512x64 tensor is flattened into a 1-dimensional vector which will have a size of 1x32,768 for each item in the batch. This allows for the model to make predictions. Once the data has been flattened, the features need to be mapped to the output classes and is done using a linear transformation. The flattened vector will be multiplied by a randomly initialized weight matrix with a randomly initialized bias to produce values referred to as a logit or raw logit. There are 256 1x32,768 vectors processed at a time with each batch being independently processed. The output of the fully connected layer will be of size 256x10 representing the batch size and number of classes. These raw logits will be put through a sigmoid function from the BCEWithLogitsLoss loss function to set the values between 0 and 1. A dropout value of 0.25179056283341855 is used for this layer as well.

Figure 4.1 indicates how the data flows through the model and how it is downsized during pooling. The input starts at 4096 pixels in width and is subsequently reduced to 2048, 1024, and 512 pixels. The figure also shows how the kernel sizes differ for the pooling layers and the convolutional layers. The data is then flattened into a vector of size 1x32,768 which is then fed to the output layer containing the ten classes. The image height stays the same for each layer.

In addition to the three convolutional layers, the myBacteria class contains a few other functions necessary to help the model process the data. A helper function is used to calculate the number of elements that are present in the output tensor that is produced by the convolutional layers. This is done so that the input is correctly sized for the fully connected layer so that the flattened output from the convolutional layers is the same size as the linear layer's input features.

The class also contains a forward method. This allows for the input to be passed through the convolutional layers so it can be transformed by the convolution functions, activation function, and the pooling function. This method also allows for the data to be flattened and passed through the fully connected layers. This will be followed by the classification. After this forward method is defined, the model is instantiated, moved to the GPU, and monitored by the Weights and Biases tool.

The Weights and Biases tool is a monitoring tool designed to be used with deep learning. It has its own Python library called wandb. This is a logging tool where the user can specify what they are looking to track, and it will track each run as well as provide graphical representation methods. The user can track metrics such as training loss,

validation loss, F_1 score, and confusion matrices. WandB is a popular tool used by many large names such as NVIDIA, OpenAI, and Co:here.

4.3.2 Data Preprocessing and Data Preparation

As previously mentioned, the BacteriaData class is defined to accept our input data, process the information through feature extraction, and capable of predicting one of the ten classes. Figure 4.3 shows an example image of the type of input the model would see. For illustration purposes, the image was stretched vertically for visibility as the input

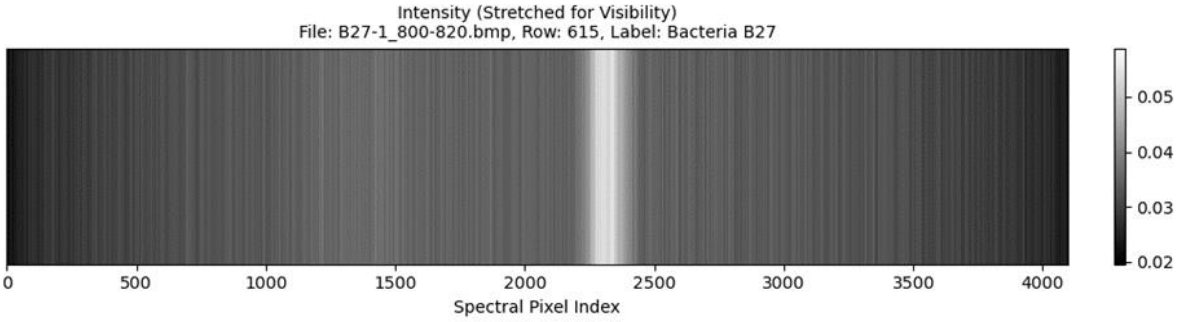

Figure 4.3 Example input image from file B27-1_800-820.bmp indicating the input image was bacteria B 27 from row 615.

image is one pixel in height. The spectral pixel index indicates the pixel location with pixels closer to the color white meaning they are reflecting more light and pixels closer to the color black meaning they are reflecting little to no light. The wavelengths being used to illuminate the image is indicated in the file name with Figure 4.3 being imaged from 800 to 820 nm. Each image, however, is from 550 nm to 850 nm with 550 nm located at pixel index 0 and 850 nm being located at pixel index 4095. This means that the image in Figure 4.3 will have the 800 to 820 nm range primarily illuminated.

Each input image can be seen as a pixel intensity graph. Starting from the left side of the image at pixel index zero, a line can be drawn with the height increasing as the

shade of the image becomes lighter and decreasing as the shade becomes darker. This was done for every image and an example of B 27 can be seen in Figure 4.4. Unlike the

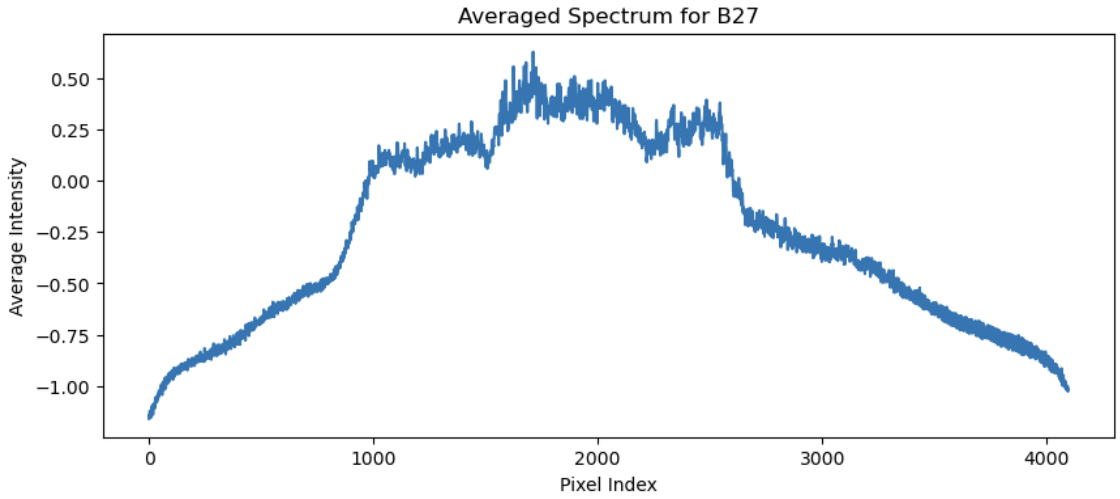

Figure 4.4 Pixel intensity graph of bacterium B 27.

input image example, this is a combination of all the B 27 images over the entire 550 to 850 nm spectrum. This was done for every input image with Figure 4.5 showing the

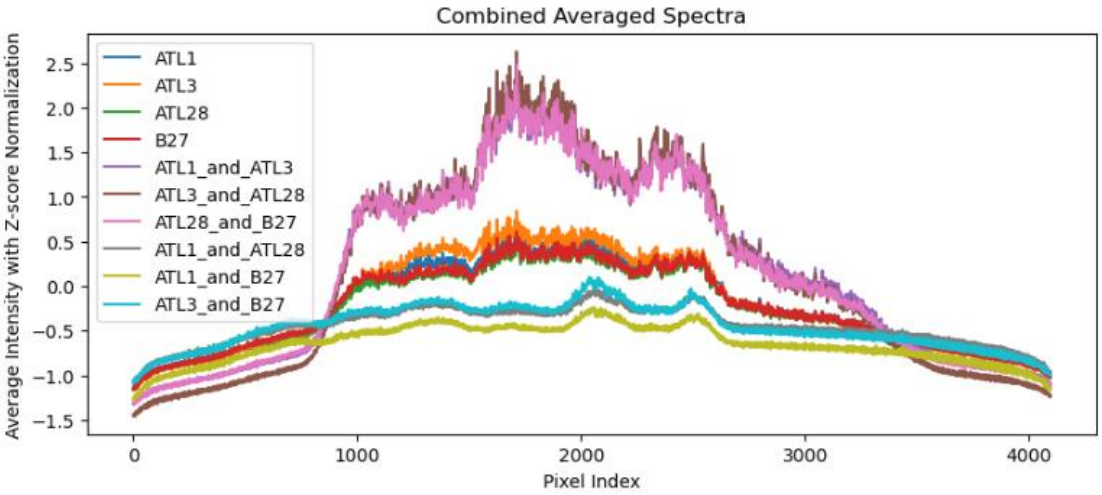

Figure 4.5 Summed pixel intensity graphs for each bacterium/bacteria class.

different bacteria on the same plot. This plot shows that there are subtle differences between the individual bacterium and the combinations of bacteria. For example, the

combinations of ATL 1 and ATL 28 as well as ATL 3 and B 27 look similar to each other with similar shapes, peaks, and minimums. However, the model can spot the subtle differences between the light reflected between the two and make proper classification predictions. These plots are not used for image processing, but primarily for visualization purposes to incorporate explainable AI into the project. It is important to note for Figure 4.5 that the 550 nm is located at pixel index 820 and 850 nm is located at pixel index 3450.

In addition to the BacteriaData class, another class, named MeanStdDataset, is defined to calculate the mean and standard deviation of the pixels across all images in the dataset. This code will iterate over every image within the dataset and will update mean and standard deviation accumulators with both the mean and standard deviation of the images while also keeping count of the total number of images processed. Any value within the standard deviation tensor equal to zero will be set to 1.0 to avoid any issues with division by zero later during future computations. It will then output the mean and standard deviation of the dataset. This is used to normalize the dataset by performing a z-score normalization which causes the normalized dataset to have a mean of 0 and standard deviation of 1. Any value below the mean will be negative and any value above will be positive. This is how the y-axis values are obtained in Figure 4.5.

In addition to calculating the mean and standard deviation of the images in the dataset, a custom collate function is defined. Collate functions are used to combine individual data samples into batches during the data loading process. This is used when the user wants to batch data in a specific way. This function is used to store what folder

the input image came from, what file the input image came from, what row the input image came from, and what the label the input image has. This function is used creating a data loader.

```
label_dict = {
    'ATL1': [1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    'ATL3': [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
    'ATL28': [0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
    'B27': [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
    'ATL1_and_ATL3': [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
    'ATL3_and_ATL28': [0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
    'ATL28_and_B27': [0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
    'ATL1_and_ATL28': [0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
    'ATL1_and_B27': [0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
    'ATL3_and_B27': [0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
}
```

Figure 4.6 Figure showing the label dictionary with the different one-hot encoded vectors and their corresponding classes.

To prepare and preprocess the data, a label dictionary must also be created. This is used for multilabel classification. The label dictionary is a dictionary of one-hot encoded vectors with each key in the dictionary representing a class label. Each label corresponds to a vector with the format [X, X, X, X, X, X, X, X, X, X] where X is either a 0 or 1. Figure 4.6 shows the different classes and their corresponding vectors. This label dictionary works by applying a sigmoid function to the raw logit outputs. This sigmoid function sets the output logit to a value between 0 and 1. A threshold value is applied during the training and validation of the model where any value greater than 0.5 is set to a one whereas any value less than or equal to 0.5 is set to 0. This means that an output vector could look like [0.9, 0.3, 0.1, 0.2, 0.4, 0.5, 0.2, 0.1, 0.1, 0.4] which would translate

to [1, 0, 0, 0, 0, 0, 0, 0, 0, 0] which corresponds to bacteria ATL 1. Each prediction follows this setup and is how the model differentiates between the ten different classes.

Following the label dictionary, transformations are applied to the image. Transformations are common for deep learning networks and can include cropping, rotating, changing contrast, and flipping. This allows for the model to see diverse types of input data to produce so the model can produce correct outputs even if the inputs are suboptimal. Transformations applied to the hyperspectral data include transforming the input data to a PyTorch tensor, converting the tensor to a float32 datatype, and normalizing the images. The normalization is applied using the mean and standard deviation values calculated earlier. The mean and standard deviation values for the data are 0.04443 and 0.02066, respectively.

The data is then loaded from the file directory and each image is opened, preprocessed, labeled, and stored in a dictionary where the keys for each image are the folder name and the values are a list of images. The data is then split into a training dataset and a validation dataset where 80% of the data is used to train the model and the remaining 20% is used to validate the model during the training process. The data is randomly shuffled for the 80/20 train/validation split. DataLoaders are then used to load the data for the model. These DataLoaders handle batching the data, randomly shuffling the data, and applying the previously defined collate function to keep track of the information for each input. The batch size used is 256 images per batch while making sure to not drop the final batch in case the final batch does not have a total of 256 images.

The final step for data preprocessing and data preparation includes the defining of a validation function. This function is designed and used to evaluate a model on the validation dataset and gather information on how the model is performing and on the processed data. This function will set the model to evaluation mode to disable training-only functions such as the nn.Dropout( ) function. It will also disable gradient computation to speed up computation while also reducing the memory being used because gradient calculations are not necessary while the model is being evaluated. The data batches provided by the DataLoader will be iterated over. This is where the inputs and labels are processed, and the information is stored in "infos" for the collate function. This function will also move the inputs and labels to the graphics processing unit (GPU) for computational purposes. The function will send the batched inputs through the model to generate output predictions for the classes. The function will extract the true labels from the label's tensor generated previously as well as the predictions from the model output. This is done to help calculate the F_1 scores for each class and obtain the information for the confusion matrices for each class (true positives, true negatives, false positives, and false negatives). The results will be accumulated in a dictionary that stores the details for each true label as a tuple that contains the folder, file, and row index the image came from. When the processing is finished for this step, the total number of rows processed are printed as a sanity check. This function is used for both training and validation datasets to see how well the model performs on both sets of data.

4.4 Hyperparameter Optimization

Following data preprocessing and preparation, the next necessary step was to perform a hyperparameter optimization. The hyperparameters defined for the model included the dropout rate for each layer, batch size, learning rate and weight decay for the Adam optimizer, kernel sizes for the layers, the number of kernels for each layer, and the pooling kernel sizes for each layer. A similar code to the previously mentioned steps above was used, however the primary difference was that the optimization code did not generate confusion matrices. This is because the main goal of hyperparameter optimization using the HyperOPT library is to perform a sweep to determine what hyperparameters will minimize some loss function.

In the case of this deep learning model, the goal was to determine the optimal F_1 score to obtain an accuracy of at least 80% for each class. However, the goal with F_1 scores is to get a value as close to one as possible and as far from zero as possible because a value of 1 indicates perfect precision and recall. Because of this, the goal of using HyperOPT will be to obtain a negative F_1 score. This will result in the model attempting to get as close to negative one as it can. This can then be converted to a positive value which will result in the most positive F_1 score, or a value as close to positive one as possible.

A hyperparameter space needs to be defined so that the model knows what parameters are necessary. Figure 4.7 shows a code snippet of how the hyperparameter space is defined. As previously mentioned, there are three different function types used

```
space = {
    'dropout_rate': hp.uniform('dropout_rate', 0.25, 0.75),
    'batch_size': hp.choice('batch_size', [64, 128, 256]),
    'learning_rate': hp.loguniform('learning_rate', np.log(0.001), np.log(0.1)),
    'weight_decay': hp.loguniform('weight_decay', np.log(1e-6), np.log(1e-4)),
    'kernel_size1': hp.choice('kernel_size1', [3, 5, 7]),
    'kernel_size2': hp.choice('kernel_size2', [3, 5, 7]),
    'pool_size': hp.choice('pool_size', [2, 3, 4]),
    'num_filters1': hp.choice('num_filters1', [16, 32, 64]),
    'num_filters2': hp.choice('num_filters2', [16, 32, 64])
}
```

Figure 4.7 Code snippet indicating the defined hyperparameter space for HyperOPT hyperparameter optimization.

for this model. Dropout rate uses hp.uniform which is a set of continuous values ranging from 0.25 to 0.75. Kernel sizes for the three convolutional layers use the hp.choice function and are a set of discrete values that are chosen randomly from the values of 3, 5, and 7. The pooling size hyperparameter uses hp.choice which will randomly choose from the values of 2, 3, and 4. The number of filters/number of kernels also use hp.choice and randomly chooses from the values 16, 32, and 64. The batch size hyperparameter uses hp.choice and randomly selects from the values 64, 128, and 256.

Learning rate and weight decay use hp.loguniform where learning rate is a set of continuous values that ranges from values of log(0.001) to log(0.1) and the weight decay ranges from $\log(1*10^{-6})$ to $\log(1*10^{-4})$. The learning rate and weight decay use the hp.loguniform function rather than hp.choice or hp.uniform primarily because these

values can affect how training the model will happen. This is because these values are multiplicative increases for the Adam optimizer rather than an additive increase. Using a logarithm allows for the values to be samples from a log-uniform distribution to ensure that the model will cover a large range of values that vary over orders of magnitude and allows for a more nuanced choice of values. The Adam optimizer is very sensitive to changes in learning rate and weight decay, so it is important that the values to be explored do not have a drastic increase in range. This is because the learning rate helps to converge at the global minima rather than a local minima and the weight decay helps the model prevent overfitting by penalizing large weights.

Following the hyperparameter space being defined and the model being setup to perform the hyperparameter sweep, the hyperparameter sweep optimization will be setup. A Trials object is initialized and is used to store the information about each trial. It stores all the information about the hyperparameters, the status of each trial, the F_1 score for the hyperparameters, and the time taken. An fmin function will be called to and contains arguments regarding the objective function to be minimized, the hyperparameter space, the number of evaluations, and the search algorithm being used. This information will all be stored in the Trials object. The output for this code will print out the best hyperparameters obtained and the corresponding negative F_1 score. In the case of this code, the model is attempting to determine the most negative F_1 score over two hundred trials using the TPE suggest algorithm which was previously explained.

4.5 Image Visualization

4.5.1 Grad-CAMs

As previously mentioned, CAMs are a visualization technique to demonstrate areas of the input data that are of interest to the deep learning model. Gradient-weighted Class Activation Maps (Grad-CAMs) are a generalized CAM approach that uses the gradient information from the output of the convolutional layers to determine what areas within the input image are of interest to the model. The gradients of the output with respect to the activation maps are pooled to obtain the weights which are then used to generate a weighted sum of the feature maps. Grad-CAMs can vary based on the input data. Because our input data is 1-dimensional, these maps will show the difference between pixel activation intensity and can show how features affect the model's output layer by layer. Because the input images differ based on the present bacterium/bacteria classes and the bandwidth they are imaged at, there is no individual Grad-CAM for each class. There are, however, Grad-CAMs for each of the images. This means that the model can show the areas of interest for each image and which parts of the image are more important for classification. It is important to note that not only bright portions of the images are important for classification. Areas with little to no reflection can also be important for image classification in addition to the wavelengths producing a spectral response.

4.5.2 False Color Images

The next step for image visualization is the usage of false color images. Because our input images are 1-dimensonal x-scans, they can be difficult to visualize and interpret

by just looking at the raw input images. As shown in Figure 5.2, the images can be stretched vertically to show light being reflected, but they are still difficult to differentiate

```
color_map = {
    "ATL1": (255, 0, 0),          # Red
    "ATL3": (0, 255, 0),          # Green
    "ATL28": (0, 0, 255),         # Blue
    "B27": (255, 255, 0),         # Yellow
    "ATL1_and_ATL3": (255, 0, 255),    # Magenta
    "ATL3_and_ATL28": (0, 255, 255),   # Cyan
    "ATL28_and_B27": (255, 165, 0),    # Orange
    "ATL1_and_ATL28": (128, 0, 128),   # Purple
    "ATL1_and_B27": (255, 105, 180),   # Hot Pink
    "ATL3_and_B27": (75, 0, 130),      # Indigo
    "default": (0, 0, 0),              # Black
}
```

Figure 4.8 Code snippet indicating the defined color mapping. Each bacterium/combination of bacteria is mapped to a specific color which will be used to create a false color image.

between. A solution to this is using false color images. False color images are where you apply a color map to the predictions so you can assign different colors to different classes. This is done by first creating a prediction map that has the information regarding the prediction, the file name the prediction came from, and the row the prediction came from. Following this step, a dictionary is initialized and maps each class, also known as a key, to a tuple of three integers that represent a color. Figure 4.8 indicates the color mapping used.

Once the color mapping is defined, the next step is to generate and save false color images. This is done by creating and setting up the save directory for the files and then defining the image dimensions. The image dimensions used are 2048x4096 pixels, but this can also manage to create false colors of the 1024x4096 images. The files and

predictions will be iterated over which can then create a false color image that has three color channels and is suitable for the RGB color mapping. The color selected is based on the prediction and is applied to the row the prediction came from. Once all the predictions for that file have been iterated over, the new false color image file will then be saved with the prefix “false_color_ ” followed by the original file name. The information regarding which files had which predictions can be printed out to specifically show which row contains which predictions.

4.6 Wavelength

Following optimization and visualization technique, determining the wavelength corresponding to the spectral pixel index is the final necessary step to obtain useful information from the images. One goal of this project is to determine what wavelengths are important for determining the difference between individual bacterium and combinations of bacteria. This is because light is differently for individual bacterium compared to the combinations of bacteria. It can be assumed that the increase of light is linear. Because of the way the imaging setup was designed, and the images were captured, the 550 nm wavelength is located at pixel index 820 and the 850 nm wavelength is located at pixel index 3450.

CHAPTER FIVE

TRAINING, VALIDATION, AND OPTIMIZATION OF CONVOLUTIONAL NEURAL NETWORK

5.1 Deep Learning Model Training and Validation

After data preparation and preprocessing, model definition and initialization, picking the initial hyperparameters, and creating the image visualization techniques, model training and validation was conducted. The 3 convolutional layer deep learning model was trained and validated over 10 epochs with a patience of 5 being used. A patience of 5 means that after 5 epochs of the model not improving, the training will stop early. Variables were created to store training loss, validation loss, accuracy, and training time for each epoch as well as overall training loss, validation loss, accuracy, and training time. Counters were used to store true positives, false negatives, and false positives for each of the 10 classes. Lists were used to store true and predicted labels to help analyze how the model was performing.

The model will iterate through the training loop and validation loop for each epoch. The training loop works by setting the model to training mode using the function model.train( ) which ensures certain training functions are used such as nn.Dropout( ). The training loop will make sure to reset the gradients and move the inputs and labels to the GPU. The input data will undergo the forward pass through the network followed by loss caluclation, backpropagation, and the updating of weights and biases within the model. This will track the training loss for each epoch and store it to help calculate the overall training loss. 80% of the randomly shuffeld data was used here for training.

Following the training loop, the model was then be set to evaluation mode using the function model.eval( ). This will turn functions used only for training off such as nn.Dropout( ). It will also disable gradient computation as this was only necessary for backpropagation which occurs during training. This loop also ensures the inputs and labels were moved to the GPU. The input data will undergo the forward pass through the network followed by loss calulcation and validation loss computation. This will apply a threshold that helps determine the predicted labels for to data. Any logit value greater than 0.5 will be set to a 1 and any logit value less than or equal to 0.5 will be set to 0. This is used with the one-hot encoded vectors. This loop also appends the true labels to the true label list and the predicted labels to the predicted label lost. This loop will track the validation loss for each epoch and store it to help calculate the overall validation loss. The remaining 20% of the randomly shuffled data was used here for validation.

Following training and validation, the data was analyzed using information stored during training and validation. Confusion matrices were generated by converting the true and predicted label lists to a numpy array and then plotted to show true labels versus the predicted labels. The Seaborn library was used to plot the confusion matrices.

The evaluation metrics were then calculated by using the true positives, false negatives, and false positives for each class. Precision and recall were calculated using the previously listed formulas to calculate the F_1 score for each class. An F_1 score for each class was caclculated in addition to a macro F_1 score so that way we could see if a certain class had more incorrect predictions than others to see what was affecting model

performance. Metrics were then logged to wandb.com. The model's state dictionary was saved to a file so that way the trained and validated model could be used on future data.

5.2 Deep Learning Model Optimization

Once the original model was implemented and we obtained output F_1 scores, the model then had to be optimized. HyperOPT was used to optimize the following hyperparameters: batch size, dropout rate, kernel size, learning rate, number of filters, pooling size, and weight decay. Using the hyperparameter space shown in Figure 5.6, a hyperparameter sweep was performed over one hundred evaluations using the TPE suggestion algorithm to determine best F_1 score. From the hyperparameter sweep, the following hyperparameter values were chosen: a batch size of 256, learning rate of 0.0013122122803649067, a weight decay of $8.74914341195787*10^{-6}$, a pooling size of 2 for each layer, a dropout rate of 0.25179056283341855 for each layer, a kernel size of 5 for the first layer, 32 kernels for the first layer, a kernel size of 7 for the second layer, and 64 filters for the second layer. The sweep was originally performed on a model with two convolutional layers to obtain the values. The F_1 macro score calculated was -0.9523260613091292. However, since we were looking for the lowest $-F_1$ score, this means the actual score would be 0.9523260613091292. This sweep took 23 hours, 34 minutes, and 14 seconds.

The following step was to use the given hyperparameters on the model to obtain F_1 scores per class and confusion matrices for each class as the determined F_1 score was the macro score rather than the individual class score. The new hyperparameter values were used and the model was trained and validated over 10 epochs and was performed

2 Layer F1 Scores												
	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Average	Standard Deviation
ATL 1	0.9522	0.9515	0.9510	0.9497	0.5922	0.9520	0.9537	0.9479	0.9517	0.9526	0.9155	0.1136
ATL 3	0.9636	0.9589	0.9627	0.9630	0.9634	0.9615	0.9624	0.9625	0.9625	0.9598	0.9620	0.0015
ATL 28	0.8979	0.8877	0.9041	0.9083	0.9092	0.8965	0.9084	0.9069	0.9031	0.9016	0.9024	0.0068
B 27	0.9766	0.8931	0.8902	0.9033	0.9052	0.8908	0.8930	0.9025	0.9030	0.8729	0.9031	0.0276
ATL 1 and ATL 3	0.9878	0.9866	0.9863	0.9897	0.9874	0.9867	0.9862	0.9875	0.9862	0.9881	0.9873	0.0011
ATL 1 and ATL 28	0.9781	0.9726	0.9696	0.9788	0.9644	0.9788	0.9700	0.9585	0.9690	0.9678	0.9708	0.0066
ATL 1 and B 27	0.9867	0.9845	0.9851	0.9855	0.9855	0.9862	0.9843	0.9834	0.9858	0.9847	0.9852	0.0010
ATL 3 and ATL 28	0.9670	0.9703	0.9722	0.9740	0.9769	0.9659	0.9774	0.9765	0.9785	0.9781	0.9737	0.0046
ATL 3 and B 27	0.9967	0.9969	0.9965	0.9942	0.9959	0.9965	0.9910	0.9939	0.9895	0.9960	0.9947	0.0026
ATL 28 and B 27	0.9901	0.9852	0.9858	0.9854	0.9887	0.9906	0.9894	0.9866	0.9879	0.9851	0.9875	0.0021
Macro F1 Score	0.9697	0.9587	0.9604	0.9632	0.9269	0.9606	0.9616	0.9606	0.9617	0.9587	0.9582	0.0114

Table 5.1 Summary table showing the individual and macro F_1 scores for a two-layer convolutional neural network with the average and standard deviation for each.

ten times. Table 5.1 shows the F_1 scores for each class for each run. A macro F_1 score was also shown for each run to show the overall model performance in addition to the individual class performance. The individual and macro scores were averaged, and the standard deviation was calculated for both the individual and macro scores. An average macro F_1 score of 0.9582 with a standard deviation of 0.0114 across the ten runs was calculated which was similar to the score obtained from the HyperOPT sweep.

After this was done to get a baseline hyperparameter sweep of the model, the number of layers were changed to try different architectures on the input data. There were ten training and validation runs over 10 epochs done for a model with one convolutional layer, three convolutional layers, and four convolutional layers. Changing the number of layers involved adding or removing four lines of code: a dropout function, convolution function, pooling function, and activation function. Each layer used the same dropout rate, pooling size, and activation function. The three- and four-layer models used kernels of size seven with a total of sixty-four kernels for the layer. The one-layer model used a kernel size of five with a total of thirty-two filters. Nothing was changed for the first

layer for the one-layer model besides the removal of the following layers. Table 5.2, Table 5.3, and Table 5.4 show the individual and macro F_1 scores for the one-layer, three-

layer, and four-layer models respectively across the ten training and validation runs. The average macro F_1 scores and standard deviation for the one-layer, three-layer, and four-

1 Layer F1 Scores												
	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Average	Standard Deviation
ATL 1	0.8119	0.8280	0.8318	0.7913	0.8113	0.8191	0.8149	0.8374	0.8191	0.8300	0.8195	0.0133
ATL 3	0.8631	0.8649	0.8635	0.8607	0.8618	0.8666	0.8656	0.8676	0.8695	0.8682	0.8652	0.0029
ATL 28	0.7846	0.7812	0.7817	0.7818	0.7765	0.7823	0.7835	0.7810	0.7863	0.7769	0.7816	0.0031
B 27	0.7038	0.7028	0.7033	0.7100	0.6932	0.6972	0.7131	0.7062	0.7031	0.7095	0.7042	0.0060
ATL 1 and ATL 3	0.8565	0.8669	0.8656	0.8641	0.8768	0.5970	0.8699	0.8793	0.8483	0.8793	0.8404	0.0861
ATL 1 and ATL 28	0.9258	0.9509	0.9306	0.9494	0.9329	0.9410	0.9436	0.9378	0.9312	0.9426	0.9386	0.0084
ATL 1 and B 27	0.8913	0.8683	0.8862	0.8605	0.8605	0.9135	0.8646	0.8904	0.8824	0.8887	0.8806	0.0170
ATL 3 and ATL 28	0.8349	0.8460	0.8391	0.9331	0.8268	0.8400	0.8383	0.8312	0.8380	0.8487	0.8476	0.0307
ATL 3 and B 27	0.9809	0.9808	0.9810	0.9791	0.9810	0.9820	0.9815	0.9890	0.9803	0.9825	0.9818	0.0027
ATL 28 and B 27	0.8998	0.8965	0.9058	0.8979	0.8898	0.8950	0.8937	0.9053	0.9082	0.8981	0.8990	0.0059
Macro F1 Score	0.8553	0.8586	0.8589	0.8628	0.8511	0.8334	0.8569	0.8625	0.8566	0.8625	0.8558	0.0087

Table 5.2 Summary table showing the individual and macro F_1 scores for a one-layer convolutional neural network with the average and standard deviation for each.

3 Layer F1 Scores												
	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Average	Standard Deviation
ATL 1	0.9658	0.9616	0.9641	0.9653	0.9655	0.9647	0.9656	0.9641	0.9648	0.9630	0.9645	0.0013
ATL 3	0.9726	0.9700	0.9700	0.9687	0.9716	0.9709	0.9717	0.9731	0.9724	0.9707	0.9712	0.0014
ATL 28	0.9290	0.9273	0.9253	0.9209	0.9178	0.9222	0.9292	0.9270	0.9300	0.9264	0.9255	0.0040
B 27	0.9039	0.9215	0.9041	0.9035	0.9151	0.9166	0.9205	0.9104	0.9238	0.9088	0.9128	0.0077
ATL 1 and ATL 3	0.9869	0.9900	0.9871	0.9873	0.9883	0.9868	0.9867	0.9862	0.9875	0.9863	0.9873	0.0011
ATL 1 and ATL 28	0.9801	0.9832	0.9794	0.9744	0.9843	0.9806	0.9789	0.9657	0.9824	0.9763	0.9785	0.0054
ATL 1 and B 27	0.9873	0.9901	0.9882	0.9867	0.9888	0.9853	0.9872	0.9864	0.9887	0.9857	0.9874	0.0015
ATL 3 and ATL 28	0.9779	0.9781	0.9794	0.9764	0.9789	0.9798	0.9789	0.9785	0.9796	0.9768	0.9784	0.0011
ATL 3 and B 27	0.9946	0.9957	0.9926	0.9924	0.9967	0.9927	0.9954	0.9887	0.9957	0.9936	0.9938	0.0023
ATL 28 and B 27	0.9794	0.9826	0.9824	0.9847	0.9794	0.9829	0.9814	0.9832	0.9817	0.9819	0.9820	0.0016
Macro F1 Score	0.9678	0.9700	0.9673	0.9660	0.9686	0.9683	0.9696	0.9663	0.9707	0.9670	0.9681	0.0016

Table 5.3 Summary table showing the individual and macro F_1 scores for a three-layer convolutional neural network with the average and standard deviation for each.

4 Layer F1 Scores												
	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Average	Standard Deviation
ATL 1	0.9398	0.9489	0.9392	0.9411	0.9499	0.9384	0.9404	0.9363	0.9421	0.9369	0.9413	0.0046
ATL 3	0.9590	0.9608	0.9585	0.9597	0.9575	0.9559	0.9599	0.9543	0.9522	0.9545	0.9572	0.0029
ATL 28	0.9192	0.9114	0.9070	0.9151	0.9174	0.9067	0.9156	0.9085	0.9127	0.9126	0.9126	0.0043
B 27	0.8959	0.9015	0.8942	0.9073	0.9049	0.8873	0.9057	0.8994	0.8951	0.9005	0.8992	0.0062
ATL 1 and ATL 3	0.9809	0.9825	0.9805	0.9823	0.9839	0.9815	0.9816	0.9825	0.9836	0.9839	0.9823	0.0012
ATL 1 and ATL 28	0.9641	0.9629	0.9407	0.9628	0.9523	0.9492	0.9572	0.9547	0.9528	0.9549	0.9552	0.0072
ATL 1 and B 27	0.9643	0.9482	0.9627	0.9680	0.9740	0.9642	0.9602	0.9658	0.9580	0.9757	0.9641	0.0079
ATL 3 and ATL 28	0.9560	0.9577	0.9586	0.9524	0.9527	0.9579	0.9583	0.9641	0.9626	0.9601	0.9580	0.0038
ATL 3 and B 27	0.9922	0.9923	0.9884	0.9903	0.9917	0.9892	0.9912	0.9901	0.9870	0.9907	0.9903	0.0017
ATL 28 and B 27	0.9726	0.9722	0.9675	0.9711	0.9716	0.9688	0.9746	0.9407	0.9660	0.9665	0.9672	0.0097
Macro F1 Score	0.9544	0.9538	0.9497	0.9550	0.9556	0.9499	0.9545	0.9496	0.9512	0.9536	0.9527	0.0024

Table 5.4 Summary table showing the individual and macro F_1 scores for a four-layer convolutional neural network with the average and standard deviation for each.

layer models are 0.8558 and 0.0087, 0.9681 and 0.0016, and 0.9527 and 0.0024 respectively. This indicates that while similar, the three-layer model performed the best overall. An example of confusion matrices for the three-layer model is shown in Figure 5.1. This set of confusion matrices shows true negatives, false positives, false negatives,

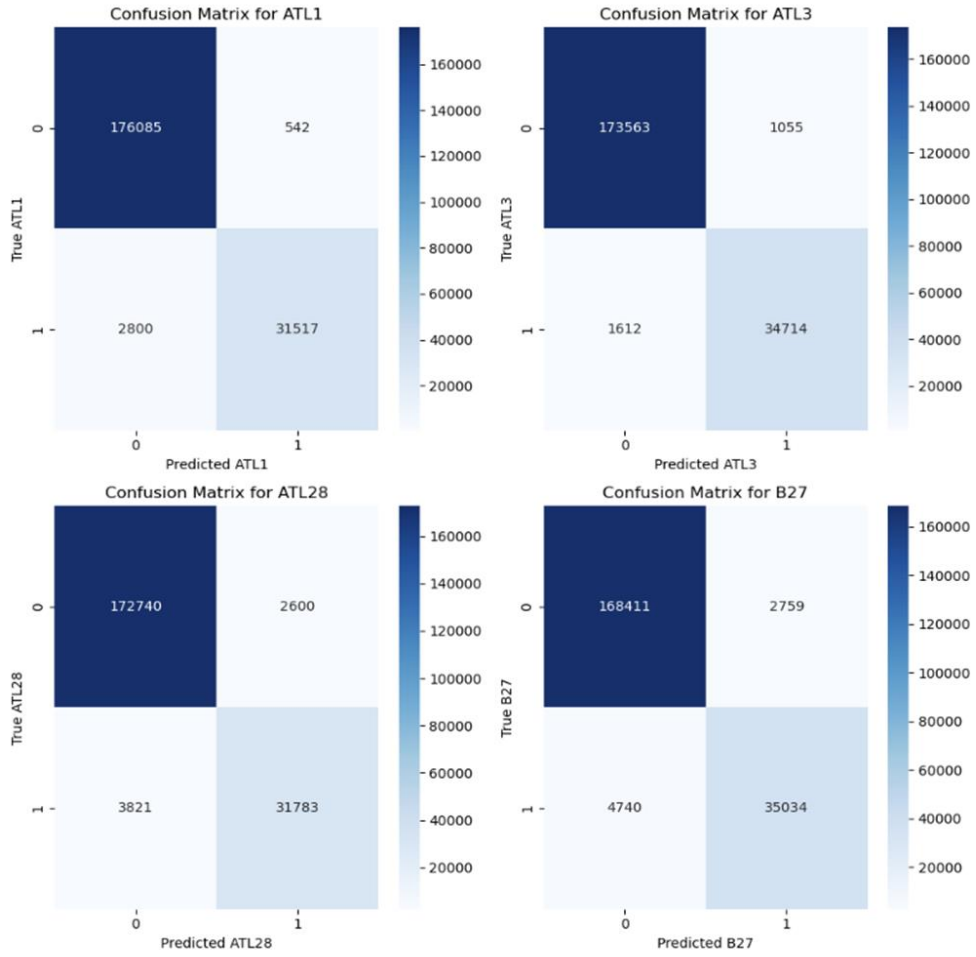

Figure 5.1 Figure showing confusion matrices for classes ATL 1, ATL 3, ATL 28, and B 27.

and true positives for individual classes.

Each model seemed to perform better on the images with combinations of bacteria rather than the individual bacterium images. This is because of the amount of data for

each class and the similarities between data. As seen in Figure 4.5, the individual bacterium reflect light similarly with slight differences. The combination of bacteria also reflect light similarly, but they are not all overlapped as the individual bacterium are. Table 4.1 indicates the amount of data used for each class with the individual bacterium having a larger amount of data compared to the combinations. The model performing better on the combinations is most likely because the model has more images to train and validate on for the individual bacterium which allows it to learn what is and what is not the individual bacterium.

5.3 Image Visualization

5.3.1 Grad-CAMs

Grad-CAMs were generated for a portion of the entire dataset following the first, second, and third convolutional layer. The Grad-CAM images indicate which areas in the images are of interest. A higher Activation Intensity indicates a more important feature. The original input is 1x4096. Following the first convolutional layer, a Grad-CAM is output showing which areas of the image are of interest. Feature Index is a 1 to 1 with Pixel Index on the input images. Following the convolutional layer, pooling occurs to downsample the image while still keeping track of important features. That means the input for the next layer will be 1x2048. Following the convolutional layer, another Grad-CAM is generated indicating the reduced size. Another pooling occurs to downsample

again meaning the next input is 1x1024. Another Grad-CAM is generated. Figure 5.2 shows an input image of bacteria ATL 28 on row 291 from file ATL28-3_580-600.bmp, bacteria B 27 on row 898 from file B27-2_580-600.bmp, and a combination of bacteria

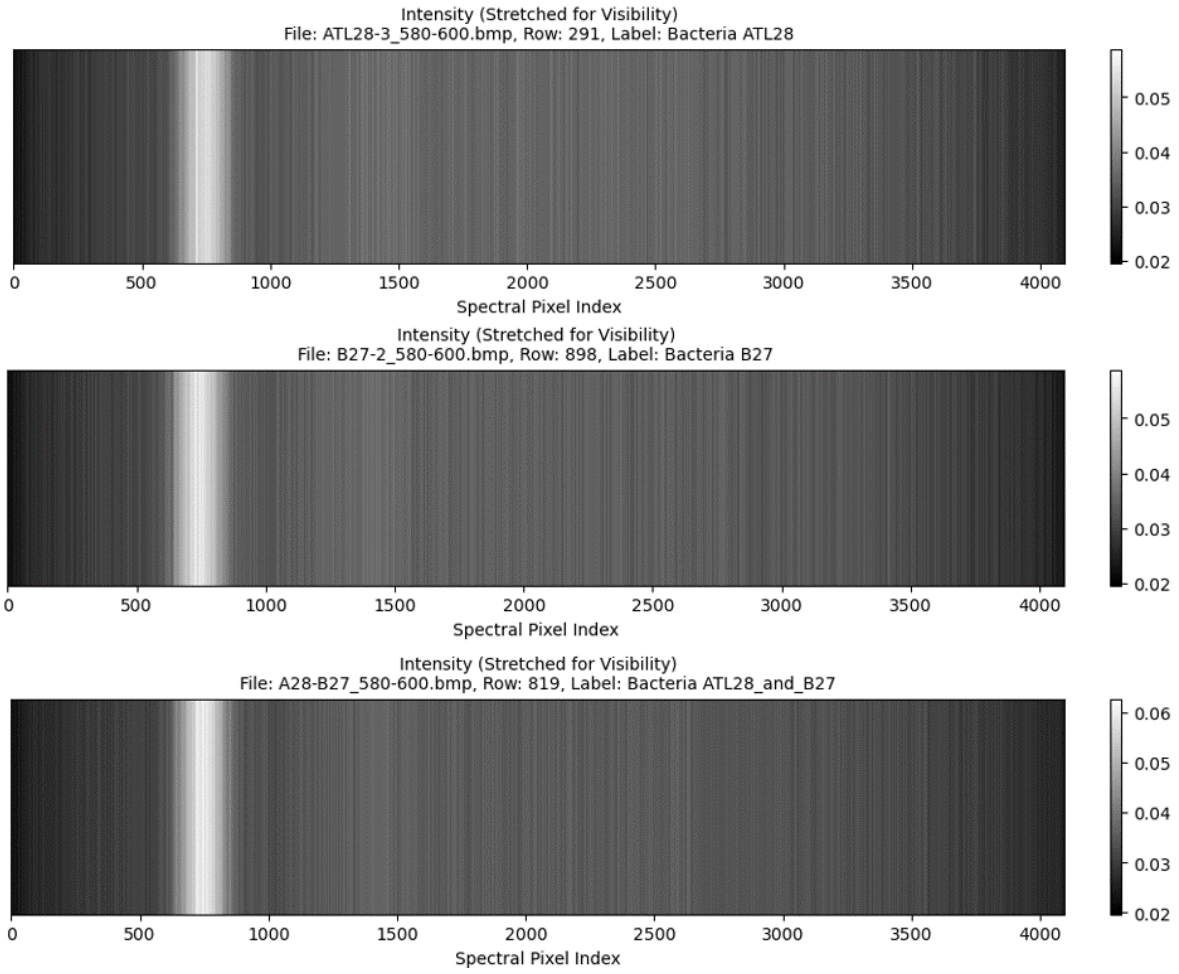

Figure 5.2 Figure showing three different input images from three different classes.

ATL 28 and B 27 on row 819 from file A28-B27_580-600.bmp. The three different images were imaged from 580 nm to 600 nm to show the similarities beteween the three at the same wavelenght range. Each image was stretched vertically for visualization purposes. The lighter the color in the image indicates more light being reflected. Grad-

CAMs were generated for each of the input images in Figure 5.2 following the first, second, and third convolutional layers. Figure 5.3 shows the Grad-CAM for each of the three classes following the first convolutional layer, Figure 5.4 shows the Grad-CAMs for each of the three classes following the second convolutional layer, and Figure 5.5 shows

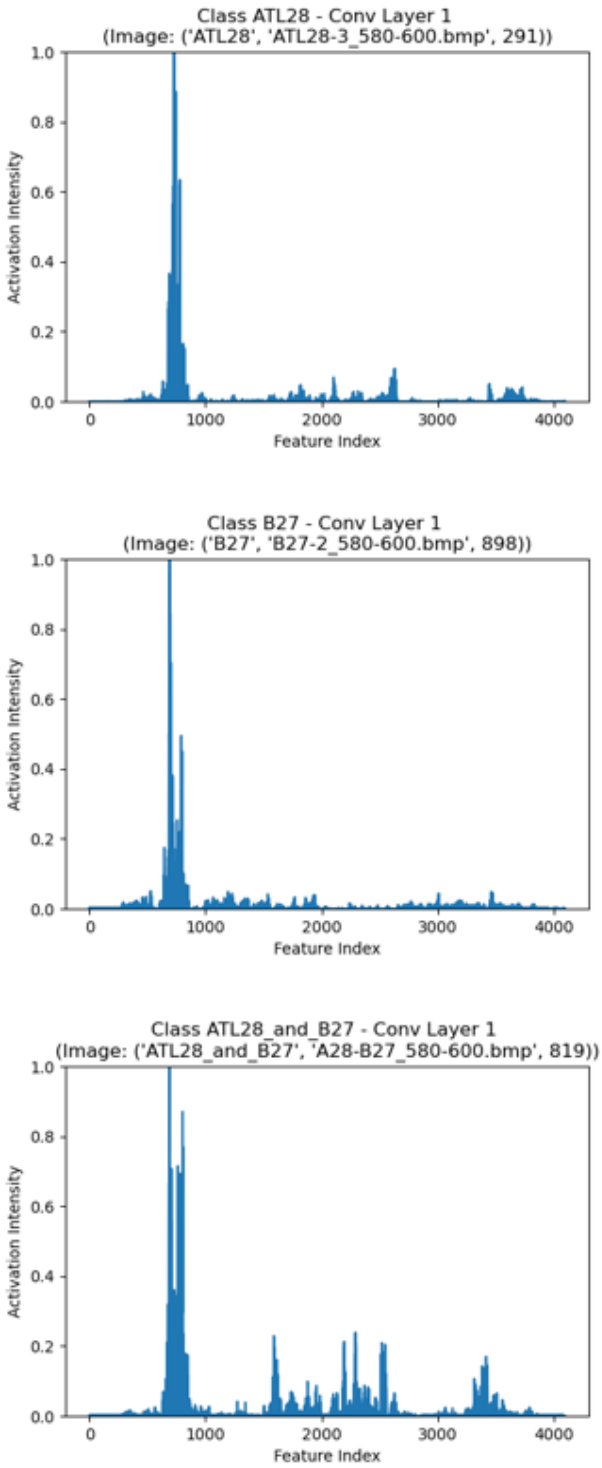

Figure 5.3 Grad-CAMs for class ATL 28, B 27, and ATL28_and_B27 for the first convolutional layer.

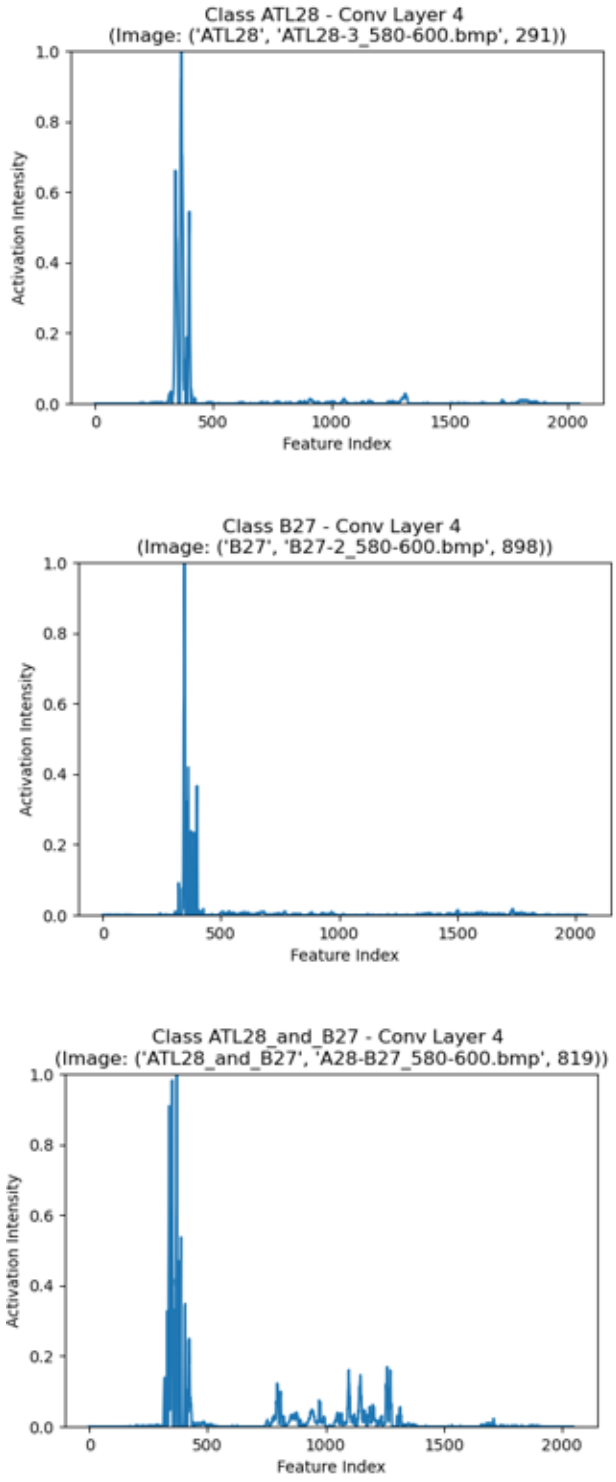

Figure 5.4 Grad-CAMs for class ATL 28, B 27, and ATL28_and_B27 for the second convolutional layer.

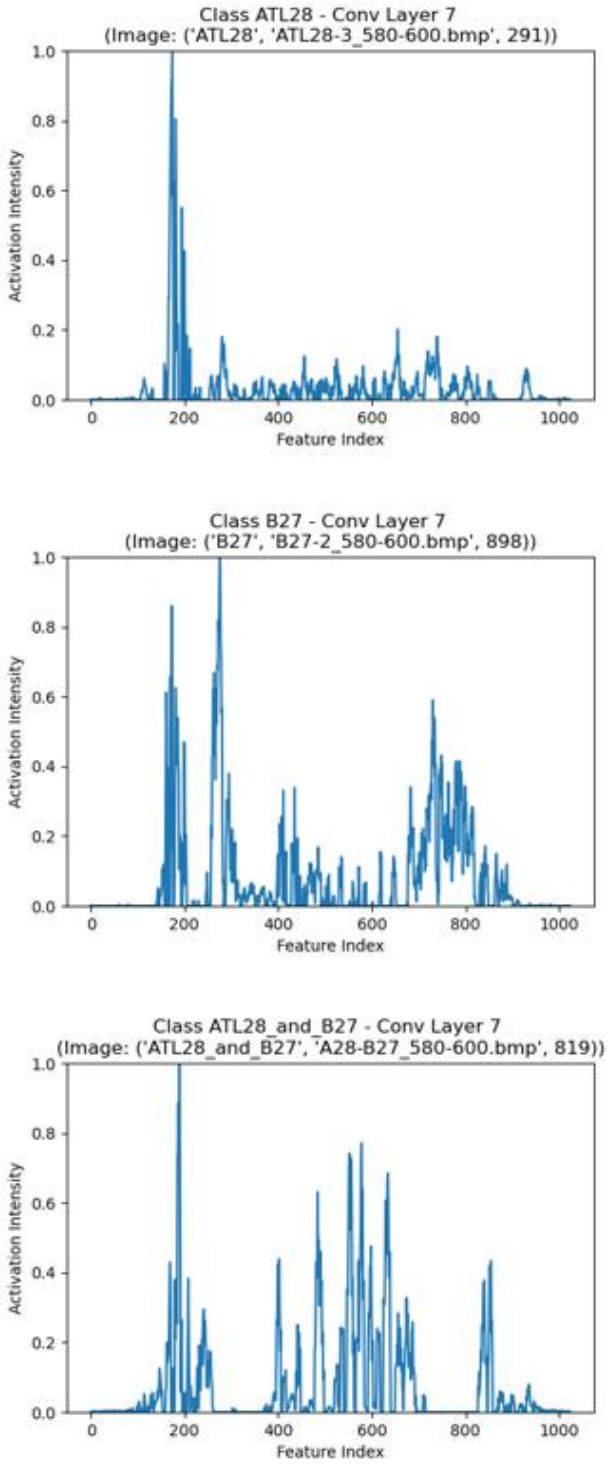

Figure 5.5 Grad-CAMs for class ATL 28, B 27, and ATL28_and_B27 for the third convolutional layer.

the Grad-CAMs for each of the three classes following the third convolutional layer. Figure 5.3, Figure 5.4, and Figure 5.5 show that the input images from Figure 5.2 look very similar, but there are a few differences between the images that the model can distinguish between.

Figure 5.3 and Figure 5.4 indicate that around Feature Index 400-500 seem to be very important information to the model when differentiating between the three classes. For bacteria ATL 28 and B 27, there seems to be some information the model finds somehwat important outside of the 400-500 Feature Index range. However, for the ATL28_and_B27 class, the information outside of the 400-500 Feature Index range seems to be more important to the model for classification compared to the individual bacterium imges. Figure 5.4 shows the Grad-CAMs right before classification is made and indicates the features that are important when it comes to classification. For example, bacteria B 27 tends to only have high activation intensity between the 400-500 Feature Index, but there is a larger range of high activation intensity following the third convolutional layer. However, ATL 28 in comparison still has relatively low activation intensity for a majority of the image following the third convolutional layer.

For the ATL28_and_B27 class, there seems to be higher activation intensity across the entire image for each of the three convolutional layers. The importance of deeep learning with these images show that while the input images look similar, the model is still able to differentiate between them even with subtle differences.

5.3.2 False Color Images

The final step for image visualization was the generation of false color images. As previously mentioned, false color images are important for our data as the input images are 1-dimensional greyscale images. A color mapping was assigned to each class with the information regarding each image and their prediction being stored and saved by the model during training and validation. Figure 5.6 shows an example of a false color image generated by the model using the color mapping shown in Figure 4.8. The false color

Figure 5.6 Example of a false color image of input file B27-2_580.600.bmp.

images were generated from the validation dataset, or only 20% of the total input data. This means that some files were recreated with very few amounts of false color images. Figure 5.6, for example, only shows a false color of bacteria B 27 which has yellow assigned to it.

It is very important to note that the purpose of false color images was to recreate the input image to obtain spatial information about the input image. Due to the nature of our input data, only the first column of pixels contains spatial information within the files. The remaining 4095 pixels contain spectral information. That means that when the false color images were created, only the first pixel is relevant for the false color images. For visualization purposes, the entire row of the false color image has the color of the labeled bacteria as our eyes cannot see an object that is one pixel in width and be able to distinguish between different colors.

While Figure 5.6 shows an example of a false color image from the input file B27-2_580-600.bmp, it still only shows one false color image within a file that is size 2048x4096. To better demonstrate and understand the false color images, false color images were created with the entire dataset. However, the model can make both correct and incorrect predictions which means some of the false color images will look correct and some will look incorrect. This can be a result of the quality of the input data, how similar the images are, or how unsure the model was with the prediction. This means that when looking at the false color images, sometimes and inference will need to be made to determine what bacteria is where in the image and what the most logical outcome is. This is because that while deep learning models can make accurate predictions, they are still algorithms capable of making mistakes. Figure 5.7 shows an example of a filled out false color image that was created as expected whereas Figure 5.8 shows an example of a filled out false color image, but there are many incorrect predictions made for the image. It is

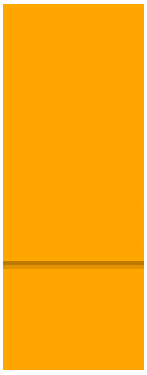

Figure 5.7 Example of a false color image of input file A28-B27_600-650.bmp indicating the presence of ATL 28 and B 27 imaged from 600 to 650 nm.

Figure 5.8 Example of a false color image of input file A28-B27_700-720.bmp indicating the presence of ATL 28 and B 27 imaged from 700 to 720 nm.

still important to note that with these images, the first column is the only column that currently provides spatial information. Figure 5.7 shows a mostly correct false color image, and the user can infer that the misclassifications were most likely to be the class ATL28_and_B27, but Figure 5.8 is harder to tell because of the amount of misclassifications.

5.4 Wavelength

After the creation of the false color images, the final step was to determine the corresponding wavelengths. While different images were captured using different

bandwidths, the increase of wavelength from pixel to pixel was the same for each image. The hyperspectral imaging system was calibrated and obtained an equation of

$$\lambda(p) = 536.537540 + 0.080543p - 0.000003p^2$$

where p is the pixel and λ is wavelength in nm. To determine the wavelength change per pixel, the difference was calculated between the wavelength at pixel p and pixel p+1. Substituting the values p and p+1 into the equation, the two following equations were obtained.

$$\lambda(p) = 536.537540 + 0.080543p - 0.000003p^2$$

$$\lambda(p+1) = 536.537540 + 0.080543(p+1) - 0.000003(p+1)^2$$

The difference between the two equations is calculated using the equation

$$\Delta\lambda = \lambda(p+1) - \lambda(p)$$

which results in

$$\Delta\lambda = 0.080540 - 0.000006p$$

and indicates that the change in wavelength is not constant when increasing in pixel index values. As the pixel index increases, the increase in wavelength decreases slightly per pixel because of this calibration curve. Figure 5.9 shows the wavelength for each pixel using the calibration curve equation and Figure 5.10 shows the derivative of the calibration curve to indicate the change in wavelength per pixel.

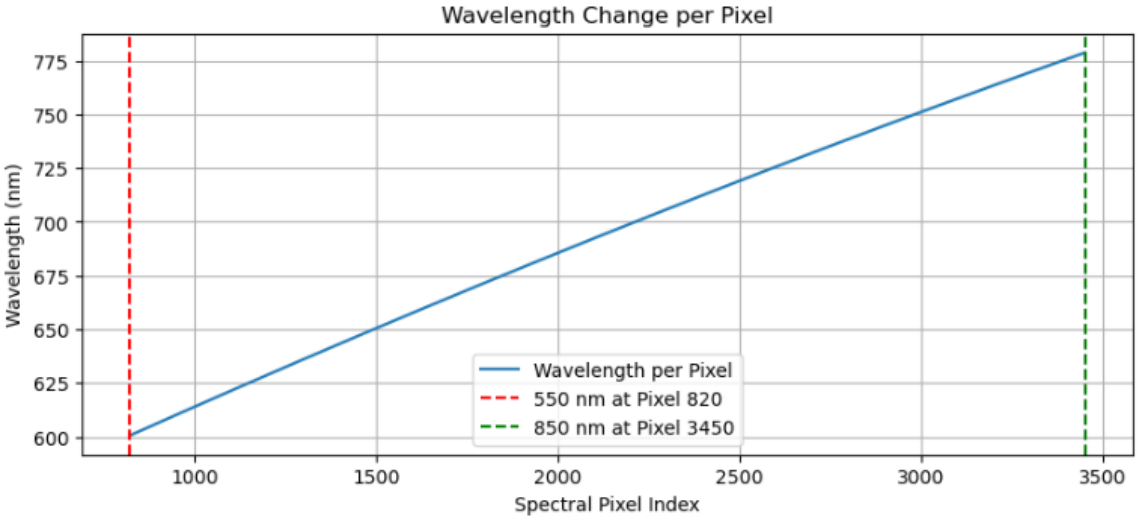

Figure 5.9 Graph showing the relationship between wavelength and pixel index using the calibration curve equation.

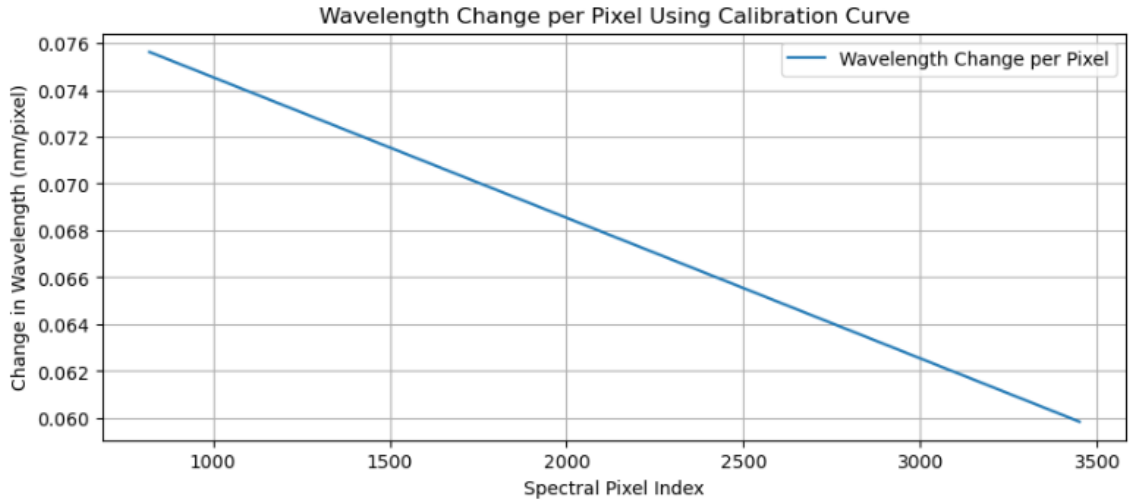

Figure 5.10 Graph showing change in wavelength per pixel based on the derivative of the calibration curve.

It is important to note that while the images are 1x4096 pixels, 550 nm is located at pixel index 820 and 850 nm is located at pixel 3450. The starting pixel value, or pixel index 820, is subtracted from the pixel of interest to normalize the pixel index. Figure 5.9 shows the wavelength change per pixel from pixel index 820 to pixel index 3450. The wavelength change for each image is the same regardless of the bandwidth being used.

The different bandwidths being used indicates the range of wavelengths that each individual pixel can detect. Larger bandwidths will indicate more light being reflected for within that range of light being used to illuminate the sample.

CHAPTER SIX

CONCLUSIONS AND FUTURE WORKS

Hyperspectral imaging is a unique imaging method that allows for the collection and analysis of information that can help with uniquely identifying different objects based on their spectral reflectance. Deep learning has recently become a standard approach for image classification tasks; however, there are some issues that must be considered and addressed when designing a model from scratch or using a pretrained model. The combination of these two methods allows for the processing of spectral data and obtaining accurate predictions in the classification of different types of bacteria. The results above show that this designed from scratch deep learning model has the following capabilities:

1. Accepting 1-diemnsional hyperspectral imaging data imaged from 550 to 850 nm.
2. Preprocess and prepare the data to pass the data through the network.
3. Generate predictions for the input data with an overall model accuracy of 96.81%.
4. Ability to have hyperparameters optimized using the TPE suggestion algorithm from HyperOPT indicating the model is well designed.
5. Generate training loss, validation loss, F_1 scores, and confusion matrices for each class.
6. Generate Grad-CAMs for each convolution layer for each bacteria class.
7. Generate false color images of the input hyperspectral images.

While the model can create false color images, the input hyperspectral images are still 1-dimensional. This means that the false colors images as of now are unable to

provide much spatial information. A future step for this project is to update the hyperspectral imaging system to generate 2- or 3-dimensional images. This will require the model to be updated as many of the functions used, such as nn.Conv1d( ), are only usable on 1-dimensional inputs. If the information is 2- or 3-dimensional, the functions will need to be changed to their higher dimension counterparts, such as nn.Conv2d( ) or nn.Conv3d( ). This will require substantial changes in the model.

In addition to updating the functions within the model, another future step with the updated hyperspectral images will be to have file names indicate not only the wavelength, but also the location the image was captured. This can be used for the model for future false color images as we will be able to stitch together the false color images to better show and understand the topology and morphology of the samples. This will also allow for us to create images that will be outside the field of view of one scan and get a better picture of the bacteria.

Finally, the goal will be to also update the Grad-CAM outputs as they are currently for 1-dimensional images. With the 1-dimensional Grad-CAMs, only the pixel activation intensity is shown. However, with updates to the data collection process, future Grad-CAMs will be shown more as heat maps than pixel activation intensity plots. That will help indicate which areas of the image are more important for bacteria classification and will provide more information about how the model classifies the different bacteria.

REFERENCES

[1] D. Uzun, S. Sezen, R. Ozyurt, M. Atlar, and O. Turan, “A CFD study: Influence of biofouling on a full-scale submarine,” Applied Ocean Research, vol. 109, p. 102561, Apr. 2021. doi:10.1016/j.apor.2021.102561

[2] D. Uzun, S. Sezen, M. Atlar, and O. Turan, “Effect of biofouling roughness on the full-scale powering performance of a submarine,” Ocean Engineering, vol. 238, p. 109773, Oct. 2021. doi:10.1016/j.oceaneng.2021.109773

[3] M. A. Calin, S. V. Parasca, D. Savastru, and D. Manea, “Hyperspectral Imaging in the medical field: Present and future,” Applied Spectroscopy Reviews, vol. 49, no. 6, pp. 435–447, Dec. 2013. doi:10.1080/05704928.2013.838678

[4] Y. Zhang et al., “Applications of hyperspectral imaging in the detection and diagnosis of solid tumors,” Translational Cancer Research, vol. 9, no. 2, pp. 1265–1277, Feb. 2020. doi:10.21037/tcr.2019.12.53

[5] G. Camps-Valls and L. Bruzzone, “Kernel-based methods for hyperspectral image classification,” IEEE Transactions on Geoscience and Remote Sensing, vol. 43, no. 6, pp. 1351–1362, Jun. 2005. doi:10.1109/tgrs.2005.846154

[6] “What is deep learning?,” NVIDIA Data Science Glossary, https://www.nvidia.com/en-us/glossary/deep-learning/ (accessed May 6, 2024).

[7] “Deep learning,” MATLAB & Simulink, https://www.mathworks.com/discovery/deep-learning.html (accessed May 7, 2024).

[8] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” Nature, vol. 521, no. 7553, pp. 436–444, May 2015. doi:10.1038/nature14539

[9] N. McCullum, “Deep Learning Neural Networks explained in plain English,” freeCodeCamp.org, https://www.freecodecamp.org/news/deep-learning-neural-networks-explained-in-plain-english/ (accessed May 9, 2024).

[10] J. Schmidhuber, “Deep learning in neural networks: An overview,” Neural Networks, vol. 61, pp. 85–117, Jan. 2015. doi:10.1016/j.neunet.2014.09.003

[11] R. Sharma, “Study of supervised learning and unsupervised learning,” International Journal for Research in Applied Science and Engineering Technology, vol. 8, no. 6, pp. 588–593, Jun. 2020. doi:10.22214/ijraset.2020.6095

[12] N. Sharma, V. Jain, and A. Mishra, “An analysis of convolutional neural networks for Image Classification,” Procedia Computer Science, vol. 132, pp. 377–384, 2018. doi:10.1016/j.procs.2018.05.198

[13] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, “A survey of Convolutional Neural Networks: Analysis, applications, and prospects,” IEEE Transactions on Neural Networks and Learning Systems, vol. 33, no. 12, pp. 6999–7019, Dec. 2022. doi:10.1109/tnnls.2021.3084827

[14] Y. Guo et al., “Deep Learning for Visual Understanding: A Review,” Neurocomputing, vol. 187, pp. 27–48, Apr. 2016. doi:10.1016/j.neucom.2015.09.116

[15] B. J. Wythoff, “Backpropagation Neural Networks,” Chemometrics and Intelligent Laboratory Systems, vol. 18, no. 2, pp. 115–155, Feb. 1993. doi:10.1016/0169-7439(93)80052-j

[16] J. Zhang, Q. Wang, and W. Shen, “Hyper-parameter optimization of multiple machine learning algorithms for molecular property prediction using hyperopt library,” Chinese Journal of Chemical Engineering, vol. 52, pp. 115–125, Dec. 2022. doi:10.1016/j.cjche.2022.04.004

[17] A. Korotcov, V. Tkachenko, D. P. Russo, and S. Ekins, “Comparison of Deep Learning with multiple machine learning methods and metrics using diverse drug discovery data sets,” Molecular Pharmaceutics, vol. 14, no. 12, pp. 4462–4475, Nov. 2017. doi:10.1021/acs.molpharmaceut.7b00578

[18] A. Koutsoukas, K. J. Monaghan, X. Li, and J. Huan, “Deep-learning: Investigating deep neural networks hyper-parameters and comparison of performance to shallow methods for modeling bioactivity data,” Journal of Cheminformatics, vol. 9, no. 1, Jun. 2017. doi:10.1186/s13321-017-0226-y

[19] D. Manolakis et al., “Longwave infrared hyperspectral imaging: Principles, progress, and challenges,” IEEE Geoscience and Remote Sensing Magazine, vol. 7, no. 2, pp. 72–100, Jun. 2019. doi:10.1109/mgrs.2018.2889610

[20] G. Camps-Valls and L. Bruzzone, “Kernel-based methods for hyperspectral image classification,” IEEE Transactions on Geoscience and Remote Sensing, vol. 43, no. 6, pp. 1351–1362, Jun. 2005. doi:10.1109/tgrs.2005.846154

[21] Y. Chen, Z. Lin, X. Zhao, G. Wang, and Y. Gu, “Deep learning-based classification of Hyperspectral Data,” IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 7, no. 6, pp. 2094–2107, Jun. 2014. doi:10.1109/jstars.2014.2329330

[22] G. Foca et al., “The potential of spectral and hyperspectral-imaging techniques for bacterial detection in food: A case study on lactic acid bacteria,” Talanta, vol. 153, pp. 111–119, Jun. 2016. doi:10.1016/j.talanta.2016.02.059

[23] Q. Li et al., “Review of Spectral Imaging Technology in Biomedical Engineering: Achievements and challenges,” Journal of Biomedical Optics, vol. 18, no. 10, p. 100901, Oct. 2013. doi:10.1117/1.jbo.18.10.100901

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional Neural Networks,” Communications of the ACM, vol. 60, no. 6, pp. 84–90, May 2017. doi:10.1145/3065386

[25] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, “Learning hierarchical features for scene labeling,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 35, no. 8, pp. 1915–1929, Aug. 2013. doi:10.1109/tpami.2012.231

[26] J. Tompson, A. Jain, Y. LeCun, and C. Bregler, “Joint training of a convolutional network and a graphical model for human pose estimation,” arXiv.org, https://arxiv.org/abs/1406.2984 (accessed May 15, 2024).

[27] C. Szegedy et al., “Going deeper with convolutions,” 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Jun. 2015. doi:10.1109/cvpr.2015.7298594

[28] T. Mikolov, A. Deoras, D. Povey, L. Burget, and J. Cernocky, “Strategies for training large scale neural network language models,” 2011 IEEE Workshop on Automatic Speech Recognition & Understanding, Dec. 2011. doi:10.1109/asru.2011.6163930

[29] G. Hinton et al., “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups,” IEEE Signal Processing Magazine, vol. 29, no. 6, pp. 82–97, Nov. 2012. doi:10.1109/msp.2012.2205597

[30] T. N. Sainath, A. Mohamed, B. Kingsbury, and B. Ramabhadran, “Deep convolutional neural networks for LVCSR,” 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, May 2013. doi:10.1109/icassp.2013.6639347

[31] J. Ma, R. P. Sheridan, A. Liaw, G. E. Dahl, and V. Svetnik, “Deep neural nets as a method for quantitative structure–activity relationships,” Journal of Chemical Information and Modeling, vol. 55, no. 2, pp. 263–274, Feb. 2015. doi:10.1021/ci500747n

[32] T. Ciodaro, D. Deva, J. M. de Seixas, and D. Damazio, “Online particle detection with neural networks based on topological calorimetry information,” Journal of Physics: Conference Series, vol. 368, p. 012030, Jun. 2012. doi:10.1088/1742-6596/368/1/012030

[33] “Higgs Boson Machine Learning Challenge,” Kaggle, https://www.kaggle.com/c/higgs-boson (accessed May 15, 2024).

[34] M. Helmstaedter et al., “Connectomic reconstruction of the inner plexiform layer in the mouse retina,” Nature, vol. 500, no. 7461, pp. 168–174, Aug. 2013. doi:10.1038/nature12346

[35] S. Das, “CNN architectures: Lenet, alexnet, VGG, googlenet, ResNet and more,” Medium, https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5 (accessed May 20, 2024).

[36] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Jun. 2016. doi:10.1109/cvpr.2016.90

[37] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization,” Journal of Machine Learning Research, vol. 13, pp. 281–305, Feb. 2012.

[38] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. D. Cox, “Hyperopt: A python library for model selection and hyperparameter optimization,” Computational Science & Discovery, vol. 8, no. 1, p. 014008, Jul. 2015. doi:10.1088/1749-4699/8/1/014008

[39] S. Watanabe, “Tree-structured Parzen estimator: Understanding its algorithm components and their roles for better empirical performance,” arXiv.org, https://arxiv.org/abs/2304.11127 (accessed May 8, 2024).

[40] T. Saito and M. Rehmsmeier, “The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets,” PLOS ONE, vol. 10, no. 3, Mar. 2015. doi:10.1371/journal.pone.0118432

[41] Haibo He and E. A. Garcia, “Learning from Imbalanced Data,” IEEE Transactions on Knowledge and Data Engineering, vol. 21, no. 9, pp. 1263–1284, Sep. 2009. doi:10.1109/tkde.2008.239

[42] D. G. Altman and J. M. Bland, “Statistics notes: Diagnostic tests 1: Sensitivity and specificity,” BMJ, vol. 308, no. 6943, pp. 1552–1552, Jun. 1994. doi:10.1136/bmj.308.6943.1552

[43] C. Goutte and E. Gaussier, “A probabilistic interpretation of precision, recall and F-score, with implication for evaluation,” Lecture Notes in Computer Science, pp. 345–359, 2005. doi:10.1007/978-3-540-31865-1_25

[44] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, “Learning deep features for discriminative localization,” 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Jun. 2016. doi:10.1109/cvpr.2016.319

[45] G. Adhane, M. M. Dehshibi, and D. Masip, “A deep convolutional neural network for classification of aedes albopictus mosquitoes,” IEEE Access, vol. 9, pp. 72681–72690, 2021. doi:10.1109/access.2021.3079700

[46] R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra, ‘Grad-CAM: Why did you say that?’, ArXiv, vol. abs/1611.07450, 2016.

[47] A. W. Decho et al., “Sediment properties influencing upwelling spectral reflectance signatures: The ‘biofilm gel effect,’” Limnology and Oceanography, vol. 48, no. 1part2, pp. 431–443, Jan. 2003. doi:10.4319/lo.2003.48.1_part_2.0431

[48] S. L. Broschat et al., “Optical reflectance assay for the detection of Biofilm Formation,” Journal of Biomedical Optics, vol. 10, no. 4, p. 044027, 2005. doi:10.1117/1.1953347

[49] B. Jesus et al., “Spectral-radiometric analysis of taxonomically mixed microphytobenthic biofilms,” Remote Sensing of Environment, vol. 140, pp. 196–205, Jan. 2014. doi:10.1016/j.rse.2013.08.040

[50] D. Vázquez-Nion, E. Fuentes, and B. Prieto, “Effect of inorganic carbon concentration on the development of subaerial phototrophic biofilms on granite,” Coatings, vol. 10, no. 11, p. 1049, Oct. 2020. doi:10.3390/coatings10111049

[51] “NN-svg,” NN SVG, https://alexlenail.me/NN-SVG/AlexNet.html (accessed Jun. 4, 2024).

[52] “CONV1D¶,” Conv1d - PyTorch 2.3 documentation, https://pytorch.org/docs/stable/generated/torch.nn.Conv1d.html (accessed Jun. 1, 2024).

[53] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in Proceedings of the 27th international conference on machine learning (ICML-10), 2010, pp. 807–814.

[54] “Leakyrelu¶,” LeakyReLU - PyTorch 2.3 documentation, https://pytorch.org/docs/stable/generated/torch.nn.LeakyReLU.html (accessed Jun. 1, 2024).

[55] J. Xu, Z. Li, B. Du, M. Zhang, and J. Liu, “Reluplex made more practical: Leaky relu,” 2020 IEEE Symposium on Computers and Communications (ISCC), Jul. 2020. doi:10.1109/iscc50000.2020.9219587

[56] “MaxPool1d¶,” MaxPool1d - PyTorch 2.3 documentation, https://pytorch.org/docs/stable/generated/torch.nn.MaxPool1d.html (accessed Jun. 1, 2024).

[57] “Dropout¶,” Dropout - PyTorch 2.3 documentation, https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html (accessed Jun. 1, 2024).

[58] “About Us,” Weights & Biases, https://wandb.ai/site/company/about-us (accessed Jun. 7, 2024).