Exploring Hyperspectral Imaging and 3D Convolutional Neural Network for Stress Classification in Plants

by

Nooshin Noshiri

A Thesis submitted to the Faculty of Graduate Studies of
The University of Winnipeg
in partial fulfillment of the requirements of the degree of

Master of Science

Department of Applied Computer Science
University of Winnipeg
Winnipeg, Manitoba, Canada

Exploring Hyperspectral Imaging and 3D Convolutional Neural Network for Stress Classification in Plants

by

Nooshin Noshiri

Supervisory Committee

Dr. Christopher J. Henry, Co-Supervisor
(Computer Science Department, University of Manitoba)

Dr. Michael A. Beck, Co-Supervisor
(Applied Computer Science Department, The University of Winnipeg)

Dr. Christopher P. Bidinosti, Internal Member
(Physics Department, The University of Winnipeg)

Dr. Christopher C. Storie, External Member
(Geography Department, The University of Winnipeg)

Abstract

Hyperspectral imaging (HSI) has emerged as a transformative technology in imaging, characterized by its ability to capture a wide spectrum of light, including wavelengths beyond the visible range. This approach significantly differs from traditional imaging methods such as RGB imaging, which uses three color channels, and multispectral imaging, which captures several discrete spectral bands. Through this approach, HSI offers detailed spectral signatures for each pixel, facilitating a more nuanced analysis of the imaged subjects. This capability is particularly beneficial in applications like agricultural practices, where it can detect changes in physiological and structural characteristics of crops. Moreover, the ability of HSI to monitor these changes over time is advantageous for observing how subjects respond to different environmental conditions or treatments.

However, the high-dimensional nature of hyperspectral data presents challenges in data processing and feature extraction. Traditional machine learning algorithms often struggle to handle such complexity. This is where 3D Convolutional Neural Networks (CNNs) become valuable. Unlike 1D-CNNs, which extract features from spectral dimensions, and 2D-CNNs, which focus on spatial dimensions, 3D CNNs have the capability to process data across both spectral and spatial dimensions. This makes them adept at extracting complex features from hyperspectral data. In this thesis, we explored the potency of HSI combined with 3D-CNN in agriculture domain where plant health and vitality are paramount. To evaluate this, we subjected lettuce plants to varying stress levels to assess the performance of this method in classifying the stressed lettuce at the early stages of growth into their respective stress-level groups.

For this study, we created a dataset comprising 88 hyperspectral image samples of stressed lettuce. Utilizing Bayesian optimization, we developed 350 distinct 3D-CNN models to assess the method. The top-performing model achieved a 75.00% test accuracy. Additionally, we addressed the challenge of generating valid 3D-CNN models in the Keras Tuner library through meticulous hyperparameter configuration. Our investigation also extends to the role of individual channels and channel groups within the color and near-infrared spectrum in predicting results for each stress-level group. We observed that the red and green spectra have a higher influence on the prediction results. Furthermore, we conducted a comprehensive review of 3D-CNN-based classification techniques for diseased and defective crops using non-UAV-based hyperspectral images.

Keywords: Hyperspectral imaging, 3D, convolutional neural network, Bayesian optimization, Keras Tuner, spectrum, channel importance, lettuce, stress detection.

Acknowledgment

I extend my deepest respect and gratitude to Dr. Christopher Henry. His exceptional guidance, consistent encouragement, and keen insights have been foundational in shaping the course of my research. Under his distinguished mentorship, I have achieved significant academic advancement and personal growth.

My profound appreciation goes to Dr. Christopher Bidinosti and Dr. Michael Beck. Their meticulous feedback and astute advisement have played an instrumental role in the successful composition and publication of our collaborative journal article.

Special thanks are due to Dr. Christopher Storie for his willingness to take the time to read and provide valuable comments on my thesis. His perspectives have been greatly appreciated.

I am sincerely thankful to Mitacs and the Enterprise Machine Intelligence & Learning Initiative (EMILI) for their steadfast financial support. Their funding has been invaluable, allowing me to dedicate my full attention and efforts towards achieving research excellence.

I would also like to acknowledge the support of Cara Godee during my thesis experiment. Her assistance has been invaluable to the success of this project. My gratitude also extends to the technician support team, Eric Benson, Connie Arnhold, and Dagmawit Habtemariam for their unwavering support during my scholarly journey.

Finally, my profoundest gratitude is reserved for my family and friends. Their unwavering faith in me and continual emotional support have been the bedrock of this academic endeavor. Their belief in my capabilities has been a constant source of strength and motivation.

This journey would not have been possible without the combined support, encouragement, and guidance of all these individuals. Their collective contributions have been pivotal in bringing this thesis to fruition.

Dedication

In dedication to my family.

Contents

List of Tables

List of Figures

Chapter 1

Introduction[1]

Plant health and productivity are crucial determinants of yield outcomes in agriculture. They serve as cornerstones for food security worldwide. Various stressors can compromise these factors and lead to decreased yield and quality of agricultural products. One of the culprits behind reduced plant health is nutrient deficiency. Such deficiency can weaken plants and make them more susceptible to diseases [2], thereby jeopardizing both the quantity and quality of agricultural produce. All plants, irrespective of the growth medium, require a specific set of nutrients to flourish. In soil-based cultivation, plants source these nutrients from both the natural composition of the soil and added fertilizers. For those venturing into hydroponics or other soil-less cultivation techniques, premixed nutrient solutions often serve as the primary nutrient source. The nutrient needs of plants can be broadly categorized into macronutrients and micronutrients. Macronutrients like nitrogen, potassium, phosphorus, calcium, sulfur, and magnesium are required in larger quantities. These elements form the foundation for various physiological processes that drive plant growth. On the other hand, micronutrients such as boron, copper, iron, manganese, molybdenum, and zinc are needed in trace amounts, but their role is no less significant [3].

The origin of nutrient deficiencies often lies in the inadequate nutrient content of the growing medium or solution. Still, environmental factors can further exacerbate the situation. Unfavorable conditions, such as extreme pH levels [4] (either too acidic or alkaline) or imbalanced moisture (either dryness or waterlogging), can impede the plant's ability to absorb these essential nutrients [5]. If these deficiencies remain unaddressed, they can severely hamper a plant's natural processes, including flowering and fruiting, ultimately leading to compromised yields. Given the profound implications of nutrient

[1]The content of the introduction was adapted from [1], which was published on September 14, 2023, in the *Elsevier's Smart Agricultural Technology* journal.

deficiencies, it is imperative for growers to recognize, understand, and address them timely. This ensures optimal plant health and maximizes agricultural productivity.

To counteract these challenges and ensure the best possible yields, the agricultural sector has experienced a transformative shift with the introduction of advanced sensing technologies [4]. These technologies are revolutionizing traditional farming practices where manual inspections of crops and soil is often reliant on subjective judgment. Traditional methods are inherently inconsistent, labor-intensive, and limited in their ability to detect underlying issues. Furthermore, many crops and their afflictions can look visually similar and it poses challenges for differentiation based solely on external appearance. That is where sensing technologies like hyperspectral imaging (HSI) bridges the gap by enabling producers to not only inspect and sort their crops with unprecedented precision but also monitor the health and productivity of their lands through real-time processing of images [6, 7]. At the core of HSI's utility is its capacity to tap into the spectrum, to capture the intricate range of light wavelengths that objects either reflect or emit. Unlike traditional imaging, which captures light in three broad bands (red, green, and blue), HSI divides the spectrum into numerous narrow bands. By doing so, it detects the spectral signature – an object's unique pattern of light reflection or emission across various wavelengths. This granular approach to capturing spectral signatures provides a deep insight into an object's composition. In the context of agriculture, this means not only identifying a plant but also understanding its health, maturity, and even its water or nutrient content [8].

HSI, also referred to as imaging spectrometry, combines two distinct technologies, imaging and spectroscopy, to provide spatial and spectral information, simultaneously. The imaging technique is used for capturing spatial data information from objects is realized using a digital camera whereas spectroscopy provides intensity values for a continuous spectrum of wavelengths. A hyperspectral camera combines both concepts in the same system. Spectral information can provide rich information about biochemical and biophysical attributes of the agricultural crops. This is due to the higher spectral resolution of hyperspectral sensors compared to multispectral [9] and RGB (red, green, and blue) ones. Therefore, this feature leads to better discrimination of objects of similar colors, higher accuracy in complex classifications, ability to predict chemical composition, and provide information about the interior of an object [8].

However, the interpretation of spectral data can be complex, especially when analyzing and comparing multiple samples over extended periods. One approach to simplify spectral analysis is the usage of spectral indices. Spectral indices are mathematical expressions that combine several spectral bands into a single value, providing an easier representation of the data. For example, the normalized

difference vegetation index (NDVI) computes the ratio between near-infrared (NIR) and red (R) bands of hyperspectral channels as follows:

$$\mathrm{NDVI} = \frac{\mathrm{NIR} - \mathrm{R}}{\mathrm{NIR} + \mathrm{R}}. \tag{1.1}$$

With the help of spectral indices, we can effectively identify trends and changes in the data without requiring an in-depth comprehension of the underlying scientific principles governing spectral data, thus enabling a simpler data analysis, enhancement of features, standardization, comparability, and calibration of data.

In the agricultural industry two common spectral indices are the already mentioned NDVI and the green chlorophyll index (GCI). The former is used to monitor vegetation growth and health, while the latter quantifies the amount of chlorophyll in plants. Further indices have been defined, usually in the context of remote sensing, to support research for example on agriculture, soil, vegetation, water and forestry. A comprehensive database of spectral indices that is searchable by application area and hyperspectral sensor is provided in [10]. Nevertheless, the applicability of this approach hinges on the availability and development of a pertinent spectral index aligned with the intended application.

From a data-perspective, a hyperspectral image is a stack of images, known as hyperspectral cube or data cube. Each image of this cube represents the response of the imager to one of the distinct hyperspectral channels [11]. This is illustrated in Figure 1.1. It shows a 3D data cube P with dimensions $M \times N \times \lambda$, where M and N represent the axes of spatial information and λ represents the spectral dimension [12]. In the hyperspectral cube, each pixel, given by its spatial coordinates, is a vector of length λ that indicates the reflected radiation of a specific part of the object.

This is where the potency of artificial intelligence (AI) comes into play. Leveraging AI offers a transformative approach to analyzing HSI data. AI allows machines to emulate human cognitive functions such as problem-solving, pattern recognition, and decision-making. Machine learning, a subset of AI, is a research area devoted to developing algorithms that can extract valuable insights from data without explicitly identifying these patterns a priori. These insights empower the algorithms to continually adapt and make informed predictions or classifications. Advances in both hardware and software components of machine vision systems have enabled machine learning algorithms to process data more rapidly, delivering dependable outcomes in a shorter time frame. This approach has been widely used in assessing the quality of agricultural products [13]. A plethora of machine learning methodologies, including but not limited to decision trees [14], Naïve Bayes [15], k-means clustering [16], support vector machines [17], random forests [18], and k-Nearest neighbors [19] have been prolifically applied across various agricultural domains [13]. However, the performance of these traditional machine learning algorithms heavily depends on manual feature engineering and this lack of adaptability to automatically

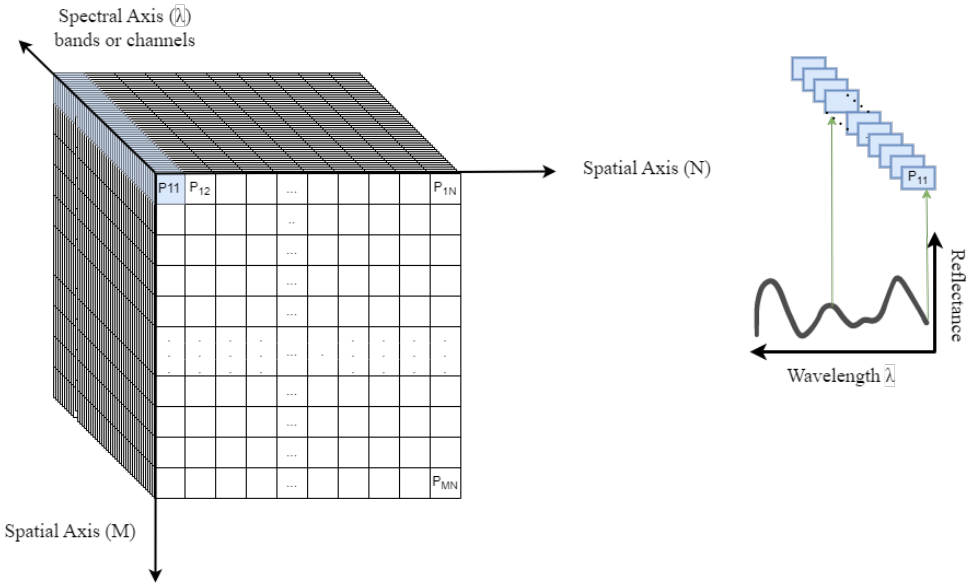


Figure 1.1: The hyperspectral cube (adapted from [12] with modification). It is a three-dimensional array where each pixel represents a spectrum containing a range of wavelengths. This spectrum can act as a fingerprint and provides information about biophysical and biochemical characteristics of the imaged object.

extract data features reduces their efficacy. In contrast, convolutional neural networks (CNNs) as a deep learning approach [20], which itself is a branch of AI, have the inherent capability to automate feature extraction. That means they can learn and identify important features from data without manual intervention. This ability is a significant leap from traditional machine learning techniques. Moreover, deep learning approaches have been shown to achieve higher classification accuracies in image classification tasks and are particularly suited to handle the high-dimensionality of HSI data. While CNNs have achieved commendable success in handling 2D data, their adaptation to 3D data within the agricultural realm remains widely unexplored. Pioneering into this underexplored avenue, this thesis aims to delve deep into the prospects of 3D-CNNs in pinpointing plant stress through the lens of HSI technology.

1.1 Problem Statement

Ensuring optimal growth of plants requires a meticulous balance of mineral nutrients. Among these, nitrogen stands out as a pivotal element, significantly enhancing both the volume and quality of crop yields. From a biological perspective, nitrogen doesn't merely influence the overt growth patterns of plants; it profoundly impacts their intrinsic metabolic processes as well [21]. Nitrogen deficiency can hinder plant growth and lead to reduced vitality. As the deficiency progresses, older leaves may

turn yellow, a phenomenon termed chlorosis, which can further develop into necrosis, characterized by browning leaf edges. In advanced stages, plants may lose leaves altogether, coupled with increased stiffness in both roots and leaves [22]. These symptoms can visually change the appearance of the plant at later stages of growth. However, recognizing nitrogen deficiency during the early growth phases is challenging as the plant's leaf almost does not show apparent changes which are noticeable by human eyes. Hence, making timely interventions is crucial as it can prevent huge crop losses in large-scale agricultural settings.

In our study, we used lettuce as a case study. However, the techniques and methods we developed are applicable over a broad range of plants, including staple crops that play an important role in food security. To illustrate this, Figure 1.2 displays three samples of Buttercrunch lettuce, each subjected to different nitrogen stress levels from July 15, 2022, to August 14, 2022. These samples were planted on July 5, 2022, in a growth chamber at The University of Winnipeg under technician supervision. However, each sample grown with distinct nitrogen fertilizer solutions: containing no ammonium nitrate (N^-), half the standard dose of ammonium nitrate ($N^{+1/2}$), and the full standard dose of ammonium nitrate (N^+). At first glance, as of July 18, distinguishing between the three nitrogen groups (N^-, $N^{+1/2}$, and N^+) presents a challenge. Differentiating $N^{+1/2}$ from N^+ becomes even more intricate by July 21.

Historically, researchers have used tissue analysis [23] as a valuable instrument for informed fertilizer management. This technique assesses nitrogen content and discerns the nutrient requirements of plants. Such assessments were contingent on invasive methodologies, involving the exhaustive sampling of plant parts such as leaves, branches, and even requiring the uprooting of entire plants in certain scenarios. This approach was not only cumbersome and labor-intensive but also carried significant environmental and safety concerns due to the use of hazardous chemicals like those involved in the Kjeldahl method. Furthermore, employing elemental analyzers incurred considerable expense. However, recent advancements in spectroscopic methodologies [24] have heralded a paradigm shift in this arena. Modern techniques, leveraging the capabilities of HSI in tandem with visible and/or NIR spectroscopy, present a swift, non-invasive alternative. This innovation has been pivotal, particularly for the quantitative assessment of variables such as soil N, plant N, and fertilizer nitrogen concentrations, ushering in a more streamlined and environmentally considerate era of agricultural research. Despite the emergence of shifting towards utilizing HSI in this domain, researchers have not yet significantly used and assessed deep learning approaches to assess plants' nutrients. This study delves into the potential of HSI across a wide spectral range (400–1000 nm), paired with 3D-CNNs, to detect nitrogen deficiency challenge in lettuce.

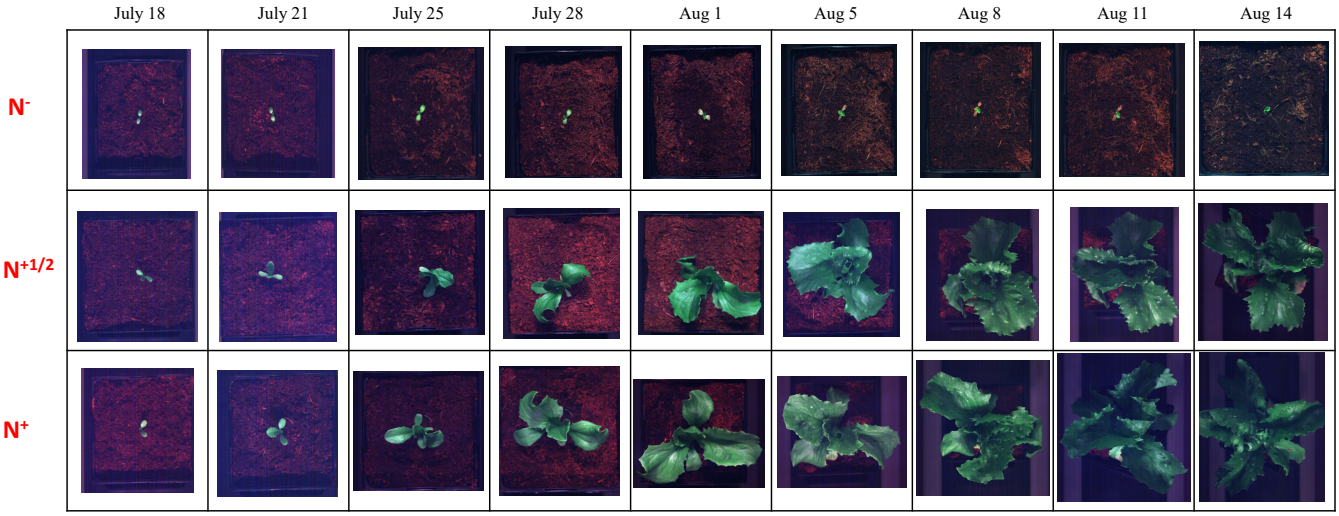


Figure 1.2: Growth stages of three lettuce samples subjected to nitrogen stress experiment. The rows, in sequence, depict distinct nutrient conditions: nitrogen-free (N^-), half-nitrogen ($N^{+1/2}$), and full-nitrogen (N^+).

1.2 Proposed Approach

To address the challenge of early detection of nitrogen deficiencies in Buttercrunch lettuce, our research employed a targeted approach, integrating HSI with 3D-CNN design and analysis. This research leverages the capabilities of 3D-CNNs to interpret complex data structures, enabling a more nuanced understanding of the spectral and spatial characteristics inherent in HSI data. Our research pivots around three main objectives:

- To address a gap in the current literature, we curated a hyperspectral dataset that meticulously captures the development of non-hydroponically planted Buttercrunch lettuce under varying nitrogen stress conditions throughout its growth phase.
- Early growth stage differentiation: Our objective is to explore the potential of 3D-CNNs in identifying early-stage nitrogen-stressed lettuce by classifying the lettuces into three distinct groups (N^-, $N^{+1/2}$, and N^+). Our goal is to introduce automation into the complex classification challenge.
- Spectral band identification: Tapping into the wealth of data provided by HSI, our goal is to discern which spectral bands, or combinations thereof, are crucial for classifying nitrogen-stressed lettuce.

Considering the absence of datasets for non-hydroponic, nitrogen-stressed lettuce, we curated a unique

collection of hyperspectral images. This experiment unfolded in the growth chambers of The University of Winnipeg. Supervised by a technician, we cultivated 15 Buttercrunch lettuce samples for each nitrogen-stressed category, namely N^-, $N^{+1/2}$, and N^+. Given our keen interest in early, subtle signs of N deficiency, our analysis was circumscribed to observations on July 18 and 21. At these initial growth stages, discerning visual differences is notoriously tricky. Imaging was executed using the SPECIM FX10 hyperspectral camera in a darkened room, followed by rigorous preprocessing to calibrate the captured data. We utilized the Bayesian optimization technique to strategically search for the optimal 3D-CNN model configuration tailored for classifying the nitrogen stress status (N^-, $N^{+1/2}$, and N^+) in lettuce. From a high-level perspective, Bayesian optimization involves a highly intelligent trial-and-error process to find the optimal model configuration, where each "trial" is chosen very thoughtfully to learn as much as possible, thus reducing the number of trials needed to find the best solution. Additionally, we aimed to pinpoint specific spectral bands pivotal for classification, potentially paving the way for cost-efficient multispectral cameras tailored to these critical bands.

1.3 Thesis Contribution

A comprehensive review was performed to bridge the existing knowledge gap by:

- Undertaking a review of 3D-CNN-based architectures tailored for classification of hyperspectral images of diseased and defective crops.
- Elaborating on the intricate deep learning pipeline, encompassing data preprocessing, band and feature selection, and visualization of 3D-CNN model's outcomes for the classification of hyperspectral images of diseased and defective crops.
- Discussions on untouched research avenues, intricacies, and the prospective challenges posed when integrating 3D-CNNs with HSI data, thereby setting the stage for future explorations in the domain.

The survey results were published in the following article:

"A comprehensive review of 3D convolutional neural network-based classification techniques of diseased and defective crops using non-UAV-based hyperspectral images." Smart Agricultural Technology 5 (published on September 14, 2023) [1]

The main contributions of this thesis are as follows:

- A novel labelled dataset of hyperspectral images of nitrogen-stressed Buttercrunch lettuce consisting of three classes: a) nitrogen-free (N^-): These plants did not receive any ammonium nitrate, b) Half-nitrogen ($N^{+1/2}$): These plants received 50% of predetermined standard amount of ammonium nitrate, and c) Full-nitrogen (N^+): These plants received 100% of the predetermined standard amount of ammonium nitrate. In total the dataset contains 404 hyperspectral images consisting of 135, 134, and 135 hyperspectral images per N^-, $N^{+1/2}$, and N^+ class, respectively.

- Generated 350 3D-CNN models utilizing Bayesian optimization technique, with the top-performing model achieving 75.00% accuracy on test dataset, which focused on the early stages of nitrogen stress treatment in lettuce observed on July 18 and July 21.

- Determined the significance of individual and collective spectral bands – spanning violet, blue, green, yellow, orange, red, and NIR within the 400 – 1000 nm wavelength range – in prediction of target class within the test dataset.

- Addressing the challenge of valid model generation by the Keras Tuner library through meticulous configuration of hyperparameters specific to our 3D-CNN model. These hyperparameters are crucial as they dictate the learning process of the model.

1.4 Thesis Layout

Chapter 2 emphasizes the critical role of the micro-ingredient "N" in optimizing crop yields. This chapter presents a comprehensive review of both conventional methods and cutting-edge research techniques for detecting and determining the presence of "N" in lettuce. Transitioning to Chapter 3, we demonstrate the deep learning pipeline for HSI data classification using 3D-CNN models. In this regard, we delve deep into architecture of CNNs and elaborate on its main building blocks, training and performance evaluation. Then, we provide a comprehensive review of 3D-CNNs-based models utilized in classifying non-UAV-based hyperspectral images of crops with diseases and defects. Also, complete HSI data preprocessing steps, band and feature selection, and visualization of HSI classification decisions are provided. We culminate the chapter by shedding light on the existing research gaps and constraints involved in employing 3D-CNN for the categorization of HSI data. Chapter 4 provides a detailed overview of the material and methodologies involved in creating the HSI dataset for nitrogen-stressed lettuce. It outlines the preprocessing pipeline for hyperspectral data and illustrates the process of designing 3D-CNN models using Bayesian optimization, along with the resultant findings. This chapter

also explores the channel importance in classification result through an ablation study. At last, Chapter 5 concludes the research and proposes directions for future works.

Chapter 2

Background and Related Works

2.1 Significance of Nitrogen in Plant Growth

Nitrogen is a crucial element for plants, vital for their growth, development, and reproduction. Although it is abundant in the environment, plants often face nitrogen deficiency due to its indirect availability. Comprising 3-4% of a healthy plant's above-ground part, nitrogen is higher in concentration than most nutrients. It is foundational in amino acids, which make up proteins, and is vital for plant tissues, cell membranes, and chlorophyll production. Nitrogen is also part of the DNA, influencing genetic traits in crops. It enhances various physiological processes, like giving plants a darker green hue, boosting root growth, and stimulating crop yield. Nitrogen elevates photosynthesis rates, seed productivity, and quality. Furthermore, in horticultural plants, it increases fruit yield and pulp content. It not only promotes the absorption of other essential nutrients but also bolsters plants' resilience against environmental stresses. However, factors like soil moisture, aeration, and salt content can impact N's efficiency in plants [25]. Considering the numerous functions nitrogen undertakes and its potential to optimize plant health and productivity, this fact drives us to inspect available methods to recognize and address nitrogen deficiency.

2.2 Overview of Techniques to Monitor and Detect Nitrogen Deficiency in Lettuce

In the historical context of agricultural research, the determination of nitrogen in plants was traditionally achieved through techniques such as Kjeldahl [26] digestion and Dumas combustion [27]. While effective, these methodologies were notably destructive and time-intensive [28]. With the surge of

technological advancements, a transition from these conventional methods was observed, and the agricultural community began embracing optical methods. These modern techniques encompass the use of tools such as spectroradiometers [29], reflectometers [30], satellite sensor imagery [31, 32], and digital cameras [33, 34, 35, 36]. They are primed to measure a myriad of properties ranging from crop canopy reflectance and leaf transmittance to the nuances of chlorophyll fluorescence, thereby estimating the N content in plants [37]. In the following, we delve into application of these advanced techniques.

Emerging from this technological shift were two prominent types of sensors. Passive sensors, like the FieldSpec spectroradiometer [38], operate by detecting the naturally reflected or emitted radiation from the plants. These sensors do not emit any radiation of their own. On the other hand, active sensors, exemplified by GreenSeeker and Crop Circle [39, 40], emit their own radiation or light source and then measure the reflected radiation from the plant surface. Empirical research showed a compelling correlation between the optical parameters gauged by these sensors and the nitrogen status in plants. This opened doors to non-invasive and, consequently, non-destructive testing alternatives [41]. Nevertheless, while these strides forward presented new avenues, they were not without challenges. Issues such as chlorophyll saturation, the capricious nature of atmospheric interference, and the high instrumentational costs proved to be significant hurdles [42].

In response to these challenges, and ever pushing the boundaries of innovation, recent trends have shifted focus towards harnessing the power of digital imaging. By leveraging commercial cameras combined with intricate image processing systems, researchers developed a cost-effective paradigm to discern crop N status. This method involved a detailed analysis of various health parameters intrinsic to plants [38, 43, 44, 45]. This epoch of research accentuated the immense potential of HSI to unearth deeper insights into plant health and the multifaceted conditions governing growth.

HSI, at its core, leverages the seemingly simple yet intricate concept of capturing how plant leaves differentially absorb and reflect light across a spectrum of wavelengths. This phenomenon results in what can be described as a spectral fingerprint, a unique signature that embodies specific wavebands. This signature is not static; it evolves, dictated by a variety of factors including the overall health of the plant, its hydration levels, and its nutrient uptake. Diving into effects of varying nitrogen concentrations on the growth of different lettuce cultivars using hydroponic tubs and the Nutrient Film Technique system, a study made use of HSI's capabilities to chronicle these spectral shifts [46]. The research was further enhanced by the inclusion of sophisticated regression models, namely the first derivative reflectance, partial least squares regression/principal component analysis (PLSR [47, 48]/PCA [49]), and variable importance in projection scores. These models were not just auxiliary tools; they deepened

the insight, enabling precise estimations and analytical interpretations of the nutrient dynamics within the lettuce cultivars. The HSI data from the 400-575 nm wavebands accurately matched laboratory measurements, offering a non-invasive method to understand plant nutrient interactions.

Moreover, in another study Reference [50] by these researchers, four lettuce cultivars (Rex, Flandria, Flandria, and Black Seeded Simpson) were grown hydroponically under varying nitrogen concentrations. The study identified significant spectral bandwidths within range 400–1000 nm using a SPECIM FX10 camera to determine the nutrient content of the different cultivars of lettuce. By employing PLSR and PCA techniques, the spectral ranges 506.33–601.11 nm and 634.76–701.13 nm shown to be significant in non-invasively estimating nutrient levels in lettuce.

Building on the potential of HSI, Reference [51] took a more focused approach towards understanding nitrogen dynamics in crops. Lettuces were grown in three distinct soilless nitrogen solutions. Upon reaching the rosette stage, 84 lettuce leaves from each nitrogen level were analyzed using HSI. The data, collected from specific regions of these leaves, was processed using standard normal variate correction and further refined using PCA. The analysis proceeded with the application of the extreme learning machine (ELM) algorithm [52] which showed notable efficiency in speed and accuracy. The ELM model achieved a 100% accuracy rate which highlights the benefits of integrating conventional methods with modern technology in agriculture. This dovetails with the earlier study, reiterating the transformative role of HSI in the agricultural landscape.

In another experiment on Lactuca sativa lettuce [53], researchers sought to optimize precision fertigation– the targeted application of water and nutrients. To achieve this, they employed a combination of the Plantarray system, an advanced phenotyping platform that monitors and analyzes plant performance in real-time, and HSI. Their goal was to identify early signs of nitrogen and water deficiencies in plants before these deficiencies could adversely impact growth. From the hyperspectral data, three crucial vegetation indices were derived: the red-edge chlorophyll index (RECI) [54], the photochemical reflectance index (PRI) [55], and the water index (WI) [56]. Each of these indices illuminated specific facets of plant health. For instance, RECI was particularly effective in pinpointing nitrogen deficiencies, while the WI was adept at identifying instances of water stress. Meanwhile, PRI provided real-time insights into the plant's transpiration rates. Through this comprehensive approach, combining both the analytical capabilities of the Plantarray system and the detailed insights from HSI, the researchers were better positioned to enhance crop yield management. By identifying plant needs early on, interventions could be made proactively, ensuring healthier growth and optimized yields.

Diving deeper into the spectral analysis of lettuce, Reference [57] integrated spectral analysis techniques

with chemometrics to estimate the nutrient content (N, phosphorous, and potassium) of aquaponically grown lettuce. By employing methods such as PCA, genetic algorithms, and sequential forward selection, critical wavelengths or spectral bands sensitive to these nutrients were identified. Notably, in the visible spectrum, specific bands like blue (475 nm), green (530 nm), red (668 nm), and the red edge (717 nm) showed significant variance based on nutrient concentration. Higher nutrient levels led to increased chlorophyll and photosynthesis activity, which influenced the absorption and reflection properties at these bands. In the NIR range (720 nm to 1300 nm), a positive correlation was observed, with higher nutrient content resulting in increased reflection. These selected spectral bands were then used as inputs for models like PLSR, back-propagation neural network (BPNN) [58], and random forest (RF) [59]. Among these, BPNN and RF demonstrated particularly high predictive accuracies, suggesting that specific spectral bands combined with machine learning can provide an effective non-invasive tool for monitoring nutrient content in plants.

While the advancements in using HSI, coupled with traditional machine learning techniques, have marked significant progress in detecting nitrogen deficiencies in lettuce and other crops, the next frontier in this research is deeply entrenched in leveraging cutting-edge computational models. Recent studies have shown the potential of employing deep learning techniques. Amidst them CNNs (refer to Chapter 3) excel in processing patterns within images, and over time, they have emerged as the dominant force in tasks like image classification, object detection, and segmentation. However, when it comes to the volumetric data of HSI, 3D-CNNs (discussed in Chapter 3) offer the capability to capture spatial and spectral patterns in data, making them particularly suited to analyze hyperspectral data cubes. While the mentioned works have laid the foundation in using HSI for plant health diagnostics, integrating 3D-CNNs may offer a more holistic and nuanced understanding of the spectral data. The aim is not just to build upon the accuracy and efficiency achieved by earlier models but also to tap into the deeper, perhaps latent, patterns in the hyperspectral data that can offer more granular insights into plant health and the intricate dynamics of nutrient interactions. In doing so, this research seeks to contribute to the ongoing efforts in precision agriculture, possibly offering a fresh perspective on diagnostic and detection tools.

Chapter 3

Deep Learning Pipeline for HSI data Classification using 3D Convolutional Neural Networks [1]

A deep learning pipeline for HSI classification is a multi-stage process that transforms raw data into a predictive model capable of making informed decisions or predictions. Such a pipeline typically consists of several stages, including data preprocessing, band and feature selection, model design, model training, testing, and evaluation (see Figure 3.1). Before we delve into the intricacies of each stage, we first provide a succinct overview of these components to establish a foundational understanding that will support the detailed discussion to follow.

Data preprocessing is the initial step where raw HSI data undergoes various enhancements to improve quality. Techniques such as noise removal and radiometric calibration are employed to ensure the reliability of the data. Noise removal helps in clearing out any irrelevant or extraneous information from the images, while radiometric calibration transforms the camera's abstract digital outputs into precise measurements of true radiation intensity or reflectance across each pixel and spectral channel. Dimension reduction techniques are also applied to distill the vast amount of data into its most informative aspects. The subsequent stage of feature extraction transforms preprocessed data into a new set of features designed to be more discriminative for the task at hand. Concurrently, band selection strategies are employed to identify and retain the most relevant spectral bands for classification purposes.

The model design stage involves carefully planning and structuring a series of layers to process and interpret image data efficiently and accurately. This includes deciding on the number and types of layers,

[1] The contents of Sections 3.1.2, 3.1.6, and 3.2 to 3.6 were adapted from [1], which was published on September 14, 2023, in the *Elsevier's Smart Agricultural Technology* journal.

their arrangement, and how they interact to extract meaningful patterns and features from the images. The goal is to create a model that can effectively learn from data and make accurate predictions or classifications based on visual inputs. Training the model is an iterative process of fine-tuning internal parameters to minimize classification errors. Upon completion of training, the model undergoes testing and evaluation to validate its predictive performance on unseen data, ensuring it has successfully learned the underlying patterns and relationships within the HSI dataset. In this respect, in Section 3.1, we delve deep into the structure of CNNs, particularly 3D-CNN as the focus of this research study, and their most important concepts. Following the typical data pipeline associated with CNNs, Sections 3.2 to 3.5 review network architecture design, data preprocessing, band and feature selection, and visualization of HSI classification decisions. This provides a convenient overview of its individual steps for plant classification problems using 3D-CNNs. Finally, Section 3.6 highlights the research gaps and limitations associated with the application of 3D-CNNs for HSI data classification.

3.1 Insights into Convolutional Neural Network Architecture

3.1.1 Artificial Neural Networks (ANNs)

Computer vision algorithms serve as interpretation tools. Their purpose is to analyze the input image received from the sensing device, extract features and patterns to recognize objects. To accomplish this, researchers drew inspiration from the workings of the human brain to guide and inform their research and algorithm development. This endeavor led to the emergence of artificial neural networks (ANNs). ANNs are composed of layers of neurons that perform calculations and generate predictions based on inputs. These networks consist of an input layer, zero or more hidden layers, and an output layer. The number of layers in the network characterize the depth of the network [60].

The term *deep neural network* traditionally referred to networks with more than one hidden layer. However, it is more commonly used now to describe networks with a considerable number of layers. Conversely, a *shallow network* denotes a network with relatively few layers. It is important to note that terms like *many* and *few* are subjective and not precisely defined. Additionally, there is no universally agreed-upon threshold for the depth that differentiates shallow networks from deep networks. Nonetheless, most researchers agree that a network is typically considered *deep* if its depth is greater than 2 [61].

Figure 3.2 shows a type of ANN which is called multi-layer perceptron (MLP). It consists of the input layer, three hidden or fully connected layers, and the output layer. Each node represents an artificial

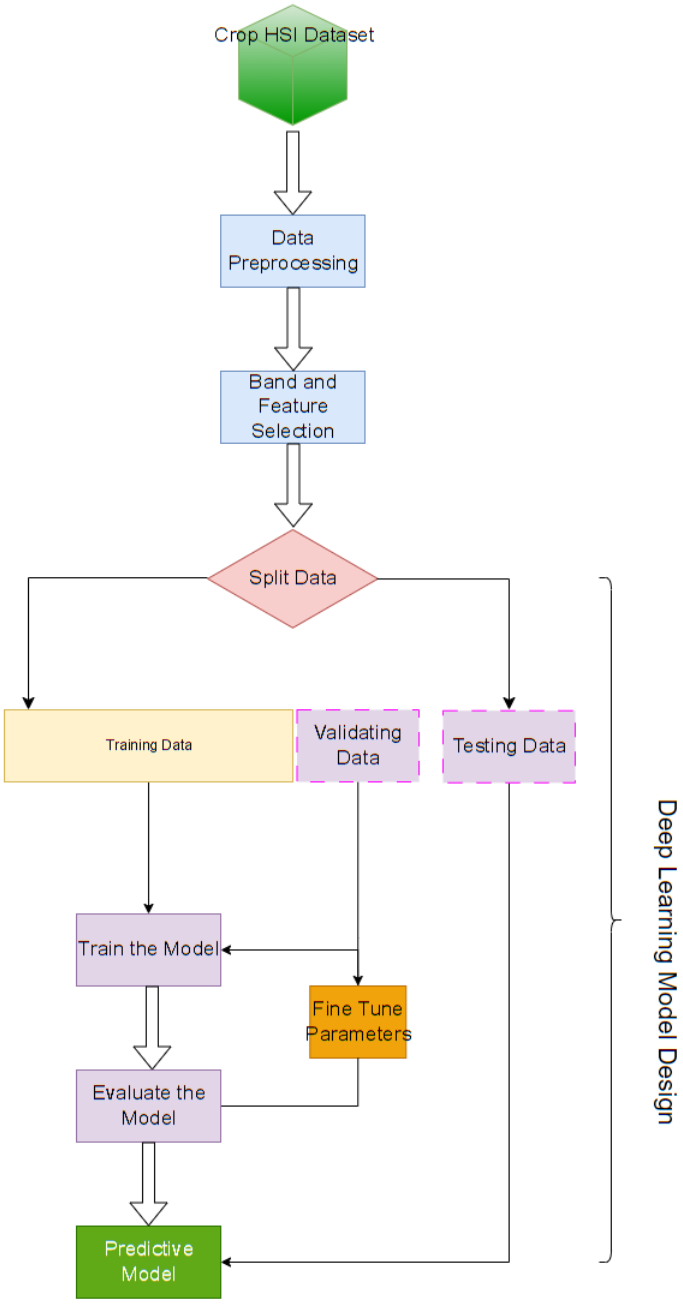


Figure 3.1: Typical deep learning pipeline for HSI data classification. First, the discriminative features and bands are extracted from preprocessed HSI dataset of crop. Then, the dataset is split to training, validation, and testing sets. The training dataset is used to train the 3D-CNN model, the validation dataset is employed to assess the model's performance and fine tune its parameters, and the testing dataset serves to evaluate the final performance and generalization ability of the trained 3D-CNN model on unseen data.

neuron which is explained in Section 3.1.1.1. Hidden layers, positioned between the input and output layers, consist of stacked neurons where each neuron connects to every neuron in the subsequent layer.

These layers are referred to as *hidden* because we have limited visibility and control over the information flowing through them. The input is fed into the input layer, and the resulting output is observed at the output layer. Each edge represents a connection between nodes in different layers. It is assigned a weight, denoting the significance of its influence on the final output prediction. The output layer is where we obtain the model's answer or prediction. The nature of the output layer depends on the specific setup of the neural network. For regression problems, which involve predicting a continuous output value, the final output may be a real-valued number. On the other hand, for classification problems, where the aim is to predict the category or label of an input, the output typically consists of a set of probabilities, with each probability corresponding to a different class or category [62].

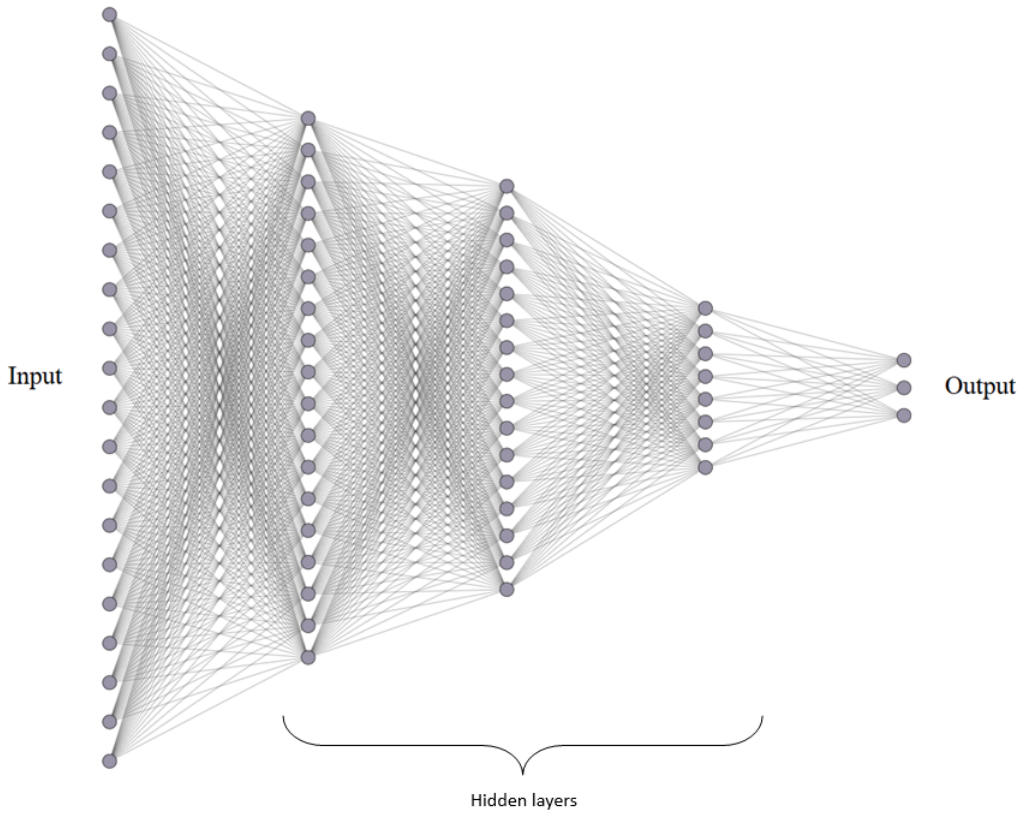


Figure 3.2: An artificial neural network comprised of neurons connected with edges.

3.1.1.1 Artificial Neuron

The artificial neuron or perceptron is the fundamental component of a neural network. Figure 3.3 illustrates the components and functions of the perceptron in detail. ANNs are limited to accepting

1D vectors with dimensions $(1, n)$ as input. The result is that images must be transformed to a single 1D vector when using ANNs. This process, known as image flattening, results in the loss of spatial and spatiospectral features in 2D and 3D images, respectively. Therefore, the input vector, denoted by N, represents the feature vector containing inputs $(n_1, n_2, n_3, ..., n_m)$. Each input is assigned a weight value $(w_1, w_2, w_3, ..., w_m)$ to indicate its relative importance in distinguishing between different input data points. Within the neuron, two calculations take place: 1) the weighted sum, where the inputs are multiplied by their corresponding weights, summed together and a bias (b as shown in Figure 3.3) is added, and 2) the activation function [62]. The weighted sum function, also known as *linear combination*, produces a straight line which passes through the origin in coordinate system. The activation function takes the weighted sum of inputs as input and produces the output of the neuron. By introducing nonlinear characteristics, activation functions enable the network to extract and identify intricate relationships between input and output variables, facilitating the recognition of complex patterns and associations. The bias is an additional parameter that adjusts the output along with the weighted sum of the input to the neuron. It allows the activation function to better map the input to the output. This ability of an ANN to approximate any function has resulted in it being called an *universal function approximator*. A comprehensive review of activation functions in deep learning is provided in [63]. In practice, the rectified linear unit (ReLU), Leaky ReLU and Parametric ReLU activation functions [64] are the most common ones used by researchers due to their simplicity and performance. For example, ReLU leads to a tradeoff between the accuracy (see Section 3.1.5) and training (see Section 3.1.3) time. As shown in Figure 3.2, the flow of information starts from the input layer, passes through the hidden layers, and ultimately reaches the output layer. This progression occurs by applying two consecutive functions in each neuron: the weighted sum and the activation function. Such process is called feedforward. In essence, the forward pass entails performing calculations across the layers to generate a prediction.

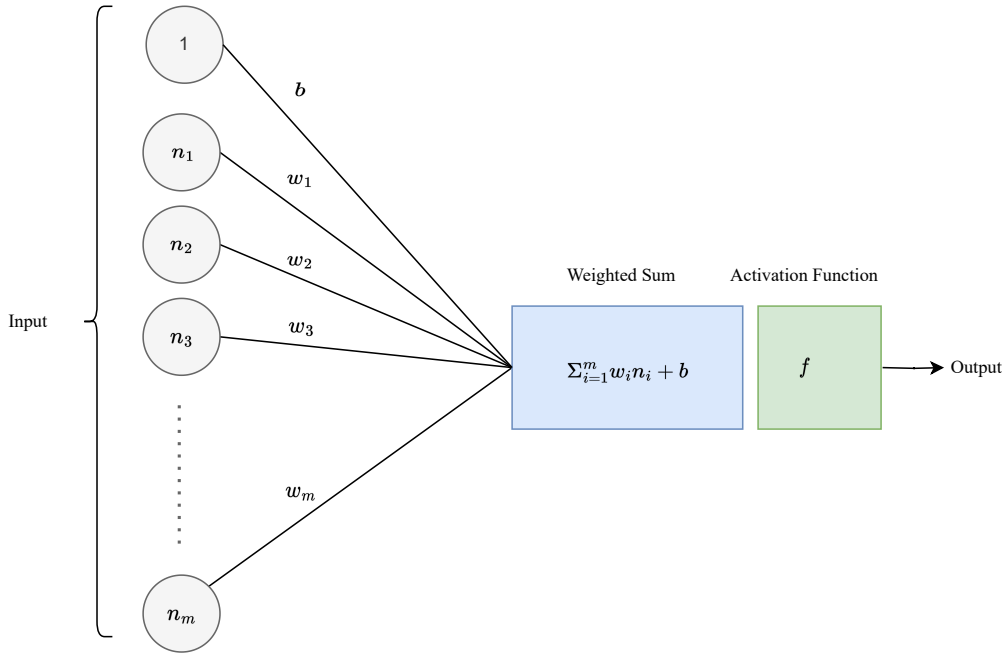


Figure 3.3: Perceptron diagram. Input vectors are passed to the neuron, with weights assigned to represent importance. Weighted sum and activation functions are performed to generate the output.

3.1.2 Convolutional Neural Network (CNN)

A CNN is a type of ANN which is comprised of a series of layers each consisting of several neurons. As shown in Figure 3.4, each layer is the input for the next layer in the network. Key building blocks of a CNN are convolution layers, detector layers [65], and pooling layers, which are explained in detail in Sections 3.1.2.1, 3.1.2.2, and 3.1.2.3, respectively. For a brief overview, a convolution layer uses convolutional kernels (a.k.a. a filter or kernel) to extract features from the 2D input data while preserving the spatial relationship within the 2D input data. However, when processing 3D input data, it can also capture crucial spectral features, whereas, ANNs transform the input data into 1D vectors regardless of its original structure. These convolutional kernels are a set of learnable weights arranged in matrix or tensor format based on the input data's shape. We will delve deeper into the convolution layer in subsequent sections. A detector layer [66] applies a non-linear function like ReLU to learn non-linear representations. Pooling layers make features invariant by reducing the dimensionality of data. Finally, we will also discuss the fully connected layer. Positioned towards the end of the network, these layers transform the extracted features by the convolutional layers into a form suitable for classification or regression. In these layers, neurons connect to every neuron from the previous layer, encapsulating the global information from the feature maps. This will be expanded upon in Section 3.1.2.4.

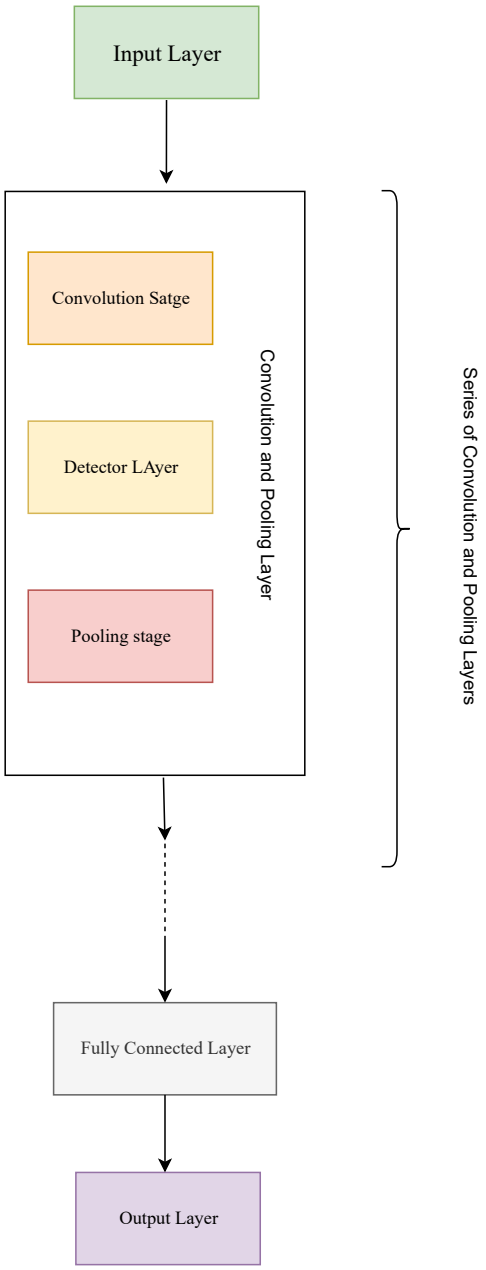


Figure 3.4: A basic conceptual CNN architecture. A CNN consists of multiple layers, including convolution, detector, and pooling layers, where each layer serves as the input for the subsequent layer, enabling the extraction of features, learning of non-linear representations, and dimensionality reduction in the network.

3.1.2.1 Convolutional Layer

The convolution layer is a cornerstone of a CNN, playing a key role in extracting and learning features from input data. Central to this layer is the concept of convolutional filters. These filters traverse the input systematically, performing specific mathematical or convolution operations to detect salient patterns, whether they be edges, textures, or more abstract shapes. To demystify the convolution process, consider two interacting functions. The first function represents our input data, while the second embodies the convolutional filter. Their interaction, through convolution, yields a transformed representation often termed as a feature or activation map.

Delving deeper, the convolutional layer's workings can be visualized effectively on 2D input data, as depicted in Figure 3.5. Here, the convolutional filter scans the input in a sliding window fashion. Each specific region of the input it covers, termed the "receptive field", undergoes an element-wise multiplication with the filter. The aggregated results from these multiplications determine individual values in the resultant feature map. It is crucial to note that in the realm of CNNs, when dealing with 2D inputs, both neurons and weights align in a matrix structure. Furthermore, the depth or complexity of the succeeding layers in the network is contingent on the number of filters employed within each convolutional layer. Each unique filter crafts its own feature map, encapsulating different aspects or features of the input [62].

Moving from 2D to a more complex dimensionality, Figure 3.6 shows 3D convolution using 3D filter over 3D input (such as a hypercube). A 3D filter can be considered as a stack of 2D filters which forms a cubic shape. To compute 3D convolution, the corresponded receptive fields (for the purpose of the explanation consider the ones in blue) 1 to 3, are convolved with 3D filter, 1 to 3, that means all 27 values from input data are multiplied by 27 associated values from 3D filter and after adding them all up, a single value generated (shown in blue in output). By sliding the 3D filter in spatial and spectral dimensions, all values of feature maps can be calculated.

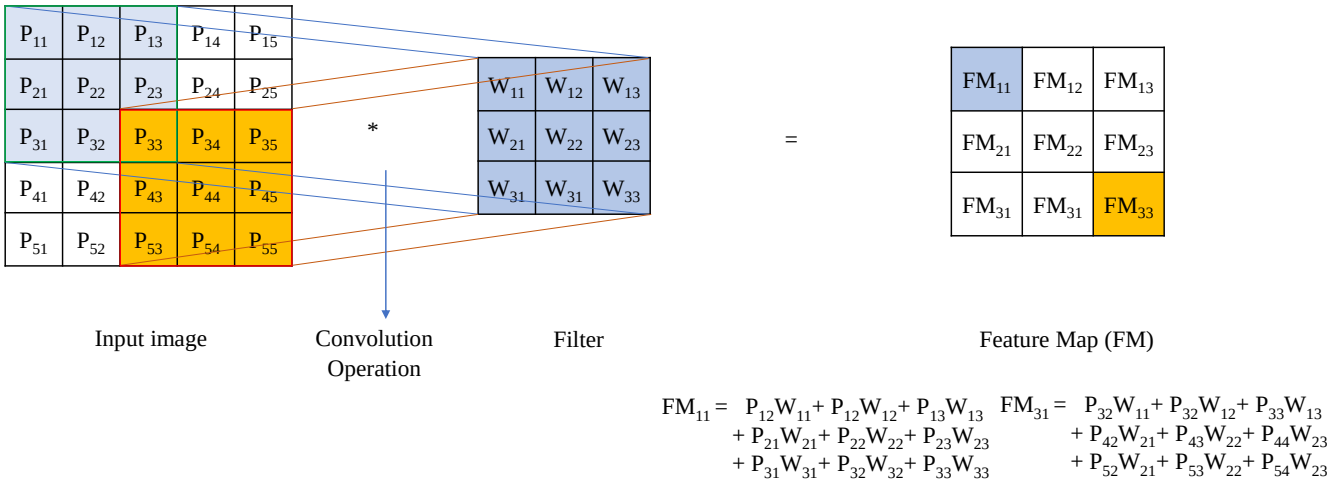


Figure 3.5: Convolution operation. The image shows sliding convolutional filter over an input image to generate a feature map.

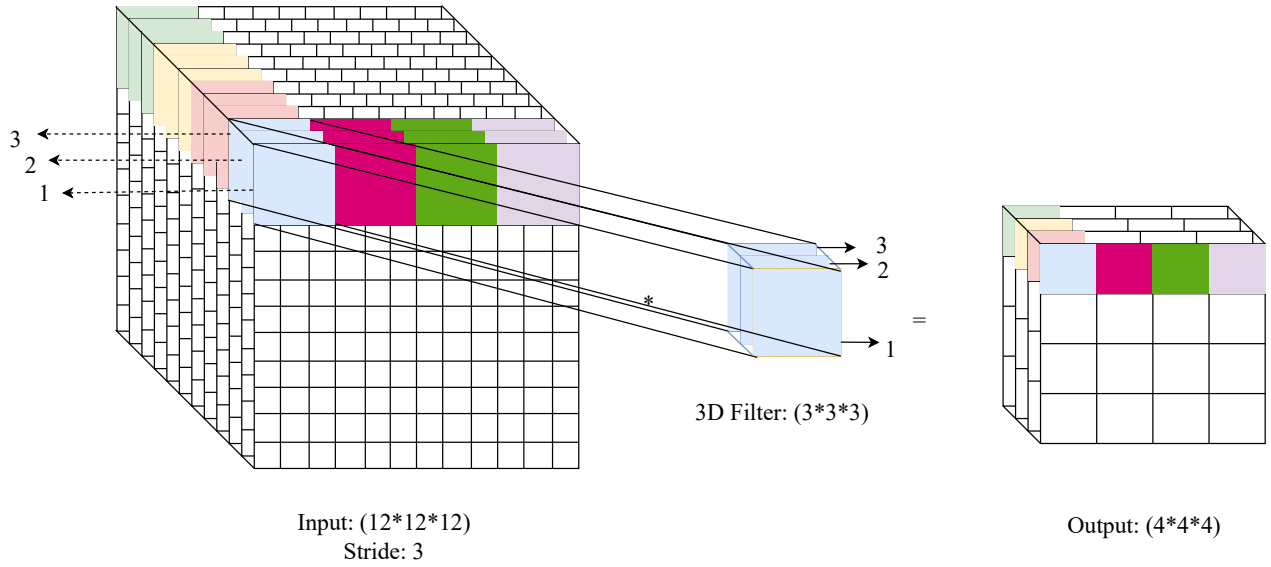


Figure 3.6: Convolution operation using 3D filter over 3D input (such as a hypercube).

In order to enhance the number of neurons within these hidden layers, one can increase the number of kernels utilized in the convolutional layers. Each convolutional filter serves as an individual neuron. For instance, in Figure 3.5 a 3×3 filter is used in a convolutional layer, it implies that there are 9 neurons within that layer. By adding two more 3×3 filter, there would be 3 filters which increases the total number of neurons in the convolutional layer to 27. Therefore, by augmenting the number of filters in a convolutional layer, one effectively increases the number of neurons, which enables the network to recognize more intricate patterns. In general, filters in CNNs have a 2D and 3D shape for 2D and 3D convolution, respectively. It is generally not recommended to use large filter size since it can potentially lead to the loss of essential input data details and are computationally heavy [62].

In order to control the size of the feature map, three parameters need to be determined [62]. The *first* parameter is the depth, which corresponds to the number of filters used for the convolution operation. The *second* parameter is the stride, which determines the number of pixels or voxels the filter moves over the input. For example, stride $(1, 1)$ means the filters move one cell at a time in 2D convolution over 2D input, while stride $(1, 1, 1)$ means the filters move one voxel at a time along the spatial and spectral dimension in 3D convolution over 3D input. Using a larger stride will produce smaller feature maps due to the reduced number of overlapping positions of the filters. The *third* parameter is zero-padding, which involves adding zeros around the border of the input along the spatial and spectral (if the input is 3D data) dimensions. This allows the filters to be applied to the bordering elements of the input data. It is primarily employed to maintain the spatial and spectral dimensions of the input, ensuring that the dimensions of both the input and output remain the same. When zero-padding is used, it is referred to as wide convolution. On the other hand, not using zero-padding would result in narrow convolution, where the filters are not applied to the bordering elements. Preserving dimension size is particularly crucial when constructing deeper networks, as it prevents the height, width, and depth (if the input data is 3D) from progressively diminishing as we delve into subsequent layers. Figure 3.7 shows a 2D convolution operation over an input data with padding and stride equal to $(1, 1)$. As it is shown the output image has the same size as input image.

3.1.2.2 Detector Layer

Within CNNs, the convolutional layer primarily produces feature maps that are linear transformations of its input data. To equip the network with the capability to model non-linear data relationships, non-linearities must be introduced. This is achieved using the detector layer.

Activation functions, integral to the detector layer, introduce the required non-linearity. They are

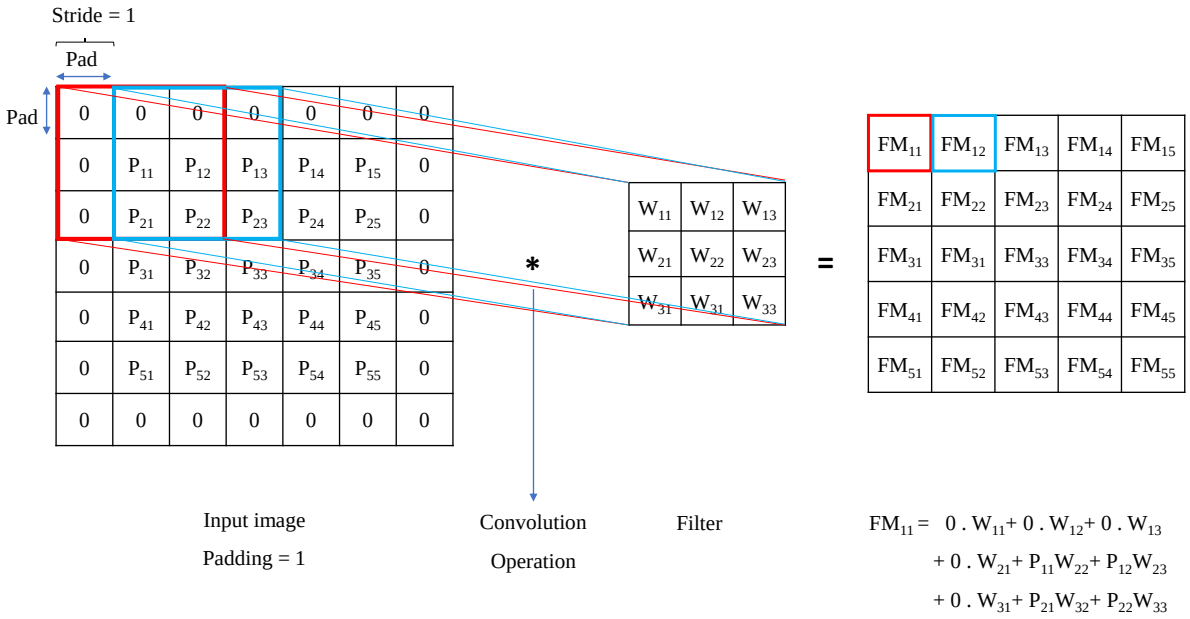


Figure 3.7: Convolution operation. The image shows sliding convolutional filter of size 3×3 over an input image with padding = 1 and stride = 1 to generate a feature map.

applied element-wise to each value of the convolutional output. A list of activation functions can be found in [67]. Applying these activation functions to 2D or 3D convolutional outputs retains the non-linearity throughout the output volume. For clarity, the ReLU is one of the most popular activation functions and is especially prevalent in CNNs due to its computational efficiency and its efficacy in mitigating the vanishing gradient problem (discussed in 3.1.4.1.3). Mathematically, ReLU is defined as

$$f(x) = \max(0, x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

This means that any negative value in the convolutional output is set to zero, while positive values remain unchanged. By introducing this non-linearity, ReLU ensures that the network can learn more complex representations, especially when stacked in deep architectures.

For a 2D convolution output $O_{2D}[i, j]$, where i and j represents spatial values, the application of ReLU would be

$$O_{2D,ReLU}[i, j] = \max(0, O_{2D}[i, j]). \tag{3.2}$$

Similarly, for a 3D convolution output $O_{3D}[i, j, k]$, where i, j represents spatial values and j represents spectral value, the ReLU activation can be applied as

$$O_{3D,ReLU}[i, j, k] = \max(0, O_{3D}[i, j, k]). \tag{3.3}$$

In essence, ReLU's simple yet powerful operation aids in the propagation of positive values while nullifying negative ones, promoting a more dynamic range of learned features in the network.

3.1.2.3 Pooling Layer

The addition of convolutional layers in a network increases the depth of the output layer, resulting in a greater number of parameters that need to be optimized (see Section 3.1.3.2 for more details about optimization). This increase in network complexity impacts the computational operations involved in the learning process. To address this, pooling layers are employed to reduce network size by decreasing the number of parameters passed to the subsequent layer. Pooling achieves this by applying statistical functions, such as maximum or average, to resize the input and minimize parameter count. The purpose of pooling layers is to downsample the feature maps generated by the convolutional layers, effectively reducing computational complexity. It is common practice to incorporate pooling layers after every one or two convolutional layers in a CNN architecture [62]. Two main types of pooling, max pooling and global average pooling, exist. Max pooling selects the maximum pixel value within sliding windows, while global average pooling calculates the average values of all pixels in the feature map. Figure 3.8 depicts the max and average pooling operation.

3.1.2.4 Fully Connected Layer

After applying convolutional and pooling layers to extract features from images, these features needed to be classified. For classification purposes, a fully connected layer or dense layer is utilized. To feed the output features into the fully connected layer for classification, the features are flattened into a vector with dimensions of $(1, n)$. For example, if the features have dimensions of (x, y, z), the flattened vector will be $(1,\ x \times y \times z)$ (see Figure 3.9). One or more fully connected layers can be incorporated, with each layer containing one or more neurons.

3.1.3 Training and Performance Evaluation of CNNs

3.1.3.1 Error Function

This function provides a numerical insight into the disparity between the anticipated outcome and the prediction made by the neural network. A high value denotes greater disparity, and a lower value indicates accuracy. This discrepancy helps assess the network's effectiveness and guides potential refinements [62].

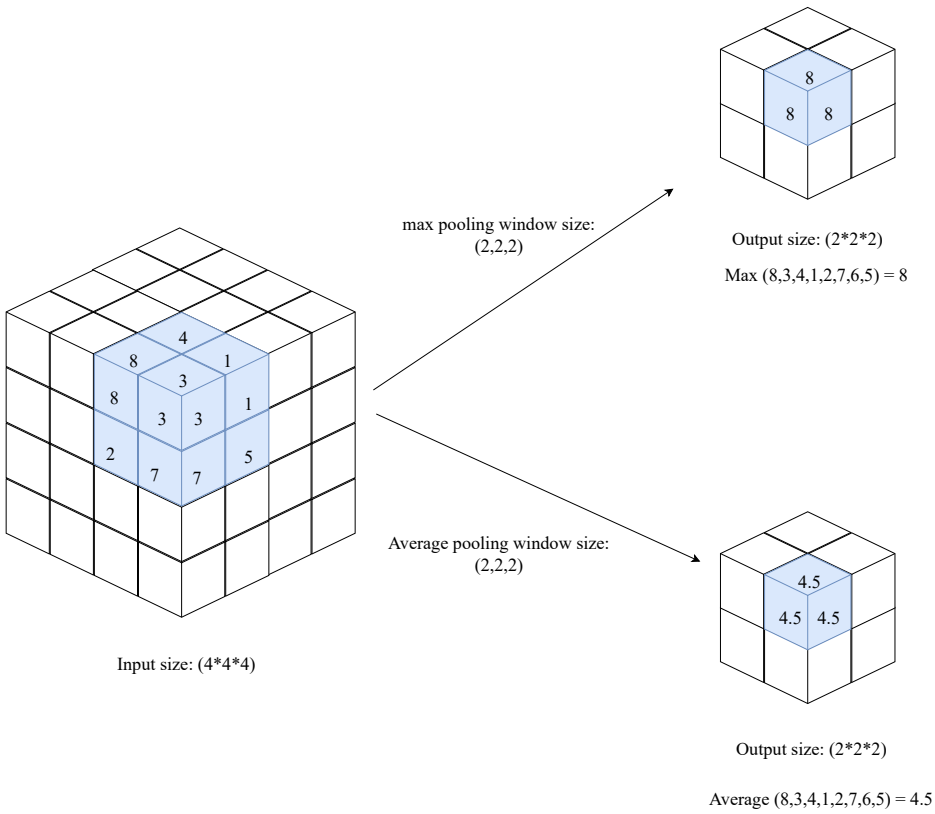


Figure 3.8: Pooling layer. The image shows operation of max and average pooling with window size of (2×2×2) over a data cube. Values in blue color from input are distribution of integer numbers from 1 to 8 without repetition.

The selection of an apt error function is crucial, setting the stage for the ensuing optimization challenge (detailed further in Section 3.1.3.2). In this context, optimization aims to tweak the network's weights to minimize the error function. With a clear error function, optimization algorithms can be employed to their utmost efficacy. From an optimization standpoint, the mission is to pinpoint the optimal parameters that minimize the error. Recognizing this deviation is paramount for adjustments during iterative training. Several techniques exist to streamline this error minimization. A salient point is that errors are consistently treated as positive values [62]. This ensures individual prediction errors don't offset one another during average error computation. Given the emphasis on classification within the research, understanding the mechanisms for evaluating predictions in binary and multiclass settings is crucial.

In binary classification tasks, we aim to distinguish between two distinct classes. These two classes are typically represented by the target variables, which can take on one of two possible values. For simplicity, we can represent these two classes using the values 0 and 1.

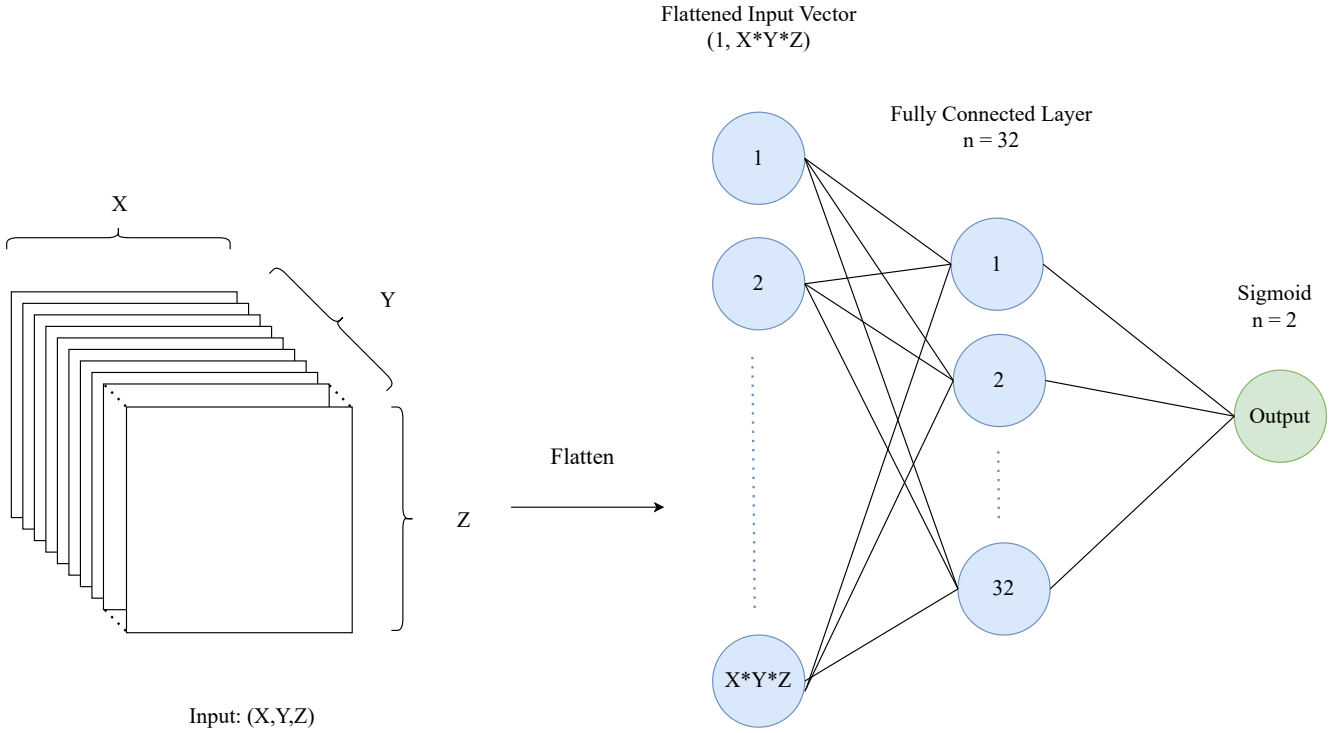


Figure 3.9: Fully connected layer. The image shows flattening the output feature of size (x, y, z) into dimension of $(1, x \times y \times z)$ and then fed to a fully connected layer of size 32, as the problem is a binary classification, sigmoid function is used.

The Sigmoid function is especially useful in this context. This function has the property of converting any real-valued input number into a value between 0 and 1, making it perfect for representing probabilities. Let P be the output of the Sigmoid function, representing the predicted probability that a given instance belongs to class 1. Given this definition, the predicted probability that the same instance belongs to class 0 is $1 - P$. In binary classification tasks, Y denotes the actual class label of each instance, where $Y = 1$ if the instance actually belongs to class 1 (positive class), and $Y = 0$ if it belongs to class 0 (negative class). The Binary Cross-Entropy loss, sometimes referred to as log loss, quantifies the difference between the predicted probabilities and the actual class labels. The formula for Binary Cross-Entropy loss is

$$\text{Binary Cross-Entropy Loss}(Y, P) = -Y \log(P) - (1 - Y) \log(1 - P) = \begin{cases} -\log(1 - P) & \text{if } Y = 0, \\ -\log(P) & \text{if } Y = 1. \end{cases} \tag{3.4}$$

For classification problems involving more than two classes, the Softmax function is commonly used to transform the output of a neural network into probabilities that sum up to 1 across all classes. Given an output vector $\mathbf{z} = [z_1, z_2, z_3]$ from the neural network, which corresponds to the classes A, B, and C,

the Softmax function returns a normalized probability distribution:

$$\text{Softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{3} e^{z_j}}, \tag{3.5}$$

Yielding probabilities are as follows:

$$P(A) = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}, \tag{3.6}$$

$$P(B) = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}, \tag{3.7}$$

$$P(C) = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}. \tag{3.8}$$

When it comes to measuring the model's performance, we can use the Multiclass Cross-Entropy loss. The Multiclass Cross-Entropy loss is an extension of the binary version, aiming to capture the divergence between true labels and the predicted probabilities. Given a true label vector $Y = [Y_1, Y_2, Y_3]$ and predicted probabilities $P = [P(A), P(B), P(C)]$, the loss becomes

$$\text{Multiclass Cross-Entropy Loss}(Y, P) = -\sum_{i=1}^{3} Y_i \log(P_i). \tag{3.9}$$

3.1.3.2 Error Minimization via Optimization

Optimization lies at the heart of training neural networks. It is through optimization algorithms [62] that we iteratively refine certain configurable parameters, known as "hyperparameters", such as weight initialization and learning rates. The ultimate goal is to minimize errors and enhance model accuracy. In this context, when we talk about optimizing the error function in neural networks, it primarily means fine-tuning the weights and biases to achieve a set of optimal values that yield the least error.

One might wonder: Why not just evaluate all possible combinations of weights and biases. While this brute-force strategy might be plausible for simpler networks with fewer inputs and neurons, it is a daunting, if not impossible task for larger, more complex networks. Even with the might of today's fastest supercomputers, comprehensively examining every conceivable combination is impractical in terms of time and computational resources.

This brings us to a more pragmatic and efficient solution: gradient descent. It is a central optimization method for neural networks. Gradient descent, along with its variants like batch, stochastic, and mini-batch methods, refines the model by iteratively adjusting its parameters based on the error. We will discuss these optimization methods in more detail in the subsequent sections.

3.1.3.2.1 Batch Gradient Descent

Batch gradient descent (also known as vanilla gradient descent) is an optimization algorithm commonly used in training neural networks. The gradient (also derivative), which indicates the slope or rate of change of an error function curve, plays a central role in this algorithm. The goal of gradient descent is to iteratively update the weights to minimize the error by descending the slope of the error curve. To understand how gradient descent works, we can visualize the error function in a 3D graph depicted in Figure 3.10 (a). Starting from an initial weight point (shown in star), we take steps down the curve by determining the direction and size of each step. The direction is determined by calculating the derivative of the error function, representing the steepest descent. By examining the surroundings, we choose the direction that leads to the deepest descent. The step size, also known as the learning rate, determines the magnitude of each step. It is a crucial hyperparameter in neural network training. A larger learning rate accelerates learning but may result in overshooting the minimum error and causing oscillation. Conversely, a smaller learning rate ensures a more precise descent but prolongs the training process. To combine direction and step size, we multiply the derivative (direction) by the learning rate (step size). This yields the change in weight for each step. Adding a minus sign ensures we move in the opposite direction of the slope to descend the error mountain. Since, the new weights can be corrected as follows:

$$W_{Next} = W_{Previous} - \alpha \frac{dE}{dW},$$

where α is the learning rate and $\frac{dE}{dW}$ is the derivative term of the error function with respect to weight.

In practice, choosing an appropriate learning rate involves careful tuning. Very large or very small values can lead to ineffective training. By initializing with a reasonable learning rate, such as 0.1 or 0.01, and observing the network's performance, we can further fine-tune the learning rate to achieve optimal results.

While batch gradient descent is a powerful algorithm for minimizing error, it has two significant drawbacks. Firstly, not all cost functions exhibit a simple bowl-like shape. Some may have irregular terrain with holes and ridges, making it challenging to reach the global minimum error. If the algorithm starts at a local minimum, it will converge to that point instead of finding the overall lowest error value. Secondly, batch gradient descent computes gradients using the entire training set at each step. The training set consists of labeled data that the algorithm uses to learn and adjust its parameters. Using the entire training set means that every single data point from this set contributes to the calculation of the gradient at each step. This becomes computationally expensive and slow when dealing with large datasets. To address these issues, an alternative approach called stochastic gradient descent is commonly used.

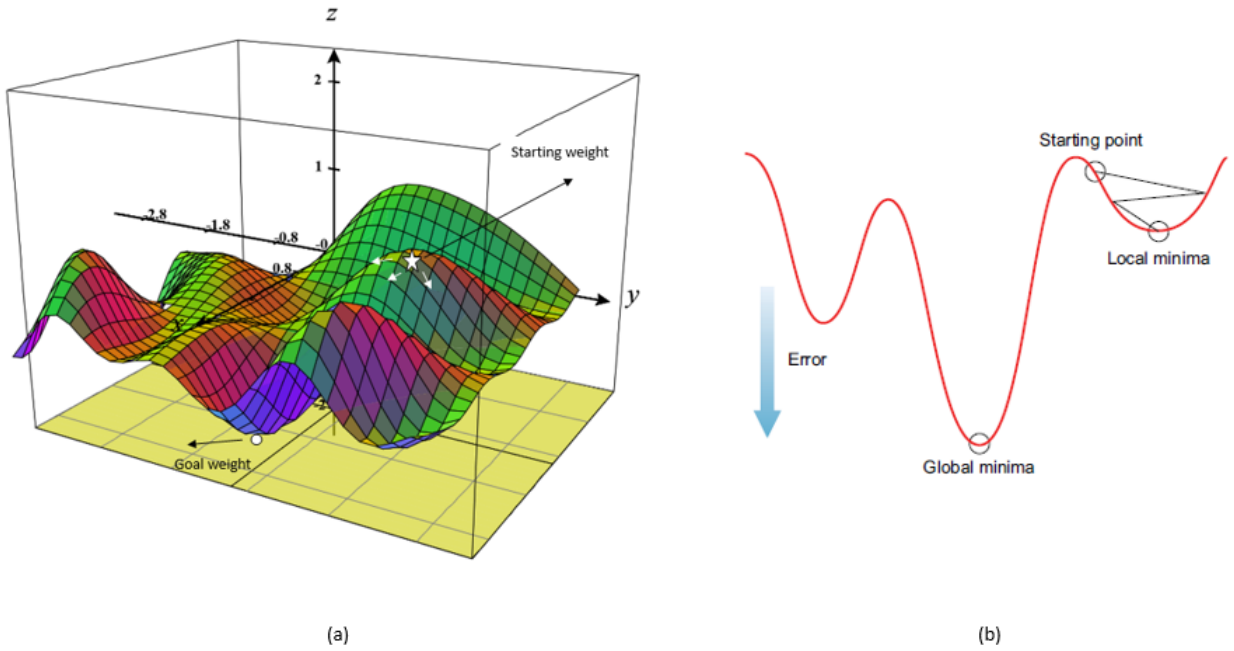


Figure 3.10: (a) Minimize the error by starting from a random initial starting weight. Move toward a direction which produces minimum error value and that is where the goal weight is located. (b) Complex error function curve with multiple local minimums [62].

3.1.3.2.2 Stochastic Gradient Descent

In stochastic gradient descent, the algorithm randomly selects individual data points and performs gradient descent on each point separately. This approach allows for exploring multiple starting points and considering local minima, leading to the identification of the global minimum as the minimum value across all local minima. Stochastic gradient descent differs from batch gradient descent, where gradients are computed over the entire training set. In batch gradient descent, the descent path appears smooth and straight, while in stochastic gradient descent, the path towards the global minimum may exhibit zigzag patterns due to its stochastic nature. Stochastic gradient descent's focus on improving the fit for a single training example results in faster progress, but without the guarantee of a downward step with each iteration. Although stochastic gradient descent may bounce around the region close to the global minimum, this is generally acceptable in practice, as proximity to the global minimum suffices for most applications. Overall, stochastic gradient descent typically outperforms batch gradient descent in terms of speed and performance [62].

3.1.3.2.3 Mini-batch Gradient Descent

Mini-batch gradient descent offers a compromise between batch gradient descent and stochastic gradient descent [62]. Unlike batch gradient descent, which updates weights using the entire dataset, and stochastic gradient descent, which updates weights based on a single data point, mini-batch gradient descent strikes a balance by utilizing mini-batches. This approach allows for more frequent weight updates than batch gradient descent, resulting in faster convergence. Additionally, mini-batch gradient descent leverages vectorized operations, which enhance computational efficiency and provide performance gains over stochastic gradient descent. Figure 3.11 shows the convergence behaviour of optimization methods discussed in this Section.

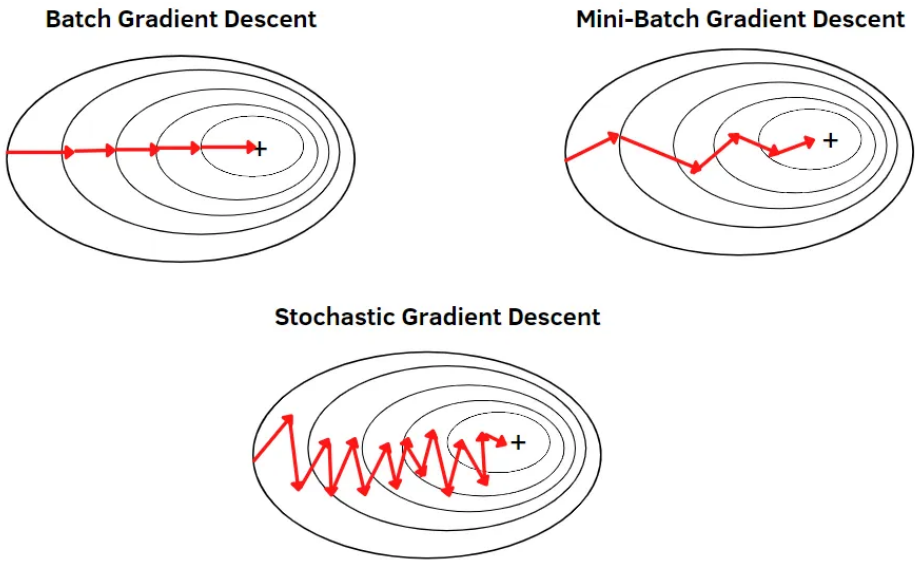


Figure 3.11: Convergence behaviour of batch gradient descent, stochastic gradient descent, and mini-batch gradient descent [68].

There have been numerous variations and enhancements to the gradient descent algorithm over the years, with ongoing research in this area. Some of the common algorithms include Adam [69], RMSprop, Adagrad, Momentum, and Nesterov accelerated gradient [70].

3.1.3.3 Backpropagation: The Learning Mechanism

Backpropagation serves as the foundation for neural network learning. The training process typically involves three steps: feedforward, error calculation, and weight update using a gradient descent optimization algorithm. Backpropagation specifically refers to the backward pass, where derivatives

of the error with respect to each weight are propagated from the output layer to the input layer to adjust the weights. This iterative process continues until the minimum error is reached. When dealing with a network with multiple layers and weights, the chain rule of calculus is employed to compute the derivatives of the total error with respect to each weight [62]. Figure 3.12 shows an example of a neural network with two inputs denoted by x_1 and x_2, one hidden layer with two neurons shown in h_1 and h_2, and one output layer that is $\hat{p}_i$ and is the predicted probability. Sigmoid is also used as an activation function in this example. The equations that describe the network in Figure 3.12 are defined as

$$z_1{}^{(1)} = w_{11}{}^{(1)}x_1 + w_{21}{}^{(1)}x_2 + b_1{}^{(1)}, \tag{3.10}$$

$$z_2{}^{(1)} = w_{12}{}^{(1)}x_1 + w_{22}{}^{(1)}x_2 + b_2{}^{(1)}, \tag{3.11}$$

$$h_1 = \sigma(z_1{}^{(1)}), \tag{3.12}$$

$$h_2 = \sigma(z_2{}^{(1)}), \tag{3.13}$$

$$z^{(2)} = w_1{}^{(2)}h_1 + w_2{}^{(2)}h_2 + b^{(2)}, \text{ and} \tag{3.14}$$

$$\hat{p_i} = \sigma(z^{(2)}). \tag{3.15}$$

Here, z in the notation indicates the input of hidden layer neurons before application of Sigmoid activation function, the superscript shows the layer number, so (1) indicates the hidden layer and (2) denotes the output layer. The subscript shows where the weight's source and destination neuron (node) are. For example, $w_{12}{}^{(1)}$ means the weight associated with the connection from input x_1 to h_2 neuron in hidden layer.

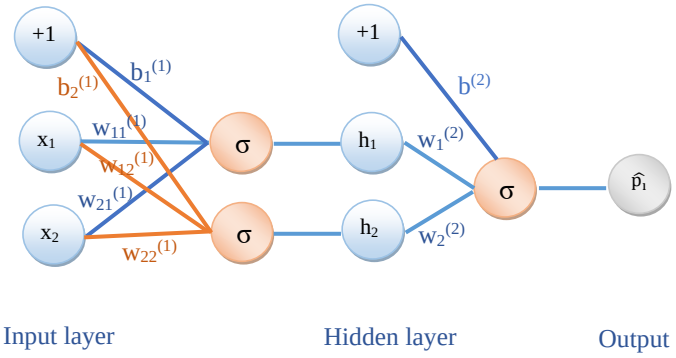


Figure 3.12: The architecture of a simple neural network during backpropagation. The network consists of an input layer with two neurons x_1 and x_2, a hidden layer with two neurons h_1 and h_2, and an output layer with a single neuron $\hat{p}_i$. Bias units are added to the input and hidden layers. The weights w and biases b are adjusted during backpropagation to minimize the error in the output $\hat{p}_i$.

The error function considered for this example is as follows,

$$\text{Error Function (E)} = \frac{1}{2}|\hat{p} - y|^2 = \sum_{i=1}^{n} \frac{1}{2}|\hat{p}_i - y_i|^2 = \sum_{i=1}^{n} E_i, \tag{3.16}$$

where $\hat{p}$ stands for predicted output for all training examples and y is the true output for all training examples. The goal is finding the effect of each terms, weighs and biases, and finding the direction where the error function decreases the fastest. In order to optimize these terms, gradients of all terms need to be calculated. In essence, the computed gradient in a layer is propagated backward. The gradient of error function (E) to ${w_1}^{(2)}$ is calculated as follows:

$$\frac{\partial E}{\partial {w_1}^{(2)}} = \sum_{i=1}^{n} \frac{\partial E_i}{\partial {w_1}^{(2)}}. \tag{3.17}$$

To compute the Equation 3.17, as ${w_1}^{(2)}$ in Equation 3.14 changes, $z^{(2)}$ gets affected and in the same way changes of $z^{(2)}$ affect $\hat{p_i}$ in Equation 3.15, and finally alteration of $\hat{p_i}$ affects error function in Equation 3.16. Using the chain rule, the Equation 3.18 shows the series of gradients that need to be computed and multiplied for final result:

$$\sum_{i=1}^{n} \frac{\partial E_i}{\partial {w_1}^{(2)}} = \frac{\partial z^{(2)}}{\partial {w_1}^{(2)}} \cdot \frac{\partial \hat{p_i}}{\partial z^{(2)}} \cdot \frac{\partial E_i}{\partial \hat{p_i}}. \tag{3.18}$$

3.1.4 Challenges in Model Training: Overfitting and Underfitting

CNNs can face performance degradation due to overfitting or underfitting. Overfitting is when a model adapts too closely to the training data, almost memorizing it, without truly grasping the underlying patterns. This results in poor performance on unseen data. Conversely, underfitting happens when a model is overly simplistic, failing to capture the complexity of the training data [62].

3.1.4.1 Mitigating Training Challenges

Various techniques have been developed to counteract overfitting, ensuring models are both accurate and generalizable. Sections 3.1.4.1.1 to 3.1.4.1.3 provides the common techniques.

3.1.4.1.1 Dropout

One of the strategic techniques devised to combat overfitting is the utilization of dropout layers. Dropout operates by randomly disabling a predefined proportion of neurons in a network layer during both

forward and backward propagation passes. This proportion, termed the dropout rate, is a tunable hyperparameter set during network design. By introducing this randomness, dropout ensures that no individual neuron or a set of neurons singularly dictate the network's behavior. This randomness pushes the network towards a more democratized decision-making process, wherein every neuron equally contributes to the final outcome.

A consistent reliance on specific neurons or features can instigate a phenomenon called neuron co-dependency, where certain neurons become overly specialized. Such specialization is a precursor to overfitting. Dropout counters this by instilling a sense of uncertainty; any neuron can be "dropped" or deactivated during training. Consequently, the network is coaxed into distributing the "knowledge" more uniformly across its neurons, thereby mitigating overfitting. Interestingly, this behavior draws parallels with ensemble learning [62]. In ensemble methods, multiple weaker models (classifiers) are trained independently, and their combined outputs often result in a more robust and generalized model. Dropout can be conceptualized similarly: by randomly deactivating neurons, multiple "sub-networks" are trained within the main network. The collective behavior of these sub-networks equips the model with better generalization capabilities.

Within the architecture of CNNs, dropout layers are predominantly placed between the fully connected layers, especially post the flattening of convolutional features. This strategic placement particularly addresses overfitting tendencies inherent to the dense, fully connected layers of CNNs. While the efficacy of dropout in convolutional and pooling layers is recognized, its optimal placement and impact in these layers remain active topics of research.

3.1.4.1.2 L1 and L2 Regularization

Regularization offers a mathematical approach to prevent overfitting by introducing certain constraints. Both L1 (Lasso) and L2 (Ridge or Weight Decay) regularization techniques add penalty terms to the model's loss function based on the magnitude of the model's weights. L2 regularization works by penalizing the error function through the addition of a regularization term. This term reduces the weight values of neurons, making them smaller and closer to zero, which simplifies the model. The regularization term is calculated using the regularization parameter δ, the number of instances (m), and the weight values (w) [62]. The formula is as follows:

$$\text{Error Function}_{new} = \text{Error Function}_{old} + \frac{\delta}{2m} \sum ||w||^2. \tag{3.19}$$

By incorporating the regularization term into the error function, the weights are updated during the backpropagation process in a way that leads to smaller weights. This reduction in weights promotes

a simpler neural network with fewer neurons. However, L2 regularization does not make the weights exactly zero; it only decreases their impact.

In L1 regularization, a regularization term equal to absolute value of the weights is added to error function as follows:

$$\text{Error Function}_{new} = \text{Error Function}_{old} + \frac{\delta}{2m} \sum ||w||. \tag{3.20}$$

L1 regularization encourages the weights to be 0, leading to a more sparse network with more 0 values. Therefore, some of the neurons outputs are not considered for computation. In contrast to L2 regularization, that L2 norm penalizes the larger weights more severely and results in less sparse weights. Both L1 and L2 regularization have a parameter δ which must be fine-tuned and plays an important role in regularization effect. A penalty that is excessively strong may result in underfitting, while one that is too weak could lead the model to overfit the training data.

3.1.4.1.3 Batch Normalization

To understand how batch normalization functions, it is imperative to first grasp the concepts of normalization and its distinction from standardization. During normalization, data is transformed to lie within a range of 0 to 1. In contrast, standardization adjusts data to have a mean of 0 and a variance of 1. The need for normalization arises when instances of a dataset each span different ranges, complicating the neural network's task of learning optimal weight values. In such scenarios, the weights could vary drastically from one another, which is not ideal for learning.

One significant challenge in training deep neural networks is the issue of vanishing and exploding gradients. As the gradient of the loss function is back-propagated through the network, layers far from the output can sometimes receive gradients that are too small (vanishing) or too large (exploding). This hampers the effective training of the network. The vanishing gradient problem can slow down learning or make it come to a complete halt, as weight updates become negligibly small. Conversely, the exploding gradient problem can cause weight updates to be exceedingly large, leading to network instability. Suitable weight initialization techniques and appropriate activation functions can provide initial remedies; however, these challenges might still manifest later during training.

In this context, batch normalization emerges as a solution, specifically designed to address the problem of covariate shift. Covariate shift refers to the change in the input distribution to a learning algorithm, which can hamper its ability to generalize. When training deep networks, internal layers can experience a shift in their input distributions as the weights of the preceding layers change, a phenomenon termed

internal covariate shift. This dynamic variability in input distributions across layers and training epochs makes the network training process less stable and requires lower learning rates and careful weight initialization.

Batch normalization, by normalizing activations of each layer, aims to mitigate this internal covariate shift, thereby improving training efficiency. Rather than limiting normalization to the input layer, batch normalization normalizes the output of every layer. This ensures consistent means and variances for inputs across layers. By applying this normalization immediately before each layer's activation function, the technique allows the network to determine optimal scales and means for its inputs.

The process first ensures data has a mean of 0 and variance of 1. Following this standardization, it introduces scaling and offset parameters with the following formula:

$$z^{(\mathrm{i})} = \gamma \hat{x}^{(\mathrm{i})} + \beta, \tag{3.21}$$

where $\hat{x}^{(\mathrm{i})}$ represents post-standardization values, γ is the scaling parameter, β denotes the offset, and $z^{(\mathrm{i})}$ is the value after applying both scaling and offset. Both γ and β are learnable parameters, adjusted during training.

The application of batch normalization accelerates convergence during training and enhances network robustness. By mitigating the effects of covariate shift, it paves the way for subsequent layers to process more consistent and stable data distributions. This normalization within the network not only bolsters performance but also expedites the network's convergence. Moreover, by maintaining consistent activations, batch normalization can sometimes reduce the need for other regularization techniques. This can potentially decrease a network's tendency to overfit and sometimes diminishing the need for other regularization techniques.

3.1.5 Evaluation Metrics

The evaluation of machine learning algorithms is a crucial aspect of any project. While a model may demonstrate satisfactory performance when assessed using a specific metric, its performance may be inadequate when evaluated using alternative metrics. One commonly employed metric is classification accuracy, which is defined in Equation 3.22 as the ratio of correct predictions to the total number of samples. This is defined as

$$Accuracy = \frac{\text{Correct predictions}}{\text{Total number of samples}} = \frac{TP + TN}{TP + FN + FP + TN}. \tag{3.22}$$

The terms in Equation 3.22 are fundamental to understanding the performance of classification algorithms:

- TP (true positive): This is when the model accurately predicts a positive class as positive.
- TN (true negative): This occurs when the model accurately predicts a negative class as negative.
- FP (false positive): This happens when the model mistakenly predicts a negative class as positive.
- FN (false negative): This is an instance where the model fails to identify a positive class, incorrectly labeling it as negative.

However, relying solely on accuracy is insufficient for providing a comprehensive evaluation of a model's performance. In the following, some common types of evaluation metrics are explained.

3.1.5.1 Precision and Recall

Precision measures the accuracy of positive predictions made by the model. It answers the question: out of all the instances predicted as positive, how many were actually true positives. In other words, precision quantifies the model's ability to avoid false positives. Whereas, recall, also known as sensitivity, measures the model's ability to correctly identify positive instances. It answers the question: out of all the actual positive instances, how many were correctly identified by the model. Recall helps assess the model's ability to avoid false negatives. In order to compute these metrics, a confusion matrix is needed (see Figure 3.13). It summarizes the predictions made by the model on a test dataset and compares them to the actual ground truth labels. A confusion matrix is typically represented as a square matrix, with rows representing the true classes and columns representing the predicted classes. Precision and recall formulas are shown below. These two metrics provide complementary information about the performance of a model. In this term F-score can mitigate this issue as described next.

$$Precision = \frac{TP}{TP + FP} \tag{3.23}$$

$$Recall = \frac{TP}{TP + FN} \tag{3.24}$$

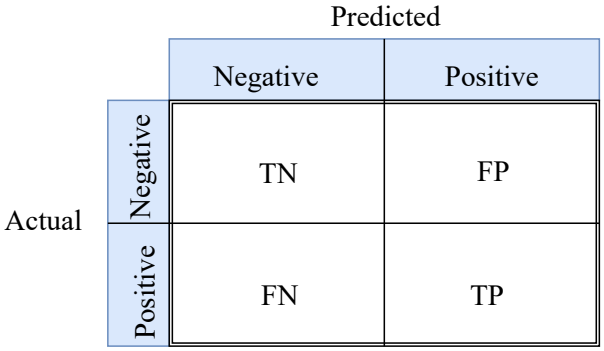


Figure 3.13: Confusion matrix depicting the classification outcomes of a predictive model. The matrix is divided into four quadrants, with TP and TN representing accurate predictions for the positive and negative classes, respectively. FP indicate instances incorrectly classified as positive, while FN represent positive instances that were incorrectly classified as negative. This matrix serves for assessing the model's predictive performance and diagnosing areas for improvement.

3.1.5.2 F1-Score

The F1-Score is a metric that combines precision and recall into a single measure. It is commonly referred to as the harmonic mean of the two metrics, which is more suitable for ratios like precision and recall compared to the traditional arithmetic mean. The Equation 3.25 for calculating the F1-Score balances the importance of precision and recall by requiring both to have higher values for the F1-Score to increase. Therefore, it ensures that very low values in either precision or recall will result in a lower overall score.

$$\text{F1-Score} = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3.25}$$

3.1.6 Comparative Analysis of 1D, 2D, and 3D Convolutional Operations in Hyperspectral Data Processing

In processing hyperspectral data, convolutional operations play a pivotal role in feature extraction and data classification. As illustrated in Figure 3.14, the convolutional operations differ based on the dimensionality of the kernel used: 1D (spectral), 2D (spatial), and 3D (spatial-spectral). The given figure provides a schematic representation of the movement directions of these kernels within a volumetric dataset, commonly referred to as a hypercube.

- **1D-CNN (Spectral Convolution)**: This approach employs a 1D kernel that navigates solely

across the spectral dimension (Z-direction) of the hypercube. While it predominantly extracts spectral features, it misses the spatial interdependencies inherent in adjacent areas of the hypercube.

- **2D-CNN (Spatial Convolution)**: In contrast to the 1D-CNN, the 2D-CNN uses a 2D kernel, sliding across the spatial dimensions (X and Y directions) of the hypercube. This method captures spatial patterns and relationships but neglects the informative spectral features residing in the depth (Z-direction) of the hypercube.

- **3D-CNN (Spatial-Spectral Convolution)**: The 3D-CNN offers a comprehensive approach that uses a 3D kernel, traversing both spatial (X and Y directions) and spectral (Z-direction) dimensions of the hypercube. Such convolution amalgamates the spatial patterns and relationships with the rich spectral information embedded in the depth of the hypercube.

Traditional classification methods based on 1D-CNN and 2D-CNN exhibit inherent limitations in hyperspectral data processing. This is attributed to the intertwined nature of spatial and spectral features in hyperspectral data. Sole reliance on one feature type can yield subpar classification results. The 1D-CNN, being spectral feature-centric, might overlook spatial correlations, while the 2D-CNN, emphasizing spatial features, might bypass spectral intricacies. However, a 3D-CNN can extract spatial-spectral features from the volumetric data. This is due to its ability to incorporate the spectral dimension in addition to the spatial dimensions, which enables it to model and learn more complex spatial-spectral representations. On the flipside, a straightforward generalization of $l \times l$ kernels to their 3-dimensional versions increases the number of operations that must be performed to apply a kernel by a factor of l. Not only the application of these kernels is computationally more costly, but also their training becomes more challenging, as the kernel has more parameters. Taking no other model-changes into account (as, for example in [71]) we would have to shorten kernel-sides by $\sqrt[3]{l^2}$ to keep computational complexity at roughly the same level.

3.2 Overview of 3D-CNNs-based Models Used in HSI Classification of Diseased and Defective Crops

The objective of neural network architecture design is to create a model that can effectively learn from the input data and generalize well to new, unseen data. In order to achieve this for HSI classification, a network architecture must be developed that can capture the complex spectral and spatial information present in the data. As an integral aspect of designing a neural network, the decision of the number of

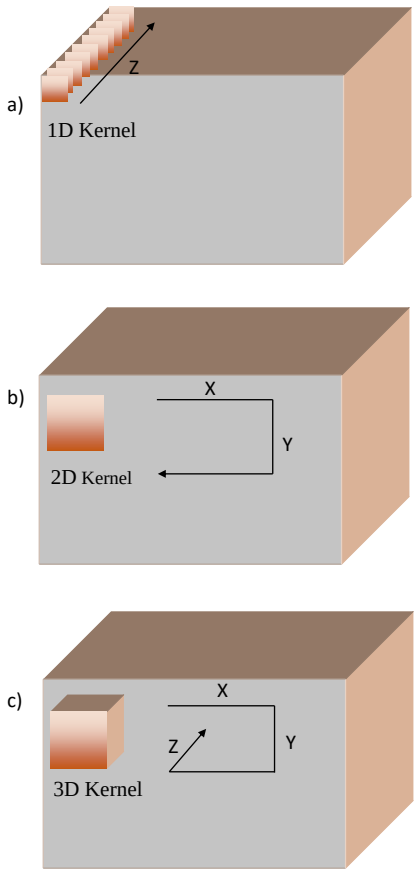


Figure 3.14: Movement direction of convolution process using 1D, 2D, and 3D kernels in hypercube. A schematic overview of movement directions for (a) 1D, (b) 2D, and (c) 3D convolutions in CNNs over a hypercube is depicted. The X and Y directions denote the movement of the kernel across the spatial dimensions, while the Z direction showcases the movement across the spectral dimension.

layers to use is a crucial one. While the inclusion of more layers in a network has been shown [72, 73] to improve performance, it can simultaneously present several challenges. A notable challenge arises from the potential occurrence of vanishing or exploding gradients [74], where the gradient signal becomes too small or too large as it backpropagates through the layers during training (see also Subsection 3.2.2.1). Such a phenomenon makes it difficult for the network to learn and adjust its weights effectively. Moreover, as the number of layers and parameters increases, the risk of overfitting rises, which is characterized by the network's ability to perform exceptionally well on the training data, but not generalize well on unseen data. Consequently, the network may become computationally expensive to train and use due to its high processing power and memory requirements. Additionally, gradient computation time increases, thereby impeding the training's efficiency. Hence, careful consideration

of these challenges and trade-offs is imperative to designing a network that balances complexity and performance.

This section provides a comprehensive review of the architectural frameworks of 3D-CNN-based models that have been employed for detecting and classifying diseases in agricultural crops through non-UAV-based HSI. This includes charcoal rot in soybeans [75], mold in peanuts and strawberries [76, 77], bacterial leaf blight (BLB) in rice [78], grapevine vein-clearing virus (GVCV) in grapevines [79], and potato late blight (PLB) in potatoes [80]. We further explore the architecture of 3D-CNN-based models for identifying specific defects in crops, such as decay in blueberries [71], bruise and brown spots in fruits [81, 82], heat stress in rice [83], as well as black, fermented, shell, and broken coffee defects in beans [7].

To categorize these models, we differentiate between hybrid and non-hybrid structures. A model is deemed hybrid if it incorporates modules that enhance feature extraction and overall performance or melds 2D-CNN elements within a 3D-CNN framework. Subsequently, we detail these 3D-CNN models' application in classifying both diseased and defected crops using HSI data, with a summarization provided in Tables 3.1 and 3.2 at the end of this Section.

3.2.1 Non-hybrid Architectures

Reference [75] presented a 3D-CNN model to classify healthy and diseased crop. This model consists of two convolution layers with max pooling layers, and two fully connected layers, trained using the Adam optimizer. To prevent overfitting, dropout mechanisms were used after the first max pooling and first fully connected layer. A Weighted Binary Cross Entropy (WBCE) function of the form

$$L_{WBCE}(y, \hat{y}) = -[\beta \cdot y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})] \tag{3.26}$$

was implemented to address imbalanced training data. Here y and $\hat{y}$ represent binary variables for whether the ground truth and predicted result belong to a given class, respectively. Using the coefficient β the WBCE loss function assigns higher weights to the minority class, for example, the false negatives rate decreases if β is set higher than 1, while setting it smaller than 1 reduces the false positive rate.

Although the above technique can classify and detect diseased crop, detection of asymptomatic diseased crops at early stage and differentiating it from the healthy crops can be more challenging. In this respect, Reference [77] developed a 3D-CNN model that improves classification accuracy of the asymptomatic diseased crop without modification and preprocessing of input HSI data. The model consists of four 3D convolution layers in which the first and fourth layers each are followed by a 3D max pooling layer

and a batch normalization. The rest of the convolution layers are only followed by batch normalization. The output of the last 3D convolution layer is passed through a global average pooling and two dense layers. In Reference [77] the results were improved further by preprocessing the input HSI data which went through spectral differentiation, vertical expansion, and smoothing.

Automating the classification procedure of HSI data using software can effectively mitigate the time-consuming process of implementing a deep learning pipeline. In this respect, Reference [83] developed an open-source software that classifies seeds at pixel level using 3D-CNN. Though at first it was developed for a specific crop (rice), the test experiments over other seeds are promising. This software utilizes a 3D-CNN that consists of two 3D convolution layers, with two and four 3D convolution kernels for the first and second layers, respectively. The output is then flattened via one fully connected layer and classification is performed using a softmax function. However, this software as part of its feature extraction process does not consider the global feature relationship which can improve its ability to recognize and classify seeds accurately.

Reference [79] implemented a 3D-CNN for classification of a small size dataset. The feature extraction part is implemented based on AlexNet [84] and constitutes 5 convolutional layers followed by a flattening layer. The input layer takes an input of size 512×512×203, where 203 stands for number of bands. The first two convolutional layers are followed by a max pooling layer and a batch normalization. After that each convolutional layer is followed by a max pooling layer and after the last max pooling layer batch normalization and flattenings is applied before feeding the result into a RF or SVM for binary classification (healthy or diseased crop).

3.2.2 Hybrid Networks

3.2.2.1 3D-CNN Architectures Based on ResNet

To address the challenges of vanishing or exploding gradients [71] proposed to leverage residual convolutional blocks within a 3D deep ResNet architecture. This approach reduces the number of channels through the use of identity residual blocks and a convolutional residual blocks. The identity residual block maintains the same input and output dimensions, while the convolutional residual block changes the number of channels. The use of a 1×1×1 convolution kernel as the shortcut in the convolutional residual block reduces the number of parameters and computational complexity.

To further improve efficiency, Reference [71] adopts a bottleneck structure that reduces the number of required convolution operations. Each convolutional layer is followed by a batch normalization layer to

prevent vanishing gradient and enhance convergence rate. The network uses Exponential Linear Unit (ELU) as the non-linear activation function, which addresses the problem of dying neurons in ReLU.

In order to identify the appropriate hyperparameters, the authors employed a Tree-structured Parzen Estimator (TPE) as an optimization algorithm. The TPE utilizes a probabilistic model to approximate the distribution of the objective function and guides the search for optimal hyperparameters. For the classification, a 7×7×1 global pooling layer and a fully connected are utilized. This model halves the number of parameters and improves the computational time up to 10%. Moreover, the study of Reference [81] over performance of well-known architectures in detection of defective crop demonstrates that residual connections achieve higher accuracy, while training faster despite having more parameters.

3.2.2.2 Hypernet-PRMF Network

Reference [76] presented a feature pre-extraction and a multi-feature fusion block to extract peanut characteristics from hyperspectral data. The feature pre-extraction includes constructing a Peanut Recognition Index (PRI) based on two informative bands to distinguish healthy, moldy, and damaged peanuts.

The multi-feature fusion block is a technique used in image segmentation to fully extract spatial and spectral features from HSI data. This technique involves using multiple types of convolution kernels including 2D convolution for common texture features, separable convolution for increased feature diversity, depthwise convolution for band feature extraction, and 3D convolution for spectral change information. The convolutions are concatenated after normalization and activation functions to enhance diversity and improve recognition accuracy.

Moreover, the authors employed feature pre-extraction and multi-feature fusion block techniques in their proposed peanut recognition model, called Hypernet-PRMF network. This model works at both peanut- and pixel-level recognition. The model consists of four parts: feature pre-extraction, down-sampling, up-sampling, and prediction. The feature pre-extraction part enhances the differentiation between different peanut features. The down-sampling part reduces the size of the image while increasing the number of convolution kernels, whereas the up-sampling part reconstructs the image while reducing the number of convolution kernels. The prediction works based on the softmax function and the class of maximum predicted probability is chosen as the final recognition result. The model achieves pixel-wise recognition accuracy with the use of the watershed segmentation algorithm. This technique has the potential to be employed for detection of other crops like sorghum.

3.2.2.3 Spectral Dilated Convolution 3D-CNN

Reference [78] proposed a spectral dilated convolution (SDC)-3D-CNN model to detect crop's asymptomatic diseases at an early stage. This model consists of SDC modules along with residual blocks that prevent the gradient vanishing problem. SDC extends the idea of dilated convolution which expands the receptive field of convolution kernels without augmenting the model's parameterization. Receptive field is the portion of the input space needed to create a filter at any convolutional layer. The 3D-SDC extends the receptive field of convolutional kernels to the spectral dimension. It works based on the principle of applying a filter to an input with intermittent intervals, which are dictated by the spectral dilation rate.

The network was tested with top 50, 100, 150, and 200 significant wavelengths extracted by RF and Principal Components (PCs) of the same ranking by PCA along with different spectral dilation rate to detect healthy, asymptomatic, and symptomatic crop. The experiment result shows higher detection performance of the network using the top 50 important features extracted by RF at a dilation rate of 5.

3.2.2.4 Merged 2D- and 3D-CNN Architectures

Reference [7] developed a 2D-3D-CNN for real-time crop defect detection. This network is the detection module of a real-time coffee-bean defect inspection algorithm (RT-CBDIA). The network consists of a 2D-CNN and a 3D-CNN, the former one is responsible to extract spatial features and the latter one is accountable for extraction of spectral features. Combining these two networks can boost feature extraction by providing robust and discriminative spectral-spatial features.

The 3D-CNN is comprised of two convolution blocks with the same structure as in the 2D-CNN except that 3D convolutions and pooling layers are used. The two networks run simultaneously and their last pooling layers will be merged and fed to the a fully connected layer and then a dropout layer to avoid overfitting. Finally, a softmax layer determines each crop health status.

Likewise, Reference [80] fully extracted spatial-spectral features by merging 2D- and 3D-CNN architectures using AttentionBlock [85] and Squeeze-and-Excitation (SE)-ResNet [86]. To accomplish this, the model first creates a neighborhood block of size $11\times11\times10$ around a center pixel from the input image. Then, 2D convolution operations are used to extract spatial correlation features from the neighborhood block, and 3D convolution operations are used to capture spectral correlation features. The model uses four 2D convolutional layers and four 3D convolutional layers to capture feature maps of various spatial and spectral dimensions. By using different sizes of convolution kernels and downsampling steps, varied types of information can be captured.

Finally, the extracted feature maps are fused together to create a final set of feature maps that contain valuable and pertinent information required for effective classification. Herein, AttentionBlock and SE-ResNet play important role, as outlined immediately below.

An AttentionBlock is used to highlight important information in the fused spectral space feature map. It works by considering the similarity between each pixel in the feature map and weighting the relevant pixels with higher importance. This is achieved through a series of 2D convolutions with a kernel size of 1×1 to transform each pixel into an λ-dimensional vector, where λ represents the number of feature channels in the input tensor. The similarity between any two pixels is then calculated using the dot-product of their transformed vectors, and the results are weighted using a softmax function.

The output of an AttentionBlock is a feature map that emphasizes the relevant information while suppressing irrelevant information. This process allows AttentionBlocks to focus on the relevance between pixels in the entire feature map, rather than just the spatial range of the convolution kernel size used in traditional convolution and pooling operations. This results in better classification results with little computational complexity.

In order to enhance the representational power of CNNs, SE modules as a type of attention mechanism can adaptively recalibrate the feature maps. It does so by capturing channel-wise feature dependencies through a squeeze operation, followed by an excitation operation that learns how to weight the importance of each feature map. This mechanism allows the model to pay more attention to salient channel features and disregards the less significant ones. By integrating AttentionBlocks and SE-ResNet, the network of Reference [80] can better generalize in classification and achieve higher accuracy.

Moreover, in the context of detection of two similar crop diseases that are indistinguishable to the naked eyes, a recent study by Reference [82] developed a new network called Y-Net. The Y-Net model takes in 10×10×203 hyperspectral data cubes as input and it consists of a channel attention mechanism, a band selection module with auxiliary classifier, a 3D-2D-CNN architecture, and a classification module.

A CNN architecture is employed to conduct the band selection, where 1×1 1D convolutions are assembled to modify the parameters of the convolutional kernel in the early phase of the network training. The magnitude of the weight of the convolution kernel is indicative of the relevance of the band, with greater absolute weight values signifying more distinctive bands. The use of group convolution helps in preventing any hindrance from nearby bands, while the ReLU activation function is adopted for fast network convergence without the problem of saturation. The output of this step will be given to an auxiliary classifier that enables early-stage weight updating in the band selection block. The auxiliary

classifier module updates the loss function of the Y-Net model:

$$L(y, \hat{y}) = (1 - \theta) \cdot L_{\text{Final Classifier}}(y, \hat{y}) + \theta \cdot L_{\text{Auxiliary Classifier}}(y, \hat{y}) + \beta \cdot \sum_{j=1}^{n} W_j, \tag{3.27}$$

where y is the ground truth label, $\hat{y}$ is the predicted label, and θ and β are hyperparameters that control the trade-off between the two losses and the sparsity of the band selection module, respectively. W_j is the weight of the j^{th} band in the band selection module and n is the number of total bands. The loss is a combination of the cross entropy losses of the final classifier and the auxiliary classifier and the sum of the weights of the band selection module. The purpose of this combination is to control the classification accuracy while updating the weights of the band selection layer. The adjustment factor θ gradually decreases as the number of training iterations increases. The presence of this adjustment factor enables the Y-Net model to update the weights in the band selection module in the early stages of training and gradually shift towards training the final classifier to learn a more accurate classification model. Additionally, the auxiliary classifier helps to constrain the weight sparsity of the band selection module, ensuring that the score of unimportant features is close to zero. The results of Reference [82] show that by removing nonessential and nondiscriminative bands the accuracy of the Y-Net model increases and reduces the model size and the number of parameters. Moreover, since the band selection module is integrated into an overall architecture, the training time does not significantly increase.

Table 3.1: Non-hybrid 3D-CNN-based architectures for detection of diseased and defected hyperspectral images of crop.

CNN Model	Information	Reference
3D-CNN-based	**Dataset**: 111 hyperspectral images of soybean **Type of Disease:** Charcoal rot **Imaging Device**: Pika XC hyperspectral line imaging scanner **Spectral range:** 400–1000 nm **GPU:** NVIDIA Tesla P40	[75]
3D-CNN based on AlexNet	**Dataset**: 40 hyperspectral images of Grapevine groups **Type of disease:** GVCV **Imaging Device**: SPECIM IQ **Spectral range:** 400–1000 nm **GPU:** -	[79]
HyperSeed	**Dataset**: 200 rice seeds (274,641 pixels) **Type of defect:** Heat stress **Imaging Device**:Micro-Hyperspec Imaging Sensors, Extended VNIR version **Spectral range:** 600-1700 nm **GPU:** -	[83]
3D-CNN-based	**Dataset**: Above 200 strawberry leaves (3,110 ROIs) **Type of disease:** Gray mold **Imaging Device**: Corning microHSI **Spectral range:** 400–1000nm **GPU:** NVIDIA RTX3090 X (24 GB memory)	[77]

Table 3.2: Hybrid 3D-CNN-based architectures for detection of diseased and defected hyperspectral images of crop.

CNN Model	Information	Reference
Hypernet-PRMF	**Dataset**: 16 hyperspectral images of peanut **Type of disease:** Mold **Imaging Device**: SOC710E portable hyperspectral imager **Spectral range:** 400–1000 nm **GPU:** NVIDIA Tesla P100 GPU (12G)	[76]
Deep ResNet 3D-CNN	**Dataset**: 16,346 hyperspectral images of blueberry **Type of defect:** Distinguishing decayed and sound blueberries **Imaging Device**: - **Spectral range:** 400–1000nm **GPU:** -	[71]
SDC-3DCNN	**Dataset**: Rice leaves (Number of taken samples are not determined.) **Type of disease:** BLB **Imaging Device**: Raptor EM285 **Spectral range:** 378.28–1033.05 nm **GPU:** NVIDIA GeForce RTX 2080Ti GPU and the AMD Ryzen 5-1600 Six-Core processor @ 3.20 GHZ CPUs	[78]
2D-3D-CNN (Defect detection module of RT-CBDIA)	**Dataset**: 1026 coffee beans **Type of defect:** Black, insect-damaged, and shell **Imaging Device**: Imec XIMEA snapshot sensor **Spectral range:** 660-980 nm **GPU:** GPU of GEFORCE GTX1660 Ti and a RAM of 16 GB	[7]
ResNet	**Dataset**: 210 lemons **Type of defect:** Bruise **Imaging Device**: It was not determined, however was provisioned by Noor Imen Tajhiz Co.. **Spectral range:** 400-1100 nm **GPU:** Trained on Google Colab	[81]
PLB-2D-3D-A	**Dataset**: 15,360 potato leaves **Type of disease:** PLB **Imaging Device**: Specim IQ **Spectral range:** 400–1000 nm **GPU:** NVIDIA Tesla V100	[80]
Y-Net	**Dataset**: 200 diseased corn leaves (extracted 6,264 regions) **Type of disease:** Brown spot and anthracnose **Imaging Device** : An HSI system provided by Head Wall **Spectral range:** 400–1000 nm **GPU:** RTX 3090 24 Gb	[82]

3.3 Preprocessing of HSI Data

Data preprocessing is a critical step in hyperspectral data analysis, aimed at optimizing the quality and quantity of the data. This step enhances the suitability of the data for downstream tasks such as classification and feature extraction. Preprocessing techniques include patch extraction, radiometric correction and calibration, smoothing, dimension reduction, and background removal. Data augmentation also falls under preprocessing and has the goal of increasing the volume of training data. We demonstrate each technique based on research works conducted in domain of agriculture as follows.

3.3.1 Patch Extraction

Patch extraction is a technique that involves dividing an image into smaller images or patches. In the context of HSI, patch extraction has significant advantages for efficient and targeted analysis of specific regions of interest within an image. By extracting image patches that contain pixels with similar properties, researchers can focus their analysis on areas of the image that are most relevant to diseased or defective areas.

To give a concrete example, Reference [75] utilized patch extraction to analyze hyperspectral images of soybean crops. In their study, they extracted spatial patches of size $64\times64\times240$ from an original image of size $500\times1600\times240$, where the first two dimensions define the spatial resolution of the image and 240 denotes the number of spectral bands. By analyzing the properties of pixels within these patches, they were able to extract features that were more representative of the target disease which ultimately improved the accuracy and efficiency of their analysis. Furthermore, patch extraction helps to expand the number of images when there is a lack of data. It also reduces computational time, as processing hyperspectral images of large sizes can be computationally demanding [80, 82].

3.3.2 Data Augmentation

Imbalanced and scarce data is a common problem in many machine learning applications and HSI is no exception. Imbalanced data, a special case of data scarcity, refers to the situation where the number of samples in each class or category of the data is not evenly distributed. This can lead to a bias towards the overrepresented classes in the analysis results, which is particularly problematic if the minority class is of interest.

To address imbalanced HSI data, one common approach is resampling [79], which involves either oversampling the minority class or undersampling the majority class to balance the class distribution.

Resampling can be done randomly or using more advanced techniques such as Synthetic Minority Oversampling Technique (SMOTE) [87] or Adaptive Synthetic (ADASYN) sampling [88]. Other widespread data augmentation approaches include transformation techniques such as mirroring [76], rotation [7, 76], horizontal and vertical flipping [7, 81], and color jittering [81]. Along with above-mentioned methods, patch extraction (Section 3.3.1) can also be used to address imbalance HSI dataset.

3.3.3 Radiometric Calibration and Correction

Radiometric calibration is an essential step in the accurate operation of hyperspectral cameras. It aims to establish a quantitative relationship between the response of the camera sensor (the radiation sensor) and the actual reflectance (radiation level) of an object in a given environment. The calibration process involves assigning a "true" value for either radiation intensity or reflectance to the digital numbers given by the camera that represent recorded outputs for each pixel and spectral channel. To that end, calibrated reflectance standards are utilized, which usually consist of a matte Lambertian reflecting surface to ensure that reflected light is uniform in all directions [89].

The calibration process is then performed as follows: After selecting a radiation source that emits a known type and amount of radiation, the detector is placed in close proximity to the source to measure the radiation. Next, using calibration factors provided by the manufacturer or based on known mathematical formulas (see, for example, [80]) for the particular detector, the expected response of the detector is determined. The measured response of the detector is then aligned with the expected response based on the known radiation level using calibration factors. This calibration process should be repeated periodically with different radiation sources to ensure the continued accuracy and reliability of the detector's measurements.

Even with calibration performed, hyperspectral cameras are susceptible to radiometric errors that can arise from a variety of sources, including sensor drift, electronic interference, light source, and data transmission and recording issues. For example, one of the common radiometric errors which relates to the camera's sensor is stripping. Each sensor consists of multiple individual detectors that sometimes do not function properly due to being out of calibration. Consider a push broom (along-track) camera where its sensor has multiple detectors aligned in a row. When one of them is calibrated slightly different from the adjacent detector the striping effect can occur. In this case, lines predominantly consisting of varying shades of dark and bright pixels formed. Radiometric correction can register and rectify incorrect pixel brightness. To achieve this, a series of procedures are employed including noise

correction, de-striping, line-dropout correction [90], and black and white image correction [7, 78, 82, 83]. Figure 3.15(b) shows the calibrated hyperspectral image of lettuce using the following computation,

$$\text{Calibrated HSI Data} = \frac{\text{Raw HSI Data} - \text{Dark Ref}}{\text{White Ref} - \text{Dark Ref}}, \tag{3.28}$$

where, Raw HSI Data is the image taken by the hyperspectral camera without modification, Dark Ref is the image captured by closing the camera's lens, and the White Ref is the imaged white reference with even and maximum reflectance across the spectral range.

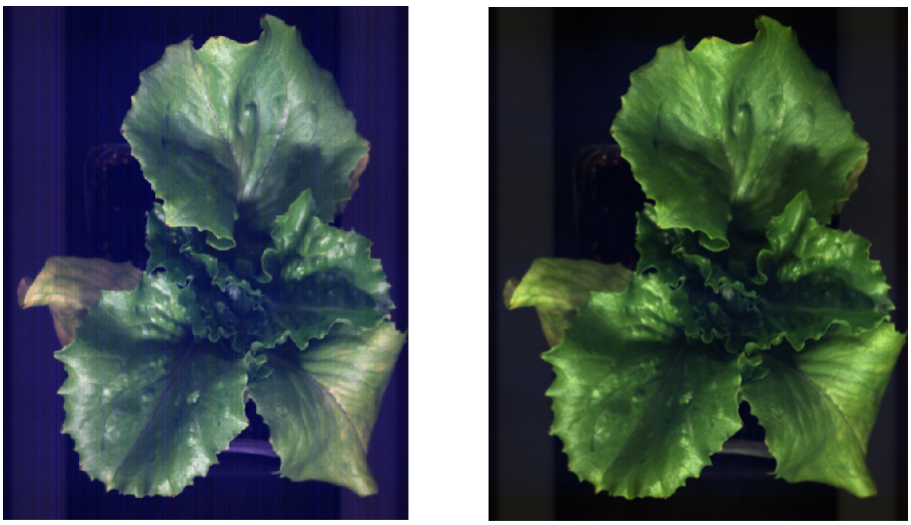


Figure 3.15: Black and white calibration of HSI data. (a) The hyperspectral image of lettuce rendered before black and white image correction; (b) The image shows the rendered hyperspectral image after black and white image correction using the Equation 3.28.

3.3.4 Smoothing

Smoothing, also known as filtering, is a technique in HSI to improve the quality of the data by reducing noise and artifacts and enhancing the signal-to-noise ratio of the data. This technique in HSI can be broadly classified into two categories: spatial smoothing and spectral smoothing.

Spatial smoothing techniques selectively smooth out certain features in an image by applying filters that amplify or attenuate certain spatial frequencies [90]. Common spatial smoothing techniques include Gaussian filtering, mean filtering [91], median filtering, bilateral filtering [92], and anisotropic diffusion filtering [93].

Spectral smoothing techniques operate on the spectral domain of an HSI image by smoothing the intensity values of neighboring spectral bands. As in spatial smoothing the goal is to remove noise [94],

reduce artifacts, and enhance features in the image. Common spectral smoothing techniques include moving average filtering [94], Savitzky-Golay (SG) filtering [95], Fourier filtering, wavelet filtering, and PCA. For example, [78] eliminated the random noise that was present in the spectral data of the different region of interests (ROIs) using SG filtering. The filtering process makes it easier to identify the true signal of the sample and remove interference caused by size and structure differences between the ROIs. Similarly, [77] and [82] used a SG filter to smooth out the spectral data and reduce the effect of noise.

As Reference [94] points out, care must be taken when applying smoothing techniques. The subjective selection of smoothing filters in hyperspectral remote sensing studies can negatively impact the statistical properties of the spectral data, which can, in turn, affect subsequent analyses. To preserve the statistical properties of the HSI data, the selection of smoothing filters should be done through a comparative t-test method that identifies the filter with the least statistical disturbances. By using this approach, it is possible to mitigate the negative effects of smoothing filters and ensure the reliability of subsequent analyses based on statistical class models.

3.3.5 Dimension Reduction

The large number of spectral bands within hyperspectral data often makes it challenging to process and analyze HSI data. Dimensionality reduction techniques retain relevant information, while allowing the model to work on smaller hypercubes downstream. These techniques are split into linear and non-linear ones.

3.3.5.1 Linear Techniques

Linear dimension reduction in HSI refers to a set of statistical and machine learning techniques that aim to find a lower-dimensional linear manifold in the high-dimensional space that captures the essential spectral information of the data [96]. By embedding the data into this lower-dimensional linear space, the dimensionality of the data is reduced while preserving as much of the spectral information as possible. PCA and Random Forest (RF) are two examples of this technique, and demonstrations of both can be seen in [78].

The process of PCA involves first zero-centering the input spectral matrix (i.e., the matrix of the spectral bands) and computing its covariance matrix. Next, the eigenvectors and eigenvalues of the covariance matrix are calculated, and the eigenvalues are sorted in descending order. The top k eigenvalues are

then selected and the corresponding eigenvectors space computed. The k-dimensional data is obtained by projecting the original spectral matrix into the new space using the selected eigenvectors.

On the other hand, an RF evaluates the importance of each wavelength by randomly replacing it and measuring its effect on the accuracy of the thus trained CNN model. To accomplish this, the RF builds many decision trees, and each tree is trained on a different subset of the data. The algorithm calculates the importance score for each wavelength by measuring the change in the prediction error rate before and after randomly replacing the wavelength in the out-of-bag data. The wavelength with the highest importance score is selected, and this process is repeated until the desired number of wavelengths is achieved. The dimensionality of the hyperspectral data is thus reduced, while retaining the most important information for accurate predictions.

Further linear algorithms for dimension reduction of HSI data are described in [97] and encompass Independent Component Analysis (ICA) and PCA-based algorithms like Incremental PCA (IPCA), Sparse PCA (SPCA), and Randomized PCA (RPCA).

3.3.5.2 Non-linear Techniques

Non-linear techniques can handle data with complex and non-linear structures. Reference [98] classified these techniques into Kernel-based and manifold learning [98]. Kernel-based techniques, like Kernel PCA (KPCA), use non-linear mappings to transform data into higher-dimensional feature spaces first on which linear techniques can be applied. Manifold learning algorithms, on the other hand, aim to directly discover the intrinsic non-linear structure of data. See [96] for well-known manifold learning techniques, including Isometric Feature Mapping (Isomap), Locally Linear Embedding (LLE), Local Tangent Space Alignment (LTSA), Diffusion Maps, Sammon's Mapping (SM), and Locality Preserving Projections (LPP).

3.3.6 Background Removal

The data captured in hyperspectral images contains both foreground, called the ROI, and background objects. In scenarios where the target object does not cover the entire scanning area the signals from background objects can interfere with the data analysis, i.e., the background can contain noise that needs to be filtered out [95]. This is especially true when dealing with images that exhibit color gradients. By masking the background from the data, researchers can focus on the spectral signature of the ROI. This

leads to improved target detection in classification task. It also reduces the computational complexity of subsequent processing steps including the training 3D-CNN models [80].

There are various traditional techniques to extract he ROI from the hyperspectral image. These techniques can be classified into several categories based on their underlying principles and are discussed separately below.

3.3.6.1 Spectral Similarity-based Methods

These methods work based on the similarity between the spectra of the pixels within an image. Examples of such methods include Spectral Angle Mapper (SAM) [99] and Spectral Information Divergence (SID) [100]. SAM computes the spectral angle

$$\alpha = \cos^{-1}\left(\frac{\sum_{i=1}^{\lambda} R_i \cdot T_i}{(\sum_{i=1}^{\lambda} R_i^2)^{\frac{1}{2}} \cdot (\sum_{i=1}^{\lambda} T_i^2)^{\frac{1}{2}}}\right) \tag{3.29}$$

between the target reference spectrum R and each pixel spectrum T for all spectral bands λ in the hyperspectral image. This results in a similarity measure that is insensitive to illumination variations [101]. On the other hand, SID works based on the concept of information theory (entropy). It compares the spectral information content of each pixel to a reference spectral information content. In general, SAM is better suited for well-defined spectral variations and low background noise, while SID is more robust to complex background noise and illumination variations.

3.3.6.2 Statistical-based Methods

These methods leverage statistical techniques to identify ROIs that share similar spectral properties and produce a set of uncorrelated components capturing different aspects of spectral variability within the image. For example, Minimum Noise Fraction (MNF) [102] is specifically designed to reduce the impact of noise in the data by separating the noise and signal components of the HSI data. This makes it particularly useful for operating on noisy HSI data but at the expense of more computational time. Another technique, ICA (see Section 3.3.5.1), aims to separate the mixed signals into their independent components, providing a more flexible approach to identify subtle spectral differences between ROIs. Finally, PCA [7] decomposes the original HSI data into orthogonal components that represent the directions of maximum variance in the data. In general, PCA is computationally more efficient than ICA and MNF, as it involves a simpler mathematical transformation that uses standard matrix operations.

3.3.6.3 Spatial-based Methods

These methods exploit the spatial correlation present in the image to differentiate between pixels within and outside the ROI. Typically, these methods apply morphological operations or spatial filtering techniques to extract features such as edges or texture, which are then used to segment the image into ROIs. The Morphological Attribute Profile (MAP) [103] and the Spatial-Spectral Endmember Extraction (SSEE) algorithm [104] are examples of spatial-based methods that use mathematical morphology and spatial filtering, respectively.

SSEE first identifies the endmembers, or pure spectral signatures, present in the HSI data, and then uses a spatial clustering algorithm to group adjacent pixels with similar spectral properties into ROIs. On the other hand, MAP applies a series of morphological opening and closing operations to the image to identify connected regions of pixels with similar morphological attributes, such as size and shape. These connected regions can then be used as ROIs. These methods are computationally efficient and can be useful in scenarios where spectral information alone is not sufficient for accurate ROI extraction, such as in cases of low spectral contrast or high noise levels.

3.3.6.4 Hybrid Methods

These methods combine techniques to improve the accuracy and efficiency of the ROI extraction process. One approach can be combining statistical-based techniques such as PCA, ICA, or MNF with spatial-based techniques such as MAP or SSEE. For example, by combining MAP and PCA we can exploit both the spatial and spectral information [105] in the HSI data for ROI extraction. These hybrid methods can be effective in cases where neither spatial nor spectral methods alone are sufficient for accurate ROI extraction.

3.3.6.5 Machine Learning-based Methods

Machine learning algorithms can also be used to extract the ROI. This process involves training a model using labeled data to identify and extract regions with specific spectral characteristics. Once trained, the model can be used to predict the presence of those characteristics in unlabeled data and extract ROIs. This procedure is computationally efficient and allows for the extraction of subtle and complex patterns that may not be easily identifiable through traditional methods. Some instances of such techniques are Support Vector Machines (SVM) [106], RF [107], and CNN [108, 109].

3.3.6.6 Software-assisted Manual Annotation

Manual definition of the ROI can be assisted by software specifically built to handle HSI data. Amongst these are for example ENVI [110] and Spectronon [111] as commercial products, as well as the MATLAB Hyperspectral toolbox. Examples for open-source and free software are SeaDAS [112], the Orfeo ToolBox [113], and RSGISLib [114]. These packages provide HSI data analysis for a wide range of tasks discussed before. An example of software-assisted annotation is the work of Reference [115] in which the authors generated an ROI using ENVI by manually selecting the tissues or areas of interest in the false color images of HSI data.

Researchers can utilize such software in addition to the above discussed algorithms. For example, Reference [83] developed an open-source software which is specifically designed for analyzing HSI data of seeds. It is able to remove the background of the HSI data and produces a binary mask using a user-defined minimum and maximum intensity threshold along with a component-searching algorithm. The results are an accurate segmentation of the seeds even when they overlap.

3.4 Band and Feature Selection

The selection of spectral bands in HSI is another crucial preprocessing step for classification tasks. It entails the identification of a subset of the most discriminative spectral bands from all available bands. This process is instrumental in reducing data dimensionality by eliminating redundant bands, thereby significantly reducing the computational overheads of downsteam tasks. Furthermore, the careful selection of spectral bands can also reduce the effects of noise in the data. In general, based on the survey of Reference [116], band selection mechanisms can be categorized into six groups as follows (another classification of methods is presented in [117]).

3.4.1 Ranking-based selection

Ranking-based band selection methods evaluate the significance of each spectral band based on a predetermined criterion and choose the most important bands in a sorted order. These methods can be categorized into two types: supervised and unsupervised. In supervised ranking-based methods, labeled training samples are utilized to determine the importance of each spectral band, while in unsupervised ranking-based methods, statistical properties of the data are used for the same purpose.

One example of such method is spectral differentiation. Reference [80] employed first and second order differentiation to decrease the computational complexity of HSI data containing 204 bands. The first

derivative can effectively pinpoint areas in the spectrum where the rate of change is highest. This indicates the presence of sharp spectral features such as absorption or emission lines. Moreover, it allows for the selection of bands that capture these features and thus, provide critical information for classification or detection tasks. The second derivative is useful for identifying regions of the spectrum where the rate of change of the first derivative is highest, signifying the presence of spectral curvature. The selection of bands that capture the shape of the spectral signature, based on this information, can enhance the differentiating power in classification or detection tasks. Likewise, Reference [77] achieved an increase in accuracy using spectral differentiation and expansion of the input in the vertical direction of the raw data. This technique was applied in addition to SG smoothing (see Section 3.3.4) to further improve data quality.

3.4.2 Searching-based Selection

Searching-based band selection methods involve the creation of a criterion function such as Euclidean distance and Bhattacharyya distance [118] to evaluate the performance of each spectral band based on a specified optimization objective. The first step involves creating an initial subset of bands, followed by an assessment of the criterion function for the subset. The next step is applying a searching strategy to identify the best subset of bands that maximizes the criterion function, and evaluating the selected subset based on data classification performance. This iterative process continues until the desired level of performance is reached. Searching-based methods largely depend on the quality of the criterion function and the optimization strategy employed. Incremental searching [119, 120], updated searching [121, 122], and eliminating searching [123] are among the commonly utilized strategies.

3.4.3 Clustering-based Selection

Clustering-based methods for hyperspectral band selection group bands into clusters and select representative bands from each cluster to create a final subset. These algorithms can be unsupervised [124, 125], supervised [126] or semisupervised [127, 128]. The selection of representative bands is typically performed using information measurements, such as mutual information or Kullback–Leibler divergence. Commonly used clustering techniques are K-means, affinity propagation, and graph clustering. K-means selects the best cluster centers that minimize the sum of distances to a set of putative center candidates. Affinity propagation selects exemplars by considering the correlation or similarity among bands and the discriminative capability of each band.

3.4.4 Sparsity-based Selection

Sparsity-based techniques for band selection rely on sparse representation or regression to identify representative bands. The most common of these methods are discussed separately below.

3.4.4.1 Sparse Nonnegative Matrix Factorization-based Methods

These methods [129] break down the hypercube into a set of building blocks, which are both nonnegative and sparsely encoded. This promotes a feature extraction process that combines these building blocks to create a parts-based representation of the original data. The goal of this method is to identify the most informative bands of the HSI data matrix by optimizing an objective function that includes sparsity constraints.

3.4.4.2 Sparse Representation-based Methods

Sparse representation-based methods [130, 131] use pre-defined or learned dictionaries to select informative bands of the HSI data matrix based on their sparse coefficients. These methods rank the bands according to the frequency of their occurrence in the sparse coefficient histograms. In some cases, sparse representation-based methods can also be designed to solve multiple tasks simultaneously, and an immune clonal strategy can be used to search for the best combinations of informative bands.

3.4.4.3 Sparse Regression-based Methods

Sparse regression-based techniques [132, 133] transform the band selection problem into a regression problem and estimate the most representative bands by solving a sparse regression problem. These methods can also include sparsity constraints to encourage the selection of only the most informative bands for the regression model.

3.4.5 Embedding-learning-based Selection

Embedding-learning based methods aim to learn a low-dimensional representation of the spectral data, also known as an embedding, that captures the most salient features of the data. There are several types of embedding learning-based methods that can be used for band selection, including autoencoders [134], Deep Neural Networks (DNN) [135], and CNNs [136]. Autoencoders learn a compact representation of the input data by training a neural network to encode the input into a lower-dimensional space and

then decode it back into the original space. DNN, on the other hand, consists of multiple layers of interconnected neurons. This architecture enables the network to learn complex patterns and features in a hierarchical manner, empowering it to extract high-level representations from the input data. Meanwhile, CNNs can learn spatially invariant features from image data.

These techniques aim to learn a set of parameters that minimize an objective function that measures the model's performance on a particular task, such as classification or target detection. Band selection is integrated into the optimization process by constraining the learning algorithm to focus on a subset of the available bands, or by assigning weights to each band that reflect its relevance to the task at hand. The resulting model can then be utilized to predict the class label of new samples or to detect the presence of specific targets in the image. For example, Reference [7] employed a DNN based binary classification to identify the foreground and background regions of the HSI data. Subsequently, connected component labeling algorithms and edge contours were used to isolate the ROI from the image for further analysis.

Moreover, Reference [82] implemented a CNN-based band selection module that works based on a group convolution technique, which involves applying a 1×1 one-dimensional convolution (equivalent to a scalar multiplication) to each band of the input hyperspectral image independently. This technique helps to overcome the problem of mutual interference between different channels. The weights of the convolutional kernel are updated in the early stage of the network training using a loss function and an auxiliary classifier. The weights represent the importance of each band, with a higher absolute value of weight indicating greater importance of the corresponding band.

3.4.6 Hybrid-scheme-based Selection

Hybrid-scheme based methods involve combining multiple band selection techniques to select the most appropriate bands. A popular combination is clustering and ranking [137, 138], where clustering is used to group bands and ranking is used to select the most important bands within each cluster. Other hybrid methods combine clustering with searching or combine ranking with searching to further optimize band selection.

3.5 Visualization Techniques for HSI Classification Decisions

Visualization techniques can be employed to observe the contribution of pixels in the classification decision. These techniques allow us to identify the pixel locations associated with the most important spectral bands that play a crucial role in the final classification results.

Saliency maps [139] are one of the essential and traditional visualization tools to identify the most sensitive regions (crucial pixels) in an image with respect to a model's predictions. This technique works by computing the gradient of the output class score with respect to the input image. This gradient represents how much each pixel in the input image contributes to the final classification decision. Next, the absolute values of these gradients are summed across the channels to obtain a saliency map, which highlights the most salient regions of the input image for the predicted class.

For example, Reference [75] discovered that saliency maps can help to locate the most sensitive pixel locations in infected crop images, which are often the severely infected areas. Conversely, both healthy and infected crop images had saliency map gradients that were primarily focused around the mid-region of the crop stem, highlighting the stem's importance in crop classification. Moreover, [78] observed that the significant wavelengths extracted by RF from raw HSI data overlaps the saliency-sensitive wavelengths. More importantly, saliency maps can determine significant wavelengths for classification which are not extracted by RF.

Another visual explanation technique for CNN decision is the Gradient-weighted Class Activation Mapping (Grad-CAM) [140]. It was introduced as an improvement over CAM [141]. CAM involves modifying a pre-existing CNN model by replacing the final FC layer with a global average pooling layer, which retains essential channel information while reducing the spatial dimensions. This modification enables the utilization of feature maps from the preceding layer. By applying learned weights to these feature maps through global average pooling, CAM generates a map that highlights the crucial regions associated with the predicted category. However, CAM provides a coarse localization of the important regions within an image. It highlights the regions that contribute most to the predicted class, but it does not provide precise boundaries of those regions. To address this limitation, Grad-CAM was introduced as an extension to CAM.

Grad-CAM enhances the CAM approach by incorporating gradient information. Similar to CAM, Grad-CAM also utilizes a global average pooling layer after the last convolutional layer to obtain importance scores for each channel in the feature maps. However, instead of learning separate linear models for each class, Grad-CAM calculates the gradients of the predicted class score with respect to the feature maps. These gradients are used to weigh the feature maps and enables Grad-CAM to identify more intricate features that play a significant role in the classification decision.

In addition, there are further statistical techniques that help in analyzing the importance of individual features (spectral bands) of HSI data in a classification. Light Gradient Boosting Machine (LightGBM) [83, 142] is one of such methods that uses decision trees in calculating feature importance. It builds

decision trees in a leaf-wise manner, meaning that the algorithm grows the tree by adding new leaves one at a time which is computationally faster compared to level-wise approach by selecting the best split based on the maximum reduction in loss function for all leaves in the tree. Therefore, LightGBM calculates the importance of each feature by evaluating how much it contributes to the reduction in loss function across all trees. The feature importance score is calculated by summing up the number of times a feature is used to split the data across all trees, weighted by the improvement in accuracy achieved by each split. Features that are used more frequently and result in larger improvements in accuracy are assigned higher importance scores.

3.6 Discussion and Conclusion

In Section 3.2, we conducted a comprehensive review of 3D-CNN-based models applied in the domain of agriculture using non-UAV-based HSI data. Our analysis delved into diseased and defective crops, focusing on the structures and efficiencies of the models, the quantity of datasets utilized, and the necessary preprocessing steps (see Section 3.3). The review indicates the advantages of 3D-CNNs in capturing spatial-spectral information within hyperspectral data, enabling them to outperform 1D- and 2D-CNN in hyperspectral image classification. With the aim of assisting computer vision experts and agriculture-domain researchers in tackling HSI classification tasks for crops experiencing stress, this comprehensive review provides valuable insights and guidance.

In general, HSI holds great potential for detecting subtle changes in crop growth and development, making it a promising technique for diagnosing crop diseases and defects. Despite this potential, our study indicates that there is still limited research conducted that use 3D-CNNs in this context. Furthermore, the studies that are performed are often very application-specific and it is unknown how well the performed methods and models generalize to a broader range of applications. We identify three major challenges that must be overcome to achieve a broader adoption of HSI in general and the usage of 3D-CNN for HSI classification problems: limited availability of hyperspectral data, computational complexity of 3D-CNN models, and the costs of HSI hardware. In the following we address each of these challenges in more detail and offer ideas on how to overcome or avoid them.

Limited availability of varied hyperspectral data on diseased and defective crops presents a significant challenge for researchers, farmers, and stakeholders in the agriculture industry who rely on data to make informed decisions. The lack of data in this area of the research hampers efforts to understand the extent of the problem and develop effective solutions. It makes it difficult to track the progress and success of any initiatives aimed at improving crop health and reducing the prevalence of disease

and defects. To address this issue, there is a need for increased investment in data collection and data sharing, as it has been done in the past, for example, for RGB-data (e.g., [143, 144]). Large-scale HSI data collection and publicly available datasets will allow research groups to develop the next generation of models, even if they have no access to the otherwise required hardware and plant material. Even with small datasets there are also techniques to models beyond the specific application case they had been trained on. In recent years, transfer learning and active learning techniques have been increasingly used together to tackle the challenges posed by limited data. By leveraging knowledge from pre-existing models, transfer learning can enhance the accuracy of models trained on limited HSI data. On the other hand, active learning involves selecting and annotating informative samples to improve the effectiveness of models trained on small HSI datasets. By combining the two techniques, transfer learning and active learning can address the bottleneck of limited HSI data and enable the development of robust 3D-CNN models capable of accurately classifying HSI data.

The computational complexity of 3D-CNN models is a barrier for their deployment in the field, for example, in edge computing devices or even as part of an embedded system in UAVs or agricultural equipment. Real-time diagnosis could significantly enable farmers to quickly detect and respond to disease outbreaks, which can help to prevent the spread of disease and reduce crop losses. This requires, however, researchers to explore ways to optimize their models for speed and efficiency, particularly their memory-footprint. This can be achieved through advancing the capabilities of single board computers, particularly their GPU and AI accelerators, on the one side, as well as developing lightweight models on the other side, for example, by reduction of used bands or grouping of bands into indices. Identifying the spectral bands that are most informative for detecting a varied range of diseases and defects would play an important role in reducing the model's training computational time, decreasing the number of parameters, achieving higher accuracy and generating a more lightweight model.

Despite the valuable insights hyperspectral imaging provides regarding the condition and health of crops, deploying this technology is relatively costly compared to RGB and multispectral imaging technologies. The higher expense is primarily attributed to the increased processing power required to analyze the HSI data and the extensive range of spectrum offered by hyperspectral cameras.

One approach to consider for cost reduction is the limitation of the number of bands in hyperspectral cameras, as not all spectral bands may be equally crucial for disease and defect detection. Therefore, instead of having an imaging device that supports full range of wavelengths from the visible to the infrared spectrum, we can deploy imaging systems that work with essential wavelengths rather than hundreds of spectral bands. In this respect, some companies already provide the facility to design customized multispectral systems, which work with bands that had been identified to be the most discriminative

(see [145] for a 3D-CNN model training advantage of this approach). Therefore, developing methods for identifying the most informative spectral bands would also be cost beneficial. Furthermore, by leveraging Machine Learning as a Service (MLaaS) platforms [146], the need for required infrastructure to process HSI data using 3D-CNN can be eliminated. Additionally, leading MLaaS providers can integrate pre-built 3D-CNN models into their offerings for training HSI data, thereby reducing the time and effort required for model development, especially for individuals in the agriculture domain who may have limited machine learning expertise.

In summary, the application of 3D-CNNs with hyperspectral data for disease and defective crop detection is a promising research area with several open research questions. Collecting and sharing HSI data on scale, identifying informative spectral bands, developing transfer and active learning techniques, and implementing light-weight architectures are some of the key research areas that can be considered for future work. The advancement of this research will lead to more accurate and efficient disease and defective crop detection, which can have significant impacts on the agricultural industry.

Chapter 4

Experimental Framework and Results

This chapter is organized as follows: In Section 4.1, we detail the planting materials, the growing conditions, and the methods used for applying nitrogen stress. Section 4.2 provides an in-depth look at the hyperspectral camera's features, along with the setup necessary for data acquisition, including defining aspect ratio and focal length. The preprocessing steps to prepare the data for neural network analysis are outlined in Section 4.3. Section 4.4 offers an overview of the hyperparameters we optimized in the design of the 3D-CNN model and outlines our rationale for choosing Bayesian optimization over other techniques. In Section 4.5, we delve into the specifics of the model designs and discuss the results obtained. Finally, Section 4.6 analyzes the importance of different channels across the visible and NIR spectrum.

4.1 Planting Materials and Procedure

In this study, Buttercrunch lettuce was used for nitrogen stress experiment. The lettuce seeds were produced by A. E. McKenzie Co. ULC. On July 5, 2022, 45 pots of lettuce were prepared by sowing 3 seeds of lettuce in each pot containing coconut coir. Coconut coir is a growing medium and soil improver, a 100% natural and renewable soil amendment harvested from coconut husks to help plants develop stronger root systems and improve nutrient and moisture retention in the soil. Moreover, coir works as a soil conditioner and is naturally disease and weed free. It also improves the flow of moisture. By planting few seeds (3 seeds) in each pot, it was assured that there would be at least one lettuce seed that could germinate. A few days after sowing the seeds, only one was kept in each pot. All the planting procedures and experiments occurred in a growth chamber at The University of Winnipeg under temperature of 21 ℃ with 16 hours of light and 8 hours of darkness. The average humidity for

the course of experiment was 50%. The lighting sources were series of LED grow lights (HEDERA APEX) with 6 individually adjustable color channels. In this study, 15 lettuces are grown per group of N^-, $N^{+1/2}$, and N^+. The watering took place every other day under the supervision of a greenhouse technician. The application of nitrogen stress treatment started on July 15, 2022, by adding 50 ml of fertilizer solution to each pot and continued till August 12, 2022, with 7-day intervals. The fertilizer ingredients were provided in Table 4.1 for each sub-experiment.

Table 4.1: Ingredients of solutions used for applying nitrogen stress.

Ingredients	No Nitrogen (mg/L)	1/2 Nitrates (mg/L)	Nitrates (mg/L)
Ammonium Nitrate	None	825	1650
Boric Acid	6.2	6.2	6.2
Calcium Chloride, Anhydrous	332.2	332.2	332.2
Cobalt Chlorid·$6H_2O$	0.025	0.025	0.025
Cupric Sulfate·$5H_2O$	0.025	0.025	0.025
Na2EDTA·$2H_2O$	37.26	37.26	37.26
Ferrous Sulfate·$7H_2O$	27.8	27.8	27.8
Magnesium Sulfate, Anhydrous	180.7	180.7	180.7
Manganese Sulfate·H_2O	16.9	16.9	16.9
Molybdic Acid (Sodium Salt)·$2H_2O$	0.25	0.25	0.25
Potassium Iodide	0.83	0.83	0.83
Potassium Phosphate, Monobasic	170	170	170
Zinc Sulfate·$7H_2O$	8.6	8.6	8.6

4.2 Data Acquisition

To generate the hyperspectral dataset of lettuces under nitrogen stress, a SPECIM FX10 hyperspectral imager was used to perform the hyperspectral imaging. The FX10 camera series is built around the monochrome complementary metal oxide semiconductor (CMOS) image sensor [147] that provides spatial resolution of 1024 pixels at a wide range of spectral sensitivity (400 to 1000 nm). It is aimed at standard applications in industrial and laboratory image processing. The SPECIM FX10 collects hyperspectral data through single fore optics. It works in a push-broom mode, which means the sensor captures one line of the scene at a time when considering a standard 2D image. Each line corresponds to a narrow strip of the target area. This camera captures the X dimension in each frame, while the Y dimension is formed by moving the target (see Figure 4.1). To achieve square pixels in both the X and Y dimensions, adjusting the speed of the target and the camera's frame rate is necessary. Additionally, the aspect ratio can be manipulated by imaging geometric shapes, such as squares or circles, and then controlling the final shape's appearance in the captured image. We used the camera's round lid, as

suggested by the manufacturer, as the target to configure the speed and frame rate. This approach provides precise control over the image's spatial characteristics. The exposure time, frames per second (FPS), and scan speed are set according to the manufacturer's manual. Figure 4.1 shows images of the camera's black lid under different configurations of FPS and scan speed, using a fixed exposure time. These images demonstrate how varying these parameters can influence the characteristics of the captured hyperspectral data.

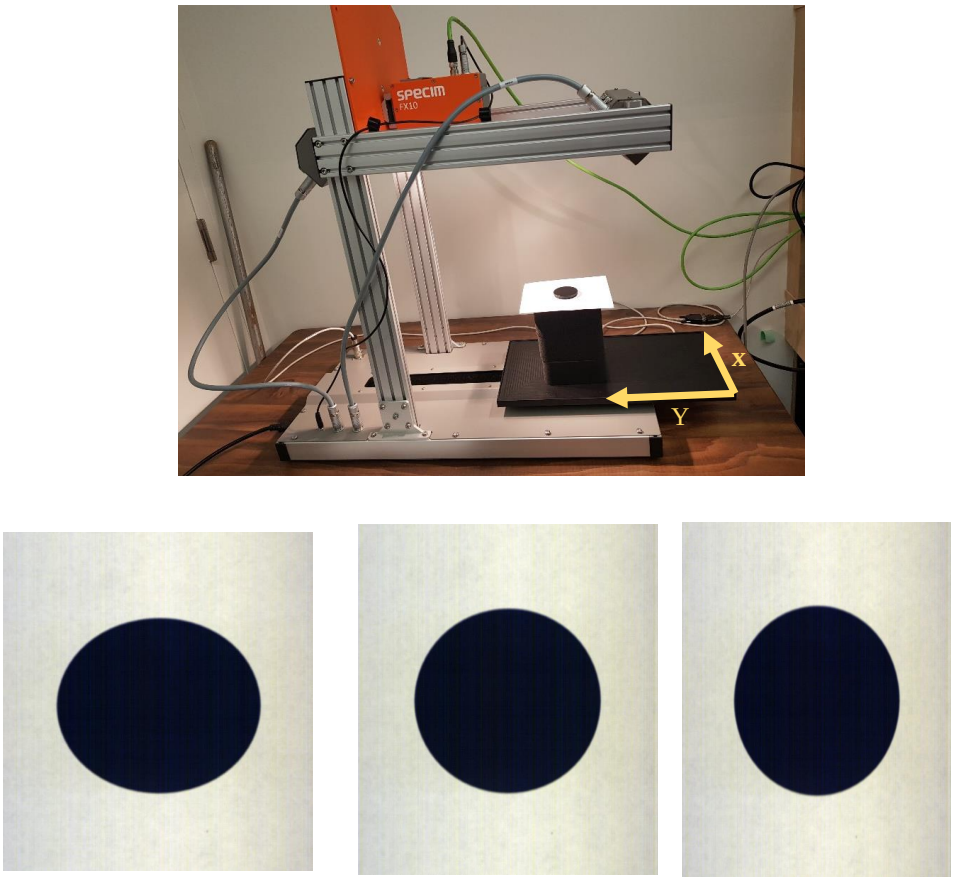


Figure 4.1: Defining the aspect ratio. Round lid shape was captured under different configurations of FPS and scan speed using fixed exposure time. The camera captures the X dimension in each frame and forms the Y dimension by moving the target, allowing for the adjustment of target movement speed and frame rate to attain square pixels in both dimensions.

To define the focal length, we created a cardboard stair-shaped structure with three levels, as depicted in Figure 4.2. Focal length is the distance between a camera's sensor and the optical center of its lens.

A longer focal length narrows the perspective and brings subjects closer, often blurring the background, while a shorter focal length enhances the perspective and typically yields a clearer, more focused image. To balance between a blurred background and image clarity, we aligned the top of the plant with level 1 and adjusted the focal length to achieve optimal clarity at both levels 1 and 2.

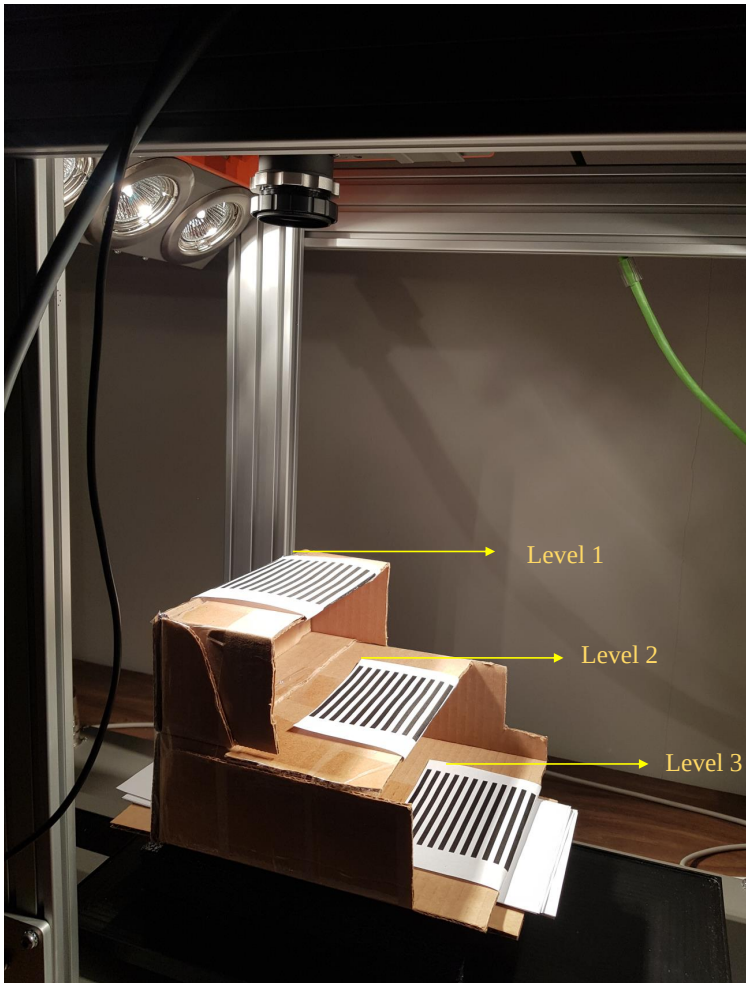


Figure 4.2: Optimizing focal length using a stair-shaped structure: aligning the plant with level 1 for balanced background blur and image clarity at levels 1 and 2.

The strategy used to produce a collection of HSI data was to image the plants in short intervals. In this study, on average the imaging was performed almost every 3 days (on July 18, July 21, July 25, July 28, August 1, August 5, August 8, August 11, and August 14). Each pot of lettuce is located on a tray table (see Figure 4.3) and the camera is perpendicular to the tray table surface. We made sure to only water the plants after they were imaged to maintain consistent imaging conditions. To minimize surrounding light effects, we kept the lights off during the imaging process. The illuminators are placed 45 degree

angle lighting based on provided assembly instruction by the manufacturer. In total 404 hyperspectral images were collected for post experiment duration consisting of 135, 134, and 135 hyperspectral images per N^-, $N^{+1/2}$, and N^+ class, respectively. Figure 4.4 displays images of lettuce under standard nitrogen treatment across blue, green, and red bands, along with the corresponding spectrum for a selected pixel on both the soil and the leaf, within the 400 to 1000 nm range.

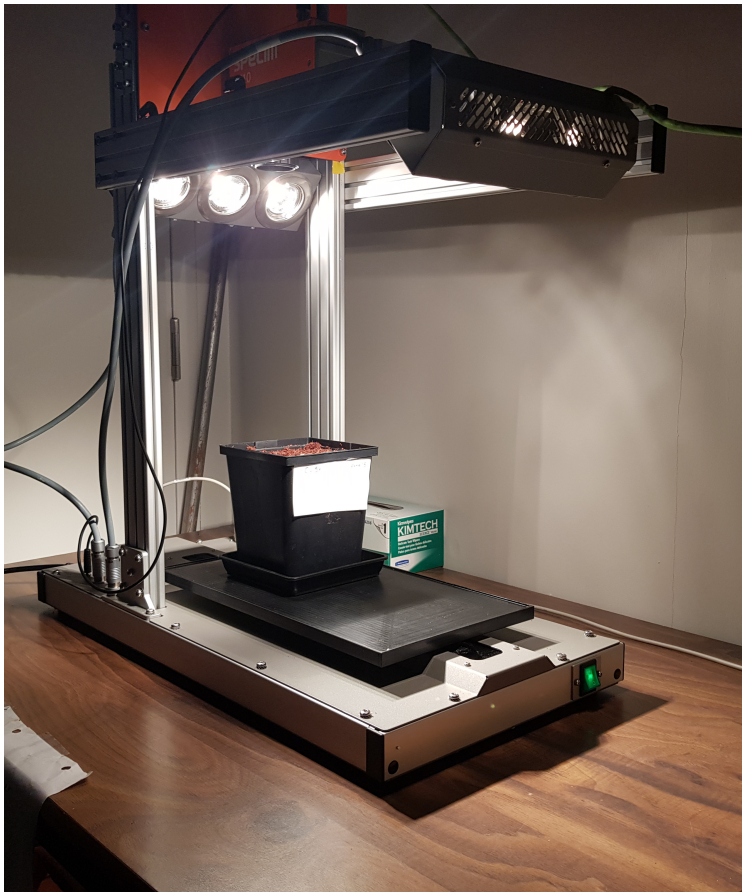
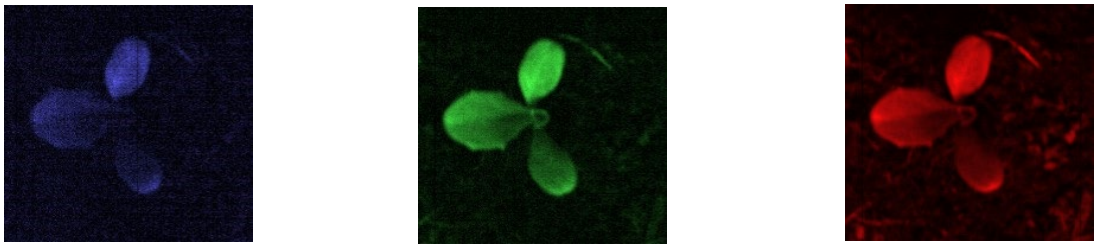
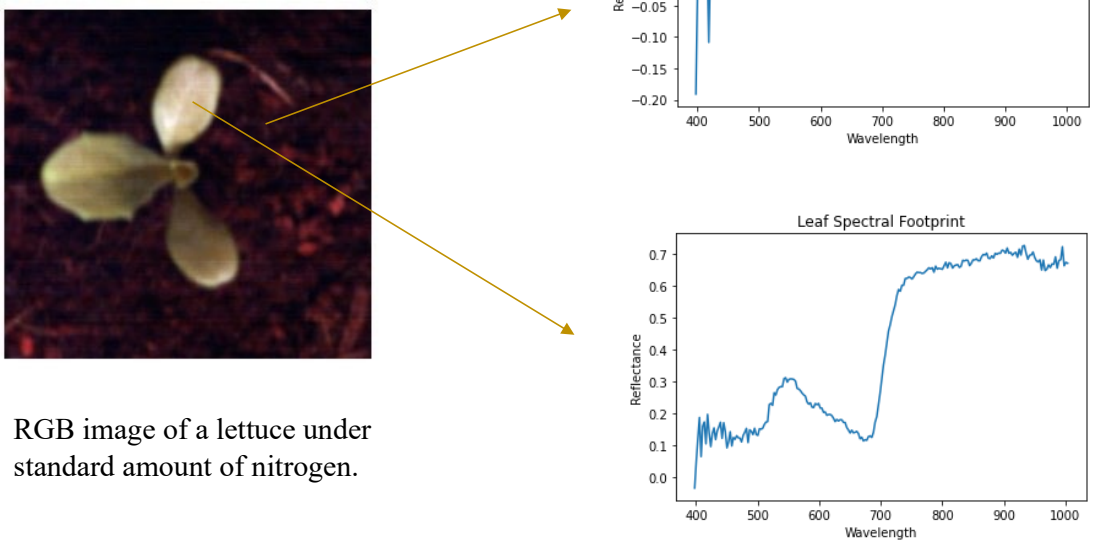


Figure 4.3: SPECIM FX10 camera. The Specim FX10 hyperspectral camera operates in the 400 to 1000 nm range and features a spatial resolution of 1024 pixels and a spectral resolution of 5.5 nm, functioning in push-broom mode.



Band 27: 465.98 nm Band 46: 516.29 nm Band 96: 650.29 nm

(a)



RGB image of a lettuce under standard amount of nitrogen.

(b)

Figure 4.4: a) Images of a lettuce plant under a standard amount of nitrogen treatment across blue, green, and red bands. b)The image displays lettuce under standard nitrogen treatment and the corresponding spectrum for a randomly selected pixel on both soil and leaf, ranging from 400 to 1000 nm.

Moreover, Figure 4.5 displays the RGB image of the same plant across all bands within the blue, green, and red ranges, where each single image per band is averaged across the corresponding band.

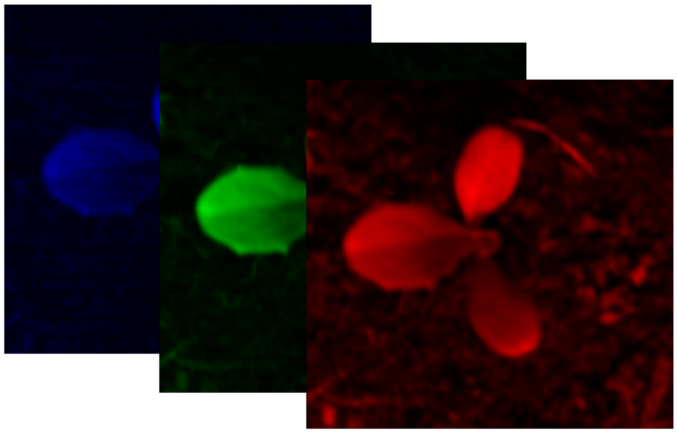


Figure 4.5: RGB visualization of lettuce plant: averaged images across blue (450-495 nm), green (495-570 nm), and red (620-750 nm) bands.

4.3 Data Preprocessing

Data preprocessing is a critical step in hyperspectral data analysis, aimed at optimizing the quality of the data. This step enhances the suitability of the data for downstream tasks such as classification and feature extraction. In Sections 4.3.1 and 4.3.2, the preprocessing steps are demonstrated. Our focus on detecting early and subtle indications of Nitrogen deficiency led us to concentrate our analysis specifically on observations made on July 18 and 21. In this regard we have to note that, during these early stages of growth, identifying visual differences presents a well-known challenge. Figure 4.6 depicts the summary of the preprocessing conducted in this research. As the aim of this research is inspecting all the 224 spectral bands, we did not perform the feature and band selection steps explained in Section 3.4.

4.3.1 Black and White Calibration

The imaged hyperspectral data is raw uncorrected data, it can be inspected as images, but the spectra do not look as real NIR spectra until the reflectance or absorbance values are calculated [148]. In order to view images with colors close to what is observed, it is necessary to calibrate them with a black reference (automatic shutter closing) and white reference (with the white reference positioned on the

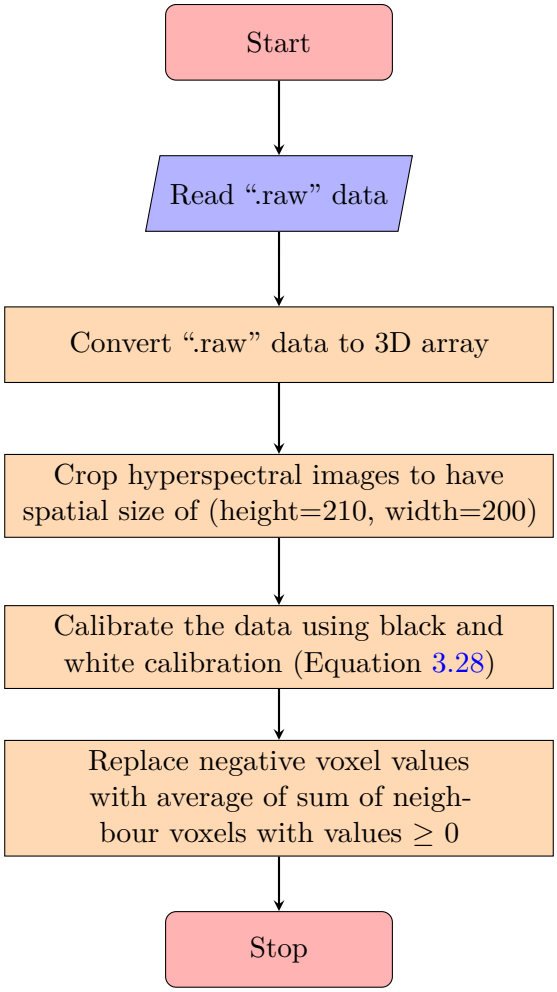
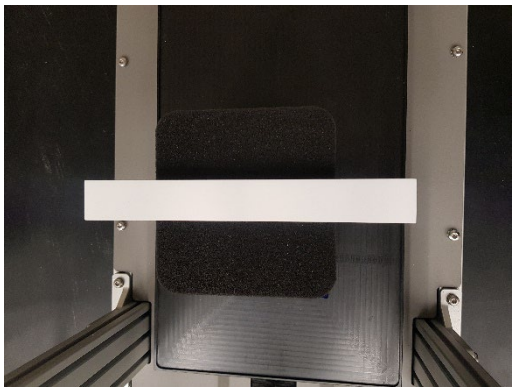


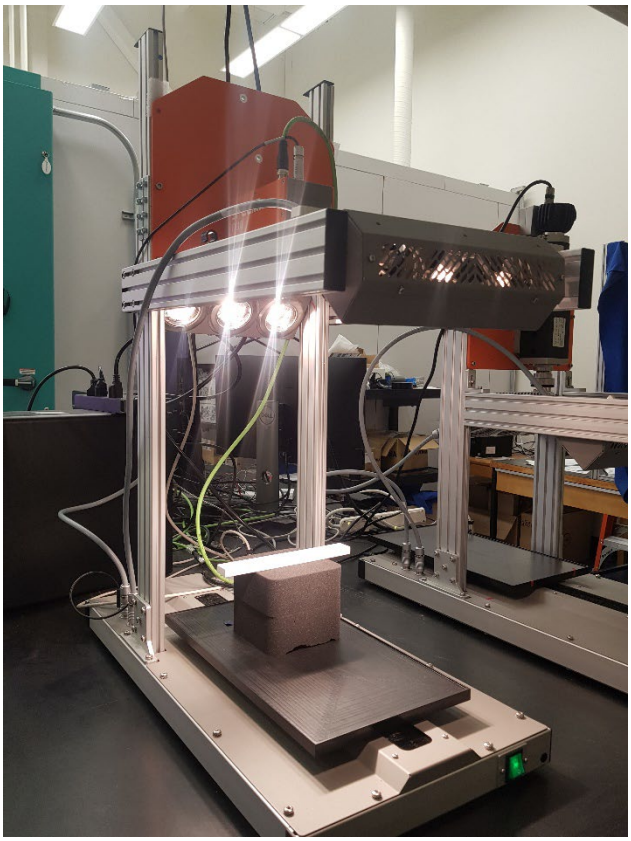
Figure 4.6: This flowchart shows the preprocessing steps of HSI lettuce dataset.

tray table) through the procedure [149] discussed in Section 3.3.3. Figure 4.7 shows the black and white references used to calibrate each hyperspectral image.

(a)



(b)



(c)

Figure 4.7: Black and white calibration. (a) shows the SPECIM FX10's shutter. (b) and (c) show the white reference.

After the calibration process, certain voxel values may be negative. These negative values are not meaningful in the context of reflectance data since reflectance typically varies between 0 and 1, corresponding to 0% to 100% reflectance. To address this issue, each negative voxel value is adjusted based on its neighboring voxel values. Specifically, any voxel with a negative value, denoted by v_{xyz}, is replaced with the average of its immediate non-negative neighbors in the 3D space. For a voxel located at position (x, y, z) in a 3D space, its immediate neighbors are voxels at positions $(x+1, y, z)$, $(x-1, y, z)$, $(x, y+1, z)$, $(x, y-1, z)$, $(x, y, z+1)$, and $(x, y, z-1)$.

Given this configuration, the corrected value for the voxel, v'_{xyz}, can be computed as:

$$v'_{xyz} = \begin{cases} \frac{1}{N} \sum_{i=x-1}^{x+1} \sum_{j=y-1}^{y+1} \sum_{k=z-1}^{z+1} v_{ijk} & \text{if } v_{xyz} < 0 \text{ and } v_{ijk} \geq 0 \\ v_{xyz} & \text{otherwise.} \end{cases} \tag{4.1}$$

Here, x, y, and z represent the spatial coordinates of a voxel in the 3D space, and N is the number of non-negative neighboring voxels around v_{xyz}.

This method ensures that negative voxel values are adjusted in a way that is consistent with their local surroundings in the 3D space. An illustrative depiction of this voxel adjustment process can be seen in Figure 4.8, where the negative voxels are shown in white and their neighboring voxels in different colors.

4.3.2 Data Normalization

Normalization seeks to adjust data such that each input, in this case a voxel in 3D data, adheres to a consistent range. In this study we used min-max normalization. It adjusts the scale of the data but maintains the relative distances between values. This means that while the scale of the data changes, the proportional differences between any two values remain the same. For instance, if one voxel's value was twice as large as another before normalization, this relationship will still hold true after normalization. This is important because it ensures that the inherent structure and relationships within the data are not distorted during the scaling process.

Let's consider a 3D dataset $\mathbf{V}$ containing voxels v_{ijk} where i, j, and k are spatial indices spanning the width, height, and depth of the voxel, respectively. To implement min-max normalization, we first determine the minimum ($\min_V$) and maximum ($\max_V$) values in the dataset. Each voxel value is then transformed using the following formula:

$$z_{ijk} = \frac{v_{ijk} - \min_V}{\max_V - \min_V}.$$

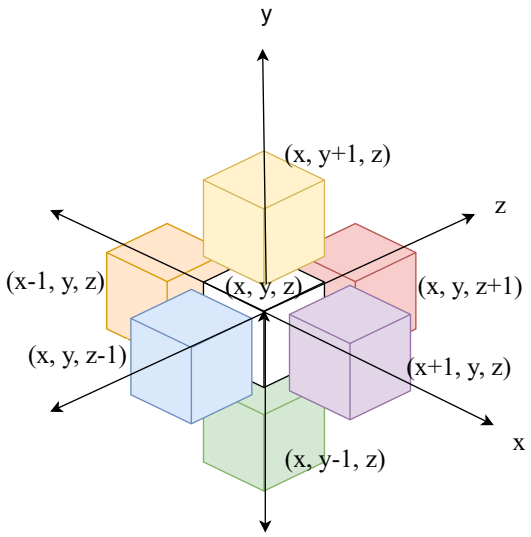


Figure 4.8: Neighbour voxels.

This formula rescales every voxel value to lie within the [0, 1] range, mapping the minimum value of the dataset to 0 and the maximum value to 1. It preserves the relationships among the original data values and it ensures: 1) each voxel's value is scaled to the [0, 1] range which promotes uniformity, 2) all voxel values now span a consistent range, mitigating the disproportionate impact of higher magnitude values.

4.4 Hyperparameter Optimization for 3D-CNN Architectures

Several hyperparameters need to be define to design the architecture of a CNN. Hyperparameters values determine the structure and behavior of the training process. Unlike the internal parameters (often referred to simply as "parameters") of a model, which are learned from data during the training process, hyperparameters are not directly learned through the training process. Table 4.2 provides list of hyperparameters we considered within this research and the details are as follows:

- *Number of convolutional layers* dictates the depth of the CNN and thus its capacity to learn hierarchical features from the input data. Deciding on the optimal number involves a balance between the model's capacity and its computational efficiency, as well as considerations about overfitting and training stability.
- *Number of filters* in a 3D convolutional layer dictates how many distinct spatial-spectral features the layer can learn from its input volume. Setting this hyperparameter involves considering both

Table 4.2: List of hyperparameters.

Category	**Hyperparameter**
Convolutional layer	Number of convolutional layers
	Number of filters
	Filter size
	Stride size
Pooling layer	Pooling size
	Stride size
Dense layer	Number of dense layers
	Number of neurons (units)
Dropout layer	Dropout rate
General hyperparameters	Batch size
	Number of Epochs
	Learning rate

the representational capacity desired and the computational costs associated with the increased complexity of 3D data.

- *Filter size* in 3D convolutional layers determines the spatial and depth extent to which the network processes the input volume at each layer. Properly tuning this hyperparameter is crucial for the balance between capturing relevant features and managing computational complexity.

- *Convolution stride size* determines the step size at which the kernel is applied to the input data. The stride affects the size of the output feature map. A larger stride will produce a smaller output dimension, as the kernel covers the input space more quickly and skips more voxels. Conversely, a smaller stride will produce a larger output dimension, as the kernel moves more slowly across the input.

- *Pooling layer size* in 3D convolution determines the volume's extent over which pooling, like max-pooling or average pooling, occurs.

- *Pooling stride size* determines how far the pooling window moves after each operation. The stride in pooling layers significantly affects the size of the output. A larger stride results in a more aggressive downsampling, producing a smaller output feature map.

- *Number of fully connected layers* affects the model's capacity to learn and represent data patterns effectively in order to make the final predictions. The complexity of the problem can guide the number of fully connected layers. More complex problems might benefit from more layers to capture high-level abstractions. While, this can lead to overfitting with smaller datasets.

- *Number of neurons* determines dense layer's capacity to learn complex data patterns and representations. More neurons enhance the layer's feature recognition ability but may lead to overfitting and higher computational demands.

- *Dropout rate* controls the percentage of neurons randomly deactivated during training. It serves as a regularization technique to prevent overfitting by reducing the reliance on specific neurons which promotes better generalization of the model.

- *Batch size* determines how many samples are processed in parallel in one iteration before updating the weights of the neural network. It affects resource requirements of the training process and speed. The acceptable range for the batch size is between 1 (using each instance individually) and the size of the full training dataset.

- *Learning rate* determines the trajectory and convergence of the optimization process. While a correct setting of this hyperparameter can lead to rapid and stable convergence, an ill-suited choice can result in slow training or even divergence.

- *Number of Epochs* determines how many iterations the network goes through during training. Increasing the number of epochs allows the model to learn more features from the training data [62]. To determine whether the network requires more or fewer training epochs, it is important to monitor the training and validation error values. Validation error provides an unbiased estimate of the test error, reflecting how accurately the model is expected to perform on unseen data. The general principle is to continue training as long as the error value is decreasing. However, it is essential to be cautious and observe the behavior of the errors over time. If the validation error starts to oscillate or increase while the training error is still improving, it suggests that the network is overfitting the training data and failing to generalize well to unseen validation data. To address this issue, a technique called early stopping can be employed. Early stopping involves stopping the training just before overfitting occurs. By monitoring the training and validation errors and stopping the training when the validation error starts to increase, the model can be prevented from memorizing the training data and achieve better generalization. By using early stopping, there is less need to be concerned about setting the exact number of training epochs. A higher number of epochs can be chosen, and the early stopping algorithm will automatically stop the training when there is no improvement in the error.

Determining the right combination of hyperparameters that maximizes model performance is a critical process in machine learning, as it directly leads to achieving optimal results for any given model. Such a

vital step is known as hyperparameter optimization. It is essential to strike a harmonious balance with hyperparameters to ensure the network adeptly captures the data's features. An undersized network might not capture the data's intricacies, leading to underfitting. Conversely, an oversized network could risk overfitting by merely memorizing the training set. The dataset's intricacy dictates the optimal network size, usually ascertained through rigorous trials and performance evaluations. Several common optimization algorithms [150] aid in this pursuit.

The most straightforward approach is the *Grid Search*, often described as a brute-force method. In Grid Search, one defines a set of possible values for each hyperparameter, and the algorithm evaluates the model performance for each combination in a Cartesian product of these sets. The primary advantage of Grid Search is its thoroughness; every possible combination is evaluated. However, this can become computationally infeasible when dealing with a high-dimensional hyperparameter space. Mathematically, for n hyperparameters each having m possible values, Grid Search evaluates the model m^n times.

To mitigate the computational cost of Grid Search, *Random Search* was introduced. Instead of exhaustively evaluating all combinations, Random Search samples points in the hyperparameter space randomly. This approach is both more efficient and has a probability of finding optimal or near-optimal solutions without evaluating the entire space. For hyperparameters in a continuous domain, Random Search samples values uniformly from a predefined range. While Grid and Random Searches are easy to implement and understand, they operate with no prior knowledge of the function's behavior—meaning they don't consider how the performance of a model changes as different hyperparameters are used. They simply try various hyperparameter combinations and observe the results.

Moving beyond these methods, *Bayesian optimization* emerges as an informative strategy, providing a more systematic way of navigating the hyperparameter space. It learns from previous evaluations to make intelligent decisions about where in the hyperparameter space to evaluate next. At the heart of Bayesian optimization is the modeling of the unknown *objective function*, which typically measures the performance of a machine learning model for a particular set of hyperparameters. In Bayesian optimization, the aim is to find the input values to the objective function that either maximizes or minimizes its output. However, instead of exhaustively trying every possible input (like in Grid Search) or randomly trying inputs (like in Random Search), Bayesian optimization uses a probabilistic model to predict which inputs are likely to be optimal.

Initially, an assumption is made about a *prior distribution*, reflecting the initial beliefs about the behavior of the objective function before observing any data. This statistical model assigns probabilities to various potential outcomes or values of the function. During the optimization process, the objective function is evaluated at specific points to gather data. Each evaluation produces a pair: the chosen

input, typically a set of hyperparameters, and the corresponding output, indicating the performance of a machine learning model configured with those hyperparameters. After collecting some data, initial assumptions about the objective function are updated. This updated set of beliefs is depicted by the *posterior distribution*. Utilizing Bayes' theorem (see Apendix A), this distribution is a combination of our prior assumptions and the data we've observed. The posterior distribution provides a refined understanding of the function's behavior, offering both an estimated value at each point and the uncertainty associated with that estimate. The posterior distribution is often termed a "surrogate". It provides an efficient way to estimate the function's value at any given point and, importantly, our degree of confidence in that estimate.

A key strength of Bayesian optimization is its ability to quantify uncertainty. In parts of the input space that are less explored, the surrogate model exhibits increased uncertainty. This aspect is crucial, guiding the exploration strategy. An area exhibiting both high uncertainty and potential high performance is considered for exploration.

Another key elements to the Bayesian optimization process is the *acquisition function*. It is responsible for selecting the next sampling point and balancing the surrogate's performance estimate and the uncertainty involved. Its main goal is to find the right balance between exploration (sampling in areas of uncertainty) and exploitation (sampling in areas where high performance is expected). By maintaining this balance, the acquisition function steers the optimization process efficiently towards the optimum of the objective function.

There are various Hyperparameter Optimization libraries which develop Bayesian optimization methods including Bayesian Optimization library [151], GPyOpt [152], RayTune [153], Optuna [154], HyperOpt [155]. In this research, we chose to use Keras Tuner library [156] to pick the optimal set of hyperparameters to design the 3D-CNN model. It allows the user to optimize integer values (such as the number of neurons in a dense layer and the size of convolutional filters), continuous values (like learning rate and dropout rate), discrete choices (such as the type of optimizer), binary choices (such as whether or not to include a specific layer), and fixed hyperparameters. Keras Tuner requires Python 3.6+ and TensorFlow 2.0+. In this study we used the Python programming language (version 3.8.10) and its open-source neural network library Keras (version 2.13.0, as integrated within TensorFlow) [157] to implement the model and analysis.

4.5 3D-CNN Model Design and Results

At this point, we have a good insight into the architecture of a CNN and role of hyperparameters in producing a trained CNN model. Our focus in this research is performing 3-class classification over hyperspectral images of nitrogen-stressed lettuce, where we classified input lettuce data into one of the three distinct classes: N^-, $N^{+1/2}$, N^+. Figure 4.9 shows the design order of layers we considered for our 3D-CNN model. Given the limited size of our dataset, which includes 30 samples for N^-, 28 samples[1] for $N^{+1/2}$, and 30 samples for N^+, we opted for a conservative approach in designing the 3D-CNN architecture. From a neural network design perspective, a high number of layers could potentially lead to overfitting due to the model's complexity relative to the modest amount of training data. Consequently, the hyperparameter ranges have been chosen to limit the network's depth and capacity, thereby mitigating the risk of overfitting and improving generalization to new data. In this term, we consider a maximum of 2 and 4 layers for convolutional and dense blocks, respectively. However, there are limitations with Keras Tuner library in hyperparameter configuration due to generation of invalid models. This library is not able to skip all invalid models during tuning process. The generation of an invalid model typically occurs when the hyperparameters selected by the tuner result in a configuration that are not compatible with the input data's shape or with the constraints of the neural network layers. For example, if a combination of kernel size and stride results in a convolutional or pooling layer output that has a negative dimension size, the model will be invalid. In addition, in optimization problems, resource allocation and consumption are major issues in configuring the search space of neural networks, specially 3D-CNNs, for HSI data. This led us to limit the range of hyperparameters to prevent resource exhaustion during the training process. Hence, we meticulously configured the range of hyperparameters based on the size of the dataset, as well as computational resources. This step was initially performed through trial and error to understand the limitation of the model for setting the hyperparameters and restriction of the GPU resources. In this research, we utilized NVIDIA QUADRO RTX 8000 with 48 GB RAM.

Tables 4.3 and 4.4 show the range of hyperparameters selected for designing 3D-CNN model based on inclusion of 1 and 2 convolutional blocks, respectively. For ease of reference in our future discussions, we will designate the 3D-CNN model with a single convolutional block as Design 1 and the model with two convolutional blocks as Design 2. This nomenclature will be used consistently throughout our documentation to differentiate between the two model architectures. Along with the hyperparameters provided in the given tables, the batch size was determined manually based on expensive computation

[1]We lost the HSI data pertaining to one pot of lettuce imaged on July 18, hence, we did not include the HSI data corresponding to this pot of lettuce in the dataset including July 18 and July 21.

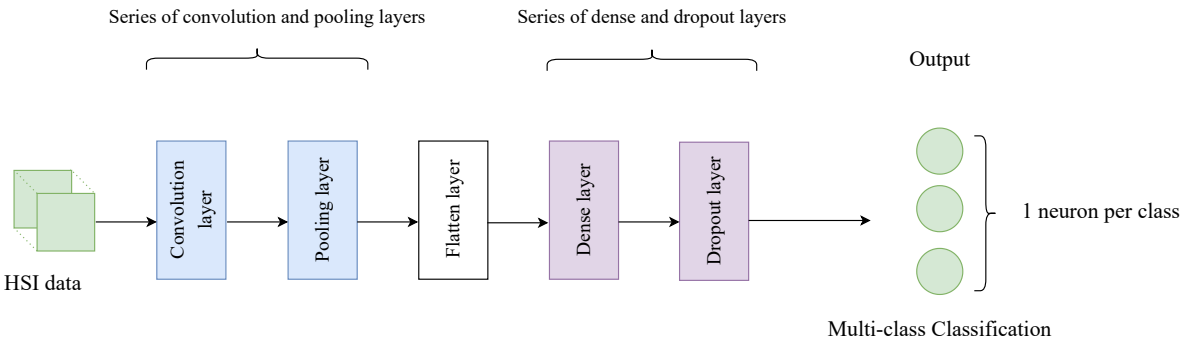


Figure 4.9: Foundation of 3D-CNN model design considered in this experiment.

and resource availability. The number of epochs is set through the *early stopping* process discussed in Table 4.4. This prevents overfitting by halting the training process when the model's performance on a validation set ceases to improve, thus optimizing generalization to new data and saving computational resources.

Table 4.3: Range of hyperparameters for 3D-CNN model with one convolutional block.

Category	Hyperparameter	Range
Convolutional layer 1	Number of filters Kernel size Stride	(2, 32), step=2 (2, 7) (1, kernel size in each dimension)
Pooling layer 1	Pooling size = Pooling stride size	(2, 5)
Dense layer	Number of dense layers Number of neurons (units)	(1,4) (64, 512), step=32
Dropout layer	Dropout rate	(0, 0.5), step=0.1
General hyperparameters	Batch size Number of Epochs Learning rate	3 200 along with Early Stopping (patience= 30) (1e-4, 1e-2), sampling=log

To find nearly optimal set of hyperparameters, we followed the procedure provided in Flowchart 4.10. First, we partitioned the dataset into two subsets: 76 samples were allocated for model development, and 12 samples were designated for model evaluation as the test set. The splitting process is stratified by both the date of image acquisition and the target class label. This ensures that for each date in the dataset, and for each label, a proportional representation is maintained in both sets. Then, we utilized 7-fold stratified cross validation to perform model development through Keras Tuner. We set the Keras Tuner to generate 25 valid models per fold for Design 1 and Design 2. 175 models were generated per design. Subsequently, we selected the best performing model per fold for each design. This yields 7 different models for Designs 1 and 2. In the next stage, each model was retrained over 7 folds for 200

Table 4.4: Range of hyperparameters for 3D-CNN model with two convolutional blocks.

Category	Hyperparameter	Range
Convolutional layer 1	Number of filters	(2, 32), step=2
	Kernel size	(1, 7)
	Stride	(2, 5)
Pooling layer 1	Pooling size = Pooling stride size	(2, 5)
Convolutional layer 2	Number of filters	(2,32), step=2
	Kernel size	(1, 5)
	Stride	(1,3)
Pooling layer 2	Pooling size = Pooling stride size	(2, 5)
Dense layer	Number of dense layers	(1, 4)
	Number of neurons (units)	(64,512), step= 32
Dropout layer	Dropout rate	(0, 0.5), step= 0.1
General hyperparameters	Batch size	3
	Number of Epochs	200 along with Early Stopping (patience= 30)
	Learning rate	(1e-4, 1e-2), sampling= log

epochs (with early stopping equals to 30 epochs) per design. The summary of hyperparameters and the corresponded neural network architecture for each of the 7 models per design is provided in Appendix B. Then we used the top two models with highest average accuracies per Design to evaluate the test set. We finally select the model with higher test accuracy as the best performing model per design. If two models per design give the same accuracies, we keep both models for further process.

In Design 1, based on an average accuracy of 7 models, Model 5 and Model 7 received the highest results. Table 4.5 shows the test accuracies per fold for Models 5 and 7. As is reported, folds 3 and 4 of Model 5 gained the highest test accuracy. Figures 4.11 and 4.12 show the evaluation results over 12 test samples which were not exposed to models during training and validation process. Model 5 (Fold 3) and Model 5 (Fold 4) were able to correctly classify 9 images out of 12 provided test samples. The confusion matrices are given in Figure 4.14.

In the same procedure within Design 2, based on an average accuracy of 7 models, Model 3 and Model 5 received the highest results. Table 4.6 shows the test accuracies per fold for Models 3 and 5. As is reported, fold 2 of Model 5 gained the highest test accuracy. Figure 4.13 shows the evaluation results over the same 12 test samples that were used in Design 1. The model was able to correctly classify 8 images out of 12 provided test samples. The confusion matrix is given in Figure 4.15.

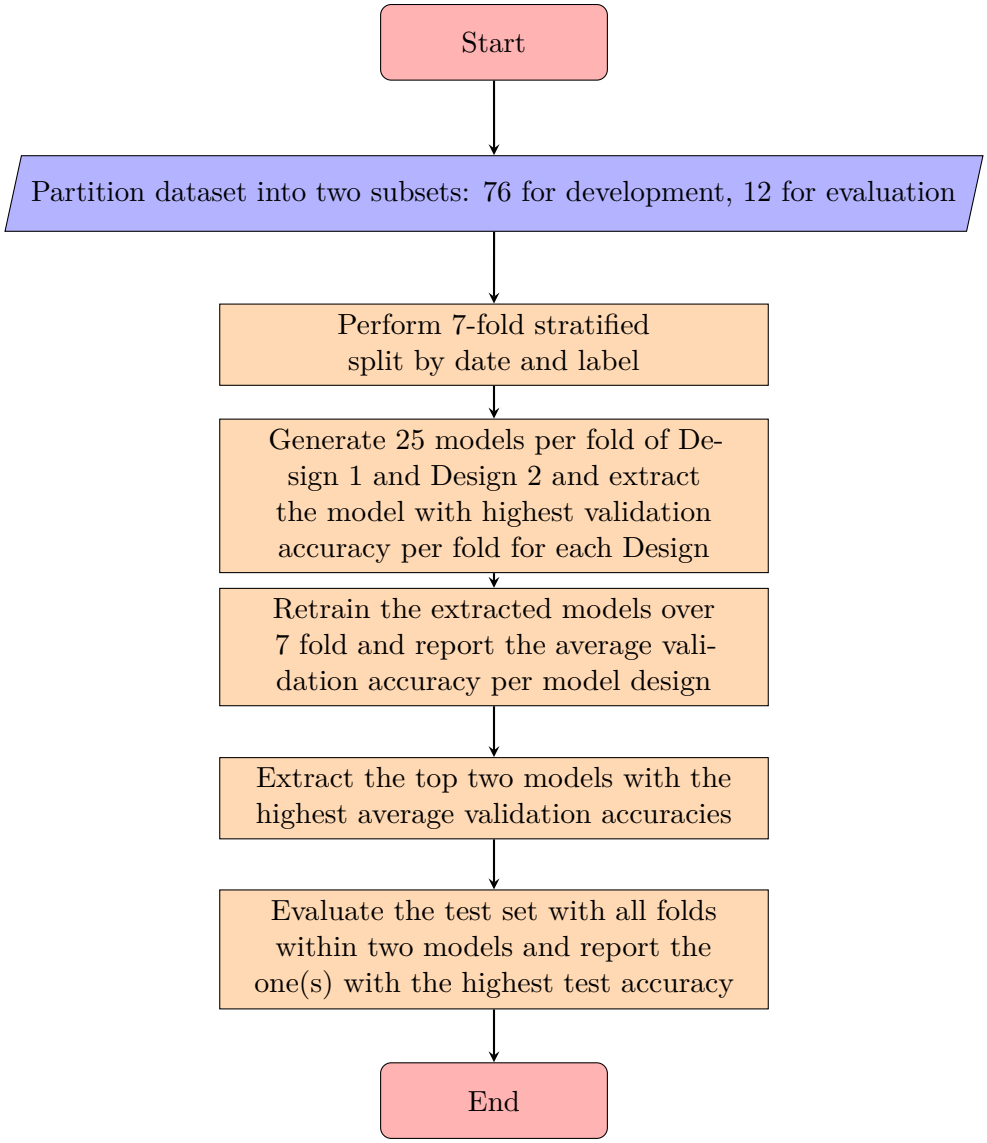


Figure 4.10: Flowchart of the model development and evaluation process.

Table 4.5: Test accuracies per fold of Models 5 and 7 of Design 1.

Fold Number	Model 5: Test Accuracy	Model 7: Test Accuracy
1	66.67%	66.67%
2	50.00%	66.67%
3	75.00%	50.00%
4	75.00%	66.67%
5	58.33%	66.67%
6	58.33%	66.67%
7	50.00%	66.67%

Table 4.6: Test accuracies per fold of Models 3 and 5 of Design 2.

Fold Number	Model 3: Test Accuracy	Model 5: Test Accuracy
1	33.33%	58.33%
2	33.33%	66.67%
3	58.33%	50.00%
4	50.00%	50.00%
5	58.33	50.00%
6	33.33%	50.00%
7	50.00%	41.67%

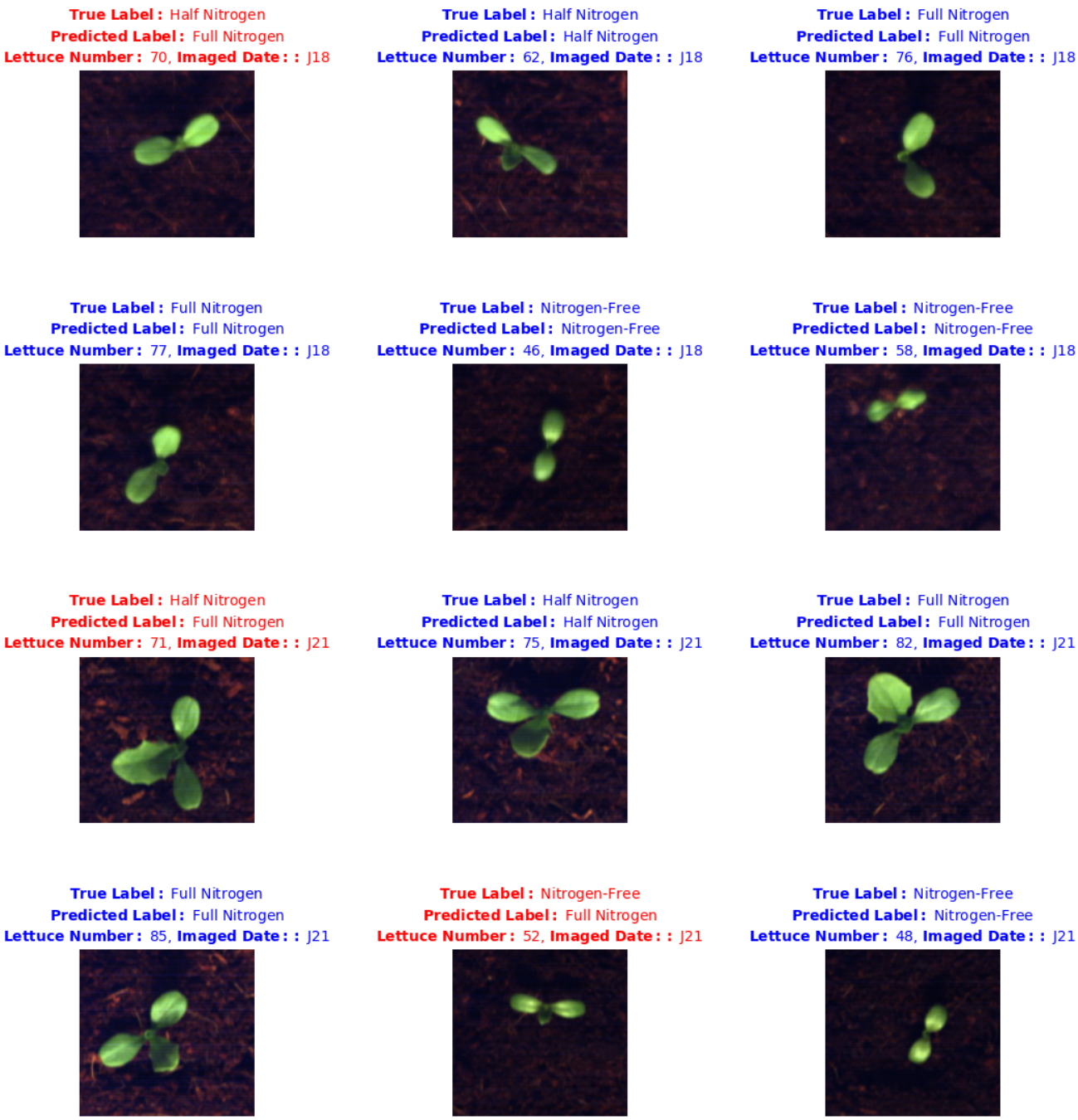


Figure 4.11: Design 1: Model 5 (Fold 3) prediction result over test set. Blue color shows correct prediction and red color indicates wrong prediction.

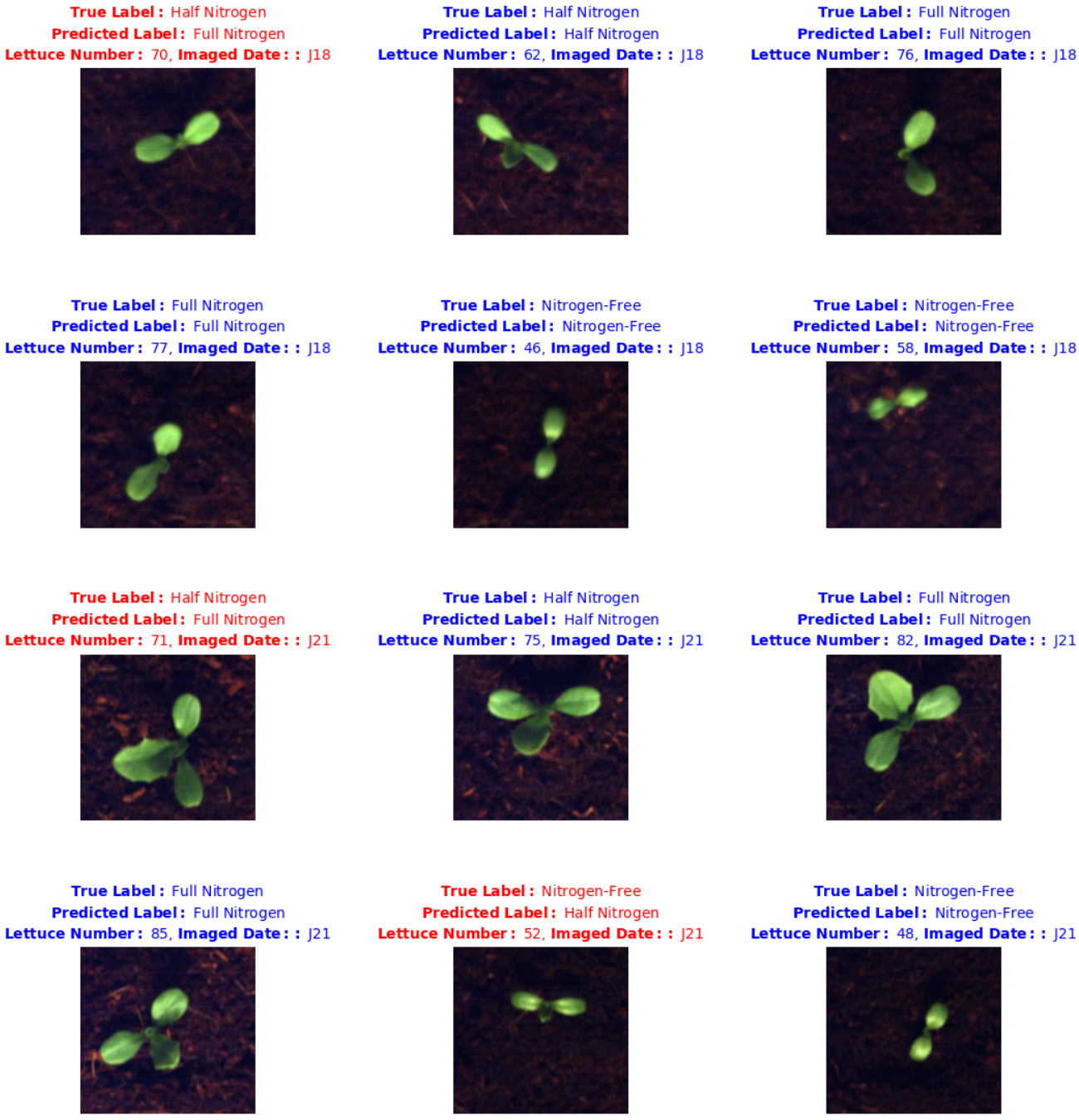


Figure 4.12: Design 1: Model 5 (Fold 4) prediction result over test set. Blue color shows correct prediction and red color indicates wrong prediction.

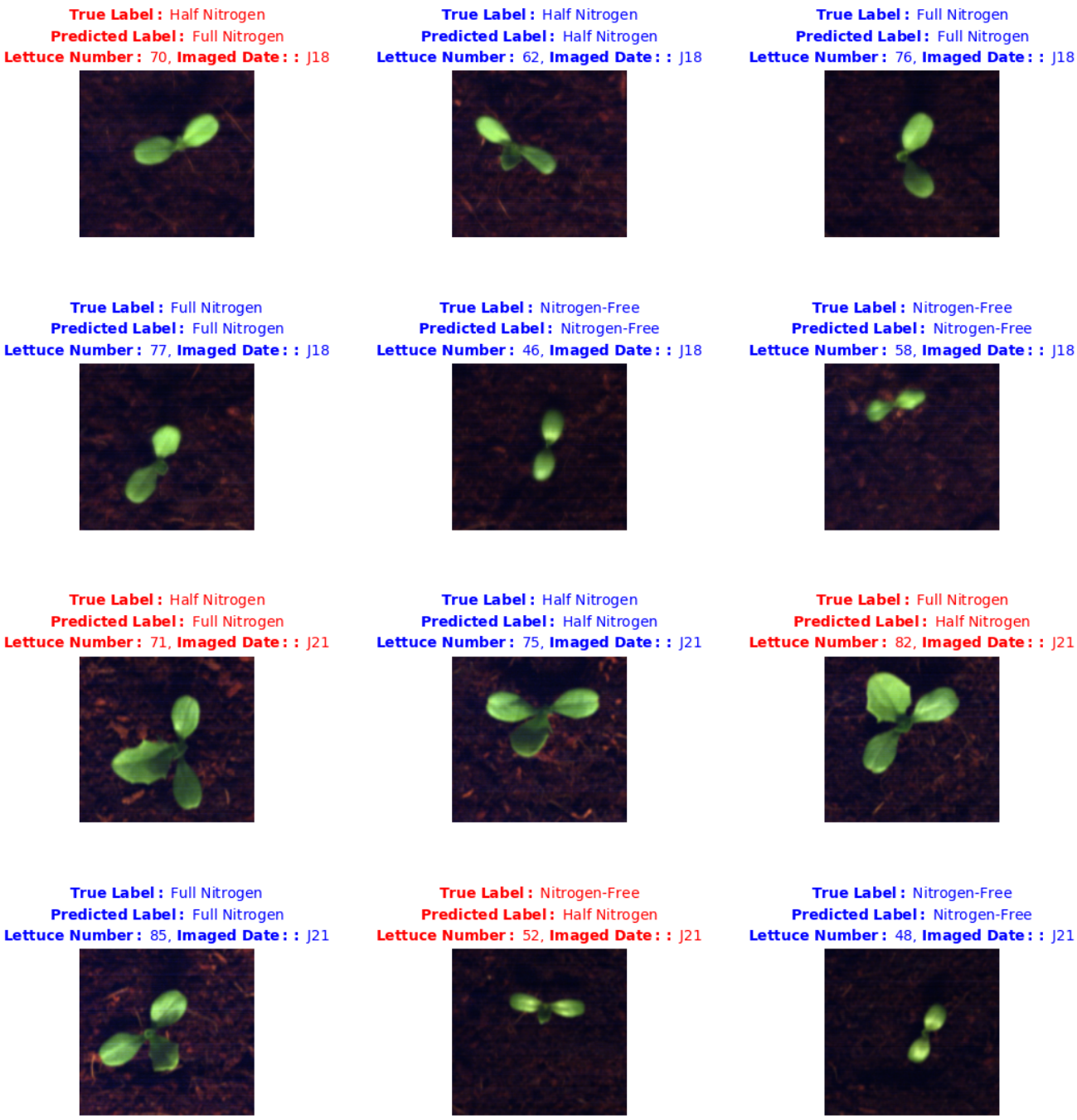
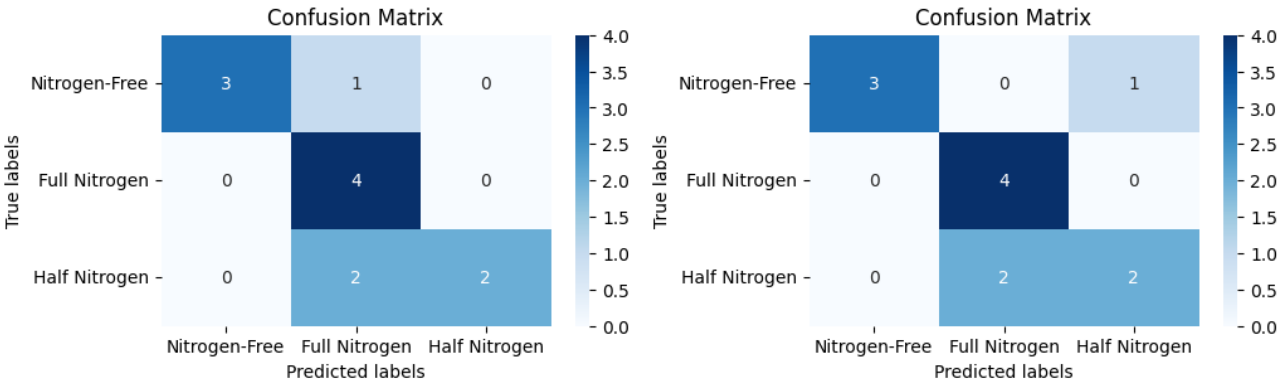


Figure 4.13: Design 2's prediction result over test set. Blue color shows correct prediction and red color indicates wrong prediction.



(a) Confusion matrix for Model 5 (Fold 3).

(b) Confusion matrix for Model 5 (Fold 4).

Figure 4.14: Confusion Matrices for Design 1's best model.

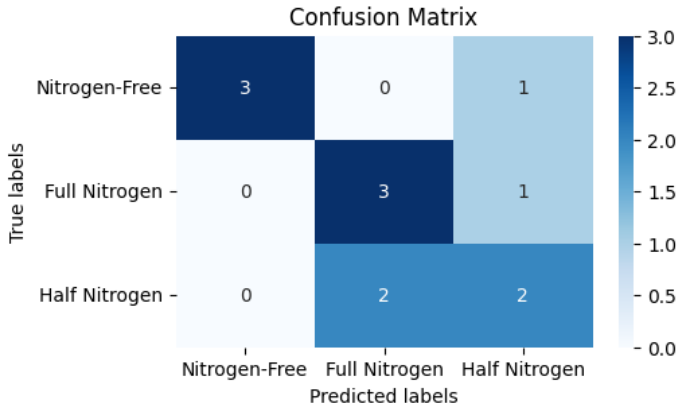


Figure 4.15: Confusion Matrix for Design 2's best model.

4.5.1 Discussion

Our empirical results show the superiority of Design 1 with 1 convolutional block over Design 2 with 2 convolutional blocks on HSI data classification. The best model in Designs 1 and 2 achieved 75.00% and 66.67% test accuracy, respectively. Considering the relative small size of the HSI dataset, the lower accuracy of Design 2 can be related to the complexity of its model with 2 convolutional blocks and 4 dense layers whereas Design 1 has 1 convolutional block and 1 dense layer which makes it a simple model. Simpler models are less prone to overfitting, and in cases where the data may not be sufficient to train complex models effectively, simplicity can lead to better generalization.

While Design 2 has more convolutional blocks, it is possible that the additional complexity did not necessarily lead to better feature extraction. If the hyperspectral data did not benefit from the deeper convolutional layers, it might explain why Design 1, with its simpler architecture, performed better. In conclusion, it is not uncommon for simpler models to outperform more complex ones, especially when dealing with limited data or when the data does not require deep hierarchical feature extraction.

4.6 Exploring Channel Importance in Classification Result using Ablation Study

HSI captures intricate details across the electromagnetic spectrum that the data is characterized by its high dimensionality. The 400–1000 nm range, which encompasses the visible to NIR regions, provides detailed spectral information. Given the data's richness, assessing the significance of each spectral channel becomes an imperative aspect of effective data utilization.

The rationale for evaluating channel importance is multifold. The vastness of hyperspectral data, attributed to its numerous spectral bands, can introduce computational challenges. Redundant or less informative channels might unnecessarily complicate the models. By identifying the most salient channels, one can streamline data which makes computations more manageable and efficient. This understanding can further enhance the accuracy and reliability of classification models since distinct bands harbor specific information. Recognizing which channels are more discriminative for a particular task can refine the models. Moreover, understanding the pivotal channels can also elucidate the spectral properties that play a role in differentiating between output classes.

From a practical standpoint, this evaluation has several implications. In the real-world sensor development process, capturing the entire spectrum can be resource-intensive. If accurate classification primarily relies on specific channels, then sensor design can focus on these channels, resulting in economic

efficiency. Additionally, leveraging all channels might lead to the risk of overfitting, especially if some channels present redundant or correlated information. Models that prioritize essential channels are typically more sturdy and demonstrates improved performance on new, unseen data. Furthermore, for scenarios demanding real-time data processing or those with limited computational capacities, a model that emphasizes the essential channels can provide quicker and more reliable outputs.

In our research, we explored the importance of individual channels and various channel groupings in classification results specific to our model through an ablation study. In this exploration, we replaced part of the input data with zero value to see how the affect on the prediction results per target class. For this experiment, we selected Model 5 (Fold 3), which, along with its counterpart Model 5 (Fold 4) in Design 1, showcased the highest test accuracy (75.00%). The focus is on assessing channel importance across target classes (N^-, $N^{+1/2}$, and N^+) within our test dataset.

Our test dataset was evenly distributed across these three nitrogen treatment classes. For each target class we computed the individual channel importance using test images corresponded to that class. The computation procedure is as follows. Each time we zeroed one channel and then computed the difference between prediction probabilities of the original test images and the test images with the zeroed channel. The difference in predictions represents the contribution of the channel to the model's prediction. The importance score for the channel is computed as the mean absolute difference in predictions, which quantifies how much the removal of that channel affects the model's output. This process is repeated for all channels and resulted in importance scores, where higher scores indicate channels that have a more significant impact on the model's predictions. Figures 4.16 to 4.18 show the corresponded results for each nitrogen treatment class, thereby providing insights into which channels are most influential in our model's performance.

Observations show that there is a general increase in importance scores moving from the violet to the red channels, peaking in the red channel range. This could suggest that the red channels are more informative for the model's predictions across all channels per class. After reaching the peak in the red channels, there is a sharp decline as it moves into the NIR range. We can see the variability within each range, indicating that not all channels within a given color and NIR range are equally important. In another experiment, we zeroed all channels within each color (including violet, blue, green, yellow, orange, and red) and NIR spectrum and computed the importance score for a group of channels for each target class using its corresponded test images. Figure 4.19 shows the importance scores for each group of bands per class. The collective channels show importance of red and green spectra in directing the prediction results for each class. The result is consistent with findings in Chapter 2. It can be observed

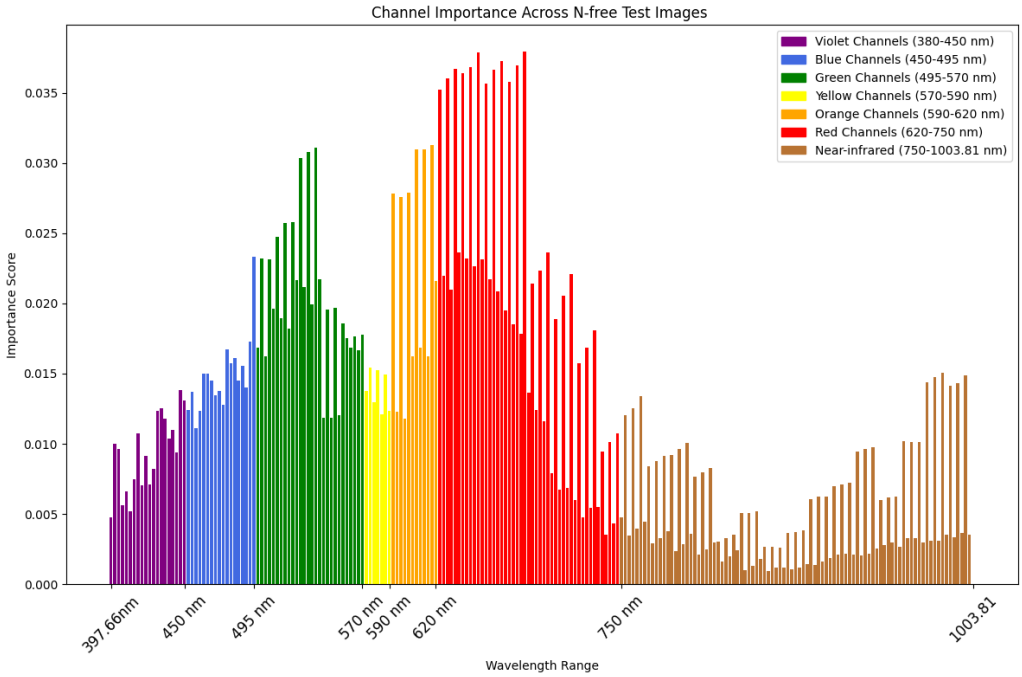


Figure 4.16: Individual channel importance for N^- class across the test dataset using Model 5 (Fold 3).

from the results that red spectrum has higher importance in prediction results of N^- and $N^{+1/2}$ classes. In addition, $N^{+1/2}$ class achieved higher importance score among all target classes in NIR range.

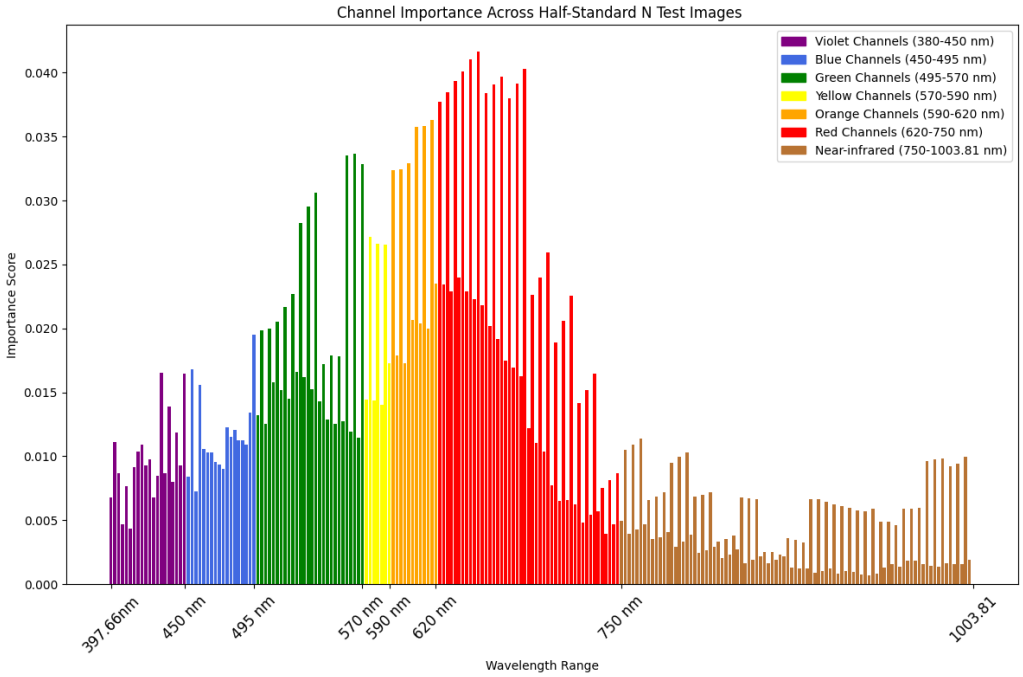


Figure 4.17: Individual channel importance for $N^{+1/2}$ class across the test dataset using Model 5 (Fold 3).

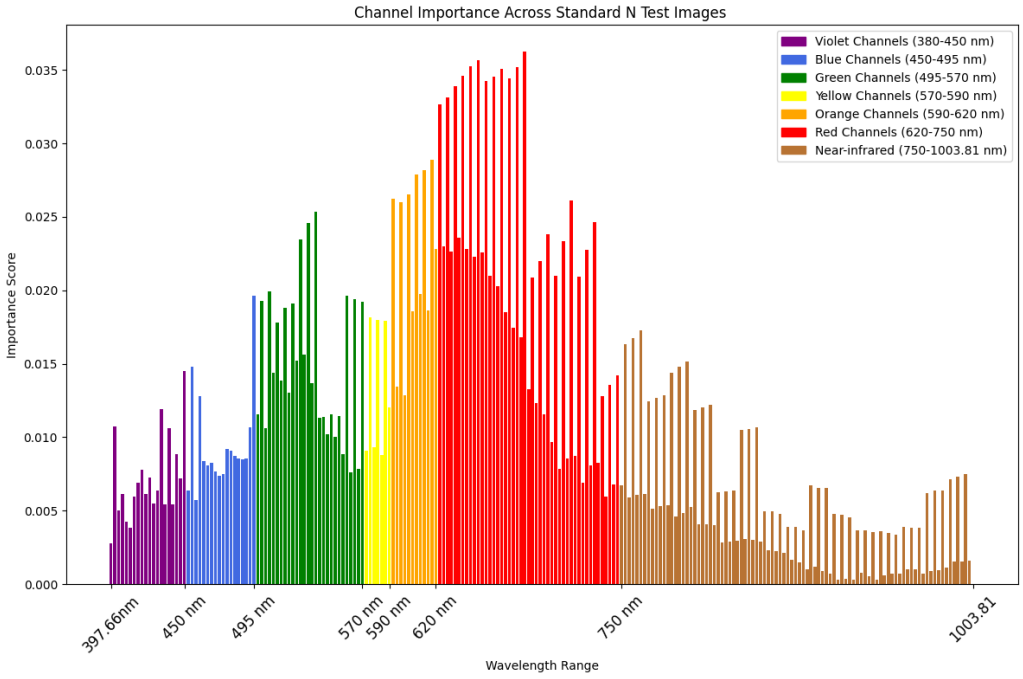


Figure 4.18: Individual channel importance for N^{+} class across the test dataset using Model 5 (Fold 3).

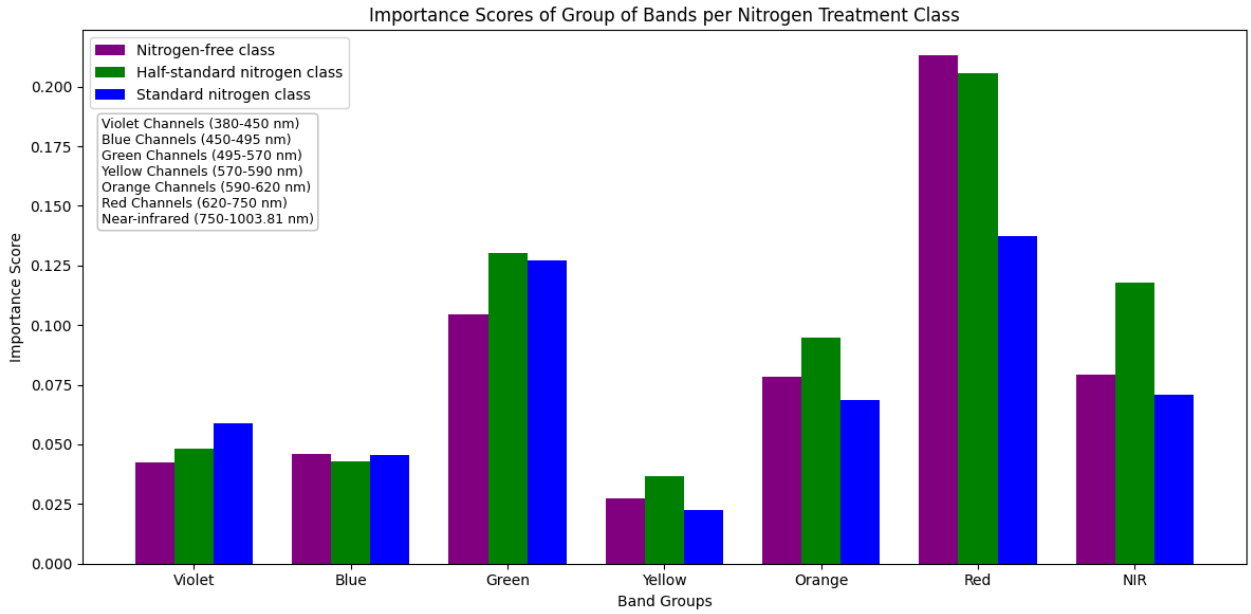


Figure 4.19: Importance scores of group of bands per nitrogen stress treatment.

Chapter 5

Conclusion and Future Works

This thesis primarily explores the capabilities of 3D-CNN models in classifying hyperspectral images of nitrogen-stressed lettuce at early stages of growth. To this end, we examined 350 different 3D-CNN models using Bayesian optimization, considering the relatively small size of our dataset, which totals 88 samples. The results show the superiority of a simpler architecture, consisting of one convolutional and dense block, in classification of unseen images by achieving 75.00% test accuracy. Although the desired result is to correctly classify all unseen images, achieving this requires a richer dataset with more samples per target class (N^-, $N^{+1/2}$, and N^+) for the early stages of growth. With a richer dataset, the model is likely to generalize better to unseen images by extracting the most significant patterns. Moreover, a larger dataset paves the way to better quantifying the power of complex 3D-CNN architecture by extracting more complex patterns which help to better differentiate between target classes.

Additionally, the importance scores for the color and NIR spectrum reveal the significant role of the red and green spectra in prediction outcomes. However, there is still ambiguity whether NIR range has the same potency as visible range in prediction results. We can further validate this effect by enriching the dataset with more samples per target class, as mentioned earlier, and quantifying the results across a larger test dataset.

Furthermore, it is interesting to consider the possibility that the model may be subtly using leaf shape as one of the features in its classification process, particularly given the uniformity in size among the majority of leaves in our dataset. This observation is not intended to undermine the model's performance but rather to highlight an area for potential exploration. This aspect can be examined using the saliency map technique mentioned in Section 3.5.

Building upon our findings, we recommend the following for future endeavors:

- CNNs are data-hungry algorithms that thrive on large datasets to improve their predictive accuracy and generalization capabilities. By extending the nitrogen-stressed dataset to include a broader range of samples, it is possible to conduct a more thorough assessment of the 3D CNN's performance. This expansion not only tests the model's robustness but also refines its efficiency in classifying samples accurately.
- We aim to broaden our research to encompass a wider range of nutrient deficiencies beyond nitrogen. The complexity of this challenge is exemplified by the symptom of leaf yellowing, commonly associated with nitrogen deficiency but also indicative of other deficiencies like magnesium, sulfur, or manganese. Leveraging the detailed spectral data from HSI and the sophisticated feature extraction of 3D-CNNs, our goal is to develop a model capable of distinguishing these subtle variances.
- While our model achieved a test accuracy of 75.00%, indicating room for improvement, it highlights an opportunity in the context of the high costs associated with HSI technology. Our research identified the importance scores for each individual band and groups of bands that impact the prediction of nitrogen deficiency. By focusing on these bands, it is feasible to develop targeted multispectral cameras that offer a more affordable solution. This approach may retain a level of precision in deficiency detection while substantially reducing overall costs.
- While 3D-CNN demonstrates impressive performance in image classification, our literature review in Section 3.2 indicates that combined 2D- and 3D-CNN architectures also yield robust results. Given this evidence, we propose further exploration and development of hybrid architectures, merging the strengths of both 2D and 3D convolutions, to potentially harness richer feature representations and optimize classification performance in HSI applications.
- In this research study we examined how to design the structure of a neural network model using optimization methods– particularly Bayesian optimization in this research- and background knowledge regarding organization of layers. However, with the advancement of AI, researchers are focusing on fully automating the process of finding the best-performing architectural design (including complex architecture) without relying on expert knowledge. This is where the neural architecture search (NAS) [158] comes into play. NAS operates in three main dimensions: search space, search strategy, and performance estimation strategy which are discussed as below:
  - *Search Space*: It defines the range of possible architectures that can be explored.
  - *Search Strategy*: It involves how the search space is navigated. It is a balance between quickly finding high-performance architectures and avoiding premature convergence to suboptimal

solutions. Various strategies can be utilized including random search, Bayesian optimization, evolutionary methods, reinforcement learning, and gradient-based methods.

– *Performance Estimation Strategy*: NAS aims to discover architectures with high performance on unseen data. Performance estimation involves assessing the predicted performance of a model, which can be computationally expensive. Recent research focuses on developing less costly methods for this estimation.

In future, we consider to utilize NAS to generate high-performing neural network models to perform the classification.

Appendix A

Bayes' Theorem

Bayes' Theorem [159] is a fundamental concept in probability theory and statistics, named after the Reverend Thomas Bayes. It provides a way to update our beliefs or probabilities in light of new evidence. The theorem is especially powerful in situations where direct computation of a probability is complex or infeasible. Bayes' Theorem is expressed as:

$$P(H|E) = \frac{P(E|H) \cdot P(H)}{P(E)}.$$

Where:

- $P(H|E)$ is the posterior probability: the probability of the hypothesis H after observing evidence E.
- $P(E|H)$ is the likelihood: the probability of observing the evidence E given that the hypothesis H is true.
- $P(H)$ is the prior probability: the initial probability of the hypothesis H before observing the evidence.
- $P(E)$ is the marginal likelihood or evidence: the total probability of observing the evidence.

Appendix B

Design 1 and 2: Models' Hyperparameters and Their Associated Neural Network Architectures

Tables B.1 to B.14 and Tables B.15 to B.28 show the 14 models with their corresponded hyperparameters configurations and model summaries for Designs 1 and 2.

Table B.1: Design 1: Hyperparameters of Model 1

Parameter	**Value**
Model ID	1
Kernel Size	(4, 4, 7)
Stride Size	(1, 2, 2)
Convolution Filters	8
Pool Size	(5, 5, 5)
Number of Dense Layers	4
Units in Dense Layer 0	448
Dropout in Dense Layer 0	0.4
Units in Dense Layer 1	256
Dropout in Dense Layer 1	0.2
Units in Dense Layer 2	256
Dropout in Dense Layer 2	0.3
Units in Dense Layer 3	64
Dropout in Dense Layer 3	0.3
Learning Rate	1.26×10^{-6}
Average Accuracy	67.27%

Table B.2: Design 1: Neural Network Architecture for Model 1

Layer (type)	**Output Shape**	**Param #**
Conv3D	(None, 197, 104, 109, 8)	904
MaxPooling3D	(None, 39, 20, 21, 8)	0
Flatten	(None, 131040)	0
Dense	(None, 448)	58,706,368
Dropout	(None, 448)	0
Dense	(None, 256)	114,944
Dropout	(None, 256)	0
Dense	(None, 256)	65,792
Dropout	(None, 256)	0
Dense	(None, 64)	16,448
Dropout	(None, 64)	0
Dense	(None, 3)	195
Total params		58,904,651 (224.70 MB)
Trainable params		58,904,651 (224.70 MB)
Non-trainable params		0 (0.00 Byte)

Table B.3: Design 1: Hyperparameters of Model 2

Parameter	**Value**
Model ID	2
Kernel Size	(4, 6, 2)
Stride Size	(1, 1, 2)
Convolution Filters	2
Pool Size	(2, 2, 2)
Number of Dense Layers	1
Units in Dense Layer 0	352
Dropout in Dense Layer 0	0.2
Learning Rate	1.21×10^{-4}
Average Accuracy	65.71%

Table B.4: Design 1: Neural Network Architecture for Model 2

Layer (type)	**Output Shape**	**Param #**
Conv3D	(None, 197, 205, 223, 2)	98
MaxPooling3D	(None, 98, 102, 111, 2)	0
Flatten	(None, 2219112)	0
Dense	(None, 352)	781127776
Dropout	(None, 352)	0
Dense	(None, 3)	1059
Total params		781128933 (2.91 GB)
Trainable params		781128933 (2.91 GB)
Non-trainable params		0 (0.00 Byte)

Table B.5: Design 1: Hyperparameters of Model 3

Parameter	Value
Model ID	3
Kernel Size	(5, 7, 7)
Stride Size	(1, 2, 2)
Convolution Filters	8
Pool Size	(4, 4, 4)
Number of Dense Layers	2
Units in Dense Layer 0	416
Dropout in Dense Layer 0	0.0
Units in Dense Layer 1	64
Dropout in Dense Layer 1	0.1
Learning Rate	1.97×10^{-4}
Average Accuracy	68.31%

Table B.6: Design 1: Neural Network Architecture for Model 3

Layer (type)	Output Shape	Param #
Conv3D	(None, 196, 102, 109, 8)	1968
MaxPooling3D	(None, 49, 25, 27, 8)	0
Flatten	(None, 264600)	0
Dense	(None, 416)	110074016
Dropout	(None, 416)	0
Dense	(None, 64)	26688
Dropout	(None, 64)	0
Dense	(None, 3)	195
Total params		110102867 (420.01 MB)
Trainable params		110102867 (420.01 MB)
Non-trainable params		0 (0.00 Byte)

Table B.7: Design 1: Hyperparameters of Model 4

Parameter	Value
Model ID	4
Kernel Size	(5, 6, 6)
Stride Size	(1, 1, 1)
Convolution Filters	10
Pool Size	(3, 3, 3)
Number of Dense Layers	4
Units in Dense Layer 0	160
Dropout in Dense Layer 0	0.4
Units in Dense Layer 1	224
Dropout in Dense Layer 1	0.0
Units in Dense Layer 2	128
Dropout in Dense Layer 2	0.1
Units in Dense Layer 3	64
Dropout in Dense Layer 3	0.2
Learning Rate	4.42×10^{-4}
Average Accuracy	71.17%

Table B.8: Design 1: Neural Network Architecture for Model 4

Layer (type)	Output Shape	Param #
Conv3D	(None, 196, 205, 219, 10)	1810
MaxPooling3D	(None, 65, 68, 73, 10)	0
Flatten	(None, 3226600)	0
Dense	(None, 160)	516256160
Dropout	(None, 160)	0
Dense	(None, 224)	36064
Dropout	(None, 224)	0
Dense	(None, 128)	28800
Dropout	(None, 128)	0
Dense	(None, 64)	8256
Dropout	(None, 64)	0
Dense	(None, 3)	195
Total params		516331285 (1.92 GB)
Trainable params		516331285 (1.92 GB)
Non-trainable params		0 (0.00 Byte)

Table B.9: Design 1: Hyperparameters of Model 5

Parameter	Value
Model ID	5
Kernel Size	(6, 2, 3)
Stride Size	(1, 2, 2)
Convolution Filters	6
Pool Size	(3, 3, 3)
Number of Dense Layers	1
Units in Dense Layer 0	320
Dropout in Dense Layer 0	0.3
Learning Rate	2.0×10^{-4}
Average Accuracy	75.06%

Table B.10: Design 1: Neural Network Architecture for Model 5

Layer (type)	Output Shape	Param #
Conv3D	(None, 195, 105, 111, 6)	222
MaxPooling3D	(None, 65, 35, 37, 6)	0
Flatten	(None, 505050)	0
Dense	(None, 320)	161616320
Dropout	(None, 320)	0
Dense	(None, 3)	963
Total params		161617505 (616.52 MB)
Trainable params		161617505 (616.52 MB)
Non-trainable params		0 (0.00 Byte)

Table B.11: Design 1: Hyperparameters of Model 6

Parameter	Value
Model ID	6
Kernel Size	(7, 2, 3)
Stride Size	(1, 2, 2)
Convolution Filters	10
Pool Size	(3, 3, 3)
Number of Dense Layers	4
Units in Dense Layer 0	480
Dropout in Dense Layer 0	0.1
Units in Dense Layer 1	320
Dropout in Dense Layer 1	0.4
Units in Dense Layer 2	64
Dropout in Dense Layer 2	0.2
Units in Dense Layer 3	64
Dropout in Dense Layer 3	0.4
Learning Rate	4.48×10^{-4}
Average Accuracy	65.84%

Table B.12: Design 1: Neural Network Architecture for Model 6

Layer (type)	**Output Shape**	**Param #**
Conv3D	(None, 194, 105, 111, 10)	430
MaxPooling3D	(None, 64, 35, 37, 10)	0
Flatten	(None, 828800)	0
Dense	(None, 480)	397824480
Dropout	(None, 480)	0
Dense	(None, 320)	153920
Dropout	(None, 320)	0
Dense	(None, 64)	20544
Dropout	(None, 64)	0
Dense	(None, 64)	4160
Dropout	(None, 64)	0
Dense	(None, 3)	195
Total params		398003729 (1.48 GB)
Trainable params		398003729 (1.48 GB)
Non-trainable params		0 (0.00 Byte)

Table B.13: Design 1: Hyperparameters of Model 7

Parameter	**Value**
Model ID	7
Kernel Size	(3, 3, 2)
Stride Size	(1, 2, 1)
Convolution Filters	6
Pool Size	(4, 4, 4)
Number of Dense Layers	4
Units in Dense Layer 0	192
Dropout in Dense Layer 0	0.2
Units in Dense Layer 1	64
Dropout in Dense Layer 1	0.3
Units in Dense Layer 2	64
Dropout in Dense Layer 2	0.1
Units in Dense Layer 3	64
Dropout in Dense Layer 3	0.0
Learning Rate	2.33×10^{-4}
Average Accuracy	78.96%

Table B.14: Design 1: Neural Network Architecture for Model 7

Layer (type)	**Output Shape**	**Param #**
Conv3D	(None, 198, 104, 223, 6)	114
MaxPooling3D	(None, 49, 26, 55, 6)	0
Flatten	(None, 420420)	0
Dense	(None, 192)	80720832
Dropout	(None, 192)	0
Dense	(None, 64)	12352
Dropout	(None, 64)	0
Dense	(None, 64)	4160
Dropout	(None, 64)	0
Dense	(None, 64)	4160
Dropout	(None, 64)	0
Dense	(None, 3)	195
Total params		80741813 (308.01 MB)
Trainable params		80741813 (308.01 MB)
Non-trainable params		0 (0.00 Byte)

Table B.15: Design 2: Hyperparameters of Model 1

Parameter	**Value**
Model ID	1
Kernel Size (Layer 0)	(4, 3, 5)
Stride Size (Layer 0)	(1, 1, 1)
Convolution Filters (Layer 0)	26
Pool Size (Layer 0)	(2, 2 , 2)
Kernel Size (Layer 1)	(5, 1, 1)
Stride Size (Layer 1)	(3, 1, 2)
Convolution Filters (Layer 1)	16
Pool Size (Layer 1)	(3, 3, 3)
Dense Layers	3
Units in Dense Layer 0	352
Dropout in Dense Layer 0	0.4
Units in Dense Layer 1	96
Dropout in Dense Layer 1	0.1
Units in Dense Layer 2	416
Dropout in Dense Layer 2	0.3
Learning Rate	2.791×10^{-3}
Average Accuracy	42.08%

Table B.16: Design 2: Neural Network Architecture for Model 1

Layer Type	**Output Shape**	**Param #**
Conv3D	(None, 197, 208, 220, 26)	1586
MaxPooling3D	(None, 98, 104, 110, 26)	0
Conv3D	(None, 32, 104, 55, 16)	2096
MaxPooling3D	(None, 10, 34, 18, 16)	0
Flatten	(None, 97920)	0
Dense	(None, 352)	34468192
Dropout	(None, 352)	0
Dense	(None, 96)	33888
Dropout	(None, 96)	0
Dense	(None, 416)	40352
Dropout	(None, 416)	0
Dense	(None, 3)	1251
Total params		34547365 (131.79 MB)
Trainable params		34547365 (131.79 MB)
Non-trainable params		0 (0.00 Byte)

Table B.17: Design 2: Hyperparameters of Model 2

Parameter	**Value**
Model ID	2
Kernel Size (Layer 0)	(5, 7, 1)
Stride Size (Layer 0)	(1, 1, 1)
Convolution Filters (Layer 0)	22
Pool Size (Layer 0)	(5, 5, 5)
Kernel Size (Layer 1)	(4, 2, 2)
Stride Size (Layer 1)	(2, 2, 2)
Convolution Filters (Layer 1)	8
Pool Size (Layer 1)	(4, 4, 4)
Dense Layers	1
Units in Dense Layer 0	64
Dropout in Dense Layer 0	0.1
Learning Rate	2.86×10^{-3}
Average Accuracy	58.83%

Table B.18: Design 2: Neural Network Architecture for Model 2

Layer (type)	**Output Shape**	**Param #**
Conv3D	(None, 196, 204, 224, 22)	792
MaxPooling3D	(None, 39, 40, 44, 22)	0
Conv3D	(None, 18, 20, 22, 8)	2824
MaxPooling3D	(None, 4, 5, 5, 8)	0
Flatten	(None, 800)	0
Dense	(None, 64)	51264
Dropout	(None, 64)	0
Dense	(None, 3)	195
Total params		55075 (215.14 KB)
Trainable params		55075 (215.14 KB)
Non-trainable params		0 (0.00 Byte)

Table B.19: Design 2: Hyperparameters of Model 3

Parameter	**Value**
Model ID	3
Kernel Size (Layer 0)	(7, 3, 2)
Stride Size (Layer 0)	(1, 1, 1)
Convolution Filters (Layer 0)	26
Pool Size (Layer 0)	(4, 4, 4)
Kernel Size (Layer 1)	(2, 3, 5)
Stride Size (Layer 1)	(2, 2, 2)
Convolution Filters (Layer 1)	22
Pool Size (Layer 1)	(3, 3, 3)
Dense Layers	3
Units in Dense Layer 0	352
Dropout in Dense Layer 0	0.4
Units in Dense Layer 1	64
Dropout in Dense Layer 1	0.0
Units in Dense Layer 2	64
Dropout in Dense Layer 2	0.0
Learning Rate	2.78×10^{-4}
Average Accuracy	73.77%

Table B.20: Design 2: Neural Network Architecture for Model 3

Layer Type	**Output Shape**	**Param #**
Conv3D	(None, 194, 208, 223, 26)	1118
MaxPooling3D	(None, 48, 52, 55, 26)	0
Conv3D	(None, 24, 25, 26, 22)	17182
MaxPooling3D	(None, 8, 8, 8, 22)	0
Flatten	(None, 11264)	0
Dense	(None, 352)	3965280
Dropout	(None, 352)	0
Dense	(None, 64)	22592
Dropout	(None, 64)	0
Dense	(None, 64)	4160
Dropout	(None, 64)	0
Dense	(None, 3)	195
Total params		4010527 (15.30 MB)
Trainable params		4010527 (15.30 MB)
Non-trainable params		0 (0.00 Byte)

Table B.21: Design 2: Hyperparameters of Model 4

Parameter	**Value**
Model ID	4
Kernel Size (Layer 0)	(5, 7, 6)
Stride Size (Layer 0)	(1, 1, 1)
Convolution Filters (Layer 0)	16
Pool Size (Layer 0)	(5, 5, 5)
Kernel Size (Layer 1)	(5, 3, 2)
Stride Size (Layer 1)	(2, 3, 1)
Convolution Filters (Layer 1)	30
Pool Size (Layer 1)	(4, 4, 4)
Dense Layers	3
Units in Dense Layer 0	160
Dropout in Dense Layer 0	0.1
Units in Dense Layer 1	96
Dropout in Dense Layer 1	0.3
Units in Dense Layer 2	448
Dropout in Dense Layer 2	0.4
Learning Rate	1.63×10^{-3}
Average Accuracy	36.89%

Table B.22: Design 2: Neural Network Architecture for Model 4

Layer (type)	**Output Shape**	**Param #**
Conv3D	(None, 196, 204, 219, 16)	3376
MaxPooling3D	(None, 39, 40, 43, 16)	0
Conv3D	(None, 18, 13, 42, 30)	14430
MaxPooling3D	(None, 4, 3, 10, 30)	0
Flatten	(None, 3600)	0
Dense	(None, 160)	576160
Dropout	(None, 160)	0
Dense	(None, 96)	15456
Dropout	(None, 96)	0
Dense	(None, 448)	43456
Dropout	(None, 448)	0
Dense	(None, 3)	1347
Total params		654225 (2.50 MB)
Trainable params		654225 (2.50 MB)
Non-trainable params		0 (0.00 Byte)

Table B.23: Design 2: Hyperparameters of Model 5

Parameter	**Value**
Model ID	5
Kernel Size (Layer 0)	(1, 5, 3)
Stride Size (Layer 0)	(1, 1, 1)
Convolution Filters (Layer 0)	32
Pool Size (Layer 0)	(3, 3, 3)
Kernel Size (Layer 1)	(1, 2, 4)
Stride Size (Layer 1)	(2, 2, 1)
Convolution Filters (Layer 1)	14
Pool Size (Layer 1)	(3, 3, 3)
Dense Layers	4
Units in Dense Layer 0	160
Dropout in Dense Layer 0	0.1
Units in Dense Layer 1	96
Dropout in Dense Layer 1	0.4
Units in Dense Layer 2	352
Dropout in Dense Layer 2	0.4
Units in Dense Layer 3	128
Dropout in Dense Layer 3	0.3
Learning Rate	1.88×10^{-4}
Average Accuracy	72.34%

Table B.24: Design 2: Neural Network Architecture for Model 5

Layer (type)	**Output Shape**	**Param #**
Conv3D	(None, 200, 206, 222, 32)	512
MaxPooling3D	(None, 66, 68, 74, 32)	0
Conv3D	(None, 33, 34, 71, 14)	3598
MaxPooling3D	(None, 11, 11, 23, 14)	0
Flatten	(None, 38962)	0
Dense	(None, 160)	6234080
Dropout	(None, 160)	0
Dense	(None, 96)	15456
Dropout	(None, 96)	0
Dense	(None, 352)	34144
Dropout	(None, 352)	0
Dense	(None, 128)	45184
Dropout	(None, 128)	0
Dense	(None, 3)	387
Total params		6333361 (24.16 MB)
Trainable params		6333361 (24.16 MB)
Non-trainable params		0 (0.00 Byte)

Table B.25: Design 2: Hyperparameters of Model 6

Parameter	**Value**
Model ID	6
Kernel Size (Layer 0)	(6, 4, 4)
Stride Size (Layer 0)	(1, 1, 1)
Convolution Filters (Layer 0)	16
Pool Size (Layer 0)	(4, 4, 4)
Kernel Size (Layer 1)	(2, 3, 2)
Stride Size (Layer 1)	(2, 3, 2)
Convolution Filters (Layer 1)	16
Pool Size (Layer 1)	(4, 4, 4)
Dense Layers	3
Units in Dense Layer 0	224
Dropout in Dense Layer 0	0.1
Units in Dense Layer 1	192
Dropout in Dense Layer 1	0.2
Units in Dense Layer 2	384
Dropout in Dense Layer 2	0.2
Learning Rate	2.15×10^{-3}
Average Accuracy	39.48%

Table B.26: Design 2: Neural Network Architecture for Model 6

Layer (type)	**Output Shape**	**Param #**
Conv3D	(None, 195, 207, 221, 16)	1552
MaxPooling3D	(None, 48, 51, 55, 16)	0
Conv3D	(None, 24, 17, 27, 16)	3088
MaxPooling3D	(None, 6, 4, 6, 16)	0
Flatten	(None, 2304)	0
Dense	(None, 224)	516320
Dropout	(None, 224)	0
Dense	(None, 192)	43200
Dropout	(None, 192)	0
Dense	(None, 384)	74112
Dropout	(None, 384)	0
Dense	(None, 3)	1155
Total params		639427 (2.44 MB)
Trainable params		639427 (2.44 MB)
Non-trainable params		0 (0.00 Byte)

Table B.27: Design 2: Hyperparameters of Model 7

Parameter	**Value**
Model ID	7
Kernel Size (Layer 0)	(4, 4, 6)
Stride Size (Layer 0)	(1, 1, 1)
Convolution Filters (Layer 0)	2
Pool Size (Layer 0)	(5, 5, 5)
Kernel Size (Layer 1)	(1, 2, 4)
Stride Size (Layer 1)	(2, 2, 3)
Convolution Filters (Layer 1)	12
Pool Size (Layer 1)	(5, 5, 5)
Dense Layers	3
Units in Dense Layer 0	128
Dropout in Dense Layer 0	0.2
Units in Dense Layer 1	64
Dropout in Dense Layer 1	0.0
Units in Dense Layer 2	64
Dropout in Dense Layer 2	0.0
Learning Rate	5.73×10^{-4}
Average Accuracy	68.70%

Table B.28: Design 2: Neural Network Architecture for Model 7

Layer (type)	**Output Shape**	**Param #**
Conv3D	(None, 197, 207, 219, 2)	194
MaxPooling3D	(None, 39, 41, 43, 2)	0
Conv3D	(None, 20, 20, 14, 12)	204
MaxPooling3D	(None, 4, 4, 2, 12)	0
Flatten	(None, 384)	0
Dense	(None, 128)	49280
Dropout	(None, 128)	0
Dense	(None, 64)	8256
Dropout	(None, 64)	0
Dense	(None, 64)	4160
Dropout	(None, 64)	0
Dense	(None, 3)	195
Total params		62289 (243.32 KB)
Trainable params		62289 (243.32 KB)
Non-trainable params		0 (0.00 Byte)

Bibliography

[1] N. Noshiri, M. A. Beck, C. P. Bidinosti, and C. J. Henry, "A comprehensive review of 3d convolutional neural network-based classification techniques of diseased and defective crops using non-uav-based hyperspectral images," *Smart Agricultural Technology*, vol. 5, p. 100316, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2772375523001454

[2] C. Dordas, "Role of nutrients in controlling plant diseases in sustainable agriculture. a review," *Agronomy for sustainable development*, vol. 28, pp. 33–46, 2008.

[3] S. Kumar, S. Kumar, and T. Mohapatra, "Interaction between macro-and micro-nutrients in plants," *Frontiers in Plant Science*, vol. 12, p. 665583, 2021.

[4] N. Barrow and A. E. Hartemink, "The effects of ph on nutrient availability depend on both soils and plants," *Plant and Soil*, pp. 1–17, 2023.

[5] E.-P. Lee, Y.-S. Han, S.-I. Lee, K.-T. Cho, J.-H. Park, and Y.-H. You, "Effect of nutrient and moisture on the growth and reproduction of epilobium hirsutum l., an endangered plant," *Journal of Ecology and Environment*, vol. 41, no. 1, pp. 1–9, 2017.

[6] Z. Gao, Y. Shao, G. Xuan, Y. Wang, Y. Liu, and X. Han, "Real-time hyperspectral imaging for the in-field estimation of strawberry ripeness with deep learning," *Artificial Intelligence in Agriculture*, vol. 4, pp. 31–38, 2020.

[7] S.-Y. Chen, M.-F. Chiu, and X.-W. Zou, "Real-time defect inspection of green coffee beans using nir snapshot hyperspectral imaging," *Computers and Electronics in Agriculture*, vol. 197, p. 106970, 2022.

[8] D.-W. Sun, *Hyperspectral imaging for food quality analysis and control*. London: Elsevier, 2010.

[9] W.-H. Su and D.-W. Sun, "Multispectral imaging for plant food quality analysis and visualization," *Comprehensive reviews in food science and food safety*, vol. 17, no. 1, pp. 220–239, 2018.

[10] V. Henrich, G. Krauss, C. Götze, and C. Sandow, "Idb-https://www.indexdatabase.de/, entwicklung einer datenbank für fernerkundungsindizes. AK Fernerkundung," 2012.

[11] J. A. Benediktsson and P. Ghamisi, *Spectral-spatial classification of hyperspectral remote sensing images*, 1st ed. Artech House, 2015.

[12] Y. Tarabalka, J. A. Benediktsson, J. Chanussot, and J. C. Tilton, "Multiple spectral–spatial classification approach for hyperspectral data," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 48, no. 11, pp. 4122–4132, 2010.

[13] D. Saha and A. Manickavasagan, "Machine learning techniques for analysis of hyperspectral images to determine quality of food products: A review," *Current Research in Food Science*, vol. 4, pp. 28–44, 2021.

[14] C.-C. Yang, S. O. Prasher, J. Whalen, and P. K. Goel, "Use of hyperspectral imagery for identification of different fertilisation methods with decision-tree technology," *Biosystems Engineering*, vol. 83, pp. 291–298, 2002.

[15] S. Ye, H. Weng *et al.*, "Identification of grapefruit black spot based on hyperspectral imaging using naïve-bayes classifier," *Agricultural Mechanization, Electrification and Automation*, vol. 1, no. 1, pp. 1–9, 2022.

[16] R. Agarwal, S. Hariharan, M. N. Rao, and A. Agarwal, "Weed identification using k-means clustering with color spaces features in multi-spectral images taken by uav," in *2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS*. IEEE, 2021, pp. 7047–7050.

[17] S. Mahesh, D. Jayas, J. Paliwal, and N. White, "Hyperspectral imaging to classify and monitor quality of agricultural materials," *Journal of Stored Products Research*, vol. 61, pp. 17–26, 2015.

[18] J. Gao, D. Nuyttens, P. Lootens, Y. He, and J. G. Pieters, "Recognising weeds in a maize crop using a random forest machine-learning algorithm and near-infrared snapshot mosaic hyperspectral imagery," *Biosystems engineering*, vol. 170, pp. 39–50, 2018.

[19] H.-Q. Dang, I. Kim, B.-K. Cho, and M. S. Kim, "Detection of bruise damage of pear using hyperspectral imagery," in *2012 12th International Conference on Control, Automation and Systems*. IEEE, 2012, pp. 1258–1260.

[20] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[21] S. J. Leghari, N. A. Wahocho, G. M. Laghari, A. HafeezLaghari, G. MustafaBhabhan, K. Hussain-Talpur, T. A. Bhutto, S. A. Wahocho, and A. A. Lashari, "Role of nitrogen for plant growth and development: A review," *Advances in Environmental Biology*, vol. 10, no. 9, pp. 209–219, 2016.

[22] D. Pitchay, "Towards a healthy plant: Lettuce," accessed on October 03, 2023. [Online]. Available: https://www.utoledo.edu/nsm/psrc/growers/pdf/Towards_A_Healthy_Plant-lettuc.pdf

[23] J. Wang, C. Shen, N. Liu, X. Jin, X. Fan, C. Dong, and Y. Xu, "Non-destructive evaluation of the leaf nitrogen concentration by in-field visible/near-infrared spectroscopy in pear orchards," *Sensors*, vol. 17, no. 3, p. 538, 2017.

[24] Z. Chen, S. Ren, R. Qin, and P. Nie, "Rapid detection of different types of soil nitrogen using near-infrared hyperspectral imaging," *Molecules*, vol. 27, no. 6, p. 2017, 2022.

[25] S. P. Yousuf, P.Y. and K. E. Hakeem, *Advances in Plant Nitrogen Metabolism (1st ed.)*. CRC Press, 2022.

[26] K. Goyal, N. Singh, S. Jindal, R. Kaur, A. Goyal, and R. Awasthi, "Kjeldahl method," *Adv. Tech. Anal. Chem*, vol. 1, p. 105, 2022.

[27] A. Sader, S. Oliveira, and T. Berchielli, "Application of kjeldahl and dumas combustion methods for nitrogen analysis," *Archives of Veterinary Science*, vol. 9, no. 2, 2004.

[28] J. V. Sinfield, D. Fagerman, and O. Colic, "Evaluation of sensing technologies for on-the-go detection of macro-nutrients in cultivated soils," *Computers and Electronics in Agriculture*, vol. 70, no. 1, pp. 1–18, 2010.

[29] L. Wan, W. Zhou, Y. He, T. C. Wanger, and H. Cen, "Combining transfer learning and hyperspectral reflectance analysis to assess leaf nitrogen concentration across different plant species datasets," *Remote Sensing of Environment*, vol. 269, p. 112826, 2022.

[30] G. D. Cruz, "Nitrogen deficiency mobile application for rice plant through image processing techniques," *International Journal of Engineering and Advanced Technology*, vol. 8, no. 6, pp. 2950–2955, 2019.

[31] L. Shou, L. Jia, Z. Cui, X. Chen, and F. Zhang, "Using high-resolution satellite imaging to evaluate nitrogen status of winter wheat," *Journal of plant nutrition*, vol. 30, no. 10, pp. 1669–1680, 2007.

[32] K. Berger, J. Verrelst, J.-B. Féret, Z. Wang, M. Wocher, M. Strathmann, M. Danner, W. Mauser, and T. Hank, "Crop nitrogen monitoring: Recent progress and principal developments in the context of imaging spectroscopy missions," *Remote Sensing of Environment*, vol. 242, p. 111758, 2020.

[33] Y. Li, D. Chen, C. Walker, and J. Angus, "Estimating the nitrogen status of crops using a digital camera," *Field Crops Research*, vol. 118, no. 3, pp. 221–227, 2010.

[34] K.-J. Lee and B.-W. Lee, "Estimation of rice growth and nitrogen nutrition status using color digital camera image analysis," *European Journal of Agronomy*, vol. 48, pp. 57–65, 2013.

[35] H. Tavakoli and R. Gebbers, "Assessing nitrogen and water status of winter wheat using a digital camera," *Computers and Electronics in Agriculture*, vol. 157, pp. 558–567, 2019.

[36] P. Shi, Y. Wang, J. Xu, Y. Zhao, B. Yang, Z. Yuan, and Q. Sun, "Rice nitrogen nutrition estimation with rgb images and machine learning methods," *Computers and Electronics in Agriculture*, vol. 180, p. 105860, 2021.

[37] E. Hoffland, M. Dicke, W. Van Tintelen, H. Dijkman, and M. L. Van Beusichem, "Nitrogen availability and defense of tomato against two-spotted spider mite," *Journal of Chemical Ecology*, vol. 26, pp. 2697–2711, 2000.

[38] N. Vigneau, M. Ecarnot, G. Rabatel, and P. Roumet, "Potential of field hyperspectral imaging as a non destructive method to assess leaf nitrogen content in wheat," *Field Crops Research*, vol. 122, no. 1, pp. 25–31, 2011.

[39] N. Tremblay, Z. Wang, B.-L. Ma, C. Belec, and P. Vigneault, "A comparison of crop data measured by two commercial sensors for variable-rate nitrogen application," *Precision Agriculture*, vol. 10, pp. 145–161, 2009.

[40] T. Shaver, R. Khosla, and D. Westfall, "Evaluation of two crop canopy sensors for nitrogen variability determination in irrigated maize," *Precision Agriculture*, vol. 12, pp. 892–904, 2011.

[41] L. Cotrozzi and J. J. Couture, "Hyperspectral assessment of plant responses to multi-stress environments: Prospects for managing protected agrosystems," *Plants, People, Planet*, vol. 2, no. 3, pp. 244–258, 2020.

[42] J. Wu, D. Wang, C. J. Rosen, and M. E. Bauer, "Comparison of petiole nitrate concentrations, spad chlorophyll readings, and quickbird satellite imagery in detecting nitrogen status of potato canopies," *Field crops research*, vol. 101, no. 1, pp. 96–103, 2007.

[43] S. Graeff, J. Pfenning, W. Claupein, and H.-P. Liebig, "Evaluation of image analysis to determine the n-fertilizer demand of broccoli plants (brassica oleracea convar. botrytis var. italica)." *Advances in optical technologies*, 2008.

[44] Y. Kim, J. F. Reid, and Q. Zhang, "Fuzzy logic control of a multispectral imaging sensor for in-field plant sensing," *Computers and electronics in agriculture*, vol. 60, no. 2, pp. 279–288, 2008.

[45] A. Mercado-Luna, E. Rico-García, A. Lara-Herrera, G. Soto-Zarazúa, R. Ocampo-Velázquez, R. Guevara-González, G. Herrera-Ruiz, and I. Torres-Pacheco, "Nitrogen determination on tomato (lycopersicon esculentum mill.) seedlings by color image analysis (rgb)," *African Journal of Biotechnology*, vol. 9, no. 33, 2010.

[46] S. Eshkabilov, J. Stenger, E. N. Knutson, E. Küçüktopcu, H. Simsek, and C. W. Lee, "Hyperspectral image data and waveband indexing methods to estimate nutrient concentration on lettuce (lactuca sativa l.) cultivars," *Sensors*, vol. 22, no. 21, p. 8158, 2022.

[47] P. Geladi and B. R. Kowalski, "Partial least-squares regression: a tutorial," *Analytica chimica acta*, vol. 185, pp. 1–17, 1986.

[48] H. Abdi and L. J. Williams, "Partial least squares methods: partial least squares correlation and partial least square regression," *Computational Toxicology: Volume II*, pp. 549–579, 2013.

[49] F. Kherif and A. Latypova, "Chapter 12 - principal component analysis," in *Machine Learning*, A. Mechelli and S. Vieira, Eds. Academic Press, 2020, pp. 209–225. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128157398000122

[50] S. Eshkabilov, A. Lee, X. Sun, C. W. Lee, and H. Simsek, "Hyperspectral imaging techniques for rapid detection of nutrient content of hydroponically grown lettuce cultivars," *Computers and Electronics in Agriculture*, vol. 181, p. 105968, 2021.

[51] J. Sun, A. Wei, H. Mao, X. Wu, X. Zhang, H. Gao *et al.*, "Discrimination of lettuce leaves' nitrogen status based on hyperspectral imaging technology and elm." *Nongye Jixie Xuebao= Transactions of the Chinese Society for Agricultural Machinery*, vol. 45, no. 7, pp. 272–277, 2014.

[52] S. Ding, X. Xu, and R. Nie, "Extreme learning machine and its applications," *Neural Computing and Applications*, vol. 25, pp. 549–556, 2014.

[53] S. Weksler, O. Rozenstein, and E. Ben Dor, "Continuous seasonal monitoring of nitrogen and water content in lettuce using a dual phenomics system," *Journal of Experimental Botany*, vol. 73, no. 15, pp. 5294–5305, 2022.

[54] Y. Sun, Q. Qin, H. Ren, T. Zhang, and S. Chen, "Red-edge band vegetation indices for leaf area index estimation from sentinel-2/msi imagery," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 58, no. 2, pp. 826–840, 2019.

[55] J. Gamon, L. Serrano, and J. Surfus, "The photochemical reflectance index: an optical indicator of photosynthetic radiation use efficiency across species, functional types, and nutrient levels," *Oecologia*, vol. 112, pp. 492–501, 1997.

[56] B.-C. Gao, "Normalized difference water index for remote sensing of vegetation liquid water from space," in *Imaging spectrometry*, vol. 2480. SPIE, 1995, pp. 225–236.

[57] M. F. Taha, A. I. ElManawy, K. S. Alshallash, G. ElMasry, K. Alharbi, L. Zhou, N. Liang, and Z. Qiu, "Using machine learning for nutrient content detection of aquaponics-grown plants based on spectral data," *Sustainability*, vol. 14, no. 19, p. 12318, 2022.

[58] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural networks for perception*. Elsevier, 1992, pp. 65–93.

[59] A. B. Shaik and S. Srinivasan, "A brief survey on random forest ensembles in classification model," in *International Conference on Innovative Computing and Communications: Proceedings of ICICC 2018, Volume 2*. Springer, 2019, pp. 253–260.

[60] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[61] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio, "On the number of linear regions of deep neural networks," *Advances in neural information processing systems*, vol. 27, 2014.

[62] M. Elgendy, *Deep learning for vision systems*. Simon and Schuster, 2020.

[63] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, "Activation functions in deep learning: A comprehensive survey and benchmark," *Neurocomputing*, vol. 503, pp. 92–108, 2022.

[64] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[65] Y. Ma and J. Tang, *Deep Learning on Graphs*. Cambridge University Press, 2021.

[66] J. T. Yao Ma, "Deep learning on graphs," accessed on Oct 13, 2023. [Online]. Available: https://cse.msu.edu/~mayao4/dlg_book/chapters/chapter3.pdf

[67] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, "Activation functions in deep learning: A comprehensive survey and benchmark," *Neurocomputing*, 2022.

[68] M. Hassan, "Optimization and regularization in machine learning," accessed on Oct 13, 2023. [Online]. Available: https://medium.com/@mohamed.hassan18/optimization-and-regularization-in-machine-learning-cfeff8656785

[69] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International conference on machine learning*. PMLR, 2013, pp. 1139–1147.

[70] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[71] S. Qiao, Q. Wang, J. Zhang, and Z. Pei, "Detection and classification of early decay on blueberry based on improved deep residual 3d convolutional neural network in hyperspectral images," *Scientific Programming*, vol. 2020, 2020.

[72] M. Uzair and N. Jamil, "Effects of hidden layers on the efficiency of neural networks," in *IEEE 23rd international multitopic conference (INMIC)*. IEEE, 2020, pp. 1–6.

[73] V. H. Josephine, A. Nirmala, and V. L. Alluri, "Impact of hidden dense layers in convolutional neural network to enhance performance of classification model," in *IOP Conference Series: Materials Science and Engineering*, vol. 1131, no. 1. IOP Publishing, 2021, p. 012007.

[74] H. H. Tan and K. H. Lim, "Vanishing gradient mitigation with deep learning neural network optimization," in *Proceedings of International Conference on Smart Computing & Communications (ICSCC)*. Sarawak, Malaysia: IEEE, 2019, pp. 1–4.

[75] K. Nagasubramanian, S. Jones, A. K. Singh, A. Singh, B. Ganapathysubramanian, and S. Sarkar, "Explaining hyperspectral imaging based plant disease identification: 3d cnn and saliency maps," *arXiv*, vol. abs/1804.08831, 2018.

[76] Z. Liu, J. Jiang, X. Qiao, X. Qi, Y. Pan, and X. Pan, "Using convolution neural network and hyperspectral image to identify moldy peanut kernels," *LWT*, vol. 132, p. 109815, 2020.

[77] D.-H. Jung, J. D. Kim, H.-Y. Kim, T. S. Lee, H. S. Kim, and S. Park, "A hyperspectral data 3d convolutional neural network classification model for diagnosis of gray mold disease in strawberry leaves," *Frontiers in Plant Science*, vol. 13, p. 620, 2022.

[78] Y. Cao, P. Yuan, H. Xu, J. F. Martínez-Ortega, J. Feng, and Z. Zhai, "Detecting asymptomatic infections of rice bacterial leaf blight using hyperspectral imaging and 3-dimensional convolutional neural network with spectral dilated convolution," *Frontiers in Plant Science*, vol. 13, 2022.

[79] C. Nguyen, V. Sagan, M. Maimaitiyiming, M. Maimaitijiang, S. Bhadra, and M. T. Kwasniewski, "Early detection of plant viral disease using hyperspectral imaging and deep learning," *Sensors*, vol. 21, no. 3, p. 742, 2021.

[80] C. Qi, M. Sandroni, J. C. Westergaard, E. H. R. Sundmark, M. Bagge, E. Alexandersson, and J. Gao, "In-field classification of the asymptomatic biotrophic phase of potato late blight based on deep learning and proximal hyperspectral imaging," *Computers and Electronics in Agriculture*, vol. 205, p. 107585, 2023.

[81] R. Pourdarbani, S. Sabzi, M. Dehghankar, M. H. Rohban, and J. I. Arribas, "Examination of lemon bruising using different cnn-based classifiers and local spectral-spatial hyperspectral imaging," *Algorithms*, vol. 16, no. 2, p. 113, 2023.

[82] Y. Jia, Y. Shi, J. Luo, and H. Sun, "Y–net: Identification of typical diseases of corn leaves using a 3d–2d hybrid cnn model combined with a hyperspectral image band selection module," *Sensors*, vol. 23, no. 3, p. 1494, 2023.

[83] T. Gao, A. K. N. Chandran, P. Paul, H. Walia, and H. Yu, "Hyperseed: An end-to-end method to process hyperspectral images of seeds," *Sensors*, vol. 21, no. 24, p. 8184, 2021.

[84] 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, 2017.

[85] S. Yin, Y. Wang, and Y.-H. Yang, "A novel image-dehazing network with a parallel attention block," *Pattern Recognition*, vol. 102, p. 107255, 2020.

[86] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of Computer Vision and Pattern Recognition Conference (CVPR).* Salt Lake City, UT, USA: IEEE, 2018, pp. 7132–7141.

[87] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[88] H. He, Y. Bai, E. A. Garcia, and S. Li, "Adasyn: Adaptive synthetic sampling approach for imbalanced learning," in *Proceedings of International Joint Conference on Neural Networks (IJCNN).* Hong Kong: IEEE, 2008, pp. 1322–1328.

[89] M. S. Shaikh, K. Jaferzadeh, B. Thörnberg, and J. Casselgren, "Calibration of a hyper-spectral imaging system using a low-cost reference," *Sensors*, vol. 21, no. 11, p. 3738, 2021.

[90] S. Duggal, *Surveying*, 4th ed. New Delhi: Tata McGraw-Hill Education, 2013, vol. 2.

[91] W. G. Rees, *Physical principles of remote sensing*, 3rd ed. Cambridge university press, 2013.

[92] X. Cao, B. Ji, Y. Ji, L. Wang, and L. Jiao, "Hyperspectral image classification based on filtering: a comparative study," *Journal of Applied Remote Sensing*, vol. 11, no. 3, pp. 035 007–035 007, 2017.

[93] M. Lennon, G. Mercier, and L. Hubert-Moy, "Nonlinear filtering of hyperspectral images with anisotropic diffusion," in *Proceedings of International Geoscience and Remote Sensing Symposium*, vol. 4. Toronto, ON, Canada: IEEE, 2002, pp. 2477–2479.

[94] C. Vaiphasa, "Consideration of smoothing techniques for hyperspectral remote sensing," *ISPRS journal of photogrammetry and remote sensing*, vol. 60, no. 2, pp. 91–99, 2006.

[95] M. Vidal and J. M. Amigo, "Pre-processing of hyperspectral images. essential steps before image analysis," *Chemometrics and Intelligent Laboratory Systems*, vol. 117, pp. 138–148, 2012.

[96] J. Khodr and R. Younes, "Dimensionality reduction on hyperspectral images: A comparative review based on artificial datas," in *Proceedings of International Congress on Image and Signal Processing*, vol. 4. Shanghai, China: IEEE, 2011, pp. 1875–1883.

[97] H. Fırat, M. E. Asker, and D. Hanbay, "Classification of hyperspectral remote sensing images using different dimension reduction methods with 3d/2d cnn," *Remote Sensing Applications: Society and Environment*, vol. 25, p. 100694, 2022.

[98] J. Yang, Z. Jin, and J. Yang, "Non-linear techniques for dimension reduction," in *Encyclopedia of Biometrics*. Springer, 2009, pp. 1003–1007.

[99] P. Kumar, D. K. Gupta, V. N. Mishra, and R. Prasad, "Comparison of support vector machine, artificial neural network, and spectral angle mapper algorithms for crop classification using liss iv data," *International Journal of Remote Sensing*, vol. 36, no. 6, pp. 1604–1617, 2015.

[100] J. Qin, T. F. Burks, M. A. Ritenour, and W. G. Bonn, "Detection of citrus canker using hyperspectral reflectance imaging with spectral information divergence," *Journal of food engineering*, vol. 93, no. 2, pp. 183–191, 2009.

[101] J. Avbelj, "Spectral information retrieval for sub-pixel building edge detection," *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. I-7, pp. 61–66, 2012.

[102] G. Luo, G. Chen, L. Tian, K. Qin, and S.-E. Qian, "Minimum noise fraction versus principal component analysis as a preprocessing step for hyperspectral imagery denoising," *Canadian Journal of Remote Sensing*, vol. 42, no. 2, pp. 106–116, 2016.

[103] M. Dalla Mura, J. A. Benediktsson, B. Waske, and L. Bruzzone, "Morphological attribute profiles for the analysis of very high resolution images," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 48, no. 10, pp. 3747–3762, 2010.

[104] A. Plaza, G. Martín, J. Plaza, M. Zortea, and S. Sánchez, *Recent Developments in Endmember Extraction and Spectral Unmixing*. Springer Berlin Heidelberg, 2011, pp. 235–267.

[105] L. Sun, X. Song, H. Guo, G. Zhao, and J. Wang, "Patch-wise semantic segmentation for hyperspectral images via a cubic capsule network with emap features," *Remote Sensing*, vol. 13, no. 17, p. 3497, 2021.

[106] A. Bojeri, F. Melgani, G. Giannotta, G. Ristorto, G. Guglieri, and J. M. Junior, "Automatic crop rows segmentation for multispectral aerial imagery," in *Mediterranean and Middle-East Geoscience and Remote Sensing Symposium (M2GARSS)*. IEEE, 2022, pp. 78–81.

[107] T. Boston, A. Van Dijk, P. R. Larraondo, and R. Thackway, "Comparing cnns and random forests for landsat image segmentation trained on a large proxy land cover dataset," *Remote Sensing*, vol. 14, no. 14, p. 3396, 2022.

[108] J. Li, H. Wang, A. Zhang, and Y. Liu, "Semantic segmentation of hyperspectral remote sensing images based on pse-unet model," *Sensors*, vol. 22, no. 24, p. 9678, 2022.

[109] D. Wan, R. Lu, S. Wang, S. Shen, T. Xu, and X. Lang, "Yolo-hr: Improved yolov5 for object detection in high-resolution optical remote sensing images," *Remote Sensing*, vol. 15, no. 3, p. 614, 2023.

[110] L3Harris Geospatial, "Envi," 2023. [Online]. Available: https://www.l3harrisgeospatial.com/Software-Technology/ENVI

[111] Resonon Inc, "Spectronon software," 2023. [Online]. Available: https://resonon.com/software

[112] NASA Ocean Biology Processing Group (OBPG), "Seadas," 2022. [Online]. Available: https://seadas.gsfc.nasa.gov/downloads/

[113] Centre National D'Etudes Spatiales (CNES), "Orfeo toolbox," 2023. [Online]. Available: https://www.orfeo-toolbox.org/download/

[114] P. Bunting, D. Clewley, R. M. Lucas, and S. Gillingham, "The Remote Sensing and GIS Software Library (RSGISLib)," *Computers and Geosciences*, vol. 62, pp. 216–226, 2014.

[115] X. Jin, L. Jie, S. Wang, H. J. Qi, and S. W. Li, "Classifying wheat hyperspectral pixels of healthy heads and fusarium head blight disease using a deep neural network in the wild field," *Remote Sensing*, vol. 10, no. 3, p. 395, 2018.

[116] W. Sun and Q. Du, "Hyperspectral band selection: A review," *IEEE Geoscience and Remote Sensing Magazine*, vol. 7, no. 2, pp. 118–139, 2019.

[117] S. S. Sawant and M. Prabukumar, "A survey of band selection techniques for hyperspectral image classification," *Journal of Spectral Imaging*, vol. 9, 2020.

[118] A. Ifarraguerri and M. W. Prairie, "Visual method for spectral band selection," *IEEE Geoscience and Remote Sensing Letters*, vol. 1, no. 2, pp. 101–106, 2004.

[119] L. Wang, X. Jia, and Y. Zhang, "A novel geometry-based feature-selection technique for hyperspectral imagery," *IEEE Geoscience and Remote Sensing Letters*, vol. 4, no. 1, pp. 171–175, 2007.

[120] L. C. B. D. Santos, S. J. F. Guimaraes, and J. A. D. Santos, "Efficient unsupervised band selection through spectral rhythms," *Journal of Selected Topics in Signal Processing*, vol. 9, no. 6, pp. 1016–1025, 2015.

[121] P. Ghamisi, M. S. Couceiro, and J. A. Benediktsson, "A novel feature selection approach based on fodpso and svm," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 53, no. 5, pp. 2935–2947, 2014.

[122] A. Shi, H. Gao, Z. He, M. Li, and L. Xu, "A hyperspectral band selection based on game theory and differential evolution algorithm," *International Journal on Smart Sensing and Intelligent Systems*, vol. 9, no. 4, pp. 1971–1990, 2016.

[123] K. Sun, X. Geng, and L. Ji, "An efficient unsupervised band selection method based on an autocorrelation matrix for a hyperspectral image," *International Journal of Remote Sensing*, vol. 35, no. 21, pp. 7458–7476, 2014.

[124] T. Imbiriba, J. C. M. Bermudez, C. Richard, and J.-Y. Tourneret, "Band selection in rkhs for fast nonlinear unmixing of hyperspectral images," in *Proceedings of European Signal Processing Conference (EUSIPCO)*. Nice, France: IEEE, 2015, pp. 1651–1655.

[125] C. Yang, Y. Tan, L. Bruzzone, L. Lu, and R. Guan, "Discriminative feature metric learning in the affinity propagation model for band selection in hyperspectral images," *Remote Sensing*, vol. 9, no. 8, p. 782, 2017.

[126] B. Mojaradi, H. Emami, M. Varshosaz, and S. Jamali, "A novelband selection methodfor hyperspectral data analysis," *Proceedings of International Archives of Photogrammry, Remote Sensing and Spatial Information Sciences*, vol. 447, p. 451, 2008.

[127] H. Su, H. Yang, Q. Du, and Y. Sheng, "Semisupervised band clustering for dimensionality reduction of hyperspectral imagery," *IEEE Geoscience and Remote Sensing Letters*, vol. 8, no. 6, pp. 1135–1139, 2011.

[128] H. Su, Y. Sheng, P. Du, and K. Liu, "Adaptive affinity propagation with spectral angle mapper for semi-supervised hyperspectral band selection," *Applied optics*, vol. 51, no. 14, pp. 2656–2663, 2012.

[129] J.-m. Li and Y.-t. Qian, "Clustering-based hyperspectral band selection using sparse nonnegative matrix factorization," *Journal of Zhejiang University Science C*, vol. 12, no. 7, pp. 542–549, 2011.

[130] H. Zhai, H. Zhang, L. Zhang, and P. Li, "Squaring weighted low-rank subspace clustering for hyperspectral image band selection," in *Proceedings of International Geoscience and Remote Sensing Symposium (IGARSS)*. Beijing, China: IEEE, 2016, pp. 2434–2437.

[131] W. Sun, L. Tian, Y. Xu, D. Zhang, and Q. Du, "Fast and robust self-representation method for hyperspectral band selection," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 10, no. 11, pp. 5087–5098, 2017.

[132] K. Sun, X. Geng, and L. Ji, "A new sparsity-based band selection method for target detection of hyperspectral image," *IEEE Geoscience and Remote Sensing Letters*, vol. 12, no. 2, pp. 329–333, 2014.

[133] B. B. Damodaran, N. Courty, and S. Lefèvre, "Sparse hilbert schmidt independence criterion and surrogate-kernel-based feature selection for hyperspectral image classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, no. 4, pp. 2385–2398, 2017.

[134] J. Tschannerl, J. Ren, J. Zabalza, and S. Marshall, "Segmented autoencoders for unsupervised embedded hyperspectral band selection," in *Proceedings of European workshop on visual information processing (EUVIP)*. Tampere, Finland: IEEE, 2018, pp. 1–6.

[135] Y. Zhan, D. Hu, H. Xing, and X. Yu, "Hyperspectral band selection based on deep convolutional neural network and distance density," *IEEE Geoscience and Remote Sensing Letters*, vol. 14, no. 12, pp. 2365–2369, 2017.

[136] V. Sharma, A. Diba, T. Tuytelaars, and L. Van Gool, "Hyperspectral cnn for image classification & band selection, with application to face recognition," KU Leuven, Tech. Rep., 2016.

[137] J. Yin, Y. Wang, and Z. Zhao, "Optimal band selection for hyperspectral image classification based on inter-class separability," in *Proceedings of Symposium on Photonics and Optoelectronics*. Chengdu, China: IEEE, 2010, pp. 1–4.

[138] A. Datta, S. Ghosh, and A. Ghosh, "Combination of clustering and ranking techniques for unsupervised band selection of hyperspectral images," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 6, pp. 2814–2823, 2015.

[139] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv*, vol. abs/1312.6034, 2013.

[140] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," *International Journal of Computer Vision*, vol. 128, pp. 336–359, 2020.

[141] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, 2016, pp. 2921–2929.

[142] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, 2017.

[143] M. A. Beck, C.-Y. Liu, C. P. Bidinosti, C. J. Henry, C. M. Godee, and M. Ajmani, "An embedded system for the automated generation of labeled plant images to enable machine learning applications in agriculture," *PLOS ONE*, vol. 15, no. 12, p. e0243923, 2020.

[144] M. A. Beck, C. P. Bidinosti, C. J. Henry, and M. Ajmani, "The terrabyte client: providing access to terabytes of plant data," *ArXiv*, vol. abs/2203.13691, 2022.

[145] O. Hamila, C. J. Henry, O. I. Molina, C. P. Bidinosti, and M. A. Henriquez, "Fusarium head blight detection, spikelet estimation, and severity assessment in wheat using 3d convolutional neural networks," *arXiv*, vol. abs/2303.05634, 2023.

[146] N. Noshiri, M. Khorramfar, and T. Halabi, "Machine learning-as-a-service performance evaluation on multi-class datasets," in *International Conference on Smart Internet of Things (SmartIoT)*. IEEE, 2021, pp. 332–336.

[147] B. J. Hosticka, "Cmos sensor systems," *Sensors and Actuators A: Physical*, vol. 66, no. 1-3, pp. 335–341, 1998.

[148] "Hyperspectral imaging: calibration problems and solutions," *Chemometrics and Intelligent Laboratory Systems*, vol. 72, no. 2, pp. 209–217, 2004.

[149] K. Jacq, R. Martinez-Lamas, A. Van Exem, and M. Debret, "Hyperspectral core-logger image acquisition," 2020.

[150] A. A. Chowdhury, A. Das, K. K. S. Hoque, and D. Karmaker, "A comparative study of hyper-parameter optimization techniques for deep learning", booktitle="proceedings of international joint conference on advances in computational intelligence," M. S. Uddin, P. K. Jamwal, and J. C. Bansal, Eds. Singapore: Springer Nature Singapore, 2022, pp. 509–521.

[151] F. Nogueira, "Bayesian optimization: Open source constrained global optimization tool for python," 2014. [Online]. Available: https://github.com/fmfn/BayesianOptimization

[152] T. G. authors, "Gpyopt: A bayesian optimization framework in python," http://github.com/SheffieldML/GPyOpt, 2016.

[153] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, "Tune: A research platform for distributed model selection and training," *arXiv preprint arXiv:1807.05118*, 2018.

[154] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyper-parameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[155] J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proceedings of the 30th International Conference on Machine Learning (ICML 2013)*, June 2013, pp. I–115–I–123.

[156] T. O'Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi *et al.*, "Keras Tuner," https://github.com/keras-team/keras-tuner, 2019.

[157] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[158] D. Baymurzina, E. Golikov, and M. Burtsev, "A review of neural architecture search," *Neurocomputing*, vol. 474, pp. 82–93, 2022.

[159] D. Berrar, "Bayes' theorem and naive bayes classifier," *Encyclopedia of bioinformatics and computational biology: ABC of bioinformatics*, vol. 403, p. 412, 2018.