3D Convolutional Neural Networks for Solving Complex Digital Agriculture and Medical Imaging Problems

by

Oumaima Hamila

A thesis submitted to the Faculty of Graduate Studies of

The University of Winnipeg

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Applied Computer Science

TerraByte Research Group

University of Winnipeg

Winnipeg, Manitoba, Canada

3D Convolutional Neural Networks for Solving Complex Digital Agriculture and Medical Imaging Problems

by

Oumaima Hamila

Supervisory Committee

Dr. C.J. Henry, Supervisor

(Department of Applied Computer Science)

Dr. C. Bidinosti, Member

(Department of Applied Computer Science)

Dr. I. Jeffrey, External Member

(Department of Electrical and Computer Engineering, University of Manitoba)

Abstract

3D signals have become widely popular in view of the advantage they provide via 3D representations of data by employing a third spatial or temporal dimension to extend 2D signals. Predominantly, 3D signals contain details inexistent in their 2D counterparts such as the depth of an image, which is inherent to point clouds (PC), or the temporal evolution of an image, which is inherent to time series data such as videos. Despite this advantage, 3D models are still underexploited in machine learning (ML) compared to 2D signals, mainly due to data scarcity. In this thesis, we exploit and determine the efficiency and influence of using both multispectral PCs and time-series data with 3D convolutional neural networks (CNNs). We evaluate the performance and utility of these networks and data in the context of two applications from the areas of digital agriculture and medical imaging. In particular, multispectral PCs are investigated for the problem of fusarium-head-blight (FHB) detection and total number of spikelets estimation, while time-series echocardiography are investigated for the problem of myocardial infarction (MI) detection.

In the context of the digital agriculture application, two state-of-the-art datasets were created, namely the UW-MRDC WHEAT-PLANT PC dataset, consisting of 216 multispectral PC of wheat plants, and the UW-MRDC WHEAT-HEAD PC dataset, consisting of 80 multispectral PC of wheat heads. Both dataset samples were acquired using a multispectral 3D scanner. Moreover, a real-time parallel GPU-enabled preprocessing method, that runs 1065 times faster than its CPU counterpart, was proposed to convert multispectral PCs into multispectral 3D images compatible with CNNs. Also, the UW-MRDC WHEAT-PLANT PC dataset was used to develop novel and efficient 3D CNNs for disease detection to automatically identify wheat infected with FHB from multispectral 3D images of wheat plants. In addition, the influence of the multispectral information on the detection performance was evaluated, and our results showed the dominance of the red,

green, and blue (RGB) colour channels over both the near-infra-red (NIR) channel and RGB and NIR channels combined. Our best model for FHB detection in wheat plants achieved 100% accuracy. Furthermore, the UW-MRDC WHEAT-HEAD PC dataset was used to develop unique and efficient 3D CNNs for total number of spikelets estimation in multispectral 3D images of wheat heads, in addition to adapting three benchmark 2D CNN architectures to 3D images to achieve the same purpose. Our best model for total number of spikelets estimation in wheat head achieved 1.13 mean absolute error, meaning that, on average, the difference between the estimated number of spikelets and the actual value is equal to 1.13. Our 3D CNN for FHB detection in wheat achieved the highest accuracy amongst existing FHB detection models, and our 3D CNN for total number of spikelets estimation in wheat is a unique and pioneer application. These results suggest that replacing arduous tasks that require the input of field experts and significant temporal resources with automated ML models in the context of digital agriculture is feasible and promising.

In the context of the medical imaging application, an innovative, real-time, and fully automated pipeline based on 2D and 3D CNNs was proposed for early detection of MI, which is a deadly cardiac disorder, from a patient's echocardiography. The developed pipeline consists of a 2D CNN that performs data preprocessing by segmenting the left ventricle (LV) chamber from the apical 4-chamber (A4C) view from an echocardiogarphy, followed by a 3D CNN that performs MI detection in real-time. The pipeline was trained and tested on the HMC-QU dataset consisting of 162 echocardiography. The 2D CNN achieved 97.18% accuracy on data segmentation, and the 3D CNN achieved 90.9% accuracy, 100% precision, 95% recall, and 97.2% F1 score. Our detection results outperformed existing state-of-the-art models that were tested on the HMC-QU dataset for MI detection. Moreover, our results demonstrate that developing a fully automated system for LV segmentation and MI detection is efficient and propitious and could enable the creation of a tool that reliably suggests the presence of MI in a given echocardiography on the fly.

All the empirical results achieved in our thesis indicate the efficiency and reliability of 3D

signals, that are multispectral PCs and videos, in developing detection and regression 3D CNN models that can achieve accurate and reliable results.

Keywords: Convolutional neural networks, 3D, point cloud, multispectral, wheat, fusarium head blight, echocardiography, myocardial infarction, detection, segmentation, estimation.

Acknowledgment

I would like to express my sincerest gratitude and appreciation for Dr. Christopher Henry, whose inestimable guidance, encouragement, support, and advice have been tremendously valuable and crucial throughout my research work. Thanks to his tireless supervision, I was able to surpass myself as a researcher as well as a person.

I also would like to express my deepest gratitude to Dr. Sheela Ramanna who was very generous in providing her invaluable guidance and supervision in the making of a journal paper.

Many thanks to Dr. Maria Antonia Henriquez, Otto Gruenke, and Debbie Miranda for their amazing collaborative work that contributed tremendously to the making of this thesis.

Thank you to Eric Benson, Ferdinand Borillo, Dylan Jones, and Connie Arnhold for their constant availability to provide me with the best student service and kindest answers.

I would like to express my gratitude to Mitacs, EMILI, NSERC, Western Economic Diversification Canada, and The Faculty of Graduate Studies, whose financial support during my studies enabled me to devote my energy to research.

Thank you to the committee members, Dr. Christopher Bidinosti and Dr. Ian Jeffrey, for taking the time to read and comment on my thesis.

Lastly, I would like to thank my parents and brother whose love, support, and trust have provided me with the will and strength to pursue my ambitions and thank you to my husband for his tireless kindness and support.

Contents

List of Tables

List of Figures

Dedication

To my family and husband.

Chapter 1

Introduction

Artificial intelligence (AI) was developed to create machines that can imitate the human way of active thinking and problem solving. In fact, the field of digital image processing (DIP) allowed the development of advanced AI algorithms throughout the years in order to speed up and improve the outcome of complex DIP applications such as pattern recognition and detection [1]. Traditionally, DIP was based on analytical methods such as Fourier analysis, linear filtering, and histogram estimation [2]. These were used for a variety of DIP applications, ranging from image transformation (*e.g.* denoising and colour enhancement) to pattern recognition (*e.g.* detection of facial features and fingerprint recognition). However, in order to obtain efficient results for a given application, there was a need to develop a deep theoretical understanding of the problem that is tackled. This required the knowledge of experts and was a time-consuming process. For instance, in the case of cloud detection in satellite images, classical methods such as histogram estimation [3] and varying thresholding [4] were employed to extract relevant features for manual engineering to identify relationships conducive to maximizing detection accuracy.

In order to minimize manual feature analysis, machine learning (ML) models such as support-vector machines (SVM) [5] and decision trees [6] have been used to automatically create a

relationship between the extracted features by learning from a significant number of examples [7, 8]. Nevertheless, these ML models still relied on mandatory feature extraction performed with classical DIP methods which were sensitive to the image acquisition contexts such as lighting, contrast, or background. Moreover, the detection could only be effective under specific conditions that may not always be met in practice such as high radiometric resolutions [3].

These limitations motivated researchers to exploit models that can imitate human traits of analyzing data and that can produce accurate results for real-world problems. In fact, these models were inspired from the investigation of the human brain, consisting of millions to billions of neurons that communicate with each other via neurotransmitters known as synapses, and the observation that the more knowledge and experience acquired by a person, the more synaptic links are created between neurons to carry valuable information [9]. Predominantly, one of these models is the multi-layer perceptron (MLP), which applies the latter mechanism in a way that produces meaningful features by learning from a set of given examples. With MLPs, features are extracted automatically using the backpropagation algorithm by exposing the model to a large set of image examples [10]. This produces features that are highly robust to noise which enables the generation of highly accurate and efficient results compared to classical models [11].

However, MLPs can only take feature vectors as input which makes them unsuitable to process 2D images directly. In fact, the spatial and spectral relationships are lost when images are transformed into vectors. Therefore, convolutional neural networks (CNNs) gained attraction due to weight sharing which enabled the automatic extraction of features that consider spatial relationships such as orientation and spatial distribution. Moreover, they became computationally practical due to advances in the computational power of computers, specifically platforms that support parallel programming, and in graphics processing units (GPUs) that support the simultaneous execution of thousands, if not millions, of floating-point instructions. Similarly, advancements of imaging tools and sensors allowed the creation of image datasets consisting of thousands of samples that were

used in training CNN models. For example, medical imaging is commonly employed in medicine to produce images such as CT scans and x-ray, which allowed the development of automated applications such as brain tumor detection from MRI images [12] and breast cancer detection using CNNs [13]. Furthermore, hyperspectral and multispectral imaging have become widely used in agriculture to create plant datasets in order to develop automated tools such as weed detection among crops [14] and plant disease detection [15]. Since CNNs are based on convolution, they are able to process images directly and extract features that can be used to issue a decision when fed to ML model such as MLP or SVM.

Although CNNs proved efficient in analyzing 2D signals, they are yet to be widely used on 3D signals. In fact, 3D signals contain details that can not be represented with 2D signals, such as the depth of an image which is inherent to point clouds (PCs), and the temporal evolution of an image which is inherent to time series data (such as videos). Therefore, in this thesis, we tackled two problems involving the two aforementioned types of 3D signals namely "Detection and Estimation in Point Clouds of Wheat" and "Myocardial Infarction Detection in Echocardiography", and we set out to determine if 3D signals allow the development of efficient and accurate 3D CNN models.

1.1 Detection and Estimation in Point Clouds of Wheat

1.1.1 Problem Statement

Fusarium Head Blight (FHB) is a devastating fungal disease caused by the fungal genus Fusarium. It mainly affects wheat, but it can also spread through other cereals like barley, oat, and corn [16]. The disease starts infecting spikelets[1] [17] during the flowering stage, causing a deficiency in the plant development and a premature grain shivering and bleaching. Moreover, in some specific weather conditions, a mycotoxin called deoxynivalenol may be triggered in the infected kernels and

[1]A spike consists of a number of spikelets, and a spikelet consists of a number of grains (2 to 3 in most cases).

cause acute toxicity to humans [18]. The development of FHB is favoured by wet, moist, and warm weather conditions. Also, wind and rainfalls are major sources of seed and crop contamination. The disease is considered a serious problem that severely decreases yield quantity and quality and reduces wheat production. In Canada, farmers have been dealing with heavy yield losses and extreme crop damage in wheat for decades, yet, FHB is still a problem to date [19].

To help reduce the impact of FHB, many practices and management strategies are adapted by farmers and researchers. Examples include using multiple varieties of wheat seeds that are known to be resistant to the disease, using seeds that are unlikely infected with FHB, or controlling the irrigation frequencies and durations during the flowering stage. This latter method aims to control humidity conditions that favor the spread of FHB. Among all these practices, using resistant wheat seed varieties seems the most effective strategy to limit the spread of the disease. Thus, the Morden Research and Development Centre (MRDC) [20], which is one of Agriculture and Agri-Food Canada's 20 research and development centres, performs research on crop pathologies, genomics, proteomics and, in particular, conducts research on FHB in wheat. One of the main goals of the centre is to develop wheat varieties that are resistant to FHB. In doing so, multiple wheat varieties are annually seeded, grown, inoculated with the fungus, and tested on their resistance level. Their ability to resist FHB is quantified based on a severity index (SI) that is determined by the ratio of the total number of infected spikelets in a wheat head to the total number of spikelets in the same wheat head. Figure 1.1 shows an example of the SI estimation in a FHB-infected wheat head, where the total number of FHB-infected spikelets is 9, and the total number of spikelets in the same wheat head is 15, which results in a 60% severity in the wheat head. This index is usually determined visually by human agents that estimate the approximate value of the SI of a wheat plant based on a subjective observation. However, determining the SI on a daily basis for thousands of wheat plants grown in a vast wheat field or even in a small growth chamber is a very time-consuming task that involves multiple experts and requires high levels of expertise,

concentration, and accuracy. Therefore, experts emphasize the need to develop an automated tool to detect whether a specific wheat plant is infected with FHB and, if so, automatically estimate its SI. Consequently, a collaboration between our TerraByte[2] research group and the MRDC was established to create a bridge between agriculture and ML, in order to develop efficient and automated 3D models for FHB detection and SI estimation in wheat. However, the scope of this thesis covers the automated detection of FHB and the first part of the SI estimation, which is the automated estimation of the total number of spikelets in a wheat head.

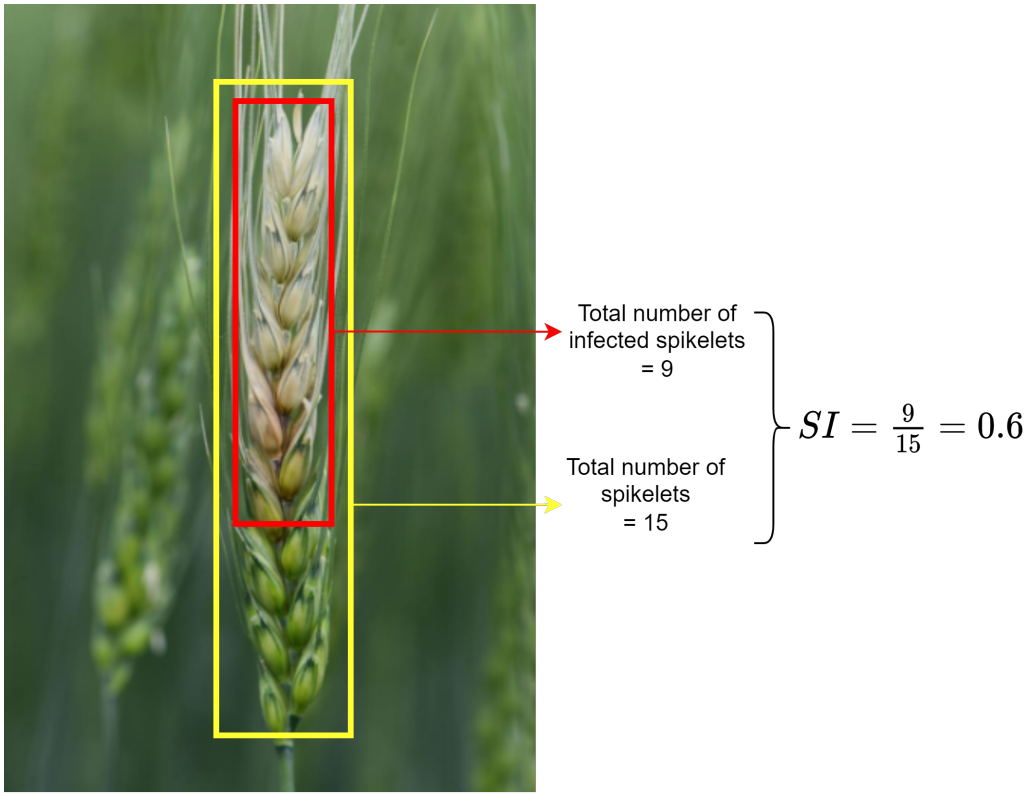


Figure 1.1: Example of the severity index estimation in a FHB-infected wheat head [21].

1.1.2 Proposed Approach

In this thesis, we propose novel datasets and methods to overcome the following issues:

[2]terrabyte.acs.uwinnipeg.ca

- unavailability of datasets consisting of PCs of wheat plants,
- unavailability of datasets consisting of PCs of wheat heads,
- unavailability of datasets of wheat PCs defined by multispectral information,
- incompatible PC representation with CNNs,
- manual, time-consuming, and subjective detection of FHB in wheat, and
- manual, time-consuming, and costly manual estimation of the total number of spikelets on a wheat head.

Due to the lack of FHB diseased wheat datasets, we proposed two novel datasets of multispectral PCs, namely the UW-MRDC WHEAT-PLANT PC dataset for FHB detection in a wheat plant and the UW-MRDC WHEAT-HEAD PC dataset for the estimation of the total number of spikelets per wheat head. The multispectral colour information is defined by the red (R), green (G), blue (B) (*i.e.* RGB), and near-infra-red (NIR) channels. Each sample within both datasets is a multispectral PC that was acquired by a multispectral 3D scanner. The UW-MRDC WHEAT-PLANT PC dataset contained 216 samples, from which 42 represent FHB-infected samples and the remaining 174 represent water-controlled (WC or healthy) samples, and the UW-MRDC WHEAT-HEAD PC dataset contained 80 samples. Moreover, a highly efficient and fast parallel CUDA algorithm that runs on GPU was introduced to convert multispectral PCs into multispectral 3D images with varying resolution factors. The conversion was necessary because PC representation is incompatible with 3D CNNs. Following that, we performed a grid search to find the best 3D CNN architectures and the most efficient multispectral information to perform FHB detection in multispectral 3D images of wheat plants. These best performing detection models were trained and evaluated on the UW-MRDC WHEAT-PLANT PC dataset. Also, we performed a grid search to find the best 3D CNN architectures to perform total number of spikelets estimation in multispectral 3D images

of wheat heads, in addition to adapting three well-know 2D CNN architectures to 3D images to achieve the same purpose. These best performing regression models were trained and evaluated on the UW-MRDC WHEAT-HEAD PC dataset. Finally, a set of the best performing 3D CNN architectures for both FHB detection and total number of spikelets estimation were evaluated and compared in terms of their accuracy, inference time, and number of parameters to determine the most efficient 3D CNN for each problem as well as the most representative multispectral information.

1.2 Myocardial Infarction Detection in Echocardiography

1.2.1 Problem Statement

Early detection of myocardial infarction (MI) [22], which is colloquially referred to as a heart attack, can prevent a patient from enduring several extreme health complications such as heart failure or sudden death [23, 24]. In fact, MI is pathologically defined as the death of the myocardial cells due to extended cardiac ischemia which, in turn, is defined as an abrupt and prolonged limitation of blood supply to the heart muscles [25]. Once a patient is suspected with MI, an immediate and accurate diagnosis should be performed to detect early and basic symptoms associated with the disease. Predominantly, a common symptom is an abnormal or a non-uniform motion, known as hypokinesia, of one or several regional sections of the left ventricle (LV) wall of the heart [26]. As soon as regional wall motion abnormalities (RWMA) of the LV are perceived [27, 28], the process of infarction can be completely aborted within the first hour [29] and the affected patient can avoid serious or fatal health complications [30]. However, the major constraints to an accurate diagnosis are the unavailability of an end-to-end, rapid, and exact assessment tool to reliably detect MI in real-time.

During the last two decades, non-invasive imaging for cardiovascular disease diagnosis and monitoring have witnessed an important evolution [31, 32, 33] that enabled cardiologists to further develop their understanding of cardiac pathologies and, in particular, benefitted MI analysis, identification, and treatment [34]. To detect and assess RWMA of the cardiac chambers, echocardiography is highly recommended by The American Society of Echocardiography because of its capability to assess, in real-time, both the cardiac function and structure [35]. It generates important amounts of visual data including the size and shape of the heart and its chambers and the motion of the heart walls while they are beating. This helps cardiologists to identify RWMA in a patient's echocardiography and assign the adequate treatment immediately [36], which may minimize the damage on the cardiac muscle tissues and prevent patients from facing death [37].

Some works in early detection of MI in echocardiography used ML and CNNs [38, 39, 40], while some others were based on classical approaches such as metaheuristics [41] and Fourier tracking [42]. Some methods either heavily rely on very specific and limited conditions of data acquisition (high-resolution echocardiograms, high frame-rate, minimal noise) [43], that require the technician or the cardiologist to perform preliminary preprocessing steps to be able to proceed with the prediction process [44], or extract finite and restricted features to be used with the classification model.

Even though echocardiography is an ideal tool to detect RWMA during a myocardial ischemia, some cardiologists find it challenging to use as a primary diagnosis tool and often employ other diagnosis methods to determine the disease, such as electrocardiogram or angiogram [45], due to their straightforwardness or simplicity to use and interpret. For instance, echocardiography produces large and complex data that needs to be entirely exploited and understood in order to make a complete diagnosis based on visual interpretation [46], which is highly dependent on the level of experience of the cardiologist in question [47]. Moreover, in some cases, an important amount of the generated data remains unused due to insufficient time and difficulty in interpretation [48].

Furthermore, data acquisition is usually performed in emergencies, which often yields images of low quality [49, 50], that may also be of a low-resolution because of the cardiac machine characteristics itself. As a result, these constraints negatively impact the accuracy of the MI diagnosis [51]. Thus, there is a need to create an advanced, reliable, and fully-automated process that efficiently uses echocardiography to perform accurate MI detection in real-time [52, 43].

1.2.2 Proposed Approach

In this thesis, we propose a novel method to overcome the following issues:

- subjective reading of the data that relies on expert cardiologists,
- generated poor-quality and low-resolution echocardiography,
- massive amounts of video data that requires preprocessing prior to detection, and
- slow, manual, and inefficient MI detection.

The proposed solution is an end-to-end and fully automated pipeline consisting of a 2D CNN that performs data preprocessing followed by a 3D CNN that performs binary classification to detect MI from an echocardiography in real-time. The pipeline begins with a 2D CNN that segments the LV region from an echocardiography, since the occurrence of MI is highly correlated with RWMA of the LV walls [26]. Then, the segmented video is fed to a 3D CNN, which extracts from it the relevant spatio-temporal features and uses them to detect MI. The input of the pipeline is an unprocessed echocardiography of a patient as acquired by a technician or a cardiologist, and the output is the detection result, which is either abnormal (MI) or normal (N). Both 2D and 3D CNNs were trained and tested on the HMC-QU benchmark dataset [53], which contains 162 4-chamber view echocardiography recordings obtained at the Hamad Medical Corporation (HMC) [54] between 2018 and 2019 and approved for scientific use in February 2019. The echocardiography

videos represent 93 patients that were diagnosed with MI, while the remaining 69 videos represent normal patients.

1.3 Thesis Contributions

The work presented in this thesis led to the following journal paper:

- "Fully Automated 2D and 3D Convolutional Neural Networks Pipeline for Video Segmentation and Myocardial Infarction Detection in Echocardiography", Multimedia Tools and Applications, Springer (Accepted on 09-21-2021 and not yet published). [55]

The main contributions of this thesis are the following:

- A novel labelled dataset of multispectral PCs of wheat plants consisting of both healthy and FHB-diseased samples for the FHB detection in wheat.
- A novel labelled dataset of multispectral PCs of wheat heads for the estimation of the total number of spikelets per wheat head.
- A real-time CUDA-based preprocessing model for the conversion of multispectral PC into multispectral 3D image that runs, on average, 1065 times faster than its CPU counterpart.
- An accurate, reliable, and real-time FHB detection model on multispectral 3D images that achieved 100% accuracy on the UW-MRDC WHEAT-PLANT PC dataset.
- An efficient and real-time model for estimating the total number of spikelets in a wheat head on multispectral 3D images that achieved a 1.13 mean absolute error (MAE) on the UW-MRDC WHEAT-HEAD PC dataset.

- The empirical determination of the most important spectral information for FHB detection with CNNs.

- A fully automated pipeline for video segmentation and MI detection in echocardiography.

- An indiscriminative pipeline that processes videos of different sizes, different frame rates, and different resolutions.

- An early and real-time MI detection model in echocardiography.

- A robust pipeline that operates on low-quality videos corrupted with intense noise.

- A system for LV segmentation in echocardiography that achieved 97.18% accuracy on the HMC-QU benchmark dataset.

- A system for MI early detection in echocardiography that achieved 90.9% accuracy, 100% precision, and 95% recall on the HMC-QU benchmark dataset.

- A lightweight system that runs on parallel threads and does not require high memory or computational power in order to be executed.

- A system that outperformed state-of-the-art methods in MI detection on the HMC-QU benchmark dataset.

1.4 Thesis Outline

Chapter 2 provides a review of the most recent works related to Chapter 4 and Chapter 5, such as a review of the applications of 3D CNNs on FHB detection and automatic counting of wheat spikelets, multispectral 3D wheat datasets, and MI detection in echocardiography. Chapter 3 gives a detailed description of CNNs. It describes the theory, the building blocks, and the concepts

behind a CNN model, and presents three benchmark CNN architectures. Chapter 4 provides a general overview of the methodologies developed while creating the datasets, a detailed explanation of the theory and implementation of the multispectral 3D PC to multispectral 3D image conversion method, and a detailed description of the data preparation, acquisition, and labelling related to the creation of the two datasets. It also provides a detailed explanation of the creation process, the experiments performed to determine the best model architectures, and the results related to both FHB detection and number of spikelets estimation models. Chapter 5 gives an overview on the methodology followed during the creation of the fully automated 2D and 3D CNN models for MI detection in echocardiography, followed by detailed explanation of video segmentation with a 2D CNN and MI detection with a 3D CNN. Then, a description of each model's experiments and results is given. Chapter 6 provides conclusions related to Chapter 4 and Chapter 5 and gives an overview on their corresponding future works.

Chapter 2

Literature Review

2.1 Detection and Estimation in Point Clouds of Wheat

The technological advances in multispectral and hyperspectral imaging, remote sensing, and 3D imaging that occurred in agriculture during recent years [56, 57] have led to the development of advanced 2D and 3D acquisition systems such as drones and 3D scanners that are used to create 3D image datasets of plants and crops [58]. Meanwhile, ML has drastically evolved due to the advance of computing power, the availability of large labelled datasets, and new algorithms with many more parameters than were previously computationally possible [59]. For these reasons, many advanced applications in agriculture were created such as yield monitoring and plant disease detection [60, 61]. However, at the moment, there are only a few 3D image datasets of certain plants and crops available, and none that contains wheat plants. Consequently, the available studies on FHB detection in wheat either use 2D images of wheat kernels or wheat plants, or use morphological and physiological features to perform the detection.

Reference [62] is one of the very few works that developed a PC dataset of plants. In their work, 7 tomatoes and 7 maize plants were captured daily using a laser scanning system for a period of

two weeks. The laser scanner captures 2D scan profiles for about a 100 mm distance generating a 2D profile every 0.012 mm consisting of a maximum of 7640 points. Then, manual outlier removal was applied on each PC to remove, as much as possible, unwanted data points such as points representing the soil or the pot. Finally, each point within every PC was given a unique label that defines its class. Only a few samples were acquired for this study since the workload to create this dataset is time-consuming and heavy. Nonetheless, this work is one of a few pioneers in the creation of plant PC datasets [63, 64].

For the detection of FHB in wheat, reference [65] used 27 hyperspectral images, each containing a mixture of 25 to 50 wheat kernels of different varieties, to develop an automated algorithm for FHB detection in wheat kernels. Their approach begins with a region of interest delimitation that locates the kernels in the image by thresholding the pixel values reflected from the 647 nm wavelength, followed by a kernel and background segmentation using a wavelength-based thresholding. Next, a convex-hull clustering was applied to isolate the kernels, followed by a calculation of a fusarium index using the mean of the values located in the 1411 nm band. Then, FHB kernels were separated from sound kernels using a threshold, which resulted in 91% accuracy in FHB detection from hyperspectral images. The authors used a detection method based on determining the reflectance band of the kernels and the bandwidth that separated diseased kernels from healthy ones with empirical observations, thus, the detection results may be subject to human error.

In reference [66], three wheat variants of different susceptibility to FHB were used to create a method for FHB detection. 12 images of a wheat plot from the field were collected such that each variant was photographed in four images. Then, the images were augmented by dividing each one into smaller sub-images of resolution 700×700 pixels (px), and a final dataset of 2829 images was generated. Next, a pretrained Mask RCNN model was fine tuned on the dataset by retraining its last few layers to segment the spikes from the background and output their boundaries. Then, a

region growing algorithm was employed to detect FHB infected zones from the segmented spikes and achieved 92% accuracy. Although this research work achieved good accuracy in detecting FHB from coloured wheat images, the model's performance is constrained to specific data acquisition conditions that could not be easily reproduced.

In reference [67], 1680 wheat head samples consisting of both FHB-diseased and non-diseased samples were captured to create a dataset of hyperspectral 2D images. Then, texture features were extracted from the images using a gray-level co-occurrence matrix and dual-tree complex wavelet transform. Colour feature processing was applied by combining three wavelengths each from the RGB spectrums, respectively, to create an RGB image, which was transformed to the YDbDr colour space for feature extraction. Next, principal component analysis was applied to reduce the feature dimensionality and compress the images. Then, gradient boosting decision tree and sequential backward elimination were used to select the relevant features for the prediction models. Lastly, a deep CNN, a support vector regressor, a random forest algorithm, and a partial least square regressor were all used to develop prediction models. The deep CNN achieved a 3.78 root mean squared error and a 0.98 R^2. Although the obtained results are good, the process used to extract the features for the detection models is very long and relies on empirical parameters, which may alter the detection performance.

While there are several works that developed FHB detection methods on wheat, there are no models for the automated estimation of the number of kernels or spikelets in wheat available at the moment of writing this thesis. In fact, the closest work found is a model for counting kernels on maize ears [68] based on image and signal processing where 8 maize varieties were used to gather a dataset consisting of 2000 RGB images of maize ears taken under LED and natural lights. Next, a Gaussian pyramid compression model was applied on the images to remove colour and pixel redundancy, followed by a canny operator for edge detection. Then, a threshold segmentation was applied to separate the maize ear from the background, followed by the mean

shift filter algorithm for kernel pixel clustering. Next, a colour deconvolution algorithm was applied to enhance the edges of the kernels, followed by an adaptive threshold segmentation algorithm for kernel segmentation. Lastly, a local maximum detection method based on Gaussian filtering was applied to detect the maize kernels. The method achieved 93.6% accuracy on kernel detection. Although the performance of the model is good, the methodology consists of a substantial number of steps where each step is very detailed and sometimes consists of multiple sub-steps, which makes it time-consuming, may reduce the efficiency and accuracy of the final outcome that relies on the good performance of all its preceding processing steps, and may reduce the generalization of the approach on new data.

Despite the existence of several research papers that use either hyperspectral or RGB 2D images, or morphological and physiological parameters to develop methods for FHB detection on wheat, there are still no research works that employ multispectral 3D PCs of wheat. Moreover, most of the existing studies use signal and image processing techniques rather than CNNs or rely on empirical features to perform the detection, which tend to generate models that are slower and less precise than neural network-based approaches, and do not generalize well on unseen datasets. Nonetheless, these studies still achieved some interesting results. However, there is room for improvement in the detection performance and execution time. Our work proposes two unique and novel datasets to overcome the unavailability of 3D data representing FHB-diseased and healthy wheat. Moreover, a parallel CUDA-based preprocessing method was created to convert multispectral 3D PCs into multispectral 3D images in order to propose a reliable and fast preprocessing model that allows CNNs to correctly read and process PCs. In addition, a novel and efficient 3D CNN for FHB detection in multispcetral 3D images of wheat is proposed to replace the manual and subjective FHB detection. A unique, fast, and reliable 3D CNN for the total number of spikletes estimation in a wheat head is also proposed to replace the time-consuming, manual, and subjective spikelets counting.

2.2 Myocardial Infarction Detection in Echocardiography

Cardiac imaging technologies have been evolving during the last few decades into more advanced machines that generate complex and detailed data, such as real-time videos of the chambers and valves of the heart. These latter achievements have inspired scientists along with cardiologists to develop newer methods that aim to detect and assess cardiac deficiencies based on evaluating cardiac imaging data. Predominantly, multiple techniques which have been produced over the years to detect cardiac diseases by evaluating the myocardial motion have been based either on signal-processing, chemical-processing, image-processing or, more recently, video-processing.

In reference [69], a contour-based technique for detecting wall motion abnormality by analyzing the temporal pattern of normalized wall thickening was proposed. Epicardium and endocardium zones were manually extracted by segmenting images representing 27 real-life patients and AHA 17-segment model was used to evaluate regional wall changes in normalized wall thickness followed by a Naïve Bayes classifier. Although the model achieved 100% true-negative, it only obtained 70% true-positive for apical 4-chamber (A4C) view, which means that the model does not predict the cardiac disease when it happens 30% of the time. Moreover, manual preprocessing is time-consuming, subject to human error, and relies on human expertise to perform the segmentation task properly, which may affect the accuracy and the quality of the results.

In reference [70], existing quantitative approaches were applied to detect and identify localized wall motion abnormalities from 12-lead ECG, 2D echocardiography images, and coronary angiography in patients affected with MI. Adequate 2D echocardiography images representing 4 different cardiac views were obtained from 74% of well-defined patients and used to assess abnormal segments. These abnormal segments were characterized by a standard deviation that is inferior to the average contraction estimated over 10 normal subjects. Then, ECG and angiography data were analyzed independently by two observers. The results concluded that 2D echocardiography images allow an extended assessment of endocardial wall motion and regional wall thickening and

can be applied for a quantitative approach to detect regional LV abnormalities, while ECG and angiography have certain limitations. Furthermore, results showed that area methods performed better than linear methods by achieving at best 95% of accuracy versus 84% for linear models in predicting localized regional LV contraction deficiencies. Even though the accuracy of the prediction is 95%, the model does not allow an early detection of LV abnormalities. In fact, it only examines the effects of MI on the myocardial performance.

Another approach to assessing data generated from cardiac imaging uses ML models. In reference [71], 723,754 clinically acquired echocardiographic videos of the heart (around 45 million images), representing 27,028 patients, were evaluated to predict 1-year mortality rate in patients who had encountered heart deficiencies. The dataset was divided into 21 groups such that each group represented a standard echocardiographic view. Then, distinct 3D CNN models were generated, trained, and tested on each data group separately. Additionally, longitudinal electronic health records were added to the videos as input data to the models during training, and the accuracy of the 1-year mortality prediction in patients with heart abnormality records was 75%. This paper shows that applying 3D CNNs to echocardiography videos to perform a prediction task is efficient and plausible. However, there is still room for improvement in the accuracy of the models. For example, adding a preprocessing method to normalize the raw videos prior to training the CNNs could enhance the accuracy of the prediction.

The authors in reference [72] used ML in order to assess regional wall motion abnormality in Echocardiographic images. Data from 300 patients with a history of MI were divided into 3 groups such that each data group contained images representing a distinct cardiac abnormality. Data from 100 healthy patients were also included as a 4^{th} data group. Then, only images with good or adequate acoustic detail were selected while poor quality images were discarded from the final dataset. Images were then standardized to the same spatial dimensions and fed to 10 versions of the same CNN model to detect the presence of RWMA. In comparison to the prediction outcome

performed by two expert cardiologists, a CNN produced similar results such that the AUC curve produced by the cardiologists was similar to that produced by the CNN model (0.99 vs 0.98). Even though the prediction results are considerably fair, models were trained only on good-quality echocardiographic images, which implies that testing it over real-life images that potentially contain noise, missed information, or inadequate acoustic details could reduce its performance and may lead to erroneous predictions.

In reference [73], both electrocardiogram and serum analysis were used to detect acute MI in 82 patients who were suspected of having MI within one hour of their arrival to the care unit. The electrical activity of the heart produced by the 12-lead electrocardiogram was recorded and a 10 ml blood sample was obtained within the first hour of their admission to the care unit. Then, all the data was analyzed by two observers. Moreover, several chemical substances such as creatine kinase and myoglobin were measured, which takes 10 minutes to perform. These parameters were combined to perform a logistic regression analysis that led to the detection of MI by 64% accuracy. The model of this work requires a chemical examination that is source-intensive in order to make an assessment that is only 64% accurate, which may not be sufficient for reliable MI detection.

A more recent study [37] developed an MI detection model based on local motion estimation in an attempt to overcome the limitations of speckle tracking methods. The model achieved at best around 85% of sensitivity and specificity by investigating more reliable and robust wall motion analysis models. Although both speckle tracking and local motion estimation used a standard LV segmentation model from which they analyse the motion of each segment of the LV wall separately to determine signs of RWMA and therefore detect MI, speckle tracking relies solely on assessing the motion of a single speckle per segment, while local motion estimation used several speckles per segment. However, the accuracy of both techniques can be negatively impacted by noisy echocardiography, since noise makes segment's tracking difficult and unreliable. Therefore, the authors further explored the use of local motion estimation by developing an approach based on

active polynomial which captures the LV contour and divides it into 7 segments. The motion of each segment was then evaluated to detect RWMA and MI was detected if at least one LV segment motion was determined to be abnormal. The model's performance metrics were estimated on two subsets on the HMC-QU benchmark dataset; the first subset contained the totality of the videos while the second subset contained only reasonable quality echocardiography videos that were selected based on a subjective visual assessment. The reported results showed that the model achieved better MI detection accuracy on reasonable quality videos than on the totality of the dataset, by obtaining 87% accuracy on the second subset and only 83% on the other. Despite attaining better results than current state-of-the-art methods, active polynomials performance decreases considerably when tested on low-quality or low-resolution echocardiography videos, which affects the efficiency and robustness of the method.

Reference [74] also employed the HMC-QU benchmark dataset and proposed a three-phase early MI detection approach for low-quality echocardiography. The architecture of the model begins with an encoder-decoder CNN for LV wall segmentation. The CNN was followed by a feature engineering method that analysed 5 and 6 segments from the segmented LV wall to extract two motion-based features and one area-based feature that were used to create two datasets. Both datasets were used to train four conventional ML models for early MI detection. The reported performance metrics showed that detecting MI from either 5-segment features data or 6-segment features data gave almost the same accuracy and precision. Furthermore, all four ML models achieved nearly the same results, and overall, the best achieved accuracy, sensitivity, and specificity were equal to 80%, 85%, and 74%, respectively. Although the model's main contribution is an operator-independent MI assessment that provides quantitative measurements for the LV wall segments, the attained evaluation metrics can be further improved possibly by a more extended feature extraction method and a more accurate classification.

Although all aforementioned studies have explored a variety of approaches to develop robust

and efficient MI detection models from echocardiography videos, and have succeeded, to a certain extent, in obtaining fairly satisfactory results, potential improvements to achieve higher and more robust performances still exist. While knowing that any non-zero false-negative rate can cause the loss of multiple human lives due to wrong diagnosis, and that late MI detections can delay the treatment of patients and cause irreversible damage on their health, developing more precise and robust models remains necessary. Our approach aims to overcome most of the drawbacks observed and learnt from most recent MI detection research works. Thus, we created an end-to-end fully automated pipeline which leaves no room for manual processing that may lead to generating unreliable, erroneous, or subjective segmentation or detection results. Furthermore, we used the totality of the HMC-QU benchmark dataset, which mainly consists of low-quality and low-resolution echocardiography videos, and we applied a 2D CNN for LV segmentation. Then, we used a 3D CNN as a feature extractor and a densely connected neural network as a classifier to achieve the highest MI detection accuracy regardless of the video quality in order to create a robust diagnosis approach. Moreover, our method is lightweight and runs in real-time on parallel threads, which accelerates the diagnosis results especially in case of emergencies.

Chapter 3

Convolutional Neural Networks

The outline of this chapter is as follow; Section 3.1 introduces artificial neurons and multi-layer perceptrons (MLP), and Section 3.2 defines the convolutional and pooling layers that makeup a CNN. Section 3.3 describes the training process of a CNN that is defined by a loss function, the backpropagation algorithm, parameter and hyperparameter optimization, regularization methods, and evaluation metrics. Finally, Section 3.4 describes the characteristics of three advanced benchmark CNN architectures used in this thesis.

3.1 Artificial Neural Networks

In this section, artificial neurons and MLPs are explained.

3.1.1 Artificial Neuron

An artificial neuron is a unit that takes a vector of real-valued inputs, for *e.g.* $(x_1, ..., x_n)$, associates a weight w_i to each input value, then adds a real-valued bias b to the inputs and performs a

weighted sum to produce an output γ. Then, it applies an activation function f to the sum γ to produce a final single output $\hat{y}$. The activation function should be differentiable. An illustration on an artificial neuron is shown in Figure 3.1. In fact, an artificial neuron is a caricature of a real neuron of the human brain, therefore, for the remainder of this thesis, the term artificial will be implied when discussing neurons and neural networks.

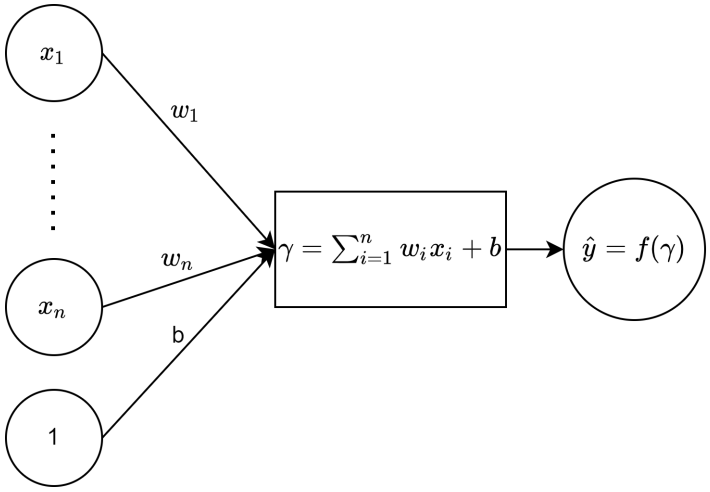


Figure 3.1: An artificial neuron.

3.1.2 Multi-Layer Perceptron

An artificial neural network (ANN) [75] is a ML model that consists of multiple neurons that are stacked in several layers, such that each layer consists of one or more neurons. A MLP is a class of ANNs that consists of two or more layers of neurons. The first layer represents a vector of real-valued scalars called the input layer and the last layer represents a single output or a vector of outputs called the output layer. The middle layers (called hidden layers) are successive layers of neurons that accumulate valuable information contained in the weights of each neuron. There can be 0 or more hidden layers. The process of transferring information from the input to output layers (via the hidden layers) is called feedforward propagation, which is the first step of the learning

process in a MLP. In order to optimize the output value, an update to the weights and biases is required to adjust the network performing a given task (based on the input data). The adjustments are determined by a back-propagation algorithm that calculates the gradient of a loss function with respect to the weights and biases of the network in order to minimize the loss value and produce a useful output. Feed-forward and back-propagation algorithms are usually applied through multiple iterations to train the network by updating its parameters in order to optimize the final output. Figure 3.2 shows a MLP that has an input layer composed of n scalars, a single hidden layer composed of m neurons, and an output layer composed of 1 neuron.

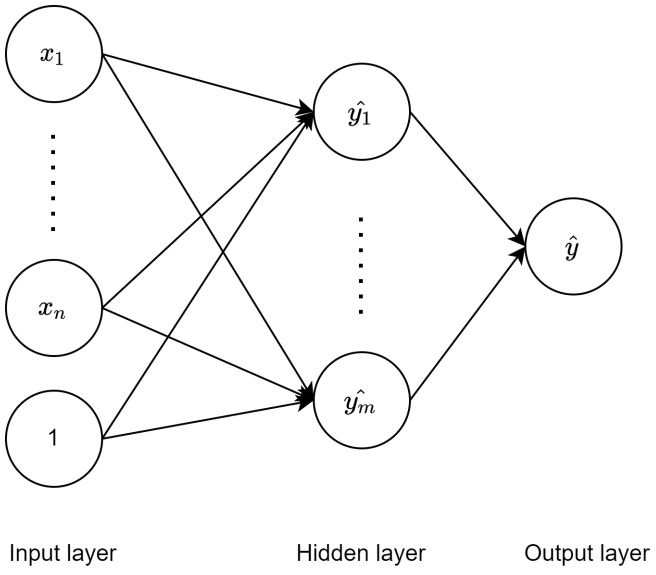


Figure 3.2: A multi-layer perceptron.

3.2 Convolutional Neural Network

In this section, convolutional layers and pooling layers are presented and explained.

3.2.1 Convolutional Layers

A CNN is a special kind of MLPs that takes a signal as input rather than a feature vector that consists of the realization of multiple independent random variables. A signal is represented as a tensor where its consecutive samples along any dimension are correlated [76]. For example, an image is a 2D signal where its consecutive samples, called pixels, along width and height are correlated, and a video is a 3D signal where its consecutive samples along width, height, and time are correlated.

A CNN is composed of consecutive layers such that each layer contains multiple filters. Each filter within a layer performs convolution operations on consecutive local regions of a given input signal and a finite kernel [77]. The kernel, also called a mask, is a small sized matrix consisting of weights. It has the same number of dimensions as the input signal. For example, a CNN processing a 2D input signal would use 2D masks. Typically, a mask is much smaller than the input signal. The process starts by each filter in a layer performing convolution operations on local regions of the input signal and the mask to produce a filtered signal. Then, all the filtered signals produced by each filter in a layer are stacked together, then, a bias is added to the stack of filtered signals and an activation function is applied to it in order to produce a feature map. The convolution process is performed by sliding the mask along the axes of the entirety of the input signal by a fixed stride, which is the step by which the mask slides along each axis. The produced feature map has a size less than or equal to the size of the input signal, depending on whether the signal was zero-padded or not during the convolution [77]. Figure 3.3 shows how the first sample q_{11} of the filtered signal is calculated with the convolution operation performed on the local region of the input image, consisting of p_{11}, p_{12}, p_{21}, and p_{22}, and the mask. The stride of the mask is 1 along both axes and the input image is zero-padded around the edges to ensure an output with the same input size and to allow the mask to cover the pixels located at the borders. Figure 3.3 also shows how the third sample q_{13} is calculated by performing the convolution operation on the local region

of the input image, consisting of p_{13}, p_{23}, and two zero-padded pixels, and the mask.

The advantages of convolutional layers over fully-connected layers found in MLPs is the weight sharing that occurs in CNNs due to convolving the same mask over the same input signal, which favors learning the same patterns from different local regions. In fact, contrary to MLPs which associate a single weight with each sample, convolutional layers allow a single weight to be associated with all samples within the same signal (except some of the boundary samples when the signal is not padded), which ensures learning relationships between the samples. Moreover, weight sharing guarantees the usage of a smaller number of parameters which reduces the complexity of a CNN [77].

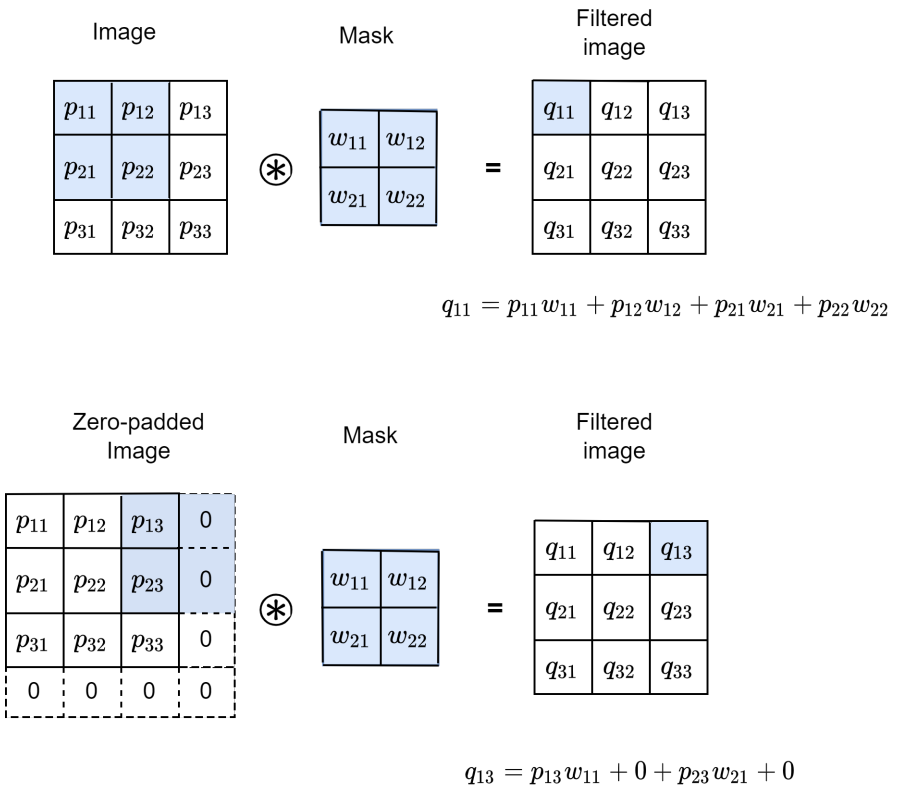


Figure 3.3: Convolution operations to show how samples of the filtered image are calculated.

Figure 3.4 shows the 3D convolution operation that occurs by using a 3D image with a 3D mask,

and the convolution operation outcome is a filtered 3D image. 3D convolution follows the same principles as its 2D counterpart, but in 3D. Meaning that, the input signal, the convolution mask, and the filtered output all have an extra dimension (*i.e.* a depth). The convolution operation in 3D is performed on local 3D regions instead of 2D regions, and the 3D mask have distinct weights in each of its voxels. The 3D mask moves along all three dimensions of the input; by sliding over the height and width, and then along the depth.

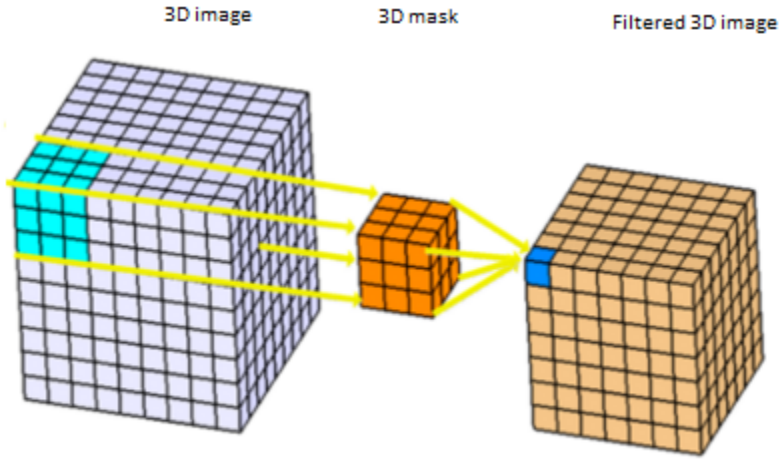


Figure 3.4: A 3D convolution operation [78].

During convolution, a multi-channel 2D image is processed as a 3D image, such that the depth of the 3D image is equal to the number of colour channels contained in the 2D image. The 3D image is generated by stacking 1-channel 2D images, where the pixels of each 2D image contain the values corresponding to one of the colour channels. For instance, Figure 3.5 shows an RGB 2D image of size $(6 \times 6 \times 3)$, such that the height and width of the 3D image are both equal to 6 and correspond to the height and width of the 2D image, while 3 represents the depth of the 3D image that is determined by the number of channels (in this case the channels are RGB) contained in the 2D image. The first, second, and third 2D images contained in the 3D image contains R, G, and B colour values, respectively. The convolution mask used to filter a multi-channel image is

a 3D mask with a depth equal to the number of colour channels of the 2D image. In Figure 3.5, the 3D mask has a depth equal to 3. To estimate the values within the filtered image, the same convolution process as described previously by 3D convolution operations, is followed.

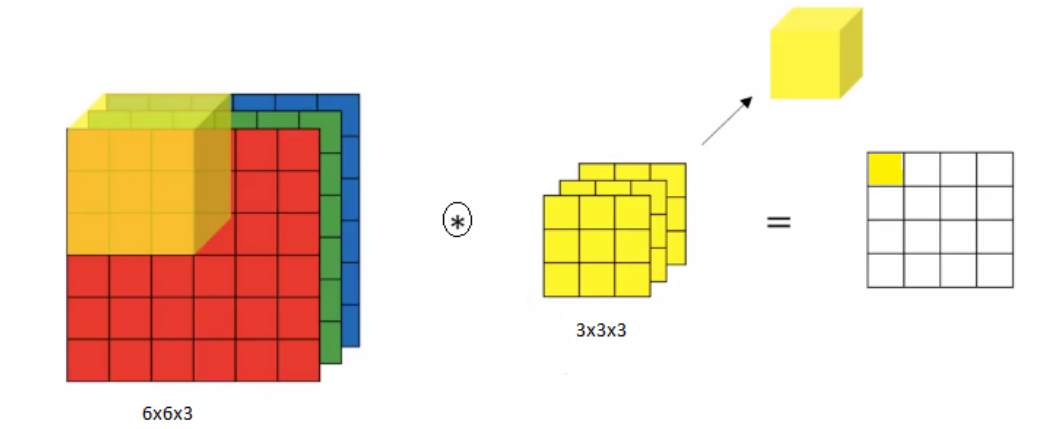


Figure 3.5: A convolution operation on a three-channel (RGB) 2D image [79].

3.2.2 Pooling Layers

Pooling layers are used to compress feature maps produced from convolutional layers, thus, they are generally used after a convolutional layer. A pooling layer uses a pooling filter that employs pooling operations to reduce the size of a feature map. The pooling filter is a small sized matrix that has the same number of dimensions as the feature map and a smaller size than the size of the feature map. It slides along local regions of the feature map by a stride, that is usually greater than or equal to 2, to perform a pooling operation that can be a maximum (max) pooling or an average pooling. For example, a 2×2 max pooling filter that slides by a stride equal to 2 and is applied over a 256×256 px feature map, will result in a 128×128 px filtered feature map where each of its samples represent the maximum value of a local 2×2 px region from the feature map. Thus, pooling layers serve mainly to reduce the dimensions of feature maps. Consequently, it reduces the number of parameters of the feature maps and results in more representative samples. Figure

3.6 shows how the first sample q_{11} of the filtered feature map is calculated by performing the max pooling operation using the max pooling filter over the local region of the feature map consisting of p_{11}, p_{12}, p_{21}, and p_{22}. Figure 3.6 also shows how the second sample q_{12} is calculated by sliding the pooling filter by the stride, which is equal to 2, to superpose it over the local region of the feature map consisting of p_{13}, p_{23}, and two zero-padded samples, then performing the max pooling operation. The input feature map is zero-padded around the edges to allow the pooling filter to cover the samples located at the borders.

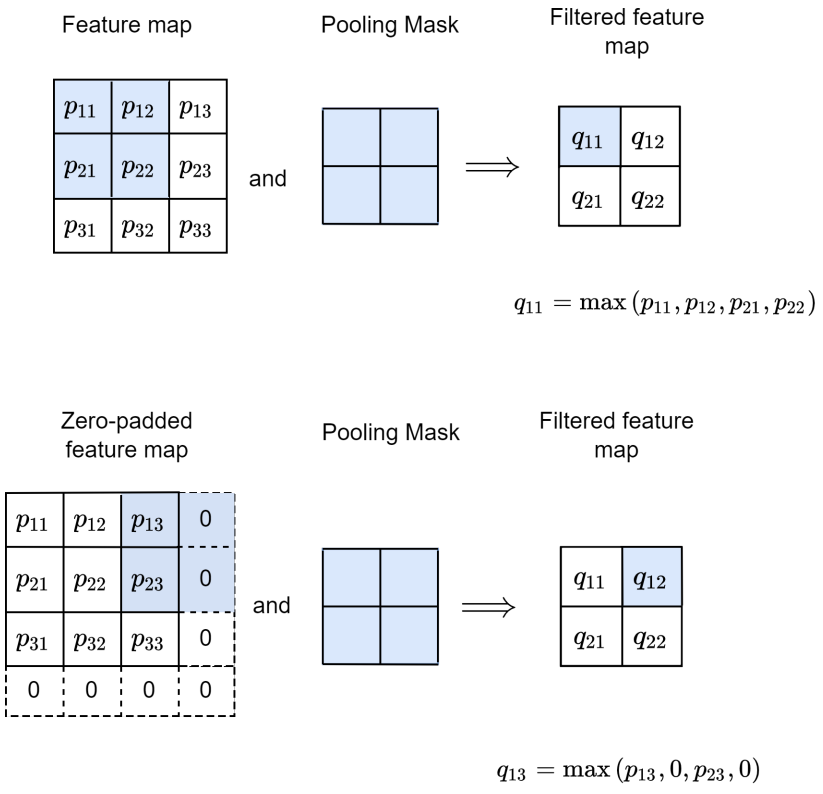


Figure 3.6: Maximum pooling operations to show how samples of the filtered feature map are calculated.

3.3 Learning Process and Evaluation

In this section, loss functions, optimization algorithms, regularization techniques, and performance metrics related to CNNs are discussed.

3.3.1 Loss Functions

A loss function, also called an objective function, is a scalar differentiable function that measures the difference between the estimated output and the actual output. In fact, the objective when developing a CNN is to estimate the most adequate values of the parameters, which are the weights and biases, to define a network capable of mapping a given input to its correct output. However, the possibility of simply defining a perfect combination of network parameters that produces an optimal approximation is very low. Thus, the strategy followed to train a CNN is to iteratively adjust the network parameters based on an optimization algorithm, called gradient descent, to minimize the loss function.

There are several objective functions that are employed depending on the type of problem. For example, binary cross entropy (CE) loss function [80] is commonly employed for binary classification (*i.e.*, detection) problems, which is defined as

$$Binary\ CE(y,\hat{y}) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y}) = \begin{cases} -\log(1-\hat{y}), & \text{if } y = 0 \\ -log(\hat{y}), & \text{if } y = 1, \end{cases}$$

where y is the actual output label and can be either 0 or 1, and $\hat{y}$ is the estimated probability that the input sample belongs to class 1. For regression problems, mean squared error (MSE) loss function is commonly used. Given an estimated output $\hat{y}$ and a ground truth y, both composed of N scalars, the MSE is defined as

$$MSE\,(y,\hat{y}) = \frac{1}{N}\sum_{n=1}^{N}(y\_\hat{y})^2\,.$$

3.3.2 Optimization

Parameter and hyperparameter optimization algorithms are discussed in this section.

3.3.2.1 Gradient Descent

Gradient descent is a first-order iterative optimization algorithm used to find a local minimum of a differentiable real-valued function. In the context of CNNs, it is employed to minimize the value of a loss function with respect to the network's parameters (weights and biases). For example, given a function f that has n parameters $(w_1, ..., w_n)$, gradient descent attempts to solve the following optimization problem:

$$\underset{(w_1,...,w_n)}{argmin} f(w_1, ..., w_n).$$

Gradient descent starts by an initial guess $(w_1^{(0)}, ..., w_n^{(0)})$ and repeats the following steps:

$$\forall t \in \mathbb{N}, 1 \leq i \leq n, w_i^{(t+1)} = w_i^{(t)} - \eta \frac{\partial f}{\partial w_i}(w_1^{(t)}, ..., w_n^{(t)}),$$

where $\eta \in]0, 1]$, is called a learning rate, and determines the convergence speed of gradient descent. t is the iteration and $w_i^{(t)}$ is the estimated value of w_i at iteration t. The algorithm stops when the minimum value is reached (all derivatives are 0) or when a predetermined number of steps is reached.

3.3.2.2 Back Propagation

Back propagation is an optimization algorithm based on the gradient descent algorithm and used in neural networks during the training process to update the weights and biases in order to minimize a loss function. Back propagation is based on the chain rule, which computes the gradient of a given layer by using the gradients of its subsequent layer. In practice, back propagation starts by computing the gradients of the last layer, which can easily be determined through the loss function, then uses them to compute the gradients of the previous layer. This process is repeated until

reaching the first layer. It is called back propagation because the gradients are propagated from the last to the first layer.

3.3.2.3 Parameter Optimization

Parameter optimization, also referred to as model training, is the process of updating the weights and biases of the neural network through the backpropagation algorithm. In practice, gradient descent is the most used algorithm for training CNNs, however, there exist several gradient descent optimization algorithms that provide a few advantages. For instance, root mean square propagation (RMSProp) [81] is a gradient descent algorithm that proposes an adaptive learning rate for each of the parameters of the network. This algorithm computes a moving average of the second moments of the gradients and uses it to update the learning rate. One of its advantages is the ability to produce good parameter optimization with mini-batches, which is advantageous in memory-bound environments. Adaptive moment estimation (Adam) [82] is another gradient descent optimization algorithm that upgrades RMSProp algorithm by taking into account (during an iteration t) all previous gradients (iterations $t-1, t-2, \dots$) when updating a parameter value. This provides more stable weight updates, which is beneficial for sparse data.

Each optimization algorithm is defined by a set of hyperparameters such as the learning rate, decay, and momentum. However, the learning rate is common to every optimizer since it determines the convergence speed of gradient descent (*i.e.* it determines the rate by which the values of the weights and biases are updated per iteration).

3.3.2.4 Hyperparameter Optimization

Hyperparameters are variables that cannot be optimized through gradient descent. They define the structure of the model (number of layers, number of neurons, type of layer, mask size, stride, padding, *etc.*) and the behaviour of the model (weight initialization, learning rate, activation

function, optimizer, regularizer, number of epochs, batch size, *etc.*). The structural hyperparameters are the following:

- *number of layers* is the number of hidden layers that define the architecture of the network.
- *number of neurons* is the number of neurons per layer.
- *type of layer* is either a convolutional layer, a pooling layer, *etc.*
- *mask size* is the size of the convolutional or pooling mask.
- *stride* is the number representing the step by which the mask slides from a local region to a consecutive one in the input signal.
- *padding* is a boolean indicating whether the input signal is zero-padded or not.

The behavioural hyperparameters are the following:

- *weight initialization* is the process of defining the initial values of the parameters of the network.
- *learning rate* is the value by which each value of the parameters is updated during a training iteration. It also defines the convergence speed of the gradient descent.
- *activation function* is a differentiable function used to define the output of a neuron. There are many activation functions used with CNNs depending on the type of problem and the layer. For instance, the rectified linear unit (ReLU) [83] activation function is commonly used in hidden layers in both classification and regression problems mainly due to its simplicity, and it is defined as $ReLU(x) = \max(0, x)$. It can also be used as the activation function of the output layer of a network in regression problems when the desired output is a positive real value. Sigmoid is another popular activation function defined as $sigmoid(x) = \frac{1}{1+e^{-x}}$. It

is commonly used in the output layer of a network in detection problems since it outputs a probability between 0 and 1.

- *optimizer* is the parameter optimization algorithm.

- *number of epochs* is the number of times the model has to go through all the samples of the training set during the learning process.

- *batch size* is the number of examples needed for one training iteration where an iteration is defined by one feedforward pass and one backpropagation pass.

Selecting the hyperparameters can be done either manually or automatically. Manual hyperparameters selection requires a thorough understanding of the role of each hyperparameter and its limitation, the problem that needs to be solved, the data used, the available computational resources (memory, runtime), *etc.* In contrast, automatic selection of hyperparameters reduces these constraints but demands more computational resources. In all cases, selecting the hyperparameters should not be performed using the training data, otherwise, the model would risk encountering overfitting, which refers to a phenomenon that can occur when the network starts to memorize the training set during the learning process and fails to generalize over the test set. In other words, selecting the hyperparameters such that it enables the model to perform well on training samples will make it fail to obtain the same results on new unseen samples. Moreover, adjusting the hyperparameters via test set evaluation may bias the model towards the test set. Models must not be exposed to the test set during the training process in order to fairly evaluate the generalization performance of the trained model. As a result, a separate dataset (*i.e.* different than the training set) is necessary to determine the best hyperparameters. Hence, a common practice used while training neural networks is to split the data into a training set, a validation set, and a test set. The training set is used to adjust the network weights and biases through the parameter optimization algorithm to form a mapping from input data into the correct output

data. The validation set is used to adjust the hyperparameters of the network to find the best hyperparameter combination and to evaluate its performance. Finally, the test set is preserved to test the generalization capability of the trained network on new data.

One of the common practices for automatic selection of hyperparameters is grid search, which is defined by selecting a finite set consisting of a few values for each hyperparameter, then training a model using each combination of hyperparameter values. Finally, the hyperparameter combination that yields the best performing model on the validation set is selected. Following that, a model with the selected hyperparameters is created and retrained from scratch on the entirety of the training and validation sets combined, and evaluated on the test set.

3.3.3 Regularization

Regularization is a strategy that aims to prevent a neural network from overfitting and memorizing the data rather than learning the important patterns during training. Therefore, several regularization techniques can be employed with neural networks such as dropout layers [84], and L1 and L2 regularization functions [85]. Dropout regularization performs a random cancellation of the values contained in certain neurons per layer, such that their information is not carried to the next layers. Dropout regularizer is characterized by a dropout rate, which is defined as a float between 0 and 1, that determines the number of cancelled neurons per layer. The dropout technique produces a robust network that is likely to succeed in generalizing over unseen data, since the random cancellation of some neurons is equivalent to adding noise to the data which makes neurons focus better on learning the most important features [84]. L1 and L2 regularizers are different than dropout layer since they regularize the bias, the weights, or the activation output per layer. They operate by adding a penalty to the loss function to favor only the small parameter values. L1 adds a penalty equal to the sum of the absolute values of the weights, while L2 adds a penalty equal to the square root of the sum of squared weights. In both cases, the penalty guides

the optimization algorithm into creating smaller parameter values, which results in a more robust network [86].

3.3.4 Performance Metrics

Performance metrics are used to evaluate the performance of the neural networks and determine their efficiency and effectiveness on any given problem. In general, the most common practice used to evaluate the performance of a CNN, is to train several models for multiple epochs, then compare evaluation metrics to determine the model that performs best on a given problem. Depending on whether the network is solving a regression or a classification problem, the adequate performance metrics change. For example, classification accuracy is commonly used to estimate the ratio of the correctly predicted samples to the total number of samples, and it is expressed as

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN},$$

where TP is true positive, FP is false positive, TN is true negative, and FN is false negative. However, in the case of a binary classification, accuracy can also be expressed as

$$Accuracy(y, \hat{y}) = 1 - MSE(y, \hat{y}),$$

where $\hat{y}$ is an estimated output and y is a ground truth, both composed of N scalars. F1 score is also commonly used to evaluate a model's performance in classification problems, and it is expressed as

$$F1 = \frac{2 \times (precision \times recall)}{precision + recall},$$

where precision is expressed as $TP/(TP + FP)$ and recall is expressed as $TP/(TP + FN)$.

For regression problems, mean absolute error (MAE) is commonly used to estimate the difference between the real output and the predicted output. Given an estimated output $\hat{y}$ and a ground truth y both composed of N scalars, the MAE is defined as

$$MAE(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N-1} |y_i - \hat{y}_i|.$$

3.4 Advanced Convolutional Neural Network Architectures

Finding the most efficient CNN architecture that yields the best results on a given classification or regression problem cannot be determined or described in a straightforward manner. The most common strategy followed for choosing the best-performing model on a specific problem is generally decided by evaluating the performance of several CNN models, each defined by a different hyperparameter combination, on the same data. Then, models that achieve the best results on the same validation set are considered as the most efficient models for the given problem. These CNN architectures can either be created from scratch (through hyeperparameter optimization), or inspired from well-known CNN architectures such as deep residual learning (ResNet) [87] and densely connected convolutional networks (DenseNet) [88]. These famous architectures have achieved outstanding performances over very large and complex datasets such as ImageNet [89] for 1000-class classification and CIFAR-100 [90] for 100-class classification. These well-known architectures also serve as benchmarks when developing new approaches since they are widely published. In order to solve new problems on new datasets, several predefined model architectures can be employed as pretrained models or trained from scratch on the new datasets. In this section, DenseNet and two versions of ResNet are explained.

3.4.1 Deep residual Learning

The depth of a CNN is defined by its number of layers. Creating a model with multiple layers usually allows it to learn different levels of patterns from the input data which leads the model to learn a better generalization of the problem. However, CNNs with deep architectures have the tendency to encounter the vanishing gradient phenomenon [91], which means that the gradients can diminish toward the lower layers of the network and reduce the model's generalization capability. As a result, ResNet proposed the concept of deep residual learning based on residual blocks that

create shortcut connections between the layers to allow the learned information to reach the first layers and reduce the impact of the vanishing gradient. ResNet comes in two versions; ResNet version 1 (v1) is built with layers each consisting of a 2D convolutional layer with a mask size of (3×3), a batch normalization layer [92], and a ReLU activation that are stacked successively. Figure 3.7 represents a shortcut connection between two layers, where Y_{l-2} and Y_{l-1} denote the feature maps resulting from $l-2$ and $l-1$ layers, respectively. Similarly, y_l refers to the pre-activation feature maps resulting from the first two parts (convolutional layer and batch normalization) at layer l, and Y_l is the final feature map resulting from applying a ReLU activation on the addition of Y_{l-2} with y_l. Y_l is defined as $Y_l = ReLU(Y_{l-2} + y_l)$. All these layers and the shortcut connection define a residual block.

The depth of a ResNet v1 is defined by its number of residual blocks, where each residual block consists of 2 ResNet v1 layers consisting each of 3 layers (2D convolutional layer, a batch normalization layer, and a ReLU activation layer) as represented in Figure 3.7. Thus, the depth of a ResNet v1 is defined as $6 \times N + 2$, where N is the number of residual blocks and 2 represents the number of output layers that come after the residual blocks. These two output layers are a 2D average pooling layer and a dense layer. For instance, the depth of a ResNet v1 consisting of 2 residual blocks is equal to $6 \times 2 + 2 = 10$ layers.

ResNet version 2 (v2) differs from ResNet v1 in that it uses a different order for the layer components, which are batch normalization, ReLU activation, and a 2D convolutional layer. A residual block in ResNet v2 consists of 3 composed layers as shown in Figure 3.8, such that the feature maps y_l of layer l are added through the shortcut connection to the feature maps Y_{l-3} outputted from layer $l-3$ to produce Y_l. Moreover, ResNet v2 uses a different mask size, equal to (1×1), for the 2D convolution in the middle layer, as shown by layer $l-1$ in Figure 3.8.

The depth of a ResNet v2 is estimated following the same principle as ResNet v1, except that a residual block in ResNet v2 consists of 3 ResNet v2 layers consisting each of 3 layers (batch

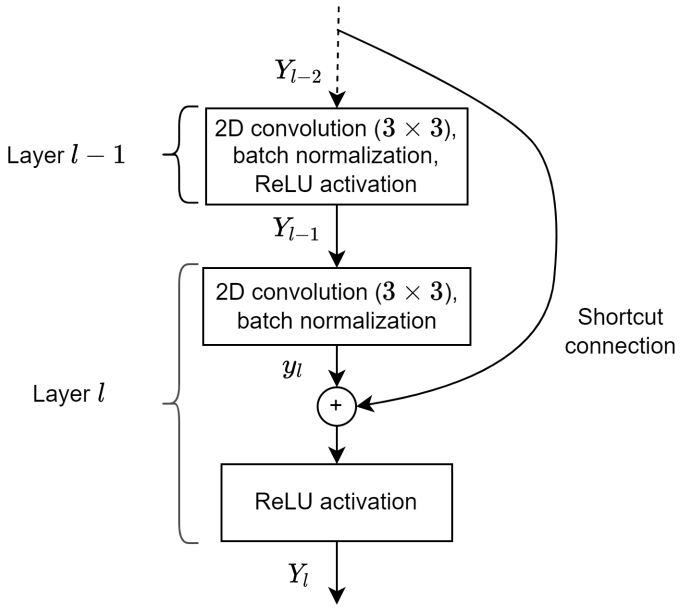


Figure 3.7: A shortcut operation in a residual block in ResNet v1 between two layers to allow the learned information to reach the first layers and reduce the impact of the vanishing gradient. A residual block in ResNet v1 consists of two layers where each layer consists of a 2D convolution, a batch normalization, and a ReLU activation.

normalization, ReLU activation, and 2D convolutional layer) as presented in Figure 3.8. Hence, the depth of a ResNet v2 is defined as $9 * N + 2$, where N is the number of residual blocks and 2 represents the number of output layers that come after the residual blocks. These two output layers are a 2D average pooling layer and a dense layer. For instance, the depth of a ResNet v2 consisting of 2 residual blocks is equal to $9 \times 2 + 2 = 20$ layers. In this thesis, both ResNet v1 and ResNet v2 are used in Section 4.5.

3.4.2 Densely Connected Convolutional Networks

DenseNet proposed another alternative for solving the vanishing gradient problem by converting the shortcut connections in ResNet v2 into dense connections that concatenate the feature maps resulting from any layer to all the feature maps of its previous layers. Figure 3.9 shows a dense

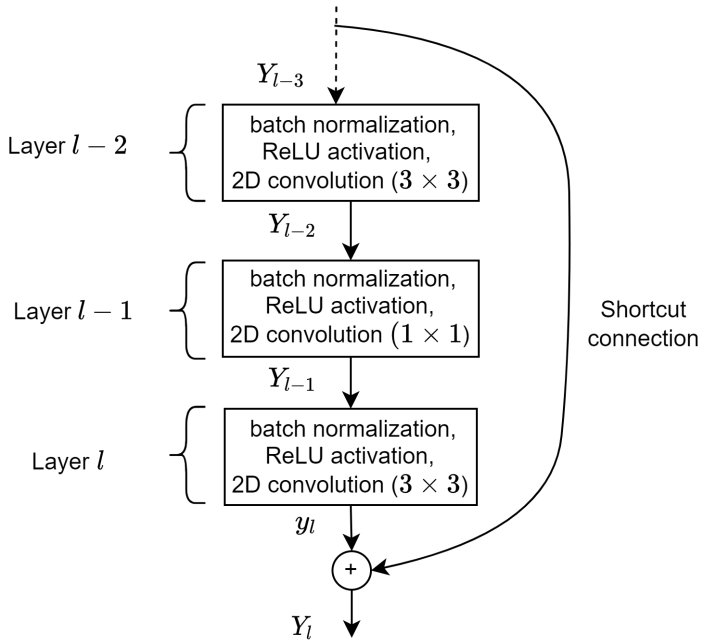


Figure 3.8: A shortcut operation in a residual block in ResNet v2 between two layers, where a residual block consists of three layers and a layer consists of a batch normalization, a ReLU activation, and a 2D convolution.

block consisting of two layers. The feature maps Y_{l-1} consists of the concatenation of the feature maps of layer $l-1$ and the feature maps of all the preceding layers. The layers in DenseNet consist of a batch normalization layer, a ReLU activation, and a 2D convolutional layer with a mask of size equal to (3×3). The growth rate, γ, represents the number of feature maps produced per layer, and it is commonly set to 12 or 24. For instance, if the number of feature maps Y_{l-2} is equal to γ_{l-2}, then, the total number of feature maps Y_l produced at the output layer of a 2-layer dense block, as represented in Figure 3.9, is equal to $2 \times \gamma + \gamma_{l-2}$.

Moreover, it is recommended to use, prior to a dense block, a bottleneck layer [88]. As shown in Figure 3.10, a bottleneck layer consists of a batch normalization, a ReLU activation, and a 2D convolutional layer with a mask size equal to (1×1) with a number of feature maps equal to $2 \times \gamma_0$. The purpose of the bottleneck layer is to decrease the total number of feature maps. Furthermore, it is recommended to use an average pooling layer prior to the output layer (dense layer). However,

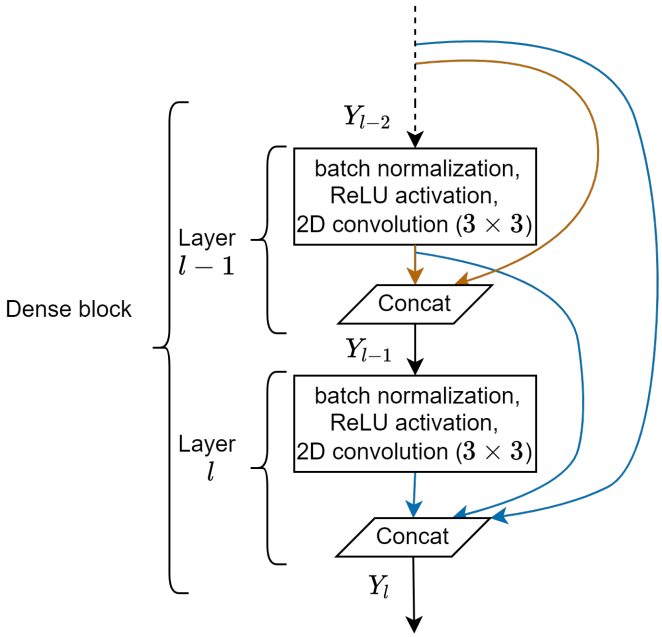


Figure 3.9: A two-layer dense block in DenseNet, where a layer consists of a batch normalization, a ReLU activation, and a 2D convolutional layer with a mask equal to (3×3). The feature maps Y_{l-1} consists of the concatenation of the feature maps of layer $l-1$ and the feature maps of all the preceding layers. Also, the feature maps Y_l consists of the concatenation of the feature maps of layer l and the feature maps of all the preceding layers.

utilizing multiple shortcut connections results in an increase in the total number of feature maps. Thus, DenseNet proposed a transition layer between two dense blocks. As shown in Figure 3.11, a transition layer consists of a batch normalization and a 2D convolutional layer of mask equal to (1×1), followed by an average pooling layer. The number of feature maps produced by the 2D convolutional layer of the transition layer is usually strictly smaller than the number of feature maps produced by its preceding dense block, in order to reduce the number of feature maps fed to the following dense block. Moreover, the size of the mask used with the average pooling layer is usually equal to (2×2) in order to reduce the dimensions of the feature maps. In this thesis, DenseNet is adapted to 3D and used in Section 4.5.

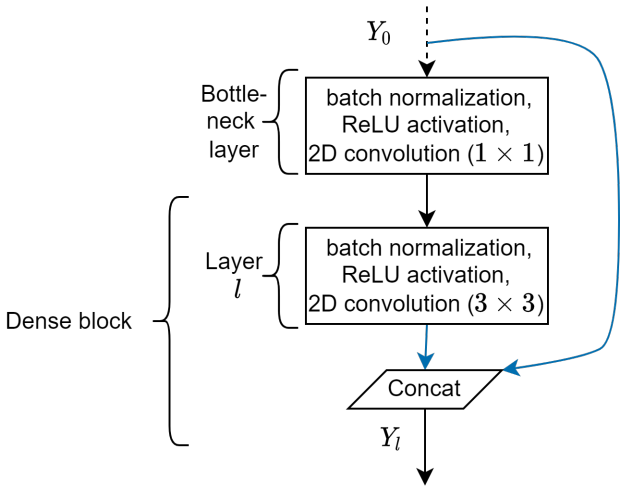


Figure 3.10: A bottleneck layer preceding a 1-layer dense block, where a bottleneck layer consists of a batch normalization, a ReLU activation, and a 2D convolutional layer with a mask equal to (1×1). The purpose of the bottleneck layer is to decrease the total number of feature maps.

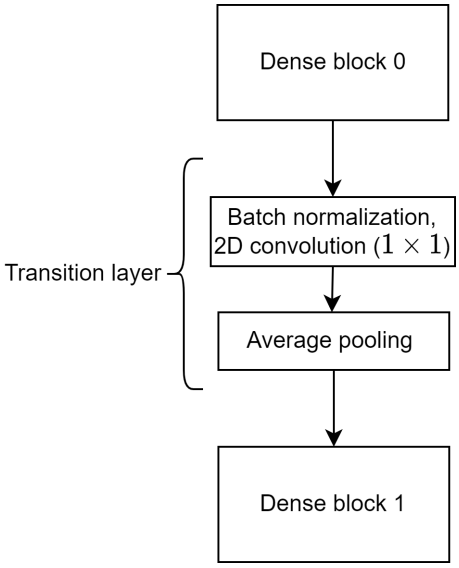


Figure 3.11: A transition layer between two dense blocks where a transition layer consists of a batch normalization and a 2D convolutional layer of mask equal to (1×1), followed by an average pooling layer.

3.5 Chapter Summary

This chapter presented and explained the building blocks of a CNN. Moreover, this chapter highlighted the fact that the backporpagation algorithm is what guides the learning process of the network by updating the weights and biases (during multiple iterations) until a model is produced that is the best approximation of a function. It is this approximation (encapsulated by the model) that minimizes the loss and therefore predicts the output - as accurately as possible - in terms of the input. By using a regularizer, the CNN can further optimize the learnt features from the data and provide more precise results. The evaluation metrics were also discussed, which determine the efficiency of the models. Finally, the ResNet v1, ResNet v2, and DenseNet models have been presented and explained as two well-known advanced architectures that are widely used to solve classification and regression problems.

Chapter 4

Detection and Estimation in Point Clouds of Wheat

The outline of this chapter is as follows. In Section 4.1, we explain the methodologies for the creation of the datasets, and the development of 3D CNNs for FHB detection and wheat head spiklet estimation. Section 4.2 describes the experimental setups for the growing, inoculation, and acquisition of the wheat plants, and the labelling of the wheat scans. Following that, Section 4.3 explains the data preprocessing with CUDA. Both Sections 4.4 and 4.5 explain the creation process and architectures of the models, the experimental setups, and the results corresponding to the 3D CNNs for FHB detection in wheat plants and the total number of spikelets estimation in wheat heads, respectively. Finally, Section 4.6 presents the chapter summary.

4.1 Methodology Overview

In this section, we broadly describe (through diagrams) the creation of datasets for FHB detection and the total number of spikelets estimation, the data preprocessing needed to ensure that both

datasets can be processed by 3D CNNs, as well as the steps needed to achieve SI estimation.

4.1.1 Datasets Creation

To achieve FHB detection in wheat, a dataset containing multiple wheat samples is needed. Therefore, in collaboration with the MRDC, we created two datasets; UW-MRDC WHEAT-PLANT PC and UW-MRDC WHEAT-HEAD PC. The UW-MRDC WHAT-PLANT PC dataset was created by scanning potted wheat plants during three different growth stages. Figure 4.1 gives the flow chart for creating these datasets, where green, gray, and yellow blocks represent activity processes, dates, and inputs/outputs, respectively; and the arrows indicate the direction and the succession of the activities.

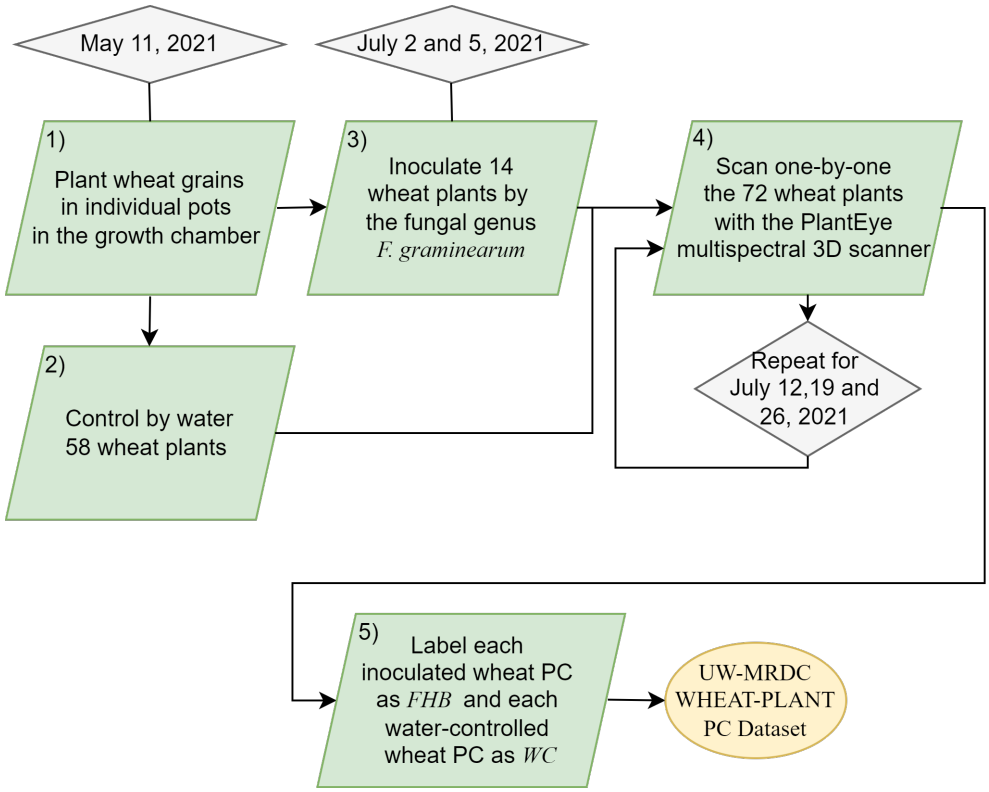


Figure 4.1: Diagram of the UW-MRDC WHEAT-PLANT PC dataset creation process.

First, the process of dataset creation started by planting the wheat grains in small pots inside the growth chamber on May 11^{th}, 2021 as shown by the first activity block and its corresponding time block in Figure 4.1. In total, 72 wheat plants were selected to proceed with the following steps. Then, the wheat plants were divided into two categories such that in the first category, 14 samples were inoculated with FHB on two different dates, the 2^{nd} and 5^{th} of July, while in the second category, the remaining 58 wheat plants were kept water-controlled and safe from any potential FHB infection. Subsequently, the 72 plants where repeatedly scanned on three dates as shown in the 4^{th} process block in Figure 4.1, where the corresponding date block and its loop arrow indicate the repetition of the activity per date. Plants were scanned on different dates to capture the development of disease symptoms in wheat over time since the early FHB symptoms are observed when at least one wheat spikelet turns yellow or pinkish and becomes distinguishable from the other green spikelets, and, as time goes by, the disease may keep developing and more spikelets may get infected in a wheat head. Finally, after scanning each plant, the PlantEye multispectral 3D scanner generates a PC that is assigned a unique label that depends on the acquisition date and on whether the plant is infected or not. The PCs of infected wheat were labelled as FHB, while the water-controlled ones were labelled as WC. The final dataset consists of 216 labelled PCs, where the first set of 72 PCs, the second set of the 72 PCs, and the third set of the 72 PCs represent the scans of wheat plants on the 12^{th}, the 19^{th}, and the 26^{th} of July, respectively.

The second dataset, UW-MRDC WHEAT-HEAD PC, was created to achieve the task of spikelet-number estimation. In this case, only scans showing wheat spikes (also called wheat heads) were needed. Thus, after finishing the scans related to the UW-MRDC WHEAT-PLANT PC dataset, all 72 wheat heads were cut from their stems, and 8 more spikes were added to increase the number of physical samples. Therefore, the UW-MRDC WHEAT-HEAD PC dataset contains 80 PCs of wheat heads in total, among which 14 are infected with FHB and the remaining 66 are water-controlled. The creation process of the UW-MRDC WHEAT-HEAD PC is shown in Figure

4.2. Since the plants used for both datasets are the same, it was needless to repeat the plantation and inoculation steps in this diagram. Therefore, the only activities depicted are cutting, scanning, and labelling wheat heads. The output of the diagram is a labelled dataset consisting of 80 samples where each sample represents a PC of a wheat spike as generated by the multispectral 3D scanner, and each corresponding label is an integer representing the corresponding number of spikelets.

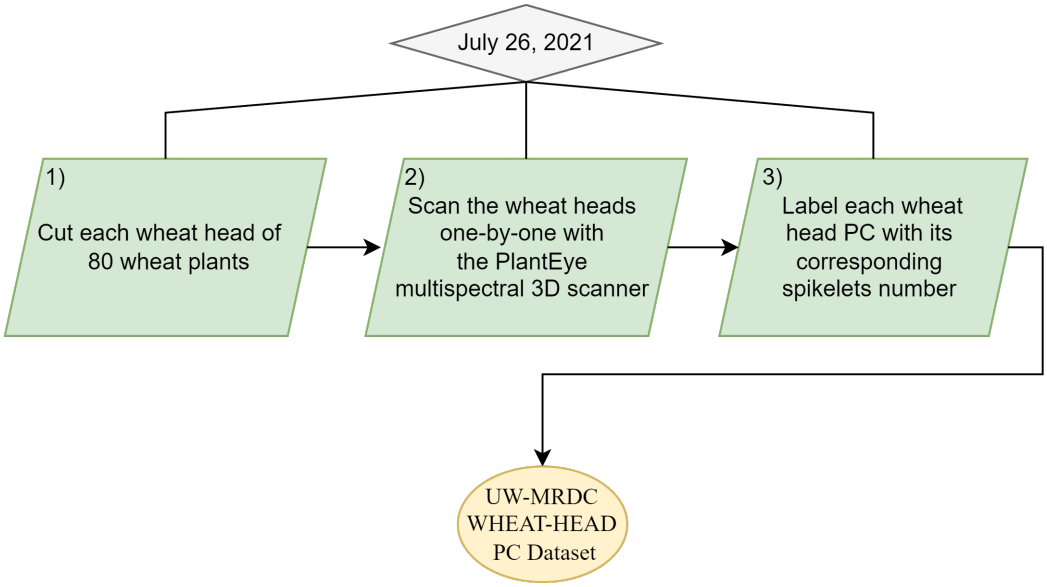


Figure 4.2: Diagram of the UW-MRDC WHEAT-HEAD PC dataset creation process.

4.1.2 Data Preprocessing

Figure 4.3 depicts the data preprocessing diagram which was developed to convert PCs into 3D images, where each voxel in the image corresponds to a point from the PC and is associated with a tuple of values. Here, a tuple can contain either RGB values, or NIR value, or RGB+NIR values. In general, the 3D images have variable sizes.

The input of the diagram is a PC that is generated by the PlantEye multispectral 3D scanner as a Polygon File Format (PLY) [93]. PLY files are designed to store 3D objects as lists of plane

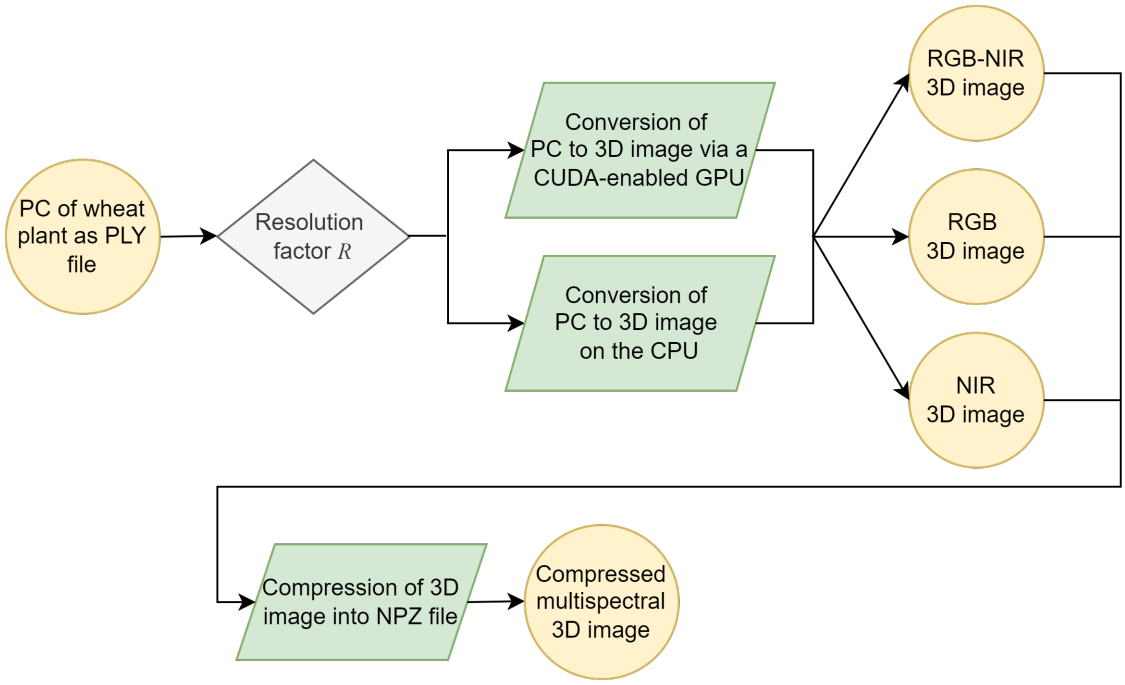


Figure 4.3: Diagram of preprocessing steps required to convert multispectral PCs into multispectral 3D images.

polygons (see Section 4.3.1 for more information). However, this representation does not fit the input requirements of CNNs, thus, a geometrical conversion is required. Further details about the reasons of the conversion are explained in Section 4.3. The next step in the process (shown in Figure 4.3 as a gray decision block) indicates the need to select a resolution factor R that controls the enlargement or reduction of the real dimensions of the PC during the conversion. When $R < 1$, the dimensions of the converted 3D image will be less than the real dimensions of the PC, while in case $R > 1$, the dimensions of the converted 3D image will be greater than the real ones. Consequently, when $R = 1$, the dimensions of the converted 3D image are kept the same as those of the PC. Once R is determined, the conversion algorithm can be executed either in parallel (via a CUDA-enabled GPU), or sequentially on a CPU. Both approaches generate three 3D images from a given PC, where voxels in each 3D image are associated with tuples consisting of RGB, NIR, and $RGB{+}NIR$, respectively. Both CPU and GPU applications generate the same

results, but with different execution times. Finally, the last part of the process is to compress the 3D images in order to reduce storage space and store them as NPZ files, which is a file format used to store compressed data and defined in Python's NumPy library [94].

4.1.3 Fusarium Head Blight Detection and Severity Index Estimation

Figure 4.4 represents an approach to automated SI estimation - for evaluating the degree of FHB infection - and it is based on several 3D CNN models. To create this automated system, a process consisting of three major models was developed. As shown by the first block in Figure 4.4, the first model detects the presence of FHB using a 3D CNN to predict whether a 3D image of wheat is infected with FHB or is WC. If the classifier predicts WC, the process is terminated since there is no need to estimate the SI of WC wheat. However, if it indicates FHB, the 3D image is simultaneously fed to a 3D CNN regression model that will estimate the total number of spikelets in a wheat head, and to a 3D CNN regression model that will estimate the total number of infected spikletes in the same wheat head. By combining both estimations, the SI is the ratio of the number of infected spikelets to the total number of spikelets. However, in this work, we only finalized the 3D CNN detection model and the first 3D CNN regression model, which are both highlighted in Figure 4.4 as the first and second blocks, while the second 3D CNN regression model, which is highlighted as red, is left for future work. Nonetheless, all the SI estimation model components were included in the diagram in order to demonstrate the general scope and objectives of this work.

4.2 UW-MRDC Dataset Creation

In this section, details about the dataset creation will be explained by describing the experimental setup of growing wheat plants in the growth chambers, detailing the data acquisition process that uses the Phenospex PlanEye multispectral 3D scanner, and describing the data labelling process.

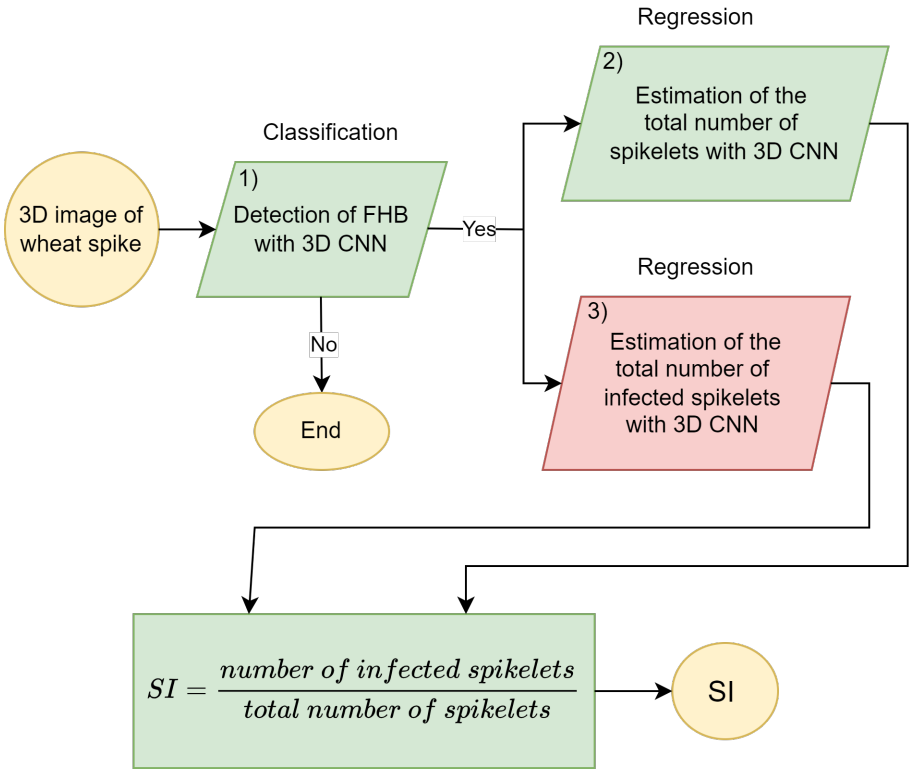


Figure 4.4: Diagram of SI estimation with 3D CNN.

4.2.1 Data Preparation

Plant material used in this study included the Canada Western Red Spring (CWRS) wheat cultivar 5602HR and CDC Teal. The 5602HR cultivar is moderately resistant to FHB and CDC Teal is susceptible. Planting and inoculation methods are identical to those described in reference [95]. A 3-acetyldeoxynivalenol producing isolate of Fusarium graminearum (Fg) (HSW-15-39), obtained from the Henriquez Spring Wheat (HSW) collection of Fusarium, was used in this study. In summary, seeds were sown in 3.5" pots with a mixture of 50% Sunshine soiless #5 mix (manufactured by Sun Gro, Horticulture) and 50% soil, plus 6 g slow-release Osmocote 14-14-14 fertilizer (manufactured by the Scotts Company). Plants were grown in controlled-environment cabinets with 16 hours (h) of light at 22°C and 8 h of dark at 15°C. Single floret inoculation at 50% anthesis with a 10 μL of Fg macroconidia suspension (5 $\times$ 104 macroconidia/mL), was performed between the lemma and

palea at the midpoint of the spike using a micro-pipette. Control plants were treated with sterile water. FHB severity was calculated counting the number of spikelets showing disease symptoms within each spike at 7, 14, and 24 days post inoculation (dpi). Figure 4.5 shows wheat spikes infected with FHB.

(a) Set of wheat spikes in the field.

(b) Close-up view of spikes infected with FHB.

(c) Close-up view of healthy spikes.

Figure 4.5: Plots of wheat spikes from the field of the MRDC showing both healthy and diseased wheat.

4.2.2 Data acquisition

The Phenospex PlantEye F500 multispectral 3D scanner [96] was used to scan all the plants in this work. It captures data non-destructively and delivers 3D representations of plants (via PCs) in real time. As shown in Figure 4.6, one or several plant containers are placed on the MicroScan under the PlantEye. Once the scanning is initiated, the PlantEye, supported by a rod, begins moving forward and emitting multispectral light beams onto the plants. The reflections of those beams are progressively acquired in high frequency (*i.e.* the frequency of emitting light beams and acquiring their reflections) to form a 3D representation of the plants, and the scanner additionally measures four light wavelengths (RGB and NIR) immediately after 3D acquisition. Multispectral information and 3D representation are then combined into a single PC. The light wavelength of the PlantEye ranges in nano-meters (nm) within [460,750], as shown in Figure 4.7, such that the peak

wavelengths in nm of the channels blue, green, red, and NIR range within $[460, 485]$, $[530, 540]$, $[620, 645]$, and $[720, 750]$, respectively. The direction of light beams emitted from the PlantEye of the multispectral 3D scanner are perpendicular to the surface of the ground. Once any of the light beams hits the surface of the scanned plant, its reflection is acquired by the PlantEye and used to estimate the spatial position of that surface point. During a scan, thousands of those acquired positions are stored and used to reconstruct a spatial representation of the external form or shape of the scanned plant in a 3D space.

(a) The PlantEye and Microscan consisting the Phenospex PlantEye F500 multispectral 3D scanner.

(b) Plants illuminated with the flashing light emitted from the PlantEye.

Figure 4.6: Phenospex PlantEye F500 multispectral 3D scanner.

The scanner mostly captures the plant as it appears from its top view, hence, scanning plants characterized by thin stems and thin heads such as wheat, barley, or oats is challenging. To overcome this limitation, wheat plants were laid horizontally while scanned in such a manner that beam lights will hit the widest surface of the plant to acquire maximum information on its form and shape.

3D models generated by the multispectral 3D scanner can be accessed through the HortControl web application [97], or any other software designed for 3D models. However, the use of HortControl is mandatory in controlling the scanner and setting the experimental parameters, such as the

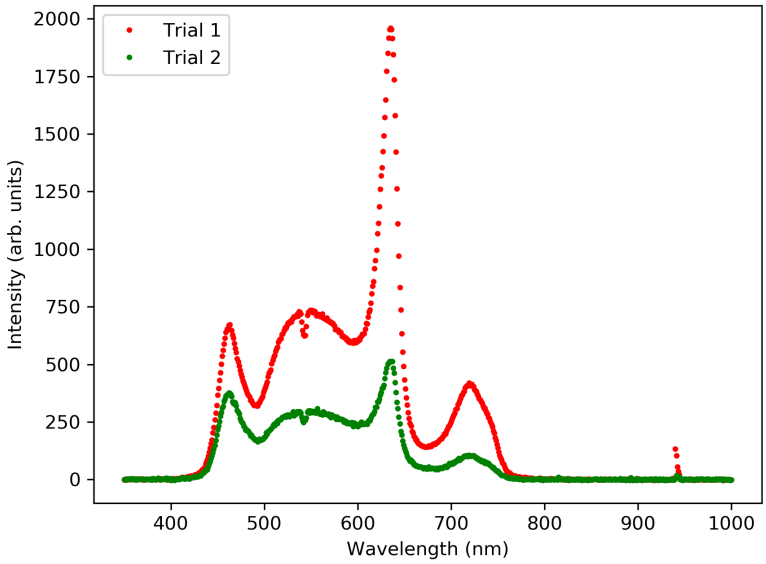


Figure 4.7: Spectral distribution of the light emitted from the PlantEye. Trail 1 and trial 2 depict two separate spectral distribution measurements that were measured at the University of Winnipeg via a spectrometer.

position of the pot under the flashing unit, the height of the pot, and the number of plants per scan. The web application also offers several visualization and data processing tools, such as a segmentation tool to remove the background or the pot from the 3D model while preserving the plant, as well as colour intensity reduction features to suppress soil or any undesired elements. In this work, HortControl was only employed to set the experimental parameters and scan the plant. The acquired data was then used to develop a CUDA-based processing techniques and ML models.

To acquire samples for the UW-MRDC WHEAT-PLANT PC dataset, 72 wheat plants were scanned three times each at 7-day intervals. During that time, the wheat plants were kept in their pots while being regularly watered to prevent water stress and permit FHB symptoms to continue spreading through the infected samples. The scanning dates were on the 12^{th}, 19^{th}, and 26^{th} of July, 2021. During these dates, plants were $10, 11$, and 12 weeks old, while the ones infected with FHB were at their $7, 14$, and 21 dpi, respectively. Table 4.1 contains a summary of the scanning dates,

wheat growth stage per date, number of samples per category (WC and FHB), and inoculation stage per date. Scanning the same plants at different stages was beneficial in many ways. First, it permitted the acquisition of more samples than the physical number of plants, and it enriched the dataset by producing wheat samples at different growth and infection stages. At the end of the experiments, a dataset consisting of 216 samples of wheat-plant PC was created.

Table 4.1: Summary of the scanning dates, wheat growth stage per date, number of samples per category (WC and FHB), and dpi for the UW-MRDC WHEAT-PLANT PC dataset.

Date	Stage	WC	FHB	dpi
July 12	10 weeks	58	14	7
July 19	11 weeks	58	14	14
July 26	12 weeks	58	14	21

The UW-MRDC WHEAT-HEAD PC dataset was created after finishing the last scanning experiments of the wheat plants dataset. First, all 72 spikes (*i.e.* wheat plant heads) were cut from their stems, in addition to 8 more spikes which were used to enlarge the dataset. Second, the spikes were scanned one by one. A final dataset consisting of 80 wheat-head PCs was created, in which 14 samples represent FHB infected spikes, while the remaining 66 represent WC ones.

4.2.3 Data Naming and Labelling

The naming convention chosen for the UW-MRDC WHEAT-PLANT PC dataset is in the form of {class}{unique_id}_{growth_stage}_{dpi}, such that

- *class* is a string whose value can be either FHB or WC,
- *unique_id* is a number representing the rank of the given sample within the class,

- *growth_stage* is a number representing the growth stage in weeks of the sample,

- *dpi* is a number representing the days post inoculation. It is only used for FHB samples.

These are two sample names from the wheat-plant dataset: FHB0_11_7, WC0_11.

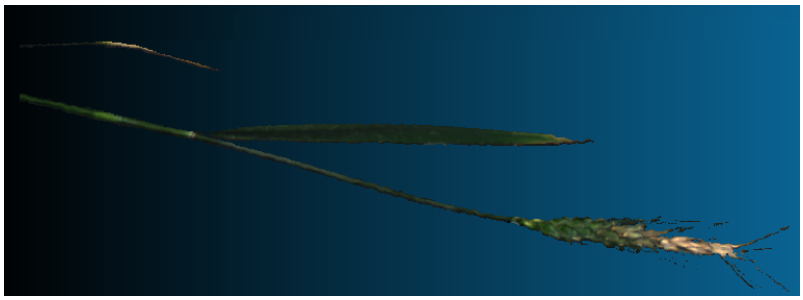
The naming convention chosen for the UW-MRDC WHEAT-HEAD PC dataset is in the form of {unique_id}_{number_of_spikelets}, such that

- *unique_id* is a number representing the rank of the given sample within the dataset.

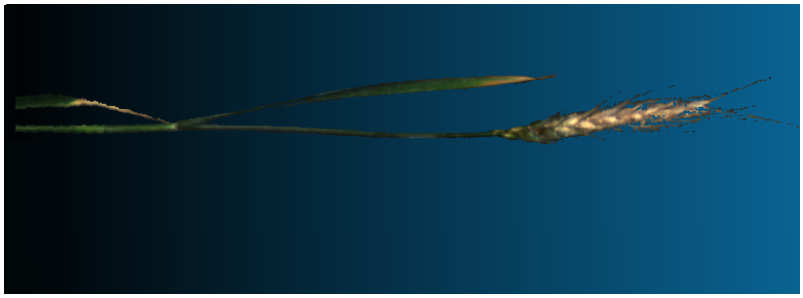
- *number_of_spikelets* is a number representing the total number of spikelets of the wheat head. It also represents the label of the sample.

These are two sample names from the wheat-head dataset: 0_17, 1_14.

Figure 4.8 shows screen shots of FHB0 using the RGB colour model and produced by CloudCompare [98], which is an open-source software for PC processing and visualization. Figure 4.8(a) shows FHB0_10_7, which is a FHB sample from the dataset at 10 weeks of growth stage and 7 dpi. The disease symptoms are quite visible to the naked eye, such that the upper half of the wheat head is yellow and shivered, while the lower part is still green. In Figure 4.8(b), FHB0_11_14 is shown at 11 weeks of growth stage and 14 dpi. The disease symptoms have evolved gradually and reached the lower spikelets of the wheat head. In Figure 4.8(c), FHB0_12_21 is shown at 12 weeks of growth stage and 21 dpi. The wheat head is fully infected, such that all the spikelets are visibly yellow and the spike looks shivered and skinny. Figure 4.9 shows three screen shots corresponding to the same FHB0 shown in Figure 4.8 but only in the NIR colour map produced by CloudCompare. With the NIR colour model, infected spikelets are coloured in shades of red, while healthy ones are seen in shades of green. Figure 4.10 shows eight samples from the UW-MRDC WHEAT-HEAD PCs dataset as seen from the CloudCompare software in the RGB colour model.



(a) FHB0_10_7.



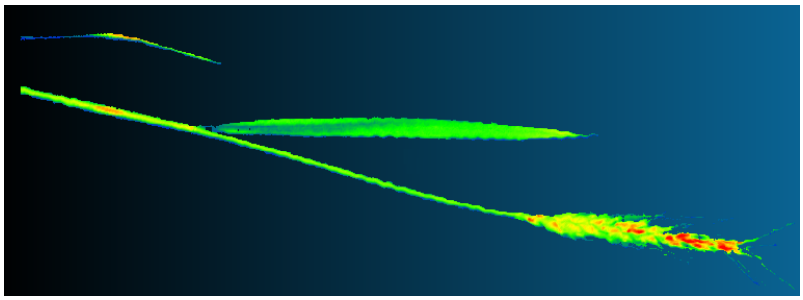
(b) FHB0_11_14.

(c) FHB0_12_21.

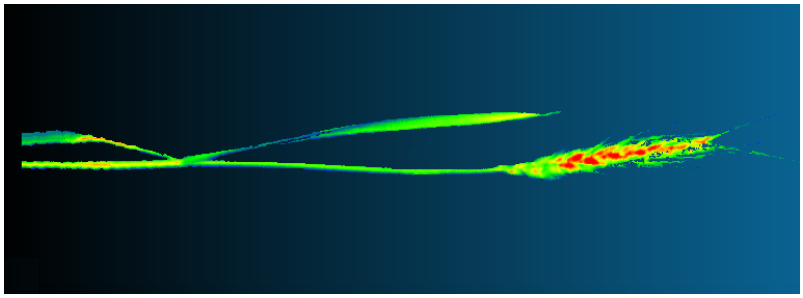
Figure 4.8: Visualizations of *RGB* PCs produced by CloudCompare software of FHB0 sample from the UW-MRDC WHEAT-PLANT PC dataset in three different growth stages and three different dpi.

4.3 Data Preprocessing with CUDA

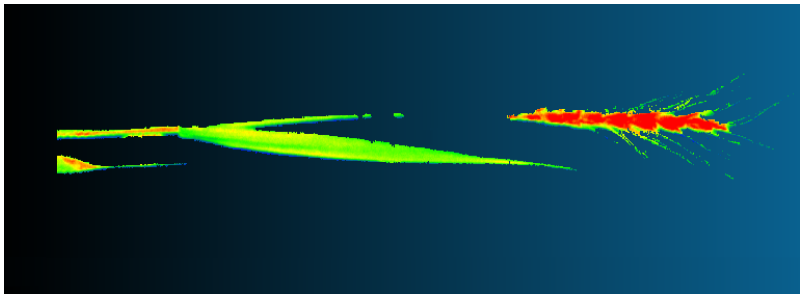
In this section, the definition of PC data and their limitations with regards to their use with 3D CNN models are discussed, and a parallel GPU algorithm is developed to convert PCs to a 3D image compatible with CNNs.



(a) NIR0_10_7.



(b) NIR0_11_14.

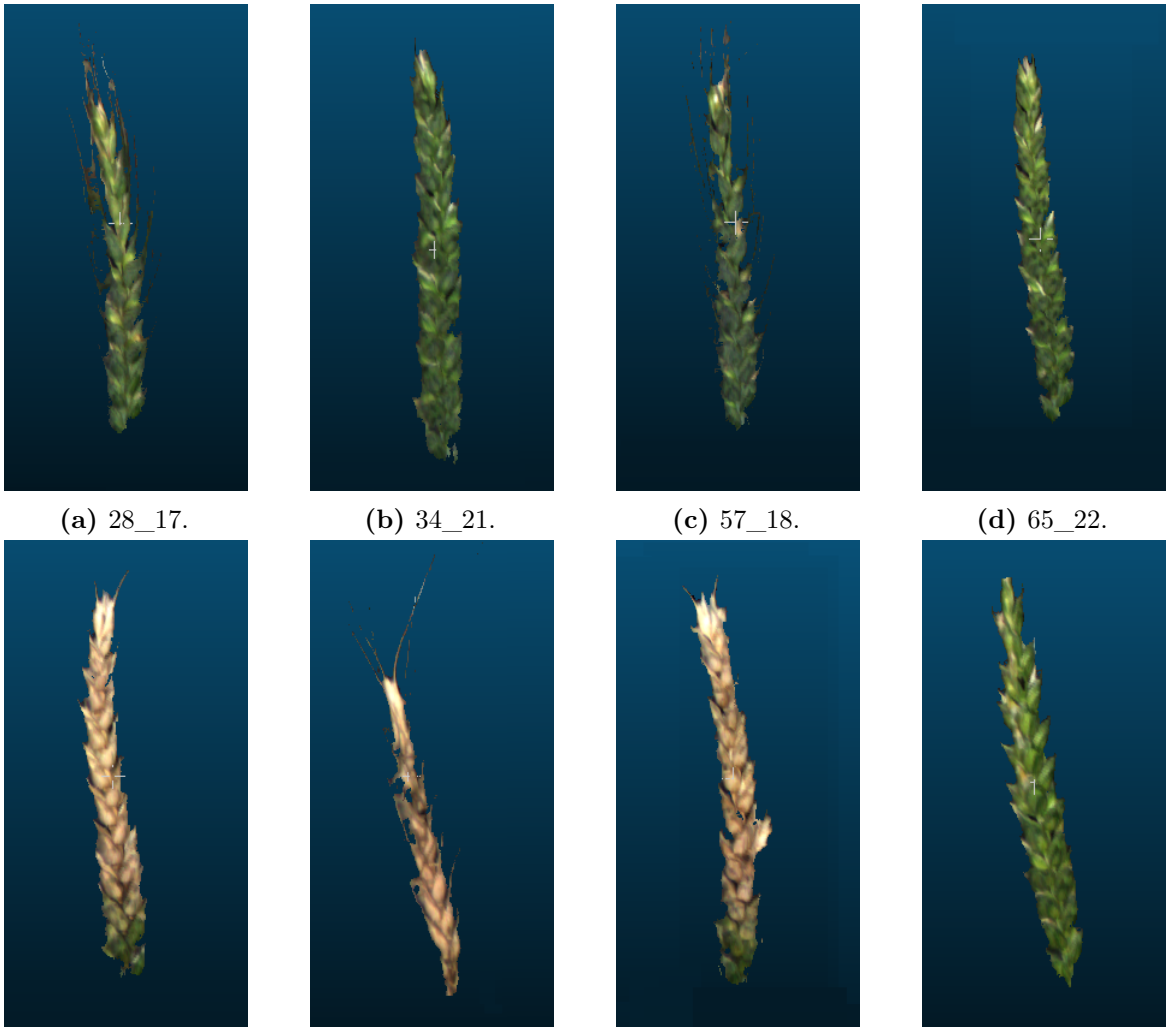


(c) NIR0_12_21.

Figure 4.9: Visualizations of NIR PCs produced by CloudCompare software of FHB0 sample from the UW-MRDC WHEAT-PLANT PC dataset in three different growth stages and three different dpi. Note, the NIR colour map was also determined by the CloudCompare software.

4.3.1 Motivation

The PlantEye multispectral 3D scanners generate 3D models of plants combined with multispectral information as PC data. PCs are becoming a powerful tool in many areas, such as Geographical Mapping, Computer Aided Design, and AI [99, 100, 101], due to their capability to represent 3D objects and physical spaces as tuples of 3D coordinates. In this work, a PC combines thousands (or millions) of spatial measurements and 4-channel colour information into one set of data. These



(a) 28_17. **(b)** 34_21. **(c)** 57_18. **(d)** 65_22.

(e) 78_20. **(f)** 79_13. **(g)** 77_19. **(h)** 81_21.

Figure 4.10: Visualizations of *RGB* PCs produced by the CloudCompare software of different wheat head samples from the UW-MRDC WHEAT-HEAD PC dataset.

measurements are point coordinates that identify the outer surface or shape of a plant. Each point is characterized by a tuple of (x, y, z) geometric coordinates in a 3D space, and a tuple of (R, G, B, NIR) colour intensities. The point coordinates are unique and independent, meaning that each point represents an individual laser scan measurement, and during the storage process of a PC, coordinates are saved independently from each other in no particular order.

After the acquisition in high frequency of the multispectral 3D representation of a plant, the PC

is stored as a polygon file format, known as PLY, which is a file format designed specifically to save 3D models. A PLY file contains tuples of flat polygons, in addition to tuples of colour information. Flat polygons and colour information are described by a tuple of (x, y, z) coordinate values varying between negative and positive floating-points, and a tuple of (R, G, B, NIR) intensity values where each value is stored as an integer varying between $[0, 255]$. Moreover, the tuple of point coordinates stored in a PLY file are unordered, such that each point is independent and unrelated to the remaining points within the tuple. The ensemble of points is useful to reconstruct 3D models in a space by placing each coordinate in its specific spatial position. However, PLY representation does not support complex operations such as convolutions and matrix manipulations that require points within a data signal to be correlated and organized such that a meaningful change in space or time between points can be defined.

To illustrate this, Figure 4.11 (a) shows a simplified representation of a PLY file defined in 2D. It is defined by a tuple of point coordinates each consisting of an (x, y) coordinate and a tuple of their corresponding colour information consisting of one value. In this case, a PLY file defines a geometrical shape in a 2D space. By observing the tuples of coordinates and colour, it is hard to determine the shape defined by the PLY file. Figure 4.11 (b) represents the conversion of the PLY file into a 2D image. By reading the pixel values within the image, it is easy to tell that the point coordinates define a square (the colour values defined in the PLY file are represented in red within the 2D image). In fact, PLY files are characterized by a compact representation of objects that does not include zero values into the tuple of colour. However, by including zero values into the converted 2D image, it contributed in illustrating the shape of the square.

Moreover, in Figure 4.12, we compared the outcome of the same element-wise product (which is the first step in a convolution operation) applied separately between both the PLY file representation and a 1D mask, and the converted 2D image and a 2D mask. In Figure 4.12 (a), a 1D mask consisting of 9 weights all equal to 1 was applied to the first 9 values of the tuple of colours, and

the outcome of the element-wise product is equal to 6. Whereas a (3×3) mask consisting of 9 weights all equal to 1 was applied to the top left (3×3) region within the 2D image and the result of the element-wise product is equal to 3. Since convolution operations can only be meaningful when applied on signals characterized by correlated samples (refer to Section 3.2.1 for more details), the result of the element-wise product obtained from the 2D mask and the 2D image is meaningful and correct. Therefore, the result of the element-wise product obtained from the 1D mask and the PLY file is not useful since it is different than the result obtained from the 2D mask and the converted 2D image. This shows that convolution operations can not be applied on a PLY file to obtain a meaningful result. Consequently, a PC to 3D image conversion method was applied in this thesis to overcome the limitations of the application of 3D CNNs on PLY files.

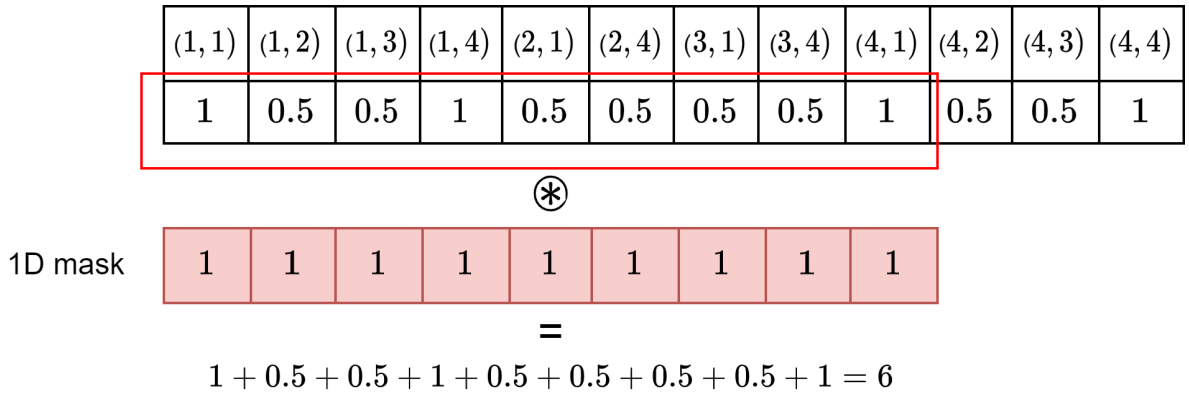
Tuple of points	$(1,1)$	$(1,2)$	$(1,3)$	$(1,4)$	$(2,1)$	$(2,4)$	$(3,1)$	$(3,4)$	$(4,1)$	$(4,2)$	$(4,3)$	$(4,4)$
Tuple of colours	1	0.5	0.5	1	0.5	0.5	0.5	0.5	1	0.5	0.5	1

(a) The representation of a PLY file defined in 2D.

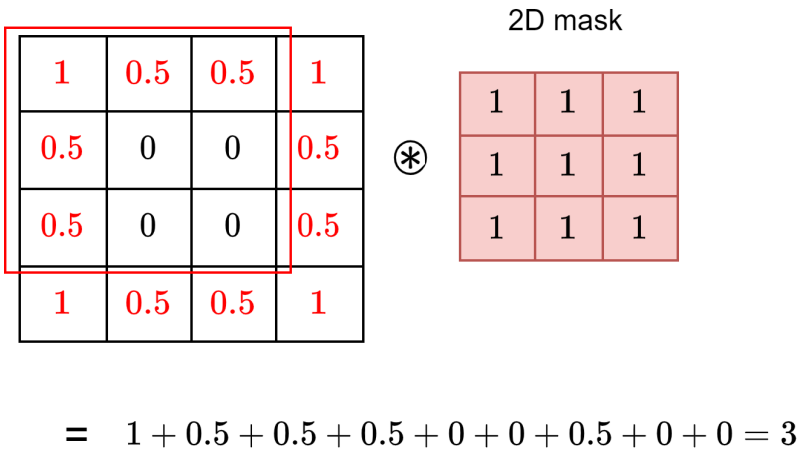
	1	2	3	4
1	1	0.5	0.5	1
2	0.5	0	0	0.5
3	0.5	0	0	0.5
4	1	0.5	0.5	1

(b) The representation of the converted PLY file in (a) into a 2D image.

Figure 4.11: The representations of a PLY file in 2D and its corresponding 2D image obtained from converting the PLY file into a 2D image.



(a) Element-wise product between a PLY file and a 1D mask.



(b) Element-wise product between a 2D image and a 2D mask.

Figure 4.12: Element-wise product to generate the first sample in the feature map on both a PLY file in 2D and its converted 2D image.

4.3.2 Point Cloud to 3D Image Conversion Theory and Implementation

In this section, the theory behind PC to 3D image conversion method is discussed. Moreover, a CUDA overview is presented and a diagram of CUDA implementation is discussed. Next, memory coalescing and PC to 3D image conversion kernel are explained.

4.3.2.1 Theory

In this work, the CUDA [102] parallel computing platform and programming model was employed to develop a C++ program that runs on a GPU and which converts PLY files into 3D images (see Section 4.3.2.2 for more details related to CUDA). Our proposed solution for converting PLY files into 3D images is based on linear interpolation, such that every point coordinate in the tuple of points stored in a PLY file is converted through linear interpolation into a new voxel coordinate within the constructed 3D image. This interpolation is necessary because the coordinates stored in a PLY file can be either negative or positive floating points, while the coordinates required by 3D CNNs have to be positive integers because 3D CNNs process 3D matrices, which are internally indexed by positive integers. The conversion operations are repetitive and separable, meaning that they can be applied independently to all the point coordinates in the PLY file, which provided a perfect opportunity to exploit GPU parallelism. Thus, the proposed CUDA-based method applies the same linear operations simultaneously on all the points in a PLY file. The conversion equations defining the linear interpolation along the x, y, and z-axes are

$$\begin{aligned} x_{matrix} &= \lceil a_x \; x_{PC} + b_x \rceil \,, \\ y_{matrix} &= \lceil a_y \; y_{PC} + b_y \rceil \,, \\ z_{matrix} &= \lceil a_z \; z_{PC} + b_z \rceil \,, \end{aligned} \tag{4.1}$$

such that a_x, a_y, and a_z are respectively the function slope corresponding to the x, y, and z-axes, and b_x, b_y, and b_z are respectively their intercepts. x_{matrix}, y_{matrix}, and z_{matrix} are the positions of the point along the width, height, and depth of the output 3D image, corresponding respectively to the transformation of x, y, and z values of a point coordinate in the PLY file, noted respectively as x_{PC}, y_{PC}, and z_{PC}. Moreover, $\lceil x \rceil$ defines the ceiling function of a real number x that is defined as the smallest integer that is not smaller than x. The function's slopes and intercepts are calculated as

$$a_x = \frac{\lceil R(\max_{1\leq i\leq N}(x_i) - \min_{1\leq i\leq N}(x_i))\rceil}{\max_{1\leq i\leq N}(x_i) - \min_{1\leq i\leq N}(x_i)}, b_x = -a_x \min_{1\leq i\leq N}(x_i),$$

$$a_y = \frac{\lceil R(\max_{1\leq i\leq N}(y_i) - \min_{1\leq i\leq N}(y_i))\rceil}{\max_{1\leq i\leq N}(y_i) - \min_{1\leq i\leq N}(y_i)}, b_y = -a_y \min_{1\leq i\leq N}(y_i), \tag{4.2}$$

$$a_z = \frac{\lceil R(\max_{1\leq i\leq N}(z_i) - \min_{1\leq i\leq N}(z_i))\rceil}{\max_{1\leq i\leq N}(z_i) - \min_{1\leq i\leq N}(z_i)}, b_z = -a_z \min_{1\leq i\leq N}(z_i),$$

such that N is the total number of points in the tuple of point coordinates in the PLY file, (x_i, y_i, z_i) is the coordinate of the i^{th} point in the tuple, and R is the resolution factor that serves to enlarge or reduce the resolution of the output 3D image. Finally, for the linear transformation, only the spatial coordinates (x, y, z) are used to estimate the new voxel coordinates (x_{matrix},y_{matrix},z_{matrix}), while their corresponding colour intensities (R, G, B, NIR) are reallocated in the new voxel coordinates within the 3D image. The dimensions of the output 3D image are

$$width = \left\lceil R(\max_{1\leq i\leq N}(x_i) - \min_{1\leq i\leq N}(x_i)) \right\rceil,$$

$$height = \left\lceil R(\max_{1\leq i\leq N}(y_i) - \min_{1\leq i\leq N}(y_i)) \right\rceil, \tag{4.3}$$

$$depth = \left\lceil R(\max_{1\leq i\leq N}(z_i) - \min_{1\leq i\leq N}(z_i)) \right\rceil,$$

such that width, height, and depth correspond to the range of values along the x, y, and z-axis, respectively.

4.3.2.2 CUDA Overview

CUDA is a programming model created by NVIDIA to enable software to run on GPUs and make use of their massive number of parallel execution units. Unlike CPUs that are designed to execute one operation per clock cycle and optimized to execute sequential operations with minimum latency, GPUs are designed to execute the same operation on multiple data at the same time following the SIMD (single instruction multiple data) execution strategy that favours high throughput. For example, to process an image consisting of millions of pixels by normalizing all its pixel values, the same normalization instruction needs to be performed as many times as the total number of pixels.

On a typical CPU, these instructions are organized in a queue and each pixel is processed per one clock cycle. However, on a GPU, all these instructions are conceptually performed simultaneously requiring a single clock cycle. In practice, the level of simultaneously executed instructions is determined by the amount of cores in the GPU. Regardless, this parallelism can tremendously reduce the execution time of a program by enabling the parallel computation of millions or trillions of the same floating-point operations with very high precision. Typically, real-world applications consist of both sequential code that runs on the CPU (called the host) and parallel code that runs on the GPU (called the device).

In order to exploit GPU parallelism, the C programming language can be used to create device code that runs on CUDA-enabled GPUs. This device code allows the development of kernel functions that are each executed by a grid of threads, consisting of multiple thread blocks, on a GPU device. Each block consists of up to 1024 threads, where a thread is an abstract representation of the smallest entity that executes a set of operations defined within a kernel. In comparison with a CPU, a thread executes a set of instructions in a sequential manner similarly to a processor. However, the strength of parallel programming is that threads within a block can execute the same set of operations simultaneously which results in very high execution throughput. Moreover, a kernel function can be programmed to execute the same instruction simultaneously using multiple grids consisting of multiple thread blocks. In the case of image pixel normalization, processing an image using CUDA would require programming a kernel function that performs one pixel value normalization and allocation on the device of a number of threads equal to the total number of pixels within the image. Consequently, the kernel function would be called once to be executed simultaneously using all the allocated threads, thereby processing all the image pixel values. Therefore, the CUDA model is a powerful tool for the development of applications that exhibit inherent parallelism for independent operations.

Executing a kernel function on the device is not straightforward. In fact, once a kernel function

is programmed for a GPU device, memory space for the data required to execute the kernel needs to be allocated in both the device memory and the host (CPU) memory. Then, data is transferred from host memory to device memory. Next, the kernel is launched to execute its operations, and lastly, the resultant data produced by the kernel is transferred from the device memory to the host memory in order to be exploited.

4.3.2.3 Diagram of CUDA Implementation

To implement Equations 4.1, 4.2, and 4.3, a general C++ API called hapPLY [103] was used to load PLY files. The API allows the reading and writing of a PLY properties, such as the point coordinates and their corresponding colour intensities, and loads them as two separate tuples of real-values. Figure 4.13 shows the implementation steps followed in the CUDA code to convert a batch of PLY files into a batch of 3D images. The code starts by reading the properties of a batch of PLY files. Processing the data in batches enables further optimization of parallel execution with CUDA, such that, in addition to processing all data points of a PC simultaneously, the code processes all data points of $n\times$PCs, where $n \in \mathbb{N}^*$. Next, all elements of the tuples of coordinates and tuples of colours of the $n\times$PCs are rearranged in a manner that ensures memory coalescing (see details in Section 4.3.2.4), which enables accessing consecutive memory locations within a single I/O operation. Following that, the maximum and minimum values of the coordinates needed to estimate the parameters of the interpolation functions and the dimensions of the output batch of 3D images are determined and used for calculations. Next, the memory space needed for the data that is used during the kernel execution is allocated on the device memory and the data is copied from the host memory to the device memory. Then, the conversion kernel, which is the function executed on the GPU, is launched to convert the batch of PLY files into their corresponding 3D images. Finally, the produced batch of 3D images is copied from the device memory to the host memory.

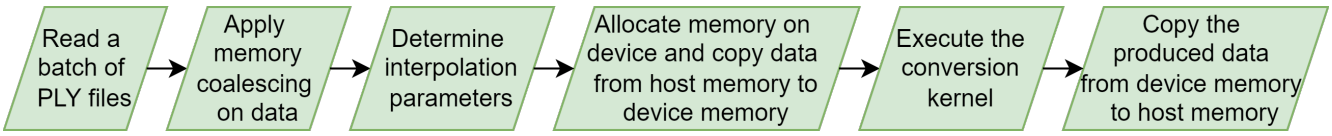


Figure 4.13: Diagram of CUDA implementation steps to convert a batch of PLY files into a batch of 3D images.

4.3.2.4 Memory Coalescing

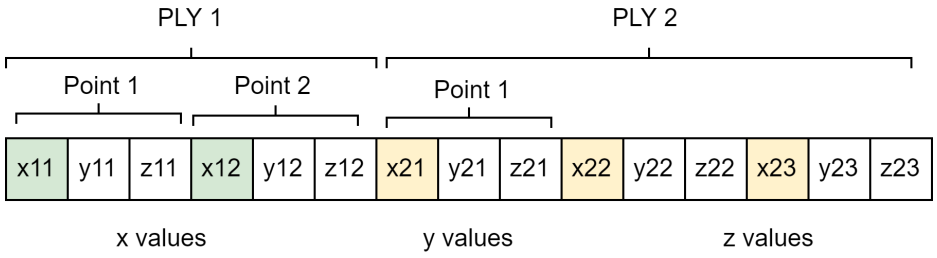
Threads within a thread block are organized into warps, where a warp is a group of 32 consecutive threads assigned to execute the same set of operations. In practice, threads within a warp access sequential memory locations for read and write operations. This means that memory access operations can be a major bottleneck for GPU applications if the data accessed by sequential threads in a warp is not sequentially ordered in memory. Therefore, the solution is memory coalescing, which is a technique used by CUDA where global memory accesses of threads in a warp are grouped together into one operation to minimize global memory bandwidth. In fact, each time a global memory location is accessed, a set of consecutive locations, including the requested location, are also accessed. Thus, the developer must ensure that the data used by consecutive threads in a warp are stored in sequential memory locations to minimize the latency caused by data access operations.

The kernel designed to perform the data conversion operations is programmed to estimate each value within the tuple of the point coordinates separately, meaning that x_{matrix}, y_{matrix}, and z_{matrix} are each estimated independently from one another. Thus, threads within a block are designed such that each block of threads is programmed to load and operate on either the x_{PC}, y_{PC}, or z_{PC} values to calculate either the x_{matrix}, y_{matrix}, or z_{matrix} values, respectively. The kernel architecture was developed to make use of memory coalescing during data loading, such that one thread within a warp loads all consecutive x_{PC}, y_{PC}, or z_{PC} values from the memory into the cache, enabling the rest of the threads to load their corresponding data directly from the cache to execute their

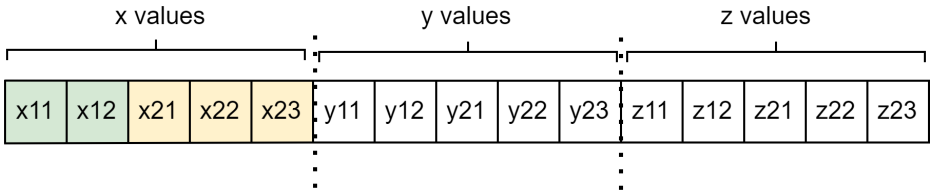
operations faster.

Figure 4.14 shows the data organization in a memory array for both coalesced and non-coalesced patterns. The illustrated examples use only a few points per PC for the purpose of demonstration only. Figure 4.14(a) shows a memory array of raw data storage where point coordinates corresponding to a batch of PLY files are arranged in the order of files and the order of (x, y, z) coordinates in a file. The first two point coordinates represent the first PLY file of the batch while the following points in the array correspond to the second PLY file within the same batch. $(x11, y11, z11)$ represents the first point of the first PLY file within the batch, followed by $(x12, y12, z12)$ which represent the second point of the first PLY file. Once all the points corresponding to the first PC within the batch are stored, points corresponding to the following PC are added to the same array. In the example, $(x21, y21, z21)$ represent the first point of the second PLY file, and so on. This kind of arrangement is not suitable for an optimized CUDA kernel execution because the memory accesses will be inefficient. Thus, to ensure that threads will access coalesced data locations during the kernel execution, point coordinates were rearranged in the memory. Figure 4.14(b) shows an array where all the x values representing all consecutive PLY files in a batch are placed consecutively, followed by all the y values, then all the z values within the array.

Not only are the tuple of point coordinates rearranged to support data coalescing, but also the tuple of colours. Colour intensities are loaded such that each (R, G, B, NIR) tuple corresponding to the first point of the first PLY file is the first element of the memory array, followed by the second (R, G, B, NIR) tuple corresponding to the first PLY file and so on. Thus, the tuples of colours are rearranged so that all R values representing the points of the first PLY file within a batch are put first in the memory array, followed by all the R values of the second PLY file, and so on. Once the R values are stored, G, B, and NIR values are then stored consecutively in the memory array according to the same memory coalescing principle.



(a) Non-coalesced memory organization.



(b) Coalesced memory organization.

Figure 4.14: Non-coalesced versus coalesced memory organization of data.

4.3.2.5 Conversion Kernel

The conversion kernel function was implemented to perform point coordinates transformation from their original spatial placement within the PC to their new voxel positions within the 3D dimensions of a 3D image. Each thread is designed to calculate the linear interpolation of a single point, which means that each thread will execute the linear interpolation functions defined in Equation 4.1 and related to the (x, y, z) values of a point coordinates. Firstly, the number of threads allocated on the device memory is determined to be $\frac{1}{3}$ of the coordinates list, and those threads are each programmed to execute three linear interpolation related to their designated (x, y, z) coordinates in order to determine the new voxel coordinates within the output 3D image. Next, each thread loads the (R, G, B) colour intensities and places the tuple of colours in their corresponding voxel position within the output 3D image. In fact, the interpolation functions defined in 4.1 convert floating-point coordinates into integer coordinates (with the ceiling operation) that define the voxel positions within the constructed 3D image. Moreover, in some cases, more than one real-valued point coordinate may get converted into the exact same voxel coordinate.

In that case, the newer point will override the existing one, resulting in a reduction in the total number of points defined in the 3D image. Furthermore, the size of the constructed 3D image, as defined in 4.3, ensures that the object defined in the PC is converted into a minimum bounding box, that is the generated 3D image. In addition, in order to create 3D images compatible with CNNs (unlike PLY file representations that do not include zero values and only use a compact representation, as explained in Section 4.3.1), the voxel values that remain empty after reassigning the colour tuples from their positions in the PC to their new voxel positions within the constructed 3D image, are set to zero. The conversion kernel described in this section produces 3-channel 3D images with each voxel value consisting of a tuple of (R, G, B) colour intensities, while NIR intensity values are processed through a second kernel to produce 1-channel 3D images.

4.3.3 Results, Complexity, and Execution Time

In this section, the results, complexity, and execution time related to the CUDA-based method for multispectral PC to multispectral 3D image conversion are discussed.

4.3.3.1 Results

The input of the kernel is two coalesced tuples: a tuple of point coordinates and a tuple of colour intensities. The tuples represent the data corresponding to a batch of PLY files (PCs), where the batch contains PCs of different lengths, meaning that each PC consists of a distinct number of points. The kernel is scalable, meaning that it is programmed to adjust the allocation of the number of threads depending on the size of the individual PCs within a batch. The output of the kernel is a batch of 3D images, where each 3D image within the output batch corresponds to a PC within the input batch of the same order. Both input and output batches contain the same number of files.

2D projections were done to visualize the 3D images produced by the CUDA conversion kernel on a 2D plane. Figure 4.15 shows three 2D projections of the PC of a soy plant. The PLY file of the soy plant was converted into three 3D images with different resolution factors (R). Figures 4.15(a), 4.15(b), and 4.15(c) represent 2D projections of the 3D image converted with R values equal to 0.5, 1, and 2, respectively. The resolution of the 2D projections differ depending on the value of R, such that in Figure 4.15(a) where $R = 0.5$, the image has a low-resolution due to the diminishing of the real dimensions by half during the conversion, while Figure 4.15(b) shows a higher resolution where more details and sharp edges can be observed due to the conservation of the real 3D dimensions of the PC during the conversion with $R = 1$. However, Figure 4.15(c) shows a better quality but with the appearance of a few artifacts due to doubling the original dimensions of the PC during the conversion with an R value equal to 2.

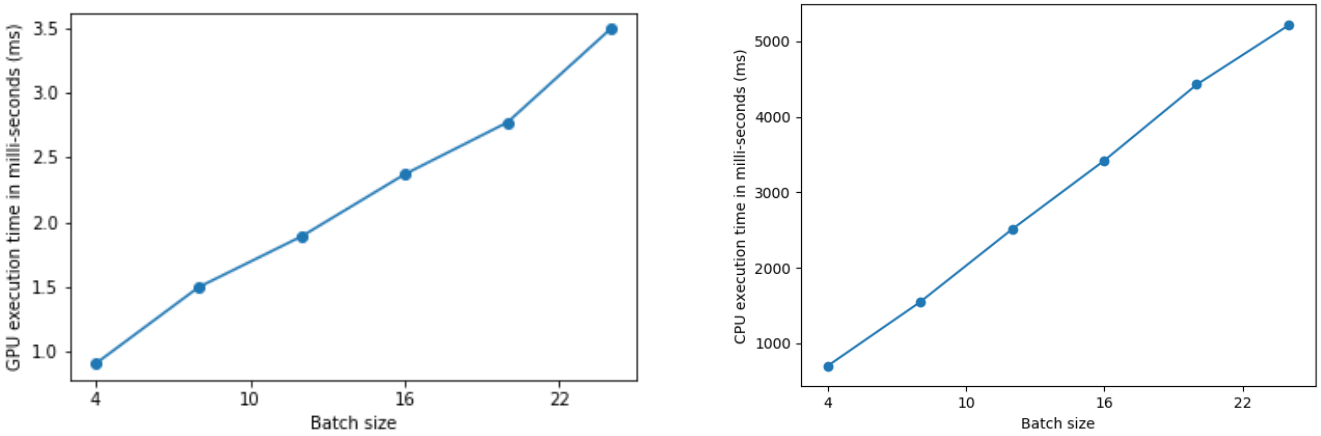
(a) $R = 0.5$. **(b)** $R = 1$. **(c)** $R = 2$.

Figure 4.15: 2D projections of a 3D image converted with different resolution factors R using the CUDA kernel.

4.3.3.2 Complexity and Execution Time

The complexity of the preprocessing method is $\mathcal{O}(N)$ since it carries out linear functions. Predominantly, the main gain from the CUDA-based implementation is the execution time, which achieved

a 1065 time reduction in runtime compared to the CPU version. Figure 4.16(a) shows the plot of the execution time of the conversion kernel on GPU with respect to the batch size that varies from 4 to 24 PLY files per batch. The plot shows a linear growth between the execution time in ms and the batch size. Figure 4.16(b) shows a plot of the execution time of the CPU-based conversion method with respect to batch size. The shape of the plot is also linear and similar to the plot of the execution time on the GPU, however, the execution time on the GPU is approximately 10^3 times less than the execution time on the CPU. In fact, the CPU method was only developed for validation purposes, therefore, the code was not optimized in any manner and it is fully sequential.



(a) GPU execution time.

(b) CPU execution time.

Figure 4.16: GPU and CPU execution times in ms with respect to batch size.

4.4 Detection of Fusarium Head Blight in Point Clouds of Wheat Using 3D Convolutional Neural Networks

In this section, details related to the creation of 3D CNNs for the task of detecting FHB in wheat are explained. Moreover, details related to the models training, datasets characteristics and data

resizing, the model architectures, and the results obtained from the experiments are all thoroughly described.

4.4.1 Experiments

In this section, monitored grid search and 5-fold cross-validation used to create and train the 3D CNN networks for FHB detection, respectively, are discussed. Moreover, the datasets characteristics and data resizing are explained, and the architectures and training parameters of the networks are discussed.

4.4.1.1 Monitored Grid Search

In our study, 3D CNNs were developed from scratch. A grid search over the number of layers and the number of neurons per layer was conducted. The objective of the grid search is to find the optimal 3D CNN architecture that produces the highest accuracy on the task of FHB detection from 3D images of wheat. The layers employed to build the models are 3D convolution layers, 3D max pooling layers, and densely-connected (or dense) layers; and the search space used to determine the optimal number of layers and neurons are the following:

- $\{3, 4, 5, 6\}$: Search space of the number of 3D convolution + 3D max pooling layers. The last layer of 3D convolution before the densely-connected layers is not followed by a 3D max pooling layer.

- $\{1, 2, 3, 4, 5, 6\}$: Search space of the number of densely-connected layers.

- $\{128, 64, 32, 16, 8\}$: Search space of the number of neurons per layer. The last densely-connected layer has always one neuron.

However, with these sets of variables, the number of possible combinations is equal to 380,835,000 networks. Definitely, to train millions of 3D CNN models, it would take months or maybe years to finalize. Thus, a monitored grid search was employed as an alternative to train only a small number of 3D CNN models. Hence, the monitored grid search works by randomly generating a batch of 20 networks at a time, such that a 5-fold cross-validation (CV) [104] (see Section 4.4.1.2 for details) is performed on each network in the batch, then the top three networks which achieve the highest average CV accuracy out of the 20 networks are retrained on the training set and evaluated on the test set. The overall architectures of the 20 models are given in Table 4.2 and a detailed description of the architectures of the best three networks are given in Tables 4.3 to 4.8 in Section 4.4.1.5. A 100% accuracy over the test set was achieved by one of the first 20 networks.

4.4.1.2 5-fold Cross Validation

Prior to training the 20 models, the data samples were divided into 90% training set consisting of 196 samples and 10% test set consisting of 23 samples. Since FHB detection is a binary classification, we ensured that the training set and the test set have the same class distribution with respect to FHB and WC classes. Next, the training samples were further split into 5 folds that have the same class distribution as the training set to perform the CV, such that each fold consists of 20% of the training data. Then, 5-fold CV was applied by training the models on the four training folds and validating it on the validation fold. The top three model architectures that achieved the highest average CV accuracy were retrained on the entire training set and evaluated on the test set, where the average CV accuracy refers to the average accuracy value achieved by the network trained on each fold of the 5 CV folds.

4.4.1.3 Datasets Characteristics

Three different versions of 3D image datasets were used in this work. Each dataset was used to train the 20 3D CNN networks. In fact, the datasets are obtained by converting the UW-MRDC WHEAT-PLANT PC dataset into 3D images with a resolution factor $R = 1$. However, the datasets differ by their pixel information, which are defined as

1. The 3D wheat-plant images in RGB (3DWP_RGB): In this dataset, the pixels of the 3D images contain RGB colour information (3 channels).

2. The 3D wheat-plant images in NIR (3DWP_NIR): In this dataset, the pixels of the 3D images contain NIR colour information (1 channel).

3. The 3D wheat-plant images in $RGB{+}NIR$ (3DWP_RGB_NIR): In this dataset, the pixels of the 3D images contain the $RGB{+}NIR$ colour information (4 channels).

4.4.1.4 Data Resizing

3D images within the datasets have different sizes, such that the width, height, and depth values corresponding to the 3D images dimensions are within $[\![25, 237]\!]$, $[\![85, 378]\!]$, and $[\![14, 384]\!]$, respectively. Since CNNs require input samples of a fixed size, resizing the totality of the 3D images within the datasets to the same size was required. The easiest option was to resize every 3D image to the maximum size that corresponds to $237 \times 378 \times 384$ px. However, this method raised the volume of the data tremendously, such that each resized 3D image contained 34,401,024 px. Training the models on big volume data consumes too much time and resources. Thus, a smaller fixed size was determined such that, any batch of resized 3D images could fit in the GPU memory along with any of the aforementioned model parameters. Since the height dimension of a 3D image represents the real height of a scanned wheat, it is important to preserve it as much as possible when resizing.

Therefore, a fixed size was determined by fixing the height to 300 and by calculating the width and depth via the average aspect ratios of the images in the datasets. As a result, the data samples were all resized to a fixed size equal to $75 \times 300 \times 95$ px while maintaining their respective original aspect ratios. This means that a 3D image is resized to the highest possible size that preserves the initial height–width proportion, preserves the height–depth proportion, and that is contained within $75 \times 300 \times 95$ px envelope. The resized images are then zero–padded to $75 \times 300 \times 95$ px.

4.4.1.5 Model Architectures

In Table 4.2, the architectures of each of the 20 models that were constructed by the monitored grid search (see Section 4.4.1.1 for details) are presented. The number of convolutional neurons refers to the number of neurons per 3D convolutional layer and the number of fully-connected neurons refers to the number of neurons per fully-connected layer. Even though the architectures of the models were randomly generated through the monitored grid search, only architectures with a descending order of the number of neurons per both convolution layers and fully connected layers were kept. In other words, given a layer l with a number of neurons equal to n_l, the number of neurons n_{l+1} in the direct subsequent layer $l+1$ have to be less than or equal to the number of neurons in layer l. (*i.e.* $n_{l+1} \leq n_l$). The choice of the decreasing number of neurons throughout the layers creates lighter models with a relatively small and condensed number of parameters. As mentioned in Section 4.4.1.1, every 3D convolutional layer is followed by a 3D max pooling layer except for the last 3D convolutional layer. By default, the last fully-connected layer has 1 neuron since the networks are solving a detection problem. For instance, model 1 has three 3D convolutional layers such that the number of neurons per layer from layer 1 to 3 are equal to 16, 8, and 8, respectively, and has four fully-connected layers such that the number of neurons per layer from layer 1 to 4 are equal to 128, 64, 8, and 8, respectively.

The detailed architectures of the three best performing networks per dataset are described

Table 4.2: Overall architectures of the 20 models created from the monitored grid search.

Model	# of convolutional neurons	# of fully-connected neurons
1	16,8,8	128,64,8,8
2	64,64,64,32,8	128,32,8
3	32,32,8,8	128,64,32
4	64,64,16	128,128,32
5	32,16,16,8	32,16
6	64,64,64,16	16
7	64,16,16,16	32,64,16
8	32,32,32,32,16	128,64,32,16
9	32,32,32,16,16	64,32,16,8
10	32,32,32,8,8	64,32,16
11	32,32,32,32,16	128,64
12	16,8,8,32,64	32
13	64,64,8,8,8	32,16
14	64,32,8	128,32,16,8
15	64,64,32	128,16,8
16	64,16,8,8	16,8
17	32,32,32,16	128
18	32,32	8,8,8
19	32,32,16,16,8	32
20	64,32,32	128

in Tables 4.3 to 4.7. Tables 4.3, 4.4, and 4.5 describe the architecture of models 8, 10, and 11, which are the top three 3D CNN networks that achieved the highest average CV accuracy on the 3DWP_RGB dataset. Moreover, both models 8 and 11 are two of the top three models that achieved the highest average CV accuracy on the 3DWP_RGB_NIR dataset. Model 8 consists of five 3D convolutional layers, each having 32 neurons, a convolution mask size equal to $(3 \times 3 \times 3)$, and followed by a 3D max pooling layer, except for the last convolutional layer that has 16 neurons, a kernel size equal to $(1 \times 3 \times 3)$, and is followed by a flattening layer. The purpose of the flattening layer is to transform the 3D feature map resulted from the convolutional layers into a 1D feature vector. The flattening layer is then followed by 5 densely-connected layers, where the number of neurons per layer is equal to 128, 64, 32, 16, and 1, respectively. The activation function in all the layers is a ReLU, except for the last densely-connected layer that uses a sigmoid activation

function. The total number of trainable parameters of model 8 is equal to 215,985.

Table 4.3: The architecture of the 3D CNN called model 8 trained on the 3DWP_RGB and the 3DWP_RGB_NIR datasets.

	Layer	# of neurons	Kernel size	Activation
Model 8	Conv3D	32	(3,3,3)	ReLU
	MaxPool3D	-	-	-
	Conv3D	32	(3,3,3)	ReLU
	MaxPool3D	-	-	-
	Conv3D	32	(3,3,3)	ReLU
	MaxPool3D	-	-	-
	Conv3D	32	(3,3,3)	ReLU
	MaxPool3D	-	-	-
	Conv3D	16	(1,3,3)	ReLU
	Flatten	-	-	-
	Dense	128	-	ReLU
	Dense	64	-	ReLU
	Dense	32	-	ReLU
	Dense	16	-	ReLU
	Dense	1	-	Sigmoid
# of trainable parameters	215,985			

Model 10 consists of five 3D convolutional layers, where the first three layers have each 32 neurons, a kernel size equal to $(3 \times 3 \times 3)$, and followed by a 3D max pooling layer. The fourth 3D convolutional layer has 8 neurons, a kernel size equal to $(3 \times 3 \times 3)$, and is followed by a 3D max pooling layer. The last 3D convolutional layers has 8 neurons, a kernel size equal to $(1 \times 3 \times 3)$, and is followed by a flattening layer. The flattening layer is followed by four densely-connected layers where the number of neurons per layer is equal to 64, 32, 16, and 1 from the first to the last dense layer, respectively. The activation function is a ReLU in all the layers except for the last densely-connected layer that uses a sigmoid activation function. The total number of trainable parameters of model 10 is equal to 96,849.

Finally, model 11 has exactly the same 3D convolutional, 3D max pooling, and flattening layers as model 8, but differs in its densely-connected layers, such that it has only three dense layers. The

Table 4.4: The architecture of the 3D CNN called model 10 trained on the 3DWP_RGB dataset.

	Layer	# of neurons	Kernel size	Activation
Model 10	Conv3D	32	(3,3,3)	ReLU
	MaxPool3D	-	-	-
	Conv3D	32	(3,3,3)	ReLU
	MaxPool3D	-	-	-
	Conv3D	32	(3,3,3)	ReLU
	MaxPool3D	-	-	-
	Conv3D	8	(3,3,3)	ReLU
	MaxPool3D	-	-	-
	Conv3D	8	(1,3,3)	ReLU
	Flatten	-	-	-
	Dense	64	-	ReLU
	Dense	32	-	ReLU
	Dense	16	-	ReLU
	Dense	1	-	Sigmoid
# of trainable parameters	96,849			

first two fully-connected layers have 128 and 64 neurons, respectively, each with a ReLU activation function per layer, while the last layer has 1 neuron and sigmoid activation. The total number of trainable parameters of model 11 is equal to 213,425.

Tables 4.6, 4.7, and 4.8 correspond to model 3, 5, and 9, respectively, which are the top three 3D CNN networks that achieved the highest average CV accuracy on the 3DWP_NIR dataset. Moreover, model 9 is one of the top three 3D CNN models that achieved the highest average CV accuracy on the 3DWP_RGB_NIR dataset. Model 3 has four 3D convolutional layers, where the first two layers have each 32 neurons, and the second two layers have each 8 neurons. The first 3D convolutional layers are followed each by a 3D max pooling layer, while the last layer is followed by a flattening layer. All 3D convolutional layers have a kernel size equal to $(3 \times 3 \times 3)$, and a ReLU activation function. Following the flattening layer, there are four densely-connected layers, and the number of neurons per layer is 128, 64, 32, and 1, respectively. The first three dense layers have a ReLU activation, while the last layer has a sigmoid function. The total number of trainable

Table 4.5: The architecture of the 3D CNN called model 11 trained on the 3DWP_RGB and the 3DWP_RGB_NIR datasets.

	Layer	# of neurons	Kernel size	Activation
Model 11	Conv3D	32	(3,3,3)	ReLU
	MaxPool3D	-	-	-
	Conv3D	32	(3,3,3)	ReLU
	MaxPool3D	-	-	-
	Conv3D	32	(3,3,3)	ReLU
	MaxPool3D	-	-	-
	Conv3D	32	(3,3,3)	ReLU
	MaxPool3D	-	-	-
	Conv3D	16	(1,3,3)	ReLU
	Flatten	-	-	-
	Dense	128	-	ReLU
	Dense	64	-	ReLU
	Dense	1	-	Sigmoid
# of trainable parameters		213,425		

parameters of model 3 is equal to 1,401,134.

Table 4.6: The architecture of the 3D CNN called model 3 trained on the 3DWP_NIR dataset.

	Layer	# of neurons	Kernel size	Activation
Model 3	Conv3D	32	(3,3,3)	ReLU
	MaxPool3D	-	-	-
	Conv3D	32	(3,3,3)	ReLU
	MaxPool3D	-	-	-
	Conv3D	8	(3,3,3)	ReLU
	MaxPool3D	-	-	-
	Conv3D	8	(3,3,3)	ReLU
	Flatten	-	-	-
	Dense	128	-	ReLU
	Dense	64	-	ReLU
	Dense	32	-	ReLU
	Dense	1	-	Sigmoid
# of trainable parameters		1,401,137		

Model 5 consists of four 3D convolutional layers, such that the number of neurons per layer are

equal to 32, 16, 16, and 8, respectively. The kernel size in all the layers is equal to $(3 \times 3 \times 3)$. The first three 3D convolutional layer are each followed by a 3D max pooling layer, while the last layer is followed by a flattening layer. Three densely-connected layers follow the flattening layer, such that the number of neurons per dense layer are equal to 32, 16, and 1, respectively. The activation function for all the layers is a ReLU, except for the last dense layer where its activation is a sigmoid function. The total number of trainable parameters of model 5 is equal to 365,353.

Table 4.7: The architecture of the 3D CNN called model 5 trained on the 3DWP_NIR dataset.

	Layer	# of neurons	Kernel size	Activation
Model 5	Conv3D	32	(3,3,3)	ReLU
	MaxPool3D	-	-	-
	Conv3D	16	(3,3,3)	ReLU
	MaxPool3D	-	-	-
	Conv3D	16	(3,3,3)	ReLU
	MaxPool3D	-	-	-
	Conv3D	8	(3,3,3)	ReLU
	Flatten	-	-	-
	Dense	32	-	ReLU
	Dense	16	-	ReLU
	Dense	1	-	Sigmoid
# of trainable parameters		365,353		

Finally, model 9 has the same 3D convolutional, 3D max pooling, and flattening layers as model 11, with only one difference in the number of neurons of the last two convolutional layers, which is equal to 16. Following the flattening layer are 5 densely-connected layers, with the number of neurons through in the layers are equal to 64, 32, 16, 8, and 1, respectively. All dense layers each have a ReLU as activation function, except for the last layer that uses a sigmoid function. The total number of trainable parameters of model 9 is equal to 134,305.

Table 4.8: The architecture of the 3D CNN called model 9 trained on the 3DWP_NIR and the 3DWP_RGB_NIR datasets.

	Layer	# of neurons	Kernel size	Activation
Model 9	Conv3D	32	(3,3,3)	ReLU
	MaxPool3D	-	-	-
	Conv3D	32	(3,3,3)	ReLU
	MaxPool3D	-	-	-
	Conv3D	32	(3,3,3)	ReLU
	MaxPool3D	-	-	-
	Conv3D	16	(3,3,3)	ReLU
	MaxPool3D	-	-	-
	Conv3D	16	(1,3,3)	ReLU
	Flatten	-	-	-
	Dense	64	-	ReLU
	Dense	32	-	ReLU
	Dense	16	-	ReLU
	Dense	8	-	ReLU
	Dense	1	-	Sigmoid
# of trainable parameters	134,305			

4.4.1.6 Model Training

To train each model, a batch size equal to 5 was used because there was not enough memory on the GPU to store a bigger batch (see Section 3.3.2.4). The RMSProp optimization algorithm was used to update each network's parameters, with a learning rate equal to $5e^{-4}$ (see Section 3.3.2.3). And, binary CE was used as loss function (see Section 3.3.1). Each model was trained for 100 epochs. To implement the 3D CNN models, we used the Python programming language and its open-source neural network library Keras [105]. We conducted the experiments on a NVIDIA Tesla P100 GPU server with 12GB of GPU memory.

4.4.2 Results

Table 4.9 shows the detection performance metrics of the top three 3D CNN models over the three versions of datasets. Models 8, 10, and 11 achieved the highest average CV values amongst the batch of 20 models on the 3DWP_RGB dataset (characterized by RGB 3D images) by achieving 88.42%, 87.36%, and 86.84% average CV accuracy, respectively. These three models were retrained and evaluated over the test set, and achieved 100%, 91.3%, and 91.3% test accuracy, respectively. Models 11, 8, and 9 are the top three models amongst the 20 models that attained the highest average CV accuracies on the 3DWP_RGB_NIR dataset (characterized by RGB+NIR 3D images) and achieved 87.36%, 86.84%, and 85.78%, respectively. These three models were retrained and tested on the dataset's test set, and despite model 11 having the highest mean CV accuracy, it did not beat model 8 in test accuracy. In fact, model 8 achieved 95.65% test accuracy, followed by models 11 and 9 that achieved 91.3% and 86.95% accuracy, respectively. Finally, models 3, 5, and 9 achieved the highest average CV accuracies on the 3DWP_NIR dataset (characterized by NIR 3D images) by attaining 84.21%, 83.68%, and also 83.68% average CV accuracy, respectively. Despite the fact that model 9 achieved the lowest mean CV accuracy amongst the top three models, it obtained the highest test accuracy of 86.95%, followed by model 5 and 3 that achieved 82.6% and 78.26%, respectively.

4.4.3 Discussion

Our empirical results show the superiority of the RGB colour model over NIR, and RGB and NIR combined in the task of FHB detection in wheat plants. In fact, adding NIR information to RGB reduced the accuracy of the FHB detection from 100% to 95.65%, which is a peculiar finding since, in general, adding more information to the data has the tendency to enrich it and to give more information that should positively impact the performance of the CNN. However, in our case, NIR information is observed to have a negative influence on the learning performance of the

Table 4.9: Evaluation metrics of the top three 3D CNN models with respect to the datasets on FHB detection in wheat.

Dataset	Models	Average CV accuracy	Test accuracy
3DWP_RGB	**Model 8**	**88.42%**	**100%**
	Model 10	87.36%	91.30%
	Model 11	86.84%	91.30%
3DWP_RGB_NIR	Model 11	**87.36%**	91.30%
	Model 8	86.84%	**95.65%**
	Model 9	85.78%	86.95%
3DWP_NIR	Model 3	**84.21%**	78.26%
	Model 5	83.68%	82.60%
	Model 9	83.68%	**86.95%**

classifiers from 3D images of wheat. Using data consisting of only NIR information resulted in the lowest accuracy of 86.95% on the task of FHB detection in 3D images of wheat. Thus, we concluded that the RGB channels are the most efficient colour channels for the task of FHB detection, while NIR information do not enhance the disease detection but rather reduces it. These findings may be explained by the fact that the symptoms of the FHB disease are clearly observed with the naked eye in wheat, and that there are potentially no hidden symptoms that could be enhanced with the NIR spectrum. Therefore, NIR channel can include misleading information to the CNNs rather than any valuable details.

4.5 Estimation of the Total Number of Spikelets in Point Clouds of Wheat Using 3D Convolutional Neural Networks

In this section, the total number of spikelets per wheat head is estimated using four regression models. Details related to the models creation and training, the data resizing, the model architectures, and the results are all explained thoroughly.

4.5.1 Experiments

In this section, monitored grid search and predefined model adaptation to create 3D CNN networks for the total number of spikelets estimation are discussed. Moreover, 5-fold CV, data characteristics, and data resizing are explained. Finally, the architectures and training setup of the 3D CNN networks are discussed.

4.5.1.1 Monitored Grid Search and Predefined Models Adaptation

Estimating the total number of spikelets per wheat head is crucial in determining the SI of infected wheat. Thus, creating a highly accurate and efficient CNN model that produces a reliable estimation is a crucial step towards finalizing the automated SI estimation tool. Hence, two approaches were followed to create regression models. In the first approach, 3D CNN networks were created from scratch through a monitored grid search. While in the second approach, three well-know CNN architectures were adapted for use with 3D data to solve the regression problem. These three networks were ResNet v1, RestNet v2, and DenseNet.

In the first approach, a monitored grid search over the number of layers and the number of neurons per layer was used to build five 3D CNN models for estimating the total number of spiklets. The monitored grid search used the same search space described in Section 4.4.1.1.

In the second approach; 3D ResNet v1, 3D ResNet v2, and 3D DenseNet models were created by adapting the ResNet v1, ResNet v2, and DenseNet models presented in Section 3.4. 3D ResNet v1 and 3D ResNet v2 were created by transforming every 2D convolutional layer and 2D average pooling layer into 3D convolutional layer and 3D average pooling layer, respectively. Moreover, the activation function of all the output layers was changed from a sigmoid function to a ReLU function. Similarly, a 3D DenseNet was created by changing every 2D convolutional layer and 2D average pooling layer in DenseNet into a 3D convolutional layer and a 3D average pooling layer,

respectively. Furthermore, the activation function of the output layer was changed from a sigmoid function to a ReLU function. In total, two 3D ResNet v1 networks, three 3D ResNet v2, and two 3D DenseNet were created. The architectures of the models are described in Section 4.5.1.4.

4.5.1.2 5-fold Cross Validation

Prior to training, the samples were divided into 90% training set, consisting of 70 samples, and 10% test set, consisting of 10 samples. Despite the problem being unrelated to the FHB disease and indifferent to the health of the wheat head, the equality of the class distribution data in all the splits was ensured in terms of FHB and WC samples. Next, the training samples were further divided into 5 balanced folds to perform 5-fold CV, such that each fold consists of 20% of the training data. Then, a 5-fold CV was performed on each model, meaning that a 5-fold CV was performed on each of the five 3D CNN networks, the two 3D ResNet v1, the three 3D ResNet v2, and the two 3D DenseNet. Then, the network that achieved the highest average CV accuracy per model was retrained on the training set and evaluated on the test set. The training parameters and results corresponding to the best performing models are described in Sections 4.5.1.5 and 4.5.2,

4.5.1.3 Dataset Characteristics and Resizing

Based on our results obtained from the classification problem, only the RGB colour channels were used for the regression task. Moreover, the UW-MRDC WHEAT-HEAD PC dataset was converted into a 3D image dataset with a resolution factor $R = 1.5$. Then, all the 3D images were resized into $(161 \times 51 \times 93)$ px, which correspond to the maximum width, height, and depth within the dataset samples. The labels of the dataset samples are integers that vary between 7 and 22.

4.5.1.4 Model Architecture

In this Section, all the model architectures were chosen in order to fit into the GPU memory. The architectures of each of the five models that were built through the monitored grid search are shown in Table 4.10 (see Section 4.5.1.1 for details). The number of convolutional neurons refers to the number of neurons per 3D convolutional layer and the number of fully-connected neurons refers to the number of neurons per fully-connected layer. As mentioned in Section 4.4.1.1, every 3D convolutional layer is followed by a 3D max pooling layer except for the last 3D convolutional layer. By default, the last fully-connected layer has 1 neuron. For instance, model 1 has two 3D convolutional layers such that the number of neurons per layer from layer 1 to 2 are equal to 32 and 16, respectively, and has three fully-connected layers such that the number of neurons per layer from layer 1 to 3 are equal to 128, 64, and 8, respectively.

Table 4.10: Overall architectures of the five 3D CNN models generated by the monitored grid search.

Model	# of convolutional neurons	# of fully-connected neurons
1	32,16	128,64,8
2	32,32,8	128,16
3	32,16,16,16	64,8
4	32,16,8,8,8	32,32,16
5	32,32,32,32	128

With respect to the 3D CNN models developed from scratch, Model 5 was the best performing model, since it achieved the best average CV MAE. As shown in Table 4.11, its architecture repeats four blocks consisting of a 3D convolutional layer and a 3D max pooling layer, where the kernel size of each convolutional layer is equal to $(3 \times 3 \times 3)$. Following the convolutional layers is a flattening layer, then two densely connected layers where the number of neurons per dense layer are equal to 128 and 1, respectively. The activation function in all the layers is a ReLU function.

Next, three 3D ResNet v1 networks were created, such that each network consists of one, two, and three residual blocks, respectively, and their depths are equal to 8, 14, and 20 layers,

Table 4.11: The architecture of the 3D CNN called model 5 trained on the *RGB* UW-MRDC WHEAT-HEAD PC dataset.

	Layer	# of neurons	Kernel size	Activation
Model 5	Conv3D	32	(3,3,3)	ReLU
	MaxPool3D	-	-	-
	Conv3D	32	(3,3,3)	ReLU
	MaxPool3D	-	-	-
	Conv3D	32	(3,3,3)	ReLU
	MaxPool3D	-	-	-
	Conv3D	32	(3,3,3)	ReLU
	MaxPool3D	-	-	-
	Flatten	-	-	-
	Dense	128	-	ReLU
	Dense	1	-	ReLU

respectively (see Section 3.4.1). The best performing model in the 5-fold CV for 3D RestNet v1 models was an architecture with a depth equal to 20 layers.

Similarly, the next models investigated were two 3D ResNet v2 models, such that each network consists of one and two residual blocks, respectively, and their depths are equal to 11 and 20 layers, respectively (see Section 3.4.1). Here, the best performing model in the 5-fold CV was an architecture with a depth equal to 11 layers.

Finally, two 3D DenseNet networks were created, each having a 4-layer dense block and a 5-layer dense block with depth values equal to 23 and 29 layers, respectively. Per each model, each dense layer was preceded by a bottleneck layer (see Section 3.4.2), and each dense or bottleneck layer was followed by a dropout layer with a dropout rate equal to 0.2 (see Section 3.3.3). The best performing 3D DenseNet model in the 5-fold CV was one with a depth equal to 23 layers.

4.5.1.5 Model Training

Once 5-fold CV is completed, the model architectures that achieved the best average CV MAE were trained on the full training set and tested on the test set. Table 4.12 shows the training parameters

(depth, optimizer, regularizer, epochs, batch size, and number of parameters) corresponding to these best performing models. Moreover, the optimizer, regularizer, epochs, and batch size also represent the training parameters for all the models. Starting with the 3D CNN, a batch size equal to 24 was used (see Section 3.3.2.4). Adam optimization algorithm was used to update the network parameters with a learning rate equal to $1e^{-3}$ (see Section 3.3.2.3). And MSE was used as the loss function (see Section 3.3.1). The 3D CNN was trained for 200 epochs, and its total number of parameters was equal to 184,225.

To train both 3D ResNet v1 and v2 models, an Adam optimizer was employed with a learning rate equal to $1e^{-3}$. Both models employed L2 regularizer with a regularization factor equal to $1e^{-4}$, and MSE as the loss function. 3D ResNet v1 and 3D ResNet v2 used a batch size equal to 12 and 6, respectively, and their total numbers of trainable parameters were equal to 810,145 and 674,561, respectively. Both models were trained for 200 epochs.

Finally, to train 3D DenseNet, an Adam optimizer was employed with a learning rate equal to $1e^{-3}$ and a dropout regularizer was used. The model was trained for 200 epochs with a batch size equal to 4. The total number of trainable parameters of 3D DenseNet was equal to 54,363. An NVIDIA Tesla P100 GPU server with 12GB of GPU memory was used to conduct all the experiments.

Table 4.12: The training parameters of the best performing architecture per model in the estimation of the total number of spikelets.

Model	Depth	Optimizer	Regularizer	Epochs	Batch size	# of trainable parameters
3D CNN	10	Adam	None	200	24	184,225
ResNet v1	20	Adam	l2	200	12	810,145
DenseNet	23	Adam	Dropout	200	4	58,363
ResNet v2	11	Adam	l2	200	6	674,561

4.5.2 Results

Table 4.13 shows the results corresponding to the best performing models (3D CNN, 3D ResNet v1, 3D ResNet v2, and 3D DenseNet) in the regression problem on the UW-MRDC WHEAT-HEAD PC dataset. The table shows the performance metrics of the models which are; the average CV MAE, the test MAE, and the average prediction time per sample in milliseconds. Both the 3D CNN and ResNet v2 achieved the best test MAE equal to 1.13. However, the 3D CNN outperformed 3D ResNet v2 in the prediction time per sample with 14 ms versus 112 ms for 3D RestNet v2. Moreover, even though 3D ResNet v1 obtained the best average CV MAE of 0.91, it failed to obtain it on the test set with a MAE of 1.23. However, 3D ResNet v1 produced the second best prediction time of 62 ms per sample. Although 3D DenseNet was ranked last in terms of average CV MAE in the group of the models by obtaining 1.28 average CV MAE, yet it succeeded in achieving 1.19 MAE on the test set, which is ranked third. 3D DenseNet also achieved the last average prediction time per sample equal to 140 ms.

Table 4.13: The evaluation metrics of the best regression models in the estimation of the total number of spikelets.

Model	Average CV MAE	Test MAE	Prediction time (ms)
3D CNN	1.26	**1.13**	**14**
ResNet v1	**0.91**	1.23	62
DenseNet	1.28	1.19	140
ResNet v2	1.05	**1.13**	l42

4.5.3 Discussion

The empirical results obtained from the 3D CNN regression model in predicting the number of spikelets per wheat head are very promising. The true labels of the UW-MRDC WHEAT-HEAD PC dataset are integer values ranging between 7 and 22, yet the 3D CNN succeeded in efficiently

predicting these numbers with an error value of only 1.13, meaning that, the difference between the predicted output and the real output is on average equal to 1.13. Even though the MAE is not negligible, it is still considered a very neat result that can be further improved, and it is still much better than the rough estimation that is manually performed by humans. Currently, the MAE of the models could be impacted by the human error that occurred during the manual spikelet counting that was used to label the samples of the UW-MRDC WHEAT-HEAD PC dataset. In addition, the model could be constrained due to the reduced number of samples in the dataset since CNNs require a large number of labelled examples in order to optimize their training process and acquire enough information to be able to generate accurate feature maps and produce results of high precision and low error. Despite the limitations, creating a regression model that estimates the total number of spikelets per wheat spike was successful. The next step would be to estimate the SI by creating a second regression model that estimates the total number of infected spikelets in a wheat head.

4.6 Chapter Summary

In this work, two novel datasets of multispectral PCs of wheat were created. The first dataset, called UW-MRDC WHEAT-PLANT PC, consists of 216 samples of multispectral PCs of wheat plants, such that every PC was acquired by scanning both the wheat spike and stem. The dataset includes two classes of samples, such that the first class represents WC wheat, while the second class represents FHB-infected wheat. The dataset includes varying FHB wheat samples that differ by their dpi stages. The second dataset, called UW-MRDC WHEAT-HEAD PC, consists of 80 samples of multispectral PCs of wheat heads, such that each PC was acquired by scanning the wheat spike only. The dataset includes samples of different classes, such that some of the samples represent WC spikes while the remaining ones represent FHB spikes. However, the labels of the

UW-MRDC WHEAT-HEAD PC dataset samples are the value of the total number of spikelets of each wheat head. The labels are integers in the range $[\![7, 22]\!]$.

A CUDA-based preprocessing method was developed to convert multispectral PCs into multispectral 3D images that can be used to perform complex mathematical operations such as convolutions. Consequently, all the samples of the UW-MRDC WHEAT-PLANT dataset and the UW-MRDC WHEAT-HEAD dataset were transformed with the CUDA-preprocessing method to generate multispectral 3D images that fit with the CNN processing and training requirements. Then, the UW-MRDC WHEAT-PLANT PC dataset was employed to train a novel and real-time 3D CNN for FHB detection from multispectral 3D images of wheat. The approach replaces the manual and time-consuming task of visually detecting a diseased wheat by an accurate, reliable, and automated model that detects (in few milliseconds) a diseased wheat regardless of the severity and the stage of the infection. The 3D CNN model for FHB detection in wheat achieved an accuracy of 100% on the UW-MRDC WHEAT-PLANT PC dataset in the RGB colour model. The performance of the 3D CNN in the FHB classification on 3D images in the NIR and NIR+RGB colour models is slightly less than 100%. The UW-MRDC WHEAT-HEAD PC dataset was used to train four models for the estimation of the total number of wheat spikelets from a multispectral 3D image of a wheat head. This automated method replaces the manual, error-prone, and time-consuming traditional approach that is widely employed by farmers and researchers to estimate the number of spikelets. Two models (3D CNN and 3D ResNet v2) succeeded in achieving an efficient number of spikelets estimation with a 1.13 MAE.

Chapter 5

Myocardial Infarction Detection in Echocardiography

The content of this chapter was adapted from [55], a paper accepted on the 20^{th} of September, 2021 by *Springer's Multimedia Tools and Applications* Journal under its Special Issue entitled *Engineering Tools and Applications in Medical Imaging*.

The outline of this chapter is as follows. In Section 5.1, we explain the pipeline architecture and discuss details related to the dataset. In Section 5.2, we explain the preprocessing techniques applied to the dataset which is used as input to a 2D CNN, as well as the details related to the 2D CNN architecture. In Section 5.3, we describe data preprocessing techniques applied to the processed videos before feeding it to the 3D CNN, as well as the details related to the 3D CNN architecture. In Section 5.4, we describe the training processes and the evaluation metrics related to each model, followed by a discussion of the results. Finally, in Section 5.5, we present the chapter's summary.

5.1 Methodology

The main goal of our work is to create a fully automated system for LV segmentation and MI detection in echocardiography in order to assist technicians and cardiologists in the process of diagnosing a patient affected or suspected with MI. This system must be robust, and as accurate and efficient as possible. In emergencies, for example, echocardiography acquisition tends to be made hastily, which may negatively impact the video's quality and content. Moreover, most of the employed echocardiogram machines produce videos of low-resolution and which their frame rate is generally below 30 frames per second (fps). In the following sections, we give an overview of the pipeline architecture and a description of the echocardiography videos acquired for this work.

5.1.1 Pipeline Overview

Figure 5.1 illustrates the flow of the automated pipeline where the input consists of *echocardiography video frames*, and the output is the MI detection result, *abnormal (MI) or normal (N)*. The echocardiography frames are processed by the sliding window technique which divides each frame into spatial windows of equal dimensions. The spatial windows are passed through the 2D CNN to segment the LV from each frame's spatial windows. Once the segmented windows are produced, they are reassembled into segmented frames. These segmented frames are reassembled to produce a segmented video, where the order of appearance of each segmented frame is kept in the same order of appearance as in the original echocardiography video. The segmented video frames are labelled in Figure 5.1 as the *segmented LV from the echocardiography video frames*. The segmented video is then processed by a temporal sliding window to produce temporal windows consisting of the the same number of frames. The temporal windows are the input of a 3D CNN that classifies them into one of the two classes: MI or N. The final class of the input video is estimated as the statistical mode of all the predictions of the frames constituting the video.

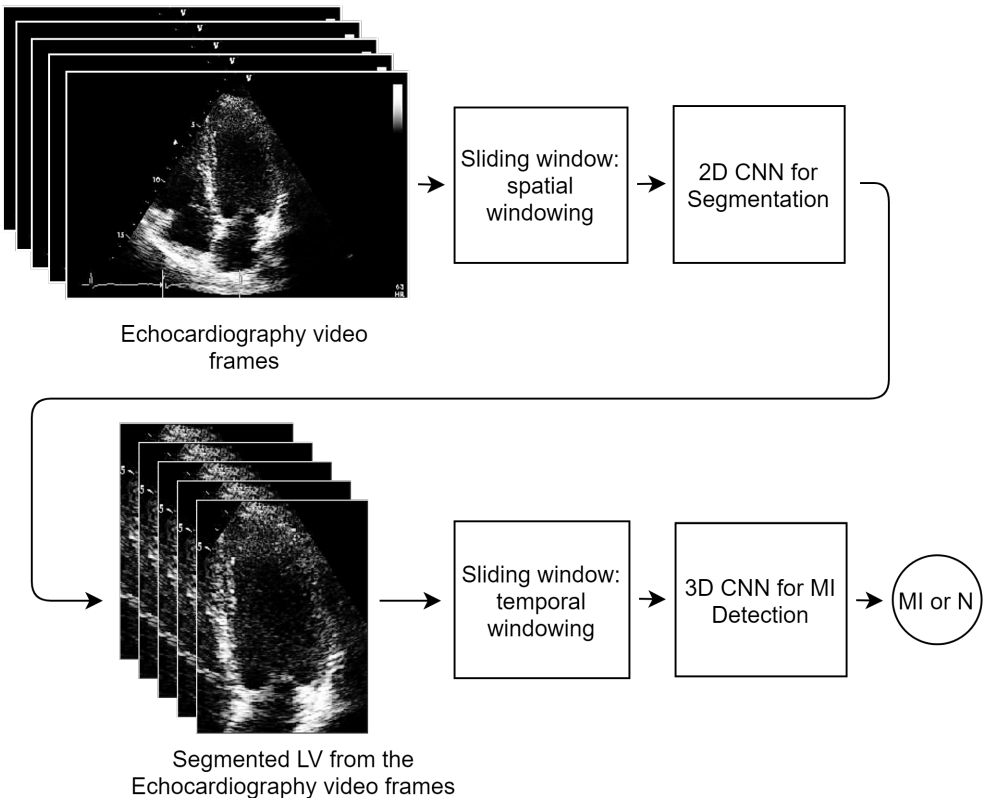


Figure 5.1: Fully automated pipeline for MI detection, where the input is an echocardiography video and the output is the prediction results.

5.1.2 HMC-QU Dataset

In collaboration with our cardiologist co-authors practising at HMC Hospital and Qatar University, we have used a dataset of 162 annotated echocardiography videos from the study number MRC-01-19-052 approved by the Medical Research Centre at HMC. The dataset contains 162 A4C view recordings of 2D echocardiography, obtained between 2018 and 2019, and were approved for scientific use by the local ethics board of HMC in February 2019. The echocardiography videos have a frame rate of 25 fps and their spatial resolution varies between 422×636 px and 768×1024 px. Furthermore, several echocardiography videos have corrupt content due to either noise or distorted representation of the A4C view, which usually consists of missing parts of the heart chambers that could not be acquired during acquisition. In this work, we used the entire HMC-QU

dataset which included videos with both reasonable quality and poor quality. The dataset became publicly available in February 2021.

This work focuses on performing early MI detection by learning RWMA of the LV wall in the A4C view echocardiography. Figure 5.2 shows a captured frame representing the A4C view, which consists of four distinct heart chambers, numbered from 1 to 4, where 1 identifies the LV, 2 to 4 identify the right ventricle, the left atrium, and the right atrium, respectively.

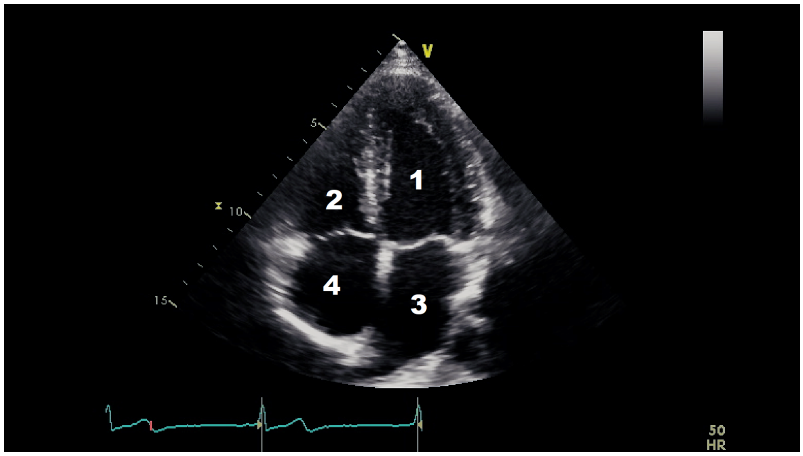
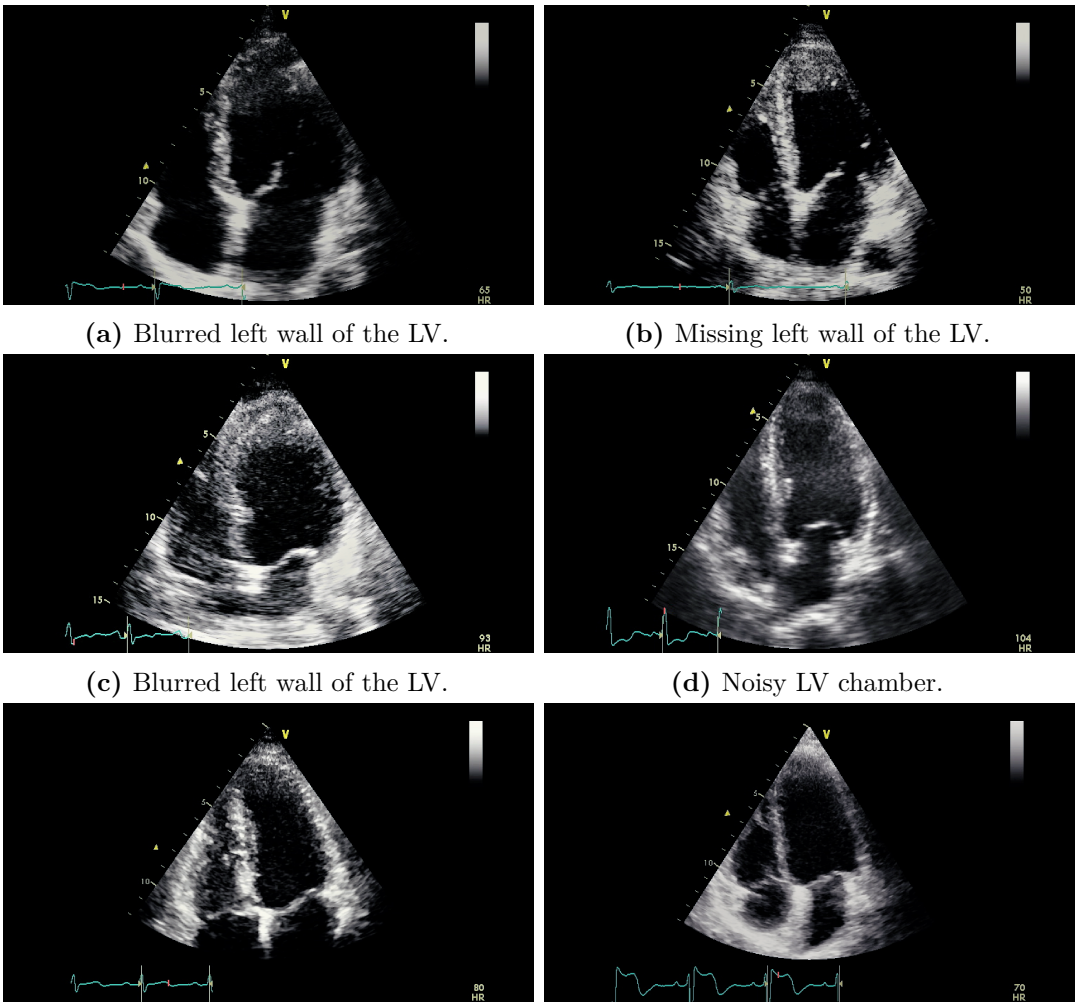


Figure 5.2: The apical four-chamber view. The numbers from 1 to 4 marking the four different chambers correspond respectively to the LV, the right ventricle, the left atrium, and the right atrium.

Figure 5.3 represents captured frames representing the quality of several videos from our dataset, which vary from good to noisy. Figures from 5.3(a) to 5.3(f) correspond to distinct frames each captured from a different video. We notice that in Figure 5.3(a) the left wall of the LV is blurred. Also, in Figure 5.3(b), the left wall of the LV is blurred and almost missing. In the same way, we observe that the totality of the LV wall is blurred in Figure 5.3(c); and that the interior of the LV is disrupted with noise in Figure 5.3(d). Finally, both Figure 5.3(e) and Figure 5.3(f) show acceptable LV representations, where the LV walls are captured and the chamber's interior is empty from noise. Moreover, since our study is centred on the LV chamber only, we purposely ignore the distortions of the rest of the cardiac chambers (right ventricle, left atrium, and right atrium) in the dataset videos. For example, in Figure 5.3(e), both the left atrium and the right atrium are



(a) Blurred left wall of the LV. **(b)** Missing left wall of the LV.

(c) Blurred left wall of the LV. **(d)** Noisy LV chamber.

(e) Clear LV. **(f)** Clear LV.

Figure 5.3: Captured frames from 6 different echocardiography videos of the HMC-QU dataset, where each image from (a) to (f) corresponds to a distinct video.

partially cut from the view, however, this does not impact our study since it is solely based on the LV chamber. Hence, our final set of videos for segmentation consists of both clear and blurred video images of the LV chamber.

5.2 Left Ventricle Segmentation with 2D Convolutional Neural Networks

The 2D CNN performs a supervised classification by learning to map the input echocardiography video to its adequate segmentation mask. Thus, we manually created segmentation masks that cover the LV chamber from the A4C view and discard the remaining chambers. The manually created segmentation masks were assigned to the dataset video frames as labels, and fed to the 2D CNN to learn the best segmentation mask from any given echocardiography video. The videos were normalized prior to training the 2D CNN by means of the spatial sliding window technique due to differences in the dimensions of the frames.

5.2.1 Data Preprocessing

In this section, data labelling and segmentation with spatial windowing are discussed.

5.2.1.1 Data Labelling

The first step was preparing a labelled dataset, where each input is an echocardiography video frame, and each output is a corresponding segmentation mask. The segmentation masks were manually created and designed to cover the area of LV from the A4C in all the frames included in a given video. In each video, at least one cardiac cycle was performed, which means that we have at least one diastole (when the heart refills with blood) and one systole (when the heart pumps the blood) per video. The segmentation mask boundaries were determined such that they form a rectangle that encompasses the totality of the LV even on the frames where the heart is fully expanded, *i.e.* during diastole when the LV reaches its maximum volume. We assigned one segmentation mask for each echocardiography video. Consequently, the segmentation mask

assigned to a video was the same assigned to each of its frames. Thus, the final dataset that was used to train the 2D CNN contained the video frames as the input samples, and the segmentation masks as the labels or the output samples.

5.2.1.2 Segmentation with Spatial Windowing Process

The next step was to produce frames of the same spatial dimensions (frame size). Thereby, we opted for the sliding window technique to create spatial windows of fixed dimensions, and we applied the technique on both the input samples and the labels. The technique consists of extracting consecutive windows of equal dimensions with an overlap between two successive windows. Normally, the dimension of the window must be less than or equal to the original dimension of the frame from which it was extracted. Also, the overlap should be less than the dimension of the window. In Figure 5.4, we illustrate the sliding window technique, where it extracts two successive windows with an overlap equal to 50%. The red square in the figure represents a window and the green square represents its successive window that overlaps with the red square by 50%.

By applying the sliding window technique on the dataset, we created windows of dimension equal to 150×150 px, with a 50% spatial overlap equal to 75 px. The dimensions of the windows are always less than the original dimensions of the video frames, where the smallest frame dimension from the input samples is equal to 422×636 px. In this manner, we obtained a larger and uniform set of input samples totalling 108,127 windows.

The 2D CNN generates an estimation of a segmentation mask for an input window where each value within the segmentation mask is in the interval $[0, 1]$. We round these values to obtain a perfect mask with pixel values equal to either 0 or 1. Once the segmentation mask corresponding to each window is estimated, the complete segmentation mask of a video frame is reconstructed using the inverse sliding window technique. The technique is performed by adding the successive estimated segmentation masks of every window from a certain frame with an overlap equal to

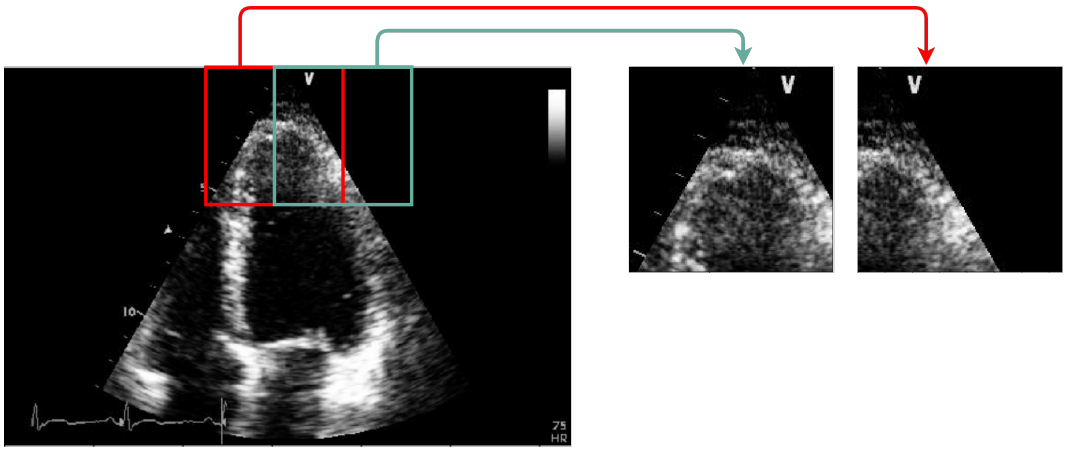
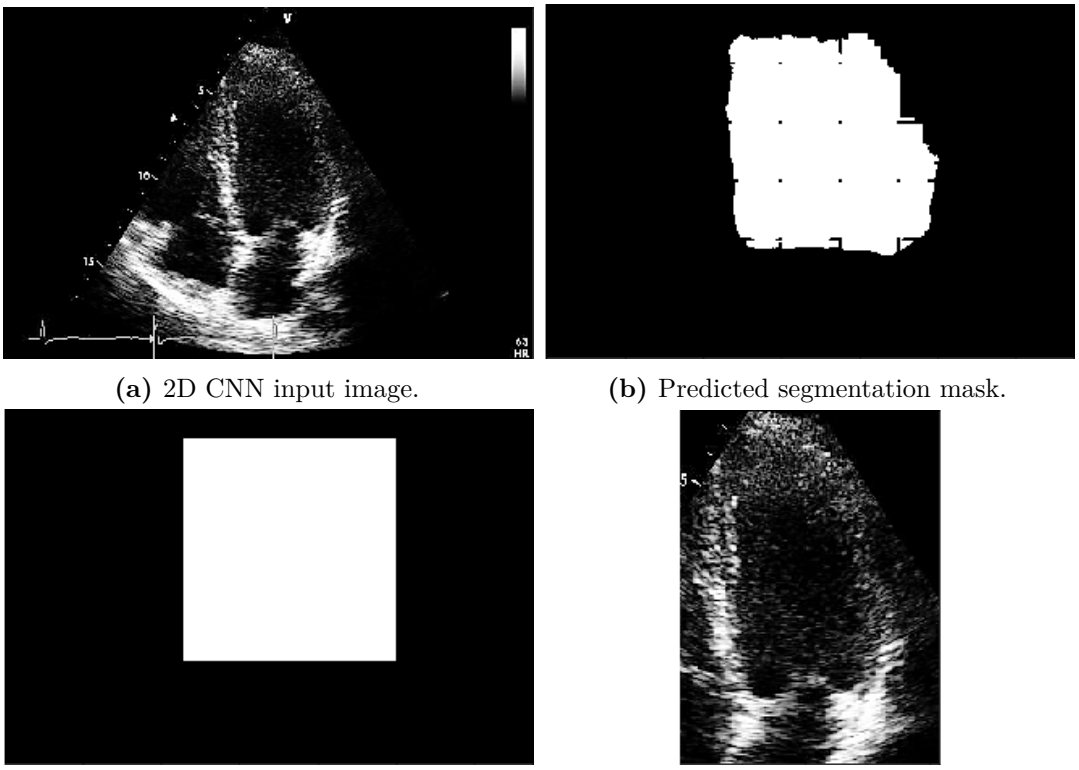


Figure 5.4: Spatial sliding window depicting the process of extracting two consecutive spatial windows from an frame with an overlap equal to 50% between the successive windows.

50% until we recover the entire frame. The reconstructed frame has the same dimension as the original frame cut from its video. With the same inverse sliding window technique, we recover all the segmented video frames and also all the segmentation masks, where each mask corresponds to a frame. Then, having all the segmentation masks predicted for each frame of a given video, we aggregate these masks employing statistical mode (*i.e.* the most represented value in each pixel is chosen) to form the segmentation mask corresponding to the totality of a video.

Figure 5.5 encapsulates the process of applying the predicted segmentation mask on a video frame. Figure 5.5(a) shows the original video frame, while Figure 5.5(b) shows its corresponding predicted mask recovered from the reverse sliding window technique, which appears as a set of points with undefined boundaries. Hence, to recover a rectangular-shaped segmentation mask we apply the minimum bounding box technique to enclose the estimated set of points into a rectangle and to produce a bounding box as shown in Figure 5.5(c). Then, each video frame is multiplied by its corresponding bounding box to produce a segmented frame as shown in Figure 5.5(d). The segmented frames belonging to the same video are then reassembled to produce a segmented video, where the order of appearance of each segmented frame is kept in its same order of appearance as in the original video. The segmented video has the same number of frames as the original video



(a) 2D CNN input image. **(b)** Predicted segmentation mask.

(c) Minimum bounding box. **(d)** Segmented video frame.

Figure 5.5: Illustration of the input and output images through the different phases of the segmentation process performed by the 2D CNN, such that (a) represents the 2D CNN input image, and (b) to (d) represent the different output images per phase: (a) represents a video frame captured from an echocardiography video, it also represents the 2D CNN input image. (b) represents the predicted segmentation mask corresponding to the input frame (a). It also represents the output image of the 2D CNN. (c) represents the minimum bounding box estimated from the predicted segmentation mask (b), and (d) represents the segmented video frame produced by multiplying the minimum bounding box (c) by the video frame (a).

prior to any preprocessing, however, with lower frame resolution.

5.2.2 2D Convolutional Neural Network Architecture

Our 2D CNN architecture follows the encoder-decoder design common to CNNs developed for semantic segmentation problems [106, 107, 108]. Figure 5.6 illustrates the detailed configuration of the 2D CNN consisting of three convolutional layers with ReLU as the activation function for each layer. Every convolutional layer is followed by a max-pooling layer to reduce the dimension of

the window. Then, the convolutional layers are followed by three transpose convolutional layers [109] with a stride equal to 2×2 in order to reacquire the initial input dimension. Each transpose layer uses a ReLU as its activation function. The last layer is a convolutional layer with a sigmoid activation function, which was selected to produce a predicted segmentation mask with pixel values equal to probabilities between the range of $[0, 1]$. The input and output dimensions are 150×150 px, which correspond to a segmentation mask adequate for the input window.

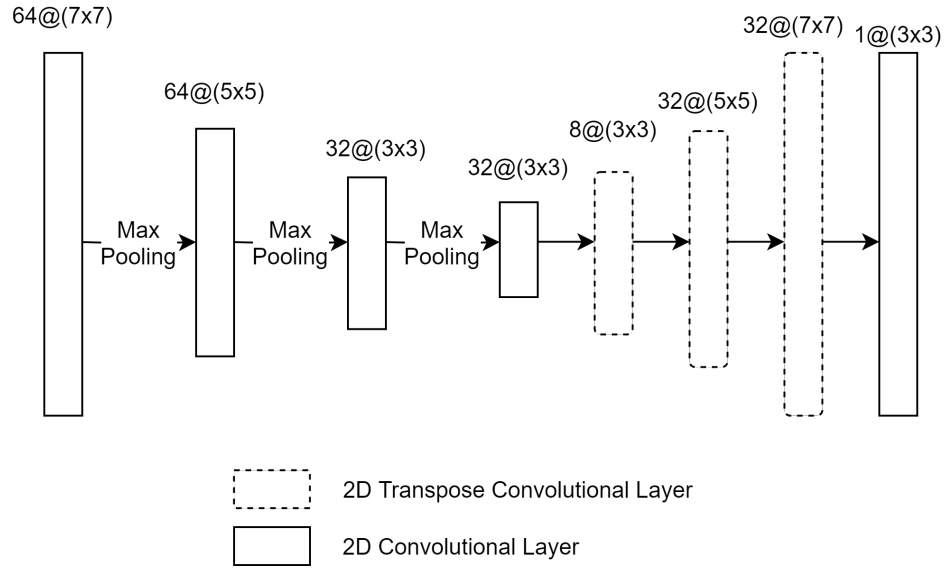


Figure 5.6: The architecture of the 2D CNN.

5.3 Myocardial Infarction Detection with 3D Convolutional Neural Networks

In this section, we give details of our proposed MI detection method using a 3D CNN over the segmented echocardiography videos obtained from a 2D CNN. However, these segmented videos have a different number of frames and different dimensions. In the following section, we give preprocessing details of segmented videos.

5.3.1 Data Preprocessing with Temporal Sliding Window

To solve the issue of differences in the spatial dimensions, all the video frames were scaled down to the smallest video size in the dataset. In our case, the smallest frame size from the segmented videos is equal to 236×183 px. Then, we applied the temporal sliding window technique to the resized videos in order to obtain a uniform number of frames. The technique consists of extracting a temporal window created from a consecutive number of frames from a given video and repeating the process by going over all the video frames with respect to an overlap between two successive temporal windows. In general, the overlap size is smaller than the temporal window size. The technique allows dividing the dataset videos into smaller temporal windows of a fixed number of frames. It also allowed us to increase the number of samples for the 3D CNN from 165 segmented videos to 2000 temporal windows. In our case, we applied the sliding window technique to extract temporal windows of size equal to 5, 7, and 9 frames per window, with an overlap equal to 4, 6, and, 8 frames, respectively (*i.e.* the sliding window moves forward by one window per step). By varying the size of the temporal windows, we created three different datasets that we used to train three different 3D CNN models.

We illustrate in Figure 5.7 the sliding window technique for a temporal window size equal to five. The red window represents a temporal window consisting of 5 successive frames. The green window is the successive temporal window of the red one that also contains 5 frames, such that the first 4 frames from the green window are the same as the last 4 frames from the red window. The labels attributed to these temporal windows are the same as the labels of the video from which these windows were extracted.

Table 5.1 shows the number of the temporal windows obtained from the dataset videos by varying the frame number of the temporal windows. For window sizes equal to 5, 7, and 9, we obtained 2841, 2511, and 2181 temporal windows, respectively, from the dataset of the segmented videos.

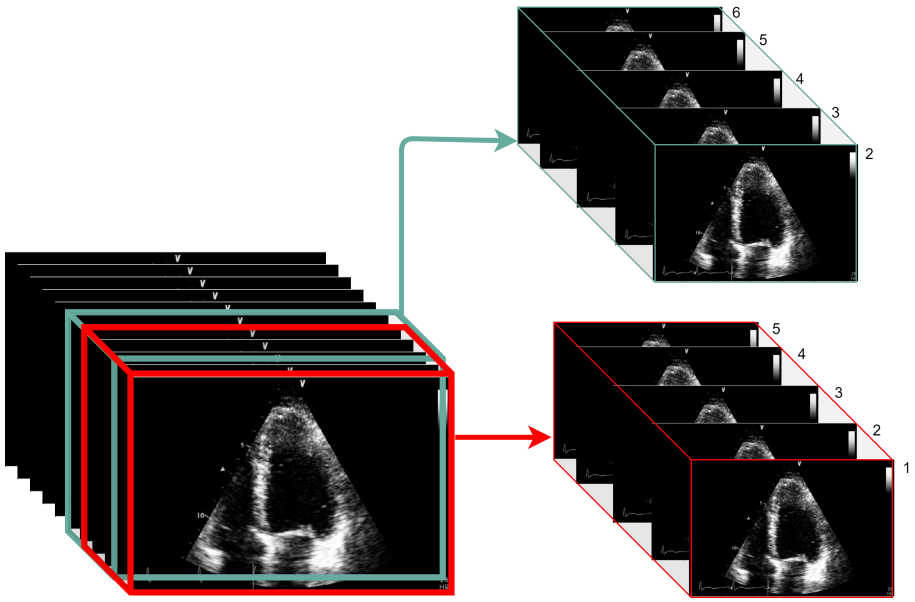


Figure 5.7: Temporal sliding window depicting the process of extracting two consecutive temporal windows of size 5 frames, with an overlap equal to 4 frames between two consecutive windows.

In another experiment, we applied a sliding window technique that extracts spatio-temporal windows from the segmented videos in an attempt to avoid rescaling the videos to the smallest dimension. The technique consists of combining the temporal and spatial sliding window techniques at the same time. Even though this process resulted in a larger dataset, the predicted accuracies were lower than those obtained by simply resizing the segmented videos and applying only temporal sliding window. Therefore, we concluded that the LV chamber should be fully preserved in a frame in the echocardiography video for the 3D CNN to capture all the details during the training process. Cutting the LV chamber from a segmented video by a spatial sliding window will degrade the information and will result in a poor model.

Table 5.1: Number of windows obtained by applying the temporal sliding window technique with different window sizes.

Size of the temporal window	Number of windows
5 frames	2841
7 frames	2511
9 frames	2181

5.3.2 3D Convolutional Neural Networks Architecture

In this section, we present the architectures of the 3D CNN models used to train the three datasets separately. For each dataset, we used the same model architecture: same number of layers, same number of neurons, and same activation functions. However, we changed the kernel size for each model to make it fit with the input dimension of the windows.

Figure 5.8 shows the architecture of the 3D CNN consisting of four 3D convolutional layers, four 2D max-pooling layers, and three dense layers. ReLU activation function was used for all the convolutional and densely-connected layers, except for the output layer in which sigmoid activation function was used. Table 5.2 gives the details of the characteristics of each 3D CNN model.

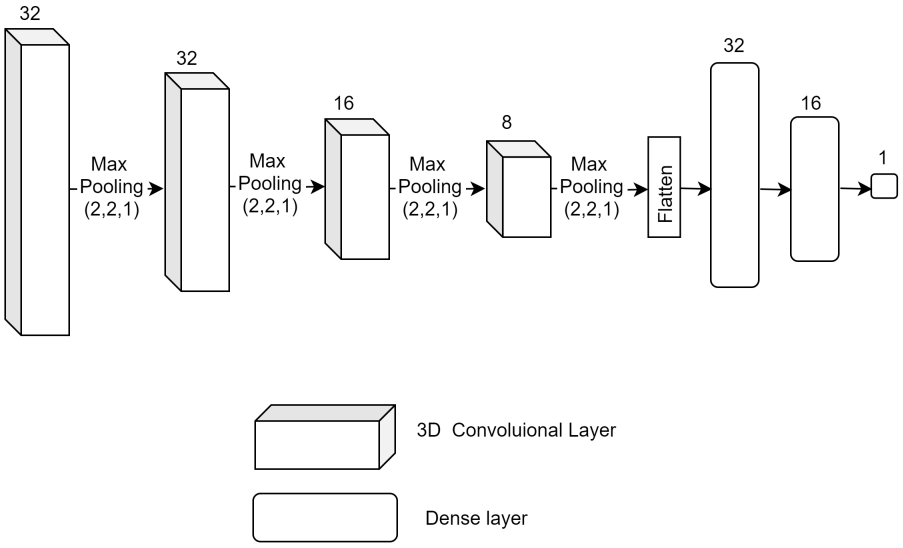


Figure 5.8: The generic architecture of the 3D CNN used to train all the datasets.

5.4 Experiments and Results

The experimental setup (training parameters and hyperparameters, and evaluation metrics) and the results corresponding to the 2D CNN for video segmentation and the 3D CNN for MI detection, respectively, are discussed in this section.

Table 5.2: 3D CNN characteristics per layer according to the size of the temporal window.

		Kernel size per window size		
Layer	# of neurons	5	7	9
Conv3D	32	$(3, 3, 3)$	$(3, 3, 3)$	$(3, 3, 3)$
MaxPooling	-	$(2, 2, 1)$	$(2, 2, 1)$	$(2, 2, 1)$
Conv3D	32	$(3, 3, 2)$	$(3, 3, 3)$	$(3, 3, 3)$
MaxPooling	-	$(2, 2, 1)$	$(2, 2, 1)$	$(2, 2, 1)$
Conv3D	16	$(3, 3, 2)$	$(3, 3, 2)$	$(3, 3, 3)$
MaxPooling	-	$(2, 2, 1)$	$(2, 2, 1)$	$(2, 2, 1)$
Conv3D	8	$(3, 3, 1)$	$(3, 3, 2)$	$(3, 3, 3)$
MaxPooling	-	$(2, 2, 1)$	$(2, 2, 1)$	$(2, 2, 1)$
Flatten	-			
Dense	32			
Dense	16			
Dense	1			

5.4.1 Left Ventricle Segmentation with 2D Convolutional Neural Networks

In this section, the training setup, evaluation metrics, and results related to the 2D CNN for video segmentation are discussed.

5.4.1.1 Training and Evaluation Metrics

In order to find the most efficient 2D CNN architecture for the task of video segmentation, we performed a grid search on the number of layers of the encoder part and the number of neurons per layer. The defined sets of values that were tested for the grid search are $\{3, 4, 5\}$ layers for the encoder and $\{8, 16, 32, 64\}$ neurons per layer. Then, we performed 5-fold CV for each architecture set and selected the one that achieved the best average CV results amongst all the models. Figure 5.6 shows the best performing 2D CNN architecture. 80% of the dataset was used for training and 20% for testing. Since 5-fold CV was used, the training set was further divided into 80%

for training and 20% for validation. The network that performs best on the validation set was retrained on the whole training set and finally evaluated on the test set. Prior to training, we normalized the data to values in the interval $[0, 1]$. We trained all networks for 100 epochs with a batch size equal to 256 resulting in 192,617 trainable parameters for the best 2D CNN. We used the sigmoid activation function for the output layer, and RMSProp optimizer as the optimization algorithm for the 2D CNN. To evaluate the performance of the model, we used a modified MSE as loss function, which is defined as

$$MSE(w, \hat{w}) = \frac{1}{w_h w_w} \sum_{i=0}^{w_h-1} \sum_{j=0}^{w_w-1} [w_{(i,j)} - \hat{w}_{(i,j)}]^2,$$

where w_h and w_w are the window's height and width, respectively, while w and $\hat{w}$ are the actual window and its corresponding prediction, respectively. All the relevant details regarding the training parameters of the 2D CNN are presented in Table 5.3.

Table 5.3: 2D CNN training parameters.

Parameters	Values
Input samples (windows)	$86,502$ (80%)
Input shape	$(150, 150)$
Output shape	$(150, 150)$
Trainable parameters	$192,617$
Loss	MSE
Optimizer	RMSProp
Epochs	100
Batch size	256

5.4.1.2 Results and Discussion

We evaluated the model using the test set by calculating its accuracy and it achieved 97.18%, which demonstrates that extracting the LV region manually can be replaced by an automatic segmentation method of high precision.

5.4.2 Myocardial Infarction Detection with 3D Convolutional Neural Networks

In this section, the training setup, evaluation metrics, and results related to the 3D CNN for MI detection are discussed.

5.4.2.1 Training and Evaluation Metrics

For the 3D CNN experiments, we performed a grid search on the number of layers and the number of neurons per layer for both the convolution layers and the classifier to find a good performing 3D CNN architecture for MI detection in echocardiography. The defined sets of values that were tested for the grid search were $\{3, 4, 5, 6\}$ layers for convolution, $\{2, 3, 4\}$ layers for the classifier, and $\{8, 16, 32, 64\}$ neurons per layer. Then, we performed 5-fold CV for each architecture set and selected the one that performed the best amongst all 3D CNN architecture sets. Prior to training, we normalized the data to values in the interval $[0, 1]$. The selected architecture was used for all the 3D CNN models. For training all three models, dataset was split into a training set and a test set consisting of 80% and 20% of the dataset, respectively. Then, the training set was further split into 80% for training and 20% for validation for the CV experiments. Since MI detection is a binary classification, we ensured that the dataset is balanced with respect to N and MI classes. Next, we applied 5-fold CV on each training and validation sets. We retrained the models with the best architectures on the entire training set and we evaluated them on the test set. However, our goal is to predict the class of a complete echocardiography video rather than the class of a temporal window. Thus, to calculate the evaluation metrics of the 3D model over the task of MI detection per video, we assigned a prediction class to each video as the result of the statistical mode calculated over all the predicted classes of the windows constituting that video. The evaluation metrics used to assess the performance of the models are accuracy, precision, recall (sensitivity), and F1 score.

To train the models, we used the same loss function and optimizer, however, the input shape varies between the models as shown in Table 5.4. We used binary CE as the loss function, and the RMSProp optimizer for all three models. Using the RMSProp optimizer is equivalent to using an adaptive learning rate, therefore, we did not include the learning rate in the set of tunable hyperparameters and we chose $1e^{-3}$ as the initial learning rate. For each fold, we trained the model for 100 epochs using a batch size equal to 8 samples. We calculated the evaluation metrics per video for each fold associated with each model. To implement the 3D CNN models, we used the Python programming language and its open-source neural network library Keras [105]. We conducted the experiments on a NVIDIA Tesla P100 GPU server with 12GB of GPU memory.

Table 5.4: Training parameters per window size corresponding to the 3D CNN models.

	Window size		
Parameters	5	7	9
Input samples (windows)	2273	2009	1745
Input shape	(236,183,5)	(236,183,7)	(236,183,9)
Trainable parameters	57,977	68,345	74,105
Learning rate	$1e^{-3}$	$1e^{-3}$	$1e^{-3}$
Loss	Binary CrossEntropy		
Optimizer	RMSProp		
Epochs per fold	100		
Batch size	8		

5.4.2.2 Results and Discussion

Table 5.5 shows the results of the evaluation metrics as produced by the fully trained 3D models using 5-fold CV and calculated with their corresponding test sets. Only the highest, lowest, and mean values are given. The best results of our models were: 90.9% accuracy, 97.2% F1 score, 100% precision, and 95% recall. However, the mean values for the evaluation metrics are slightly lower than the maximum values, and this is explained by the fact that the training sets contain distinct training samples, where some of the windows contain more noise and hence are of poorer quality

than other windows. Therefore, even though all the sets contain balanced and equal proportions of samples representing both classification classes, some of the folds may contain more noisy samples than the remaining folds, which influences the learning performance of the model at each fold. Hence, the model trained over the dataset of windows with the size equal to 5 frames, achieved the following mean values over the 5 folds of CV experiments: 84.6% accuracy, 87% F1 score, 89% precision, and 85.1% recall. Furthermore, we observe that the mean values of the evaluation metrics obtained from the dataset of windows with a size equal to 7 frames are slightly inferior to those attained from the dataset of windows with a size equal to 5. Likewise, the mean values of the evaluation metrics achieved over the dataset of windows equal to 9 frames, are less than those obtained over the windows of size equal to 7 frames. The mean values of the metrics obtained from the second dataset (window size 7) are 82.5% accuracy, 83.2% F1 score, 83.5% precision and 83.1% recall, whereas, the values obtained from the third (window size 9) are 81.3% accuracy, 83.1% F1 score, 84.6% precision and 82% recall. Thus, we conclude that enlarging the size of the temporal window reduces the performance of the 3D CNN. On an average, the end-to-end system lasts 13.12 ms to predict MI per video.

Table 5.5: 3D CNN models' evaluation metrics per window size.

Evaluation metrics		Window size		
		5	7	9
Accuracy %	Max	90.3	**90.9**	90.0
	Mean	**84.6**	82.5	81.3
	Min	**77.1**	72.9	68.4
F1 score %	Max	94.8	**97.2**	92.3
	Mean	**87.0**	83.2	83.1
	Min	68.7	**75.0**	68.4
Precision %	Max	94.7	**100**	94.7
	Mean	**89.0**	83.5	84.6
	Min	73.0	**75.0**	72.2
Recall %	Max	**95.0**	94.7	90.0
	Mean	**85.1**	83.1	82.0
	Min	65.0	**75.0**	65.0

Table 5.6 represents a performance comparison between the results of our 3D CNN models on MI detection in echocardiography and two recent state-of-the-art works [37, 74]. Both the results of our model and the state-of-the-art works were achieved on the HMC-QU dataset. [37] used an Active Polynomial-based model to detect MI in the HMC-QU dataset and reported the results from two separate tests. The first test was conducted on the HMC-QU dataset, while the second test was conducted over a partition of the dataset that included only videos with reasonable quality. [74] used an E-D CNN to achieve the MI detection in the HMC-QU dataset. To report the classification results, two types of experiments were conducted. In the first type, they extracted 6-segment features form the echocardiography and used them to train four conventional classification models: linear discriminant analysis (LDA), decision tree (DT), random forest (RF), and support vector machine (SVM). While in the second type of experiment, they extracted 5-segment features and used them to train the same four classifiers. In Table 5.6, we report the results of our three 3D CNN models, in addition to the results of Active Polynomials model tested over both reasonable quality videos and the entirety of the HMC-QU dataset, and the results of the four classifiers of the E-D CNN tested over both 6-segment features data and 5-segment features data.

The values of the performance metrics show the significant superiority of our 3D CNN MI prediction model in terms of accuracy, F1 score, precision and recall over state-of-the-art models on the HMC-QU dataset. Even compared to the results obtained by the Active Polynomials model tested only on reasonable-quality videos, and which has the highest performance metrics amongst the remaining state-of-the-art methods, our 3D CNN achieved remarkably higher results of accuracy, F1 score, precision and recall over the entirety of the HMC-QU dataset that contains both low-quality and reasonable-quality videos. The Active Polynomials model tested over reasonable quality videos only achieved 87.9%, 90.1%, 87.6%, and 92.8% of accuracy, F1 score, precision, and recall, respectively, while the 3D CNN achieved 90.9%, 97.2%, 100%, and 95% of accuracy, F1 score, precision and recall respectively, which shows the robustness and the efficiency of the 3D

CNN. Active Polynomials tested on the HMC-QU dataset achieved slightly better results compared to the remaining methods, followed by the SVM model with 5-segment features, however, their corresponding results are still lower than those achieved by the 3D CNN.

Table 5.6: Performance comparison of the results from the 3D CNN model presented in this work with two state-of-the-art methods: Active polynomials [37] and E-D CNN [74]. All MI detection results in this table were produced using the HMC-QU dataset.

	3D CNN			Active polynomials [37]	
	Window size			Video quality	
Evaluation metrics	5	7	9	Reasonable	All
Accuracy %	90.3	**90.9**	90.0	87.9	83.1
F1 score %	94.8	**97.2**	92.3	90.1	85.7
Precision %	94.7	**100**	94.7	87.6	82.6
Recall %	**95.0**	94.7	90.0	92.8	89.0

	E-D CNN [74]							
	6-segment features				5-segment features			
	LDA	DT	RF	SVM	LDA	DT	RF	SVM
Accuracy %	75.6	72.6	77.4	80.2	78.5	73.5	76.6	80.2
F1 score %	80.6	79.4	82.5	85.2	83.2	80.0	82.6	84.8
Precision %	83.8	80.4	85.9	85.5	86.6	81.7	84.9	86.8
Recall %	78.5	79.0	80.2	85.9	81.3	79.0	82.2	83.0

5.5 Chapter Summary

In this work, the proposed 2D CNN model for video segmentation achieved a high accuracy of 97.18% in segmenting LV from the A4C view, therefore, it could be a very reliable and valuable tool to assist cardiologists. Moreover, the proposed 3D CNN models showed that real-time prediction

of MI from a patient's echocardiography is realizable, reliable and efficient by achieving 90.9% accuracy, 100% precision, 95% recall, and 97.2% F1 score. Since the HMC-QU dataset included a mix of videos of poor-quality, low-resolution and corrupted by noise, we believe that these factors had a negative impact on the performance of the 3D CNN models on MI detection from the segmented LV views. Nevertheless, the results showed the robustness and efficiency of the proposed approach. Temporal window size has a significant bearing on the performance of the 3D CNN models as evidenced by our experiments. We related this variability in performance to the difference in the 3D CNN model's characteristics, which may have altered the ability of the model to extract relevant prediction features with the given neuron and layer parameters. The 3D CNN models were built with the objective of assigning the least possible number of layers and neurons while still being able to extract relevant spatio-temporal features from the spatio-temporal windows to achieve the most accurate MI detection.

Chapter 6

Conclusions and Future Work

In this thesis, the main objective was to explore and determine the accuracy and efficiency of the application of 3D CNN models on 3D signal data in both detection and regression problems. We considered the cases where the third dimension of the 3D signal is either spatial or temporal with the FHB detection and spikelet number estimation in multispectral PCs of wheat, and the MI detection in echocardiography, respectively.

With multispectral PCs, where the third dimension of the 3D signal is spatial, three major issues were investigated. The first issue was the ability of 3D CNNs to learn from PCs and generate accurate results, the second issue was the ability of CNNs to learn significant information from a relatively small number of samples of 3D data, and the third issue was the influence of multispectral information on the detection performance of 3D CNNs. In order to answer these questions, two state-of-the-art datasets were proposed: the UW-MRDC WHEAT-PLANT PC dataset, consisting of 216 multispectral PC of wheat plant, and the UW-MRDC WHEAT-HEAD PC dataset, consisting of 80 multispectral PC of wheat head. A novel and real-time GPU-enabled preprocessing method that runs 1065 times faster than its CPU counterpart was created to convert a batch of multispectral PCs into a batch of multispectral 3D images compatible with 3D CNNs.

Furthermore, we proposed a novel and efficient 3D CNN for FHB detection in wheat that achieved 100% accuracy on the UW-MRDC WHEAT-PLANT PC dataset. The detection model replaces the manual, prone-to-error, and subjective detection with reliable automated model. Moreover, the obtained results indicating the superiority of the RGB colour channel in the task of FHB detection in wheat than both the NIR channel and NIR+RGB combined. Finally, we proposed a unique and efficient 3D CNN model for the automated estimation of the total number of spikelets on a wheat head that achieved 1.13 MAE on the UW-MRDC WHEAT-HEAD PC dataset. The estimation model proposes an alternative to replace the costly manual and prone to human-error classical estimation with a reliable and efficient automated model.

With echocardiography, where the third dimension of the 3D signal is temporal, we proposed a novel, real-time, and fully automated approach consisting of 2D and 3D CNN models to early detect MI from the echocardiography of a patient. This approach replaces the time-consuming and manual preprocessing with an automated, fast, and reliable LV segmentation; and enhances the accuracy and efficacy of MI detection. This system is indiscriminative, in that it processes echocardiography videos of different sizes, frame rates, and resolutions, and it is lightweight in that it runs on parallel threads and does not require high memory or computational power.

Hence, our obtained results show the tremendous capability of 3D CNNs to solve complex problems and achieve results of high precision and accuracy. It also established the ability of 3D CNNs to process different types of 3D signals, such as videos and PCs, and to learn from different spectral information. In fact, based on the achieved results, it is possible to assume that CNNs are more than capable to propose solutions for real-world problems related to scientific fields such as agriculture and medical imaging. However, to what extent are these models reliable and efficient, and what are their limitations? At this point, it is plausible to suspect that the created detection model that achieved 100% accuracy on FHB detection can possibly succumb when tested on new field data, or that the estimation model that achieved 1.13 MAE can estimate erroneous values

when subject to new samples, since both models were not trained on sufficiently large datasets. Therefore, a larger dataset with samples representing more wheat varieties and FHB infection stages may improve the generalization capabilities of both the detection and estimation models. Moreover, although the 2D and 3D MI detection pipeline is lightweight and produces fast and highly accurate results, it is still not tested in a real-time embedded environment in the field. In addition, many samples of the HMC-QU dataset were highly corrupted by noise, which may have influenced the model's performance. Therefore, a cleaner and larger dataset may help to improve the accuracy of the models developed for MI detection.

For our future works, we suggest the following:

- Extend the UW-MRDC WHEAT-PLANT PC dataset with more WC and FHB samples in order to further evaluate the efficiency, robustness, and accuracy of the 3D CNN for the FHB detection on unseen samples.
- Extend the UW-MRDC WHEAT-HEAD PC dataset with more FHB diseased samples in order to develop a 3D CNN that estimates the total number of infected spikelets per wheat head, eventually to estimate the SI in wheat.
- Investigate the influence of NIR channel on FHB detection in wheat using a hyperspectral camera.
- Improve the efficiency and accuracy and reduce the MAE of the 3D CNN model that estimates the total number of spikelets in a wheat head.
- Improve the 3D CNN for MI detection model by enlarging the HMC-QU dataset with more echocardiography videos [110].
- Train the 3D CNN for MI detection model using spatio-temporal windows of higher temporal size, which could boost the 3D CNN accuracy by enabling it to learn more temporal features.

- Merge our end-to-end automated pipeline into an embedded system using TensorRT [111].

Bibliography

[1] J. F. Gilmore, "Artificial Intelligence In Image Processing," in *Digital Image Processing*, A. G. Tescher, Ed., vol. 0528, International Society for Optics and Photonics. SPIE, 1985, pp. 192 – 201. [Online]. Available: https://doi.org/10.1117/12.946419

[2] R. C. Gonzalez and R. E. Woods, *Digital image processing*. Upper Saddle River, N.J.: Prentice Hall, 2008. [Online]. Available: http://www.amazon.com/Digital-Image-Processing-3rd-Edition/dp/013168728X

[3] T. Phulpin, M. Derrien, and A. Brard, "A two-dimensional histogram procedure to analyze cloud cover from noaa satellite high-resolution imagery," *Journal of Applied Meteorology and Climatology*, vol. 22, no. 8, pp. 1332 – 1345, 1983.

[4] G. J. Jedlovec, S. L. Haines, and F. J. LaFontaine, "Spatial and temporal varying thresholds for cloud detection in goes imagery," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 46, no. 6, pp. 1705–1717, 2008.

[5] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[6] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, S. Y. Philip *et al.*, "Top 10 algorithms in data mining," *Knowledge and information systems*, vol. 14, no. 1, pp. 1–37, 2008.

[7] C. Latry, C. Panem, and P. Dejean, "Cloud detection with svm technique," pp. 448 – 451, 08 2007.

[8] B. Han, L. Kang, and H. Song, "A fast cloud detection approach by integration of image segmentation and support vector machine," pp. 1210–1215, 2006.

[9] F. Rosenblatt, *The perceptron: a probabilistic model for information storage and organization in the brain.*, 1958, vol. 65 6.

[10] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[11] A. Taravat, S. Proud, S. Peronaci, F. Del Frate, and N. Oppelt, "Multilayer perceptron neural networks model for meteosat second generation seviri daytime cloud masking," *Remote Sensing*, vol. 7, no. 2, pp. 1529–1539, 2015. [Online]. Available: https://www.mdpi.com/2072-4292/7/2/1529

[12] P. G. Brindha, M. Kavinraj, P. Manivasakam, and P. Prasanth, "Brain tumor detection from MRI images using deep learning techniques," *IOP Conference Series: Materials Science and Engineering*, vol. 1055, no. 1, p. 012115, feb 2021. [Online]. Available: https://doi.org/10.1088/1757-899x/1055/1/012115

[13] Y. J. Tan, K. S. Sim, and F. F. Ting, "Breast cancer detection using convolutional neural networks for mammogram imaging system," in *2017 International Conference on Robotics, Automation and Sciences (ICORAS)*, 2017, pp. 1–5.

[14] A. dos Santos Ferreira, D. Matte Freitas, G. Gonçalves da Silva, H. Pistori, and M. Theophilo Folhes, "Weed detection in soybean crops using convnets," *Computers and Electronics in Agriculture*, vol. 143, pp. 314–324, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169917301977

[15] A. V. Panchal, S. C. Patel, K. Bagyalakshmi, P. Kumar, I. R. Khan, and M. Soni, "Image-based plant diseases detection using deep learning," *Materials Today: Proceedings*, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214785321051403

[16] B. Ghimire, S. Sapkota, B. A. Bahri, A. D. Martinez-Espinoza, J. W. Buck, and M. Mergoum, "Fusarium head blight and rust diseases in soft red winter wheat in the southeast united states: State of the art, challenges and future perspective for breeding," *Frontiers in Plant Science*, vol. 11, 2020. [Online]. Available: https://www.frontiersin.org/article/10.3389/fpls.2020.01080

[17] S. Sakuma, G. Golan, Z. Guo, T. Ogawa, A. Tagiri, K. Sugimoto, N. Bernhardt, J. Brassac, M. Mascher, G. Hensel, S. Ohnishi, H. Jinno, Y. Yamashita, I. Ayalon, Z. Peleg, T. Schnurbusch, and T. Komatsuda, "Unleashing floret fertility in wheat through the mutation of a homeobox gene," *Proceedings of the National Academy of Sciences*, vol. 116, no. 11, pp. 5182–5187, 2019. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.1815465116

[18] D. Ferrigo, A. Raiola, and R. Causin, "Fusarium toxins in cereals: Occurrence, legislation, factors promoting the appearance and their management," *Molecules*, vol. 21, p. 627, 05 2016.

[19] M. K. Khan, A. Pandey, T. Athar, S. Choudhary, R. Deval, S. Gezgin, M. Hamurcu, A. Topal, E. Atmaca, P. A. Santos, M. R. Omay, H. Suslu, K. Gulcan, M. Inanc, M. S. Akkaya, A. Kahraman, and G. Thomas, "Fusarium head blight in wheat: contemporary status and molecular approaches," *3 Biotech*, vol. 10, pp. 1–17, 2020.

[20] *Morden Research and Development Centre*, 2022 (accessed March 3, 2022), available at "https://profils-profiles.science.gc.ca/en/research-centre/morden-research-and-development-centre".

[21] S. Wegulo, "Strategies for handling wheat grain affected by fusarium head blight," https://cropwatch.unl.edu/2019/handling-FHB-wheat, 2019, (accessed 20 April 2022).

[22] M. F. Lüscher, Thomas F., "Myocardial infarction: mechanisms, diagnosis, and complications," *European Heart Journal*, vol. 36, no. 16, pp. 947–949, 04 2015. [Online]. Available: https://doi.org/10.1093/eurheartj/ehv071

[23] J. Hubbard, "Complications associated with myocardial infarction," *Nursing times*, vol. 99, no. 15, p. 28—29, 2003. [Online]. Available: http://europepmc.org/abstract/MED/12733287

[24] L. J. Laslett, P. Alagona, B. A. Clark, J. P. Drozda, F. Saldivar, S. R. Wilson, C. Poe, and M. Hart, "The worldwide environment of cardiovascular disease: Prevalence, diagnosis, therapy, and policy issues," *Journal of the American College of Cardiology*, vol. 60, no. 25_Supplement, pp. S1–S49, 2012. [Online]. Available: https://www.jacc.org/doi/abs/10.1016/j.jacc.2012.11.002

[25] K. Thygesen, J. S. Alpert, A. S. Jaffe, B. R. Chaitman, J. J. Bax, D. A. Morrow, and H. D. White, "Fourth universal definition of myocardial infarction (2018)," vol. 72, no. 18. Journal of the American College of Cardiology, 2018, pp. 2231–2264. [Online]. Available: http://www.onlinejacc.org/content/72/18/2231

[26] I. Kosmidou, B. Redfors, H. P. Selker, H. Thiele, M. R. Patel, J. E. Udelson, and et al., "Infarct size, left ventricular function, and prognosis in women compared to men after primary percutaneous coronary intervention in st-segment elevation myocardial infarction," *European Heart Journal*, vol. 38, no. 21, pp. 1656–1663, 04 2017. [Online]. Available: https://doi.org/10.1093/eurheartj/ehx159

[27] A. F. Hernandez, E. J. Velazquez, S. D. Solomon, R. Kilaru, R. Diaz, C. M. O'Connor, G. Ertl, A. P. Maggioni, J.-L. Rouleau, W. van Gilst, M. A. Pfeffer, and R. M. Califf,

"Left Ventricular Assessment in Myocardial Infarction: The VALIANT Registry," *Archives of Internal Medicine*, vol. 165, no. 18, pp. 2162–2169, 10 2005. [Online]. Available: https://doi.org/10.1001/archinte.165.18.2162

[28] V. Palmieri, P. M. Okin, J. N. Bella, E. Gerdts, K. Wachtell, J. Gardin, V. Papademetriou, M. S. Nieminen, B. Dahlöf, and R. B. Devereux, "Echocardiographic wall motion abnormalities in hypertensive patients with electrocardiographic left ventricular hypertrophy," *Hypertension*, vol. 41, no. 1, pp. 75–82, 2003. [Online]. Available: https://www.ahajournals.org/doi/abs/10.1161/01.HYP.0000045081.54784.36

[29] R. Roberts and N. Kleiman, "Earlier diagnosis and treatment of acute myocardial infarction necessitates the need for a 'new diagnostic mind-set'." *Circulation*, vol. 89 2, pp. 872–81, 1994.

[30] F. Van de Werf, D. Ardissino, D. V. Cokkinos, K. A. A. Fox, D. Julian, and et. al, "Management of acute myocardial infarction in patients presenting with st-segment elevation," *European heart journal*, vol. 24, pp. 28–66, 01 2003.

[31] G. Dwivedi, K. L. Chan, M. G. Friedrich, and R. S. Beanlands, "Cardiovascular imaging: New directions in an evolving landscape," *Canadian Journal of Cardiology*, vol. 29, no. 3, pp. 257 – 259, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0828282X13000329

[32] J. Bax and V. delgado, "Advanced imaging in valvular heart disease," *Nature Reviews Cardiology*, vol. 14, 01 2017.

[33] P. S. Douglas, M. D. Cerqueira, D. S. Berman, K. Chinnaiyan, M. S. Cohen, J. B. Lundbye, and et al., "The future of cardiac imaging: Report of a think tank convened by the american college of cardiology," *JACC: Cardiovascular Imaging*, vol. 9, no. 10, pp. 1211 – 1223, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1936878X16302418

[34] R. K. Sevakula, W. M. Au-Yeung, J. P. Singh, E. K. Heist, E. M. Isselbacher, and A. A. Armoundas, "State-of-the-art machine learning techniques aiming to improve patient outcomes pertaining to the cardiovascular system," *Journal of the American Heart Association*, vol. 9, no. 4, p. e013924, 2020. [Online]. Available: https://www.ahajournals.org/doi/abs/10.1161/JAHA.119.013924

[35] J. Gottdiener, J. Bednarz, R. Devereux, J. Gardin, A. Klein, W. Manning, A. Morehead, D. Kitzman, J. Oh, M. Quinones, N. Schiller, J. Stein, and N. Weissman, "American society of echocardiography recommendations for use of echocardiography in clinical trials," *Journal of the American Society of Echocardiography", issn = "0894-7317*, vol. 17, no. 10, pp. 1086–1119, oct 2004.

[36] R. S. Horowitz, J. Morganroth, C. Parrotto, C. C. Chen, J. Soffer, and F. J. Pauletto, "Immediate diagnosis of acute myocardial infarction by two-dimensional echocardiography," *Circulation*, vol. 65, no. 2, pp. 323–329, 1982. [Online]. Available: https://www.ahajournals.org/doi/abs/10.1161/01.CIR.65.2.323

[37] S. Kiranyaz, A. Degerli, T. Hamid, R. Mazhar, R. E. Fadil Ahmed, R. Abouhasera, M. Zabihi, J. Malik, R. Hamila, and M. Gabbouj, "Left ventricular wall motion estimation by active polynomials for acute myocardial infarction detection," *IEEE Access*, vol. 8, pp. 210 301–210 317, 2020.

[38] M. Chen, L. Fang, Q. Zhuang, and H. Liu, "Deep learning assessment of myocardial infarction from mr image sequences," *IEEE Access*, vol. PP, pp. 1–1, 01 2019.

[39] S. Narula, K. Shameer, A. M. Salem Omar, J. T. Dudley, and P. P. Sengupta, "Machine-learning algorithms to automate morphological and functional assessments in 2d echocardiography," *Journal of the American College of Cardiology*, vol. 68, no. 21, pp. 2287–2295, 2016. [Online]. Available: https://www.onlinejacc.org/content/68/21/2287

[40] S. Kiranyaz, M. Zabihi, A. B. Rad, T. Ince, R. Hamila, and M. Gabbouj, "Real-time phonocardiogram anomaly detection by adaptive 1d convolutional neural networks," *Neurocomputing*, vol. 411, pp. 291–301, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231220309085

[41] A. Mishra, P. Dutta, and M. Ghosh, "A ga based approach for boundary detection of left ventricle with echocardiographic image sequences," *Image and Vision Computing*, vol. 21, pp. 967–976, 10 2003.

[42] Y. Zhu, M. Drangova, and N. J. Pelc, "Fourier tracking of myocardial motion using cine-pc data," *Magnetic Resonance in Medicine*, vol. 35, no. 4, pp. 471–480, 1996. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/mrm.1910350405

[43] C. Wu, W. Hsu, M. Islam, T. Poly, H. Yang, P. Nguyen, Y. Wang, and Y. Li, "An artificial intelligence approach to early predict non-st-elevation myocardial infarction patients with chest pain," *Computer Methods and Programs in Biomedicine*, vol. 173, pp. 109–117, may 2019.

[44] K. E. T. Upendra, G. A. C. Ranaweera, N. H. A. P. Samaradiwakara, A. Munasinghe, K. L. Jayaratne, and M. I. E. Wickramasinghe, "Artificial neural network application in classifying the left ventricular function of the human heart using echocardiography," in *2018 National Information Technology Conference (NITC)*, 2018, pp. 1–6.

[45] H. O. Klein, T. Tordjman, R. Ninio, P. Sareli, V. Oren, R. Lang, J. Gefen, C. Pauzner, E. D. Segni, D. David, and E. Kaplinsky, "The early recognition of right ventricular infarction: diagnostic accuracy of the electrocardiographic v4r lead." *Circulation*, vol. 67, no. 3, pp. 558–565, 1983. [Online]. Available: https://www.ahajournals.org/doi/abs/10.1161/01.CIR.67.3.558

[46] N. Gahungu, R. Trueick, S. Bhat, P. P. Sengupta, and G. Dwivedi, "Current challenges and recent updates in artificial intelligence and echocardiography," *Current Cardiovascular Imaging Reports*, vol. 13, no. 2, Feb 2020.

[47] J. E. O'Boyle, A. F. Parisi, M. Nieminen, R. A. Kloner, and S. Khuri, "Quantitative detection of regional left ventricular contraction abnormalities by 2-dimensional echocardiography," *The American Journal of Cardiology*, vol. 51, no. 10, pp. 1732 – 1738, 1983. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0002914983902205

[48] G. Wharton, R. Steeds, B. Rana, R. Wheeler, S. N, D. Oxborough, B. H, J. Allen, C. J, J. Sandoval, G. Lloyd, P. Kanagala, M. T, M. N, and R. Jones, "A minimum dataset for a standard transthoracic echocardiogram, from the british society of echocardiography education committee," *Echo Research and Practice*, vol. 2, 03 2015.

[49] M. Kurt, K. Shaikh, L. Peterson, K. Kurrelmeyer, G. Shah, S. Nagueh, R. Fromm, M. Quinones, and W. Zoghbi, "Impact of contrast echocardiography on evaluation of ventricular function and clinical management in a large prospective cohort," *Journal of the American College of Cardiology*, vol. 53, pp. 802–10, 03 2009.

[50] A. N. Neskovic, A. Hagendorff, P. Lancellotti, F. Guarracino, A. Varga, B. Cosyns, F. A. Flachskampf, B. A. Popescu, and et al., "Emergency echocardiography: the European Association of Cardiovascular Imaging recommendations," *European Heart Journal - Cardiovascular Imaging*, vol. 14, no. 1, pp. 1–11, 01 2013. [Online]. Available: https://doi.org/10.1093/ehjci/jes193

[51] Y. Nagata, Y. Kado, T. Onoue, K. Otani, A. Nakazono, Y. Otsuji, and M. Takeuchi, "Impact of image quality on reliability of the measurements of left ventricular systolic function and global longitudinal strain in 2d echocardiography," *Echo Research and Practice*, vol. 5, pp. 27 – 39, 2018.

[52] M. Qazi, G. Fung, S. Krishnan, R. Rosales, H. Steck, R. B. Rao, D. Poldermans, and D. Chandrasekaran, "Automated heart wall motion abnormality detection from ultrasound images using bayesian networks," p. 519–525, 2007.

[53] A. Degerli, M. Zabihi, S. Kiranyaz, T. Hamid, R. Mazhar, R. Hamila, , and M. Gabbouj, "Hmc-qu dataset," (accessed 25-04-2021).

[54] "Hamad medical corporation," https://www.hamad.qa/EN/Pages/default.aspx, accessed: 2020-09-24.

[55] O. Hamila, S. Ramanna, C. J. Henry, S. Kiranyaz, H. Ridha, R. Mazhar, and T. Hamid, "Fully automated 2d and 3d convolutional neural networks pipeline for video segmentation and myocardial infarction detection in echocardiography," *Multimedia Tools and Applications*, 2021, doi: 10.1007/s11042-021-11579-4.

[56] B. Lu, P. D. Dao, J. Liu, Y. He, and J. Shang, "Recent advances of hyperspectral imaging technology and applications in agriculture," *Remote Sensing*, vol. 12, no. 16, 2020. [Online]. Available: https://www.mdpi.com/2072-4292/12/16/2659

[57] M. Teke, H. S. Deveci, O. Haliloğlu, S. Z. Gürbüz, and U. Sakarya, "A short survey of hyperspectral remote sensing applications in agriculture," in *2013 6th International Conference on Recent Advances in Space Technologies (RAST)*, 2013, pp. 171–176.

[58] Y. Lu and S. Young, "A survey of public datasets for computer vision tasks in precision agriculture," *Computers and Electronics in Agriculture*, vol. 178, p. 105760, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169920312709

[59] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.aaa8415

[60] K. P. Ferentinos, "Deep learning models for plant disease detection and diagnosis," *Computers and Electronics in Agriculture*, vol. 145, pp. 311–318, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169917311742

[61] R. Singh, S. Srivastava, and R. Mishra, "Ai and iot based monitoring system for increasing the yield in crop production," in *2020 International Conference on Electrical and Electronics Engineering (ICE3)*, 2020, pp. 301–305.

[62] D. Schunck, F. Magistri, R. A. Rosu, A. Cornelißen, N. Chebrolu, S. Paulus, J. Léon, S. Behnke, C. Stachniss, H. Kuhlmann, and L. Klingbeil, "Pheno4d: A spatio-temporal dataset of maize and tomato plant point clouds for phenotyping and advanced plant analysis," *PLoS ONE*, vol. 16, 2021.

[63] H. Dutağacı, P. Rasti, G. Galopin, and D. Rousseau, "Rose-x: an annotated data set for evaluation of 3d plant organ segmentation methods," *Plant Methods*, vol. 16, 03 2020.

[64] R. Khanna, L. Schmid, A. Walter, J. Nieto, R. Siegwart, and F. Liebisch, "A spatio temporal spectral framework for plant stress phenotyping," *Plant Methods*, vol. 15, 02 2019.

[65] J. G. Barbedo, C. S. Tibola, and J. M. Fernandes, "Detecting fusarium head blight in wheat kernels using hyperspectral imaging," *Biosystems Engineering*, vol. 131, pp. 65–76, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1537511015000136

[66] R. Qiu, C. Yang, A. Moghimi, M. Zhang, B. Steffenson, and C. Hirsch, "Detection of fusarium head blight in wheat using a deep neural network and color imaging," *Remote Sensing*, vol. 11, p. 2658, 11 2019.

[67] D.-Y. Zhang, G. Chen, X. Yin, R.-J. Hu, C.-Y. Gu, Z.-G. Pan, X.-G. Zhou, and Y. Chen, "Integrating spectral and image data to detect fusarium head blight of wheat,"

Computers and Electronics in Agriculture, vol. 175, p. 105588, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169920306943

[68] D. Wu, Z. Cai, J. Han, and H. Qin, "Automatic kernel counting on maize ear using rgb images," *Plant Methods*, vol. 16, 06 2020.

[69] M. Wael, E.-S. Ibrahim, and A. S. Fahmy, "Detection of lv function abnormality using temporal patterns of normalized wall thickness," *Journal of Cardiovascular Magnetic Resonance*, vol. 17, p. 47, 02 2015.

[70] P. F. Moynihan, A. F. Parisi, and C. L. Feldman, "Quantitative detection of regional left ventricular contraction abnormalities by two-dimensional echocardiography. i. analysis of methods." *Circulation*, vol. 63, no. 4, pp. 752–760, 1981. [Online]. Available: https://www.ahajournals.org/doi/abs/10.1161/01.CIR.63.4.752

[71] A. Ulloa, L. Jing, C. W. Good, D. P. vanMaanen, S. Raghunath, J. D. Suever, and et al., "A deep neural network predicts survival after heart imaging better than cardiologists," *CoRR*, vol. abs/1811.10553, 2018. [Online]. Available: http://arxiv.org/abs/1811.10553

[72] K. Kusunose, T. Abe, A. Haga, D. Fukuda, H. Yamada, M. Harada, and M. Sata, "A deep learning approach for assessment of regional wall motion abnormality from echocardiographic images," *JACC: Cardiovascular Imaging*, vol. 13, 05 2019.

[73] E. M. Ohman, C. Casey, J. R. Bengtson, D. Pryor, W. Tormey, and J. H. Horgan, "Early detection of acute myocardial infarction: additional diagnostic information from serum concentrations of myoglobin in patients without st elevation," *British heart journal*, vol. 163(6), p. 335–338, 05 1990.

[74] A. Degerli, M. Zabihi, S. Kiranyaz, T. Hamid, R. Mazhar, R. Hamila, and M. Gabbouj, "Early detection of myocardial infarction in low-quality echocardiography," *IEEE Access*, vol. 9, pp. 34 442–34 453, 2021.

[75] G. G. Daniel, *Artificial Neural Network*. Dordrecht: Springer Netherlands, 2013, pp. 143–143. [Online]. Available: https://doi.org/10.1007/978-1-4020-8265-8_200980

[76] M. A. Islam, S. Jia, and N. D. B. Bruce, "How much position information do convolutional neural networks encode?" in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=rJeB36NKvB

[77] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016, http://www.deeplearningbook.org.

[78] S. Bansal, "3d convolutions : Understanding + use case," https://www.kaggle.com/code/shivamb/3d-convolutions-understanding-use-case/notebook, 2019, (accessed 13 April 2022).

[79] A. Ng, "Convolutions over volumes," https://www.youtube.com/watch?v=KTB_OFoAQcc&t=364s, 2019, (accessed 24 April 2022).

[80] S. Mannor, D. Peleg, and R. Rubinstein, "The cross entropy method for classification," pp. 561–568, 01 2005.

[81] Y. N. Dauphin, H. d. Vries, and Y. Bengio, "Equilibrated adaptive learning rates for non-convex optimization," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'15. Cambridge, MA, USA: MIT Press, 2015, p. 1504–1512.

[82] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015,*

Conference Track Proceedings, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[83] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*. Madison, WI, USA: Omnipress, 2010, p. 807–814.

[84] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, p. 1929–1958, jan 2014.

[85] C. Cortes, M. Mohri, and A. Rostamizadeh, "L2 regularization for learning kernels," ser. UAI '09. Arlington, Virginia, USA: AUAI Press, 2009, p. 109–116.

[86] R. Atienza, *Advanced Deep Learning with Keras: Apply Deep Learning Techniques, Autoencoders, GANs, Variational Autoencoders, Deep Reinforcement Learning, Policy Gradients, and More*. Packt Publishing, 2018.

[87] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[88] G. Huang, Z. Liu, L. V. D. Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, jul 2017, pp. 2261–2269. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPR.2017.243

[89] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[90] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-100 (canadian institute for advanced research)." [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

[91] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, Y. W. Teh and M. Titterington, Eds., vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. [Online]. Available: https://proceedings.mlr.press/v9/glorot10a.html

[92] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15. JMLR.org, 2015, p. 448–456.

[93] G. Turk, "The ply polygon file format," 1994 (accessed March 3, 2022), available at: "http://gamma.cs.unc.edu/POWERPLANT/papers/ply.pdf".

[94] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[95] K. T. Nilsen, S. Walkowiak, S. V. Kumar, Ó. I. Molina, H. S. Randhawa, R. Dhariwal, B. Byrns, C. J. Pozniak, and M. A. Henríquez, "Histology and rna sequencing provide insights into fusarium head blight resistance in aac tenacious," *Frontiers in Plant Science*, vol. 11, 2020.

[96] PlantEye F500 multispectral 3D scanner, (accessed March 3, 2022), available at: "https://phenospex.com/products/plant-phenotyping/planteye-f500-multispectral-3d-laser-scanner/".

[97] HortControl, (accessed March 3, 2022), available at: "https://phenospex.com/products/plant-phenotyping/science-hortcontrol-data-management-software/".

[98] CloudCompare, (accessed March 3, 2022), available at: "https://www.danielgm.net/cc/".

[99] H. Chen and J. Shen, "Denoising of point cloud data for computer-aided design, engineering, and manufacturing," *Engineering with Computers*, vol. 34, 07 2018.

[100] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun, "Deep learning for 3d point clouds: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 12, pp. 4338–4364, 2021.

[101] J. Otepka, S. Ghuffar, C. Waldhauser, R. Hochreiter, and N. Pfeifer, "Georeferenced point clouds: A survey of features and point cloud management," *ISPRS International Journal of Geo-Information*, vol. 2, pp. 1038–1065, 12 2013.

[102] NVIDIA, P. Vingelmann, and F. H. Fitzek, "Cuda, release: 10.2.89," 2020. [Online]. Available: https://developer.nvidia.com/cuda-toolkit

[103] N. Sharp *et al.*, *hapPLY API*, 2015 (accessed November 30, 2020), available at "https://github.com/nmwsharp/happly".

[104] P. Refaeilzadeh, L. Tang, and H. Liu, "Cross-validation," pp. 532–538, 2009. [Online]. Available: https://doi.org/10.1007/978-0-387-39940-9_565

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

[106] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[107] Y. Xing, L. Zhong, and X. Zhong, "An encoder-decoder network based fcn architecture for semantic segmentation," *Wireless Communications and Mobile Computing*, vol. 2020, pp. 1–9, 07 2020.

[108] V. Alhassan, C. J. Henry, S. Ramanna, and C. Storie, "A deep learning framework for land-use/land-cover mapping and analysis using multispectral satellite imagery," *Neural Computing and Applications*, vol. 32, 06 2020.

[109] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," pp. 18–23, 03 2016.

[110] A. Madani, J. Ong, A. Tibrewal, and M. Mofrad, "Deep echocardiography: data-efficient supervised and semi-supervised deep learning towards automated diagnosis of cardiac disease," *npj Digital Medicine*, vol. 1, 12 2018.

[111] "TensorRT cardiovascular diseases," https://developer.nvidia.com/tensorrt-getting-started, accessed: 2021-02-16.