LEVERAGING PYTORCH/XLA AND TPU FOR ENHANCED PLANT DISEASE DETECTION IN THE PLANTVILLAGE DATASET

by

Tomas Smetana

Thesis submitted to University of Plymouth
in partial fulfilment of the requirements for the degree of

MSc Artificial Intelligence

University of Plymouth
Faculty of Science & Engineering

September 2023

Abstract

Plant diseases pose a significant threat to global agriculture, affecting crop yields and food security. Accurate and timely detection of these diseases is crucial for effective crop management. This research aims to enhance plant disease detection by employing machine learning techniques, focusing specifically on a Convolutional Neural Network (CNN) model tailored for this task. Utilizing the PlantVillage dataset, which presents class imbalances, the study introduces class weights during model training to ensure fair representation of minority classes, thereby enhancing model performance across various plant diseases. The CNN architecture comprises multiple layers optimized for feature extraction and classification, resulting in a model with over 18 million trainable parameters. To efficiently manage this computational complexity, the study leverages Tensor Processing Units (TPUs), integrated through the torch_xla library. The model achieves impressive accuracy of 99.1% and demonstrates robustness in identifying both common and rare plant diseases. Its application promises significant contributions to the field of agricultural technology, providing a scalable and efficient tool for improved plant disease management.

Word Count

7746 words

GitHub Repository

https://github.com/Smety2001/PROJ518

Dataset

https://data.mendeley.com/datasets/tywbtsjrjv/1/files/d5652a28-c1d8-4b76-97f3-72fb80f94efc

Contents

List of Tables

List of Figures

Acknowledgements

I would like to express my deepest gratitude to the individuals who have been instrumental in my academic journey and the completion of this thesis.

First and foremost, I want to extend my heartfelt thanks to my supervisor Dena Bazazian. Your guidance, expertise, and unwavering support have been invaluable throughout every step of this project. Your insights and mentorship have not only enriched the quality of this thesis but have also played a pivotal role in shaping my growth as a researcher.

I am also deeply appreciative of all the dedicated lecturers who have imparted their knowledge and wisdom during my degree program. Your passion for teaching and commitment to excellence have had a profound influence on my academic development. Each of you has contributed in unique ways to my understanding of the subject matter and my overall intellectual growth.

Last but certainly not least, I extend my heartfelt gratitude to my family. Your unwavering encouragement, patience, and belief in my abilities have been my pillars of strength throughout this academic journey. Your sacrifices and understanding during the demanding periods of my studies have been deeply appreciated, and I could not have reached this milestone without your love and support.

1 Introduction

The agricultural sector plays a vital role in ensuring global food security and sustaining human life. However, this sector faces significant challenges, including the rapid spread of plant diseases. These diseases can cause substantial crop yield losses, leading to economic hardships for farmers and potential food shortages for communities. Timely and accurate disease diagnosis is crucial to mitigate these losses and protect food production.

In recent years, the integration of artificial intelligence (AI) and machine learning (ML) techniques has revolutionized various industries, including agriculture. One such advancement is the application of Convolutional Neural Networks (CNNs) for the identification and classification of plant diseases. CNNs, a subset of deep learning algorithms, have demonstrated remarkable capabilities in image analysis and pattern recognition tasks. Leveraging CNNs to detect and diagnose plant diseases can potentially transform agricultural practices, making them more efficient and sustainable.

1.1 Aim of the Study

The primary aim of this thesis is to investigate and demonstrate the effective utilization of PyTorch with XLA (Accelerated Linear Algebra) and TPU (Tensor Processing Unit) hardware for the purpose of enhancing plant disease detection within the PlantVillage dataset with a custom CNN model.

1.2 Objectives

To achieve the overarching aim of this study, the following specific objectives have been identified:

1.2.1 Explore the Dataset

Comprehensive understanding of the dataset forms the cornerstone of any successful machine learning endeavour, and this study is no exception. By delving deep into the intricacies of the dataset, a foundation for designing a robust Convolutional Neural Network (CNN) architecture, selecting suitable data augmentation strategies, and optimizing the training process is established. Ultimately, dataset exploration serves as the compass directing the pursuit of more accurate and efficient plant disease detection.

1.2.2 Dataset Preprocessing and Augmentation

Dataset preprocessing and augmentation are critical steps in ensuring the effectiveness of the deep learning model. Standardizing image dimensions and normalizing pixel values create a consistent and manageable input space for the model. Additionally, the careful partitioning of data into training, validation, and testing subsets ensures that the model's performance is rigorously evaluated.

Data augmentation strategies, including diverse transformations and class weight handling, empower the model to generalize better. It prepares the model to handle variations in real-world scenarios and mitigates the challenges posed by imbalanced class distributions.

1.2.3 Develop a Custom CNN Model

Developing a custom CNN model serves the purpose of designing a neural network architecture explicitly suited for the task of plant disease detection within the PlantVillage dataset. Unlike utilizing pre-existing models, a custom model can be fine-tuned and optimized to identify the unique features and patterns associated with various plant diseases, which in theory could lead to higher accuracies and better hardware utilization.

1.2.4 Optimize Training Pipeline for TPU

One of the key components of this study is the utilization of Tensor Processing Units (TPUs) in conjunction with PyTorch and XLA. TPUs are specialized hardware accelerators developed by Google for accelerating machine learning workloads. Their immense computational power makes them particularly well-suited for training deep neural networks at lightning speeds. To harness the full potential of TPUs for enhanced plant disease detection, an optimized training pipeline is crucial.

1.2.5 Evaluate Model Performance

The objective of evaluating model performance is a critical phase in this research, as it encompasses the rigorous assessment of the custom Convolutional Neural Network (CNN) model's effectiveness in plant disease detection within the PlantVillage dataset.

1.3 Potential Impact

The implications of this research are manifold and could profoundly influence several facets of agriculture and technology.

1.3.1 Enhancing Food Security:

The foremost impact lies in the potential to substantially boost food security. Early and accurate detection of plant diseases can lead to timely intervention, reducing crop losses and thereby contributing to a more reliable food supply chain. Given the rising global population and increasing incidences of extreme weather events affecting agriculture, such an advantage could be pivotal.

1.3.2 Reducing Pesticide Overuse:

Effective diagnosis enables targeted treatment, which in turn could significantly reduce the overuse of pesticides and other chemical treatments. This could lessen the environmental footprint of agriculture and mitigate risks to both human health and local ecosystems.

1.3.3 Democratizing Access to Expertise:

The development of an intuitive, AI-driven diagnostic tool can help democratize expertise by making high-level plant disease diagnostics accessible even to small-scale or less-experienced farmers. This could reduce disparities within the agricultural sector, fostering more equitable growth.

1.3.4 Economic Benefits:

The scalability of machine learning algorithms offers the potential for economic efficiencies. With a one-time investment in technology, farmers and agricultural organizations could see ongoing returns in the form of increased yields and reduced operating costs, creating a more sustainable business model for agriculture.

1.3.5 Technological Spillovers:

The advancements in machine learning and AI techniques engineered for this application may find utility in other sectors, creating opportunities for technological spillovers. For instance, improved CNN architectures could benefit medical imaging, facial recognition, or wildlife monitoring.

1.3.6 Policy Implications:

The findings could influence agricultural policies, prompting the integration of AI tools into national and global farming guidelines and regulations. This could accelerate the widespread adoption of sustainable practices, influencing agriculture at a systemic level.

By addressing these multifaceted impacts, the study aims not merely to develop a technical solution for plant disease diagnosis but to contribute to a larger dialogue on the future of sustainable agriculture and technology's role within it.

1.4 Ethical Considerations

The ethical landscape surrounding the application of machine learning in agriculture is complex and warrants careful consideration to ensure the responsible conduct of research and subsequent implementation of findings.

1.4.1 Data Privacy and Security:

While using the PlantVillage dataset, it is imperative to consider data privacy issues. Although the dataset is publicly available, and under the CC0 1.0 license, the ethical use and storage of data are critical to maintain research integrity.

1.4.2 Bias and Fairness:

Machine learning models are susceptible to biases present in the data. It is essential to scrutinize the PlantVillage dataset for any biases in representation of diseases, plant types, or other variables that might make the model less equitable or effective for a diverse range of users.

1.4.3 Environmental Impact:

The use of TPUs or any high-computational resource for training deep learning models raises questions about the carbon footprint of such activities. It is important to be transparent about the environmental costs and consider ways to mitigate this impact.

1.4.4 Accessibility:

As the research aims to democratize plant disease diagnosis, the developed tool or model should ideally be made accessible to farmers and agricultural organizations regardless of their technological proficiency or financial resources.

1.4.5 Unintended Consequences:

The development of an effective disease detection tool could potentially lead to unintended consequences, such as misuse of the technology or reliance on automated diagnosis at the expense of expert human judgment in complex cases.

1.4.6 Transparency and Accountability:

Clear documentation of the methods, architecture, and decision-making processes of the machine learning model is essential for ethical accountability. This transparency ensures that the model can be audited, evaluated, and improved upon in the future.

1.4.7 Economic Disruption:

While automation can bring about economic efficiencies, it is essential to consider the impact on labour markets, especially for those whose livelihoods depend on traditional agricultural roles that might be supplanted by automated systems.

By acknowledging and addressing these ethical considerations, the study aims to contribute to the responsible development and application of AI in the field of agriculture.

1.5 Contributions

The contributions of this research are manifold and hold the potential to significantly impact both the field of machine learning and agriculture. Below are some of the key contributions of this study:

1.5.1 Custom CNN Architecture:

This thesis introduces a novel Convolutional Neural Network (CNN) architecture specifically optimized for plant disease identification within the PlantVillage dataset. Unlike prior models, the architecture is tailored to extract features relevant to plant diseases, aiming for superior diagnostic performance.

1.5.2 TPU and PyTorch XLA Integration:

One of the standout contributions of this research is the innovative use of Tensor Processing Units (TPUs) in conjunction with PyTorch and XLA. This not only accelerates the model training but also offers a framework for efficient deep learning applications in agriculture.

1.5.3 Comprehensive Dataset Exploration:

Through in-depth data analysis and exploration, this study aims to provide a foundational understanding of the PlantVillage dataset's intricacies, setting a benchmark for future research.

1.5.4 Data Augmentation Strategies:

The research delves into unique data augmentation techniques tailored for plant disease images. This contribution aims to help the model generalize better, especially in real-world scenarios, and could inform data augmentation practices in similar problems.

1.5.5 Scalability and Efficiency:

By optimizing the CNN architecture for TPU deployment, this research introduces a scalable and efficient approach to plant disease identification that can be adapted for larger and more complex datasets.

By addressing these facets, the study enriches the growing body of literature on machine learning in agriculture, offering valuable insights and practical solutions for researchers and practitioners.

2 Review

2.1 The Problem: Prevalence of Plant Diseases and Its Economic Impact

Plant diseases are a considerable threat to global food security and economic stability. According to the Food and Agriculture Organization (FAO), up to 40% of global food crops are lost due to plant diseases and pests (Oerke, 2006). This problem is not merely confined to food security but extends to economic repercussions as well. The losses translate into hundreds of billions of dollars and can particularly hit hard in countries where agriculture is a primary industry. The increasing rate of new pathogen emergence and the limited effectiveness of chemical controls highlight the need for new solutions (Oerke, 2006).

2.2 Early Solutions: Visual Inspection and Traditional Methods

Before the advent of digital technology, plant disease diagnosis relied heavily on visual inspection by trained experts (Nutter et al., 1993). Farmers would also often employ preventive chemical treatments, although these were sometimes applied too liberally, leading to additional issues such as pesticide resistance and environmental damage. While expert visual inspection was considered the gold standard, it is impractical on a large scale due to time, cost, and the potential for human error.

2.3 Introduction of Machine Learning Algorithms

The field of plant disease identification underwent a paradigm shift with the introduction of machine learning techniques. Algorithms like Support Vector Machines (SVM) and k-Nearest Neighbours (k-NN) were explored as means to automate the diagnostic process (Sankaran et al., 2010). Despite being innovative, these traditional machine learning methods often required manual extraction of features like colour, texture, and shape, thereby necessitating expert input and limiting scalability (Hu et al., 2014).

2.4 Evolution to Deep Learning: CNNs Take the Stage

The use of Convolutional Neural Networks (CNNs) was a ground-breaking advancement in automating the identification of plant diseases. CNNs not only allowed for automated feature extraction but also enabled more complex and effective representations of data (LeCun et al., 2015). These networks consist of multiple layers, including convolutional layers, pooling layers, and fully connected layers, each contributing to the model's ability to understand and categorize complex patterns in images (Krizhevsky, Sutskever, and Hinton, 2012).

One significant area of investigation within CNNs is the selection of appropriate hyperparameters and network configuration. Various optimization algorithms like Stochastic Gradient Descent (SGD), RMSprop, and Adam each offer unique advantages and trade-offs in training neural networks, for example Adam is known for its adaptive learning rates and efficient performance (Kingma and Ba, 2014).Optimizers dictate how a model updates its weights in response to the calculated loss, thus influencing both training efficiency and final model accuracy.

Another critical component is the loss function, usually chosen based on the specific task at hand. For classification problems, which is often the case in plant disease identification, Cross-Entropy Loss is commonly employed (Goodfellow, Bengio, and Courville, 2016). This loss function calculates the difference between the predicted probabilities and the actual labels, helping the model to fine-tune its predictions.

Learning rates, which determine the step size during optimization, and epochs, the number of times the entire dataset is passed forward and backward through the CNN, are also essential hyperparameters to consider. Too high a learning rate can cause the model to converge too quickly and possibly overshoot the minimum cost, while too low a learning rate may cause the model to learn too slowly or not converge at all. The number of epochs is often set to a value that allows the model to converge while avoiding overfitting, guided by techniques such as early stopping (Prechelt, 1998).

In addition to these, techniques like data augmentation, dropout layers, and batch normalization have been employed to improve the generalizability and robustness of CNN models (Srivastava et al., 2014; Ioffe and Szegedy, 2015).

Recent studies have reported CNNs achieving accuracies up to 98% on specific plant disease datasets like PlantVillage, thereby making them the state-of-the-art models for this task (Mohanty et al., 2016). While CNNs have proven highly effective, they are not without challenges. Their heavy computational requirements and the need for large, labelled datasets for training are among the limitations that ongoing research aims to address.

2.5 Scalability and Computational Power: The Rise of TPUs

As deep learning models grew in complexity, so did their computational requirements. Traditional CPUs and even GPUs started to show their limitations, especially for large-scale models and datasets. This led to the development and introduction of Tensor Processing Units (TPUs), which are custom-built to accelerate tensor computations, the cornerstone of machine learning tasks (Jouppi et al., 2017).

TPUs have contributed to the feasibility of training more extensive and more complex models efficiently, thereby speeding up the research and development process.

2.6 Toward Real-world Application: Generalization and Independent Testing

While many deep learning models have achieved high performance on benchmark datasets like PlantVillage, the next critical step is ensuring they perform well on real-world, unseen data. A growing body of research is looking at how these models generalize across different types of data and environmental conditions, acknowledging that many existing models still need improvements in robustness (Barbedo, 2018).

2.7 Interpretability and Evaluation: A Look Ahead

As machine learning models are increasingly deployed in decision-making systems, the need for interpretability has never been greater. Understanding why a model makes a specific prediction is crucial for trust, especially in critical applications like healthcare or agriculture (Doshi-Velez and Kim, 2017). Current research is also focusing on refining evaluation metrics beyond mere accuracy, employing more comprehensive statistics like confusion matrices, precision, recall, and F1-scores for a better assessment of model performance.

3 Methodology/Procedure

The methodology employed in this research encompasses a systematic and comprehensive approach to enhance plant disease detection leveraging PyTorch/XLA and TPU computing. The following sections outline the key steps and procedures undertaken to achieve the research objectives:

3.1 Environment

The development and implementation of the plant disease detection model leveraging PyTorch/XLA and TPU acceleration within the PlantVillage dataset necessitated a carefully configured coding environment. This section outlines the hardware, software, and tools employed to facilitate the research:

3.1.1 Python and Basic Frameworks

The research and development were conducted using Python, with a version of 3.10.12. Python is a widely adopted programming language in the field of machine learning due to its extensive libraries, community support, and ease of use, which made it an easy choice.

In addition to Python, several basic frameworks and libraries played a crucial role in this project. These libraries include:

- os: The os module was utilized for various operating system-related tasks, such as file and directory manipulation.
- shutil: The shutil library was used for high-level file operations, making tasks like file copying and removal more efficient.

- pandas: The pandas library provided powerful data structures and data analysis tools, facilitating data manipulation and analysis.
- numpy: The numpy library was essential for efficient numerical computations, including array operations and mathematical functions.
- matplotlib.pyplot: With the help of matplotlib.pyplot, data visualization, including creating plots and charts, became straightforward.
- random: The random module allowed for the generation of random numbers and the implementation of various randomization techniques.
- time: The time module was used for measuring execution times, adding delays, and implementing timing-related functionality.
- seaborn: seaborn enhanced data visualization by providing a high-level interface for creating informative and attractive statistical graphics.
- itertools: The itertools library offered tools for working with iterators and creating efficient loop structures.
- pickle: For serializing and deserializing Python objects, the pickle module was employed, enabling the storage and retrieval of complex data structures.
- math: The math module provided a wide range of mathematical functions and constants for mathematical computations.
- PIL (Python Imaging Library): The PIL library, also known as Pillow, was used for image processing tasks, such as opening, manipulating, and saving various image file formats.

3.1.2 Deep Learning Frameworks

For implementing and training deep learning models, the following software libraries played a key role:

- The deep learning framework PyTorch, with a version of 2.0.1, was used for building and training neural networks. PyTorch is known for its flexibility, dynamic computation graph, and extensive community support.
- Torchvision, with a version of 0.15.2 was employed, as a companion library to PyTorch for computer vision tasks. Torchvision provides pre-trained models and data augmentation tools, simplifying the development of image-based deep learning models.
- Torch_XLA, specifically version 2.0.0, was integrated into the environment to leverage Google's TPU support. This allowed for the immense computational power of TPUs to be harnessed, further accelerating model training.

The versions of PyTorch, Torchvision, and Torch_XLA were carefully selected to ensure compatibility.

3.1.3 Google Colab

Google Colab, short for "Google Colaboratory," served as the primary computational environment for this thesis. It provided a versatile workspace that allowed for the development, training, and evaluation of deep learning models. Some of the notable features of Google Colab that contributed to the success of this research project include:

- **Cloud Computing:** Google Colab offers free access to GPUs (Graphics Processing Units) and TPUs (Tensor Processing Units) for running machine learning workloads, significantly accelerating model training.

- **Jupyter Notebook Integration:** Google Colab seamlessly integrates with Jupyter notebooks, enabling the creation of interactive and well-documented code.
- **Python Environment:** The default Python environment in Google Colab is highly customizable, making it possible to install and manage various Python packages, including those required for deep learning.

3.1.4 Data Integrity and Reproducibility

To ensure the integrity of the research findings and facilitate the reproducibility of the study, set seeds for random number generators were utilized in Python, numpy, PyTorch, random, and torch_xla. These seeds control the randomness in the data splitting, data augmentation, and model initialization processes, making it possible to replicate the experimental results.

3.2 Dataset

3.2.1 Dataset Description

The study utilizes the PlantVillage dataset, a collection of images containing various plant species and diseases. This dataset is employed for training and evaluating a custom Convolutional Neural Network (CNN) model for plant disease detection. Since the dataset is no longer accessible through the official Plant Village website, it was sourced from a scholarly paper that had utilized it for their study. The dataset was downloaded from https://data.mendeley.com/datasets/tywbtsjrjv/1/files/d5652a28-c1d8-4b76-97f3-72fb80f94efc and is available under a CC0 1.0 license, which permits its use for this thesis research.

3.2.2 Data Exploration

An initial exploration of the dataset was conducted to gain insights into its composition and characteristics. This included analysing the number of classes, unique image sizes, image types, and image file sizes for each class, plant, and disease within the dataset. The findings can be seen below.

Table 1: Key Dataset Information

No. of classes	Unique image sizes	Unique image types	Min size(kb)	Max size(kb)	Total size(mb)	Total images
38	"[256x256]"	"[JPG, jpg]"	3.4	30.0	811.2	54303

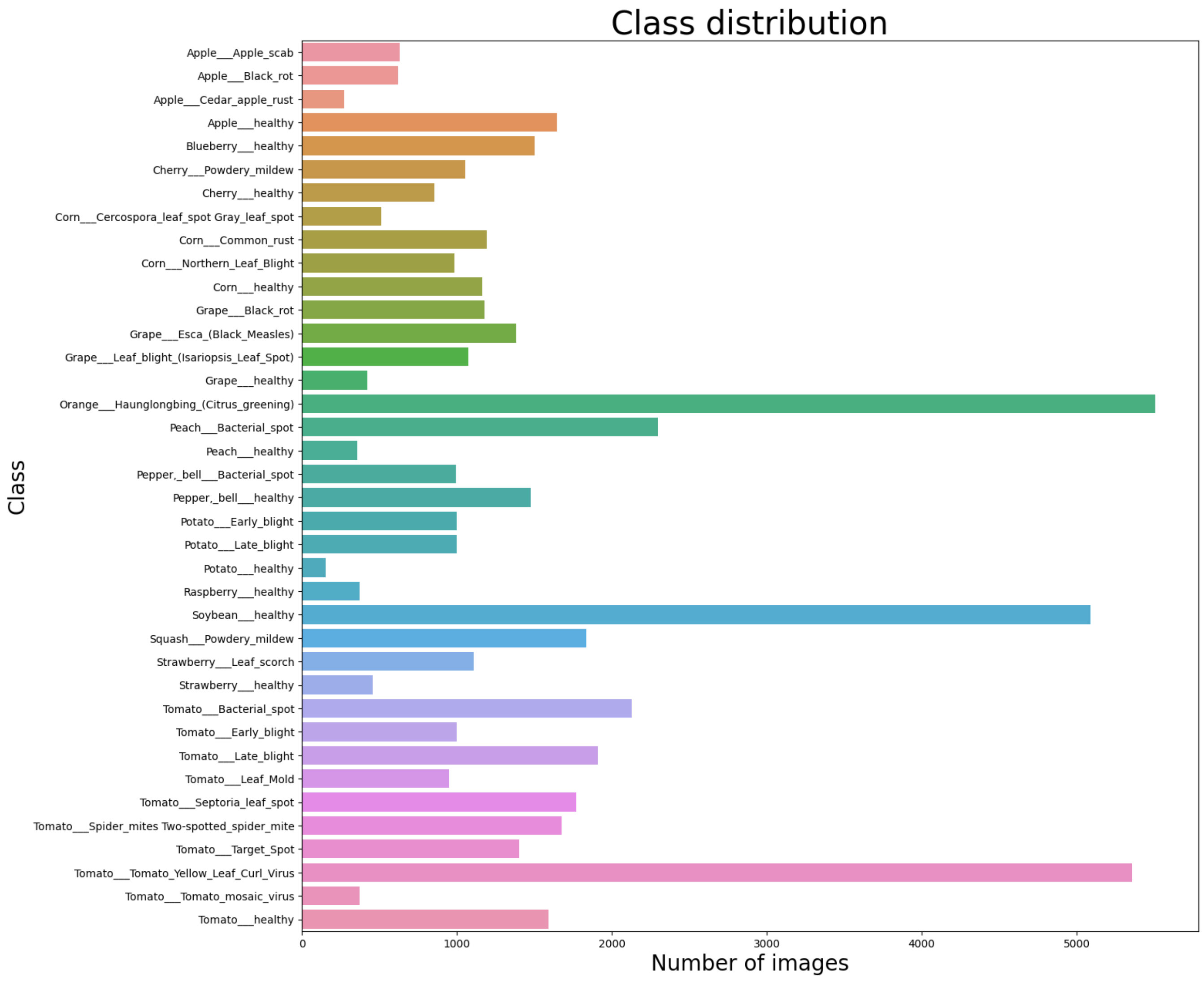


Figure 1: Class Distribution

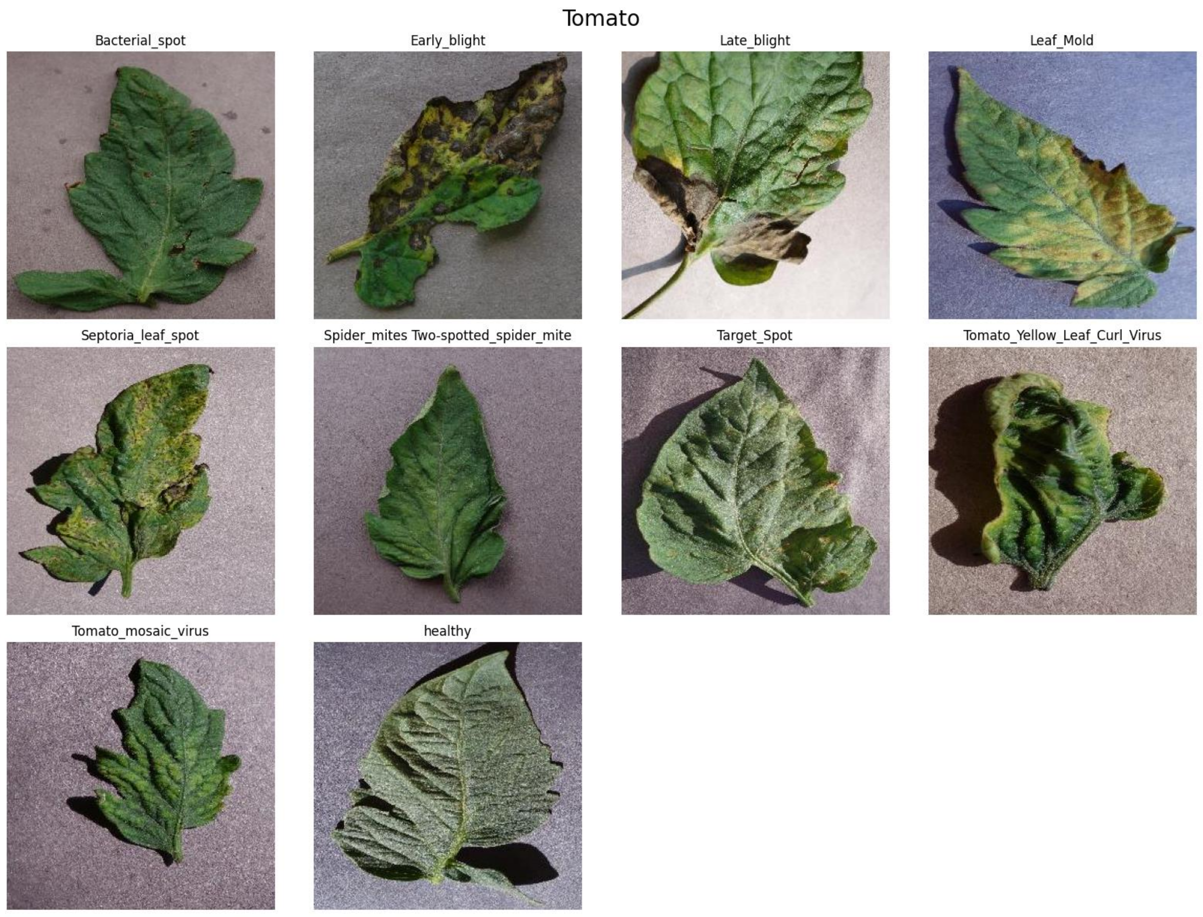


Figure 2: Sample of Images

The first notable finding was that all the images in the dataset are in JPEG (.jpg) format. This consistent format not only facilitates uniformity in image processing but also helps in streamlining data augmentation techniques later in the pipeline.

Secondly, the resolution of each image in the dataset was found to be 256x256 pixels. This uniformity in resolution simplifies the input requirements for the Convolutional Neural Network (CNN) and reduces the computational load for resizing images, thereby increasing efficiency in model training.

As for class distribution, it was observed to be imbalanced across the dataset. This is a critical finding, as an imbalanced dataset could introduce biases during model training, affecting the model's generalizability to diverse plant diseases. This necessitates the use of techniques such as class weighting, oversampling, or synthetic data generation to mitigate the imbalance, as will be discussed in subsequent sections.

Finally, a visual inspection of a random sample of images from different classes was conducted. This not only provided an understanding of the visual characteristics that differentiate various plant diseases but also highlighted the challenges that might be faced in classifying diseases that have visually similar features.

3.3 Pre-processing

3.3.1 Data Splitting

The process of dividing the dataset into training, validation, and testing subsets is a critical step in ensuring the integrity of model training, validation, and evaluation. In this research, the data splitting strategy was meticulously designed to address not only the need for diverse training data but also the imperative of maintaining a consistent class distribution across all subsets.

The dataset was divided into training, validation, and testing subsets following a specified ratio of 70% for training, 15% for validation, and 15% for testing. However, it is essential to emphasize that this division was not carried out randomly; rather, it was executed in a class-wide manner to ensure that each dataset subset contained the same class distribution.

The choice to partition the data by class was deliberately made to guarantee a balanced class distribution in both the validation and testing subsets. This approach ensures that the model is assessed on a well-distributed sample from each disease category, thereby offering a more comprehensive and equitable evaluation of its performance across the full range of diseases.

Table 2: Number of Images After Split

Dataset	No. of Images
Training dataset	37995
Validation dataset	8145
Testing dataset	8163

3.3.2 Class Imbalance and Class Weights

As the PlantVillage dataset suffers from class imbalance, some diseases are less frequently represented than others. Class imbalance can skew the model's performance, often leading it to favour the majority class and overlook rare diseases, a serious issue in the context of crop health.

To mitigate this, class weights during the training phase were employed. These weights disproportionately favour the minority classes to counterbalance their scarcity. Class weight for a single class is typically calculated as:

- Class Weight = Total Number of Samples / (Number of Classes * Number of Samples per Class)

These weights are integrated directly into the PyTorch loss computation. This weighted loss adjustment prompts the model to focus more on under-represented classes.

By implementing class weights, the model's sensitivity to rarer diseases is enhanced, thereby contributing to more balanced and accurate disease identification and ultimately, to more effective crop management.

3.4 Convolutional neural network design

In the pursuit of enhanced plant disease detection within the PlantVillage dataset, a well-structured Convolutional Neural Network (CNN) architecture was meticulously designed. The CNN model serves as the cornerstone of the research, responsible for capturing intricate spatial features and patterns within plant images for accurate disease classification.

3.4.1 Convolutional Layer

The CNN model commences with a series of convolutional layers, designed to extract relevant features from the input images. This initial layer, comprising of three blocks, each consisting of convolution, Rectified Linear Unit (ReLU) activation, batch normalization, and max-pooling, plays a pivotal role in feature extraction:

First Block

Convolution: The first convolutional layer employs a 3x3 kernel, with 3 input channels corresponding to colour images and 128 output channels. A padding of 1 maintains spatial dimensions.

ReLU Activation: The Rectified Linear Unit (ReLU) activation function introduces non-linearity, promoting feature representation.

Batch Normalization: Batch normalization enhances training stability by normalizing activations.

Max Pooling: Max-pooling reduces spatial dimensions, preserving essential information.

Second and Third Blocks

Similar Structure: Subsequent blocks retain a comparable structure, progressively increasing the number of output channels (128 to 256 to 512) for capturing increasingly complex features.

3.4.2 Fully Connected Layer

Following the convolutional layers, the model transitions to fully connected layers, responsible for mapping the high-level features extracted by the convolutional layers to the final output classes. This section comprises:

Dropout: A dropout layer with a dropout rate of 20% is applied to regularize the model. Dropout randomly deactivates a fraction of input units during each forward pass, preventing overfitting.

First Linear Layer: This fully connected layer takes as input the flattened feature maps obtained after the final max-pooling operation. It has 512 output features and introduces non-linearity with a ReLU activation function.

Second Linear Layer: The last fully connected layer maps the 512-dimensional feature vector to the number of output classes, which in this case is 38, representing various plant diseases.

3.4.3 Forward Pass

The forward pass of the CNN model is defined within the forward method. It begins by applying the convolutional layers to the input images, extracting meaningful features. Subsequently, the feature maps are flattened to create a one-dimensional feature vector. This vector is then passed through the fully connected layers to produce class predictions.

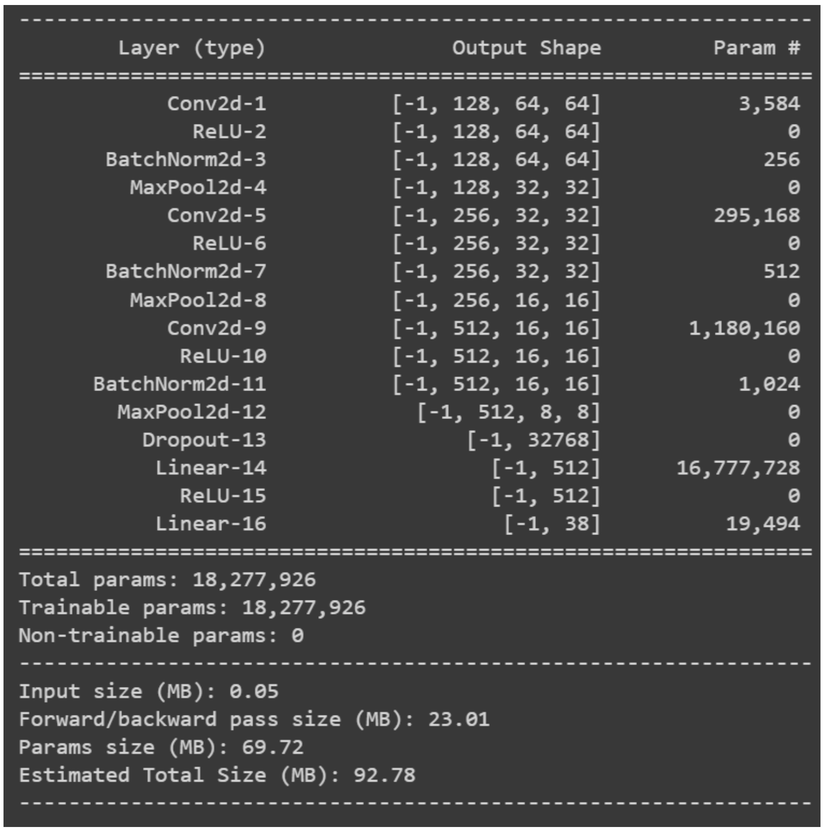
3.4.4 Model Complexity and Trainable Parameters

The Convolutional Neural Network (CNN) architecture employed in this research exhibits a notable degree of model complexity, characterized by an extensive number of trainable parameters. Specifically, the model comprises a total of 18,277,926 trainable parameters, each contributing to the model's ability to learn and represent intricate patterns within plant images.

The total number of trainable parameters in a deep learning model is a critical factor that influences its capacity to capture and generalize from the data. In the context of the CNN model designed for plant disease detection, the significance of these parameters can be summarized as follows:

- **Feature Representation:** Trainable parameters, primarily found within the convolutional layers and fully connected layers, serve as learnable filters and weights. These parameters enable the model to extract relevant features and representations from the input images. The increased number of parameters allows the model to capture a wide range of visual patterns and details related to plant diseases.

- **Model Flexibility:** A larger number of trainable parameters provides the model with greater flexibility and expressive power. This flexibility is crucial for accommodating the inherent complexity and variability present in the PlantVillage dataset, where different diseases may manifest in diverse ways.
- **Risk of Overfitting:** While an extensive parameter count can enhance a model's ability to fit the training data, it also poses a risk of overfitting, where the model may learn noise or idiosyncrasies present in the training data. To mitigate this risk, regularization techniques such as dropout have been incorporated into the model architecture.
- **Computational Demands:** It is important to acknowledge that a higher number of trainable parameters results in increased computational demands during both training and inference phases. However, the utilization of TPUs, helps mitigate computational challenges and accelerates model training.



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 128, 64, 64]           3,584
              ReLU-2         [-1, 128, 64, 64]               0
       BatchNorm2d-3         [-1, 128, 64, 64]             256
         MaxPool2d-4         [-1, 128, 32, 32]               0
            Conv2d-5         [-1, 256, 32, 32]         295,168
              ReLU-6         [-1, 256, 32, 32]               0
       BatchNorm2d-7         [-1, 256, 32, 32]             512
         MaxPool2d-8         [-1, 256, 16, 16]               0
            Conv2d-9         [-1, 512, 16, 16]       1,180,160
             ReLU-10         [-1, 512, 16, 16]               0
      BatchNorm2d-11         [-1, 512, 16, 16]           1,024
        MaxPool2d-12           [-1, 512, 8, 8]               0
          Dropout-13               [-1, 32768]               0
           Linear-14                 [-1, 512]      16,777,728
             ReLU-15                 [-1, 512]               0
           Linear-16                  [-1, 38]          19,494
================================================================
Total params: 18,277,926
Trainable params: 18,277,926
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 23.01
Params size (MB): 69.72
Estimated Total Size (MB): 92.78
----------------------------------------------------------------
```

Figure 3: CNN Architecture

3.5 Leveraging TPUs for Enhanced Training Efficiency

3.5.1 Introduction to TPUs and Integration with PyTorch

This research significantly benefits from the computational advantages offered by Tensor Processing Units (TPUs). Custom-designed to speed up matrix operations essential to machine learning algorithms, TPUs demonstrate superior parallelism and are therefore capable of accelerating large-scale deep learning models. To bridge TPUs with the PyTorch framework, the torch_xla library is employed, allowing for a seamless integration of TPUs within PyTorch-based machine learning workflows.

3.5.2 Configurations and Strategies for Efficient TPU Utilization

The computational workload is offloaded to TPUs using the spawn() and xla_device() function, thereby capitalizing on the hardware's parallel processing capabilities. To further enhance training efficiency, multiple strategies are employed:

- **Parallel Execution:** The torch_xla library facilitates the distribution of computations across multiple TPUs, which speeds up convergence times.
- **Data Parallelism:** Data is strategically divided among TPUs, allowing for parallel processing, and thereby increasing overall training speed.

These strategies are incorporated into the research code, optimizing data loading and training procedures.

3.5.3 Monitoring and Advantages of TPU-Based Training

Performance metrics like loss, accuracy, and learning rate are actively monitored to fine-tune the training process. The utilization of TPUs introduces several benefits:

- **Speed:** Drastic reductions in training time enable faster experimental iterations.
- **Energy Efficiency:** TPUs are more energy-efficient compared to other computing resources, thus minimizing computational costs.
- **Scalability:** The architecture of TPUs allows for incremental scaling, making them ideal for various workloads, from small-scale research to large-scale production deployments.

3.6 Optimizer, loss function and hyperparameters

The effectiveness of training a Convolutional Neural Network (CNN) model for plant disease detection significantly depends on the careful selection of optimization techniques, loss functions, and learning rates. This section delves into the choices made in these critical aspects of the training pipeline.

3.6.1 Optimizer: Adam

The optimizer plays a pivotal role in adjusting the model's parameters during training to minimize the loss function. In this research, the Adam optimizer is employed. Adam stands for "Adaptive Moment Estimation" and is known for its efficiency in optimizing deep neural networks. Key characteristics of the Adam optimizer include:

- **Adaptivity:** Adam adapts the learning rates for each parameter individually, which helps in training when dealing with varying gradients across different layers of the network.
- **Momentum:** The optimizer employs a momentum term, similar to stochastic gradient descent with momentum, to accelerate convergence and escape local minima.

- **Efficiency:** Adam combines the benefits of both AdaGrad and RMSProp optimizers, offering fast convergence and efficient memory usage.

The choice of the Adam optimizer aligns with the goal of efficiently training the CNN model for plant disease detection.

3.6.2 Loss Function: Cross-Entropy Loss

Selecting an appropriate loss function is crucial, as it quantifies the dissimilarity between predicted and actual target values. In this research, the Cross-Entropy Loss (also known as Log Loss) is utilized. The Cross-Entropy Loss is well-suited for multi-class classification tasks and offers several advantages:

- **Class Imbalance Handling:** The Cross-Entropy Loss can accommodate class imbalance through the use of class weights. In this research, class weights are incorporated to address the imbalanced nature of the PlantVillage dataset, giving each class proportional importance during training.
- **Softmax Activation Compatibility:** Cross-Entropy Loss complements the softmax activation function, which is often used in the final layer of classification models. It encourages the model to produce probabilities that sum to one, facilitating class probability estimation.

3.6.3 Learning Rate and Scheduler

The learning rate is a hyperparameter that controls the step size at which the optimizer updates the model's parameters. Learning rate scheduling is employed to adjust the learning rate during training, enhancing convergence and model generalization. In this research, the following learning rate-related choices were made:

- **Initial Learning Rate:** The initial learning rate is set to 0.001, indicating the magnitude of updates applied to the model's parameters during each training iteration.
- **Weight Decay:** A weight decay of 0.001 is applied to the optimizer, serving as a form of L2 regularization. It helps prevent overfitting by penalizing large parameter values.
- **Learning Rate Scheduler:** The Cosine Annealing Learning Rate Scheduler is employed. This scheduler cyclically varies the learning rate according to a cosine function. It begins with the initial learning rate and gradually decreases it, allowing the model to converge smoothly.
- **Minimum Learning Rate (eta_min):** The learning rate is annealed to a minimum value of 1e-6 to fine-tune the model's performance in the later stages of training.

These choices in learning rate and scheduling strategies aim to strike a balance between rapid convergence and avoiding overfitting, ultimately contributing to the robustness of the plant disease detection model.

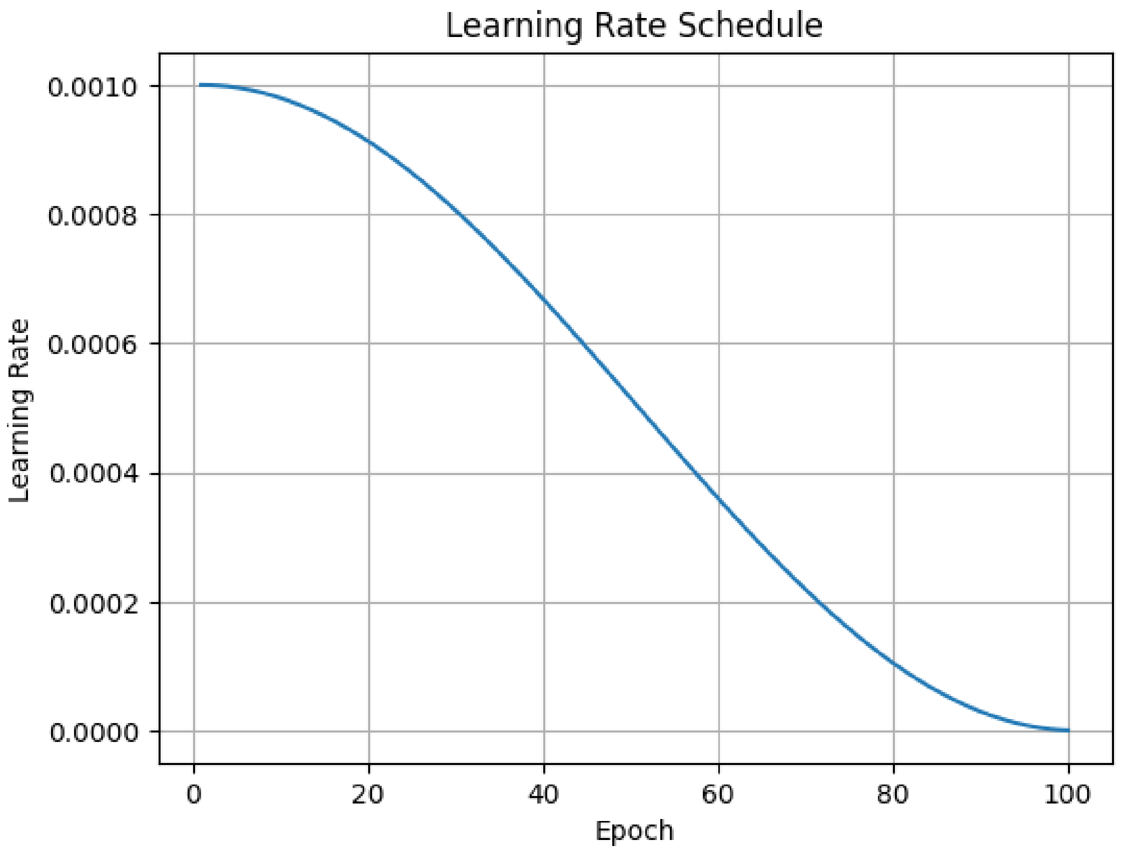


Figure 4: Learning Rate Schedule

3.6.4 Batch Size and Image Size

The batch size is another important hyperparameter in training deep learning models. It specifies the number of training examples used in a single iteration to update the model's weights. In this research, a batch size of 128 is used for several reasons:

- **Computational Efficiency:** A larger batch size can make better use of hardware capabilities, particularly when using accelerators like TPUs or GPUs.

- **Stable Gradients:** Larger batch sizes often produce gradients that better approximate the "true" gradient over the entire dataset, providing a more stable and smoother optimization process.
- **Memory Limitations:** While larger batch sizes can stabilize the training, they also consume more memory. The choice of 128 strikes a balance between computational efficiency and hardware limitations.

The original images in the PlantVillage dataset are of size 256x256 pixels. However, these images are resized to 64x64 pixels during preprocessing for several reasons:

- **Reduced Computational Load:** Smaller image sizes significantly reduce the number of computations and memory usage, making the training faster and more memory efficient.
- **Generalization:** Downscaling images can act as a form of data augmentation, helping the model generalize better to different scales of input.
- **Hardware Constraints:** TPUs and GPUs have memory limitations, and using smaller images allows for larger batch sizes or more complex models without exceeding these constraints.

The choice to resize the images to 64x64 pixels is a trade-off. While we lose some level of detail that might be critical for certain applications, the gains in computational efficiency and generalization are deemed more beneficial for the scope of this research.

3.7 Training Loop

The training loop is the heartbeat of the machine learning model, dictating how data is processed, computations are performed, and parameters are updated. In a TPU environment, special considerations are in place to ensure that the system operates efficiently and leverages the full computational power of the hardware.

3.7.1 Data Preparation and Augmentation

The training loop begins with data preparation and augmentation. Before training commences, the input data is loaded and transformed. Data augmentation techniques, such as random horizontal flips, colour jittering, and random rotations, are applied to the training dataset. These augmentations help diversify the data, preventing the model from overfitting to the training set. The transformations used can be seen in the table below:

Table 3: Data Augmentation

Technique	**Values**
Random horizontal flip	default

Normalization	mean = [0.485,0.456,0.406]
	std = [0.229,0.224,0.225]

Colour jitter	brightness = 0.2
	contrast = 0.2
	saturation = 0.2
	hue = 0.1

Random rotation	degrees = 30

3.7.2 Model Wrapping and Multiprocessing

The model is initialized once globally using xmp.MpModelWrapper(Model()). This is crucial when using the xmp.spawn(..., start_method='fork') API, as it minimizes the consumption of host memory. Unlike scenarios where each multiprocessing process creates its own model, thus replicating the initial memory usage, the model is created once and subsequently moved into each device inside the spawn() target function.

3.7.3 Serial Executor for Data Loading

The training and validation datasets are loaded using a serial executor (serialExe.run(loadDatasets)). This avoids the redundancy of each TPU core downloading the same dataset, leading to more efficient memory usage and faster data loading times.

3.7.4 Tensor Data Type (BF16)

TPUs inherently operate using the bfloat16 (BF16) datatype, which offers a good balance between precision and range, while being memory efficient. Though the code explicitly sets the default tensor type to torch.FloatTensor, TPUs internally optimize these operations using BF16.

3.7.5 Distributed Sampling

The training loop uses torch.utils.data.distributed.DistributedSampler to distribute data among the available TPU cores. This ensures that each core processes only a subset of the dataset, thus enabling true parallel processing.

3.7.6 Training and Validation Functions

For each epoch, both training (trainFun()) and validation (validFun()) functions are invoked. These functions encapsulate the essence of the learning process, from forward and backward passes to metric calculation. Notably, the training function employs mesh reduction via xm.mesh_reduce() for aggregated loss and accuracy metrics across TPU cores. This ensures a more precise evaluation of the model’s performance.

Forward and Backward Pass: The model's forward and backward passes are carried out using standard PyTorch functionalities. Losses are computed using Cross-Entropy Loss, and backpropagation is done to update the model parameters.

Metric Calculation: After each forward pass, predictions are made, and both loss and accuracy metrics are calculated. These are then aggregated across all TPU cores to provide a holistic view of the model’s performance.

Learning Rate Scheduling: A Cosine Annealing learning rate scheduler is used to adjust the learning rate across epochs dynamically.

3.7.7 Saving Model and Metrics

At the end of all epochs, the model's state dictionary and various performance metrics are saved. This is achieved using xm.save(), which saves data in a manner that is compatible with TPU multiprocessing.

By employing these techniques and architectural decisions, the training loop efficiently leverages the computational prowess of TPUs, while also ensuring optimal memory usage and performance monitoring.

4 Results

The following sections present a comprehensive evaluation of the plant disease classification model across various facets, including its performance on training and validation datasets, computational efficiency, and real-world applicability. Initial metrics reveal a high degree of accuracy in both training and validation phases, affirming the model's robustness and its potential for practical applications. Furthermore, an in-depth analysis using an independent testing dataset and class-wise performance metrics offers a nuanced view of the model's capabilities. Finally, considerations around training time and scalability emphasize the model's efficiency and readiness for larger-scale deployments.

4.1 Training and Validation data

4.1.1 Accuracy

The model's accuracy on the training dataset reached an impressive 99.80%. This indicates the capability of the model to correctly classify a substantial proportion of the images it was trained on. Moreover, the validation accuracy, which provides a better insight into the model's ability to generalize on unseen data, was recorded to be 99.15%. The high validation accuracy suggests that the model has been effectively trained without significant overfitting.

4.1.2 Loss

To further understand the model's convergence, the loss values during training and validation were analysed. The last five recorded loss values for the training dataset were relatively low, ranging from approximately 0.0074 to 0.0077. These values are indicative of the model's effectiveness in minimizing the discrepancy between its predictions and the actual labels of the training dataset. The exact values are as follows: 0.00757, 0.00741, 0.00769, 0.00762, 0.00760

In comparison, the loss values for the validation dataset were slightly higher but still low, lying in the vicinity of 0.0222. The specific recorded values are: 0.0222, 0.0224, 0.0222, 0.0222, 0.0221

These values for the validation loss support the observed high validation accuracy and further confirm the model's robust performance.

4.1.3 Graphical Analysis

For a more granular insight into the model's performance across all epochs, graphs depicting the accuracy and loss for each epoch have been constructed. The graphs visually represent the trend and fluctuations in the model's performance, thus providing an intuitive understanding of its learning trajectory. The juxtaposition of training and validation metrics in these plots accentuates their correlation and differences, offering an elaborate narrative on the model's learning dynamics.

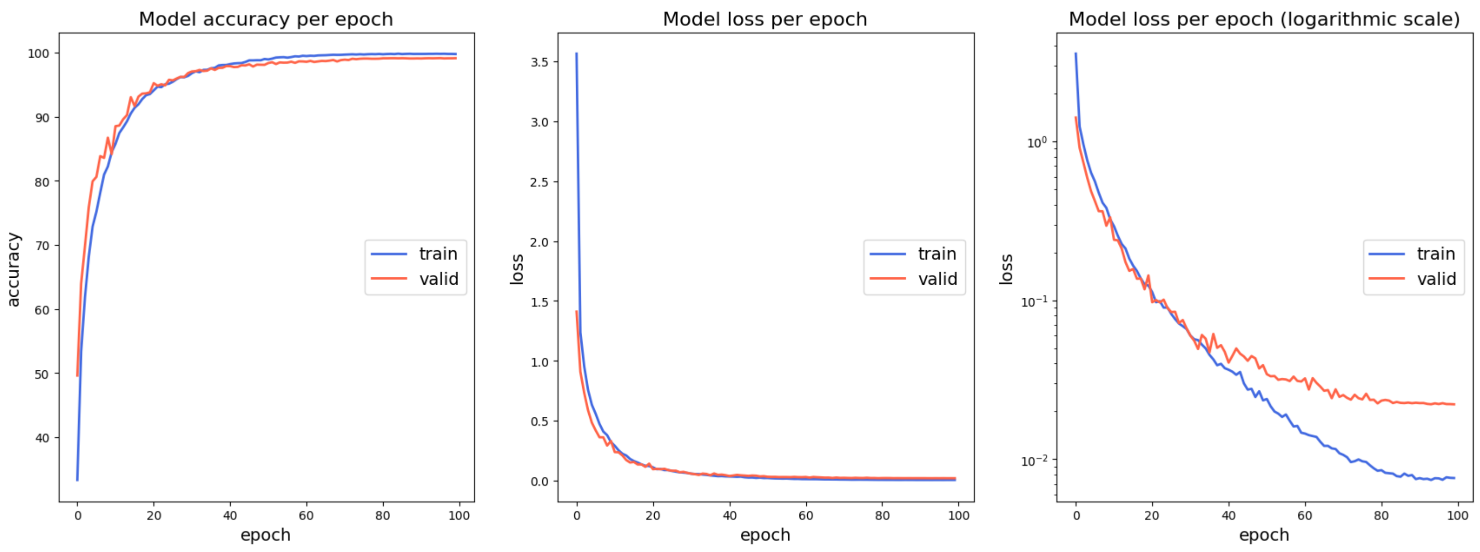


Figure 5: Accuracy & Loss

4.1.4 Training Time

Training time is a crucial aspect when evaluating the efficiency of a machine learning model. In the case of the before mentioned image classification model, the training was carried out on an 8-core TPU.

The model was trained over a total of 100 epochs. Despite the computational complexity of the architecture and the extent of the training period, the total time required to train the model was just 29.2 minutes. This relatively short training time can be attributed to both the efficiency of the model architecture and the high computational throughput provided by the 8-core TPU.

It is worth mentioning that training on a TPU not only expedites the training process but also allows for a more extensive exploration of hyperparameters or model architectures within a reasonable timeframe. The rapid training time thus augments the model's practical applicability, especially for tasks requiring quick iterative development or real-time deployment.

One of the noteworthy advantages of using TPUs is their scalability. While the present study employed an 8-core TPU, it is possible to scale up the training to TPU Pods, which can have up to hundreds of cores. TPU Pods offer an even higher level of parallelism and computational power, drastically reducing the training time further and making it feasible to train much larger models or datasets.

The architecture's scalability means that the model can be easily adapted to more complex tasks or larger datasets without a proportional increase in training time. This is particularly beneficial for real-world applications where quick iterative development and deployment are crucial. Leveraging TPU Pods could be an avenue for future research and production deployment, providing an opportunity to unlock even greater performance gains.

4.2 Testing data

4.2.1 Model Accuracy

The model was subjected to rigorous evaluation using an independent testing dataset to gauge its real-world applicability. Overall, the model correctly classified 8,089 out of 8,163 samples, yielding an accuracy rate of 99.09%. This high degree of accuracy validates the model's robustness and its ability to generalize to unseen data, following from high training and validation accuracies of 99.80% and 99.15% respectively.

4.2.2 Class-wise Performance

In addition to the overall testing accuracy, it is crucial to assess the model's performance across individual classes to ensure it is consistently accurate and not biased towards specific categories. The table below details the class-wise accuracy for the model:

Table 4: Class Specific Accuracies

Class	Accuracy
Apple___Apple_scab	100.00%
Apple___Black_rot	100.00%
Apple___Cedar_apple_rust	95.24%
Apple___healthy	97.98%
Blueberry___healthy	100.00%
Cherry___Powdery_mildew	100.00%
Cherry___healthy	99.22%
Corn___Cercospora_leaf_spot Gray_leaf_spot	90.91%
Corn___Common_rust	100.00%
Corn___Northern_Leaf_Blight	97.30%
Corn___healthy	100.00%
Grape___Black_rot	100.00%

Grape___Esca_(Black_Measles)	98.56%
Grape___Leaf_blight_(Isariopsis_Leaf_Spot)	100.00%
Grape___healthy	100.00%
Orange___Haunglongbing_(Citrus_greening)	100.00%
Peach___Bacterial_spot	99.13%
Peach___healthy	94.55%
Pepper, _bell___Bacterial_spot	98.67%
Pepper, _bell___healthy	99.10%
Potato___Early_blight	100.00%
Potato___Late_blight	97.33%
Potato___healthy	95.65%
Raspberry___healthy	100.00%
Soybean___healthy	100.00%
Squash___Powdery_mildew	100.00%
Strawberry___Leaf_scorch	98.80%
Strawberry___healthy	100.00%
Tomato___Bacterial_spot	98.75%
Tomato___Early_blight	96.67%
Tomato___Late_blight	94.77%
Tomato___Leaf_Mold	98.60%
Tomato___Septoria_leaf_spot	99.62%
Tomato___Spider_mites Two-spotted_spider_mite	99.60%
Tomato___Target_Spot	98.10%
Tomato___Tomato_Yellow_Leaf_Curl_Virus	99.75%
Tomato___Tomato_mosaic_virus	98.21%
Tomato___healthy	100.00%

Several classes, such as 'Apple___Apple_scab', 'Blueberry___healthy', and 'Corn___Common_rust' among others, exhibited a perfect accuracy of 100.00%. Most other classes also displayed high accuracy, with the lowest being 'Corn___Cercospora_leaf_spot Gray_leaf_spot' at 90.91%. These metrics confirm the model's capacity to identify and classify a wide array of plant diseases and conditions with a high degree of accuracy.

However, upon examining the confusion matrix below, it becomes apparent that the reason for this slightly reduced accuracy is that the model tends to confuse this class with the ‘Corn___Northern_Leaf_Blight’ class. This indicates that these two particular diseases may share some similar features that make them harder to distinguish, at least with the current configuration of the model.

This insight, derived from the Confusion Matrix, points to potential areas for improvement. Future iterations of the model could benefit from techniques designed to make finer distinctions between these two classes, such as specialized data augmentation or feature engineering.

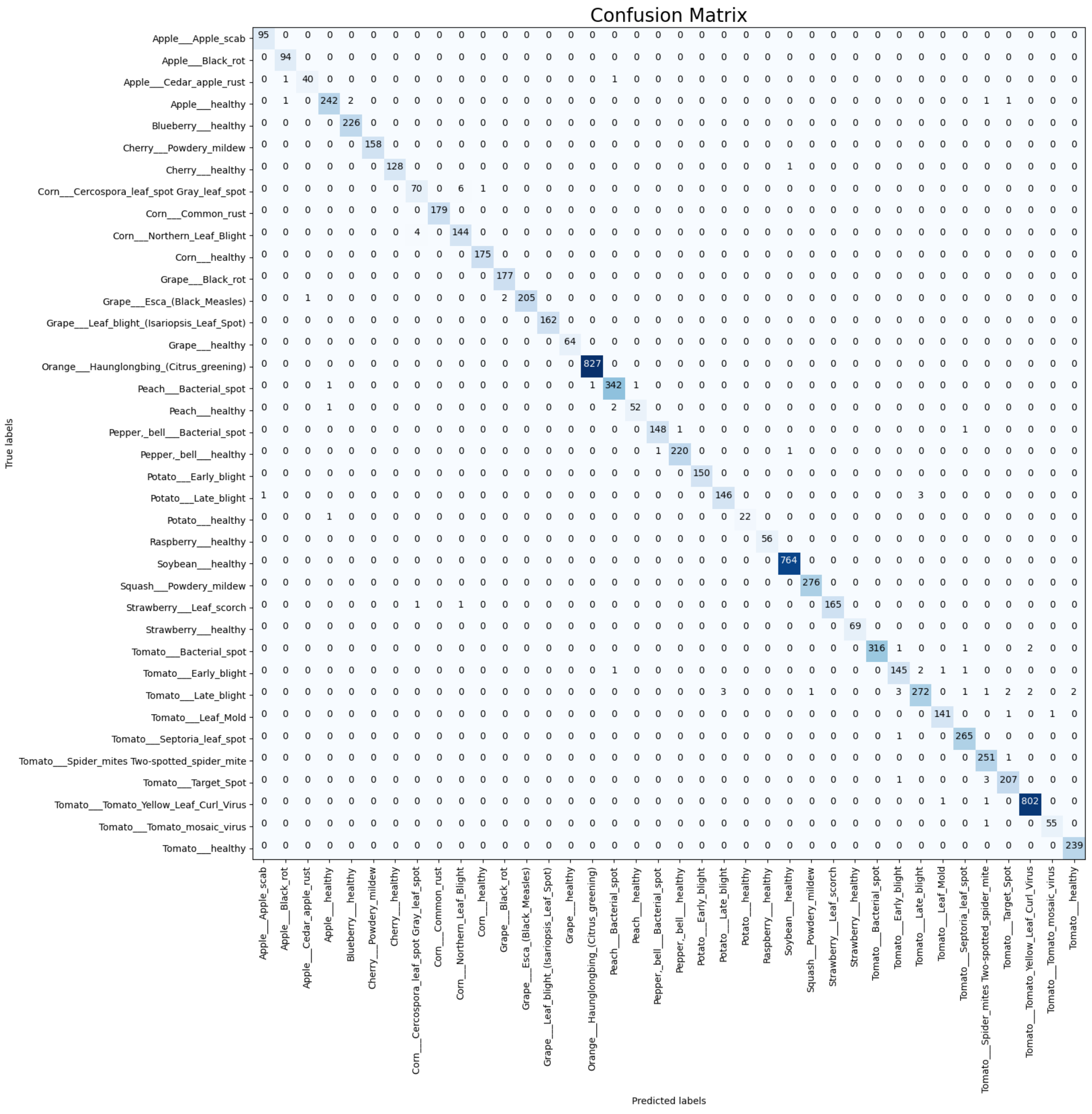


Figure 6: Confusion Matrix

5 Discussion

The results obtained from training and testing the deep learning model for plant disease classification are promising. The model achieved an outstanding accuracy rate of over 99% on both training and validation sets, as well as a similarly high accuracy on an independent testing set extracted from the PlantVillage dataset. This level of performance suggests a high degree of robustness and generalizability, making the model a strong candidate for practical applications in agricultural settings for real-time plant disease identification.

5.1 Training Efficiency and Scalability

The training time of approximately 29.2 minutes on an 8-core TPU v2 demonstrates the efficiency of both the chosen architecture and the TPU hardware. The prospect of scaling up the computational resources with TPU Pods offers an exciting avenue for future research and production deployments. It allows for a more rapid iteration, even for more complex models or larger datasets.

5.2 Class-wise Performance

The class-wise performance analysis provides additional depth to the understanding of the model's capabilities and limitations. While the model achieved perfect or near-perfect accuracy in many classes, it struggled slightly with certain diseases like 'Corn___Cercospora_leaf_spot Gray_leaf_spot.' These findings pinpoint areas where the model could be fine-tuned for even better performance.

5.3 Testing Dataset

It might be worth noting that although the testing dataset was independent in the context of this experiment, further validation could be beneficial. This could include the use of completely different datasets that the model has never encountered or field-testing in real-world agricultural settings. Such additional validation would provide more confidence in the model's utility and robustness, beyond the scope of the PlantVillage dataset used for training, validation, and testing.

5.4 Future Directions

Given the promising results and the identified areas for improvement, future work could focus on various aspects. This might include exploring more advanced architectures, optimizing hyperparameters further, or applying more specialized data augmentation techniques to improve class-specific performance.

6 Conclusion

In this study, a deep learning model for the identification of plant diseases using a Convolutional Neural Network (CNN) architecture was presented. The model was trained, validated, and tested on a dataset derived from the PlantVillage repository, encompassing a wide array of plant diseases and conditions across various plant species.

The results indicate a remarkable level of performance. With a training accuracy of 99.80%, a validation accuracy of 99.15%, and a testing accuracy of 99.09%, the model demonstrates both robustness and an excellent capacity to generalize to new, unseen data. Specific attention was paid to class-wise performance, showing high accuracy across a broad range of diseases. Minor shortcomings were identified through the use of a confusion matrix, pointing towards areas for future refinement.

The model was trained efficiently in just 29.2 minutes on an 8-core TPU v2, highlighting the computational effectiveness of the chosen architecture and the benefits of hardware acceleration. Scalability remains an exciting prospect, with TPUs offering the possibility for even more extensive computational power if needed.

It is important to note that while the testing dataset was independent within the scope of this study, additional validation on entirely different datasets or real-world scenarios would further substantiate the model's efficacy.

In summary, this research contributes to the growing body of work on applying machine learning techniques to agricultural challenges, offering a fast, accurate, and scalable solution for plant disease classification. Future work may focus on model refinement, scalability, and real-world applicability, including the implementation in smart agricultural systems.

List of References

Oerke, E.-C. (2006). Crop losses to pests. The Journal of Agricultural Science, 144(1), 31–43.

Nutter, F.W., et al. (1993). Disease assessment terms and concepts. Plant Disease, 77(3), 207-208.

Sankaran, S., et al. (2010). A review of advanced techniques for detecting plant diseases. Computers and Electronics in Agriculture, 72(1), 1-13.

Hu, M.G., et al. (2014). Support vector machine-based classification of leaf diseases using texture and color features. Optik-International Journal for Light and Electron Optics, 125(20), 5618-5621.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. Nature, 521, 436–444.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

Prechelt, L. (1998). Early stopping-but when? In Neural Networks: Tricks of the trade (pp. 55-69). Springer, Berlin, Heidelberg.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1), 1929-1958.

Ioffe, S., & Szegedy, C. (2015). 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 (pp. 448-456).

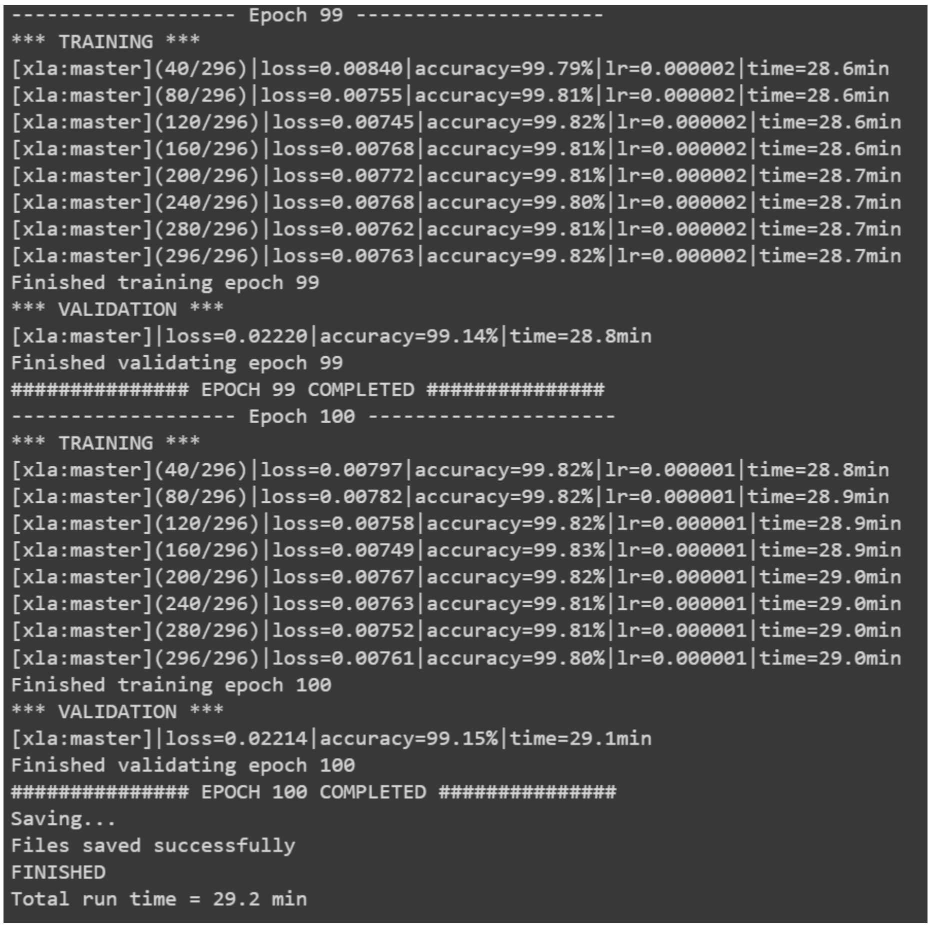
Mohanty, S.P., Hughes, D.P., and Salathé, M. (2016). Using deep learning for image-based plant disease detection. Frontiers in Plant Science, 7, 1419.

Jouppi, N.P., et al. (2017). In-datacenter performance analysis of a tensor processing unit. In Proceedings of the 44th Annual International Symposium on Computer Architecture, 1-12.

Barbedo, J.G.A. (2018). Factors influencing the use of deep learning for plant disease recognition. Biosystems Engineering, 172, 84-91.

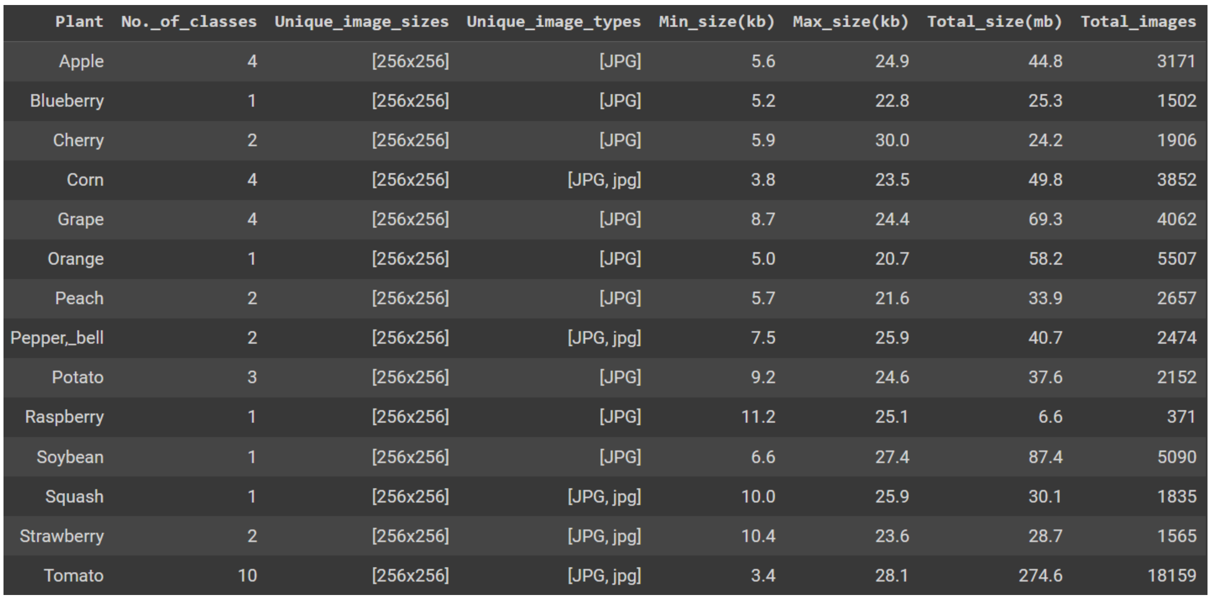
Doshi-Velez, F., and Kim, B. (2017). Towards a rigorous science of interpretable machine learning. arXiv preprint arXiv:1702.08608.

Appendices



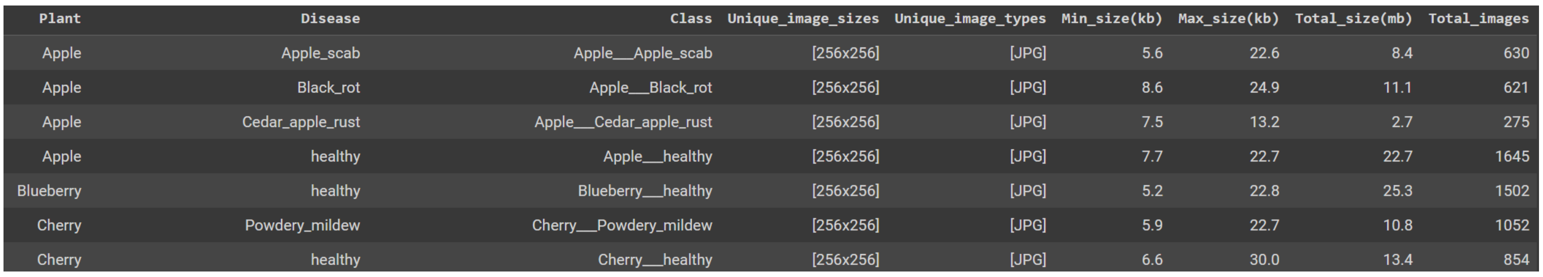
```
------------------- Epoch 99 ---------------------
*** TRAINING ***
[xla:master](40/296)|loss=0.00840|accuracy=99.79%|lr=0.000002|time=28.6min
[xla:master](80/296)|loss=0.00755|accuracy=99.81%|lr=0.000002|time=28.6min
[xla:master](120/296)|loss=0.00745|accuracy=99.82%|lr=0.000002|time=28.6min
[xla:master](160/296)|loss=0.00768|accuracy=99.81%|lr=0.000002|time=28.6min
[xla:master](200/296)|loss=0.00772|accuracy=99.81%|lr=0.000002|time=28.7min
[xla:master](240/296)|loss=0.00768|accuracy=99.80%|lr=0.000002|time=28.7min
[xla:master](280/296)|loss=0.00762|accuracy=99.81%|lr=0.000002|time=28.7min
[xla:master](296/296)|loss=0.00763|accuracy=99.82%|lr=0.000002|time=28.7min
Finished training epoch 99
*** VALIDATION ***
[xla:master]|loss=0.02220|accuracy=99.14%|time=28.8min
Finished validating epoch 99
############### EPOCH 99 COMPLETED ###############
------------------- Epoch 100 ---------------------
*** TRAINING ***
[xla:master](40/296)|loss=0.00797|accuracy=99.82%|lr=0.000001|time=28.8min
[xla:master](80/296)|loss=0.00782|accuracy=99.82%|lr=0.000001|time=28.9min
[xla:master](120/296)|loss=0.00758|accuracy=99.82%|lr=0.000001|time=28.9min
[xla:master](160/296)|loss=0.00749|accuracy=99.83%|lr=0.000001|time=28.9min
[xla:master](200/296)|loss=0.00767|accuracy=99.82%|lr=0.000001|time=29.0min
[xla:master](240/296)|loss=0.00763|accuracy=99.81%|lr=0.000001|time=29.0min
[xla:master](280/296)|loss=0.00752|accuracy=99.81%|lr=0.000001|time=29.0min
[xla:master](296/296)|loss=0.00761|accuracy=99.80%|lr=0.000001|time=29.0min
Finished training epoch 100
*** VALIDATION ***
[xla:master]|loss=0.02214|accuracy=99.15%|time=29.1min
Finished validating epoch 100
############### EPOCH 100 COMPLETED ###############
Saving...
Files saved successfully
FINISHED
Total run time = 29.2 min
```

Figure 7: End of the training loop



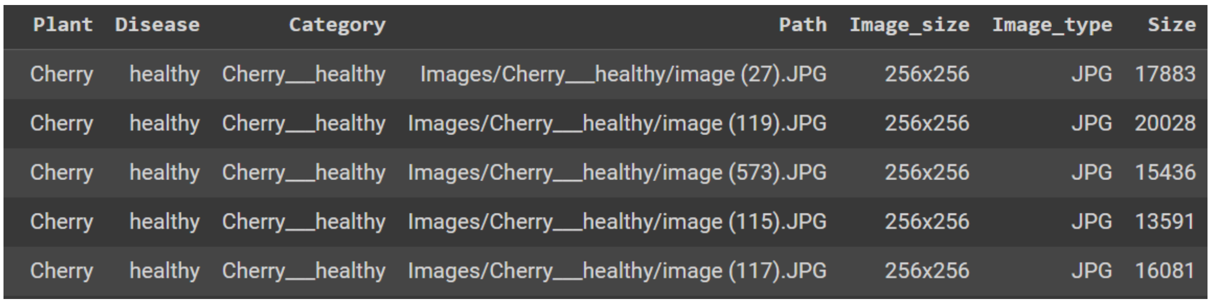
Plant	No._of_classes	Unique_image_sizes	Unique_image_types	Min_size(kb)	Max_size(kb)	Total_size(mb)	Total_images
Apple	4	[256x256]	[JPG]	5.6	24.9	44.8	3171
Blueberry	1	[256x256]	[JPG]	5.2	22.8	25.3	1502
Cherry	2	[256x256]	[JPG]	5.9	30.0	24.2	1906
Corn	4	[256x256]	[JPG, jpg]	3.8	23.5	49.8	3852
Grape	4	[256x256]	[JPG]	8.7	24.4	69.3	4062
Orange	1	[256x256]	[JPG]	5.0	20.7	58.2	5507
Peach	2	[256x256]	[JPG]	5.7	21.6	33.9	2657
Pepper,_bell	2	[256x256]	[JPG, jpg]	7.5	25.9	40.7	2474
Potato	3	[256x256]	[JPG]	9.2	24.6	37.6	2152
Raspberry	1	[256x256]	[JPG]	11.2	25.1	6.6	371
Soybean	1	[256x256]	[JPG]	6.6	27.4	87.4	5090
Squash	1	[256x256]	[JPG, jpg]	10.0	25.9	30.1	1835
Strawberry	2	[256x256]	[JPG, jpg]	10.4	23.6	28.7	1565
Tomato	10	[256x256]	[JPG, jpg]	3.4	28.1	274.6	18159

Figure 8: Information about dataset grouped by plant species



Plant	Disease	Class	Unique_image_sizes	Unique_image_types	Min_size(kb)	Max_size(kb)	Total_size(mb)	Total_images
Apple	Apple_scab	Apple___Apple_scab	[256x256]	[JPG]	5.6	22.6	8.4	630
Apple	Black_rot	Apple___Black_rot	[256x256]	[JPG]	8.6	24.9	11.1	621
Apple	Cedar_apple_rust	Apple___Cedar_apple_rust	[256x256]	[JPG]	7.5	13.2	2.7	275
Apple	healthy	Apple___healthy	[256x256]	[JPG]	7.7	22.7	22.7	1645
Blueberry	healthy	Blueberry___healthy	[256x256]	[JPG]	5.2	22.8	25.3	1502
Cherry	Powdery_mildew	Cherry___Powdery_mildew	[256x256]	[JPG]	5.9	22.7	10.8	1052
Cherry	healthy	Cherry___healthy	[256x256]	[JPG]	6.6	30.0	13.4	854

Figure 10: Sample of dataset information grouped by plant and disease



Plant	Disease	Category	Path	Image_size	Image_type	Size
Cherry	healthy	Cherry___healthy	Images/Cherry___healthy/image (27).JPG	256x256	JPG	17883
Cherry	healthy	Cherry___healthy	Images/Cherry___healthy/image (119).JPG	256x256	JPG	20028
Cherry	healthy	Cherry___healthy	Images/Cherry___healthy/image (573).JPG	256x256	JPG	15436
Cherry	healthy	Cherry___healthy	Images/Cherry___healthy/image (115).JPG	256x256	JPG	13591
Cherry	healthy	Cherry___healthy	Images/Cherry___healthy/image (117).JPG	256x256	JPG	16081

Figure 9: Sample of dataset information for each individual sample