Università degli Studi di Milano
Data Science and Economics

Convolutional Neural Networks for automated recognition of plants leaves

Course: Algorithms for Massive Data
Student: Marta Magnani (961071)

Academic year 2022-23

Contents

List of Figures

List of Tables

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university, and I accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

1 Introduction

Deep Learning, a powerful and transformative field within Artificial Intelligence, has gained prominence in today's era of advanced technology. It has revolutionized various domains, including image classification, by learning intricate patterns and representations directly from data. Image classification, a task within Computer Vision, involves automatically categorizing images into predefined classes or categories. This ability to analyze and comprehend visual data has widespread applications, such as autonomous driving, medical diagnosis, facial recognition, and content organization.

Convolutional Neural Networks (CNNs) are commonly used for image recognition. These networks excel at extracting features from images and learning to recognize patterns. Inspired by the functioning of the animal visual cortex, CNNs are a specialized class of deep neural networks where small regions of input activate neurons. By applying convolutional filters to local image regions and gradually aggregating information, CNNs effectively identify intricate features and patterns. This report delves into the task of Image Classification with a focus on a dataset encompassing twelve distinct plant leaf species. The main objectives are threefold: firstly, to accurately identify the leaf's condition (healthy or diseased); secondly, to classify images based on the twelve species; and finally, to categorize into 22 divisions using both the species' name and the state of the leaf. These aims were met by developing six Convolutional Neural Network (CNN) architectures.

The structure of this report is as follows: Section 2 introduces the theory underlying Feedforward Neural Networks, with a special emphasis on Convolutional Neural Networks. Section 3 outlines the computational environment and the packages used in constructing the Machine Learning models. The structure of the dataset and the preprocessing steps undertaken are discussed in Section 4. Section 5 presents the implemented CNN models. Section 6 provides an analysis of the results along with associated visualisations. The concluding suggestions for improving model performance and scalability are presented in Section 7.

The code for this project is available on GitHub[1].

2 Neural Networks architecture

2.1 Feedforward Neural Networks

Artificial Neural Networks (ANNs) are computational structures designed to mimic the organization and functionality of biological neural networks. They consist of interconnected artificial neurons that can receive input signals, process them, and transmit output signals to other neurons through weighted connections. These connections, represented by coefficients called weights, are adjusted during the training phase to determine the significance of each input signal. Neurons within ANNs are organized into layers, forming interconnected groups of nodes. The configuration and purpose of these layers define the type of neural network being used. One well-known type is the Feed Forward Neural Network (FFNN), where neurons in each layer are exclusively connected to neurons in the subsequent layer. This arrangement forms an acyclic graph, enabling the flow of information from the input layer through the hidden layers and eventually to the output layer.

[1] GitHub repository: https://github.com/mmartamagna/Plant-leaves-classification-AMD

When an FFNN incorporates multiple hidden layers, it is referred to as a Multilayered Feedforward Neural Network (MFFNN). This expanded architecture allows for the extraction of increasingly complex features and representations as information traverses through the network. By stacking layers and adjusting the weights, MFFNNs possess the capability to learn and model intricate relationships within input data, making them highly effective in tasks such as pattern recognition, classification, and regression. Figure 1 shows an example of the general architecture[2].

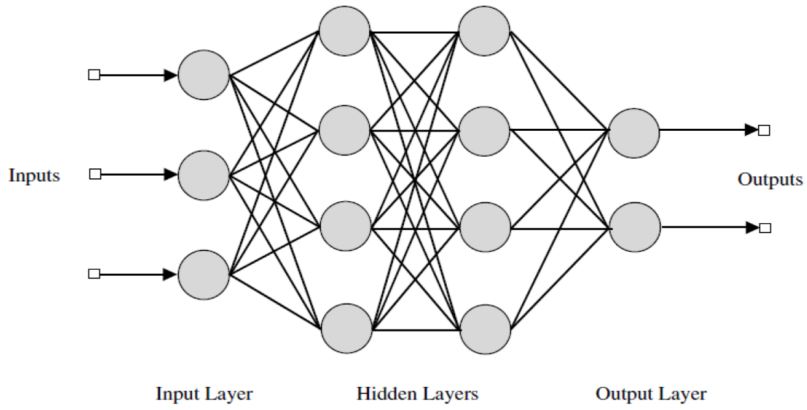

Figure 1: Multilayered Feedforward Neural Network architecture

2.2 Convolutional Neural Networks

This project centres around implementing Convolutional Neural Networks (CNNs) for both binary classification (diseased-healthy) and multiclass classification (plant species). CNNs belong to a specialized class of Feed Forward Neural Networks that excel in visual tasks and find widespread application in domains like spatial data analysis, computer vision, natural language processing, signal processing, and more. The architecture of CNNs draws inspiration from the visual cortex in animals, where neurons respond selectively to specific regions of their input. In CNNs, a key concept is the utilization of a kernel, represented by a square matrix, to process localized regions of the input image. By sliding this kernel across the entire input matrix, the convolution operation is performed, extracting relevant features and generating activation maps or feature maps. This process is repeated systematically, enabling the network to uncover intricate patterns and spatial relationships within the input data. This approach offers a significant advantage over traditional fully connected topologies, as it allows the network to capture spatial correlations in the inputs, whereas conventional networks necessitate the linearization of input data.

Each convolutional layer in the network plays a crucial role in downsizing the input matrix by consolidating regions into singular outputs for subsequent layers. Typically, the final segment of a Convolutional Neural Network (CNN) consists of a fully connected network. As illustrated in Figure 2[3], the convolutional part of the network focuses on extracting patterns from the input, encompassing even complex patterns. These patterns are then forwarded as features to a traditional network, which specializes in learning classification tasks based on these extracted features. Consequently, the overall structure of the CNN can be divided into two distinct sections: the initial section dedicated to feature learning and extraction and the subsequent

[2]Source: https://cse22-iiith.vlabs.ac.in/exp/forward-neural-networks/theory.html
[3]Source: https://developersbreach.com/convolution-neural-network-deep-learning/

section devoted to the classification task.

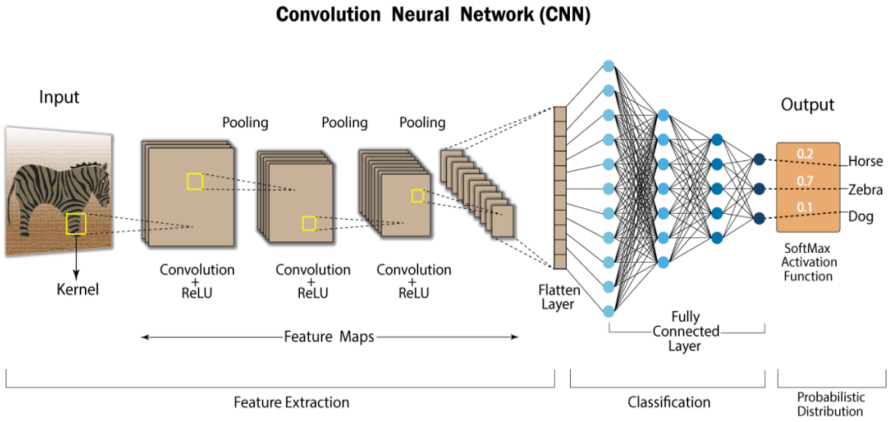

Figure 2: Convolutional Neural Network architecture

2.2.1 CNN architecture

One of the crucial aspects of the CNN implementation consists of the choice of layers. Going into more detail, a CNN generally has three layers: a convolutional layer, a pooling layer, a flattening layer and a fully connected layer.

- **Convolutional layer**. This is the first type of layer commonly found in CNNs and applies a set of learnable filters (also known as kernels) to the input image, performing a convolution operation that extracts local features. The filters slide over the input, capturing spatial patterns and creating feature maps or activation maps that highlight the relevant information.
- **Pooling layer**. After the convolution, the pooling layer further modifies and down-sample the output of the layer. The pooling layer replaces the output of the network at certain locations by deriving a summary statistic of the nearby outputs. This helps reduce the representation's spatial size, reducing computational costs and allowing the learning of high-level patterns by deeper layers of the network. Common pooling operations include max pooling, which selects the maximum value in each pooling region, and average pooling, which calculates the average value.
- **Flattening layers**. After the convolutional and pooling layers, the resulting feature maps are typically in a multidimensional format. Flattening is the process of converting these multidimensional feature maps into a one-dimensional vector.
- **Dense layers**.The one-dimensional vector created by the Flatten layer can then pass through one or a series of Dense Layers, where each neuron performs a weighted sum of the inputs received from the previous layer. These weights are learned during the training process. After the weighted sum, an activation function is applied to introduce non-linearity into the network. Common activation functions include the sigmoid function, ReLU (Rectified Linear Unit), and others. The output of each neuron in the Dense layer is then passed to the next layer, or in the case of the output layer, it represents the final prediction or value.

- **Output layer**. The output layer is the final layer of a neural network that produces the desired output based on the specific task at hand. It is responsible for generating the predictions or classifications for the input data. The configuration of the output layer depends on the nature of the problem being solved. For binary classification tasks, where the goal is to classify input data into two classes, the output layer typically consists of a single neuron with a sigmoid activation function. The output of this neuron represents the probability or confidence of the input belonging to the positive class.

Additional layers that can be inserted are Dropout and Batch Normalization layers, which consist of regularization techniques that can improve the generalization performance of a network.

- **Dropout layers**. Dropout layers are typically added after fully connected layers in a network and they nullify the contribution of some neurons towards the next layer and leave all others unmodified. During training, dropout layers randomly set a fraction of the input units (neurons) to zero at each update, effectively "dropping out" those units. This prevents the network from relying too much on specific neurons and encourages the network to learn more robust and generalized features. These layers are usually implemented to prevent overfitting the training data.
- **Batch normalization layers**. Batch normalization layers are usually placed after a network's convolutional or fully connected layers and it is a technique used to improve the stability and speed of training deep neural networks. It is a method used to make the training of artificial neural networks faster and more stable through the normalization of the layers' inputs (i.e., it normalizes the input of a layer by subtracting the mean and dividing by the standard deviation of the mini-batch). This helps to alleviate the problem of internal covariate shift and ensures that the input to each layer remains within a suitable range.

3 Environment and tools

The code for this project was written in Python 3 on Google Colab, an online environment allowing the use of free cloud-based Jupyter Notebooks, a well-suited tool for machine learning and data analysis work. This platform also provides access to computing resources, including some powerful GPUs, which are useful when training ML models. During the development of the project and the implementation of the models, we primarily used the following libraries:

- **OpenCV**: This open-source library, short for Open Source Computer Vision, is highly useful for image manipulation and processing in the preprocessing phase. It contains several hundreds of computer vision algorithms.
- **Python Imaging Library - PIL**: This library supports opening, manipulating, and saving many different image file formats. It is used in tandem with OpenCV for broader image format recognition.
- **Numpy**: This library is fundamental for numerical computing in Python. It allows efficient operations on large arrays and matrices of numeric data, which is crucial in machine learning tasks.
- **Tensorflow**: This is an open-source library widely used in Machine Learning. It provides a framework for defining, training, and deploying machine learning models. It's particularly well-suited for building neural networks used in deep learning.

- **Matplotlib**: This plotting library allows us to create a wide range of static, animated, and interactive plots in Python. It's used for visualizing data and model performance.
- **Seaborn**: This is a Python data visualization library based on Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. Seaborn helps to improve the aesthetics of Matplotlib plots and create more complex visualizations.
- **Pandas**: Pandas stands out for its proficiency in data manipulation and analysis, providing us with robust tools to handle, structure, and interpret our data effectively. It streamlines data pre-processing by offering functionalities for tasks like handling missing data, merging or reshaping data frames, and applying complex filtering rules.

Keras and scalability

Most notably, the project harnesses the power of Keras, a high-level neural networks API, to build and train our deep learning models. Keras is instrumental in managing large datasets due to its simplicity and modularity, enabling rapid model design and experimentation. It features comprehensive utilities for data loading and preprocessing, aiding in efficient handling of our data. The built-in support for convolutional networks makes it an excellent choice for our leaf image classification task.

Moreover, Keras supports both CPU and GPU computation, allowing the project to scale as per the data and computational resources. It also supports distributed training, enabling us to efficiently train our models across multiple GPUs or even across a network of computers. This is particularly relevant when working with extensive datasets, as it accelerates the training process. Further, Keras' compatibility with TensorFlow ensures a suite of advanced functionalities like the tf.data API for efficient data pipelining and TensorFlow Extended (TFX) for end-to-end platform support. This makes it possible to seamlessly transition from model development and experimentation to deployment and serving in large-scale, production environments.

4 Dataset

The original dataset[4] is a collection of twelve plant species pictures classified among two classes (healthy and diseased). It is a collection of about 4503 images of which contains 2278 images of healthy leaf and 2225 images of the diseased leaf. The twelve plant species are named as Mango, Arjun, Alstonia Scholaris, Guava, Bael, Jamun, Jatropha, Pongamia Pinnata, Basil, Pomegranate, Lemon, and Chinar. The downloaded data were already divided as follows:

- **Train**: 4274 images
- **Validation**: 110 images
- **Test**: 110 images

Each folder is divided into 22 subfolders containing images named after the individual species and the state of health (e.g., Basil healthy (P8); Mango healthy (P0a); Mango diseased (P0b); etc.). In Appendix A are reported all the healthy and diseased species pictures, respectively in Figure 3 and Figure 4. The histograms illustrating the number of images per species for verifying their balance across the folders are instead in Figure 5.

[4]Source: https://www.kaggle.com/datasets/csafrit2/plant-leaves-for-image-classification

Data preprocessing

Data preprocessing for Convolutional Neural Networks (CNN) is a crucial step in preparing data for modelling. This step involves a number of processes aimed at converting raw data into a format that is easily understood by CNNs. Here the steps the aim of this project:

1. Resize (128x128) The first step was to resize all the images to 128x128 pixels to give more homogeneity and uniformity to the dataset.

2. Three-Channel Pixels The downloaded images were already provided in a three-channel format. To preserve the richness of the information contained within, I opted to retain their original configuration.

3. Normalizing pixel values To improve model performances, the pixel values have been re-scaled from the [0, 255] range to the [0, 1] range.

4. Label encoding One-hot encoding is employed to transform categorical or text data into a numerical format compatible with machine learning algorithms, thus enhancing their predictive capacity. Through this process, each unique category in the original data becomes a new column containing binary indicators (1s and 0s) representing the presence or absence of the given category. This encoding is pivotal in enabling machine learning models to process and effectively utilize categorical data for predictions. In this project, one-hot encoding was applied in three distinct instances: firstly, the health statuses of the plants were encoded as '0' for 'Healthy' and '1' for 'Diseased'. Secondly, the encoding was applied to the twelve distinct plant species. Finally, it was utilized to represent the twenty-two unique complete names of plants.

5 Convolutional Neural Networks architecture

5.1 CNNs Models

Binary classification (healthy - diseased)

The first two CNN models (Model 1 and Model 2) developed for this project consist of multiple layers for extracting features from input images and making binary predictions (i.e., predicting whether the leaf is healthy or diseased). The two models are similar in terms of layers. Still, the second one is enriched with two regularisation layers (i.e., Dropout(0.5) and Batch Normalization layers) which can help to reduce overfitting and improve the training process.The input images are sized (128x128) and have 3 channels; the final layer activation function is Sigmoid; before the training phase the 2 labels were hot-encoded (0 for "Healthy" and 1 for "Diseased"). he whole architecture of Model 1 and Model 2 can be seen in Table 5.1 and Table 5.2.

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 126, 126, 8)	224
max_pooling2d_6 (MaxPooling2D)	(None, 42, 42, 8)	0
conv2d_7 (Conv2D)	(None, 40, 40, 16)	1168
max_pooling2d_7 (MaxPooling2D)	(None, 13, 13, 16)	0
conv2d_8 (Conv2D)	(None, 11, 11, 32)	4640
max_pooling2d_8 (MaxPooling2D)	(None, 3, 3, 32)	0
flatten_2 (Flatten)	(None, 288)	0
dense_4 (Dense)	(None, 32)	9248
dense_5 (Dense)	(None, 2)	66
Total params		15,346
Trainable params		15,346
Non-trainable params		0

Table 5.1: Model 1 architecture

Layer (type)	Output Shape	Param #
conv2d_9 (Conv2D)	(None, 126, 126, 8)	224
max_pooling2d_9 (MaxPooling2D)	(None, 42, 42, 8)	0
conv2d_10 (Conv2D)	(None, 40, 40, 16)	1168
max_pooling2d_10 (MaxPooling2D)	(None, 13, 13, 16)	0
conv2d_11 (Conv2D)	(None, 11, 11, 32)	4640
max_pooling2d_11 (MaxPooling2D)	(None, 3, 3, 32)	0
flatten_3 (Flatten)	(None, 288)	0
dense_6 (Dense)	(None, 32)	9248
dropout_1 (Dropout)	(None, 32)	0
batch_normalization_1 (BatchNormalization)	(None, 32)	128
dense_7 (Dense)	(None, 2)	66
Total params		15,474
Trainable params		15,410
Non-trainable params		64

Table 5.2: Model 2 architecture

Multiclass classification (12 species)

Model 3 and Model 4 are similar to the previous ones in terms of CNN structure but are meant for predicting more than one category. In particular, Model 3 and Model 4 aim at predicting the right plant species among the 12 made available (e.g., Mango, Basil, Pongama Pinnata, Gauva etc). Model 5 and Model 6, instead, aim at predicting the right complete label among the 22 presented in the original dataset and containing both the plant name and the state of health (e.g., Pomegranate diseased (P9b), Mango diseased (P0b), etc.). As in the binary classification case, Model 4 and Model 6 were enriched with two regularisation layers (i.e., Dropout(0.5) and Batch Normalization layers) to reduce overfitting and improve the training process. The input images are sized (128x128) and have 3 channels; the final layer activation function is ReLu; before the training phase, all the 12 labels were hot-encoded. The architecture summary of Model 3 and Model 4 are in Table 5.3 and Table 5.4.

Layer (type)	Output Shape	Param #
conv2d_12 (Conv2D)	(None, 126, 126, 8)	224
max_pooling2d_12 (MaxPooling2D)	(None, 42, 42, 8)	0
conv2d_13 (Conv2D)	(None, 40, 40, 16)	1168
max_pooling2d_13 (MaxPooling2D)	(None, 13, 13, 16)	0
conv2d_14 (Conv2D)	(None, 11, 11, 32)	4640
max_pooling2d_14 (MaxPooling2D)	(None, 3, 3, 32)	0
flatten_4 (Flatten)	(None, 288)	0
dense_8 (Dense)	(None, 32)	9248
dense_9 (Dense)	(None, 12)	396
Total params		15,676
Trainable params		15,676
Non-trainable params		0

Table 5.3: Model 3 architecture

Layer (type)	Output Shape	Param #
conv2d_15 (Conv2D)	(None, 126, 126, 8)	224
max_pooling2d_15 (MaxPooling2D)	(None, 42, 42, 8)	0
conv2d_16 (Conv2D)	(None, 40, 40, 16)	1168
max_pooling2d_16 (MaxPooling2D)	(None, 13, 13, 16)	0
conv2d_17 (Conv2D)	(None, 11, 11, 32)	4640
max_pooling2d_17 (MaxPooling2D)	(None, 3, 3, 32)	0
flatten_5 (Flatten)	(None, 288)	0
dense_10 (Dense)	(None, 32)	9248
dropout_2 (Dropout)	(None, 32)	0
batch_normalization_2 (BatchNormalization)	(None, 32)	128
dense_11 (Dense)	(None, 12)	396
Total params		15,804
Trainable params		15,740
Non-trainable params		64

Table 5.4: Model 4 architecture

Multiclass classification (22 complete names)

Model 5 and Model 6 are similar to the previous ones in terms of CNN structure but are meant for predicting the right complete label among the 22 presented in the original dataset and containing both the plant name and the state of health (e.g., Pomegranate diseased (P9b), Mango diseased (P0b), etc.). As in the binary classification case, Model 6 contains two regularisation layers (i.e., Dropout(0.5) and Batch Normalization layers) to reduce overfitting and improve the training process. The input images are sized (128x128) and have 3 channels; the final layer activation function is ReLu; before the training phase all the 22 labels were hot-encoded. The architecture summary of Model 5 and Model 6 are in Table 5.5 and Table 5.6.

Layer (type)	Output Shape	Param #
conv2d_18 (Conv2D)	(None, 126, 126, 8)	224
max_pooling2d_18 (MaxPooling2D)	(None, 42, 42, 8)	0
conv2d_19 (Conv2D)	(None, 40, 40, 16)	1168
max_pooling2d_19 (MaxPooling2D)	(None, 13, 13, 16)	0
conv2d_20 (Conv2D)	(None, 11, 11, 32)	4640
max_pooling2d_20 (MaxPooling2D)	(None, 3, 3, 32)	0
flatten_6 (Flatten)	(None, 288)	0
dense_12 (Dense)	(None, 32)	9248
dense_13 (Dense)	(None, 22)	726
Total params		16,006
Trainable params		16,006
Non-trainable params		0

Table 5.5: Model 5 architecture

Layer (type)	Output Shape	Param #
conv2d_21 (Conv2D)	(None, 126, 126, 8)	224
max_pooling2d_21 (MaxPooling2D)	(None, 42, 42, 8)	0
conv2d_22 (Conv2D)	(None, 40, 40, 16)	1168
max_pooling2d_22 (MaxPooling2D)	(None, 13, 13, 16)	0
conv2d_23 (Conv2D)	(None, 11, 11, 32)	4640
max_pooling2d_23 (MaxPooling2D)	(None, 3, 3, 32)	0
flatten_7 (Flatten)	(None, 288)	0
dense_14 (Dense)	(None, 32)	9248
dropout_3 (Dropout)	(None, 32)	0
batch_normalization_3 (BatchNormalization)	(None, 32)	128
dense_15 (Dense)	(None, 22)	726
Total params		16,134
Trainable params		16,070
Non-trainable params		64

Table 5.6: Model 6 architecture

5.2 Hyperparameters, activation functions and performance metrics

The two models are compiled with the following hyperparameters:

- Epochs: 30
- Mini-batch size: 128
- Optimizer: Adam optimizer
- Metrics: Binary accuracy (Model 1, Model 2); Accuracy (Model 3, Model 4, Model 5, Model 6).
- Loss function: Binary cross-entropy (Model 1, Model 2); Categorical cross-entropy (Model 3, Model 4, Model 5, Model 6).

Binary cross-entropy

Binary Cross-Entropy is a loss function used in binary classification tasks (Healthy or Diseased). Similar to categorical cross-entropy, binary cross-entropy is a measure from the field of information theory, building upon entropy and generally calculating the difference between two probability distributions. It is used to quantify the difference between the true and predicted class labels. The formula gives the Binary Cross-Entropy loss:

$$-\sum_{i=1}^{n}[y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)]$$

where: n is the number of instances. y_i is the true class label (0 or 1) for instance i. p_i is the predicted probability that instance i is of class 1. This loss function increases when the prediction diverges from the target label. As the prediction approaches 1, the loss function slowly decreases. Conversely, when the prediction decreases, the log loss increases quickly. Therefore, there is higher penalisation for those errors with high confidence.

Multiclass cross-entropy

Categorical Cross-Entropy is a loss function that is used in multi-class classification tasks. These are tasks where an example can belong to one of multiple possible categories, and the model must decide which one. Cross-Entropy is a measure from the field of information theory, building upon entropy and generally calculating the difference between two probability distributions. It is used to quantify the difference between the true and predicted class labels. The formula gives the Categorical Cross-Entropy loss:

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

where: M is the number of classes. $y_{o,c}$ is a binary indicator of whether class c is the correct classification for observation o. $p_{o,c}$ is the predicted probability that observation o is of class c.

6 Results and visualization

Accuracy and loss

The table 6.1 below, titled "Training, Validation, and Test Performance for Each Model," provides a comprehensive comparison of the performance of six different models on the training, validation, and test datasets. The performance is evaluated based on two metrics - accuracy and loss. For each model, the training accuracy ('Train Acc.') and training loss ('Train Loss') are presented first. This indicates how well the model has learned from the training data. The next two columns show the validation accuracy ('Valid Acc.') and validation loss ('Valid Loss'), which give an indication of how well the model generalizes to new data, by testing it on the validation set which the model has not seen during training. The final two columns display the test accuracy ('Test Acc.') and test loss ('Test Loss'). These metrics show the model's performance on the test data, providing the ultimate measure of its ability to make accurate predictions on unseen data. A quick examination of the results reveals that:

- Model 1 exhibits a good balance between training accuracy and validation accuracy, with values of 0.895 and 0.827, respectively. The test accuracy is also relatively high at 0.809, demonstrating reasonable generalization to unseen data.

- Model 2 presents the highest training accuracy at 0.949, which suggests it learned the training dataset most effectively. However, the higher validation and test losses imply that this model may be overfitting, as it is not generalizing as well to unseen data.

- Model 3 shows an interesting pattern where its test accuracy (0.845) is higher than its training accuracy (0.849). This scenario might indicate that the model is underfitting the training data but somehow manages to perform well on the test set.

- Models 4, 5, and 6 progressively exhibit lower accuracies and higher losses. Specifically, Model 6 has the lowest accuracy and highest loss, indicating that it is the least effective model among those developed.

Further experimentation and model fine-tuning could potentially yield better results, as these metrics only represent the performance of the initial iterations of the models.

Model	**Train Acc.**	**Train Loss**	**Valid Acc.**	**Valid Loss**	**Test Acc.**	**Test Loss**
Model 1	0.895	0.253	0.827	0.406	0.809	0.453
Model 2	0.949	0.133	0.836	0.578	0.818	0.410
Model 3	0.849	0.464	0.764	0.814	0.845	0.517
Model 4	0.796	0.626	0.827	0.537	0.800	0.556
Model 5	0.806	0.607	0.600	1.312	0.673	1.184
Model 6	0.581	1.270	0.500	1.266	0.682	1.108

Table 6.1: Training, Validation, and Test Performance for Each Model

Graphical interpretation

The presented visuals serve to elucidate the performance of our CNN models, providing key insights into potential issues of overfitting and underfitting. Comparing Model 1 and Model 2 (Figure 6, Appendix B), a discernible divergence in the trajectory of the training and validation curves is noticeable. In Model 1, the training and validation lines follow parallel paths, albeit spaced apart. The validation line consistently remains above the training line in the loss graph and below it in the accuracy graph. This indicates a potential overfitting issue throughout all epochs, where the model performs well on the training data but does not generalize well to the validation set. The overfitting issue becomes even more apparent in Model 2. Here, the model is over-specializing to the training data, which results in poorer performance when presented with unseen validation data. This is evidenced by a significant gap between the validation and training loss functions throughout all epochs, indicating the model's struggle to generalize beyond its training data. Despite the implementation of regularization layers intended to mitigate overfitting, Model 2 shows no discernible improvement. This suggests that these regularization measures were not successful in enhancing the model's performance. This highlights the need for further exploration of model parameters and additional strategies to improve generalization.

When examining the multiclass classification conducted by Model 3 and Model 4, which differentiate between 12 species, we can observe their performance metrics in Figure 7 (Appendix B). Model 3 demonstrates an intriguing pattern where the training and validation loss curves overlap until epoch 3. Notably, the loss values at these early stages are high, suggesting an underfitting scenario. Beyond the third epoch, the curves begin to diverge. Both curves decrease in a logarithmic pattern, but the validation loss consistently stays above the training loss. Concurrently, the validation accuracy lags behind the training accuracy. These observations point towards the existence of overfitting - the model seems to learn the training data too

well, hindering its ability to generalize to unseen data. On the other hand, when we introduce regularization layers in Model 4, the curves exhibit a different dynamic. Starting from epoch 15, the validation loss curve begins to converge with the training curve, and subsequently oscillates below it. This pattern might be due to the regularization layers of Model 4, which can sometimes cause the training loss to be higher than the validation loss because regularization adds a penalty to the loss function calculated on the training set.

Finally, Model 5 and Model 6 conducted a multiclass classification among 22 complete names, for which we can observe their performance metrics in Figure 8 (Appendix B). Model 5 demonstrates a pattern where the training and validation loss tend to stay separate across all epochs. In particular, the validation loss curve is above the train one indicating a clear overfitting behaviour - the model seems to learn the training data too well, hindering its ability to generalise to unseen data. On the other hand, when we introduce regularization layers in Model 6, the curves exhibit a different dynamic. Starting from epoch 17, the validation loss curve begins to converge with the training curve, and subsequently oscillates mainly below it. This pattern might be due to the regularization layers of Model 6, which cause the training loss to be higher than the validation loss because regularisation adds a penalty to the loss function calculated on the training set.

Confusion matrix

A confusion matrix, also known as an error matrix, is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one. It's often used in machine learning and statistics to understand the nature of errors in the classification problem. Each row in the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa). The name stems from making it easy to see if the system is confusing two classes (i.e., mislabeling one as another). In particular, for multi-class classification problems, the matrix will be NxN, where N is the number of classes. The corresponding confusion matrices for the various models are detailed in the Appendice B. For binary models, namely Model 1 and Model 2, refer to Figure 9. For the classification of the 12 plant species (Model 3 and Model 4), Figure 10 can be consulted. For the comprehensive 22 label classification (Model 5 and Model 6), Figure 11 offers pertinent insights.

Upon scrutinizing these matrices, the binary classification performance of Model 1 and Model 2 exhibits remarkable resemblance. For instance, both models misclassified 16 healthy leaves as diseased. Model 1 also inaccurately identified 5 diseased images as healthy, while Model 2 had 4 such instances. In the context of the 12-species classification, certain plant species demonstrate exceptional predictability. In Model 3, 'Jamun' is consistently predicted with flawless accuracy. Similarly, 'Arjun', 'Jamun' and 'Lemon' exhibited impeccable prediction in Model 4. Nevertheless, Model 4 struggles significantly with the 'Pongamia Pinnata' species, as it misclassified half of the corresponding images. The 22-label classification models, Model 5 and Model 6, yield some noteworthy observations. Model 5 consistently failed to correctly predict 'Guava diseased', 'Lemon diseased' and 'Pongamia Pinnata diseased'. The presence of zeros in the matrix diagonal for these species signifies their total misclassification. On the other hand, Model 6 inaccurately predicted all instances of 'Lemon diseased', 'Pomegranate healthy', and 'Pongamia Pinnata diseased'.

Performance metrics

The confusion matrices presented above form the basis for many classifier metrics, including accuracy, precision, recall and F1-score. These performance metrics provide a complete evaluation

of a classification model.

1. Accuracy is the most intuitive performance measure. It is simply a ratio of correctly predicted observations to the total observations.

2. Precision is the ratio of correctly predicted positive observations to the total predicted positives. It gives us information about how many of the identified positive instances are actually positive.

3. Recall (Sensitivity) is the ratio of correctly predicted positive observations to all observations in actual class. It tells us what proportion of actual positives was identified correctly.

4. F1-Score is the harmonic mean of precision and recall. It tries to find the balance between precision and recall.

The performance metrics for the six models in question are tabulated in Table 6.2. These metrics provide a detailed understanding of each model's prediction capabilities. Model 1 offers balanced performance, with all metrics — accuracy, precision, recall, and F1-score — hovering around the 0.81 mark. This indicates a well-rounded model with consistent results across the board.Model 2 improves on Model 1's performance marginally, with all scores rising to approximately 0.82. This shows that the model is slightly better at both predicting and excluding the correct classes.Model 3 outperforms both Model 1 and Model 2, achieving an accuracy and precision of 0.85 and 0.86 respectively. However, its recall and F1-score are similar to Model 2's, indicating a potential overemphasis on positive predictions. Model 4 demonstrates a slight decrease in performance metrics compared to the first three models, with an accuracy of 0.80 and an F1-score of 0.79. Although its precision is comparable to Model 3's, the lower recall indicates that this model might miss a significant number of positive cases. Model 5 and Model 6 are the least successful in terms of performance metrics, indicating that they might not be as effective as the previous models. With accuracy, precision, recall, and F1-scores falling into the 0.60 range, these models demonstrate a greater level of error in prediction and classifying correct and incorrect instances.

As an overall observation, while the models' performance varies, Models 1 to 4 provide reasonably strong predictive capabilities. However, the drop in performance metrics for Models 5 and 6 suggest the need for additional refinement or a reconsideration of their structure.

Model	**Accuracy**	**Precision**	**Recall**	**F1-score**
Model 1	0.81	0.82	0.81	0.81
Model 2	0.82	0.83	0.82	0.82
Model 3	0.85	0.86	0.84	0.84
Model 4	0.80	0.84	0.78	0.79
Model 5	0.67	0.65	0.67	0.64
Model 6	0.68	0.64	0.68	0.63

Table 6.2: Performance metrics for different models

Wrong predictions

A final visualization worth noting presents the misclassified leaf images from each model. Specifically, Model 1 misclassified 21 images when distinguishing between diseased and healthy states (Figure 12, Appendix B), while Model 2 made errors with 20 images (Figure 13, Appendix B). In the multiclass classification of the 12 species, Model 3 and Model 4 each incorrectly predicted 16 images (Figure 14 and 15, Appendix B). Lastly, both Model 5 and Model 6 made 20

erroneous predictions each in the multiclass setting (Figure 16 and 17, Appendix B).

7 Conclusion and future directions

In conclusion, this report presented a comprehensive evaluation of six convolutional neural network models trained to classify leaf images. The performance of each model was examined using accuracy and loss metrics across training, validation, and test datasets. Graphical interpretation of the results further clarified model performance trends and possible issues related to overfitting and underfitting. Models 1 and 2, tasked with binary classification, demonstrated a fair balance between training and validation accuracy. Although Model 2's high training accuracy may indicate overfitting, it still manages to exhibit the highest overall performance metrics among all models. The multiclass classification models, Model 3 and Model 4, displayed intriguing dynamics with Model 3 showing potential underfitting and Model 4 struggling with certain species despite the introduction of regularization layers. Models 5 and 6, given the more complex task of classifying among 22 complete names, experienced challenges leading to the group's poorest performance metrics.

Visualization of the loss and accuracy trends provided an understanding of each model's learning dynamics. Overfitting was observed in several models, with Model 2 demonstrating severe overfitting despite the implementation of regularization layers. This result emphasizes the need for further exploration of model parameters and additional strategies to improve generalization.When it came to multiclass classifications, the models demonstrated different performance behaviors. Model 3 initially suggested underfitting, followed by overfitting beyond the third epoch. In contrast, Model 4's loss validation curve began to oscillate around the training curve from the 15th epoch onwards, suggesting possible impacts of regularization layers.Furthermore, a detailed evaluation of the confusion matrices highlighted specific misclassifications in all models, revealing that even the top-performing models could benefit from additional tuning and refinement. In terms of precision, recall, accuracy, and F1-score, Models 1 to 4 generally offered robust performance, while Models 5 and 6 were found to be less effective.

In light of the findings presented, our future endeavors will be centered around fine-tuning and refining the models. Our focus will be on addressing the issues of overfitting and underfitting, and enhancing the overall prediction capabilities of these models. For instance, a promising route to enhance model robustness is through data augmentation, which essentially generates a more diverse set of training instances. To counteract overfitting, we could implement a greater range of regularization techniques. An added layer of improvement could be achieved by meticulously fine-tuning the model hyperparameters, and ensuring an equitable distribution of classes in the training data. The incorporation of pre-trained models, leveraging the power of transfer learning, or the use of ensemble methods could significantly elevate overall model performance. Furthermore, exploring alternate model architectures could unearth potential enhancements, paving the way for improved accuracy and precision. These steps form an iterative cycle of refinement and optimization, driving our pursuit of more robust and effective predictive models.

The scalability to larger datasets is another vital aspect that must be considered. With the use of Keras, which is already integral to our project as a high-level neural networks API powered by TensorFlow, it is possible to effectively ensure this scalability. Through batch processing, the management of larger datasets is more effective, by dividing them into more manageable batches. Also, it is possible to leverage the distributed training capabilities of Keras to handle the computational demands of training with larger datasets, thus accelerating the

learning process. The use of the tf.data API, for example, is an efficient way of data pipelining into the model, optimizing memory usage. In addition, to handle larger datasets, there are optimization techniques such as pruning and quantization, along with efficient architectural choices to minimize models' computational footprint. These strategies, powered by Keras, might allow to scale the project to handle larger and more diverse datasets, enhancing the predictive performance and applicability of CNN models.

A Appendix

Figure 3: Healthy plant species

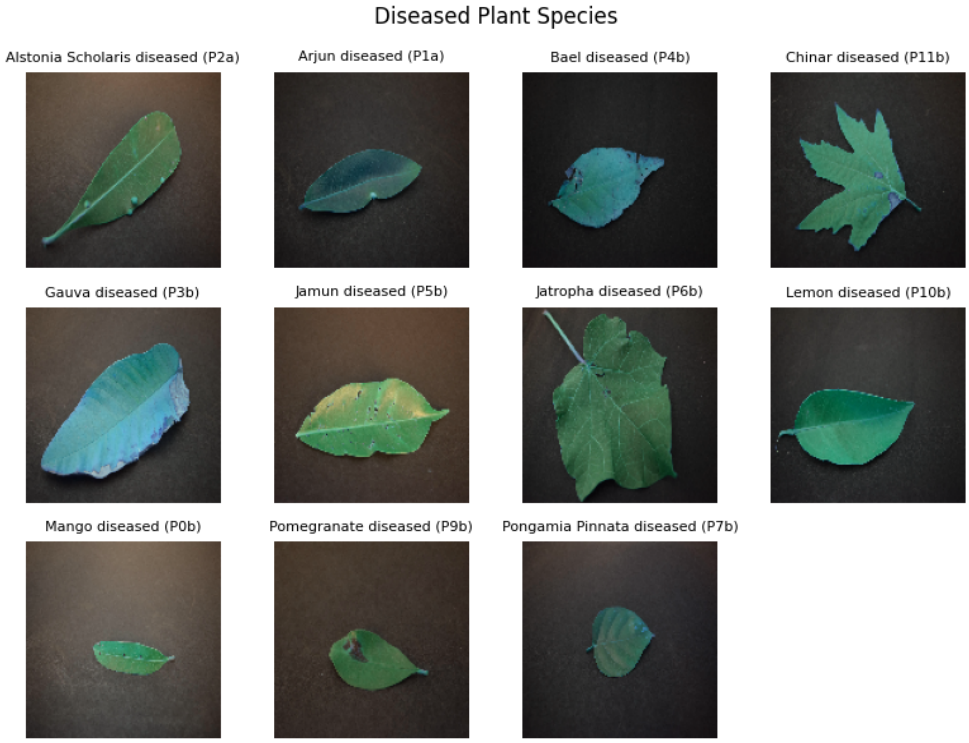

Figure 4: Diseased plants species

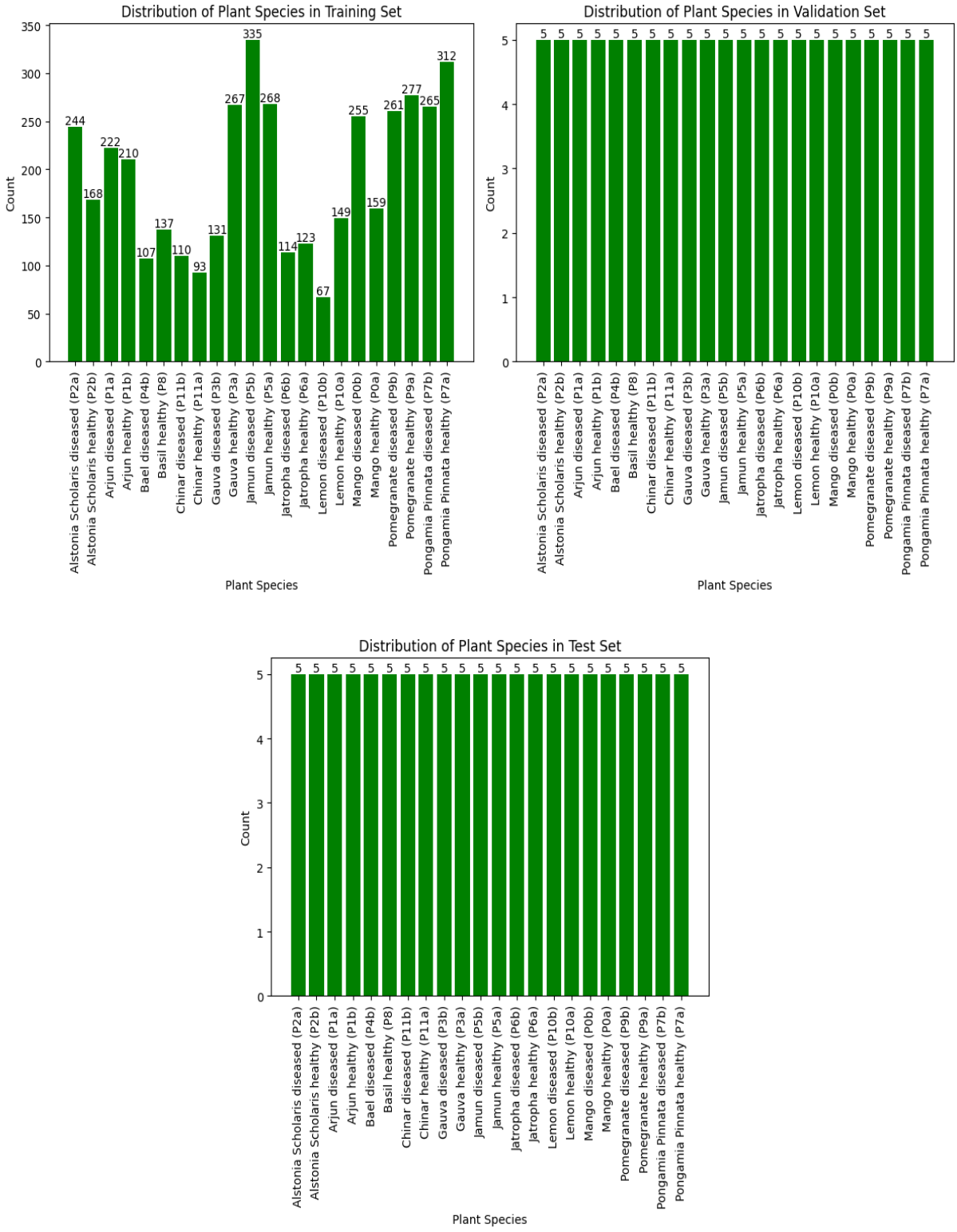

Figure 5: Number of images per species in each folder

B Appendix

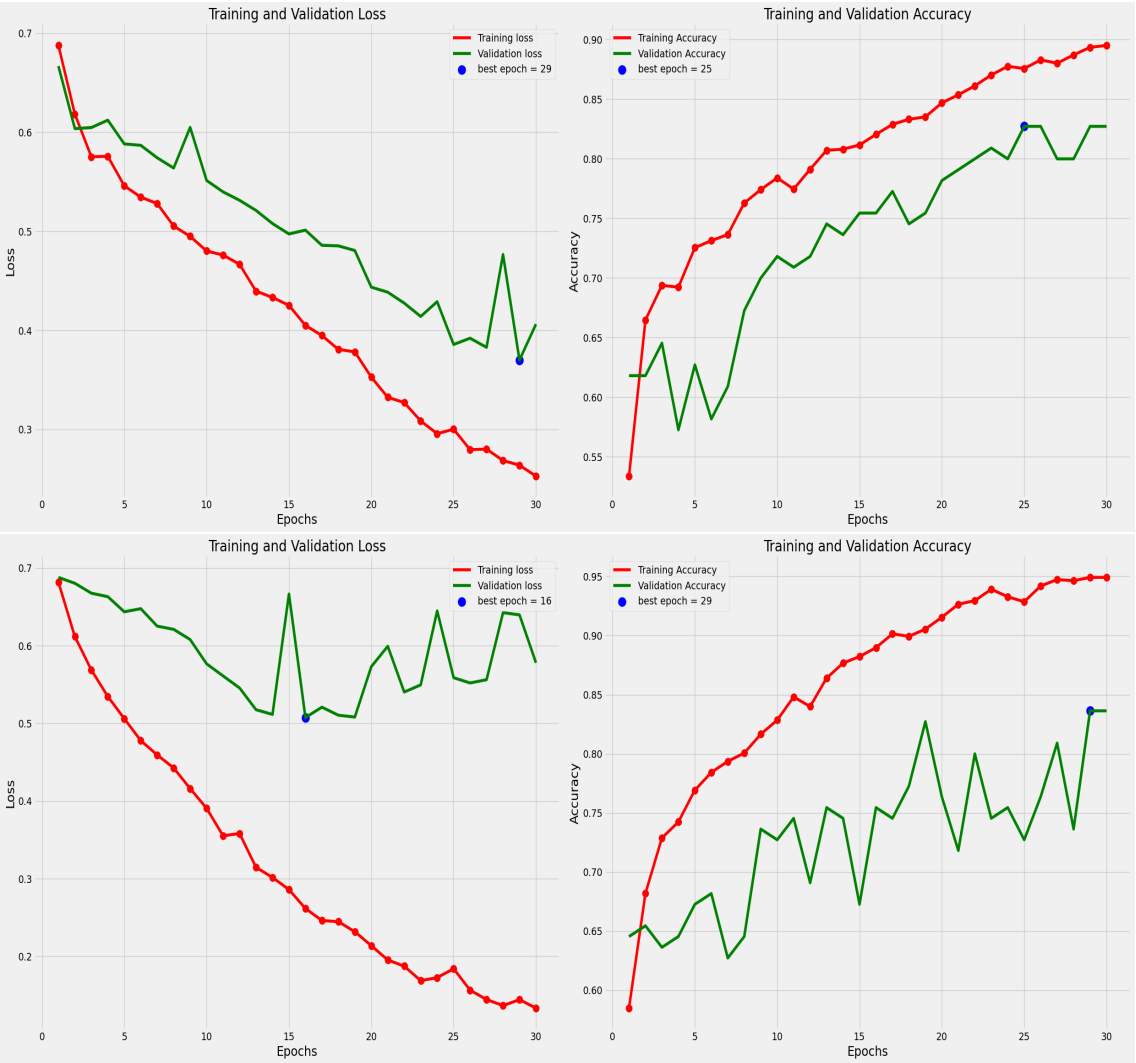

Figure 6: Training-validation loss and accuracy for Model 1 and Model 2

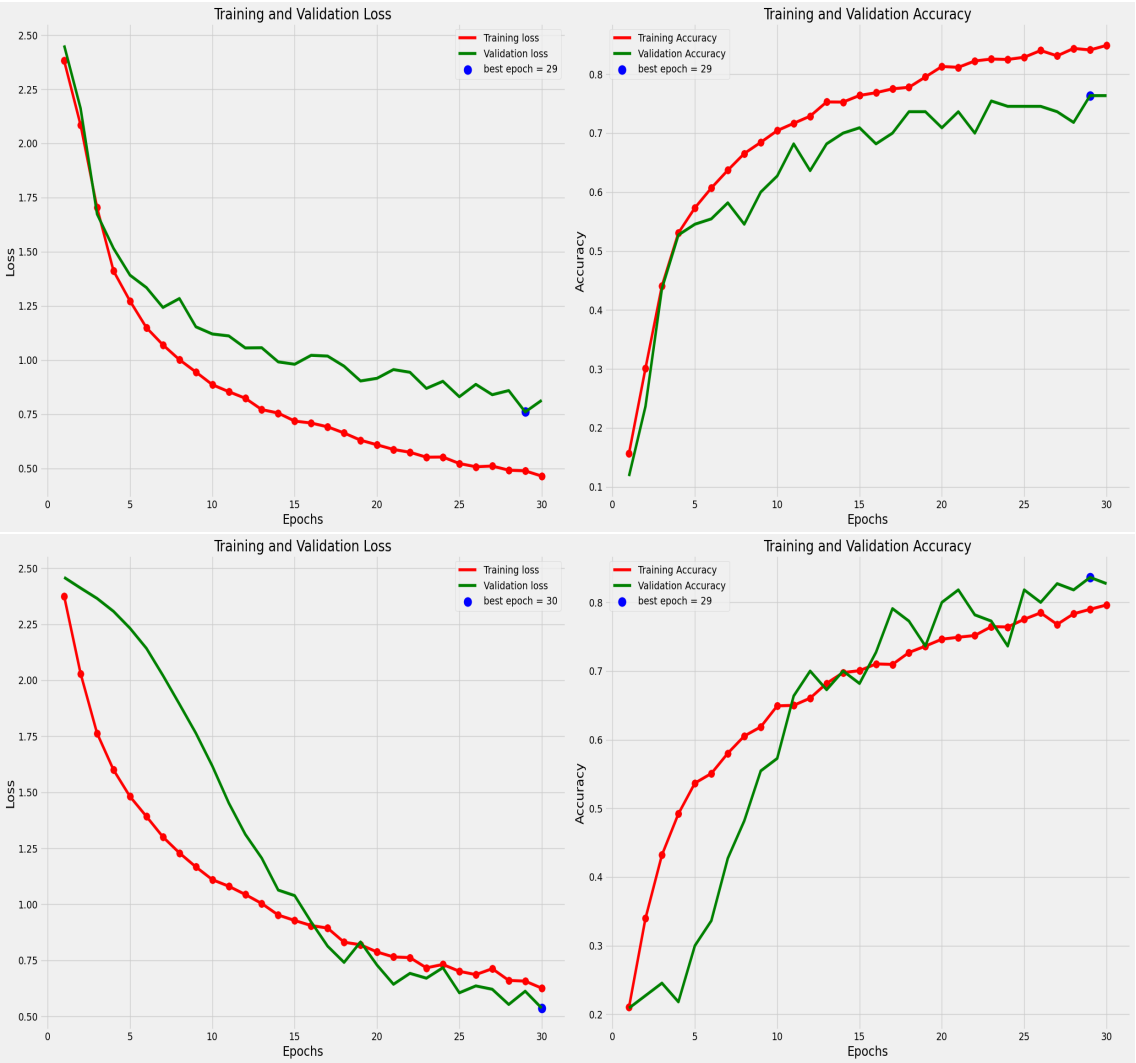

Figure 7: Training-validation loss and accuracy for Model 3 and Model 4

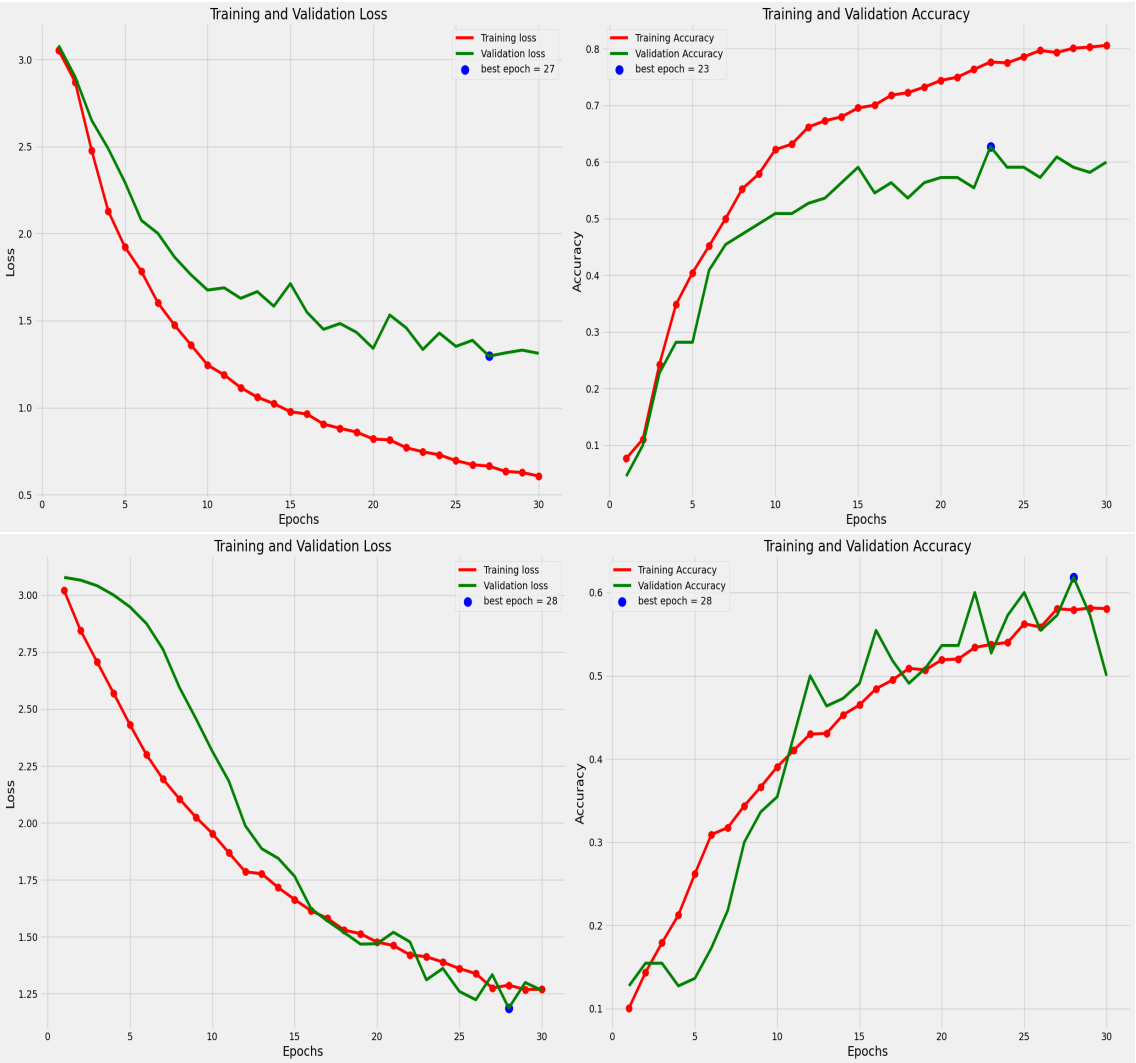

Figure 8: Training-validation loss and accuracy for Model 5 and Model 6

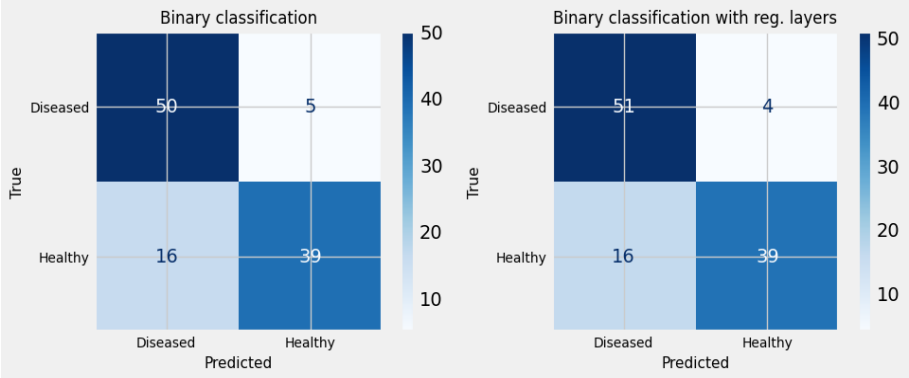

Figure 9: Confusion matrices for Model 1 and Model 2

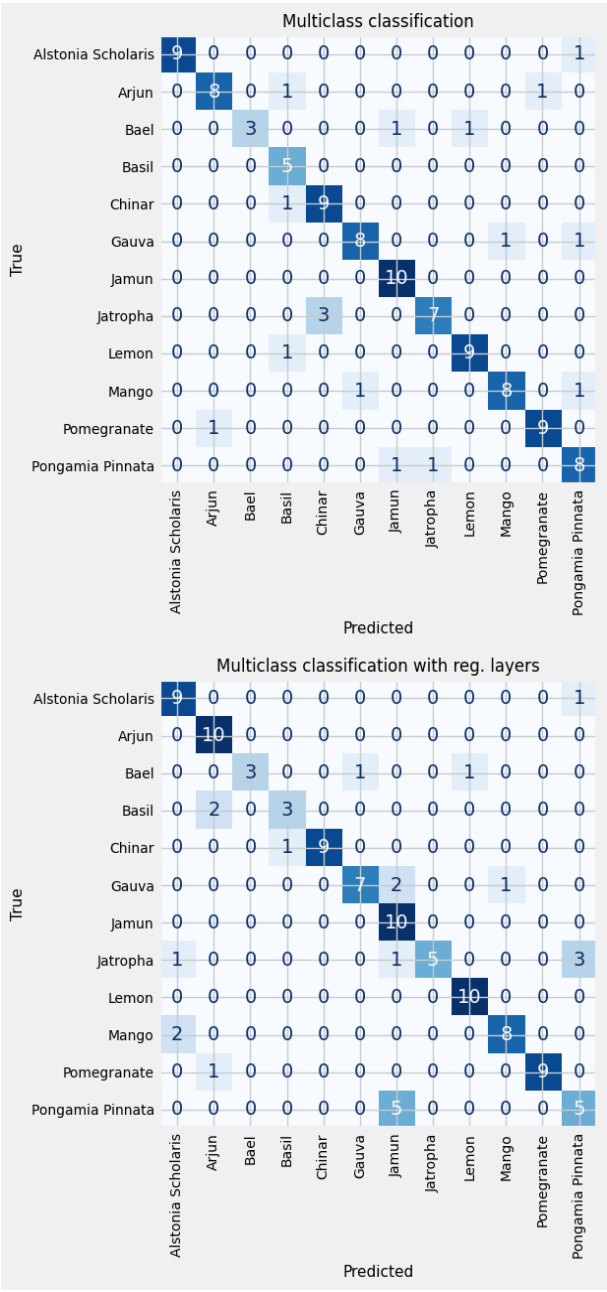

Figure 10: Confusion matrices for Model 3 and Model 4

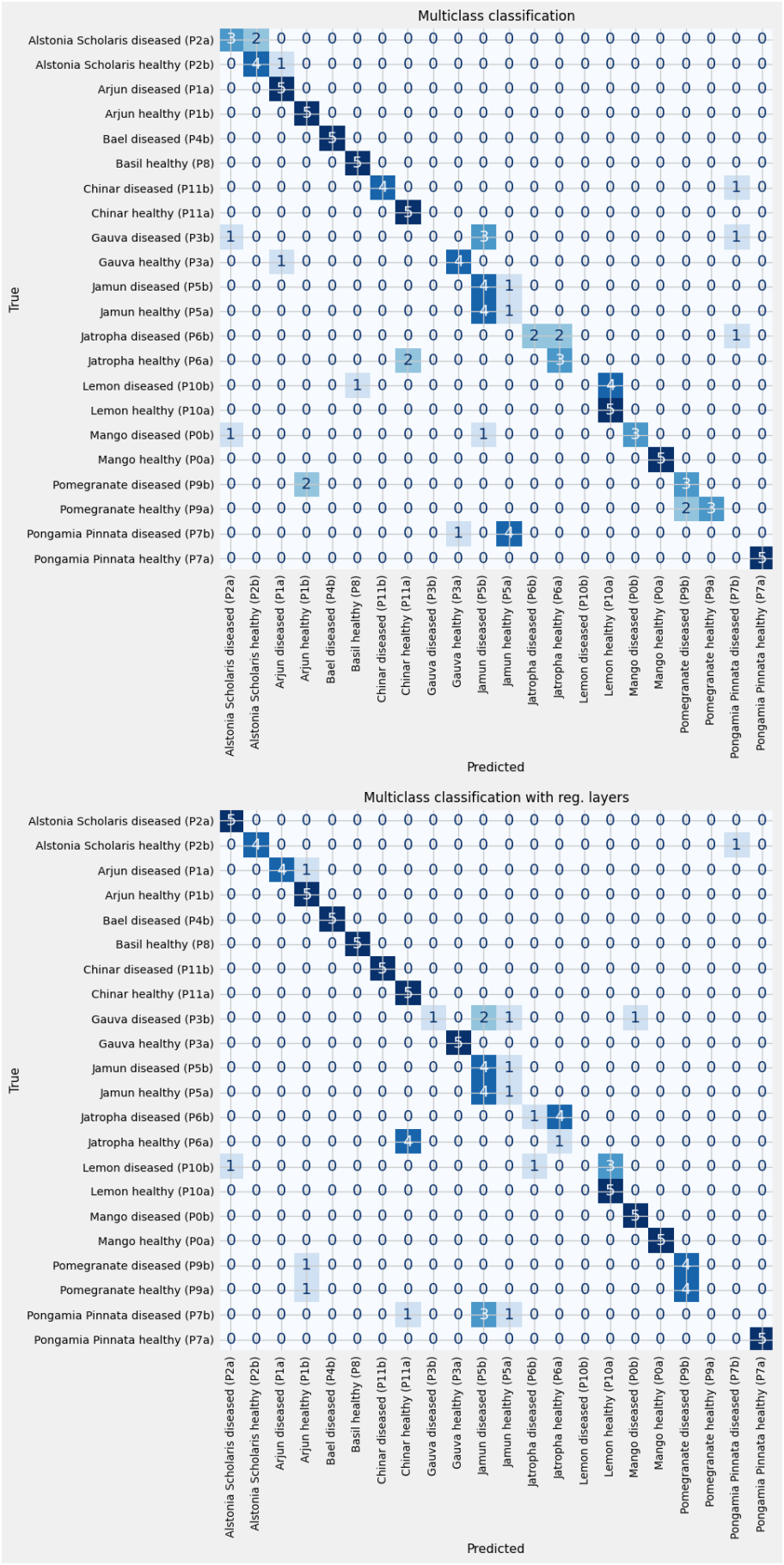

Figure 11: Confusion matrices for Model 5 and Model 6

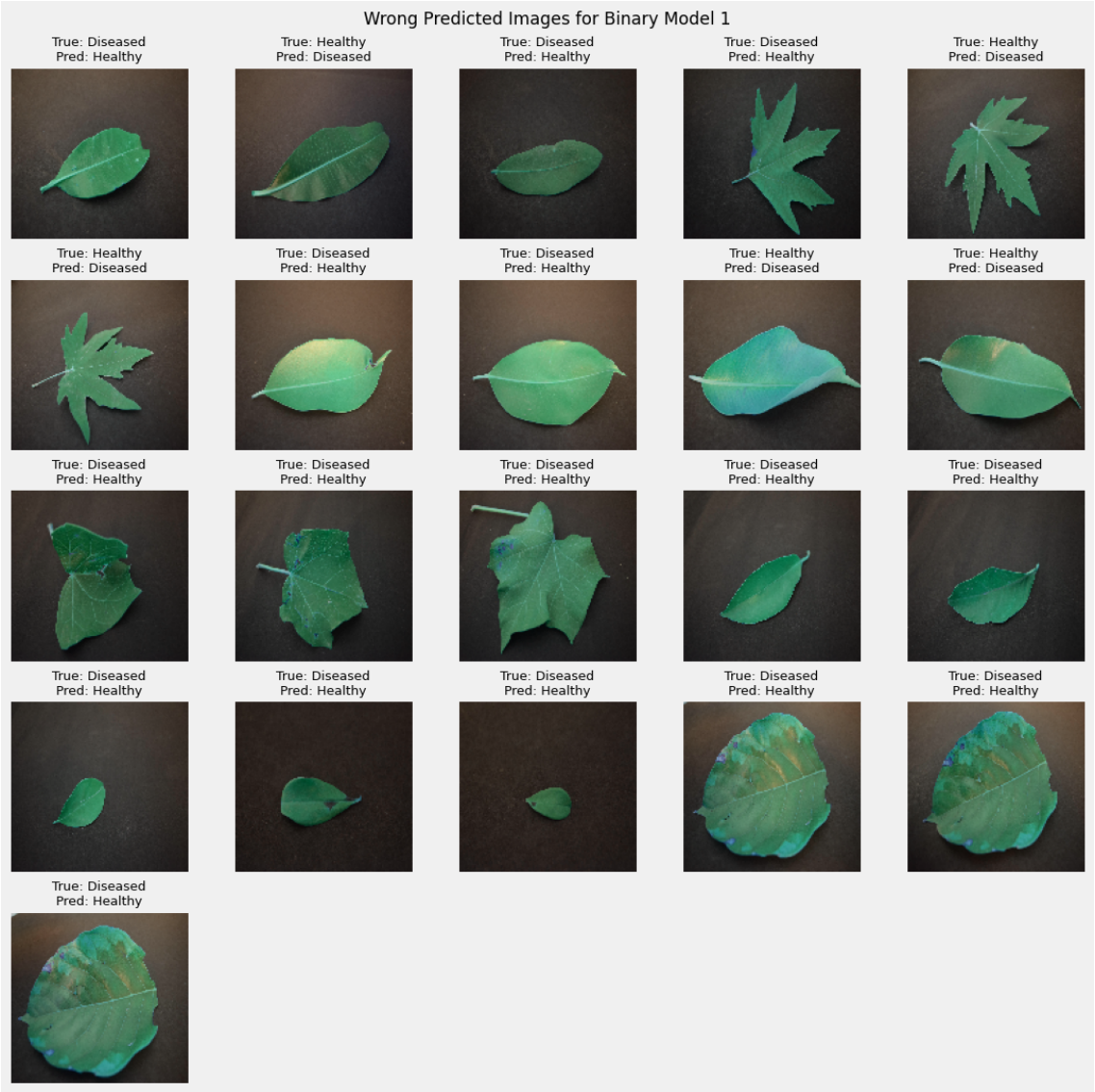

Figure 12: Wrong predictions Model 1

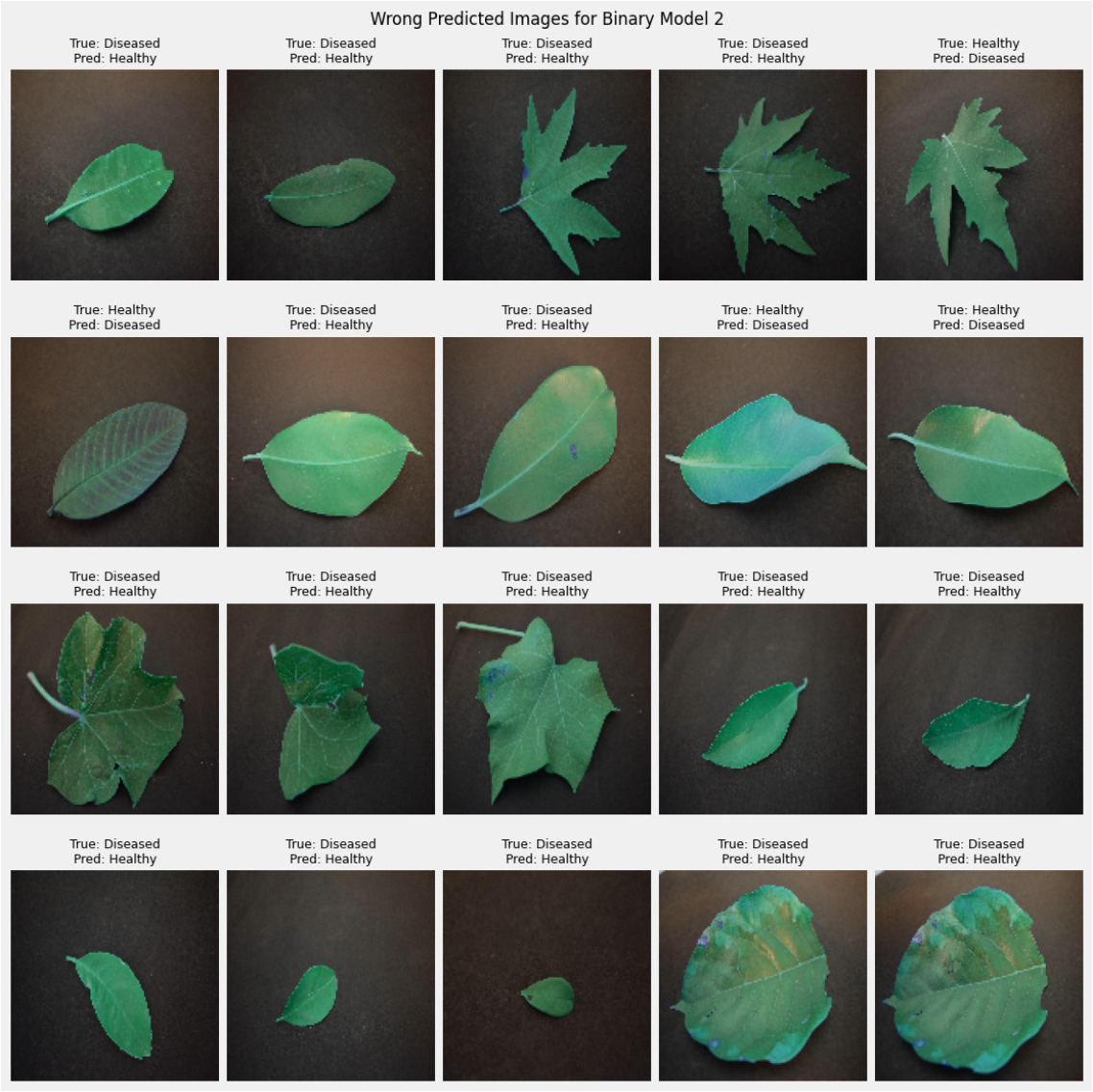

Figure 13: Wrong predictions Model 2

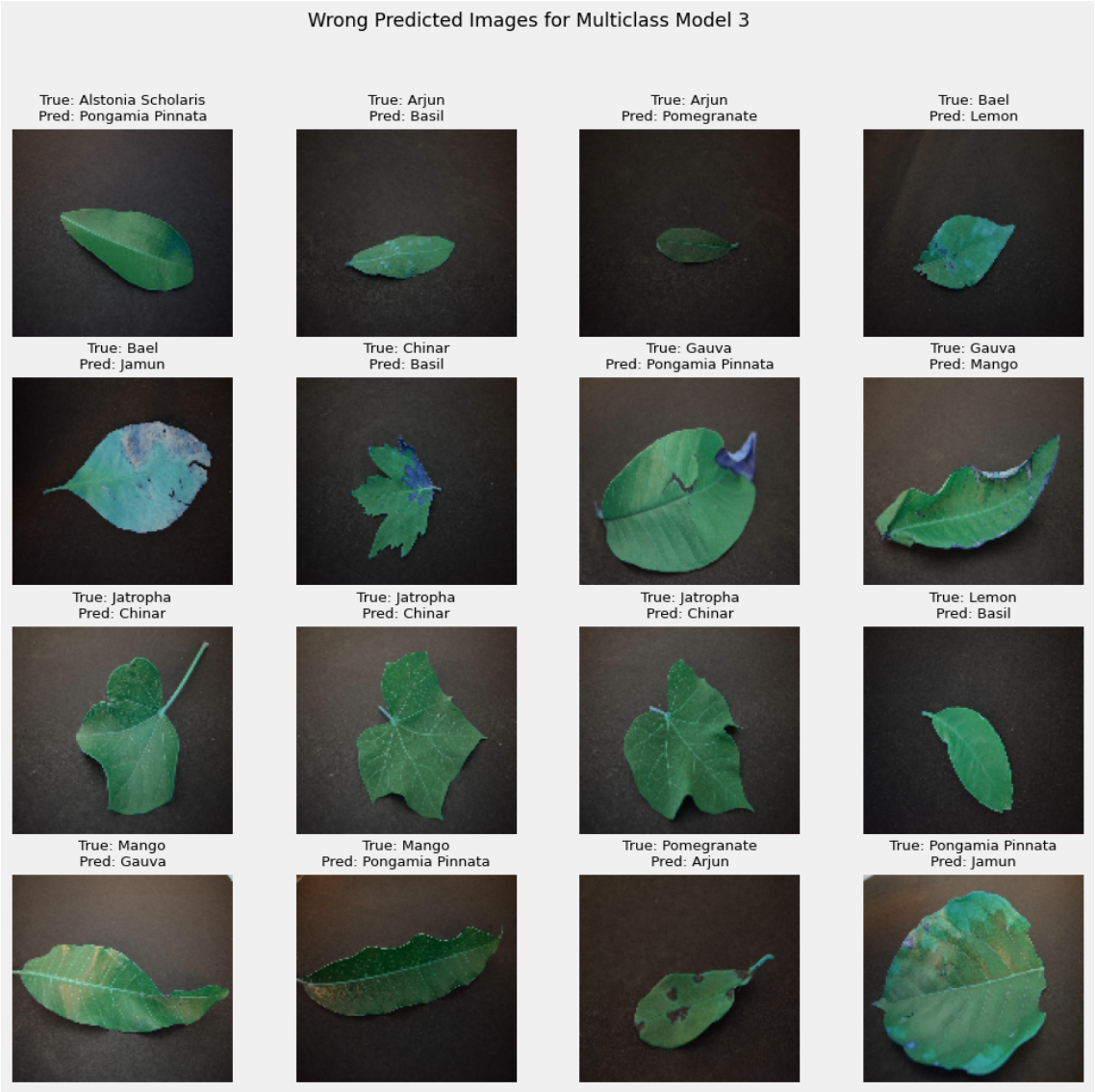

Figure 14: Wrong predictions Model 3

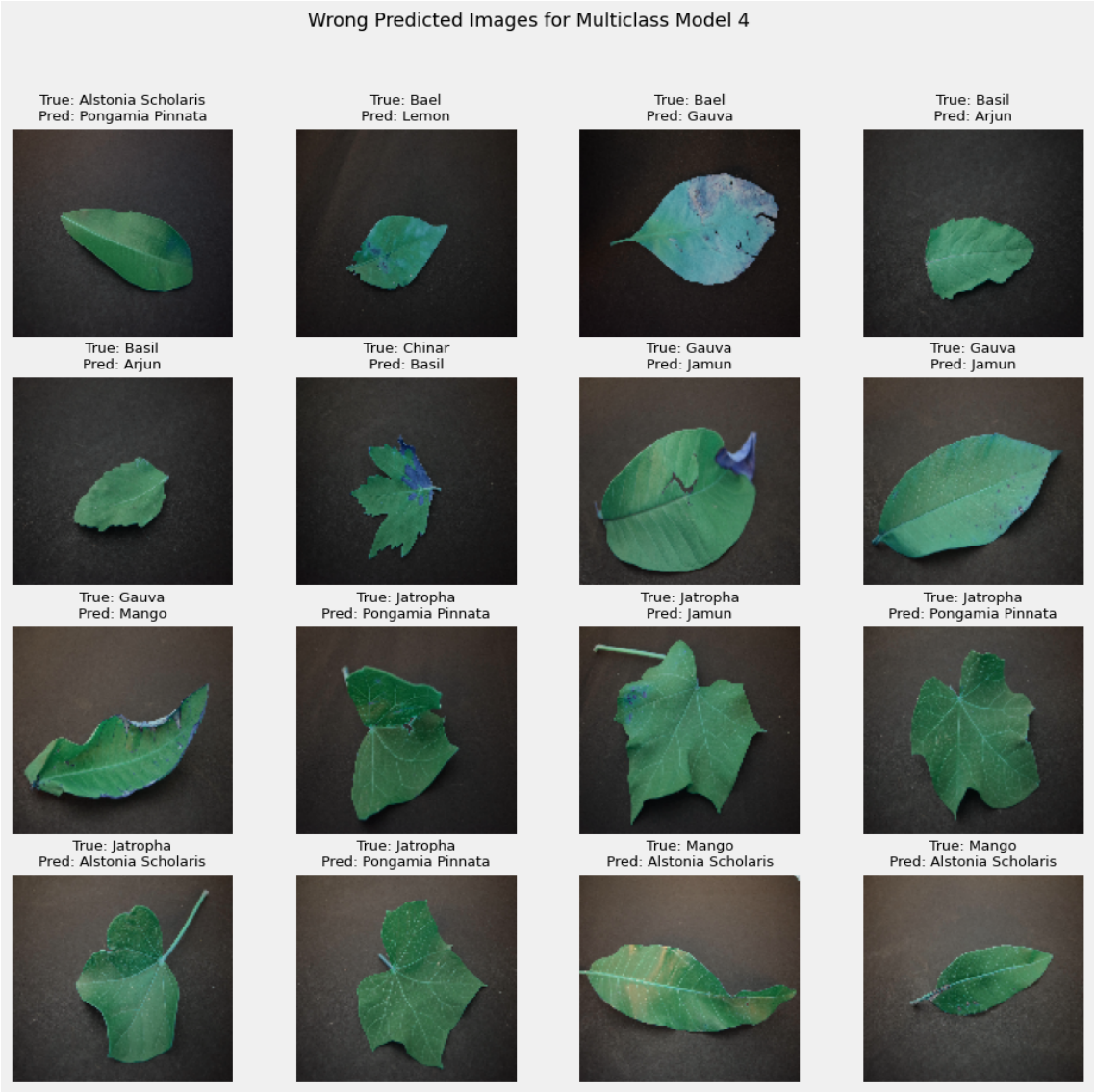

Figure 15: Wrong predictions Model 4

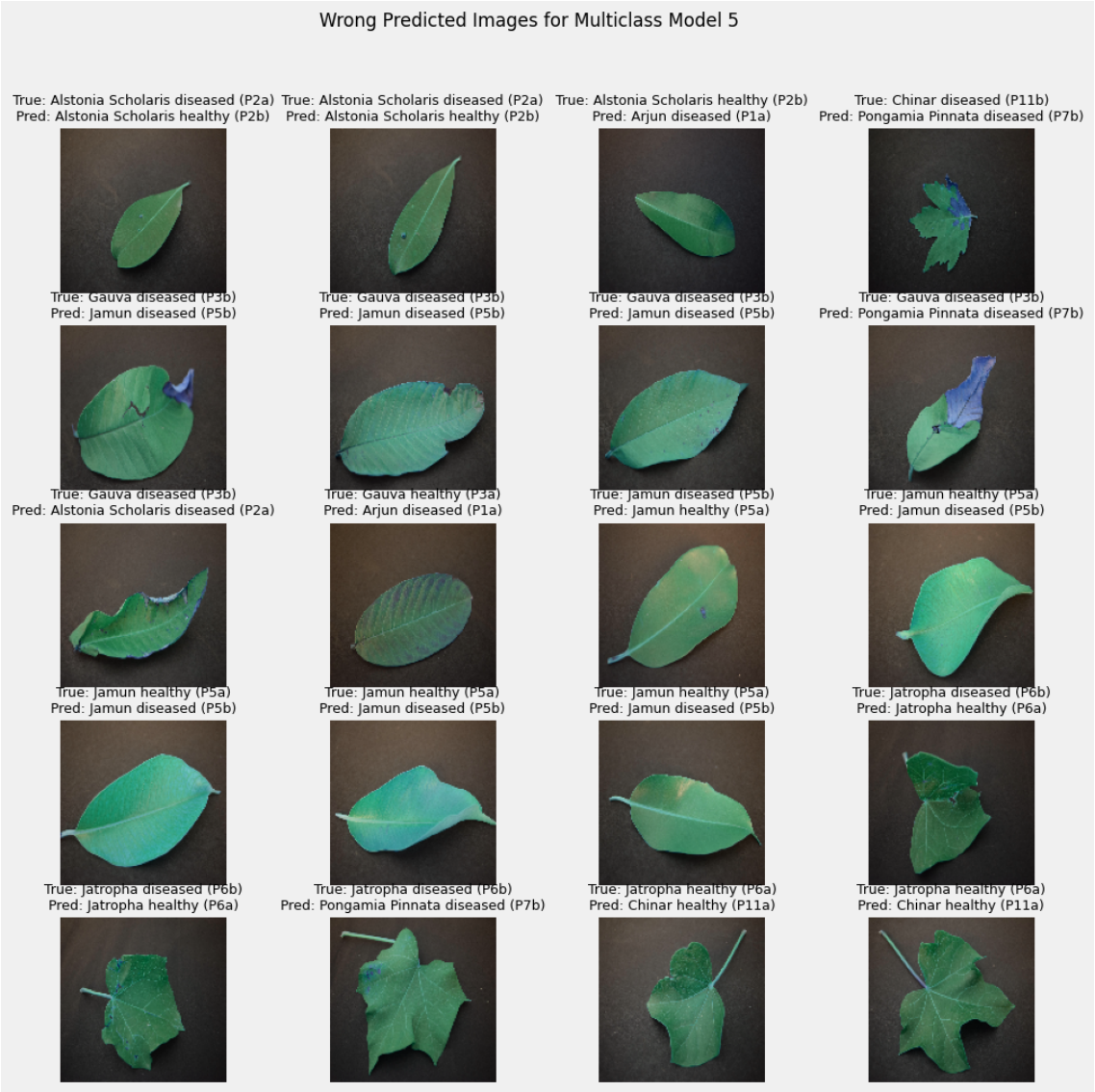

Figure 16: Wrong predictions Model 5

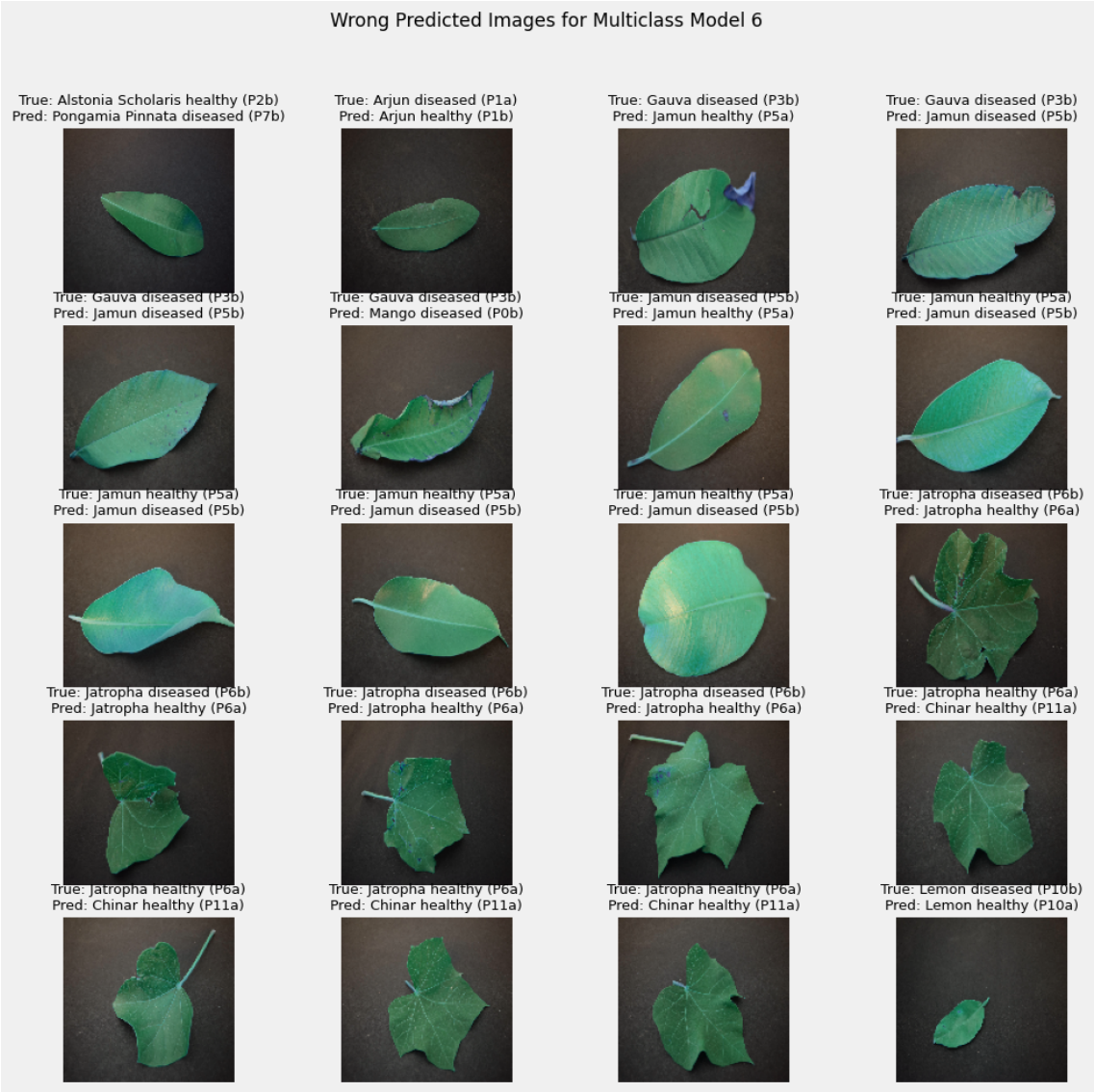

Figure 17: Wrong predictions Model 6