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

## 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 different 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 and the hyperparameter tuning technique for selecting the best combination of hyperparameters. Section 6 provides an analysis of the results along with associated visualisations. The concluding suggestions for improving models performance 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.

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,

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

and regression. Figure 1 shows an example of the general architecture[2].

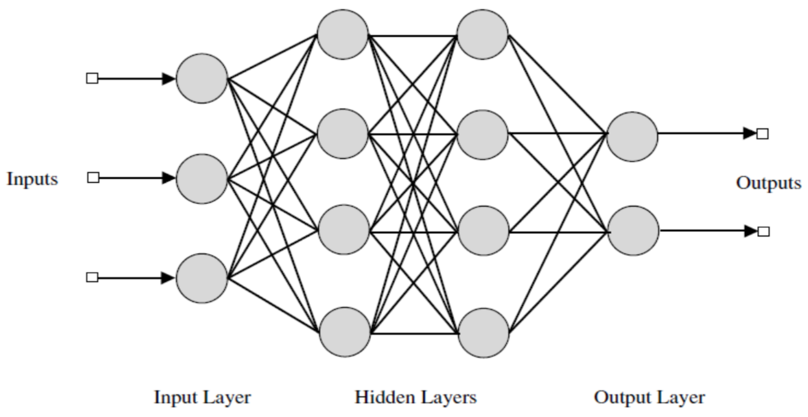


Figure 1: Multilayered Feedforward Neural Network architecture

## 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 section devoted to the classification task.

### 2.2.1 CNN architecture

One of the crucial aspects of the CNN implementation consists of the choice of layers. Below are listed those composing a general CNN.

- **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

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

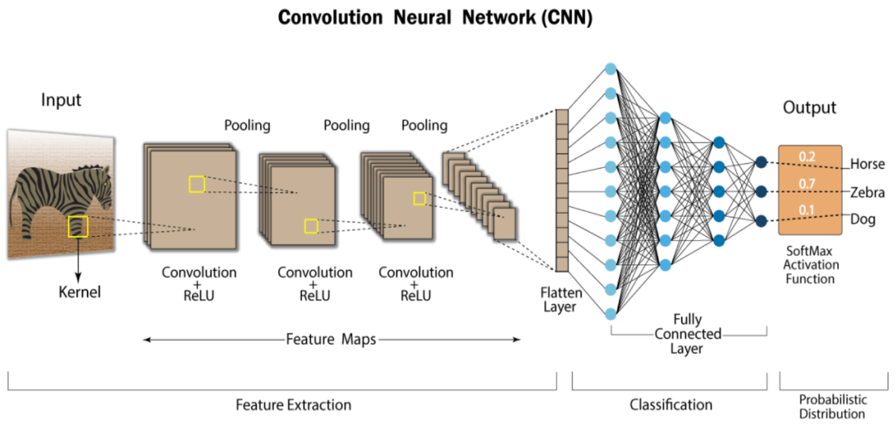


Figure 2: Convolutional Neural Network architecture

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, the output layer typically consists of a single neuron with a Sigmoid activation function. For multlabel classification, instead, Softmax can represent the right choice. 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 and reduce phenomeon such as overfitting.

## 3 Environment and tools

The code for this project was written in Python 3.10.11 on Google Colab, an online environment providing access to free cloud-based Jupyter Notebooks. This platform is ideal for machine learning and data analysis work, offering high computational resources, including powerful GPUs, critical for training ML models. The implementation of the models and the project development primarily utilized the following libraries:

- **OpenCV**: This open-source library, dedicated to Open Source Computer Vision, plays a crucial role in image manipulation and processing during the preprocessing phase.
- **Python Imaging Library - PIL**: Complementing OpenCV, this library supports opening, manipulating, and saving various image file formats.
- **Numpy**: Fundamental for numerical computing in Python, Numpy facilitates efficient operations on large arrays and matrices of numeric data.
- **Tensorflow**: An open-source library frequently employed in Machine Learning, Tensorflow provides a framework for defining, training, and deploying machine learning models. Its suitability for building neural networks makes it essential for deep learning tasks.
- **Matplotlib**: This plotting library allows us to generate a diverse range of static, animated, and interactive plots in Python. It is essential for data visualization and model performance representation.
- **Seaborn**: A Python data visualization library based on Matplotlib, Seaborn assists in enhancing Matplotlib plots aesthetics and in crafting more sophisticated visualizations.
- **Pandas**: Known for its data manipulation and analysis proficiency, Pandas equips us with robust tools to handle, structure, and interpret our data effectively.

#### Keras and scalability

This project prominently harnesses Keras, a high-level neural networks API, which was instrumental in building and training our deep learning models. Keras stands out for its scalability in managing large datasets due to its simplicity and modularity, which enables swift model design and experimentation. Its comprehensive utilities for data loading and preprocessing assist in the efficient management of our data. The built-in support for convolutional networks makes Keras an ideal choice for tasks such as our leaf image classification. More importantly, Keras was employed to perform GridSearchCV, a method that allows us to systematically work through multiple combinations of hyperparameters to determine the optimal values for our model.

## 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 leaves and 2225 images of the diseased leaves. 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:

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

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

Each folder in the dataset was already divided into 22 subfolders, with each subfolder containing images named after the individual species and their respective health status. For example, there were subfolders like 'Basil healthy (P8),' 'Mango healthy (P0a),' and 'Mango diseased (P0b).' In Appendix A, you can find the collection of pictures depicting healthy species in Figure 3, and diseased species in Figure 4.

To gain insights into the dataset's distribution, we analyzed the number of images based on their health status (healthy or diseased), individual species, and complete labels. The details are summarized in Table A.1. It becomes evident that the dataset is well balanced in terms of health status across all sets, except for a slight difference observed in the training set. However, a notable imbalance arises when considering the 12 individual species within the training set. Some species have a limited representation with only 107 images, while others have a significantly larger representation with nearly 600 images. Moreover, the training set also exhibits variations in the number of occurrences for the specific 22 complete labels. The presence of such an imbalance in the training set can pose challenges during model training. It could lead to biases and difficulties in accurately predicting the less represented species or complete labels. One common approach to tackle this issue is to assign different weights to the classes during the loss calculation based on their representation. The weights' assignment is explained in Paragraph 5.5.

## 4.1 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 and performance metrics

## 5.1 CNNs Models

In this project, the focus is on training CNN models for image classification. Specifically, three types of classification tasks were explored:

i) **Binary Classification**: the first CNN model is designed to classify images into two categories: healthy or diseased leaves. The model architecture consists of multiple layers for extracting features from input images, with the final layer using the Sigmoid activation function. The input images are resized to (128x128) pixels and have 3 color channels. Prior to training, the two labels (”Healthy” and ”Diseased”) were one-hot encoded for categorical representation.

ii) **12 Species Classification**: the second CNN model follows a similar structure to the previous one but aims to predict the correct plant species among the 12 available options (e.g., Mango, Basil, Pongamia Pinnata, Guava, etc.). The input images are again resized to (128x128) pixels with 3 color channels. The final layer uses the ReLU activation function. Before training, all 12 labels were one-hot encoded to represent the multi-class classification problem.

iii) **22 Leaves Classification**: the third type of classification also uses the same CNN architecture but focuses on predicting the complete label among the 22 provided in the original dataset. These labels contain both the plant name and the state of health (e.g., Pomegranate diseased (P9b), Mango diseased (P0b), etc.). The input images follow the same size and channel specifications as the previous models. The final layer uses the ReLU activation function. Similar to the previous tasks, the 22 labels were one-hot encoded before training.

For what concerns the CNNs architecture, it is the same for all three classifications, except for the final output layer. Here the description of the layers:

```
model = Sequential([
    Conv2D(8, (3,3), input_shape=(128, 128, 3), activation="relu"),
    MaxPooling2D((3,3)),
    Conv2D(16, (3,3), activation="relu"),
    MaxPooling2D((3,3)),
    Conv2D(32, (3,3), activation="relu"),
    MaxPooling2D((3,3)),
    Flatten(),
    Dense(32, activation="relu"),
    Dense(<number_of_classes>, activation="<activation_function>")
])
```

1. `Conv2D(8, (3,3), input_shape=(128, 128, 3), activation='relu')`: This is the first convolutional layer with 8 filters, each of size 3x3. It takes an input image of size 128x128 with 3 channels (RGB) and applies the ReLU activation function to introduce non-linearity.

2. `MaxPooling2D((3,3))`: This is a max pooling layer with a pool size of 3x3. It reduces the spatial dimensions of the previous layer's output by taking the maximum value within each 3x3 window.

3. `Conv2D(16, (3,3), activation='relu')`: This is the second convolutional layer with 16 filters of size 3x3. It applies the ReLU activation function.

4. `MaxPooling2D((3,3))`: Another max pooling layer with a pool size of 3x3.

5. `Conv2D(32, (3,3), activation='relu')`: The third convolutional layer with 32 filters of size 3x3. It also uses the ReLU activation function.
6. `MaxPooling2D((3,3))`: Another max pooling layer with a pool size of 3x3.
7. `Flatten()`: This layer flattens the 3D output of the previous layer into a 1D vector, preparing it for the fully connected layers.
8. `Dense(32, activation='relu')`: A fully connected layer with 32 units, using the ReLU activation function.
9. `Dense(<number_of_classes>, activation="<activation_function>")`: The final fully connected layer has 2, 12, or 22 number of classes, according to the classification task. It uses the Sigmoid activation function for binary classification and the Softmax activation function to produce class probabilities for multi-class classification.

## 5.2 Hyperparameter tuning

Hyperparameter tuning is the process of finding the optimal values for the hyperparameters of a machine learning model. Hyperparameters are configuration settings that are not learned from the data, but set by the user before training the model. They determine the behavior and performance of the model, such as the learning rate, the number of hidden layers in a neural network, or the regularization parameter. The process of hyperparameter tuning involves systematically searching through the hyperparameter space to find the combination of values that yields the best model performance.

### Grid Search Cross-Validation

In this project the implemented technique for performing hyperparameter tuning is the **Grid Search Cross Validation**. It systematically searches through a specified hyperparameter grid and evaluates each combination using cross-validation. In grid search cross-validation, a grid of hyperparameter values is defined, specifying different values or ranges for each hyperparameter of interest. The grid search algorithm then exhaustively searches through this grid, evaluating the model's performance for each combination of hyperparameters.

Cross-validation is then used to assess the model's performance on unseen data and mitigate the risk of overfitting. During cross-validation, the dataset is divided into k subsets or folds (in this case k=3). The model is trained and evaluated three times, each time using a different combination of training and validation data. For each combination, the scoring metric (i.e., accuracy) is computed based on the model's performance on the validation data. The average of these scoring metric values across all the folds is then used to assess the overall performance of the model.

In this setting, the hyperparameter grid was composed of:

- Epochs: 30, 50
- Mini-batch size: 32, 64
- Optimizer: Adam, Stochastic gradient descent

Since I opted for a **3-fold cross-validation**, each fold tested 8 combinations of hyperparameters, resulting in a total of 24 combinations tested across all 3 folds.

## 5.3 Loss functions

### Binary cross-entropy

Binary Cross-Entropy is the loss function used in the binary classification task (Healthy or Diseased). It quantifies 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 the loss function used in the multi-class classification tasks of this project. It quantifies 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 $c$.

## 5.4 Regularization techniques

After having obtained the three best models through GridSearchCV, some regularization techniques were applied for improving the overall models' performance and reduce overfitting that emerged from the plots (the comments regarding this issue are in the section below). To tackle this issue, Dropout and BatchNormalization layers were added to the models architecture.

- The **Dropout** layer helps preventing overfitting by randomly dropping out a fraction of the input units during training, which helps to reduce the reliance of the model on specific features and encourages the learning of more robist representations. In particular, the Dropout was set at 0.5, which means that during training, each input unit will be retained with a probability of 0.5 and dropped out with a probability of 0.5.
- The **BarchNormalization** layer normalizes the activations of the previous layers, which helps in stabilizing the learning process and makes the model more resilient to changes in input distribution.

Also in this case, the regularized model presents the same architecture for all the three classification tasks, except for the final fully connected layer. As before, it assumes 2, 12 or 22 as units depending on the number of the predicted labels, and it uses Sigmoid for binary classification and Softmax for multiclass classification.

```
model = Sequential([
    Conv2D(8, (3,3), input_shape=(128, 128, 3), activation="relu"),
    MaxPooling2D((3,3)),
    Conv2D(16, (3,3), activation="relu"),
    MaxPooling2D((3,3)),
    Conv2D(32, (3,3), activation="relu"),
    MaxPooling2D((3,3)),
    Flatten(),
    Dense(32, activation="relu"),
    Dropout(0.5),
    BatchNormalization(),
    Dense(<number_of_classes>, activation="<activation_function>")
])
```

### 5.5 Inverse class frequency: assigning class weights

As previously said, the imbalance in the number of images for each species, especially in the training set, could affect the performance of the models. After having added the regularization layers, higher weights are assigned to the minority classes in the 12 species classification and in the 22 complete names models. Specifically, the weights are computed by taking the total number of samples divided by the product of the number of unique labels and the count of each label. In this way, the weight for each class is proportional to the inverse of its occurrence in the dataset. Weights distribution across categories can be seen in Figure 5 and Figure 6.

### 5.5. Performance metrics and confusion matrix

Together with the canonical computation of accuracy and loss values, the following metrics were considered for evaluating the models' performance in a more complete perspective: Precision, Recall, F1-score.

- **Precision**. For the binary classification, precision is the ratio of correctly predicted positive (diseased) observations to the total predicted positives. It gives information about how many of the identified positive instances are actually positive.

- **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.

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

Furthermore, for each model a **Confusion matrix** is plotted. 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.

## 6 Results and visualization

### Best Models - GridSearchCv

According to the Grid Search Cross Validation method, in each of the three classification tasks (Binary classification, 12 species classification, 22 leaves names classification), the hyperparameter combination that results in the best scoring metric (i.e., accuracy) is the following:

- Epochs: 50
- Mini-batch size: 32
- Optimizer: Adam

**Best Binary model**:
```
Best score for 0.962330957253774 using batch_size: 32, epochs: 50, optimizer: adam}
```

**Best 12 species model**:
```
Best score for 0.3294753432273865 using batch_size: 32, epochs: 50, optimizer: adam}
```

**Best 22 leaves names model**:
```
Best Parameters: batch_size: 32, epochs: 50, optimizer: adam Best Score: 0.3236040423313777}
```

### Models accuracy and loss

The three best models were then tested on the respective test sets and the results in terms of accuracy and loss are reported in Table 6.3. In a second phase, the models were enriched with regularization layers and labels weights, and tested on test data. The results for these augmented models are reported in Table 6.2.

By looking at the performance in terms of accuracy and loss (Table 6.3), it emerges that the binary classification model achieved the highest accuracy value during the training phase (0.9983) and lower during validation (0.8999): When using unseen data, however, the test accuracy increased again (0.9454) suggesting a good capability of predicting new images. The 12 species classification model also performed well, with relatively high accuracy in training (0.9775). However, both the validation (0.8454) and the test accuracy (0.8273) decreased. These results indicate still a quite good capability of the model in classifying never seen images according to the 12 species, but with less accuracy with respect to the training phase. Finally, the 22 leaves classification model showed a good training accuracy (0.9656), but a drastic lower accuracy values in the validation (0.7181) and the test (0.7909). Furthermore, the loss dramatically increases comparing the train line (0.1145) and the test one (1.9495). It is evident that in this case the model performs worse when working on unseen data and, also, the distance between the train and validation value suggests a problem of overfitting.

Table 6.1: Best models accuracy and loss

| **Model** | **Train Accuracy**<br>**Train Loss** | **Valid Accuracy**<br>**Valid Loss** | **Test Accuracy**<br>**Test Loss** |
|---|---|---|---|
| Binary Classification | 0.9983<br>0.0112 | 0.8999<br>0.5420 | 0.9454<br>0.3772 |
| 12 Species Classification | 0.9775<br>0.0703 | 0.8454<br>0.6017 | 0.8273<br>0.9083 |
| 22 Leaves Classification | 0.9656<br>0.1145 | 0.7181<br>1.6962 | 0.7909<br>1.9495 |

When adding the regularization layers and the labels weights, the accuracy and loss metrics present some interesting changes (Table 6.2). In the binary classification case, where labels weights were not introduced, the train accuracy has slightly decreased with respect to the previous case (from 0.9983 to 0.9382), and both validation and test accuracy doesn't overcome the 90% threshold (respectively 0.8554 and 0.8818). In the 12 species classification case, where the labels' weights were added, we see poor prediction accuracy in the training phase (0.6359), but an increase when considering validation (0.7356) and test accuracy (0.8909). This gap might suggest a mild underfitting, which occurs when validation accuracy is higher than training one. In general, this model seems to perform better on test data with respect to the previous case without regularization layers and weights, but definitely worse on training data.For what concerns the 22 leaves classification, the train and validation accuracy are very poor (respectively 0.6105 and 0.6325). The test accuracy, instead, is higher (0.7727) but still indicating a poor predicting capability.

Table 6.2: Accuracy and loss of models with regularization layers and weights

| **Model** | **Train Accuracy**<br>**Train Loss** | **Valid Accuracy**<br>**Valid Loss** | **Test Accuracy**<br>**Test Loss** |
|---|---|---|---|
| Binary Classification<br>(reg. layers) | 0.9382<br>0.1469 | 0.8554<br>0.5143 | 0.8818<br>0.5416 |
| 12 Species Classification<br>(reg. layers + weights) | 0.6359<br>1.0399 | 0.7356<br>0.8538 | 0.8909<br>0.3215 |
| 22 Leaves Classification<br>(reg. layers + weights) | 0.6105<br>1.1832 | 0.6325<br>1.2020 | 0.7727<br>0.7746 |

### Graphs interpretation

By looking at the accuracy and loss curves we can understand whether the phenomenons of overfitting or underfitting occurred during the training and validation phases of the six inspected models. The binary model curves (Figure 7) show that the model learns effectively from the training data, as evidenced by the increasing accuracy and decreasing loss as the number of epochs increases. However, the validation curves remain consistently below the training curves, indicating that the model's performance on the validation data is not as good as on the training data. This discrepancy between training and validation performance, apparent in both the accuracy and loss graphs, strongly indicates the presence of overfitting. Despite the addition of regularization layers, this overfitting issue is still prominent, suggesting that further strategies might be necessary to mitigate it.

Turning our attention to the 'best model' for classifying the 12 species (Figure 8), the degree of overfitting is less severe. There is still a noticeable gap between the training and validation curves, but it is less pronounced than in the binary model. The incorporation of regularization layers and label weights appears to have a significant effect. As evident in the second row of graphs, the validation accuracy curve remains above the training curve throughout all epochs, and the validation loss steadily decreases. These observations suggest that the use of regularization and label weights has improved the model's generalization performance on unseen data, but also indicate a slight underfitting.

In the final set of models, which classify the 22 complete names (Figure 9), the 'best model' (depicted in the first row of graphs) exhibits a significant issue with overfitting. The training accuracy steadily increases across all epochs, while the validation accuracy remains notably lower, signifying a subpar performance on unseen data. Likewise, the loss graphs reveal a similar trend, with the training and validation curves staying apart, further indicating the model's struggle to learn effectively from the validation data. However, with the incorporation of regularization layers and label weights, notable changes in the model's performance are observed (illustrated in the second row of graphs). Now, the validation accuracy curve tends to align more closely with the training curve, oscillating around it. This behavior is mirrored in the loss plot, where the validation and training curves are closer together. These modifications in the curves suggest that the introduction of regularization and label weights has successfully mitigated the overfitting issue, enhancing the model's ability to generalize from the training data to unseen validation data.

## Performance metrics

Together with test accuracy and loss, the performance metrics of Precision, Recall and F1-score were computed for each model. Table 6.3 shows that the binary classification model achieved the highest scores across all metrics compared to the other models in their original configuration, with a precision, recall, and F1-score of 0.95. This suggests that the binary model was the most effective at correctly classifying the data into the two respective classes and minimizing both false positives and false negatives. The 12-species and 22-leaves classification models scored lower across all metrics, with the 22-leaves model scoring the lowest. This suggests a greater difficulty in correctly classifying data into multiple classes, as the precision, recall, and F1-score for the 22-leaves model were 0.81, 0.79, and 0.78 respectively.

In Table 6.4, after the addition of regularization layers, the binary model's performance decreased slightly, with precision, recall, and F1-score all dropping to 0.84. This suggests that the additional layers may have introduced some complexity that slightly hindered the model's performance. Conversely, the 12-species model's performance improved across all metrics, achieving a precision, recall, and F1-score of 0.88. This indicates that the additions of layers and labels' weights were beneficial in increasing the model's ability to handle multi-class data. However, the 22-leaves model's performance significantly decreased, with all metrics dropping to 0.69, implying that the additional layers and weights were not beneficial for this model.

Table 6.3: Best models performance metrics (macro average)

| **Model** | **Precision** | **Recall** | **F1-score** |
|---|---|---|---|
| Binary classification | 0.95 | 0.95 | 0.95 |
| 12 Species classification | 0.85 | 0.83 | 0.83 |
| 22 Leaves classification | 0.81 | 0.79 | 0.78 |

Table 6.4: Models with reg. layers and weights performance metrics (macro average)

| **Model** | **Precision** | **Recall** | **F1-score** |
|---|---|---|---|
| Binary classification | 0.84 | 0.84 | 0.84 |
| 12 Species classification | 0.88 | 0.88 | 0.88 |
| 22 Leaves classification | 0.69 | 0.69 | 0.69 |

Appendix B contains precision, recall, f1-score metrics for each label. Figure 10 refers to binary classification; Figure 11 to 12 species classification; Figure 12 to 22 leaves names classification.

## Confusion matrix

From the confusion matrices corresponding to the three models, we gain comprehensive insight into the count of false positives and false negatives for each classification task. Starting with the binary classification models, as displayed in Figure 13, it can be observed that out of 55 healthy leaves, 53 were accurately predicted, thereby yielding a high true positive rate. Similarly, for the diseased images, only 51 out of 55 were correctly classified, suggesting room for improving the model's sensitivity. The number of correct predicitions, instead, decreased once adding the regularization layers. From the confusion matrix on the right is in fact evident that 48 healthy leaves and 49 diseased images were correctly predicted out of 55 respectively.

In the 12-species classification task (refer to Figure 14), the species of Arjun, Basil and Jamun are perfectly predicted by the 'best model' (on the left). After the addition of the regularization layers and the weights, however, the number of perfectly predicted species increases by one (Arjun, Bael, Basil, Chinar) (on the right). Overall, it can be seen that the 12 species classification models made less wrong predictions than the original one.

Lastly, in the comprehensive 22 leaves name classification task (see Figure 15), the model's performance varied significantly. The 'best model' predicted perfectly 12 leaves names out of 22, whereas the regularized model only 10 out of 22. In fact, the metrics confirms that the accuracy decreased from 81% to 77% in this task after adding layers and labels' weights.

## 7 Conclusion

The use of grid search cross-validation proved to be a crucial strategy in handling the complexity of hyperparameters tuning. By automating the search process, it allowed for the identification of the best-performing models in terms of cross-validated accuracy. However, the subsequent implementation of regularization layers and class weights yielded varying results depending on the classification task at hand.

In the case of the binary model, the introduction of regularization layers resulted in a decrease in test accuracy from 0.9454 to 0.8818. Performance metrics such as precision, recall, and F1-score also saw a reduction from 0.95 to 0.84 across the board. These observations suggest that the addition of regularization layers did not yield the expected improvements for this model. Moreover, the learning curves in the associated plots still exhibit signs of severe overfitting, indicating that the model continues to struggle with generalizing its learning to unseen data. Given these outcomes, it might be beneficial to adjust the architecture of the Convolutional Neural Network (CNN), or to explore other regularization techniques or strategies. The objective would be to achieve a better balance between fitting the training data and maintaining good generalization performance on new, unseen data.

In the case of the model predicting the 12 species, the test accuracy witnessed a substantial improvement from 0.8273 to 0.8909, and the loss significantly reduced from 0.9083 to 0.3215. The issue of overfitting was efficiently addressed. However, the training accuracy of 0.6359 was relatively low, hinting at the presence of underfitting, which is corroborated by the corresponding plots. But upon analyzing the performance metrics, there was a noticeable increase in precision, recall, and F1-score. Given these enhancements and the successful mitigation of overfitting, I would recommend the regularized model for further experiments and enhancements.

As for the model categorizing the 22 plant leaf names, the introduction of regularization led to a substantial improvement in the learning curves, effectively addressing the overfitting problem. This progress indicates that the model is now more adept at generalizing its learning from the training data to unseen data, thereby enhancing its predictive performance. While the test accuracy didn't change significantly, remaining between 0.7909 and 0.7727, the loss of the regularized model decreased markedly from 1.9495 to 0.7746. However, when looking at the performance metrics, we see a significant decrease of about 10 points in precision, recall, and F1-score. Despite the reduction in loss and the mitigation of overfitting, this drop in key performance metrics suggests that the regularized model may not be the best choice for this task. Further refinements or a different regularization strategy might be needed to balance the model's generalization ability with its predictive performance.

In conclusion, the approach of hyperparameter tuning combined with the use of regularization techniques and class weights proved to be a powerful strategy in optimizing the performance of the models. However, the effectiveness of these techniques can vary depending on the specific classification task, and therefore a degree of experimentation and adjustment may be required to achieve the best results.

## A Appendix

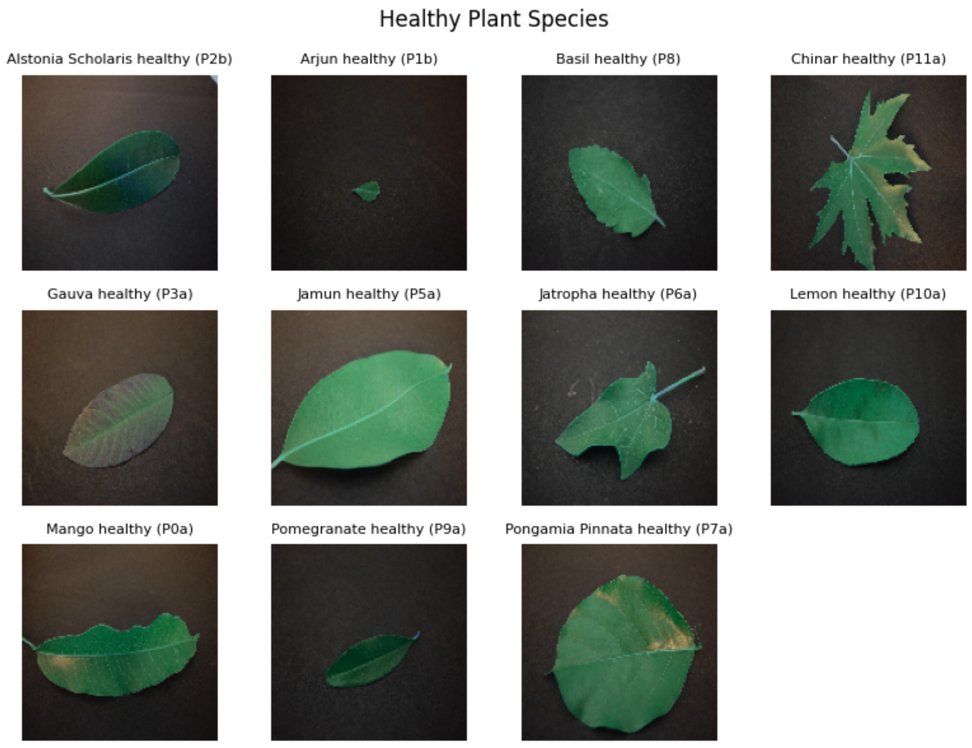


Figure 3: Healthy plant species

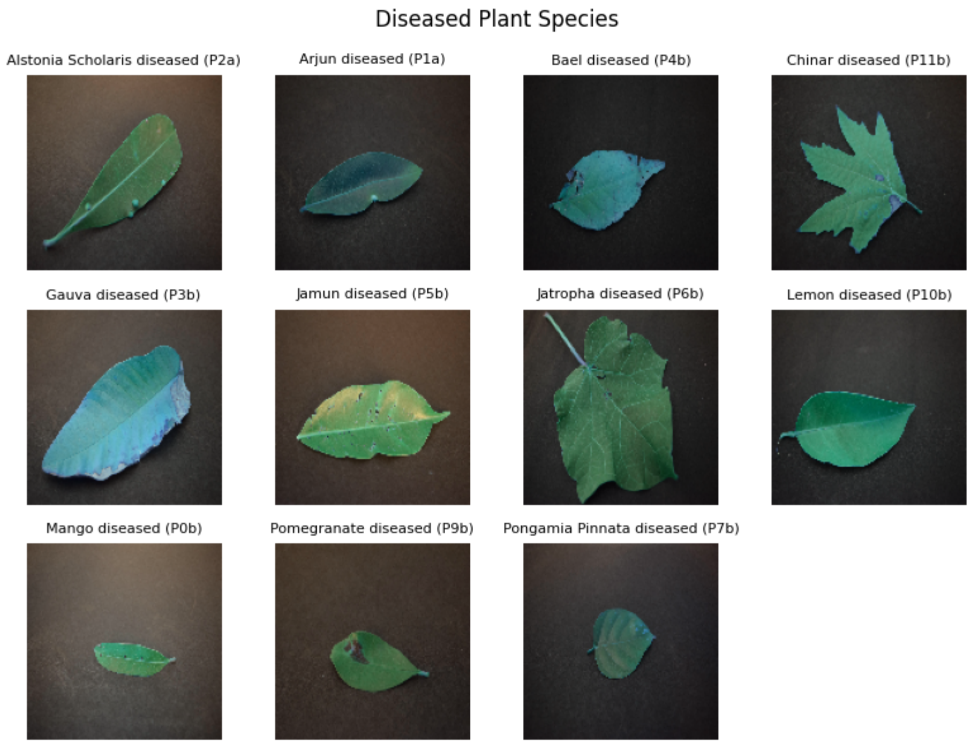


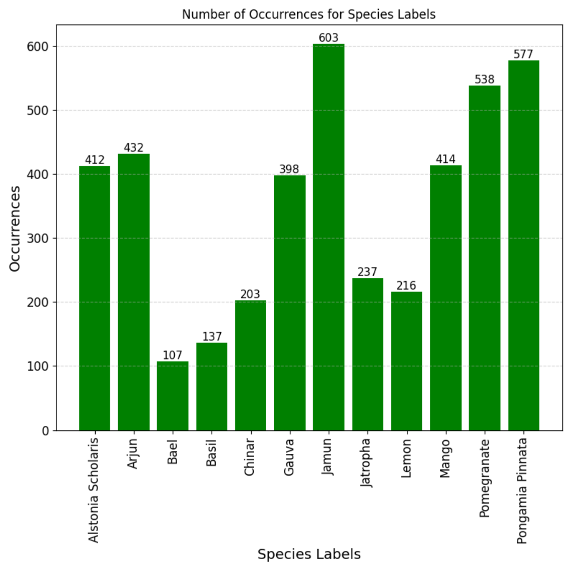
Figure 4: Diseased plants species

Table A.1: Label Counts in Training, Validation, and Test Sets

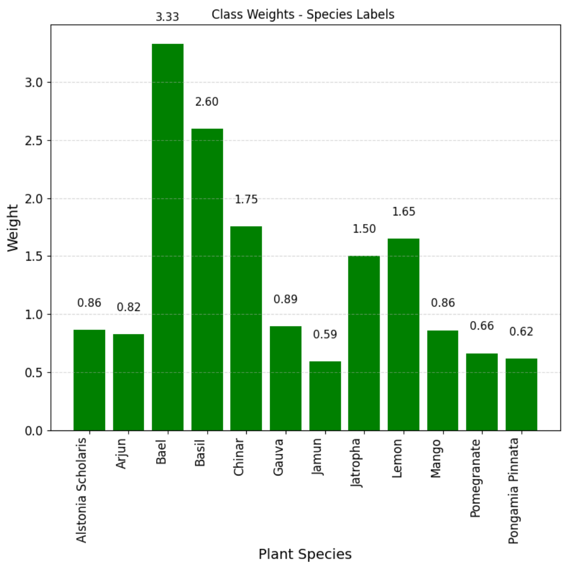
| **Binary Labels** | **Training Set** | **Validation Set** | **Test Set** |
|---|---|---|---|
| 0 (Healthy) | 2163 | 55 | 55 |
| 1 (Diseased) | 2111 | 55 | 55 |

| **Species Labels** | **Training Set** | **Validation Set** | **Test Set** |
|---|---|---|---|
| Alstonia Scholaris | 412 | 10 | 10 |
| Arjun | 432 | 10 | 10 |
| Bael | 107 | 5 | 5 |
| Basil | 137 | 5 | 5 |
| Chinar | 203 | 10 | 10 |
| Gauva | 398 | 10 | 10 |
| Jamun | 603 | 10 | 10 |
| Jatropha | 237 | 10 | 10 |
| Lemon | 216 | 10 | 10 |
| Mango | 414 | 10 | 10 |
| Pomegranate | 538 | 10 | 10 |
| Pongamia Pinnata | 577 | 10 | 10 |

| **Complete Labels** | **Training Set** | **Validation Set** | **Test Set** |
|---|---|---|---|
| Alstonia Scholaris diseased (P2a) | 244 | 5 | 5 |
| Alstonia Scholaris healthy (P2b) | 168 | 5 | 5 |
| Arjun diseased (P1a) | 222 | 5 | 5 |
| Arjun healthy (P1b) | 210 | 5 | 5 |
| Bael diseased (P4b) | 107 | 5 | 5 |
| Basil healthy (P8) | 137 | 5 | 5 |
| Chinar diseased (P11b) | 110 | 5 | 5 |
| Chinar healthy (P11a) | 93 | 5 | 5 |
| Gauva diseased (P3b) | 131 | 5 | 5 |
| Gauva healthy (P3a) | 267 | 5 | 5 |
| Jamun diseased (P5b) | 335 | 5 | 5 |
| Jamun healthy (P5a) | 268 | 5 | 5 |
| Jatropha diseased (P6b) | 114 | 5 | 5 |
| Jatropha healthy (P6a) | 123 | 5 | 5 |
| Lemon diseased (P10b) | 67 | 5 | 5 |
| Lemon healthy (P10a) | 149 | 5 | 5 |
| Mango diseased (P0b) | 255 | 5 | 5 |
| Mango healthy (P0a) | 159 | 5 | 5 |
| Pomegranate diseased (P9b) | 261 | 5 | 5 |
| Pomegranate healthy (P9a) | 277 | 5 | 5 |
| Pongamia Pinnata diseased (P7b) | 265 | 5 | 5 |
| Pongamia Pinnata healthy (P7a) | 312 | 5 | 5 |

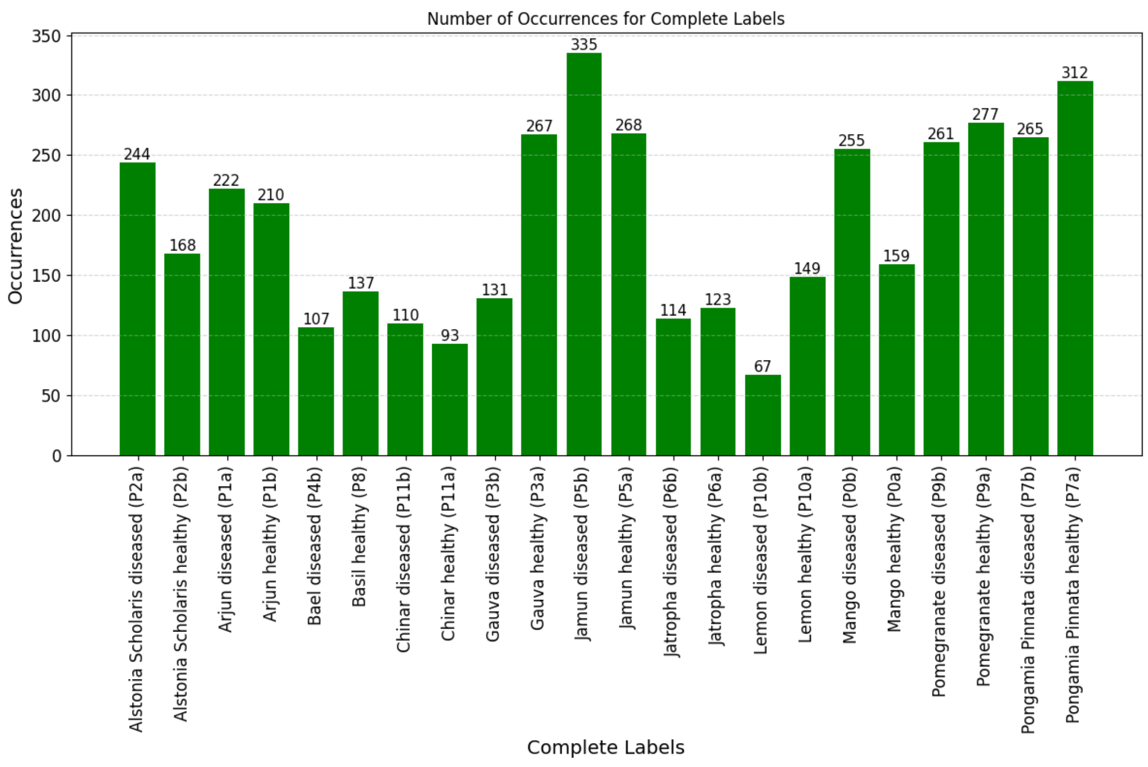


(a) Frequency of images per species

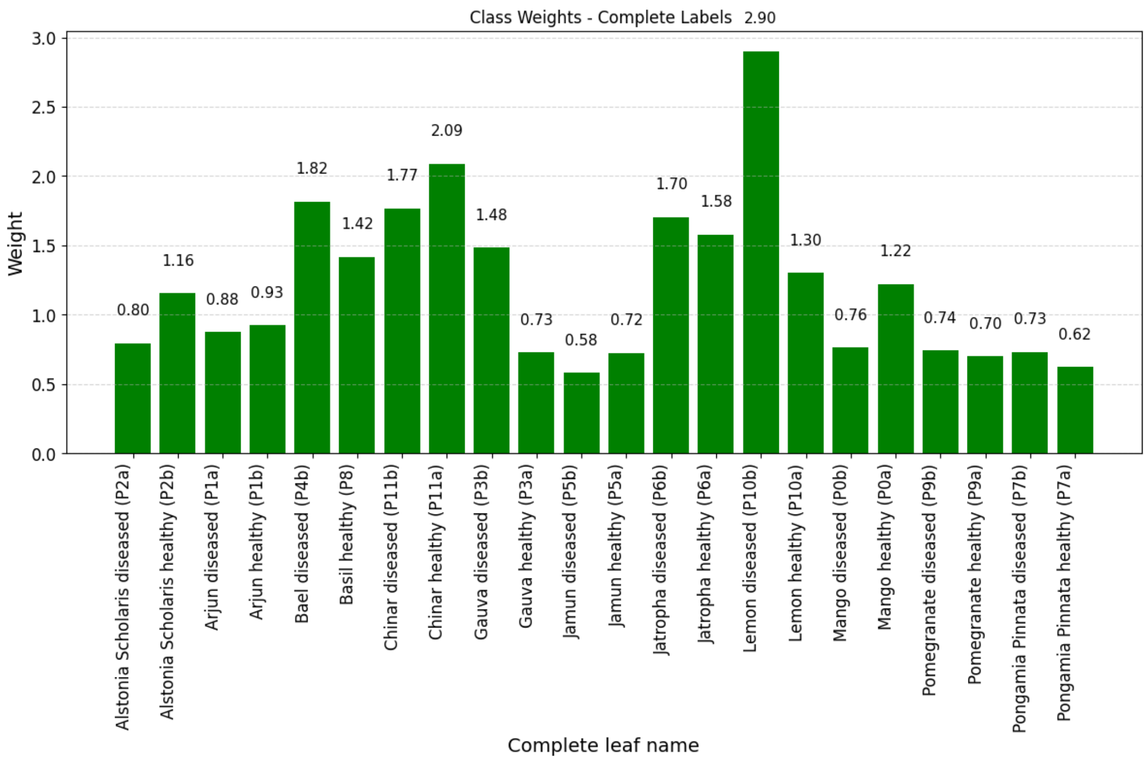


(b) Weights per species

Figure 5: Numerosity per species and class weights (training set)



(a) Numerosity per complete leaf name (training set)



(b) Class weights

Figure 6: Numerosity per complete leaf name and class weights

## B Appendix

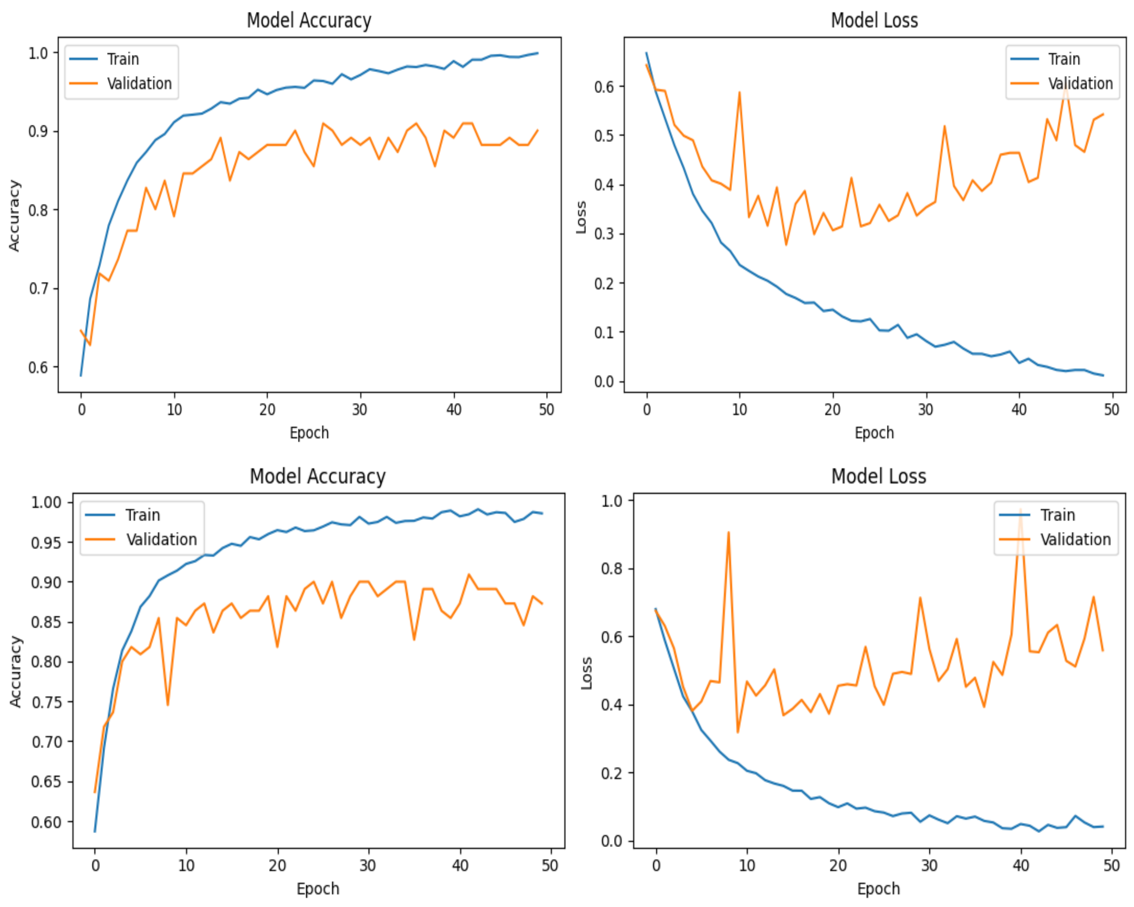


Figure 7: Training-validation loss and accuracy curves for the best binary model (above) and the binary model with regularization layers (below).

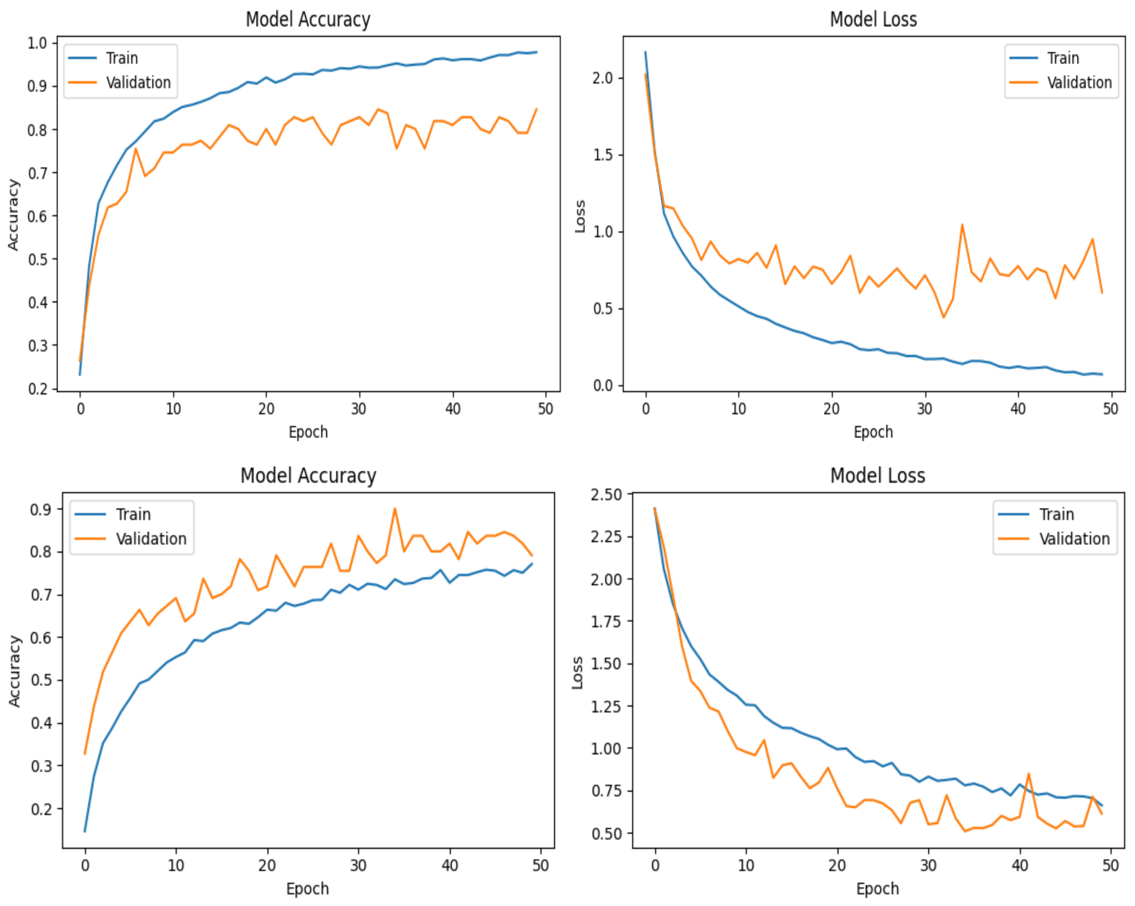


Figure 8: Training-validation loss and accuracy curves for the best 12 species model (above) and the 12 species model with regularization layers and weights (below).

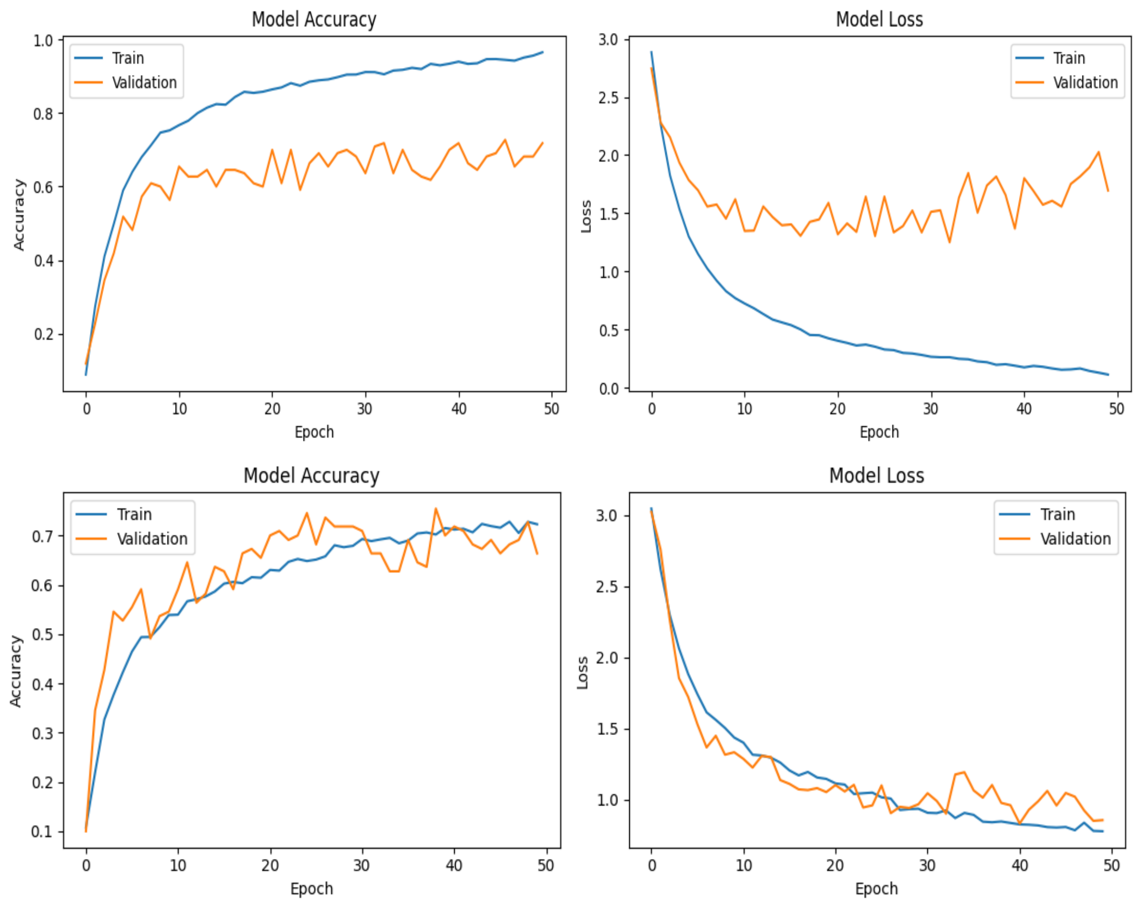


Figure 9: Training-validation loss and accuracy curves for the best 22 leaves names model (above) and the 22 leaves names model with regularization layers and weights (below).

```
           precision    recall  f1-score   support
   Healthy       0.93      0.96      0.90        55
  Diseased       0.96      0.82      0.88        55
 macro avg       0.95      0.95      0.95       110

           precision    recall  f1-score   support
   Healthy       0.89      0.87      0.88        55
  Diseased       0.88      0.89      0.88        55
 macro avg       0.88      0.88      0.88       110
```

Figure 10: Classification report of binary models (Best model above, regularized model with weights below)

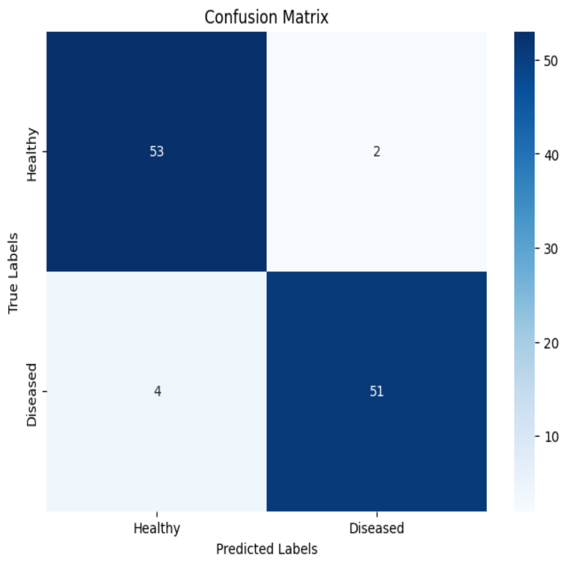
```
                  precision    recall  f1-score   support
Alstonia Scholaris       0.89      0.80      0.84        10
             Arjun       0.91      1.00      0.95        10
              Bael       1.00      0.80      0.89         5
             Basil       0.71      1.00      0.83         5
            Chinar       0.82      0.90      0.86        10
             Gauva       0.78      0.70      0.74        10
             Jamun       0.62      1.00      0.77        10
          Jatropha       1.00      0.70      0.82        10
             Lemon       0.90      0.90      0.90        10
             Mango       0.89      0.80      0.84        10
       Pomegranate       1.00      0.90      0.95        10
  Pongamia Pinnata       0.62      0.50      0.56        10
         macro avg       0.85      0.83      0.83       110

                  precision    recall  f1-score   support
Alstonia Scholaris       0.90      0.90      0.90        10
             Arjun       0.91      1.00      0.95        10
              Bael       1.00      1.00      1.00         5
             Basil       0.71      1.00      0.83         5
            Chinar       1.00      1.00      1.00        10
             Gauva       0.80      0.80      0.80        10
             Jamun       1.00      0.80      0.89        10
          Jatropha       0.89      0.80      0.84        10
             Lemon       1.00      0.90      0.95        10
             Mango       1.00      0.90      0.95        10
       Pomegranate       0.89      0.80      0.84        10
  Pongamia Pinnata       0.69      0.90      0.78        10
         macro avg       0.89      0.89      0.89       110
```

Figure 11: Classification report of 12 species models (Best model above, Model with reg. layers and weights below)

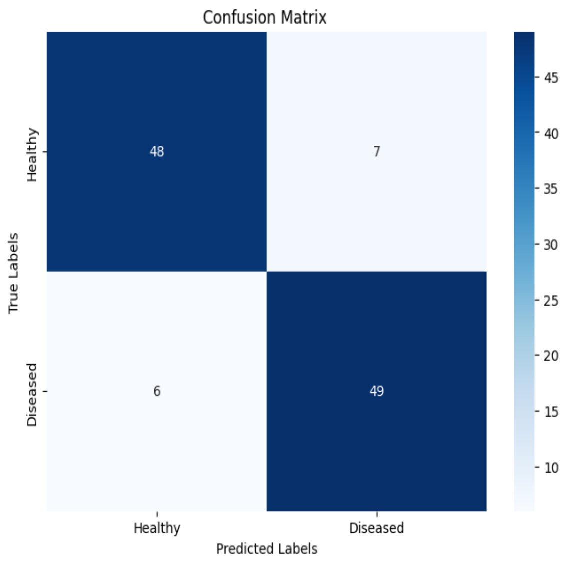
```
                                   precision    recall  f1-score   support
Alstonia Scholaris diseased (P2a)       1.00      1.00      1.00         5
Alstonia Scholaris healthy (P2b)        0.62      1.00      0.77         5
Arjun diseased (P1a)                    0.83      1.00      0.91         5
Arjun healthy (P1b)                     0.83      1.00      0.91         5
Bael diseased (P4b)                     1.00      0.80      0.89         5
Basil healthy (P8)                      1.00      1.00      1.00         5
Chinar diseased (P11b)                  1.00      1.00      1.00         5
Chinar healthy (P11a)                   0.83      1.00      0.91         5
Gauva diseased (P3b)                    1.00      0.60      0.75         5
Gauva healthy (P3a)                     1.00      1.00      1.00         5
Jamun diseased (P5b)                    0.50      0.80      0.62         5
Jamun healthy (P5a)                     0.12      0.20      0.15         5
Jatropha diseased (P6b)                 1.00      0.40      0.57         5
Jatropha healthy (P6a)                  0.60      0.60      0.60         5
Lemon diseased (P10b)                   1.00      0.40      0.57         5
Lemon healthy (P10a)                    0.62      1.00      0.77         5
Mango diseased (P0b)                    1.00      1.00      1.00         5
Mango healthy (P0a)                     1.00      0.80      0.89         5
Pomegranate diseased (P9b)              1.00      1.00      1.00         5
Pomegranate healthy (P9a)               1.00      0.80      0.89         5
Pongamia Pinnata diseased (P7b)         0.00      0.00      0.00         5
Pongamia Pinnata healthy (P7a)          0.83      1.00      0.91         5
macro avg                               0.81      0.79      0.78       110
```

```
                                   precision    recall  f1-score   support
Alstonia Scholaris diseased (P2a)       1.00      0.40      0.57         5
Alstonia Scholaris healthy (P2b)        0.50      1.00      0.67         5
Arjun diseased (P1a)                    1.00      0.40      0.57         5
Arjun healthy (P1b)                     0.62      1.00      0.77         5
Bael diseased (P4b)                     1.00      1.00      1.00         5
Basil healthy (P8)                      1.00      1.00      1.00         5
Chinar diseased (P11b)                  0.71      1.00      0.83         5
Chinar healthy (P11a)                   1.00      0.80      0.89         5
Gauva diseased (P3b)                    1.00      0.40      0.57         5
Gauva healthy (P3a)                     1.00      1.00      1.00         5
Jamun diseased (P5b)                    0.57      0.80      0.67         5
Jamun healthy (P5a)                     0.38      0.60      0.46         5
Jatropha diseased (P6b)                 1.00      0.80      0.89         5
Jatropha healthy (P6a)                  0.57      0.80      0.67         5
Lemon diseased (P10b)                   1.00      0.20      0.33         5
Lemon healthy (P10a)                    0.62      1.00      0.77         5
Mango diseased (P0b)                    1.00      1.00      1.00         5
Mango healthy (P0a)                     1.00      1.00      1.00         5
Pomegranate diseased (P9b)              1.00      0.80      0.89         5
Pomegranate healthy (P9a)               0.80      0.80      0.80         5
Pongamia Pinnata diseased (P7b)         1.00      0.20      0.33         5
Pongamia Pinnata healthy (P7a)          1.00      1.00      1.00         5
macro avg                               0.77      0.77      0.77       110
```

Figure 12: Classification report of 22 leaves names (Best model above, Model with reg. layers and weights below)

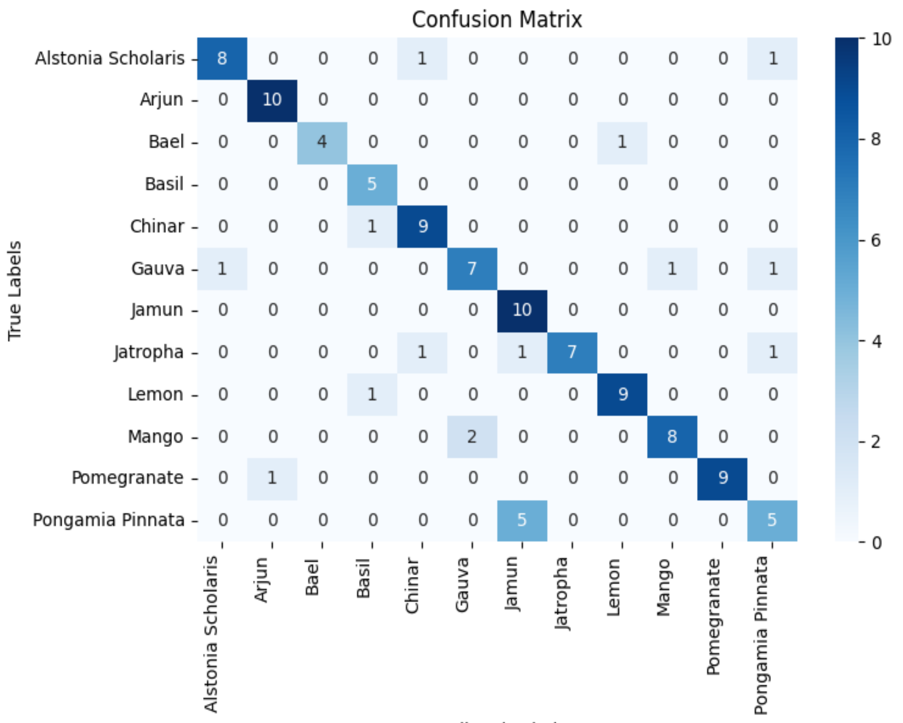


(a) Best model

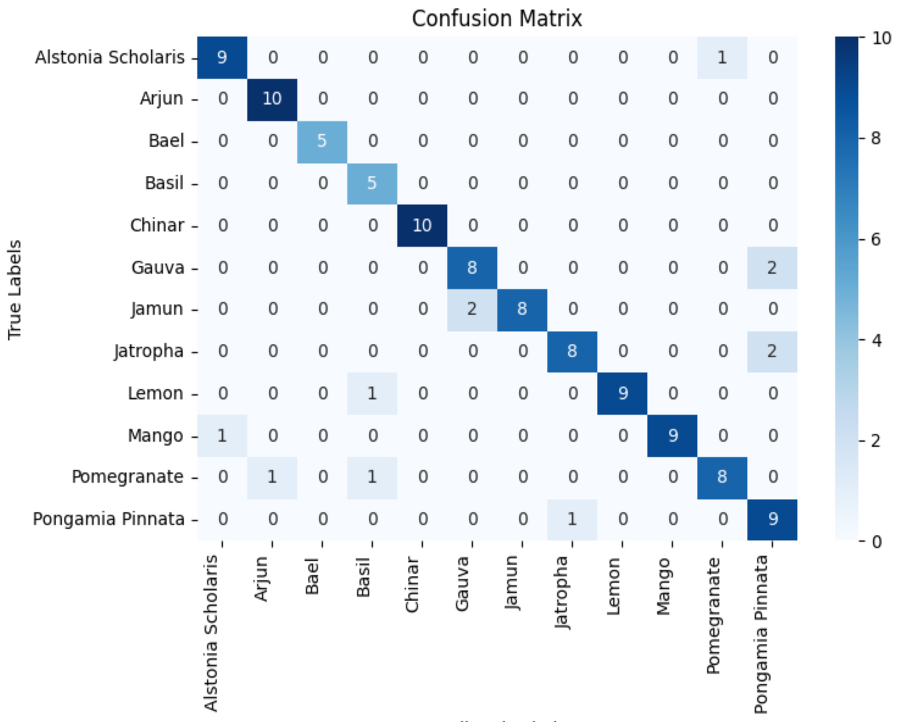


(b) Best model with regularization layers

Figure 13: Confusion matrices for binary classification models

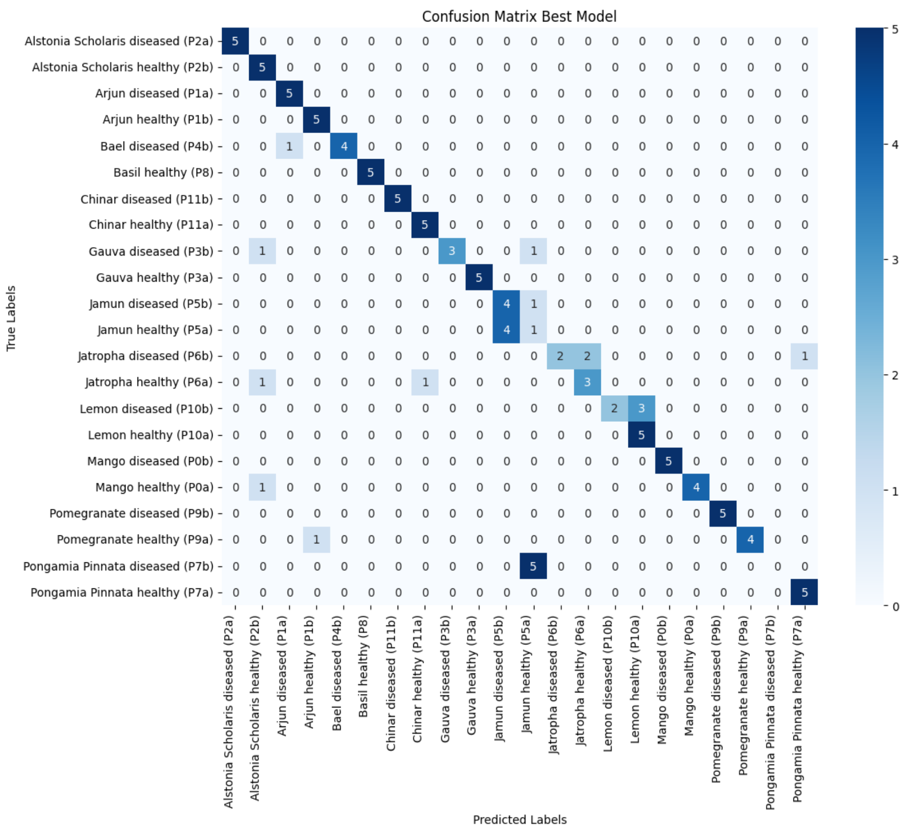


(a) Best model

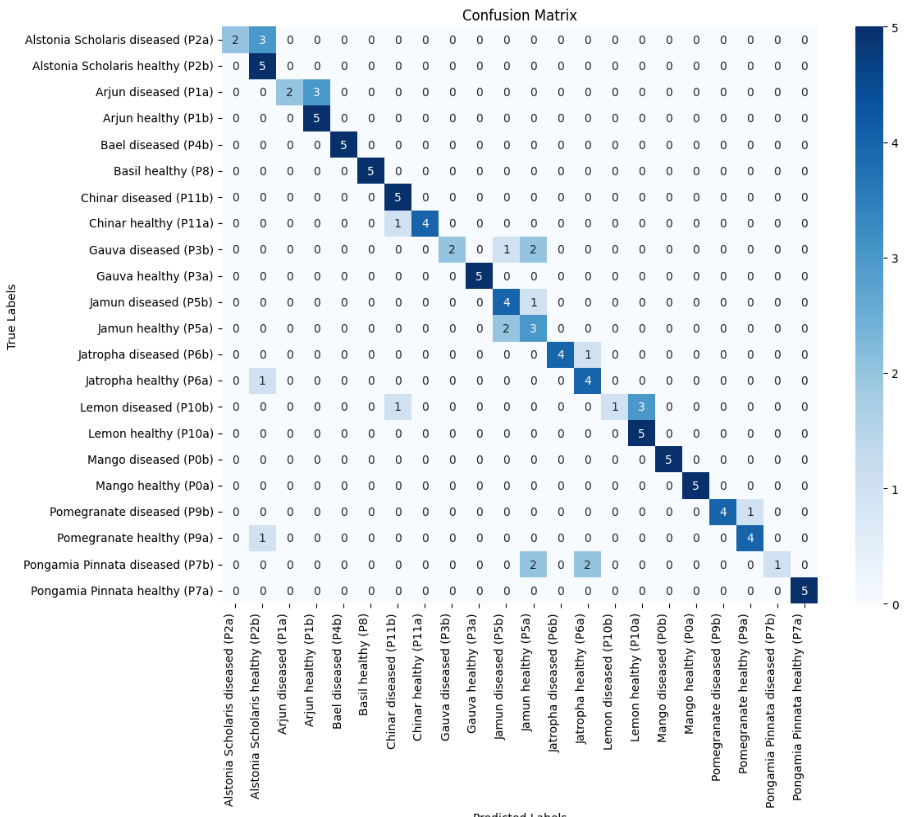


(b) Best model with regularization layers and labels weights

Figure 14: Confusion matrices for multiclass classification models (12 species)



(a) Best model



(b) Best model with regularization layers and labels weights

Figure 15: Confusion matrices for multiclassification models (22 names)