Leaves Image Classification using Neural Networks

Giuseppe Scaffidi Caruso, Andrew Thomas Costa

March 2023

Sommario

In the last decade, there has been rapid growth in the amount of image data generated worldwide. This has made automated image processing essential for simplifying image-related tasks, for example the detection of certain objects. These structure of the neural networks - modelled after the neural networks found in the human brain - contain neurons with learnable weights and biases. These parameters are trained to learn the patterns found within the data that its fed, with the goal of producing an accurate prediction.

This paper presents various implementations of a convolutional neural network for the classification of plant images. The proposed model is designed with less complexity while still providing good classification accuracy for all tested datasets. Furthermore, evaluation metrics were highest when the image data was not augmented.

Indice

Elenco delle figure

Elenco delle tabelle

1 Goal of the analysis

1.1 Model Applications

This paper aims to compare two versions of Convolutional Neural Networks (CNNs) that were built, trained and tuned in different ways. The first model was developed as a binary classifier, assigning either labels 0 if the leaf is healthy or 1 if the leaf is unhealthy. T

The comparison between the two models will be based specifically on evaluation metrics, with particular focus on the F1 (the harmonic mean of the precision and recall) macro-average score as this better accounts for imbalanced datasets by weighting classes equally. The results of this study will provide insight into the effectiveness of different training and hyperparameter tuning techniques as methods to improving the performance of CNN models for image classification tasks.

The second approach was to explore the development of a custom CNN model based on the format of the original dataset - classification based on the plant type and whether that plant was infected or not. Four models were developed and tested, one of those being a model whose hyperparameters were then tuned and tested, comparing its performance to the other models.

Hyperparameter tuning entails searching through the hyperparameter space to find the optimal combination of hyperparameters that results in the highest validation accuracy on the given dataset. This involved tweaking the values of various parameters, such as the learning rate for the Adam optimizer, batch size, the number of units in the dense layer and number of filters in the convolutional layer to find the optimal combination that resulted in the highest accuracy with the least computational resources and model size.

Models
Efficient net
CNN 1st Config
CNN 2nd Config
CNN 3rd Config
CNN tuned

Tabella 1: Algorithm Developed

2 Analysis

2.1 Dataset Description

The dataset used in this project consists of approximately 4,503 images of healthy and diseased leaves from 12 different plants, including Mango, Arjun, Alstonia Scholaris, Guava, Bael, Jamun, Jatropha, Pongamia Pinnata, Basil, Pomegranate, Lemon, and Chinar.

The images were classified and labeled based on the plant species, ranging from P0 to P11. The entire dataset was then divided into 22 subject categories, with classes labeled from 0000 to 0011 marked as healthy and from 0012 to 0022 they were marked as diseased.

The dataset contains 2,278 images of healthy leaves and 2,225 images of diseased leaves, making it a balanced dataset in terms of class distribution. The leaf images of each plant were acquired in both healthy and diseased conditions, and they were divided into two separate modules for analysis.

2.2 Data Visualization

By visualizing the number of images in each class, it is possible to determine whether the dataset is balanced or imbalanced. If the number of images in each class is roughly equal, the dataset is balanced. However, if the number of images is significantly different for different classes, the dataset is considered imbalanced.
In the given code, the bar chart plot of class counts shows that the dataset is imbalanced, since there are a different number of images for each class label. This will provide an idea of the characteristics of the leaves in each class and how they differ from one another.

Class labels Analysis

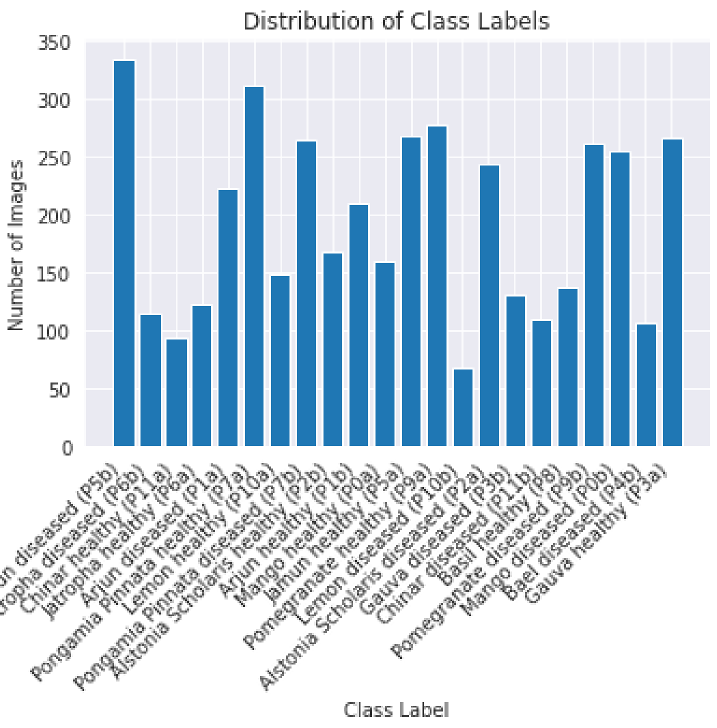


Figura 1: Distribution of class labels

The data visualization for the first model is framed as a binary classification problem, where the input images are labeled as either "sick" or "not sick" (as shown in Figure 1). The binary encoding of 0 or 1 is used to represent the labels, where 0 corresponds to "sick" and 1 corresponds to "not sick". On the other hand, in Figure 3, the visualization

Visualization of Binary Image Labels

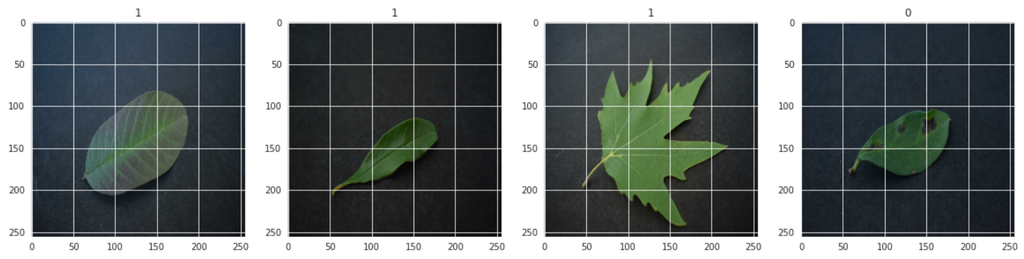


Figura 2: Plot of leaves

will show a collection of leaves from each class, where the classes have not been encoded as binary 0 or 1. This will provide an idea of the characteristics of the leaves in each class, and how they differ from one another.

Visualization of Non Binary Image Labels

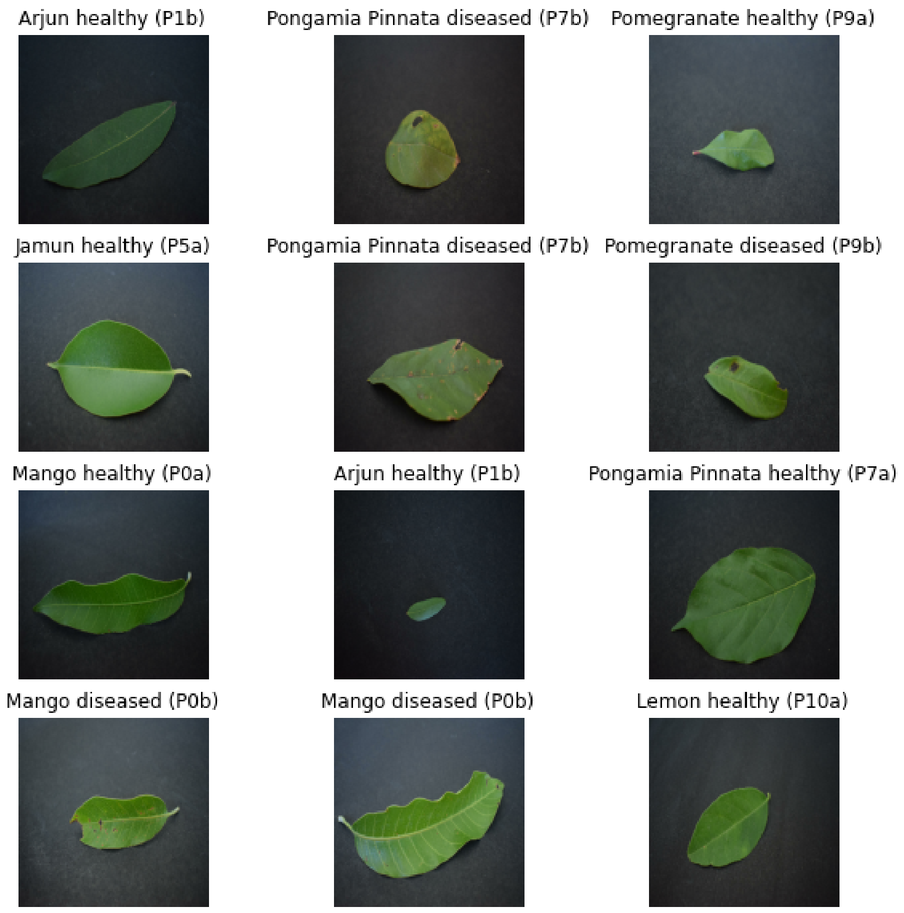

Arjun healthy (P1b)
Pongamia Pinnata diseased (P7b)
Pomegranate healthy (P9a)
Jamun healthy (P5a)
Pongamia Pinnata diseased (P7b)
Pomegranate diseased (P9b)
Mango healthy (P0a)
Arjun healthy (P1b)
Pongamia Pinnata healthy (P7a)
Mango diseased (P0b)
Mango diseased (P0b)
Lemon healthy (P10a)

Figura 3: Non Binary Images, each leaf has its code

3 Binary Model

3.1 Pre-Processing

Firstly, a train, validation, and test sets set by splitting images from a source directory (original dataset) and copying them to a destination directory (binary dataset) based on their labels. The images are split into two sub-directories based on their labels: healthy and diseased. The labels for each image were determined by parsing the name of the sub-folders of the source image path, resulting in the correct number of folder splits that were ready to be ingested.
In the second part, using TensorFlow Keras API, it was possible to load and prepare image datasets for the training and testing of the model. This method is useful for loading and preprocessing image data as it automates the process of reading and downloading the images from the file whilst also creating labeled datasets.

The sizes of the created train, test and validation sets are the following:

- The train dataset contains 4274 images for training the model
- The validation dataset contains 110 images for validating the model
- The test dataset contains 110 images for testing the model

The next step after we obtained the datasets is to iterate over train, validation and test using the numpyiterator() method to convert the data into NumPy arrays. This allows the process to easily extract the data from the dataset and use it in their model without performing additional conversions.

Overall, after the above process was implemented, there would be batches consisting of a tuple containing the image data and corresponding labels in NumPy format. The next() function is applied to iterate through the batches until all data has been processed correctly.

As the last step of pre-processing, a **lambda function** will be applied to the train, validation and test datasets. By applying this function, it will be possible to map the function X (batch of input images), and y (batch corresponding target labels) and normalize each pixel coming from X dividing by 255 while the y label remains the same. This process will generate 3 new normalized datasets ready to be used in the model.

3.2 Convolutional Neural Network (CNN)

The model created is a CNN with a Sequential class from the Keras API composed of the following layers:

- **Conv2D**: these are convolutional layers that apply filters to the input images to extract features.
- **MaxPooling2D**: these are pooling layers that downsample the feature maps generated by the convolutional layers
- **Flatten layer**: this layer flattens the output of the last MaxPooling2D layer into a 1D array.
- **Dense**: these are fully connected layers that apply linear transformations to the input data

One can see the training process of the first model - that was ran for 20 epochs - in Figure 4.

3.2.1 First Model

The base model shape comes from many trials applied to arrive at this result. This first model has 10 layers in his architecture. The number of nodes in the hidden layers of the provided code is not explicitly specified, although a brief recap of the architecture is as follows:

- The first convolutional layer has 16 filters of size 5x5, which will produce 16 feature maps.
- The second convolutional layer has 32 filters of size 5x5, which will produce 32 feature maps.
- The third convolutional layer has 64 filters of size 5x5, which will produce 64 feature maps.
- The dense layer prior to the output layer has 128 nodes.

By computing the neural networks with this specific pre-compiled model, we get poor results in terms of its performance. The main problem arising from this trained model is underfitting, where the model cannot capture the underlying patterns in the data.

The model trained on the validation data for 10 epochs with a batch size of 50 yielded a training accuracy of 0.87% and a test accuracy of 0.70

Taking a look at the **Loss per Epochs** for both Accuracy and Validation: The

Training and Validation Accuracy

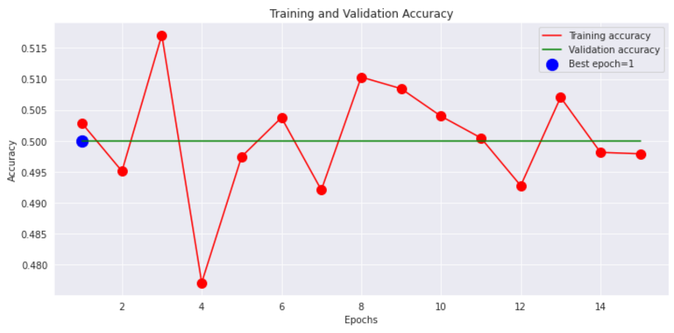


Figura 4: Accuracy Score 1st Model

Training and Validation Loss

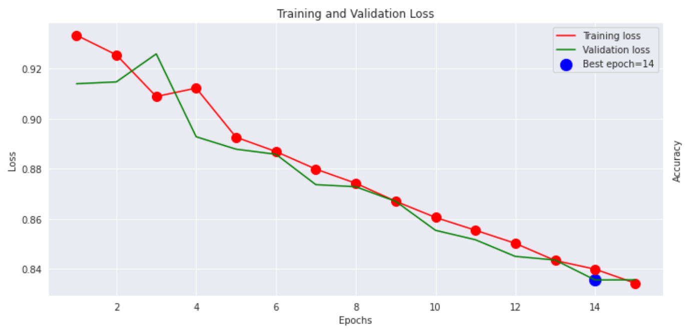


Figura 5: Loss Score 1st Model

model has a test accuracy of 50%, which means it is not performing better than random guessing. The confusion matrix has predicted all samples as the majority class, resulting in a precision of 0.0 and recall of 0.0 for the minority class. The F1 score is also 0.0 due to the undefined precision. This suggests that the model is not learning to distinguish between the two classes and is likely underfitting or has not been trained for enough epochs. It is necessary to improve the model's performance before using it for practical purposes.

3.2.2 Second Model

It's clear how the model is struggling to learn the patterns in the data, resulting in poor validation accuracy. To improve the poor performance from the first model architecture these changes have been implemented. The depth of the model has been increased by adding more convolutional layers, increasing the kernel size to 7x7, and adding batch normalization layers after each convolutional layer. Using Batch normalization can help stabilize the learning process by normalizing the inputs to each layer while increasing larger kernel size to allow the model to capture more significant patterns in data. The activation function was also changed from ReLU to ELU. ELU has a range of values between negative infinity and infinity, whereas ReLU has a range of values between 0 and infinity. The optimizer was changed from Stochastic Gradient Descent (SGD) to Adam. One of the main benefits of switching from SGD to the Adam optimizer was that Adam is an adaptive, learning rate optimization algorithm, which means that it can dynamically adjust the learning rate during training. This can help accelerate the model's convergence by quickly adapting the learning rate based on the gradient of the loss function.

The only specification this model needs on layer architecture concerns the last layer. The output layer has a single sigmoid unit for binary classification. The optimizer used for training is Adam with a learning rate of 0.0001 and the loss function is binary cross-entropy.

Training and Validation Accuracy

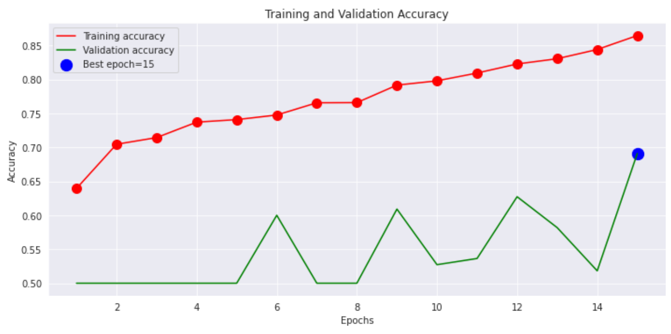


Figura 6: Accuracy Score 2nd Model

Training and Validation Loss

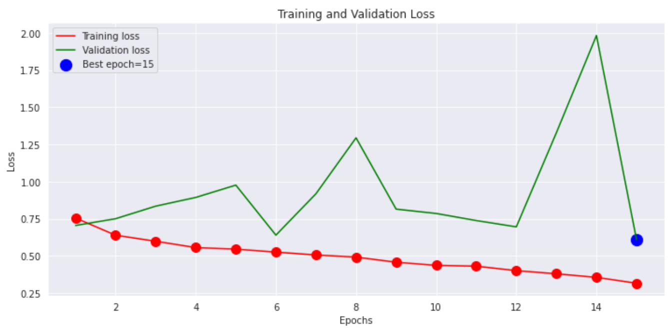


Figura 7: Loss Score 2nd Model

4 CNN - Multilabel Classification

4.1 Pre-Processing

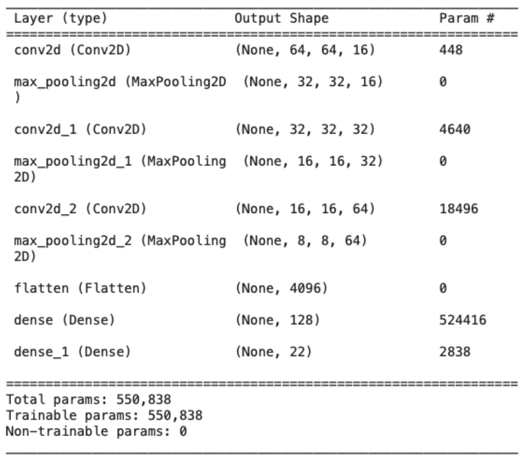
The data for this approach was processed in a similar manner to the model used for the binary classification task, however, here, the data was left in its original format where the sub-folders were divided along plant type and disease.

The images were resized to 64x64 pixels in order to speed-up the computation of the model. The original format of the images (256x256) would have provided more data for the model to learn but would have consequently increased the training time due to hardware limitations.

The augmentation of the data was also explored in this project. For example, flipping, rotating, adjusting the brightness and adjusting the colour, but we discovered that this actually decreased the performance of the model (as a measure of the Macro-average F1-score). The only augmentation that was used for this process that didn't see a reduction in model performance was the rescaling of the images between values [0,1], prompting the model to treat all images in the same manner. If images were left in their original format, different pixel configurations would impact the loss of the model i.e. higher pixel values could lead to a higher loss, whereas the opposite may also stand. In addition, without scaling higher pixel range images would then have a larger number of votes in how to update the weights.

4.2 First Model

The first model was composed of three convolutional Layers, two max pooling layers and two dense layers, the second of those dense layers being the output layer.



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 64, 64, 16)        448

 max_pooling2d (MaxPooling2D  (None, 32, 32, 16)       0
 )

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        4640

 max_pooling2d_1 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 flatten (Flatten)           (None, 4096)              0

 dense (Dense)               (None, 128)               524416

 dense_1 (Dense)             (None, 22)                2838

=================================================================
Total params: 550,838
Trainable params: 550,838
Non-trainable params: 0
_________________________________________________________________
```

Figura 8: First Model Architecture

The results of the model are shown in Figure 9. As one can see, the training loss for the model decreases rapidly over the first epoch and then steadily over the remaining epochs. The validation loss however, does not decrease in conjunction with training loss; it begins at 1.8 and remains mostly constant throughout. The path of the validation loss is indicative of underfitting and its fluctuations underline the presence of noise in the data. A method to alleviate this is to increase the complexity of the model so that it can learn data better.

The accuracy follows a similar trajectory to that of the loss, where the training loss smoothly converges to 1.0 but the validation loss tapers off at approximately 0.61.

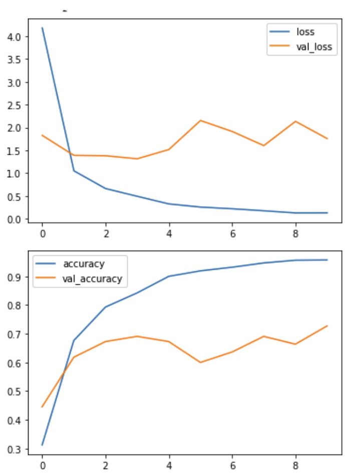
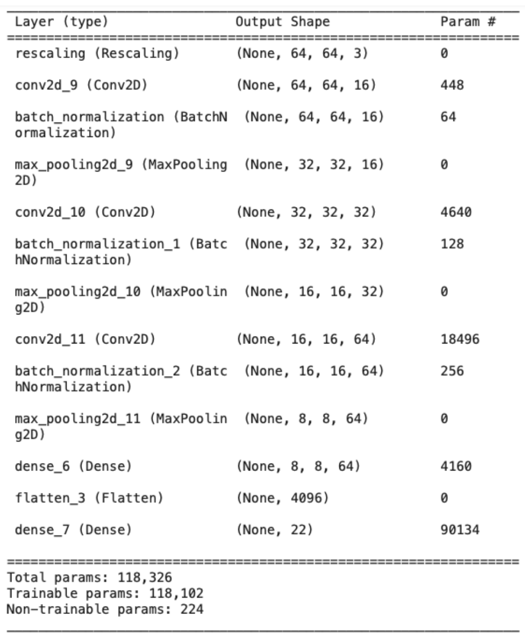


Figura 9: First Model Performance

The Macro-average F1-score, precision and recall scores were all 0.5 - this can be significantly improved and will be the focus of the next section.

4.3 Second Model

The architecture for the second model includes a greater number of convolutional blocks, as well as batch normalisation layers in an attempt to combat this underfitting. We also increased the number of epochs to 30, to allow the model to learn for a greater period of time.



```
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 64, 64, 3)         0

 conv2d_9 (Conv2D)           (None, 64, 64, 16)        448

 batch_normalization (BatchN  (None, 64, 64, 16)       64
 ormalization)

 max_pooling2d_9 (MaxPooling  (None, 32, 32, 16)       0
 2D)

 conv2d_10 (Conv2D)          (None, 32, 32, 32)        4640

 batch_normalization_1 (Batc  (None, 32, 32, 32)       128
 hNormalization)

 max_pooling2d_10 (MaxPoolin  (None, 16, 16, 32)       0
 g2D)

 conv2d_11 (Conv2D)          (None, 16, 16, 64)        18496

 batch_normalization_2 (Batc  (None, 16, 16, 64)       256
 hNormalization)

 max_pooling2d_11 (MaxPoolin  (None, 8, 8, 64)         0
 g2D)

 dense_6 (Dense)             (None, 8, 8, 64)          4160

 flatten_3 (Flatten)         (None, 4096)              0

 dense_7 (Dense)             (None, 22)                90134

=================================================================
Total params: 118,326
Trainable params: 118,102
Non-trainable params: 224
_________________________________________________________________
```

Figura 10: Second Model Architecture

There was an improvement in results for the second model, as one can observe from the graph below. The training accuracy and loss converge to 1.0 smoothly. The validation accuracy and loss to eventually converge and stabilise around 0.8 and 1.4 respectively. The increased number of convolutional layers meant that the model could learn more patterns within the data, and this enhancement was also reflected in the Macro-average F1-score, which was 1.0.

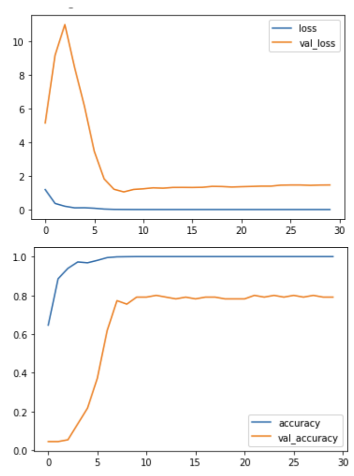


Figura 11: Second Model Results

4.4 Hyperparameter Tuning

The model can nevertheless still be improved, especially to alleviate the discrepancy seen in the plots of Figure 11, where there's a sharp spike in the loss and accuracy plots. This can potentially be achieved by fine-tuning the hyperparameters. The parameters of focus will be the kernel size, the number of filters for each convolutional block and the learning rate. After running this process, the model underperformed in relation to the other models developed; specifically, the best validation accuracy achieved was 0.54, whilst the Macro-average F1-score was 0.3. Tuning the model was computationally expensive, given the expansive hyperparameter search space and the hardware limitations.

5 Conclusion

Image data represent a high-dimensional feature space, so it is necessary for the model to cope with this complexity. As best practice, we created a baseline model to compare the subsequent models against. The first multi-label model failed to adequately grasp this complexity, as the model severely underfitted. We then increased the number of convolutional layers in the second model and found that the performance significantly improved, achieving a Macro-average F1-score of 1.0.

The predictive power of the second binary classification model, which aims to determine if leaves are diseased or healthy, has been evaluated using various metrics. The **test loss** highlights the extent of discrepancies between the model's predictions and the ground truth, indicating that the predictions contain a certain degree of error. The **test accuracy** reveals that the model has correctly classified approximately 67.27% of the images in the test dataset. Although this performance is better than random chance, it suggests there is still potential for improvement.

Examining the **confusion matrix** provides a more granular perspective on the model's performance in a binary classification problem. The matrix's main diagonal represents true positive and true negative values, while the off-diagonal elements correspond to false positive and false negative values. By taking a closer look at these values, we can better understand the model's performance and potential areas of improvement.

- 21 True Negatives (TN): Accurate classification of healthy leaves
- 34 False Positives (FP): Erroneous classification of diseased leaves as healthy
- 24 False Negatives (FN): Erroneous classification of healthy leaves as diseased
- 31 True Positives (TP): Accurate classification of diseased leaves

The **precision** value helps us understand that, among all leaves predicted as diseased, approximately 47.69% were indeed diseased. The **recall** value reveals that the model correctly identified about 56.36% of all diseased leaves within the test dataset. The **F1 score**, in this case, stands at 0.5167, which indicates the potential for further enhancement of the model's performance.

In conclusion, the binary classification CNN, trained on leaf images, has achieved moderate performance with an accuracy of 67.27% and an F1 score of 0.5167. There is room for improvement, and refining the model or supplementing the training data may contribute to more robust performance.
The binary classification model's poor performance might be attributed to the imbalance in size between the training data (comprising 4,000 images) and the smaller validation and test sets (with 100 images each). This discrepancy can lead to several issues:

- Insufficient validation and test data: With only 100 images in each set, they may not adequately represent the underlying data distribution. Consequently, the evaluation metrics obtained from these sets might not accurately reflect the model's true performance.
- Overfitting: In this situation, the model performs exceptionally well on the training data but fails to generalize to new, unseen data, leading to poorer performance on the validation and test sets.
- Variability in results: Smaller validation and test sets can result in higher variability in evaluation metrics, as a few misclassifications can significantly impact the overall performance measures.

We declare that this material, which I/We now submit for assessment, is entirely my/our 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/our work. I/We understand that plagiarism, collusion, and copying are grave and serious offences in the university and 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/us or any other person for assessment on this or any other course of study.