Plant leave recognizer

Master in Data Science and Economics
Project for algorithms for massive data course with prof. Malchiodi

Luca Merlini
Matriculation n°: 983220

September 2023

Abstract

The objective of this project is to develop a scalable system to address a supervised learning problem. More precisely, the primary aim is to accurately categorize images of leaves into 12 distinct classes. This project will involve the implementation of a deep learning model using the Keras package from TensorFlow to train a neural network capable of performing multiclass classification on the "Plant Leaves for Image Classification" dataset. In order to distinguish these images, the training is done with Convolutional Neural Networks. These kinds of network are a popular choice to solve image recognition problems.

Declaration

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 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.

Contents

1 Introduction

The following project applies Convolutional Neural Networks (CNNs) to tackle a multi-class image classification problem. CNNs are chosen as the primary algorithm due to their specialized design for image processing and their well-established effectiveness in image classification tasks.

The project workflow begins with a thorough examination and exploration of the dataset, followed by a meticulous data organization and the establishment of a scalable and distributed computing environment. Initial network architectures are developed, with a focus on identifying those with the best performance. Several techniques are introduced to reduce overfitting, ensuring the models generalize well. Hyperparameter tuning is performed by optimizing some a priori defined parameters. Towards the conclusion of the project, transfer learning is explored by utilizing pretrained architectures trained on large datasets. The best performing network is selected and tested on an unseen test dataset.

2 Preliminary operations

2.1 Data loading and organization

The dataset in question is taken from Kaggle. For better code reusability and reproducibility, data is managed through the kaggle API, an automated way to download and load data directly within the code, ensuring consistency across different environments.

The dataset contains 4494 images, of which 4274 for the training set, 110 for the validation set, and 110 for the test set. The images belong to 22 classes, each containing images of a specific type of plant with its health status. In Table 1 it is possible to see the organization of the images in the different folders.

Subfolder	Df train files	Df valid files	Df test files
Arjun diseased (P1a)	222	5	5
Mango diseased (P0b)	255	5	5
Pomegranate diseased (P9b)	261	5	5
Pongamia Pinnata diseased (P7b)	265	5	5
Mango healthy (P0a)	159	5	5
Lemon diseased (P10b)	67	5	5
Bael diseased (P4b)	107	5	5
Arjun healthy (P1b)	210	5	5
Lemon healthy (P10a)	149	5	5
Pongamia Pinnata healthy (P7a)	312	5	5
Chinar healthy (P11a)	93	5	5
Jatropha diseased (P6b)	114	5	5
Pomegranate healthy (P9a)	277	5	5
Alstonia Scholaris diseased (P2a)	244	5	5
Jatropha healthy (P6a)	123	5	5
Alstonia Scholaris healthy (P2b)	168	5	5
Jamun healthy (P5a)	268	5	5
Gauva healthy (P3a)	267	5	5
Jamun diseased (P5b)	335	5	5
Basil healthy (P8)	137	5	5
Gauva diseased (P3b)	131	5	5
Chinar diseased (P11b)	110	5	5
Total	4274	110	110

Table 1: Distribution of images in train,validation, and test folders

Table 1 illustrates the limited size of the validation dataset, comprising only 110 images, which accounts for approximately 2.4% of the total dataset. Ensuring an appropriately sized validation set is crucial to avoid overfitting and ensure that the model generalizes well to unseen data. To achieve this, a decision is made to merge the training and validation folders and then to perform train-validation splits with a more balanced ratio.

Furthermore, considering the project's objective of predicting 12 distinct plant species, regardless of their health status, subfolders containing the same plant species are merged.

Table 2 presents the new file organization. The code for this reorganization is designed with scalability in mind, accommodating the potential expansion of the dataset to include more classes. In fact, the number of classes is dynamically determined based on the post-organization folder count.

Subfolder	df train valid files	df test files
Basil	142	5
Arjun	442	10
Jatropha	247	10
Chinar	213	10
Lemon	226	10
Bael	112	5
Jamun	613	10
Pomegranate	548	10
Mango	424	10
Pongamia Pinnata	587	10
Alstonia Scholaris	422	10
Gauva	408	10
Total	4384	110

Table 2: New distribution of images in train,validation, and test folders

2.2 Environment setup

The experiment[1] is conducted through Google Colab with GPU T4 due to neural networks being computationally complex and expensive. In fact, the GPU-driven approach significantly reduces computational time and sets a benchmark for timing standards. To ensure replicability from the project's outset, keras.utils.set_random_seed(42) is defined. It allows for the same seed to be set for numpy, tensorflow, and python environments. Furthermore, as per the commitment to replicability, a random initialization approach is adopted; more specifically, the HeNormal distribution technique with the seed set to 42 is applied. This method defines the way to set the initial random weights of Keras layers. Appropriate initial values for weights can help mitigate the issue of vanishing gradients during training process. In this project, the HeNormal initialization stands as the recommended choice, especially for networks employing the ReLU activation function.

Furthermore, to enhance scalability for larger datasets, the MultiWorkerMirroredStrategy is defined. This distribution strategy is specifically designed for data parallelism in distributed training across multiple GPUs and/or machines, making it particularly convenient for scenarios where the dataset is large and cannot fit into a single GPU or machine.

By employing the MultiWorkerMirroredStrategy, the classification model can be efficiently replicated across multiple GPUs or machines, allowing each worker to process a portion of the training data in parallel. This parallelism not only accelerates the training process but also ensures that each worker learns from a diverse subset of the data, which can lead to better generalization.

Before defining the strategy for distributed training, it is essential to set the number of workers and configure a JSON document for the cluster of workers. Specifically, the local address of the used machines has to be specified, and the "chief" worker (index=0), responsible for saving checkpoints and writing summary files, has to be designated.

The strategy.scope method is pivotal, enabling data parallelism[2]. Here, a single model is replicated across multiple devices or machines, each processing distinct batches of data before merging their results.

[1]Exception made for the hyperparameter tuning for lack of computional resources.

[2]Model building and model compiling need to be within this method

2.3 Data preprocessing

Before constructing the deep learning model, a series of preprocessing steps are applied to the image data. These crucial steps include:

1. **Scaling**: The images are rescaled, ensuring that pixel values range between 0 and 1, instead of the original range 0-255.
2. **Resizing**: The images are resized to a uniform size of (224, 224) pixels.
3. **Color Conversion**: The images are converted from their initial format (jpg) to RGB, resulting in three-channel tensors (224, 224, 3).

Additionally, to optimize memory usage and computational efficiency, the images are divided into batches, each containing 128 samples. These preprocessing operations are performed through the keras data_generator function applying the flow_from_directory method. Notably, this method enables to perform a split between training and validation data, with a split ratio of 0.9 for training and 0.1 for validation.

A visual representation of a random sample of resized images (224x224) is provided in Figure 1.

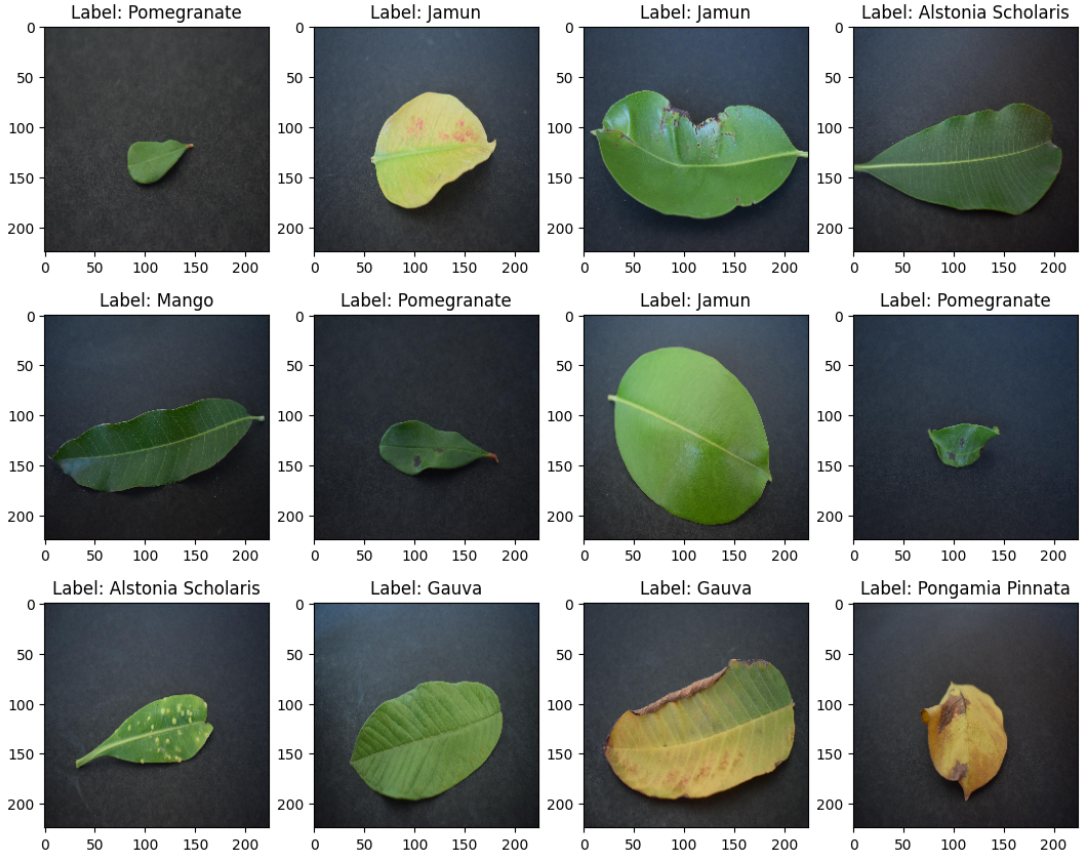


Figure 1: Random sample of 12 pictures belonging to the training set.

From the images sample above, it becomes apparent that there is significant similarity between images from different classes, and that the black background is a consistent feature. This uniformity presents a challenge, especially when distinguishing between classes, particularly if one is unfamiliar with the various types of leaves. Consequently,

the development of a precise model capable of discerning intricate image details becomes important.

3 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are specialized networks designed and optimized for the task of image recognition. These networks draw inspiration from the functioning of the human brain, particularly the cortex, where specific neurons respond to patterns of activations, such as horizontal, vertical, or oblique lines. Some neurons combine these simple patterns to recognize more complex features.

Two types of layers are introduced with this type of neural network: the convolutional layer and the pooling layer.

3.1 Convolutional layer

In the convolutional layer, neurons are not connected to every input unit, but are selectively linked to input units within their receptive field. This pyramidal structure allows for the representation of low-level features in the initial layers, progressively combining them into more intricate features in the upper layers. Unlike dense layers, each layer in the convolutional layer is represented in 2D or 3D.

Neurons within a receptive field contribute to a small image known as a filter. When a filter is applied to the entire layer, a new layer, referred to as feature map, is generated.

Convolutional layers typically incorporate multiple filters, each producing a distinct feature map. These layers effectively identify regions that strongly activate the filter. During training, the network learns the most "useful" filters, and subsequent layers learn from previous ones how to combine filters in order to identify more complex patterns.

3.2 Pooling layer

Pooling layers play a crucial role in reducing computational complexity, memory usage, and the number of parameters. Each unit in a pooling layer connects to a limited set of units from the previous layer. The idea is similar to the filter in a convolutional layer: however, a pooling layer aggregates the input values using an aggregation function such as maximum or mean.

3.3 Architecture example

Typically, CNN architectures stack multiple convolutional layers followed by pooling layers iteratively. Towards the end of a sequence, a flattening operation is applied to transform the data into a format suitable for a feedforward network, which consists of hidden layers for further processing. An illustrative example is shown in Figure 2.

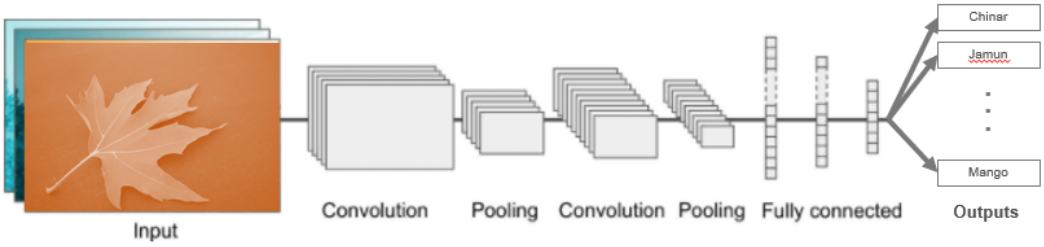


Figure 2: Example of the architecture of a possible CNN with convolutional, pooling, and dense layer

Best practices suggest employing smaller filter sizes and combining multiple layers with small filter sizes, rather than using a single layer with a larger filter size. This approach enhances the network's ability to capture intricate features in the data.

4 Experiment

4.1 Model basic

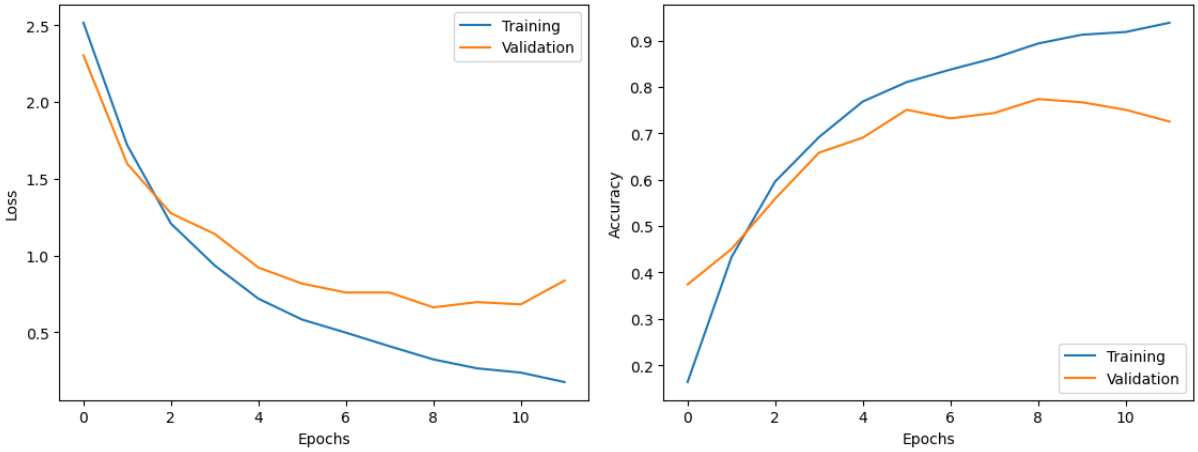
Following established best practices in creating a Convolutional Neural Network (CNN), an initial network architecture is designed as follows:

- Two convolutional layers are employed, with the first using 16 three-by-three filters and a ReLU activation function, and the second using 32 three-by-three filters. A stride of 1 is applied to preserve the size of the data.
- These convolutional layers are followed by maxpool2D pooling layers, which select the maximum value within a 2x2 input window.
- In addition to the convolutional structure, the neural network includes a dense layer with 64 neurons and a ReLU activation function. The output layer consists of 12 units with a softmax activation function.
- Training occurs over 15 epochs, with early stopping implemented, monitoring validation loss with a patience of 3 epochs.

Rectified Linear Unit (ReLU) activation functions are chosen for the hidden layers due to their nonlinearity, preventing gradient vanishing issues during backpropagation through deep layers. This activation function has demonstrated strong empirical performance in this type of network. For the output layer, softmax activation is selected to transform raw scores into interpretable probability distributions, with the number of output neurons matching the number of different classes.

The Adam optimizer is selected for its capability to adapt the learning rate, ensuring robust gradient descent.

This first model exhibits significant overfitting, as evidenced by superior performance on the training data compared to validation data. In fact, the validation loss shows an increasing trend in the last epochs. The performance achieved in the final epoch is as follows: loss: 0.1752, acc: 0.9382, val_loss: 0.8365, val_acc: 0.7252. Figure 3 provides a representation of the loss and accuracy trends across the epochs.



(a) Training and validation losses

(b) Training and validation accuracy

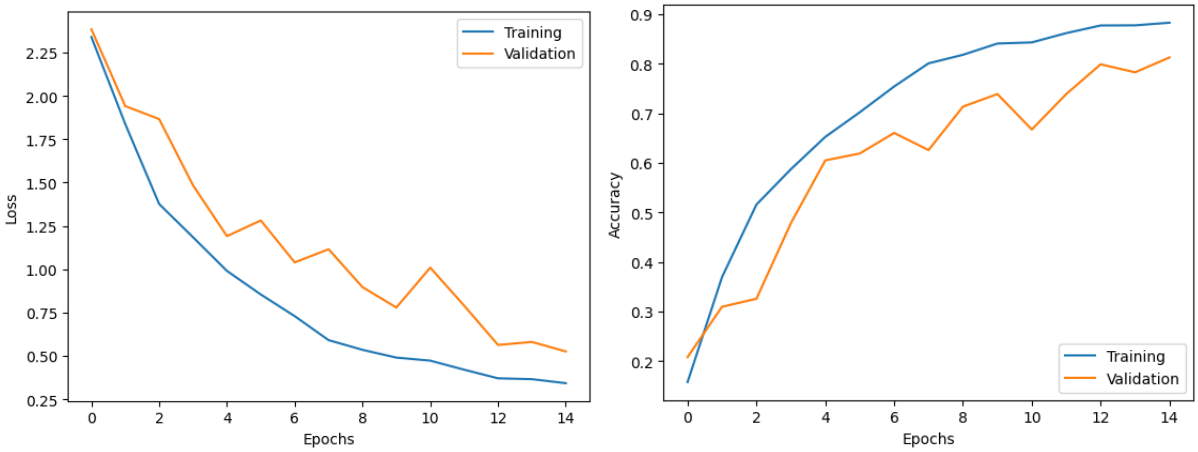
Figure 3: Training and validation losses and accuracy of model basic

4.2 Model with dropout

To address the overfitting issue, a second neural network with a distinct architecture is implemented. This revised architecture consists of the following components:

- Four convolutional layers, each with an increasing number of filters: 16, 32, 64, and 128, respectively. All convolutional layers utilize a three-by-three kernel size and a ReLU activation function.
- A dropout layer with a dropout fraction of 0.15 is introduced between each convolutional layer. Dropout is a regularization technique used to mitigate overfitting. Dropout helps to address this issue by randomly deactivating a fraction of input units during each training step. This forces the network to learn more robust and generalized features.
- Subsequently, a dense layer with 512 neurons is added. A dropout layer follows the dense layer, with a dropout rate of 0.2. The output layer keeps 12 units with a softmax activation function.

In this second architecture, dropout helps to reduce overfitting. The results obtained from this second model continue to exhibit suboptimal performance: loss: 0.3429 - acc: 0.8831 - val_loss: 0.5262 - val_acc: 0.8129. Figure 4 provides representation of the loss and accuracy trends across the epochs.



(a) Training and validation losses

(b) Training and validation accuracy

Figure 4: Training and validation losses and accuracy of model with dropout

4.3 Model with dropout and data augmentation

Another approach to improve model performance is through the application of data augmentation, a common technique in CNN training. Data augmentation involves applying various transformations to images, including rotation, zooming, blurring, contrast adjustment, and image flipping. This technique effectively expands the training dataset by introducing realistic variations of the original training images. The application of transformations to the training dataset introduces variability and helps to reduce overfitting, which is particularly useful when dealing with data sparsity, a common challenge in neural networks since they need a large amount of data. Data augmentation compensates for data sparsity by generating new examples that are similar but not identical to the original data, as demonstrated in Figure 5.

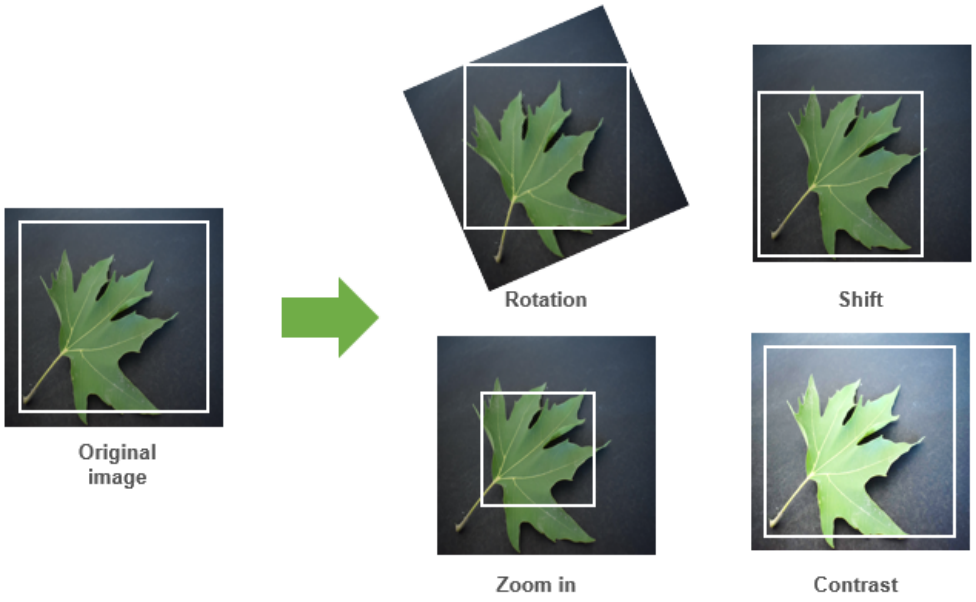
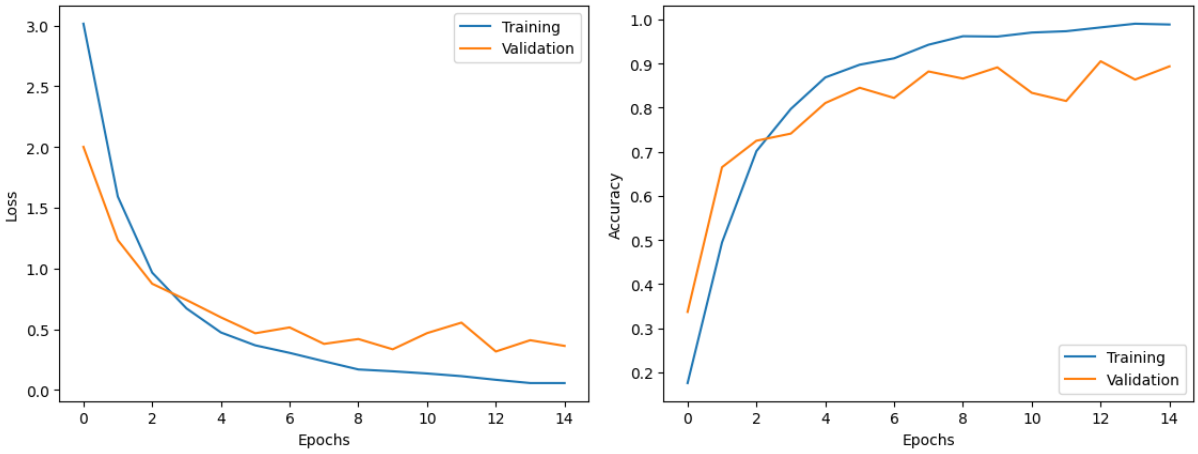


Figure 5: Example of some transformation of data augmentation

In this modified architecture, a single layer of data augmentation is integrated, specifically employing a RandomRotation with a factor of 0.1. Like the previous network, the subsequent architecture consists of the following components:

- Five convolutional layers with filter counts of 16, 32, 64, 32, and 16, respectively. All convolutional layers utilize a three-by-three kernel size and ReLU activation functions.
- A dropout layer with a dropout rate of 0.1 is placed between each convolutional layer.
- Again, convolutional layers are alternated with maxpool2D pooling layers, extracting the maximum value from 2x2 input windows.
- Finally, a dense layer with 512 neurons is introduced, followed by a dropout layer with a dropout rate of 0.2 for the dense layer. The output layer consists of 12 units with a softmax activation function.

Even though this network represents an improvement over the previous one, a notable 9% gap persists between the training and validation accuracy. The resulting performance metrics are as follows: loss: 0.0573 - acc: 0.9886 - val_loss: 0.3632 - val_acc: 0.8938. Figure 6 illustrates the trends in loss and accuracy across the epochs.



(a) Training and validation losses

(b) Training and validation accuracy

Figure 6: Training and validation losses and accuracy of model with dropout and data augmentation

4.4 Hyperparameter tuning

In this section, an attempt to enhance the performance of the previous network through hyperparameter tuning is shown. The fundamental architecture retained includes:

- A single data augmentation layer, RandomRotation with a factor of 0.1.
- Five convolutional layers featuring three-by-three filters, ReLU activation functions, followed by max pool2D layers with a 2x2 size.
- Two dense layers.

- An output layer with 12 neurons and a softmax activation function.

The objective here is to optimize following parameters:

- The number of filters for the five convolutional layers, selected from the options: 16, 32, 64, 128.
- The number of neurons for the two dense layers, chosen from the options: 1024, 512, 256, 128 for the first layer and 128, 64, 32 for the second layer.
- The dropout rate for the convolutional layers (maintaining the same rate for all layers) within the range of 0.05 to 0.3 at intervals of 0.05.
- The dropout rate for dense layers (maintaining the same rate for all layers) varying from 0.1 to 0.4 at intervals of 0.1.
- The learning rate for the Adam optimizer, selected from the values: 0.0005, 0.0010, and 0.0015.

The hyperparameter optimization is realized using the keras-tuner library, automating the tuning process to identify the combination which maximizes model performance. The first tested method of keras tuner is the RandomSearch method with a maximum of 10 iterations, which is relatively low given the expansive search space. However, even though convergence is difficult with such a low number, it is not possible to increase it due to computational time issues.

To overcome this constraint, the BayesianOptimization method with a maximum of 10 iterations is implemented. This approach achieves convergence more efficiently, due to the fact that it exploits the information acquired during the search process to select successive hyperparameter values. Although these methods provide valuable insights into optimal hyperparameter combinations, the constraints of a limited number of attempts and a reduced epoch count (10 instead of 15) present limitations in exploring the entire optimization space.

In light of these constraints, the hyperparameter tuning did not yield substantial improvements in the performance of the previous network. Consequently, a new approach is pursued: transfer learning.

4.5 Transfer learning

Transfer learning is an advanced machine learning technique in which a model pre-developed to perform a specific task is reused as a starting point for the creation of a new model tailored to a different task. Typically, the lower layers of a complex architecture are retained, while the upper layers are modified to fit the new task. This approach is favored when dealing with complex datasets or when resource constraints make it difficult to develop a model from scratch.

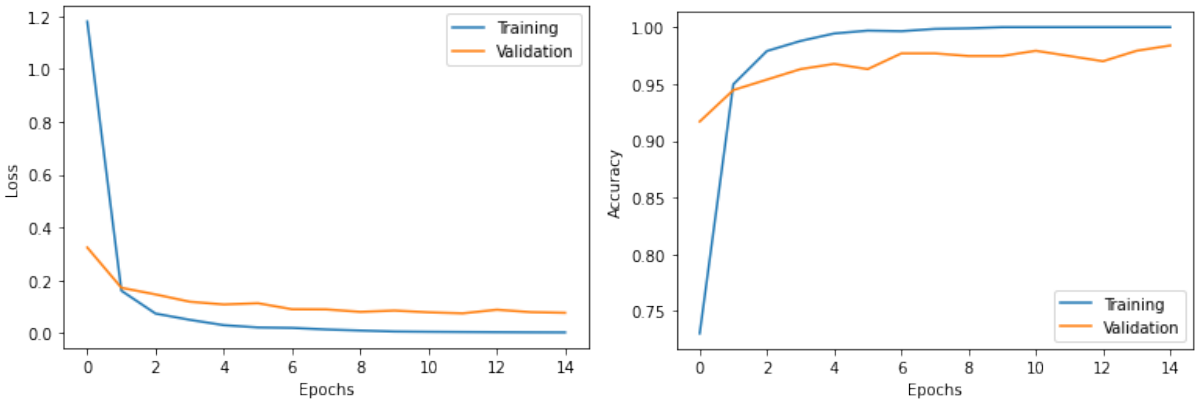
The first model analyzed is ResNet50. It is used with parameters estimated on the "imagenet dataset", an extremely large dataset of images, which is a benchmark for computer vision models.

ResNet50 is a complex architecture consisting of:

- An initial block with a convolutional layer with an input size requirement of 224x224x3.

- A pooling layer with Batch Normalization and ReLU Activation.
- Four macroblocks, each with three convolutional layers. These macroblocks exhibit similarities, with the number of filters in the convolutional layers doubling from one block to the next.
- A Global Average Pooling layer that condenses the feature map output into a 1x1x2048 vector.
- A fully connected dense layer with 1000 neurons and a Softmax Activation.

The pre-trained model is employed in its entirety minus the output dense layer. The pre-trained layers are frozen, preventing re-training, while the model is personalized with a single dense layer featuring 2048 neurons and ReLU activation. In output there is always one layer with 12 neuron and softmax activation function. To employ this network with transfer learning, it is necessary to use an ImageDataGenerator using a ResNet50-specific preprocessing function. The performance of this network surpasses that of all previously used networks: loss: 0.0027 - acc: 1.0000 - val_loss: 0.0769 - val_acc: 0.9838. Since it is a pre-trained network, achieving outstanding performance requires only a low number of epochs. Figure 7 shows the trends in loss and accuracy across the 15 epochs.



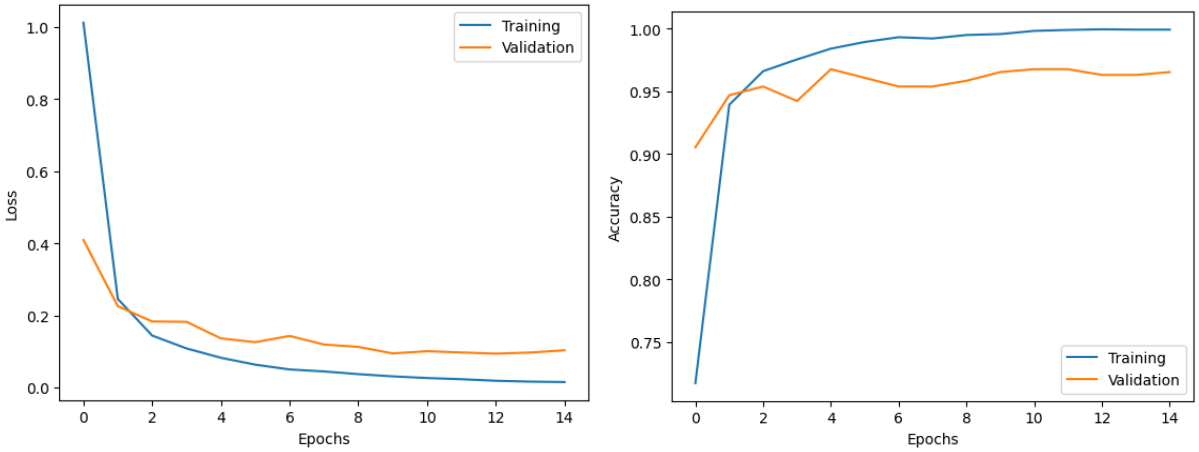
(a) Training and validation losses

(b) Training and validation accuracy

Figure 7: Training and validation losses and accuracy of ResNet50

DenseNet121 is another network architecture that is experimented with, following the same methodology as before. Compared to ResNet50, DenseNet121 offers a more complex architecture. It also requires an input size of 224x224x3. DenseNet121 consists of several dense blocks, and each dense block contains multiple convolutional layers. Within each dense block, the output feature maps of all previous layers are concatenated as inputs to the current layer. Towards the end of the network, there are fully connected layers with a softmax activation function for classification. The exact number of layers and neurons in the dense blockslayers may vary. Similar to the approach used with ResNet50, the pre-trained DenseNet121 model is employed in its entirety, except for the dense and output layers. The model is specialized with a dense layer consisting of 256 neurons, ReLU activation function, and in output one layer with 12 neuron and softmax activation function. Just like ResNet50, DenseNet121 also needs its own unique preprocessing function. This network also demonstrates strong performance within a

small number of epochs, yielding the following results: loss: 0.0152 - acc: 0.9992 - val_loss: 0.1033 - val_acc: 0.9654 Figure 8 shows a representation of the trends in loss and accuracy across the epochs.



(a) Training and validation losses

(b) Training and validation accuracy

Figure 8: Training and validation losses and accuracy of Densenet121

5 Evaluation

5.1 Predictions on test set

As the last step of the project, the performance evaluation is conducted on an unseen test dataset comprising 110 images, distributed as outlined in Table 1. The selected model for this evaluation is ResNet50, which exhibited the best performance during validation. Remarkably, it achieves an accuracy of 0.9909 and a loss of 0.0312. Table 3 shows a report analyzing in detail the main performances across the 12 classes.

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Alstonia Scholaris	1.00	1.00	1.00	10
Arjun	1.00	1.00	1.00	10
Bael	1.00	1.00	1.00	5
Basil	1.00	1.00	1.00	5
Chinar	1.00	1.00	1.00	10
Gauva	1.00	0.90	0.95	10
Jamun	0.91	1.00	0.95	10
Jatropha	1.00	1.00	1.00	10
Lemon	1.00	1.00	1.00	10
Mango	1.00	1.00	1.00	10
Pomegranate	1.00	1.00	1.00	10
Pongamia Pinnata	1.00	1.00	1.00	10
Accuracy			0.99	110
Macro Avg	0.99	0.99	0.99	110
Weighted Avg	0.99	0.99	0.99	110

Table 3: Classification Report

Table 3 reveals that out of the 110 images, only one is misclassified. Specifically, the model predicts a Jamun leaf as a Guava leaf. This misclassification can be inferred from the metrics, with Guava recall at 0.90 and Jamun precision at 0.91. All other images are correctly classified, reflecting outstanding results, with all metrics surpassing 99%. These results are in line with those obtained during validation by this model. To concretely visualize the results of the model and understand its predictive behavior, Figure 9 shows a sample of 15 images with their respective true and predicted labels.

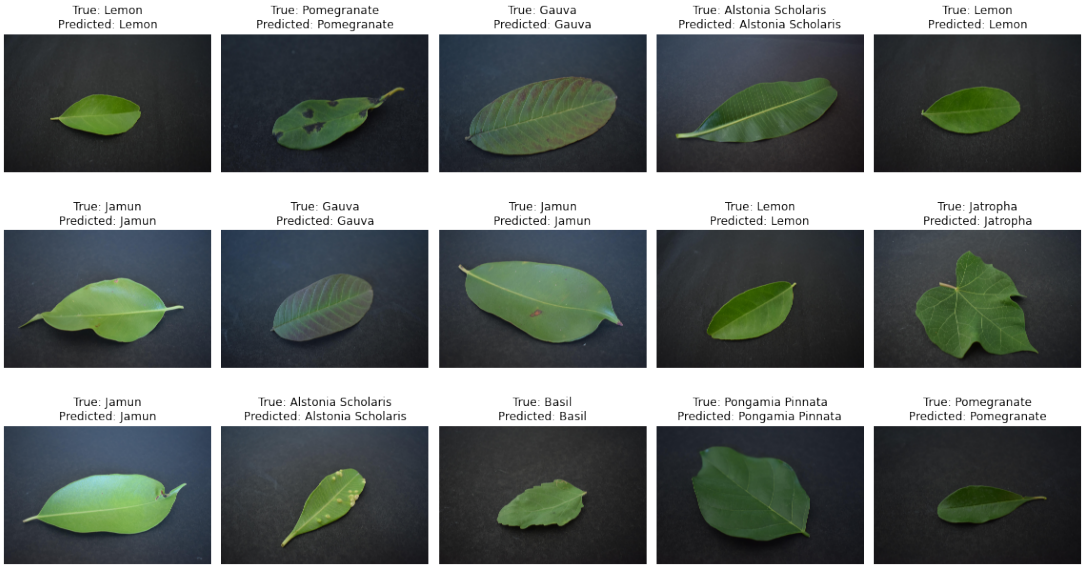


Figure 9: Example of 15 pictures with their true and predicted labels

6 Conclusion

Throughout this project, it is demonstrated how to use convolutional neural networks (CNNs) to solve a multiclass image classification problem in scalable way. After experimenting with the initial networks, model engineering practices are adopted to address overfitting. Specifically, it is shown how the use of data augmentation and dropout helps the model to better generalize, significantly reducing overfitting and improving performances. Hyperparameter tuning is performed as an attempt of increasing performances. The true turning point came with the introduction of transfer learning. By using pretrained models trained on large datasets, the model is able to understand intricate image features, which significantly improves model's performance. In the end, the performances obtained on the unseen test set confirm the effectiveness and robustness of the methodology used.