Università degli Studi di Milano

MSc. in Data Science and Economics

Algorithms for Massive Data

Plant leave recognizer

Alessia Di Giovanni
alessia.digiovanni2@studenti.unimi.it
01207A

September 13, 2023

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.

1 Dataset

The dataset is the **Plant Leaves for Image Classification** dataset published on Kaggle, `https://www.kaggle.com/datasets/csafrit2/plant-leaves-for-image-classification`.
The complete dataset consists of images that have been categorized into two classes: *healthy* and *diseased* leaves. Initially, the images were sorted based on the plant species, which were assigned names from P0 to P11. Then, the entire dataset was further divided into 22 distinct subject categories, denoted by codes ranging from 0000 to 0022. The classes labeled with codes 0000 to 0011 were designated as the *healthy* classes, while those labeled with codes 0012 to 0022 were categorized as the *diseased* classes. In total, the dataset comprises approximately 4503 images, with 2278 images representing *healthy* leaves and 2225 images representing *diseased* leaves.
For my analysis, I decide to eliminate the *diseased* leaves and to keep only the *healthy* ones.
The images are of twelve different plant species, including Mango, Arjun, Alstonia Scholaris, Gauva, Basil, Bael, Jamun, Jatropha, Pongamia Pinnata, Pomegranate, Lemon and Chinar. There are only Bael *diseased* leaves and Basil *healthy* leaves, so when I drop *diseased* leaves, only eleven species remain.
The dataset is already divided in four folders: train, validation, test and images to predict. The first three folders contains subfolders of the species of leaves. There are 2163 images belonging to 11 classes in the train folder, 55 images belonging to 11 classes in the validation folder and 55 images belonging to 11 classes in the test folder.
All images are in *.jpg* format and they are high quality 24MP images, i.e. 6000x4000 pixels. The images are represented in the RGB color space. RGB stands for Red, Green, and Blue, which are the primary colors used to create a wide spectrum of colors in digital images. Each pixel in an RGB image is defined by three color channels (Red, Green and Blue) and the combination of these channels determines the color of the pixel.

2 Preprocessing phase

For the preprocessing phase, I use the `keras.preprocessing.image.ImageDataGenerator` class. I rescale the images by a factor 1/255: in RGB images, each pixel typically has three color channels (Red, Green, and Blue) and each channel can have values ranging from 0 to 255. With normalization each pixel value is divided by 255, scaling the values down to a range between 0 and 1.
I apply zoom, rotation, vertical and horizontal flip on training and validation images: this data augmentation process helps the model in reducing the overfitting problem because it forces the neural network to identify the images when these are being slightly modified.
The `flow_from_directory()` method allows also to set the target size (256x256) of the input images and the batch size (32) for model training. Since there more than two classes, the categorical class mode is used.

3 Convolutional Neural Network

Deep neural networks have started to be very powerful in classifying images when their structure has been modified by introducing the convolutional neural networks.[3] A CNN has three layers.[1]

3.1 Convolution Layer

It involves a filter, or kernel, moving across the receptive fields of the image to identify features. The kernel sweeps over the entire image. At the end of each iteration, a dot product between the filter and the input pixels is calculated. The final output of the Convolution Layer is a feature map. This first operation is defined by four parameters:

- number of filters;
- kernel size: it is fixed as 3x3, meaning each filter is a 3x3 grid that slides over the input image during convolution;
- padding: with "same" padding, padding is added so that the filter can fully traverse the input and the output feature map retains the same spatial dimensions as the input. The same padding adds additional rows and columns of pixels around the edges of the input data so that the size of the output of the feature map is the same of the input data;
- activation function: the activation parameter specifies the activation function applied element-wise to the output of this layer. ReLu function is used and it stands for Rectified Linear Unit, which is a common

activation function used in convolutional neural networks to introduce non-linearity:

$$relu(x) = max(0, x). \tag{1}$$

In the first convolution layer there is also another parameter, the input shape, that is the size of the input images: in this case it is (256, 256, 3), that indicates that the input images are expected to be 256 pixels in height, 256 pixels in width and have 3 color channels (RGB).

3.2 Pooling Layer

It works as the convolution layer: a filter is selected and slides over the output feature map of the preceding convolution layer. The chosen filter size is 2x2 and the approach used to pooling is the max-pooling one. In max-pooling, for each 2x2 receptive field, the pooling layer selects the maximum pixel value among the four pixels in that field. This maximum value becomes the new value in the downsampled feature map. Essentially, max-pooling retains the most prominent or activated feature in each local region, helping to reduce the spatial dimensions while preserving important information.

3.3 Flatten Layer and Fully Connected Layer

The Flatten layer is used to reshape the input data into a 1D vector. This is necessary before connecting to a Fully Connected (Dense) layer. In this layer the features are combined together to create the model and, at the end, the softmax activation function is applied to classify the outputs. The softmax function converts a vector of values to a probability distribution. The number of neurons (units) in the Dense layer determines the dimensionality of the output from that layer.

3.4 Dropout Layer

A Dropout layer is a regularization technique used in neural networks to prevent overfitting. Overfitting occurs when a model learns to perform exceptionally well on the training data but struggles to generalize to unseen data. Dropout helps address this issue. The dropout rate is the fraction of randomly selected neurons (units) in the layer that will be "dropped out" or temporarily turned off during each training batch; it's a value between 0 and 1.

4 Model Architecture

The structure of the convolutional neural network (CNN) that I defined can be summarized as follows:[2]

- Convolution Layer 1:
  - Filters: determined by the filters hyperparameter (16, 32 or 64)
  - Kernel size: (3, 3)
  - Input shape: (256, 256, 3)
  - Padding: same
  - Activation function: ReLU;
- Max Pooling Layer: 2x2;
- Dropout Layer: controlled by the dropout rate hyperparameter;
- Convolution Layer 2:
  - Filters: determined by the filters hyperparameter (16, 32 or 64)
  - Kernel size: (3, 3)
  - Padding: same
  - Activation function: ReLU;
- Max Pooling Layer: 2x2;

- Dropout Layer: controlled by the dropout rate hyperparameter;
- Convolution Layer 3:
  - Filters: determined by the filters hyperparameter (16, 32 or 64)
  - Kernel size: (3, 3)
  - Padding: same
  - Activation function: ReLU;
- Max Pooling Layer: 2x2;
- Dropout Layer: controlled by the dropout rate hyperparameter;
- Flatten Layer;
- Dense Layer 1:
  - Units: determined by the units hyperparameter (128 or 256)
  - Activation function: ReLU;
- Dropout Layer: controlled by the dropout rate hyperparameter;
- Dense Layer 2 (Output Layer):
  - Units: 11 (the number of classes)
  - Activation fucntion: softmax.

The CNN architecture consists of 3 convolutional layers to extract hierarchical features from input images, followed by fully connected (dense) layers for classification. Dropout layers are used for regularization to prevent overfitting. The specific configuration of the layers, including the number of filters, dense units and dropout rates, can be tuned using a hyperparameter optimization framework to find the best configuration.

5 Model compile

To compile the model the optimizer chosen is the Adam optimizer, which is a popular optimization algorithm used for training deep neural networks. It adapts the learning rates for each parameter, incorporates momentum to improve convergence and uses techniques like RMSProp and bias correction for more efficient and stable optimization. It is known for faster convergence and less need for hyperparameter tuning compared to traditional optimization methods like stochastic gradient descent (SGD).
The loss function is the designed to handle integer class labels.
The evaluation metric is the "accuracy", which measures the fraction of correctly classified samples during training.

6 Hyperparameter Tuning

Keras Tuner is used to perform hyperparameter tuning to search for the best hyperparameters for the neural network. The hyperparameters I have tuned are the number of filters in the convolution layers (16, 32, 64), the units of the dense layer (128, 256) and the rate of the dropout layers, that could be a float number between 0.2 and 0.5 with a step of 0.1. `RandomSearch` is a hyperparameter tuning technique that explores different hyperparameter combinations randomly to find the best set of hyperparameters for the model.
I want to maximize the validation accuracy and the max number of trials determines the total number of different hyperparameter combinations that the tuner will explore. In this case, it's set to 3, meaning the tuner will try 3 different sets of hyperparameters.
The tuner uses traning data to assess the performance of different hyperparameter configurations and select the best ones based on the validation results. The number of training epochs for each hyperparameter configuration is set to 3, which means that the tuner will train the model for 3 epochs with each set of hyperparameters, before evaluating the results. After the search is complete, the best hyperparameters can be retrieved and used to train the final model for the classification task.
The actual hyperparameter values that produced the best results during the tuning process are 32 for filters, 128 for units of dense layer and 0.3 for the dropout rate.

7 Results

The model is trained using hyperparameters found through hyperparameter tuning and monitors its performance during training. The callback monitors the validation loss during training and stops the training process early if the validation loss does not improve for a certain number of epochs. In this case, it stops training if there is no improvement for 5 consecutive epochs. The model is trained for 20 epochs. The results on test set are 0.89 for test accuracy and 0.27 for test loss, so 89% of the samples in test dataset are classified correctly by the model. As it can be seen from the figures below, this model does quite a good job. Training loss and validation loss are close to each other and they both decrease, so there is not overfitting. The training and validation accuracy curves are very close and both are improving: this means that the model is likely learning well and generalizing effectively.

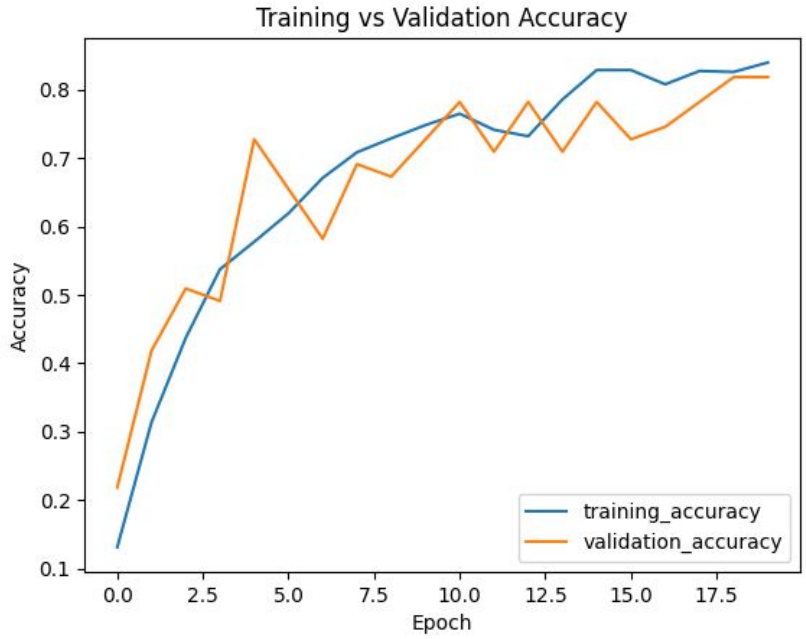

Figure 1: *Traning and valiation accuracy curves*

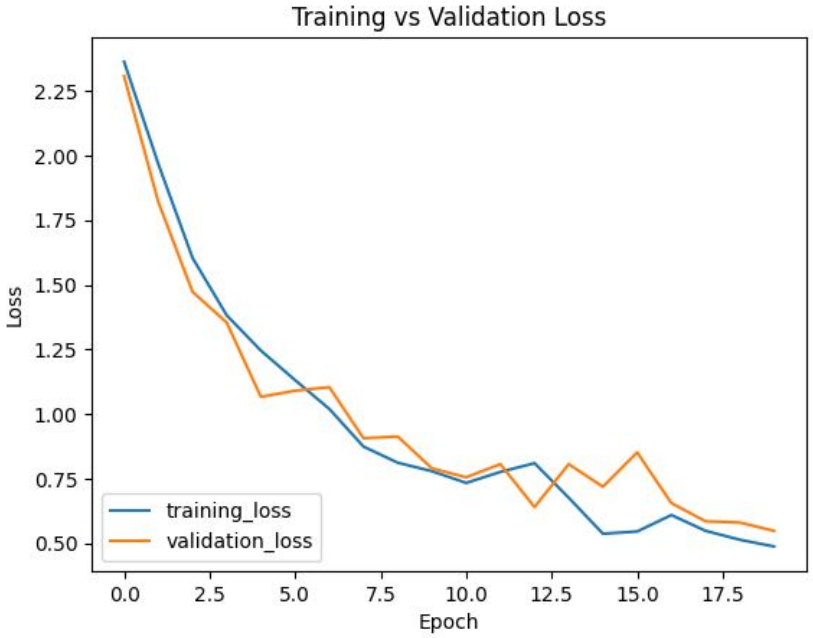

Figure 2: *Training and validation loss curves*

8 Prediction

As the final step, I want to assess the classification performance of the model by visually inspecting how well it predicts the different classes in the dataset. I plot the confusion matrix that has as rows the true labels obtained from the test dataset and as columns the predicted labels generated by the classification model. It can be seen that only four leaves are misclassified.

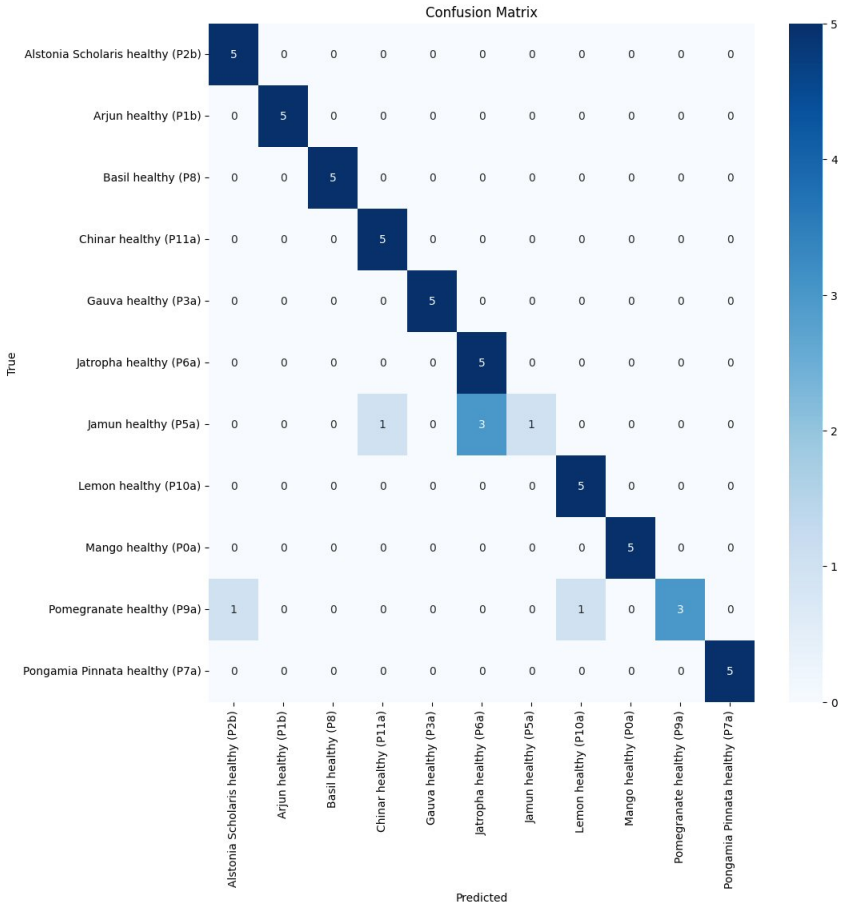

Figure 3: *Confusion matrix*

References

[1] Convolutional neural networks, explained. https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939.

[2] Convolutional neural networks in python with keras. https://www.datacamp.com/tutorial/convolutional-neural-networks-python.

[3] Malchiodi Dario. Algorithms for massive data. University Lectures, 2022-2023.