Leaf classification using convolutional neural networks for Deep learning in pytorch with hyper parameters optimization

Miguel Eduardo Correa Gonzalez
September 10, 2023

1. Introduction

This project was based on the application of convolutional neural networks (CNN) to extract features from images and use them as inputs for training a model that can recognize 11 different types of plant leaves. The model was trained over 50 epochs using a dataset of 2163 training images. The hyperparameters were optimized using a validation dataset of 55 images, and the final results were tested on a separate test dataset of 55 images. All of the images were uniformly distributed among the 11 classes in each dataset.

2. Description of Dataset

The original dataset consisted of 22 classes, with each type of plant having two classes: healthy and diseased. However, since the purpose of this project is to focus solely on classifying the plants without distinguishing between healthy or diseased, it is advisable to use only the data related to the healthy leaves. This approach helps prevent potential confusion for the model caused by shared features among diseased plants.

The dataset comprises a total of 2273 images, which were divided into three groups: training, validation, and testing. The training set consists of 2163 images, while the validation and testing sets each contain 55 images. The dataset can be accessed on Kaggle with the identifier *"csafrit2/plant-leaves-for-image-classification."* To access the dataset, authentication using a username and key is required.

3. Data preprocessing

Resizing:

This step is crucial in this project to ensure optimal performance of the model. One of the initial challenges encountered was dealing with a large number of photos with high resolutions. Although a DataLoader was implemented to enhance data loading efficiency, the images remained too large and contained excessive details (features) that were unnecessary for the project's scope. To address this, the first transformation applied to the downloaded data was resizing the images from (6000x4000) pixels to (224x224) pixels. This reduction in image size helped decrease the total number of features passed to the model as inputs.

To avoid repetitive image processing during each epoch, it was important to perform this resizing step separately from the creation of the data loader.

Data Augmentation:

The next transformation applied to the model involved Data Augmentation, which included random horizontal and vertical flips, as well as rotations applied to the training data. This step aimed to enhance the model's generalization since the orientation of the images does not impact the classification. Additionally, it helped prevent overfitting by incorporating more samples for training.

Data Normalization:

Subsequently, the data was normalized to minimize the influence of outliers, improve generalization, reduce bias caused by color variations, and facilitate faster model convergence.

4. Model

A Convolutional Neural Network (CNN) is a specialized type of artificial neural network designed specifically for processing and analyzing visual data, such as images or videos. CNNs are widely used in computer vision tasks, including image classification, object detection, and image segmentation. The architecture of a CNN consists of two main parts:

- Convolutional Part: The convolutional part is the fundamental component of a CNN and is responsible for extracting features from the input data. It comprises several layers:
- Convolutional Layers: These layers perform convolutions on the input data. Convolutions involve sliding small filters (kernels) over the input image to extract meaningful patterns and features. Each filter learns to detect specific features like edges, textures, or shapes. Consequently, the network learns hierarchical representations of the input data, capturing low-level features in early layers and more complex patterns in deeper layers.

Activation Function: Following the convolution operation, an activation function (ReLU - Rectified Linear Unit) is applied to introduce non-linearity. This step enables the network to learn complex relationships and enhance its expressive capacity.

- Pooling Layers: Pooling layers are used between convolutional layers to downsample the spatial dimensions of the feature maps while preserving important information. This project utilizes max pooling, which selects the maximum value from each region.
- Dense Layer Part: The dense layer part follows the convolutional part and is responsible for making final predictions based on the extracted features. It consists of fully connected layers:
- Flattening: Before passing the feature maps from the convolutional part to the dense layers, the feature maps are flattened into a 1D vector. This operation converts the spatial information into a linear array of values.

- Dense Layers: These layers are standard neural network layers, where each neuron is connected to every neuron in the previous layer. They combine the learned features from the convolutional part to make high-level decisions or classifications. The number of neurons in the last dense layer corresponds to the number of classes in a classification task.

5. Architecture

Below is shown the architecture of the project which consists of 15 layers and is divided into the following sections: feature extractor and classifier

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 32, 224, 224]             896
              ReLU-2         [-1, 32, 224, 224]               0
         MaxPool2d-3         [-1, 32, 112, 112]               0
            Conv2d-4         [-1, 64, 112, 112]          18,496
              ReLU-5         [-1, 64, 112, 112]               0
         MaxPool2d-6           [-1, 64, 56, 56]               0
            Conv2d-7          [-1, 128, 56, 56]          73,856
              ReLU-8          [-1, 128, 56, 56]               0
         MaxPool2d-9          [-1, 128, 29, 29]               0
          Dropout-10               [-1, 107648]               0
           Linear-11                 [-1, 1000]     107,649,000
             ReLU-12                 [-1, 1000]               0
           Linear-13                  [-1, 100]         100,100
             ReLU-14                  [-1, 100]               0
           Linear-15                   [-1, 11]           1,111
================================================================
Total params: 107,843,459
Trainable params: 107,843,459
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 49.13
Params size (MB): 411.39
Estimated Total Size (MB): 461.09
----------------------------------------------------------------
```

Feature Extractor:

- Input: Images with three color channels (RGB).
- Convolutional layers: Three sets of Convolutional Layers are used, each followed by a ReLU activation function and Max Pooling layer.

  - ➢ First 2D Convolutional Layer: 32 filters of size 3x3 with stride 1 and padding 1.
  - ➢ First ReLU
  - ➢ First 2D Max Pooling Layer: Max pooling with a 2x2 window and stride 2.
  - ➢ Second 2D Convolutional Layer: 64 filters of size 3x3 with stride 1 and padding 1.

- ➢ Second ReLU
- ➢ Second 2D Max Pooling Layer: Max pooling with a 2x2 window and stride 2.
- ➢ Third 2D Convolutional Layer: 128 filters of size 3x3 with stride 1 and padding 1.
- ➢ Third ReLU
- ➢ Third 2D Max Pooling Layer: Max pooling with a 2x2 window and stride 2 (padding 1).

Classifier:

- Fully Connected Layers: Three fully connected layers (also known as dense layers) are used as the classifier, each followed by a ReLU activation function.
  - ➢ A dropout layer: Applied after the flattening operation and before the first fully connected layer to prevent overfitting during training.
  - ➢ First Fully Connected Layer: 1000 neurons.
  - ➢ Four ReLU
  - ➢ Second Fully Connected Layer: 100 neurons.
  - ➢ Fifth ReLU
  - ➢ Third Fully Connected Layer: Output layer with 11 neurons, where 11 is the number of classes in the classification task. The final layer provides the raw scores or logits for each class.

6. *Hyperparameter optimization*

This project implements the hyperparameter optimization using the *Optuna library* for a "*Leaves_classifier*" model. Here's a summary of the optimization process:

- Objective Function: The hyper parameter optimization is driven by the *Optimize_hyp* function, which takes an *Optuna trial* object and the training and validation data loaders as inputs. Inside this function, specific hyper parameters are defined for optimization: *alpha, beta, mu0*, and *dropout_rate*.
- Model Initialization: The *Leaves_classifier* model is initialized with the hyper parameters sampled from the Optuna trial, and the model is moved to the appropriate device (e.g., GPU if available).
- Loss and Optimizer: The CrossEntropyLoss is chosen as the criterion for the classification task, and the Stochastic Gradient Descent (SGD) optimizer is used to update the model's weights during training. The learning rate and momentum values are set for the SGD optimizer.
- Learning Rate Scheduler: A learning rate scheduler is created using torch.optim.lr_scheduler.LambdaLR. It adjusts the learning rate based on the epoch number and the hyper parameters mu0, alpha and beta.

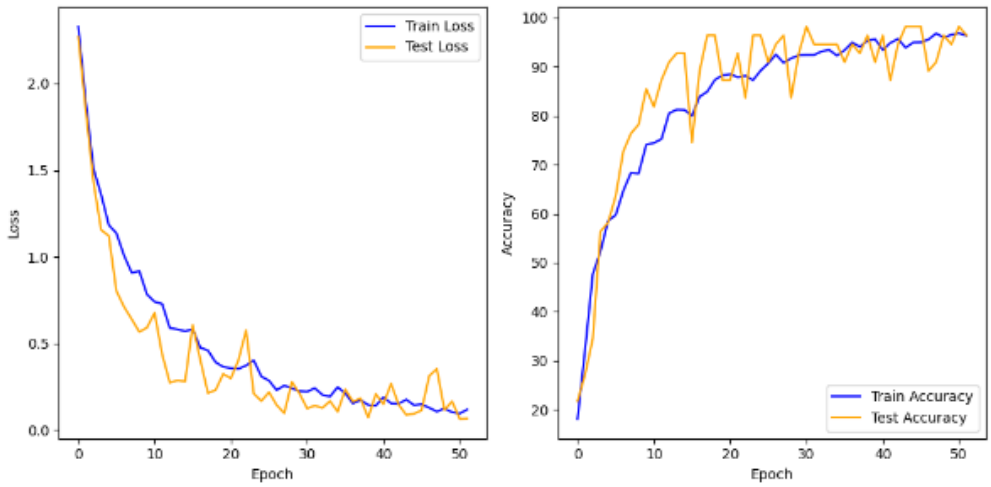
- Training and Validation: The model is trained over a specified number of epochs (6 epochs in this case) using the *Train_Model* and *Validate_Model* functions. During each epoch, the model's performance metrics of *trainin_loss, training_accuracy, validation_loss* and *validation_accuracy.*
- Hyperparameter Optimization: An optimization study is created using *optuna.create_study*, specifying the objective as maximize. The *Optimize_hyp* function is used as the objective function for the study. The *n_trials* parameter sets the number of trials for hyper parameter optimization (in this case, 30 trials).
- Best Hyper parameters: After the optimization study is complete, the best hyperparameters found by *Optuna* are extracted using *study.best_params*.

7. Training:

In this part is defined a training loop for the neural network model "Leaves_classifier" with a specified number of epochs (50 epochs). It initializes the model with the optimized hyperparameters finded in the previous step, sets up the loss function, and defines the optimizer with a learning rate scheduled to change during training using again the optimized hyper parameters. The loop also tracks and prints the training and test losses as well as the accuracies for each epoch. Additionally, it saves the model's state and optimizer when achieving the best test accuracy and terminates the training early if the test accuracy does not improve after 7 epochs when the number of epochs already done is over 30.

8. *Results:*

After the training stage, it can be observed that the best model found during this phase achieves an accuracy of 98.18 percent. Moreover, in the loss and accuracy plots, the performance behavior for both training and testing is similar, with no significant differences that could indicate overfitting of the model. However I notice a particular behavior that is really interesting. During the training, we can notice that the model presents a better performance at some point for the test dataset, even better than the results of the training dataset. This event could be associated with the fact that the dataset of training has been processed with data augmentation techniques in order to augment the variability of the data and also the capability of the model to generalize features in the data considering rotation of the input images. this variability is not present in the test set, whats could contribute to reduce the complexity of the classification and get higher values of accuracy during this evaluation.

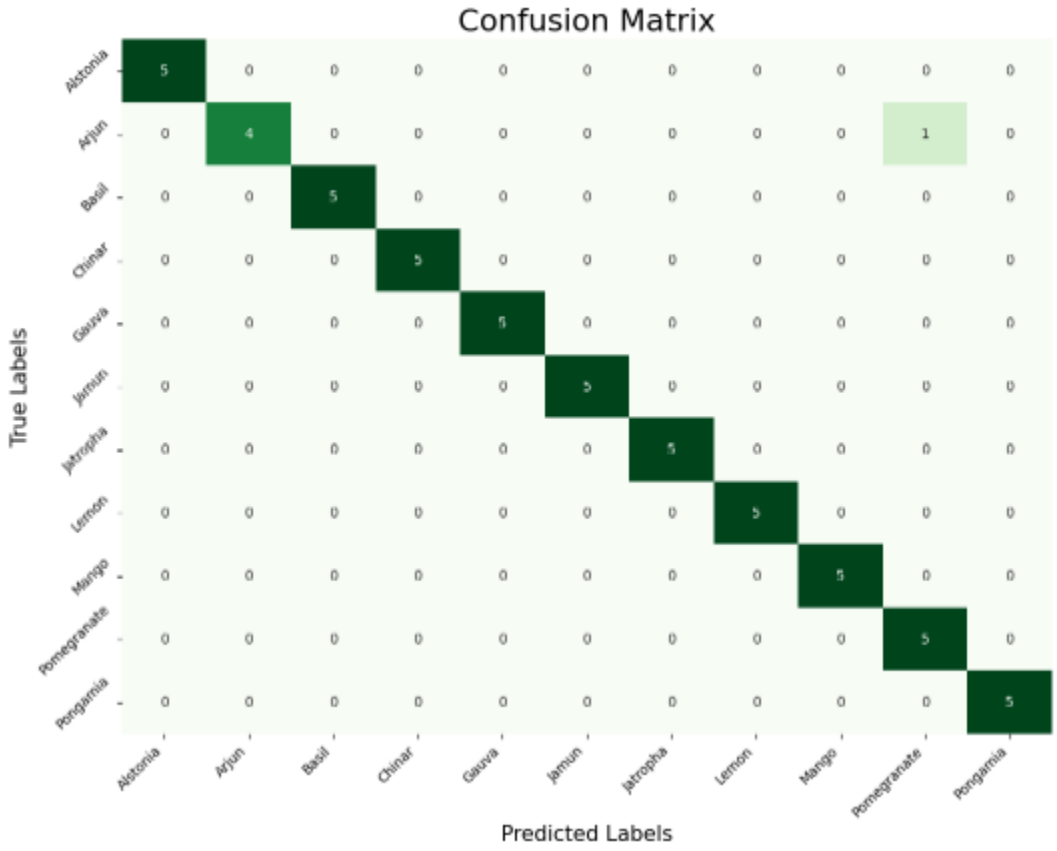


On the other hand, the classification report shows very good performance in the recall, precision, and f1-score metrics for all classes, indicating that there are no classes with systematic errors or that the model performs better only in some of them. For this case the only missclassification was present in the Arjun and Pomegranate, because one of the leave that belong to the Pomegranate was classified as Arjun, affecting on this way the value of recall for Arjun and the Value of accuracy for Pomegranate.

Classification Report:

	precision	recall	f1-score	support
Alstonia	1.00	1.00	1.00	5
Arjun	1.00	0.80	0.89	5
Basil	1.00	1.00	1.00	5
Chinar	1.00	1.00	1.00	5
Gauva	1.00	1.00	1.00	5
Jamun	1.00	1.00	1.00	5
Jatropha	1.00	1.00	1.00	5
Lemon	1.00	1.00	1.00	5
Mango	1.00	1.00	1.00	5
Pomegranate	0.83	1.00	0.91	5
Pongamia	1.00	1.00	1.00	5
accuracy			0.98	55
macro avg	0.98	0.98	0.98	55
weighted avg	0.98	0.98	0.98	55

Finally, the confusion matrix also showed really good results where just one of the classification was wrong in the test dataset of 55 images.



"I/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."