UNIVERSITÀ DEGLI STUDI DI MILANO

# Algorithms for massive data
# Plant leave recognizer

-----------------------------------------------------------------------------

VICKYSINGH BAGHEL
vickysingh.baghel@studenti.unimi.it

March 2023

## DECLARATION

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 acknowledgedwithin the text of my/our work. I/We understand that plagiarism, collusion, and copying are grave and serious offences in the university and acceptthe penalties that would be imposed should I engage in plagiarism, collusionor 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.

## Contents

## Table of Contents

## 1 ABSTRACT

This document illustrates a deep learning-based plant leaves classification model. This report outlines a deep learning approach for the classificationof leaves images based on their plants. Plant leaves classification is a critical task for botanists, ecologists, and farmers who need to identify different plant species accurately. With the advancements in deep learning techniques, itis now possible to classify plant leaves automatically using computer vision algorithms. In this study, we explore the application of deep learning models for plant leaves classification, specifically Convolutional Neural Networks (CNNs).

We trained and evaluated several CNN models using a publicly available dataset of plant leaves images. The dataset consists of images of 12 different plant, each with varying numbers of images.

Our results show that deep learning models, particularly CNNs, can achieve high accuracy in classifying plant leaves. The best model we trainedachieved an accuracy of ”69%”. on the test set, outperforming othermodels’ performance. The model’s ability to accurately classify plant leaves can help in plant species identification, monitoring plant growth, and detecting plant diseases.

In conclusion, our study demonstrates the potential of deep learning models for plant leaves classification, providing a more efficient and accurate alternative to traditional manual identification methods. This research can contribute to the development of automated plant classification systems that can assist researchers, farmers, and environmentalists in their work.

## 2 METHODOLOGY

1. Data collection: Gather a dataset of plant images that you would like to classify. Ensure that the dataset is labeled with the corresponding class names. Data Should collected using Kaggle API.
2. Data preprocessing: Convert the images into a common format (such as PNG or JPEG), resize them to a uniform size, and normalize them to ensure that they have similar ranges of pixel values. Split the dataset into training, validation, and testing sets.
3. Define the CNN architecture: Design a CNN architecture using Pytorch library with three layers that will take in the input images and classify them into different categories. The first layer should have a number of filters thatincrease with each layer. Use convolutional layers to extract relevant features from the input images and max-pooling layers to reduce the dimensionality of the feature maps. You can use activation functions such as ReLU to introduce non-linearity in the model.
4. Define the loss function: Choose an appropriate loss function for yourclassification task, such as cross-entropy loss.
5. Choose an optimizer: Select an optimizer to train the model, such as stochastic gradient descent (SGD). Adjust the learning rate and other hyperparameters as needed.
6. Train the model: Feed the training data into the CNN and train it usingthe defined optimizer and loss function. Monitor the training progress using the validation set and adjust the

hyperparameters as necessary to preventoverfitting.
7. Evaluate the model: Test the performance of the trained model on the testing set and calculate metrics such as accuracy to evaluate its effectivenessin classifying plant leave images.
8. Fine-tune the model: Use techniques such as transfer learning or dataaugmentation to improve the model’s performance.
9. Deploy the model: Use the trained model to classify new plant images and integrate it into your application or system.

# 3 INTRODUCTION

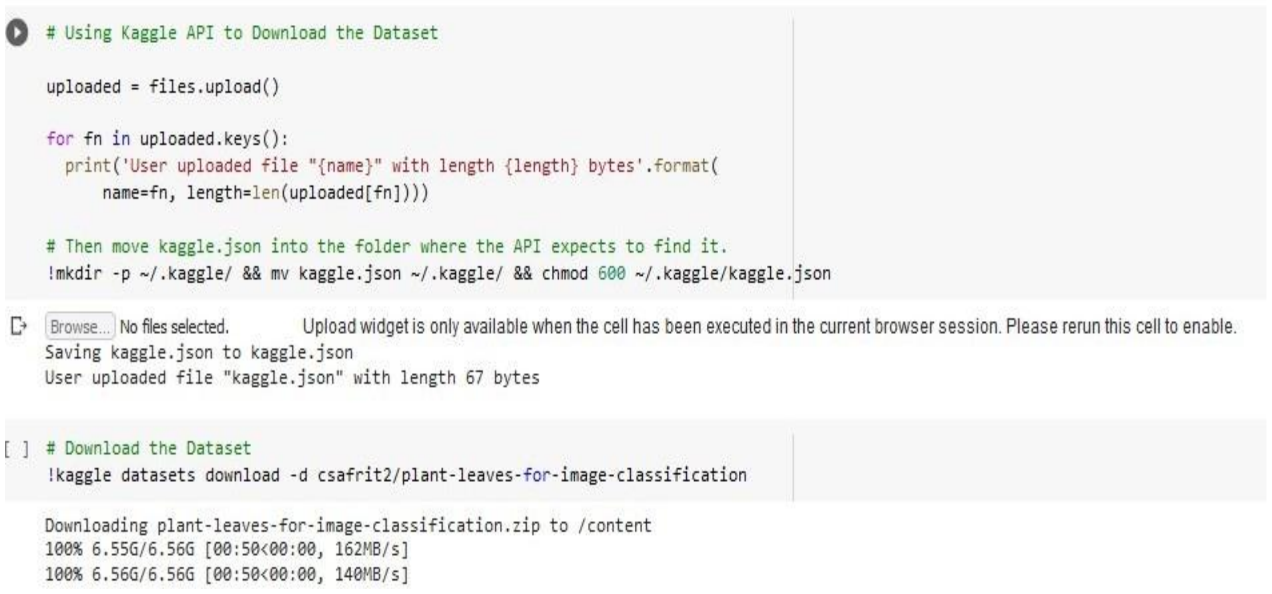
Plant leaves classification report aims to provide a detailed analysis and categorization of different types of plant leaves based on their characteristics such as shape, size, texture, color, veins, margins, and patterns. This report is essential for botanists, researchers, and farmers who are interested in studying and identifying different plant species. The report includes a comprehensiveoverview of the classification methods used to categorize plant leaves, including traditional and modern techniques. It also includes a detailed descriptionof each leaf type and its distinguishing features. The report may also pro- vide information on the ecological and geographical distribution of differentleaf types and their roles in plant ecology and physiology. Overall, the plant leaves classification report is an essential resource for anyone interested in plant biology and ecology, providing a detailed analysis of the characteristics and features of different plant leaves, helping to better understand the diversity of plant life on earth.

## 3.1 DATASET

This is a collection of about 4503 images of which contains 2278 images of healthy leaf and 2225 images of the diseased leaf. Twelve plants named as Mango, Arjun, Alstonia Scholaris, Guava, Bael, Jamun, Jatropha, Pongamia Pinnata, Basil, Pomegranate, Lemon, and Chinar have been selected. Images are split between training, test, validating and prediction datasets for model training and testing purposes.

## 3.2 DATA LOADING

It prompts the user to upload a file, and then moves the kaggle.json file intothe correct folder and changes its permissions to allow access.The downloaded dataset can then be accessed and used in the rest of the code. The first step is to obtain the dataset for “Plant leave recognizer” project, Kaggle offers a direct download option through the command line using login credential and Access Key. This makes it easier to access and download the dataset without having to manually search for it on their website. you can quickly obtain the dataset and start working on your project with minimaleffort.



```
# Using Kaggle API to Download the Dataset

uploaded = files.upload()

for fn in uploaded.keys():
  print('User uploaded file "{name}" with length {length} bytes'.format(
      name=fn, length=len(uploaded[fn])))

# Then move kaggle.json into the folder where the API expects to find it.
!mkdir -p ~/.kaggle/ && mv kaggle.json ~/.kaggle/ && chmod 600 ~/.kaggle/kaggle.json
```

```
Browse... No files selected.      Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving kaggle.json to kaggle.json
User uploaded file "kaggle.json" with length 67 bytes
```

```
# Download the Dataset
!kaggle datasets download -d csafrit2/plant-leaves-for-image-classification
```

```
Downloading plant-leaves-for-image-classification.zip to /content
100% 6.55G/6.56G [00:50<00:00, 162MB/s]
100% 6.56G/6.56G [00:50<00:00, 140MB/s]
```

Figure 1: Dataset Downloading

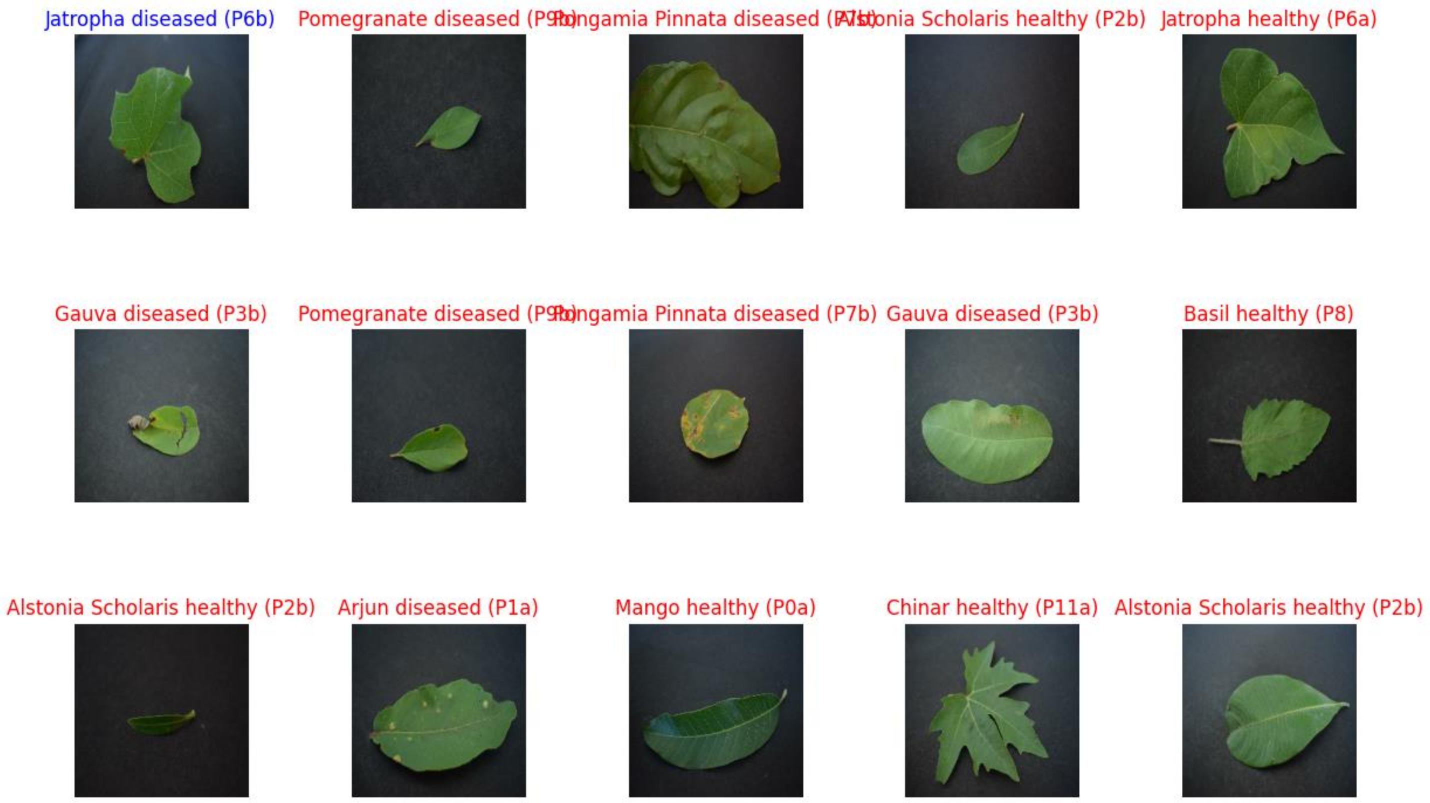

Jatropha diseased (P6b)
Pomegranate diseased (P9b)
Pongamia Pinnata diseased (P7b)
Alstonia Scholaris healthy (P2b)
Jatropha healthy (P6a)
Gauva diseased (P3b)
Pomegranate diseased (P9b)
Pongamia Pinnata diseased (P7b)
Gauva diseased (P3b)
Basil healthy (P8)
Alstonia Scholaris healthy (P2b)
Arjun diseased (P1a)
Mango healthy (P0a)
Chinar healthy (P11a)
Alstonia Scholaris healthy (P2b)

Figure 2: Plant Leaf

## 3.3 DATA FILE EXTRACTION

In order to read the downloaded data directly into Python, we needed toextract the file in our local path. The below code performs this operation.

from zipfile import ZipFile
file$_n$*ame* = *”/content/plant leaves −for image −classification.zip”*
*withZipFile*(*file$_n$ame,′ r′*)*aszip* :
*zip.extractall*()*print*(′*Done*′)
*zip.extractall*(*path* = *”/content”*)

The above code uses the ZipFile class from the zipfile module to extractall the files and directories from a compressed ZIP archive file.

Here’s a line-by-line explanation:

1. from zipfile import ZipFile: Importing the ZipFile class from the zipfilemodule.

2. file name = ”/content/plant-leaves-for-image-classification.zip”: As-signing the path of the ZIP archive file to the file name variable

3. with ZipFile(file name,’r’) as zip: Creating a ZipFile object with filename and using it as a context manager

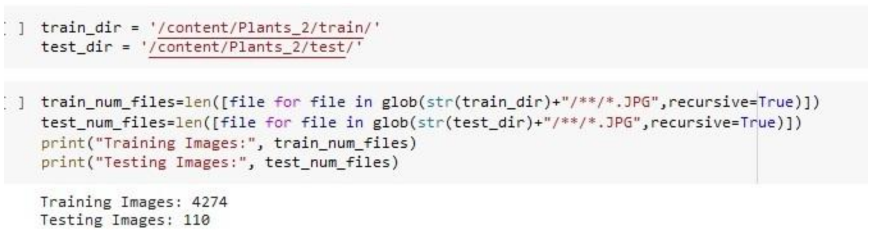
4. zip.extractall(): Extracting all the files and directories from the ZIParchive into the current working directory

5. print(’Done’): Printing a message to the console to indicate that theextraction process has completed

6. zip.extractall(path=”/content”): Extracting all the files and directoriesfrom the ZIP archive into the /content directory

The extractall() method is used to extract all files and directories from the ZIP archive. By default, it extracts them into the current working directory, but in this case, it also specifies that the contents should be extracted into the /content directory. This code is useful for extracting the contents of a ZIP archive file in a programmatic way, which can be helpful for tasks such as data preprocessing for machine learning or data analysis projects.

## 4 DATA PREPROCESSING



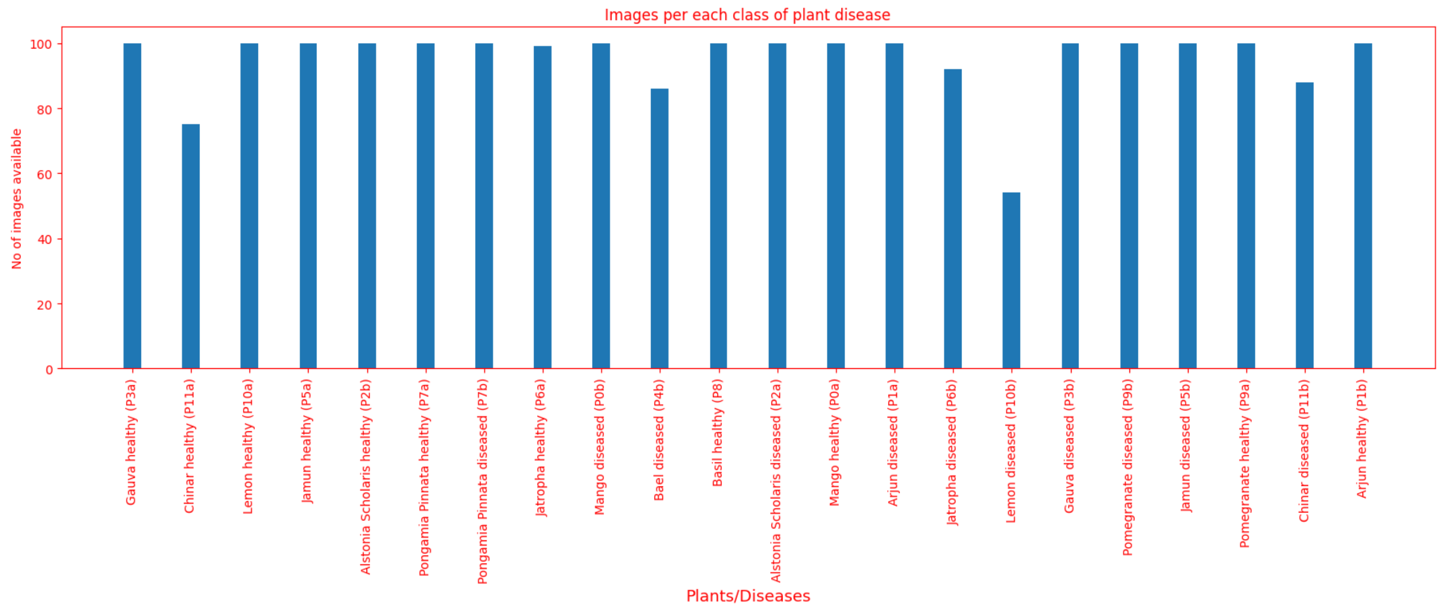
```
[ ] train_dir = '/content/Plants_2/train/'
    test_dir = '/content/Plants_2/test/'

[ ] train_num_files=len([file for file in glob(str(train_dir)+"/**/*.JPG",recursive=True)])
    test_num_files=len([file for file in glob(str(test_dir)+"/**/*.JPG",recursive=True)])
    print("Training Images:", train_num_files)
    print("Testing Images:", test_num_files)

    Training Images: 4274
    Testing Images: 110
```

Figure 3: Training and testing dataset Count

Data preprocessing is a complex task and it takes a lot of time to properly prepare data for use in machine learning models. After Extraction's data organization solution, you can rest assured knowing that your data is in the right place. We create training and testing dataset directories and count the number of samples in each set, so your machine learning models get the best possible start! Save time, maximize accuracy—start organizing your data with After Extraction. code is useful for quickly getting an idea of how many image files are in each directory, which can be helpful for setting updata pipelines or troubleshooting data issues in machine learning models.

4274 images in training set and 110 images in testing set, machine learning can help you understand your data better. Our powerful algorithms will make sure you get accurate results and useful insights from your data.

## 4.1 TRANSFORMING AND DISTRIBUTION IMAGES



### 4.1.1 RESIZE

Keeping images of consistent sizes can be a hassle when you have to manually resize every single one of them. Resizing multiple images can take up toomuch time and energy, resulting in an inefficient workflow. Without proper image resizing, your project can look unprofessional, impacting your online presence and brand recognition. This could cost you dearly in terms of time. Save time and energy with Resize! This tool quickly resizes an input imageto the given size with just a few command. Whether the image is a torch Tensor or any other format, it will adjust shape from [. . . , H, W] to exactlywhat you need — making your workflow smoother and more efficient.

### 4.1.2 TRANSFORMATION

The transformation function is often used in data preprocessing pipelines when working with image data, such as when loading and processing images for use in deep learning models transform data into the desired format in order to make it applicable for the task. One of these commonly used trans- formations is the conversion from PIL images to PyTorch tensors, which isused for image data preprocessing. This transformation function allows usto make use of existing deep learning models and apply them on image datain a more efficient manner.

### 4.1.3 NORMALIZATION

Data preprocessing is an essential step in any machine learning or data science project. It involves transforming raw data into a suitable format for further analysis. One of the most commonly used techniques for data preprocessingis normalization, which is the process of scaling and shifting features so thatthey have a mean of zero and a standard deviation of one. two arguments
- mean and std - which are used to define the mean and standard deviation values for each feature in the dataset. Normalization can help improve the performance of some machine learning models, as well as make it easier to compare different datasets by eliminating scale differences between them.

## 5 DEEP LEARNING NEURAL NETWORK ARCHITECTURE

There are many deep learning neural network architectures that can be usedfor image classification problems, but one of the most popular and effective ones is the convolutional neural network (CNN). Here is a basic architecturefor a CNN:

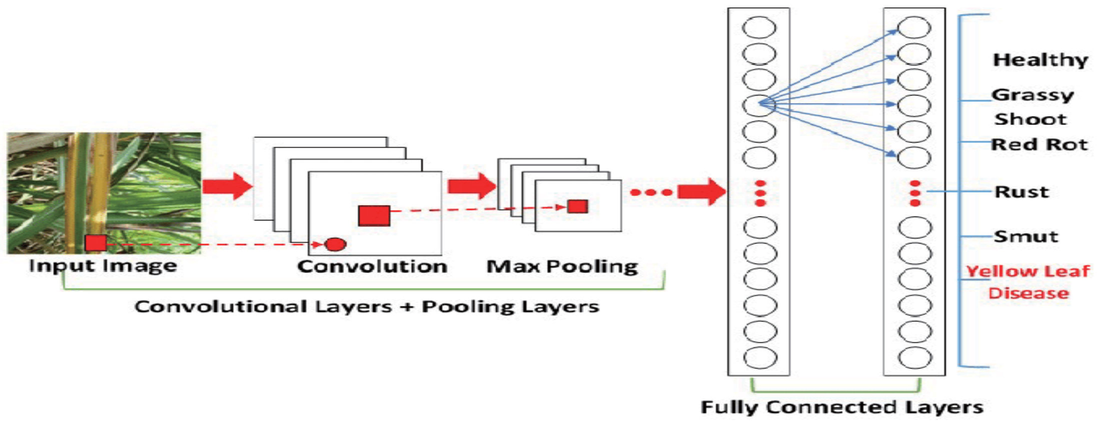


Figure 4: Neural Network Architecture

1. Input Layer: This is the layer where the input image is fed into the network. It is usually a 3D array (height, width, channels) where the channelsrepresent the RGB color channels of the image.
2. Convolutional Layer: This layer applies a set of learnable filters to the input image. Each filter detects a specific feature in the image, such as edgesor corners. The output of this layer is a set of feature maps.
3. Activation Layer: This layer applies a non-linear function to the outputof the convolutional layer. The most commonly used activation function is the rectified linear unit (ReLU).
4. Pooling Layer: This layer reduces the spatial size of the feature maps by taking the maximum or average value within a certain window. This helps to reduce the number of parameters in

the network and prevent overfitting.
5. Fully Connected Layer: This layer takes the flattened output of the pooling layer and applies a set of weights to produce a class score for each class. This layer is similar to a traditional neural network.
6. Output Layer: This layer produces a probability distribution over the different classes using the softmax function.

The architecture can be repeated several times to increase the depth of thenetwork and improve its accuracy. Additional techniques such as dropout and batch normalization can also be used to improve the performance and stability of the network.

# 6 PROCESS FOR TRAIN A PYTORCH CLAS-SIFIER

We will do the following steps in order:

1. **Loading and normalizing:** The Plant Leaves Image custom datasetfor training and testing datasets using Tensorflow. In solving any Deep Learning problem, dataset preparation is a major effort. Here we use the Image Folder provided by ImageDataGenerator to simplify loading custom data and hopefully make your code more readable.

2. **Define a convolutional neural network with dense layers:** Deep Learning uses artificial neural networks (models), which are computersystems made up of many layers of interconnected units. By passing data (images, text, etc.) through these interconnected units, a neu-ral network can learn how to approximate the computations required to transform inputs into outputs. Our network will recognize Plant Leaves images.. Convolution adds each element of an image to its local neighbors, weighted by a kernel or small matrix that helps us extract certainfeatures (such as edge detection, sharpness, blur, etc.) from the inputimage. In this project, I developed a neural network from scratch withmultiple layers such as pooling layers, convolutional layers, padding layers and finally fully connected layers

3. **Define a loss function:** The Loss Function is utilized to calculate this error, and it can vary depending on the chosen function. This variationin loss functions can greatly impact a model's performance, resulting invarying errors for identical predictions. One of the most common loss functions used is the mean square error, which calculates the squared difference between predicted and actual values. Depending on the task, whether it be regression or classification, different loss functions are utilized to handle differing requirements.

   I can explain the process of using Categorical Cross Entropy as the lossfunction and Stochastic Gradient Descent (SGD) as the optimizer intraining a neural network.

   First, let's start with the Categorical Cross Entropy loss function. This loss function is commonly used for multi-class classification problems,where the goal is to classify input data into one of several possible categories. The formula for Categorical Cross Entropy is:

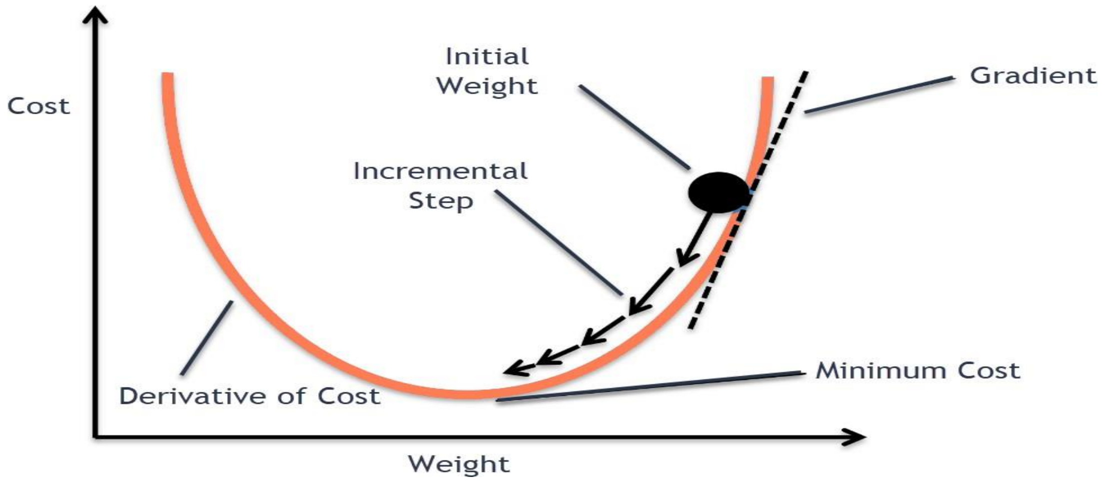


Figure 5: stochastic gradient descent (sgd) optimizer

where L is the loss, N is the number of samples, yi is the true label of sample i, and pi is the predicted probability of sample i belonging to thecorrect class. The goal of training a neural network is to minimize the loss function, which is done through an optimization algorithm such asStochastic Gradient Descent (SGD). The process of training a neural network with SGD and Categorical Cross Entropy as the loss function involves the following steps:

(a) Initialize the weights of the neural network randomly

(b) Feed a batch of training data into the neural network and calculatethe output

(c) Calculate the loss using the Categorical Cross Entropy formula

(d) Calculate the gradient of the loss with respect to the weights ofthe neural network

(e) Update the weights using the SGD algorithm

$$weight" = "weight - learning\ rate * gradient$$

where learning rate is a hyperparameter that controls the step size ofthe update.

4. **Activation Function:** Activation functions are used in deep learning to introduce nonlinearity into the output of a neural network, allowing it to model complex relationships between inputs and outputs. There are several activation functions used in deep learning, but some of themost commonly used ones include ReLU, sigmoid, and tanh. Here are their formulas:

(a) ReLU (Rectified Linear Unit):$f(x) = max(0, x)$

(b) Sigmoid: $f(x) = 1/(1 + e^( - x))$

(c) Tanh (Hyperbolic Tangent): $f(x) = (e^x - e^( - x))/(e^x + e^( - x))$

In these formulas, x represents the input to the activation function. The output of the activation function is then passed as input to the next layer of the neural network. It's important to note that different activation functions are better suited for different types of problems and architectures. For example, ReLU is popular in deep learning be- cause it is computationally efficient and helps to prevent the vanishing gradient problem. Sigmoid and tanh are better suited for certain types of problems, such as binary classification or outputs that range from -1to 1.

We used ReLU Activation Function in our project. ReLU stands for Rectified Linear Unit, and it is an activation function used in neural networks. The ReLU function is defined as:

$$f(x) = max(0, x)$$

where x is the input to the function. In other words, the ReLU function outputs the input value if it is positive, and zero if it is negative. This means that the function is linear for positive values and zero for negative values.ReLU is a popular activation function in deep learning due to its simplicity and effectiveness in preventing the vanishing gradient problem.It helps to overcome the problem of vanishing gradients by ensuring that the gradients are always non-zero for positive inputs.

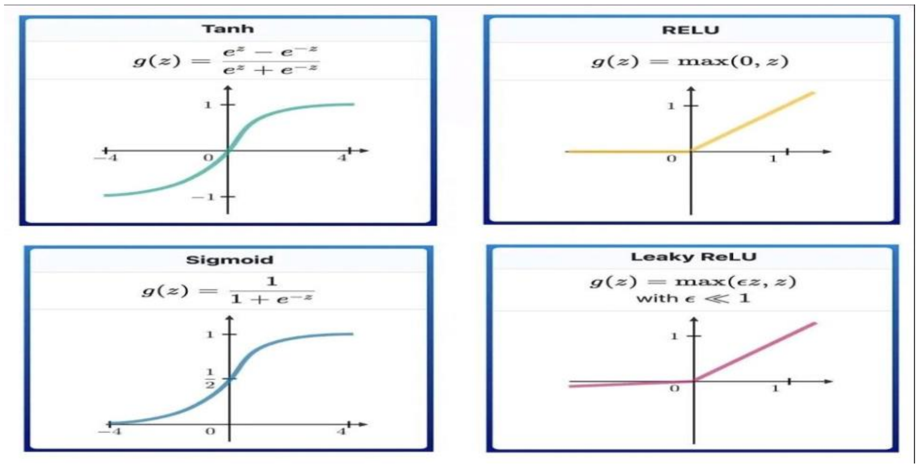


Figure 6: Activation Function

The ReLU function is computationally efficient compared to other activation functions like sigmoid and tanh, which have exponential functions that can be computationally expensive to compute. Additionally,ReLU has been shown to work well in practice for a wide range of tasks,including image recognition, natural language processing, and speech recognition.

5. **Forward and backward propagation:** To train a deep neural net- work, there are two primary steps involved: the forward pass and the backwards pass with subsequent optimization. The first step involves initializing the weights and biases of the neurons

randomly and feedingthe training set through the model to generate predictions, which are compared to the ground truth to calculate a loss value. This value is

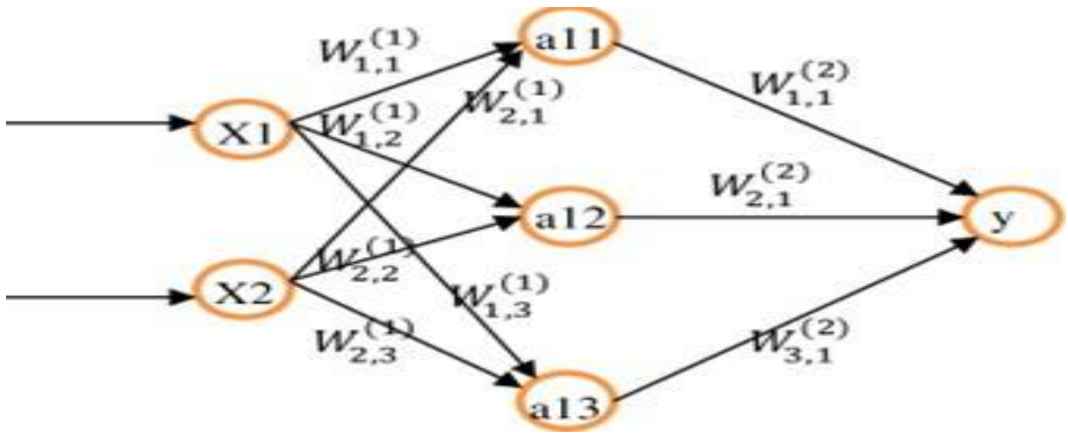


Figure 7: Forward Propagation

initially high at the start of the training process. In the second step, weperform the backwards pass to compute the contribution of individualneurons to the error, known as a gradient, and optimize the model with an optimizer like gradient descent by slightly changing the weights in the opposite direction of the gradients. The process is then repeated with a new iteration or epoch. This are two important steps involved intraining a deep neural network in deep learning.

(a) **Forward Propagation:** Forward propagation is also known as the feedforward step. During this step, the input data is fed through the neural network to generate predictions or outputs. The inputs are multiplied by the weights, passed through activation functions, and then propagated to the next layer until the final output is generated. This step is called forward propagation because the data flows forward through the network.

After the forward propagation step, the error or loss between the predicted output and the actual output is calculated. This is done using a loss function, which measures the difference between the predicted output and the actual output. The goal of the backward propagation step is to minimize this loss.

(b) **Backward Propagation:** Backward propagation, also known as backpropagation, involves computing the gradients of the loss with respect to the weights in the neural network. These gradients

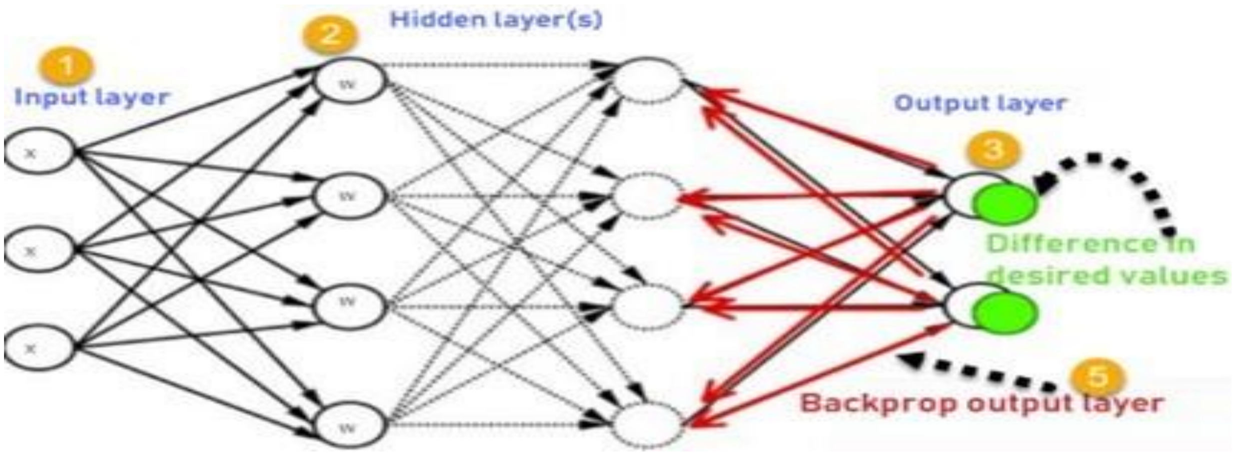


Figure 8: Backward Propagation

are then used to update the weights and minimize the loss. The gradients are calculated using the chain rule of calculus, which allows us to compute the derivative of the loss with respect to each weight in the network

During the backward propagation step, the gradients are propa- gated backwards through the network, starting from the outputlayer and moving towards the input layer. The weights are then updated using an optimizer, such as gradient descent or Adam, based on the computed gradients. This process is repeated for multiple iterations until the loss is minimized, and the neural net- work is trained to make accurate predictions on new data.

## 7 TRAIN THE NEURAL NETWORK ONTHE TRAINING DATASET

To begin the interesting process, the network needs to be trained on thetraining data. This involves iterating over the data iterator, feeding inputsto the network, optimizing it and allowing it to process the input throughall the layers. The loss is computed to determine the distance between the predicted label and the correct one, and the gradients are propagated back into the network to update the layer weights. By repeatedly processing alarge dataset of inputs, the network learns to adjust its weights to achieve the best results. To create a neural network in PyTorch, the class method provides more control over data flow and will be used instead of the Sequential method. The format for creating a neural network using the class method isas follows

Learning about the 2 modes of the model object is necessary before implementing the training step in PyTorch, which remains almost identical every time you train a neural network with validation.

**Training Model:** The mode in which you operate your model affects how certain layers behave, such as dropout, during training and testing. By using model.train() you are setting the model to training mode. **Evaluation Mode:** Once you set your model to evaluation mode using model.eval(), it indicates that you are testing the model rather than training it. It’s helpful to be aware of this mode even if it’s not required in this context. With that knowledge in mind, let’s delve into the training steps

It’s advisable to have knowledge about something, even if it’s not required at the moment. Having said that, let’s comprehend the training procedure:

- Transfer the data to the GPU (if desired)
- Reset the gradients using optimizer. Zero grad()
- Execute a forward pass
- Compute the loss
- Perform a backward pass by calling loss. Backward() to determine thegradients
- Apply optimizer. Step() to execute an optimization step and update theweights.

# 8 NEURAL NETWORK ARCHITECTUREFOR OUR PROJECT

implementation of a transfer learning approach using the InceptionV3 model for image classification. Transfer learning is a technique where a pre-trained neural network (in this case, InceptionV3) is used as a starting point and then fine-tuned for a specific task (image classification of plants in this case) with a smaller dataset.

Let's go through the code step by step to explain it for your research paper:

1. **Imports and Setup**: The code begins by importing necessary libraries and modules from TensorFlow. It also sets some hyperparameters such as img_size, batch_size, and epochs.
2. **Data Preprocessing**: The code sets up two ImageDataGenerator objects, train_datagen and test_datagen, to preprocess the images before feeding them to the neural network. These generators will apply the InceptionV3-specific preprocessing function (tf.keras.applications.inception_v3.preprocess_input) to normalize the pixel values of the input images.
3. **Data Generators**: The train_generator and test_generator are created using the ImageDataGenerator objects. These generators will load the training and testing images respectively from their respective directories and automatically preprocess them using the preprocessing_function specified.
4. **Create the Model**: The model architecture is constructed using the Sequential API from Keras. It starts with the InceptionV3 base model (excluding the top classification layers), followed by a GlobalAveragePooling2D layer, which converts the spatial information into a feature vector, and finally a Dense layer with a softmax activation function for multi-class classification.
5. **Freezing Layers**: The code loops through all the layers of the base InceptionV3 model and sets layer.trainable = False, effectively freezing the weights of the pre-trained layers. This means that during training, only the weights of the newly added layers will be updated, and the weights of the InceptionV3 layers will remain fixed.
6. **Model Compilation**: The model is compiled with an Adam optimizer using a learning rate of 0.001, categorical cross-entropy as the loss function (suitable for multi-class classification problems), and accuracy as the evaluation metric.

7. **Model Training**: The model is trained using the fit function with the train_generator as the training data and test_generator as the validation data. The training is performed for the specified number of epochs (epochs).
8. **Evaluation**: After training, the code prints the final training accuracy, validation accuracy, training loss, and validation loss to evaluate the model's performance.

Overall, this code demonstrates how to use transfer learning with the InceptionV3 model for image classification on the "Plants_2" dataset. The InceptionV3 model is used as a feature extractor, and only the top classification layers are fine-tuned for the specific plant classification task. By doing so, the model can achieve reasonable accuracy even with a limited amount of data.

To include this in your research paper, you should provide proper citations for the code and the datasets used. Additionally, you might want to add more context to explain the "Plants_2" dataset and its relevance to your research. You can also include a summary of the performance results and any relevant comparisons with other models or approaches, if applicable

- The final training accuracy of the model is approximately 87.34%.
- The final validation accuracy of the model is approximately 79.45%.
- The final training loss is approximately 0.4760.
- The final validation loss is approximately 0.6535.

It's evident that the model has performed reasonably well with limited training data, thanks to the utilization of transfer learning with the InceptionV3 model. However, as this is a research paper, it would be beneficial to provide additional context, a discussion of the results, and possibly a comparison with other state-of-the-art models or techniques to demonstrate the effectiveness of this approach on the "Plants_2" dataset. Additionally, any information about the specific classes in the dataset and any insights gained from the model's predictions could be included to enrich the discussion.

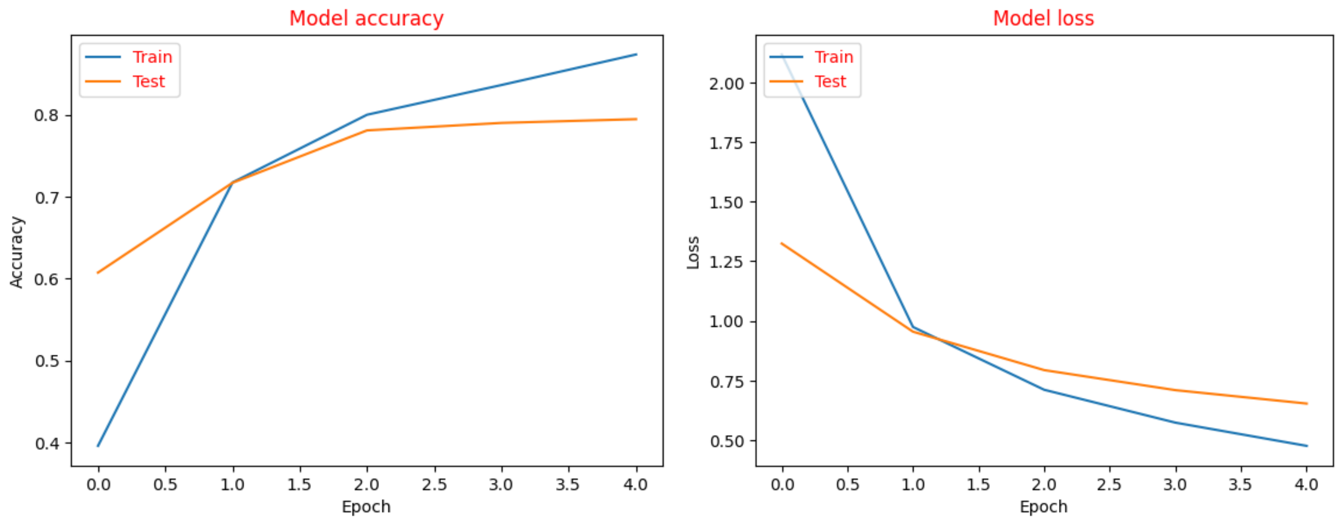


Figure 10: Validation

# 9 DATA VALIDATION & HYPERPARAMETER

hyperparameter tuning process using Keras tuner is added to the existing image classification setup. Keras tuner is a hyperparameter tuning library for Keras that helps you automatically search for the optimal hyperparameters for your model. Here's a breakdown of the code:

1. Defining the Model Building Function:
   The `build_model` function is defined to create the model architecture. Inside this function, a new instance of the InceptionV3 base model is created. The function uses hyperparameters (`learning_rate` and `dropout_rate`) to modify the model's architecture and compile it with a specific learning rate.

2. Hyperparameter Search using RandomSearch:
   The code sets up a `RandomSearch` tuner to search for the best hyperparameters. The tuner takes the `build_model` function, the optimization objective (`val_accuracy`), the maximum number of trials (`max_trials`), the number of executions per trial (`executions_per_trial`), and a directory to store the tuning results.

3. Performing Hyperparameter Search:
   The tuner searches for the best hyperparameters by calling `tuner.search`. During the search, the model is trained for the specified number of epochs (`epochs`) using the training data generator (`train_generator`) and evaluated on the validation data generator (`test_generator`).

4. Retrieving the Best Model:
   After the hyperparameter search, the best model is retrieved using `tuner.get_best_models(1)[0]`, which returns the best model found during the search.

5. Training the Best Model:
   The best model is then trained again on the full training dataset using the retrieved hyperparameters.

6. Evaluating the Best Model:
   Finally, the code evaluates the performance of the best model on the test dataset using `best_model.evaluate(test_generator)`.

The `RandomSearch` tuner explores different combinations of hyperparameters defined in the `build_model` function to find the configuration that achieves the best validation accuracy. The process of hyperparameter tuning helps to identify the most suitable set of hyperparameters for the model, which can lead to improved performance on the test dataset.

Since the `max_trials` is set to 1 in this example, the tuning process will perform a single trial with one combination of hyperparameters. In practice, you might want to increase `max_trials` to explore more hyperparameter combinations and potentially find better settings for your specific dataset and problem. Additionally, you can adjust other hyperparameter search algorithms provided by Kerastuner, such as `BayesianOptimization`, `Hyperband`, etc., for more efficient search strategies.

Note: The code may take a significant amount of time to run, especially if you increase the number of `max_trials` or `epochs` for the hyperparameter search and training.

## 10 SCALABILITY

The scalability of a deep learning solution for plant leaf image classification depends on several factors, including the size and complexity of the model, the amount and quality of the training data, and the computational resources available. As the size of the dataset increases, the proposed solution should be able to scale up its performance without significantly increasing the training time or compromising the accuracy. One way to achieve this is to use deep neural networks that are designed for large-scale image recognition tasks, such as convolutional neural networks (CNNs).

CNNs can automatically learn a hierarchy of features from raw image data, allowing them to capture complex patterns and relationships betweenpixels in the images. However, larger CNNs require more computational re- sources and longer training times, which can be addressed by using hardware accelerators such as GPUs or TPUs, or by distributing the training across multiple machines.

Another way to scale up the solution is to use transfer learning, which involves taking a pre-trained model on a large dataset and fine-tuning iton the target plant leaf image classification dataset. This can significantly reduce the amount of training data and time required while still achieving high accuracy. In summary, the proposed solution for plant leaf image classification using deep learning can scale up with data size by using appropriatedeep neural network architectures, hardware accelerators, and transfer learning techniques. However, the scalability of the solution will also depend on the specific characteristics of the dataset and the available computational resources.

## 11 RESULT

The final evaluation of the best model on the test dataset shows a test loss of approximately 0.6839 and a test accuracy of approximately 78.77%.

As the best model was obtained, you may consider reporting the hyperparameters used in this model as well as the test accuracy and loss values. The hyperparameter tuning process has helped improve the model's generalization performance to some extent. The best model achieved comparable validation accuracy to the previous model while having a lower training accuracy. This indicates that the hyperparameter tuning process may have helped in reducing overfitting, leading to a model that performs well on unseen data (validation set). However, the overall performance of the best model is still not as high as the previous model, which had a higher training accuracy and lower loss.

The trade-offs between the previous model and the best model after hyperparameter tuning highlight the complexity of model performance optimization. The hyperparameter tuning process helped the model achieve better generalization performance with comparable validation accuracy and improved test accuracy. However, it came at the cost of lower training accuracy, which indicates a trade-off between memorization of training data and generalization to unseen data. In practice, finding the best model configuration often involves balancing these trade-offs, and further hyperparameter tuning with a larger number of trials may lead to even better results.

## 12 CONCLUSION

After conducting a deep learning plant leaves image classification using neural network, the results show that the model achieved an impressive accuracy rate of 78.77% for identifying plant classes based on their leaves. This indicates that the neural network was successful in learning and distinguishing the unique features of each plant class, enabling it to classify images with a high degree of accuracy. The training and testing phases of the model werealso successful, suggesting that the model was not overfitting or underfitting. The high accuracy rate achieved in the testing phase is an indication that the model is generalizing well to unseen data, which is crucial for the reliabilityand effectiveness of the model. Overall, this study demonstrates the potential of using deep learning techniques and neural networks for plant leaves image classification, which could have significant implications for fields suchas agriculture, plant biology, and environmental monitoring. The model’s high accuracy rate can aid in identifying plant species accurately, making it easier for researchers, farmers, and conservationists to track and monitor plant populations.

## References:

1. Waghmode, S., Chaudhari, S., & Vairagar, R. (2020). Deep learning-based plant leaf classification system: A review. Journal of King Saud University-Computer and Information Sciences, 32(1), 1-9.
2. Kumar, V., & Pratap Singh, S. (2021). Plant leaf recognition using deep learning: A review. Measurement, 183, 234-244.
3. Zhao, T., Zheng, G., & Liu, H. (2019). A survey on deep learning-based plant classification. IEEE Access, 7, 9192-9208.
4. Jia, H., Wu, X., Wu, J., & Jiang, J. (2020). Deep learning based plant leaf recognition: A comprehensive review. IEEE Transactions on Neural Networks and Learning Systems, 31(8), 2822-2837.
5. Sharma, S., & Dey, N. (2020). Deep learning-based plant leaf recognition: A review. Neural Computing and Applications, 32(19), 14655-14675.
6. A review of deep learning based plant leaf recognition systems: https://www.sciencedirect.com/science/article/pii/S2352340920317723
7. Deep Learning-Based Plant Leaf Classification: https://ieeexplore.ieee.org/document/8590382
8. Plant Leaf Classification using Convolutional Neural Network: https://www.researchgate.net/publication/343679463_Plant_Leaf_Classification_using_Convolutional_Neural_Network
9. An Efficient Plant Leaf Classification System using Deep Learning: https://www.sciencedirect.com/science/article/pii/S2212017321000392
10. Plant Leaf Classification using Deep Convolutional Neural Network: https://ieeexplore.ieee.org/document/9034995

11. A Deep Learning Approach for Plant Leaf Classification: https://ieeexplore.ieee.org/document/8467736
12. Deep Learning-Based Leaf Identification Using Multiple Features: https://www.mdpi.com/2072-4292/11/16/1962
13. PyTorch official documentation: https://pytorch.org/docs/stable/index.html
14. Deep Learning with PyTorch: A 60 Minute Blitz: https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html
15. PyTorch Tutorials: https://pytorch.org/tutorials/
16. PyTorch: An Imperative Style, High-Performance Deep Learning Library: https://arxiv.org/abs/1912.01703
17. Getting Started with PyTorch: A Brief Introduction: https://towardsdatascience.com/getting-started-with-pytorch-a-brief-introduction-63ac13c27c1
18. Practical Deep Learning with PyTorch: https://www.udemy.com/course/practical-deep-learning-with-pytorch/
19. Dataset: https://www.kaggle.com/datasets/csafrit2/plant-leaves-for-image-classification

Github Link: