**IMPLEMENTATION**

**MODULES:**

* Dataset
* Importing the necessary libraries
* Retrieving the images
* Splitting the dataset
* Building the model
* Apply the model and plot the graphs for accuracy and loss
* Accuracy on test set
* Saving the Trained Model

**MODULES DESCSRIPTION:**

**Dataset:**

In the first module of Rice Leaf Disease Detection, we developed the system to get the input dataset. Data collection process is the first real step towards the real development of a machine learning model, collecting data. This is a critical step that will cascade in how good the model will be, the more and better data that we get, the better our model will perform. There are several techniques to collect the data, like web scraping, manual interventions. Our dataset is placed in the project and it’s located in the model folder. The dataset is referred from the popular standard dataset repository kaggle where all the researchers refer it. The dataset consists of 2558 Rice leaf images.

Dataset Kaggle Link:

<https://www.kaggle.com/datasets/jayaprakashpondy/rice-leaf-disease>

**Importing the necessary libraries:**

In this project, we have chosen to utilize the Python programming language as our primary tool. Our initial step involves importing essential libraries to facilitate various tasks. These libraries include Keras, which will aid in constructing the core model, as well as Scikit-Learn (sklearn), which we will employ to partition the data into training and testing sets. Additionally, we will utilize the Python Imaging Library (PIL) to convert the images into numerical arrays. To assist with data manipulation and analysis, we will also make use of other widely-used libraries such as Pandas, NumPy, Matplotlib, and TensorFlow.

**Retrieving the images:**

In this module we will retrieve the images from the dataset and convert them into a format that can be used for training and testing the model. This involves reading the images, resizing them, and normalizing the pixel values. We will retrieve the images and their labels. Then resize the images to (224, 224) as all images should have same size for recognition. Then convert the images into numpy array.

**Splitting the dataset:**

In this module, the image dataset will be divided into training and testing sets. Split the dataset into Train and Test. 80% train data and 20% test data. This will be done to train the model on a subset of the data, validate the model's performance, and test the model on unseen data to evaluate its accuracy. Split the dataset into train and test. 80% train data and 20% test data.

**EfficientNetB5** **| CNN model**

Architecture:

This function returns a Keras image classification model, optionally loaded with weights pre-trained on ImageNet.

For transfer learning use cases, make sure to read the [guide to transfer learning & fine-tuning](https://keras.io/guides/transfer_learning/).

Note: each Keras Application expects a specific kind of input preprocessing. For EfficientNet, input preprocessing is included as part of the model (as a Rescaling layer), and thus [tf.keras.applications.efficientnet.preprocess\_input](https://www.tensorflow.org/api_docs/python/tf/keras/applications/efficientnet/preprocess_input) is actually a pass-through function. EfficientNet models expect their inputs to be float tensors of pixels with values in the [0-255] range.

**Arguments**

include\_top: Whether to include the fully-connected layer at the top of the network. Defaults to True.

weights: One of None (random initialization), 'imagenet' (pre-training on ImageNet), or the path to the weights file to be loaded. Defaults to 'imagenet'.

input\_tensor: Optional Keras tensor (i.e. output of layers.Input()) to use as image input for the model.

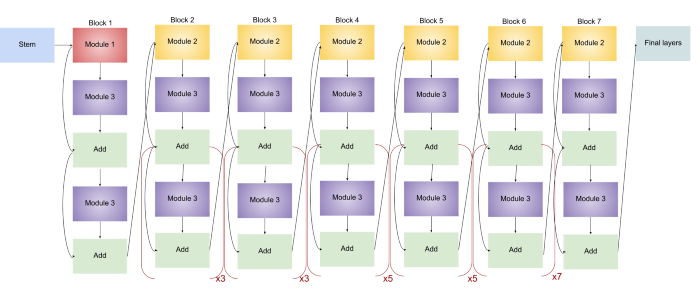
input\_shape: Optional shape tuple, only to be specified if include\_top is False. It should have exactly 3 inputs channels.

pooling: Optional pooling mode for feature extraction when include\_top is False. Defaults to None. - None means that the output of the model will be the 4D tensor output of the last convolutional layer. - avg means that global average pooling will be applied to the output of the last convolutional layer, and thus the output of the model will be a 2D tensor. - max means that global max pooling will be applied.

classes: Optional number of classes to classify images into, only to be specified if include\_top is True, and if no weights argument is specified. Defaults to 1000 (number of ImageNet classes).

classifier\_activation: A str or callable. The activation function to use on the "top" layer. Ignored unless include\_top=True. Set classifier\_activation=None to return the logits of the "top" layer. Defaults to 'softmax'. When loading pretrained weights, classifier\_activation can only be None or "softmax".

**EfficientNet-B5**



**Building the model:**

The concept of convolutional neural networks, are very successful in image recognition. The key part to understand, which distinguishes CNN from traditional neural networks, is the convolution operation. Having an image at the input, CNN scans it many times to look for certain features. This scanning (convolution) can be set with 2 main parameters: stride and padding type. As we see on below picture, process of the first convolution gives us a set of new frames, shown here in the second column (layer). Each frame contains an information about one feature and its presence in scanned image. Resulting frame will have larger values in places where a feature is strongly visible and lower values where there are no or little such features. Afterwards, the process is repeated for each of obtained frames for a chosen number of times. In this project I chose a classic EfficientNetB5  model which contains only two convolution layers.

The latter layer we are convolving, the more high-level features are being searched. It works similarly to human perception. To give an example, below is a very descriptive picture with features which are searched on different CNN layers. As you can see, the application of this model is face recognition. You may ask how the model knows which features to seek. If you construct the CNN from the beginning, searched features are random. Then, during training process, weights between neurons are being adjusted and slowly CNN starts to find such features which enable to meet predefined goal, i.e. to recognize successfully images from the training set.

Between described layers there are also pooling (sub-sampling) operations which reduce dimensions of resulted frames. Furthermore, after each convolution we apply a non-linear function (called **ReLU**) to the resulted frame to introduce non-linearity to the model.

Eventually, there are also fully connected layers at the end of the network. The last set of frames obtained from convolution operations is flattened to get a one-dimensional vector of neurons. From this point we put a standard, fully-connected neural network. At the very end, for classification problems, there is a softmax layer. It transforms results of the model to probabilities of a correct guess of each class

**Apply the model and plot the graphs:**

Once the model is built, it will be applied to the validation set to evaluate its accuracy and loss. The accuracy and loss will be plotted as a function of the number of epochs to visualize the performance of the model. We will compile the model and apply it using fit function. The batch size will be 20. Then we will plot the graphs. We got average Training accuracy of 95.34%

**Accuracy on test set:**

After training and evaluating the model on the validation set, the accuracy of the model will be assessed on the test set. The accuracy on the test set will be an important metric for evaluating the model's performance. We got an accuracy of 97.00% on test set.

**Saving the Trained Model:**

Once you’re confident enough to take your trained and tested model into the production-ready environment, the first step is to save it into a .h5 or . pkl file using a library like pickle .

Make sure you have pickle installed in your environment.

Next, let’s import the module and dump the model into .h5 file.