# Project Title: GrainPalette - A Deep Learning Odyssey In Rice Type Classification Through Transfer Learning

**1. Introduction**

GrainPalette is a deep learning-based image classification project that identifies the type of rice from a given image. It leverages **Transfer Learning** using the MobileNetV2 architecture and provides a Flask-based web application for user interaction.

The primary objective is to automate rice variety identification from images using a pre-trained Convolutional Neural Network (CNN).

**2. Technologies Used**

* Python 3.9
* TensorFlow / Keras
* NumPy, Pandas, Matplotlib
* Scikit-learn
* Flask (for web UI)
* Pillow (for image handling)
* HTML/CSS (for frontend UI)

**3. Dataset**

**Source:** Kaggle - Rice Image Dataset by Murat Koklu  
**Classes:**

* Arborio
* Basmati
* Ipsala
* Jasmine
* Karacada

**Structure after preprocessing:**

rice\_dataset/

├── train/

│ ├── Arborio/

│ ├── Basmati/

│ ├── Ipsala/

│ ├── Jasmine/

│ └── Karacadag/

└── test/

├── Arborio/

├── Basmati/

├── Ipsala/

├── Jasmine/

└── Karacadag/

**4. Project Workflow**

1. **User** uploads a rice image via Flask UI
2. **Flask backend** loads the rice.h5 trained model
3. The image is **preprocessed** and passed to the model
4. **MobileNetV2** (Transfer Learning) extracts features and classifies the image
5. **Prediction** is returned to the UI with the rice type

**5. Steps Involved**

**Data Preprocessing**

* Load and inspect original dataset
* Run split\_rice\_dataset.py to create train and test directories
* Normalize image sizes and apply ImageDataGenerator

**Model Building**

* Use MobileNetV2 (without top layer)
* Add custom Dense layers
* Compile with categorical\_crossentropy and Adam optimizer
* Train the model for 10 epochs
* Save the model as rice.h5

**Web Application (Flask)**

* Upload image through HTML form
* Process image and predict with the model
* Display prediction result

**6. Files and Directories**

project-root/

├── rice\_dataset/ # Train/Test dataset

├── rice.h5 # Trained model

├── train\_rice\_model.py # Model training script

├── split\_rice\_dataset.py # Script to split raw dataset

├── app.py # Flask application

├── templates/

│ ├── index.html # File upload page

│ └── result.html # Prediction display page

**7. Requirements (install via pip)**

pip install tensorflow flask numpy pandas matplotlib scikit-learn pillow

**8. Screenshots & Diagrams**

* Architecture Diagram
* Project Workflow Diagram
* Sample Web UI (Image Upload and Prediction Output)

**9. Conclusion**

GrainPalette offers an intuitive interface and accurate deep learning backend to classify rice grain types efficiently. It demonstrates the potential of Transfer Learning in real-world agricultural applications.

**10. Future Scope**

* Deploy on AWS/Heroku for online access
* Improve model accuracy with more image augmentations
* Add support for mobile image capture and prediction.

# Project workflow

1: Data Collection

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

**Download the dataset**

Collect images of Tomato Leaves. Images are then organized into subdirectories based on their respective names as shown in the project structure.

In this project, we have collected images of 10 types of Tomato Leaf images like Heatly, Spider Mites, Yellow leaf curl, etc. and they are saved in the respective sub directories with their respective names.

You can download the dataset used in this project using the below link

Dataset: <https://www.kaggle.com/datasets/muratkokludataset/rice-image-dataset>

Note: For better accuracy train on more images

   We are going to build our training model on Kaggle as they provide accelerators like GPUs and TPUs.

Note:  The Google Drive notebook will also be provided in the GitHub link mentioned at the end of the project

A new Kaggle Notebook should be created under the dataset link provided.

This notebook will directly link to the Kaggle Dataset.     A screen shot of a computer code

Description automatically generated with low confidence

**Splitting Data on Classes**

Inside the data folder there are several folders for different classes.

A screen shot of a computer code

Description automatically generated with low confidence

**2: Image Preprocessing**

In this milestone we will be improving the image data that suppresses unwilling distortions or enhances some image features important for further processing, although perform some geometric transformations of images like rotation, scaling, translation, etc.

Link : <https://thesmartbridge.com/documents/spsaimldocs/CNNprep.pdf>

**Importing the libraries**

Import the necessary libraries as shown in the image

.

A screenshot of a computer program

Description automatically generated with medium confidence

**Changing size of the images:**

Since the input dimensions of the MobileNet are (224,224,3). We have to resize our images in the same way.

Currently the size of images is (250,250,3).

  
  
A close-up of a computer code

Description automatically generated with low confidence

**Link images to different classes**

Here we have 5 classes and the images need to be labelled with appropriate classes.

A screenshot of a computer code

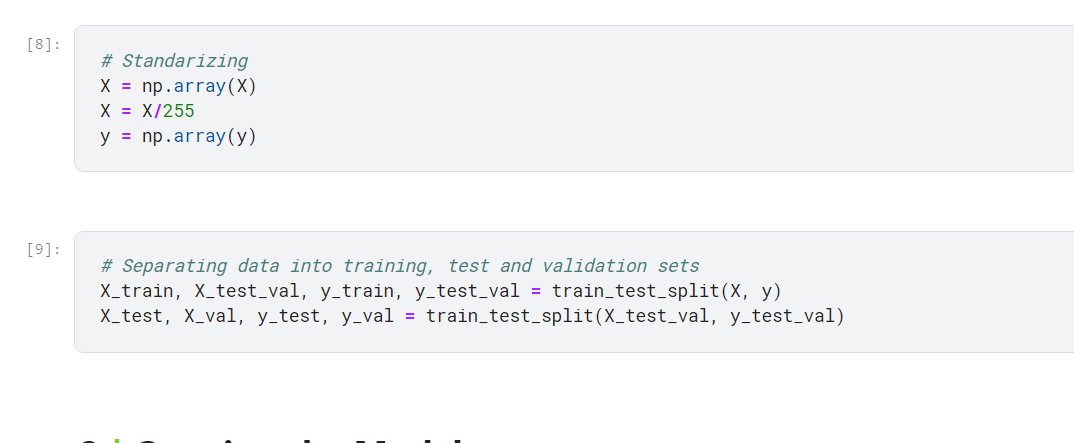
Description automatically generated with medium confidence

A picture containing text, screenshot, font, number

Description automatically generated

**Splitting Data in Train set , Validation and Test set**

We will split the data in training, validation and testing sets.



**Preview of images**

A screen shot of a computer code

Description automatically generated with low confidence

A picture containing text, screenshot, font, design

Description automatically generated

Here we can see that there are 5 different classes, we can see their names above the images. We can see that each disease can be seen directly from the image.

**3: Model Building**

Now it's time to build our model. Let’s use the pre-trained model which is MobileNetv4, one of the convolution neural net (CNN) architecture which is considered as a very good model for Image classification.

**Pre-trained CNN model as a Feature Extractor**

For one of the models, we will use it as a simple feature extractor by freezing all the convolution blocks to make sure their weights don’t get updated after each epoch as we train our own model.

Here, we have considered images of dimension (224, 224, 3).

 Also, we have assigned trainable = False because we are using convolution layer for features extraction and wants to train fully connected layer for our images classification.



**Adding Dense Layer**

A dense layer is a deeply connected neural network layer. It is the most common and frequently used layer.

Let us create a model object named model with inputs as mobile\_net and output as dense layer.

A screenshot of a computer code

Description automatically generated with low confidence

The number of neurons in the Dense layer is the same as the number of classes in the training set. The neurons in the last Dense layer, use softmax activation to convert their outputs into respective probabilities.

Understanding the model is a very important phase to properly use it for training and prediction purposes. Keras provides a simple method, summary to get the full information about the model and its layers.

A screenshot of a computer

Description automatically generated with medium confidence

**Configure the Learning Process**

The compilation is the final step in creating a model. Once the compilation is done, we can move on to the training phase. The loss function is used to find errors or deviations in the learning process. Keras requires a loss function during the model compilation process.

Optimization is an important process that optimizes the input weights by comparing the prediction and the loss function. Here we are using adam optimizer

Metrics are used to evaluate the performance of your model. It is similar to the loss function, but not used in the training process

A picture containing text, screenshot, font

Description automatically generated

**Train the model**

Now, let us train our model with our image dataset. The model is trained for 10 epochs and after every epoch, the current model state is saved if the model has the least loss encountered till that time. We can see that the training loss decreases in almost every epoch.

fit functions used to train a deep learning neural network

Arguments:

* Epochs: an integer and number of epochs we want to train our model for.
* validation\_data can be either:

                      - an inputs and targets list

                      - a generator

                      - an inputs, targets, and sample\_weights list which can be used to evaluate

                        the loss and metrics for any model after any epoch has ended

A screenshot of a computer

Description automatically generated with medium confidence

**Testing the Model**

Model testing is the process of evaluating the performance of a deep learning model on a dataset that it has not seen before. It is a crucial step in the development of any machine learning model, as it helps to determine how well the model can generalize to new data.

A picture containing text, font, screenshot

Description automatically generated

A screenshot of a computer code

Description automatically generated with low confidence

**Visualizing Accuracy and Loss**

The accuracy and loss can be visualized to check the correlation between the epochs and loss or epochs and accuracy.

A picture containing text, font, screenshot

Description automatically generated

A graph with red and blue lines

Description automatically generated with low confidence

A picture containing text, font, line, screenshot

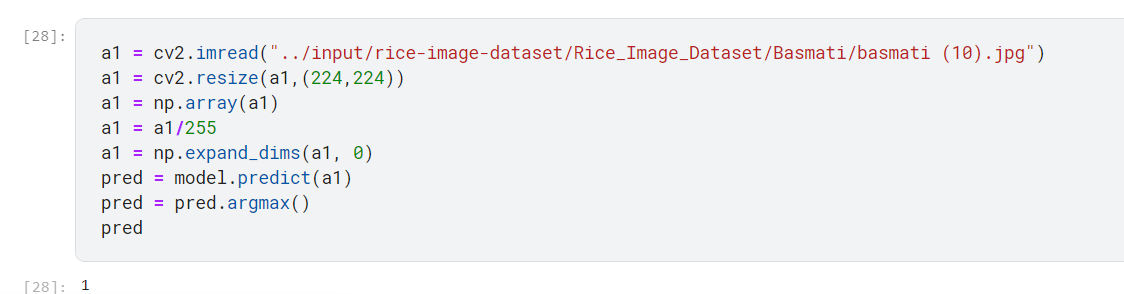
Description automatically generated

A graph with red and blue lines

Description automatically generated with low confidence

**Testing the Model:**

Here we will take a image of basmati rice and check what our model predicts for the same.

 A screen shot of a computer code

Description automatically generated with low confidence

As we can see our model has predicted the rice to be Basmati rice, means our model is giving correct predictions.

**Save the Model**

The model is saved as rice.h5

A .h5 file is a data file saved in the hdf5 format. It contains multidimensional arrays of scientific data.

A picture containing text, screenshot, font, white

Description automatically generated

**5: Application Building**

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the user where they have to upload the image for predictions. The entered image is given to the saved model and prediction is showcased on the UI.

This section has the following tasks

* Building HTML Pages
* Building server side script

**Building Html Pages:**

For this project create 3 HTML files namely

* Index.html
* Details.html
* Results.html

Let’s see how our index.html page looks like:

A screenshot of a computer

Description automatically generated

All the sections below are included in the index.html page.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

When you click on the predict button, it will display the below page. You can test the model by passing a image

A screenshot of a computer

Description automatically generated

**Build Python code:**

mport the libraries

A screen shot of a computer program

Description automatically generated with low confidence

Loading the saved model and initializing the flask app



Render HTML pages:

A screen shot of a computer program

Description automatically generated with low confidence

Once we uploaded the file into the app, then verifying the file uploaded properly or not. Here we will be using declared constructor to route to the HTML page which we have created earlier.

In the above example, ‘/’ URL is bound with index.html function. Hence, when the home page of the web server is opened in browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.

A screen shot of a computer program

Description automatically generated with low confidence

Here we are routing our app to predict function. This function retrieves all the values from the HTML page using Post request. That is stored in variable image and then converted into an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will rendered to the text that we have mentioned in the result.html page earlier.

Main Function:

A picture containing text, font, screenshot, clock

Description automatically generated

**Run the application**

* Open the Anaconda prompt from the start menu.
* Navigate to the folder where your Python script is.
* Now type the “python app.py” command.
* Navigate to the localhost where you can view your web page.
* Click on the predict button from the top right corner, enter the inputs, click on the submit button, and see the result/prediction on the web.

A screen shot of a computer

Description automatically generated with medium confidence

The home page looks like this. When you click on the button “Predict”, you’ll be redirected to the predict section

A screenshot of a computer

Description automatically generated

click on PREDICT button

Input 1:

A screenshot of a computer

Description automatically generated

Once you upload the image and click on upload button, the output will be displayed in the below page

Output1:

A screenshot of a computer

Description automatically generated with medium confidence

Input2:

A screenshot of a computer

Description automatically generated

Output2 :

A screenshot of a computer

Description automatically generated with medium confidence