GrainPalette - A Deep Learning Odyssey In Rice Type Classification Through Transfer Learning

System Required:

Windows 8 machine Install with two web browser Bandwidth of 30mbps

The Rice Type Identification AI model provides a solution for farmers and agriculture enthusiasts to identify various types of rice grains quickly and accurately. By uploading an image of a rice grain and clicking the submit button, users receive predictions for the probable type of rice, enabling informed decisions on cultivation practices such as water and fertilizer requirements. Built using Convolutional Neural Networks (CNN) and employing transfer learning with MobileNetv4, this model offers reliable classification of up to five different types of rice, catering to the needs of farmers, agriculture scientists, home growers, and gardeners.

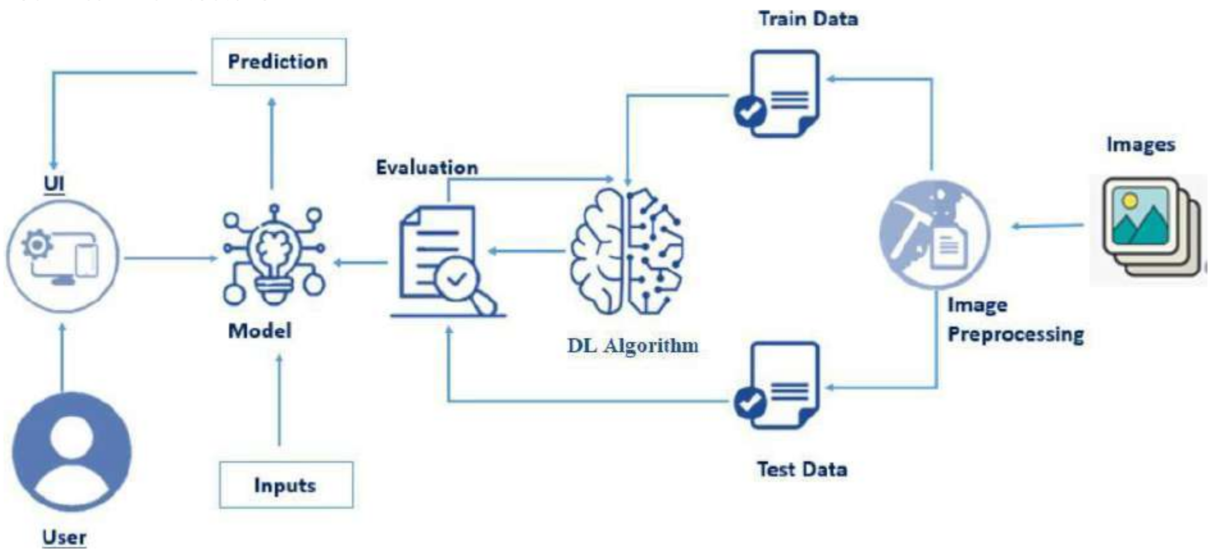
Scenarios:

Farmers' Crop Planning: Farmers can use the Rice Type Identification AI model to plan their crop cultivation strategies effectively. Before planting, they can upload images of rice grains from their seed stock to determine the specific types of rice they possess. Based on the predictions provided by the model, farmers can adjust their agricultural practices such as irrigation schedules, fertilization methods, and pest management strategies tailored to the requirements of each rice variety.

Research and Agricultural Extension Services: Agriculture scientists and extension workers utilize the AI model to assist farmers in identifying rice varieties accurately. During field visits or research trials, they can capture images of rice grains and input them into the model to obtain rapid classifications. This facilitates data collection for research studies, variety testing, and extension programs, ultimately enhancing productivity and sustainability in rice cultivation.

Home Gardening and Education: Home growers and gardening enthusiasts leverage the AI model to learn about different rice varieties and enhance their gardening skills. By uploading images of rice grains from seed packets or harvested crops, they can explore the diversity of rice types and understand their unique characteristics. This fosters learning and appreciation for agricultural biodiversity, promoting sustainable practices in home gardening and education initiatives.

Technical Architecture:



Project Objectives

By the end of this project, you'll understand:

- Preprocessing of images and augmentation of images.
- Applying Transfer learning algorithms on the dataset.
- How deep neural networks detect the disease.
- You will be able to know how to find the accuracy of the model.
- You will be able to Build web applications using the Flask framework.

Project Flow

- The user interacts with the UI (User Interface) to choose the image.
- The chosen image analyzed by the model which is integrated with flask application.
- The MobileNet Model analyzes the image, then the prediction is showcased on the Flask UI.

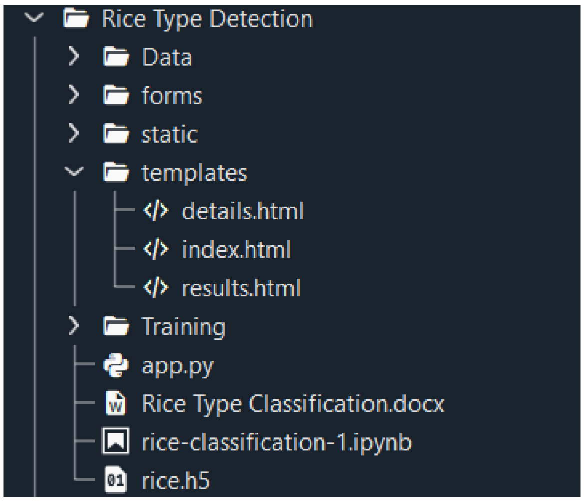
To accomplish this, we must complete all the activities and tasks listed below

- Data Collection.
    - Create a Train and Test path.
- Data Pre-processing.
- Import the required library
- Configure ImageDataGenerator class
- ApplyImageDataGenerator functionality to Trainset and Test set

- Model Building
    - Pre-trained CNN model as a Feature Extractor
    - Adding Dense Layer
    - Configure the Learning Process
    - Train the model
    - Save the Model
    - Test the model
- Application Building
    - Create an HTML file
    - Build Python Code

Project Structure

Create a Project folder which contains files as shown below



```
Rice Type Detection
    Data
    forms
    static
    templates
        details.html
        index.html
        results.html
    Training
    app.py
    Rice Type Classification.docx
    rice-classification-1.ipynb
    rice.h5
```

- Static folder contains css files
- Template folder contains all 3 HTML pages.
- Data folder contains Training and Validation images
- Training file consist of train.ipynb , rice.h5 model

- ## 1: Data Collection
- There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.
-

  1: 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.

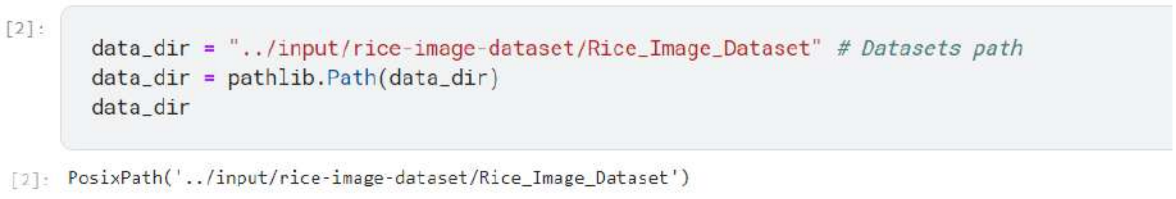
- In this project, we have collected images of 10 types of Tomato Leaf images like Heatly, Spider Mites, Yellow leaf curl, etc. and they are saved in the respective sub directories with their respective names.
- You can download the dataset used in this project using the below link
- Dataset: https://www.kaggle.com/datasets/muratkokludataset/rice-image-dataset
-

- 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
-

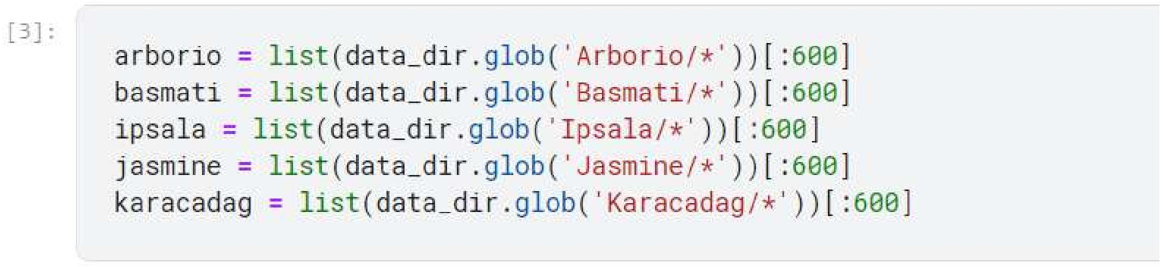
- A new Kaggle Notebook should be created under the dataset link provided.
- This notebook will directly link to the Kaggle Dataset.



```
[2]:
data_dir = "../input/rice-image-dataset/Rice_Image_Dataset"  # Datasets path
data_dir = pathlib.Path(data_dir)
data_dir

[2]: PosixPath('../input/rice-image-dataset/Rice_Image_Dataset')
```

2: Splitting Data on Classes

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

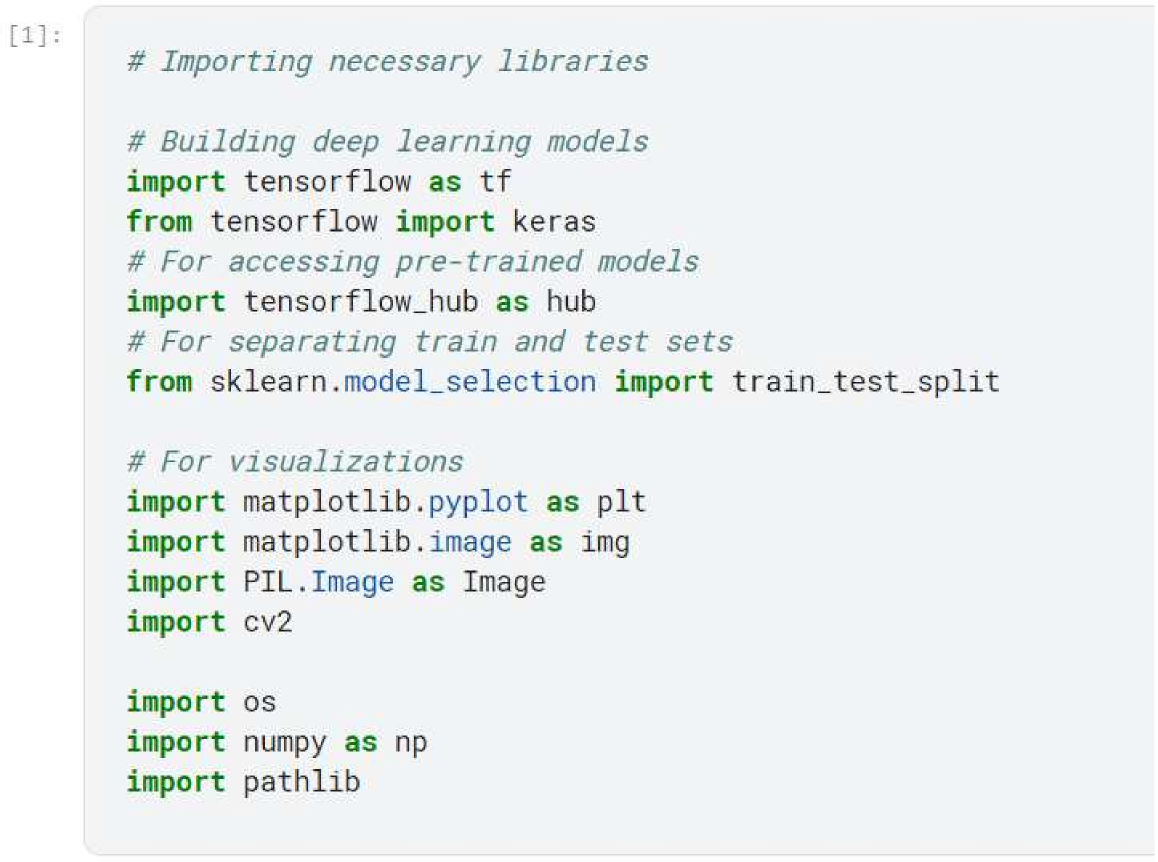


```
[3]:
arborio = list(data_dir.glob('Arborio/*'))[:600]
basmati = list(data_dir.glob('Basmati/*'))[:600]
ipsala = list(data_dir.glob('Ipsala/*'))[:600]
jasmine = list(data_dir.glob('Jasmine/*'))[:600]
karacadag = list(data_dir.glob('Karacadag/*'))[:600]
```

-

1: Importing the libraries

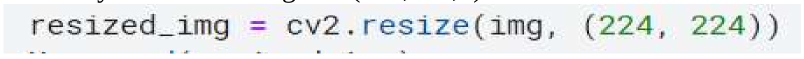
- Import the necessary libraries as shown in the image
- .



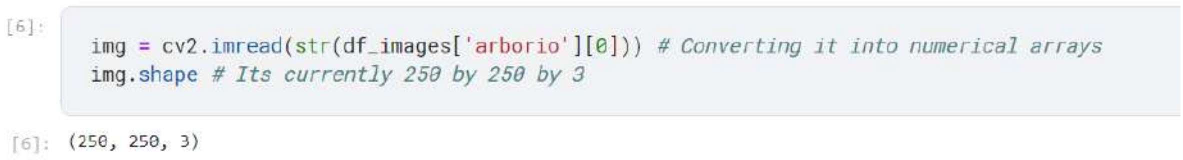
```
[1]:
# Importing necessary libraries

# Building deep learning models
import tensorflow as tf
from tensorflow import keras
# For accessing pre-trained models
import tensorflow_hub as hub
# For separating train and test sets
from sklearn.model_selection import train_test_split

# For visualizations
import matplotlib.pyplot as plt
import matplotlib.image as img
import PIL.Image as Image
import cv2

import os
import numpy as np
import pathlib
```

2: 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).



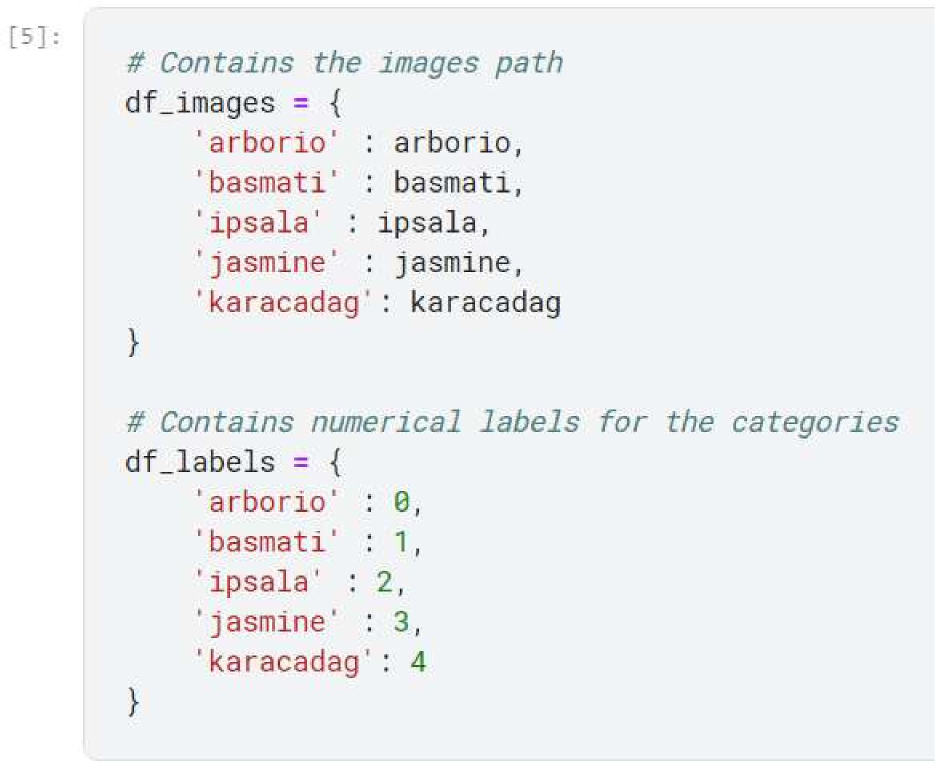
```
resized_img = cv2.resize(img, (224, 224))
```



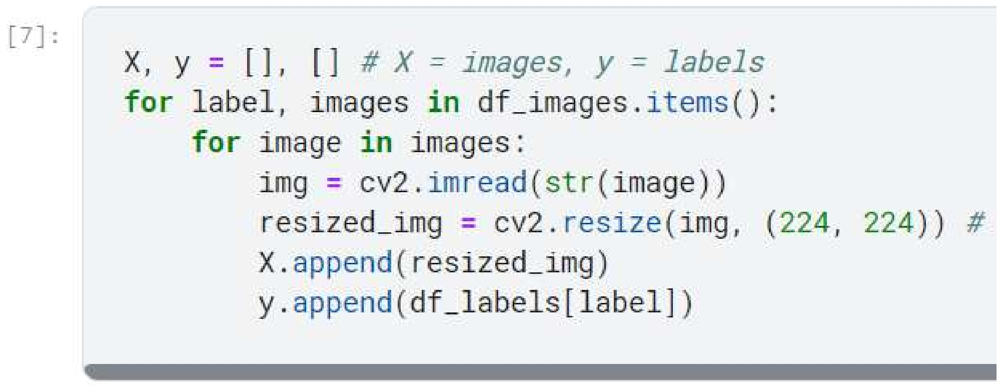
```
[6]:
img = cv2.imread(str(df_images['arborio'][0])) # Converting it into numerical arrays
img.shape # Its currently 250 by 250 by 3

[6]: (250, 250, 3)
```

3: Link images to different classes

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

- 

```
[5]:
# Contains the images path
df_images = {
    'arborio' : arborio,
    'basmati' : basmati,
    'ipsala' : ipsala,
    'jasmine' : jasmine,
    'karacadag': karacadag
}

# Contains numerical labels for the categories
df_labels = {
    'arborio' : 0,
    'basmati' : 1,
    'ipsala' : 2,
    'jasmine' : 3,
    'karacadag': 4
}
```

- 

```
[7]:
X, y = [], [] # X = images, y = labels
for label, images in df_images.items():
    for image in images:
        img = cv2.imread(str(image))
        resized_img = cv2.resize(img, (224, 224)) #
        X.append(resized_img)
        y.append(df_labels[label])
```

4: Splitting Data in Train set , Validation and Test set

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

[8]:
```
# Standarizing
X = np.array(X)
X = X/255
y = np.array(y)
```

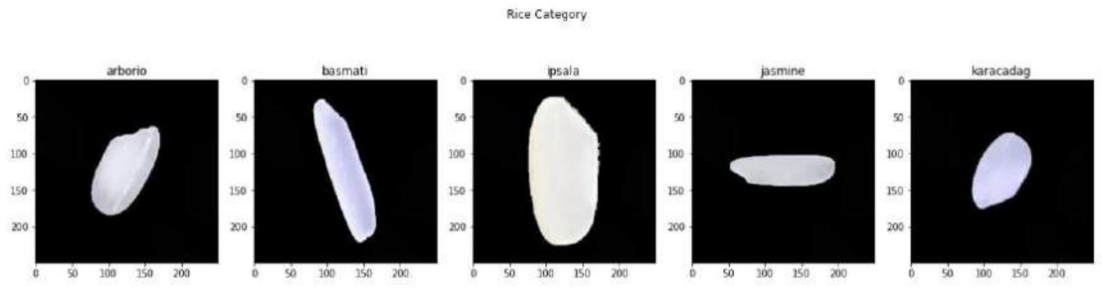
[9]:
```
# Separating data into training, test and validation sets
X_train, X_test_val, y_train, y_test_val = train_test_split(X, y)
X_test, X_val, y_test, y_val = train_test_split(X_test_val, y_test_val)
```

5: Preview of images

[4]:
```
fig, ax = plt.subplots(ncols=5, figsize=(20,5))
fig.suptitle('Rice Category')
arborio_image = img.imread(arborio[0])
basmati_image = img.imread(basmati[0])
ipsala_image = img.imread(ipsala[0])
jasmine_image = img.imread(jasmine[0])
karacadag_image = img.imread(karacadag[0])

ax[0].set_title('arborio')
ax[1].set_title('basmati')
ax[2].set_title('ipsala')
ax[3].set_title('jasmine')
ax[4].set_title('karacadag')


ax[0].imshow(arborio_image)
ax[1].imshow(basmati_image)
ax[2].imshow(ipsala_image)
ax[3].imshow(jasmine_image)
ax[4].imshow(karacadag_image)
```

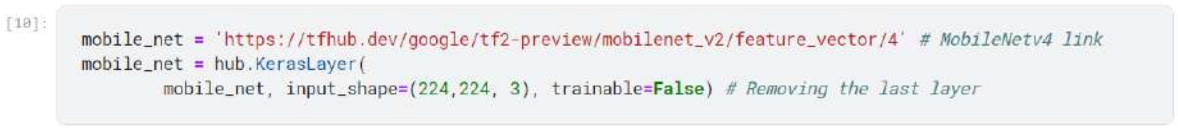
```
[4]: <matplotlib.image.AxesImage at 0x7f55e1c0d6d0>
```

```
[4]: <matplotlib.image.AxesImage at 0x7f55e1c0d6d0>
```



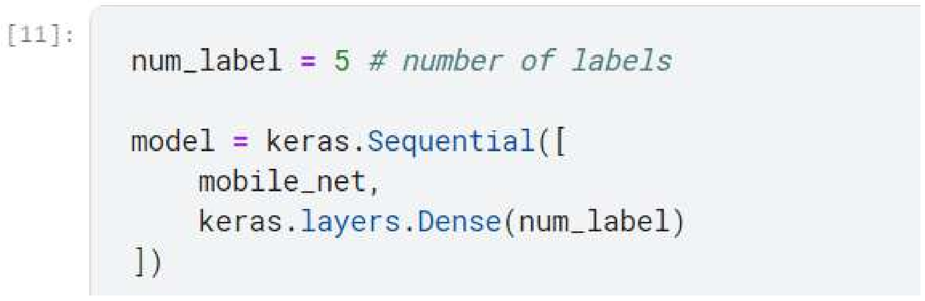
- Here we can see that there are 5 different classes, we can see their names above the images. We can see that each disease can be seen directly from the image.

- ## 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.
- ## 1: 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.



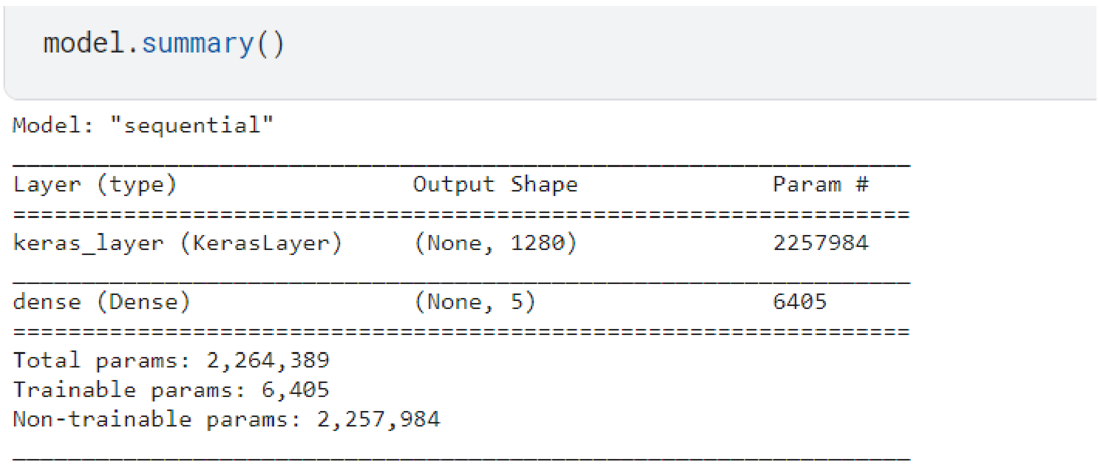
```
[10]:
mobile_net = 'https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4' # MobileNetv4 link
mobile_net = hub.KerasLayer(
        mobile_net, input_shape=(224,224, 3), trainable=False) # Removing the last layer
```

-
- ## 2: 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.



```
[11]:
num_label = 5 # number of labels

model = keras.Sequential([
    mobile_net,
    keras.layers.Dense(num_label)
])
```

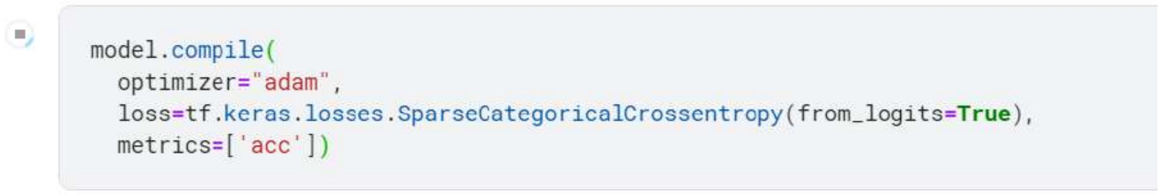
-
-
- The number of neurons in the Dense layer is the same as the number of classes in the training set. The neurons in the last Dense layer, use softmax activation to convert their outputs into respective probabilities.
- Understanding the model is a very important phase to properly use it for training and prediction purposes. Keras provides a simple method, summary to get the full information about the model and its layers.
-



```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
keras_layer (KerasLayer)     (None, 1280)              2257984
_________________________________________________________________
dense (Dense)                (None, 5)                 6405
=================================================================
Total params: 2,264,389
Trainable params: 6,405
Non-trainable params: 2,257,984
_________________________________________________________________
```

-
- ## 3: 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



```
model.compile(
  optimizer="adam",
  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=['acc'])
```

-
- **4: 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
-

```
[14]:  history = model.fit(X_train, y_train, epochs=10, validation_data=(X_val, y_val))
```

```
Epoch 1/10
71/71 [==============================] - 13s 47ms/step - loss: 0.5700 - acc: 0.8333 - val_loss: 0.2491 - val_acc: 0.9362
Epoch 2/10
71/71 [==============================] - 2s 32ms/step - loss: 0.1594 - acc: 0.9680 - val_loss: 0.1619 - val_acc: 0.9574
Epoch 3/10
71/71 [==============================] - 2s 31ms/step - loss: 0.1114 - acc: 0.9769 - val_loss: 0.1280 - val_acc: 0.9521
Epoch 4/10
71/71 [==============================] - 2s 31ms/step - loss: 0.0853 - acc: 0.9813 - val_loss: 0.1157 - val_acc: 0.9628
Epoch 5/10
71/71 [==============================] - 2s 31ms/step - loss: 0.0719 - acc: 0.9840 - val_loss: 0.0968 - val_acc: 0.9681
Epoch 6/10
71/71 [==============================] - 2s 31ms/step - loss: 0.0589 - acc: 0.9862 - val_loss: 0.0903 - val_acc: 0.9681
Epoch 7/10
71/71 [==============================] - 3s 36ms/step - loss: 0.0518 - acc: 0.9889 - val_loss: 0.0817 - val_acc: 0.9734
Epoch 8/10
71/71 [==============================] - 2s 31ms/step - loss: 0.0452 - acc: 0.9911 - val_loss: 0.0774 - val_acc: 0.9734
Epoch 9/10
71/71 [==============================] - 2s 31ms/step - loss: 0.0411 - acc: 0.9889 - val_loss: 0.0750 - val_acc: 0.9734
Epoch 10/10
71/71 [==============================] - 2s 31ms/step - loss: 0.0377 - acc: 0.9916 - val_loss: 0.0754 - val_acc: 0.9681
```

-
- ## 5: 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.

```
[15]:  model.evaluate(X_test,y_test)

18/18 [==============================] - 1s 35ms/step - loss: 0.0943 - acc: 0.9751
[15]:  [0.09426731616258621, 0.9750889539718628]
```

-

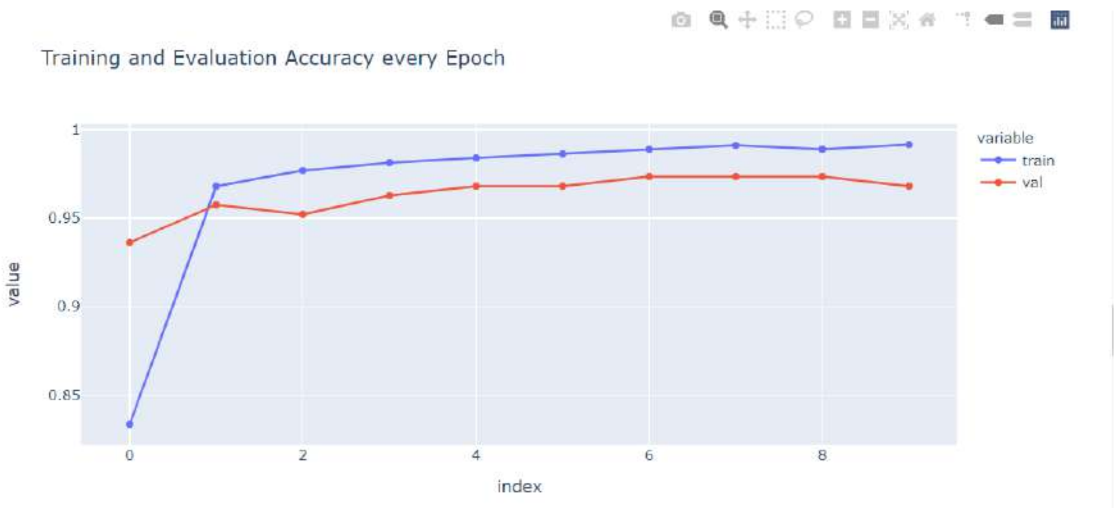
```
[16]:  from sklearn.metrics import classification_report

       y_pred = model.predict(X_test, batch_size=64, verbose=1)
       y_pred_bool = np.argmax(y_pred, axis=1)

       print(classification_report(y_test, y_pred_bool))
```

```
9/9 [==============================] - 1s 63ms/step
              precision    recall  f1-score   support

           0       0.96      0.97      0.96       118
           1       0.96      0.99      0.98        99
           2       1.00      1.00      1.00       104
           3       0.96      0.95      0.96       109
           4       0.99      0.97      0.98       132

    accuracy                           0.98       562
   macro avg       0.97      0.98      0.98       562
weighted avg       0.98      0.98      0.98       562
```

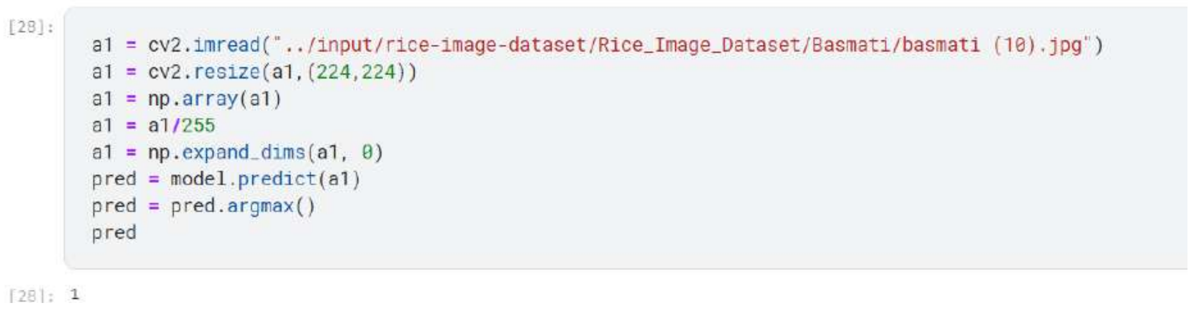
-
- ## 6: Visualizing Accuracy and Loss
- The accuracy and loss can be visualized to check the correlation between the epochs and loss or epochs and accuracy.

```
[17]:  from plotly.offline import iplot, init_notebook_mode
       import plotly.express as px
       import pandas as pd

       init_notebook_mode(connected=True)

       acc = pd.DataFrame({'train': history.history['acc'], 'val': history.history['val_acc']})

       fig = px.line(acc, x=acc.index, y=acc.columns[0::], title='Training and Evaluation Accuracy every Epoch', m
       fig.show()
```

-

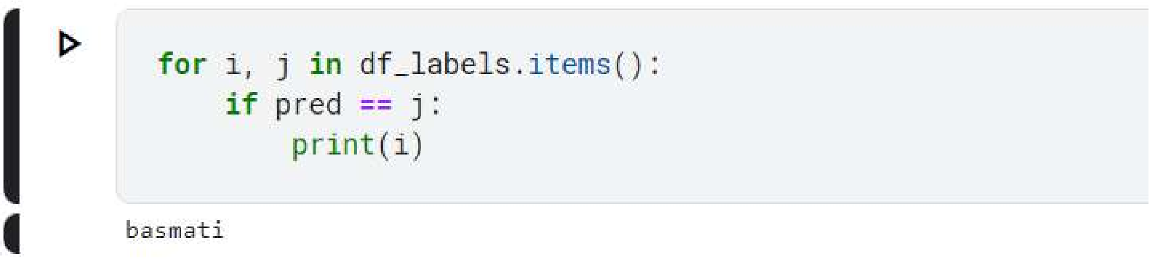


-
- ## 7: Testing the Model:
- Here we will take a image of basmati rice and check what our model predicts for the same.
-



```
[28]:
a1 = cv2.imread("../input/rice-image-dataset/Rice_Image_Dataset/Basmati/basmati (10).jpg")
a1 = cv2.resize(a1,(224,224))
a1 = np.array(a1)
a1 = a1/255
a1 = np.expand_dims(a1, 0)
pred = model.predict(a1)
pred = pred.argmax()
pred
```

-
```
[28]: 1
```



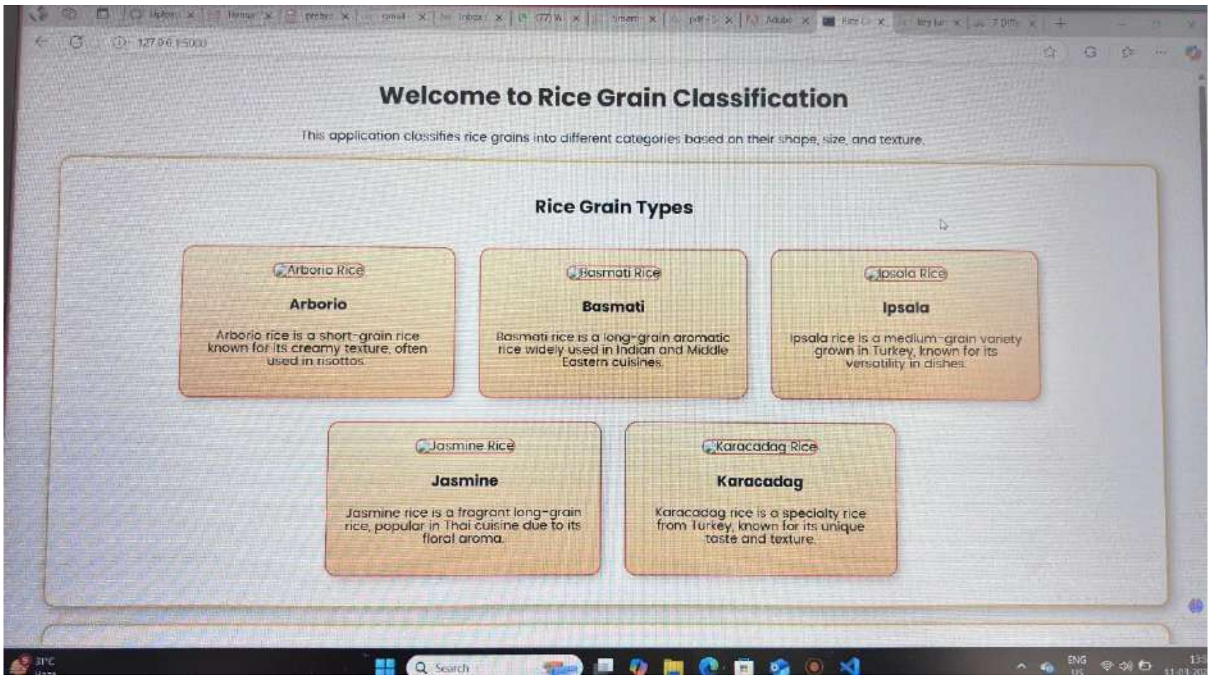
```
for i, j in df_labels.items():
    if pred == j:
        print(i)
```

```
basmati
```

-
- 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.

```
[25]:  model.save("rice.h5")
```

-
- **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
- **1: 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:



Welcome to Rice Grain Classification

This application classifies rice grains into different categories based on their shape, size, and texture.

Rice Grain Types

Arborio Rice
Arborio
Arborio rice is a short-grain rice known for its creamy texture, often used in risottos.

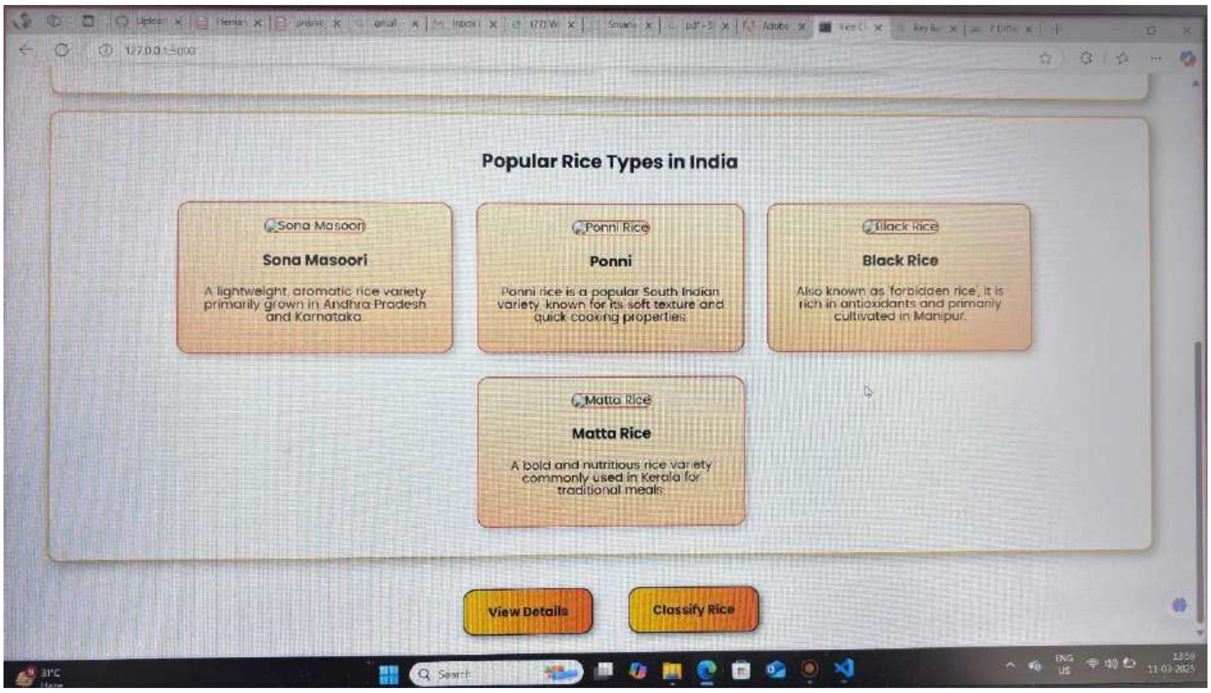
Basmati Rice
Basmati
Basmati rice is a long-grain aromatic rice widely used in Indian and Middle Eastern cuisines.

Ipsala Rice
Ipsala
Ipsala rice is a medium-grain variety grown in Turkey, known for its versatility in dishes.

Jasmine Rice
Jasmine
Jasmine rice is a fragrant long-grain rice, popular in Thai cuisine due to its floral aroma.

Karacadag Rice
Karacadag
Karacadag rice is a specialty rice from Turkey, known for its unique taste and texture.

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


Popular Rice Types in India
Sona Masoori
A lightweight aromatic rice variety primarily grown in Andhra Pradesh and Karnataka
Ponni
Ponni rice is a popular South Indian variety known for its soft texture and quick cooking properties
Black Rice
Also known as forbidden rice, it is rich in antioxidants and primarily cultivated in Manipur.
Matta Rice
A bold and nutritious rice variety commonly used in Kerala for traditional meals
View Details
Classify Rice

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

- ## 2: Build Python code:
- mport the libraries
-

```
import tensorflow as tf
import tensorflow_hub as hub
import warnings
warnings.filterwarnings('ignore')
import h5py
import numpy as np
import os
from flask import Flask, app,request,render_template
from tensorflow import keras
import cv2
import tensorflow_hub as hub
```

-
-
- Loading the saved model and initializing the flask app
-

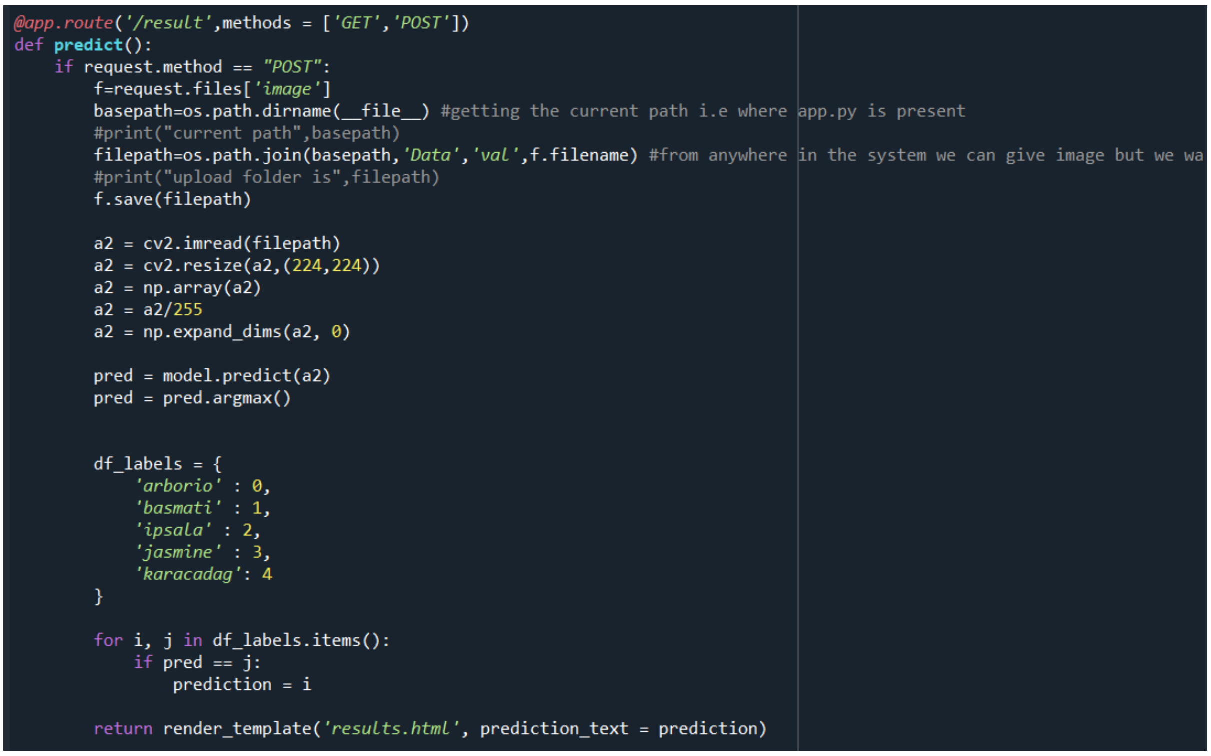
```
model = tf.keras.models.load_model(filepath='rice.h5',custom_objects={'KerasLayer':hub.KerasLayer})
app = Flask(__name__)
```

-
-
-
- Render HTML pages:
-

```
@app.route('/')
def home():
    return render_template('index.html')

@app.route('/details')
def pred():
    return render_template('details.html')
```

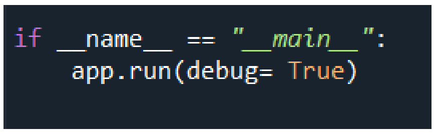
-

- 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.



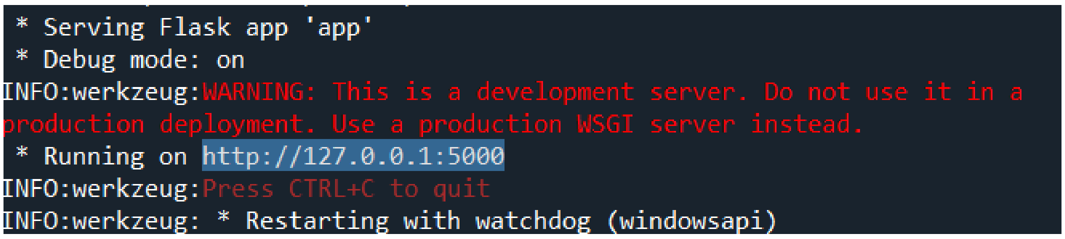
```python
@app.route('/result',methods = ['GET','POST'])
def predict():
    if request.method == "POST":
        f=request.files['image']
        basepath=os.path.dirname(__file__) #getting the current path i.e where app.py is present
        #print("current path",basepath)
        filepath=os.path.join(basepath,'Data','val',f.filename) #from anywhere in the system we can give image but we wa
        #print("upload folder is",filepath)
        f.save(filepath)

        a2 = cv2.imread(filepath)
        a2 = cv2.resize(a2,(224,224))
        a2 = np.array(a2)
        a2 = a2/255
        a2 = np.expand_dims(a2, 0)

        pred = model.predict(a2)
        pred = pred.argmax()


        df_labels = {
            'arborio' : 0,
            'basmati' : 1,
            'ipsala' : 2,
            'jasmine' : 3,
            'karacadag': 4
        }

        for i, j in df_labels.items():
            if pred == j:
                prediction = i

        return render_template('results.html', prediction_text = prediction)
```

- 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:

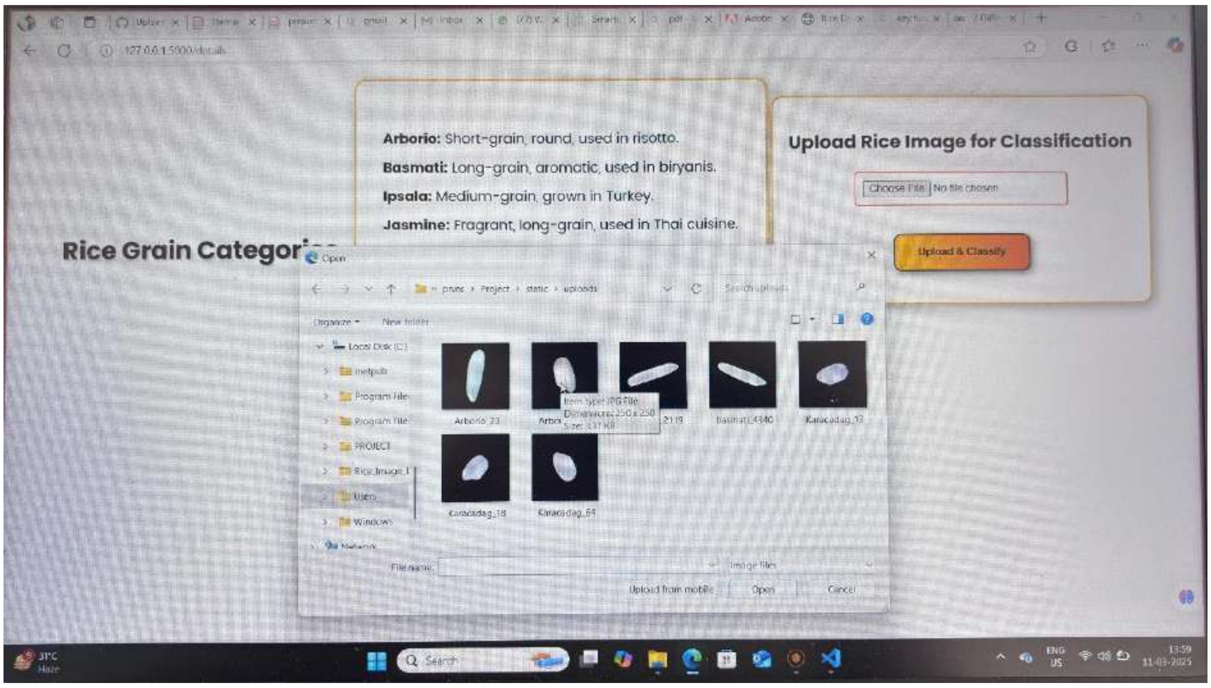


```python
if __name__ == "__main__":
    app.run(debug= True)
```

- **3: 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.

- 

```
 * Serving Flask app 'app'
 * Debug mode: on
INFO:werkzeug:WARNING: This is a development server. Do not use it in a
production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug: * Restarting with watchdog (windowsapi)
```

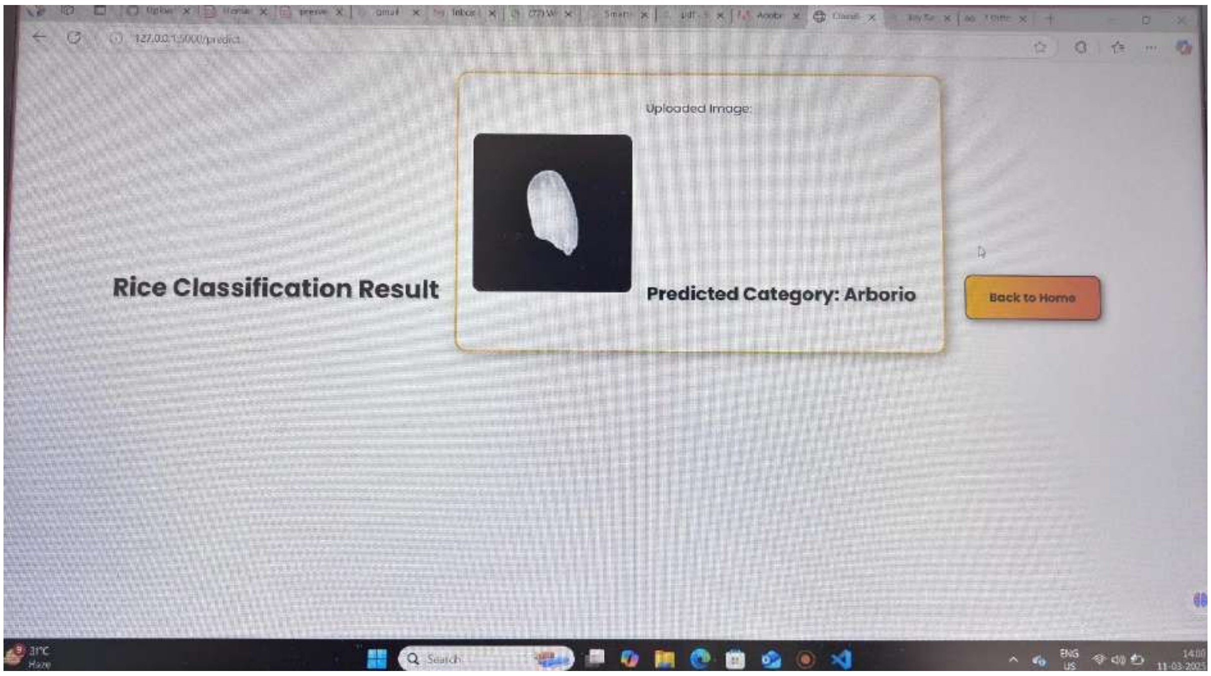
-
- The home page looks like this. When you click on the button “Predict”, you’ll be redirected to the predict section
- Input 1
- 

Rice Grain Categor

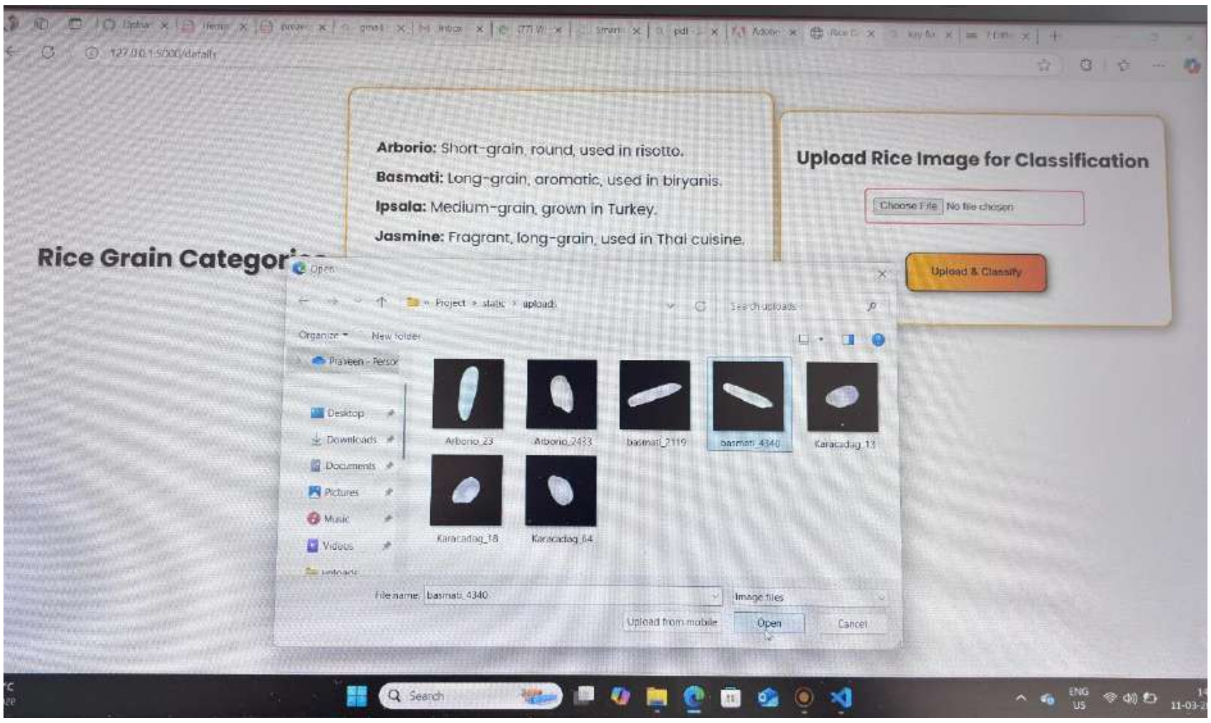
Arborio: Short-grain, round, used in risotto.
Basmati: Long-grain, aromatic, used in biryanis.
Ipsala: Medium-grain, grown in Turkey.
Jasmine: Fragrant, long-grain, used in Thai cuisine.

Upload Rice Image for Classification

Output 1

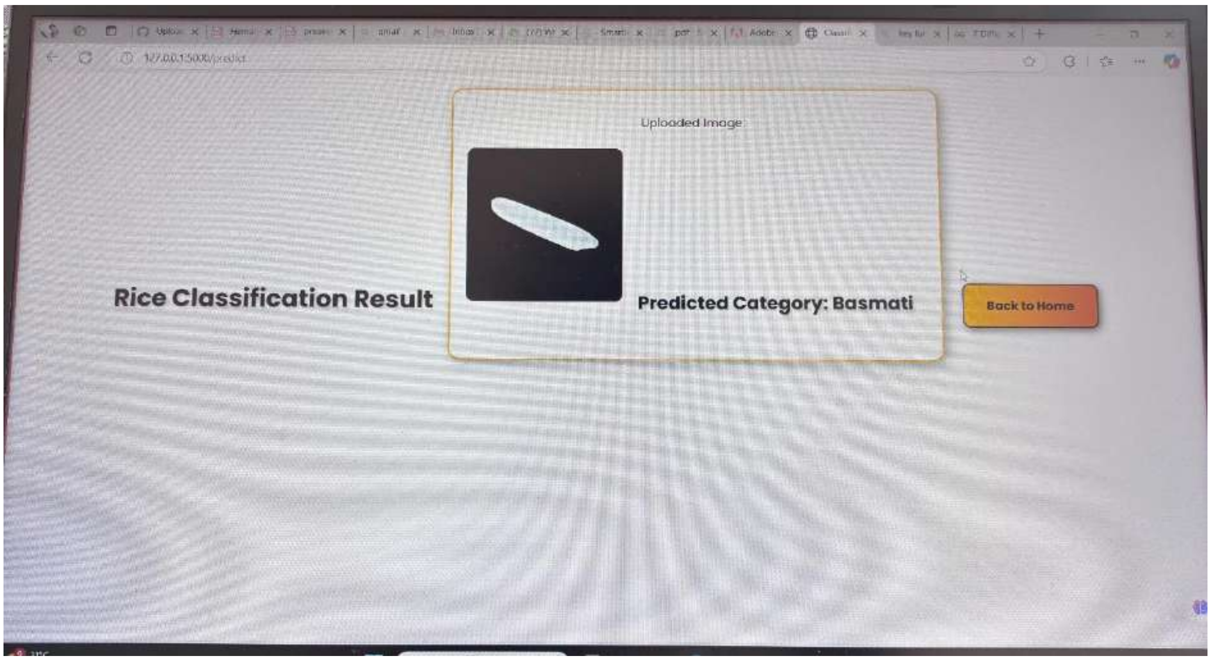


Input 2



Arborio: Short-grain, round, used in risotto.
Basmati: Long-grain, aromatic, used in biryanis.
Ipsala: Medium-grain, grown in Turkey.
Jasmine: Fragrant, long-grain, used in Thai cuisine.

Output 2



127.0.0.1:5000/predict

Uploaded Image

Rice Classification Result

Predicted Category: Basmati

Back to Home