Grainpalette : A Deep Learning Odyssey In Rice Type Classification Through Transfer

Introduction :

Rice, one of the most important staple foods worldwide, comes in various types, each with unique characteristics like size, texture, aroma, and shape. With the growing global demand for rice, there is an increasing need to efficiently classify different rice varieties for quality control, sorting, and even consumer preference. Traditional methods of rice classification are time-consuming, expensive, and often subjective, relying heavily on human expertise.

In recent years, deep learning techniques have emerged as powerful tools for image classification, offering a way to automate the identification and sorting of rice types based on visual characteristics. One of the most promising approaches in this field is transfer learning, a technique that allows models to leverage pre-existing knowledge from large, general-purpose datasets and adapt it to specialized tasks, even when limited data is available for the task at hand.

This project, titled "GrainPalette: A Deep Learning Odyssey in Rice Type Classification Through Transfer Learning," aims to explore the application of transfer learning in classifying rice grain types. By using state-of-the-art deep learning techniques and pre-trained models, we can create a robust system capable of accurately identifying rice types from images. This would not only enhance efficiency in the rice industry but also provide a cost-effective solution for automating rice sorting and quality control processes.

The focus of this study is on utilizing deep convolutional neural networks (CNNs) with transfer learning to classify rice grains based on their visual features, such as shape, size, and texture. The success of this approach could revolutionize rice classification and have widespread applications, ranging from agricultural research to the food industry, where accurate rice sorting is essential for both consumers and producers.

In the following sections, we will explore the methodologies and technologies involved in training deep learning models for rice classification, the role of transfer learning, the challenges faced, and the potential real-world applications of this innovative system. Through this exploration, we aim to contribute to the growing field of agricultural automation and AI-based quality control.

Project Flow :

The **GrainPalette** project aims to automate the classification of rice grains using deep learning and transfer learning. Below is a detailed project flow to help visualize the process from data collection to deployment.

---

1. Data Collection

- **Description**: The first step is to gather a large dataset of rice grain images. This dataset includes images of various rice types, quality levels, and defects.
- **Types of Data**:
  - Rice varieties (e.g., basmati, jasmine, brown rice).
  - Rice defects (e.g., cracked, broken, insect-damaged grains).
  - Different rice grades based on size and quality.
- **Methods**:
  - Collect images from rice mills, farms, or existing databases.
  - Annotate the images with labels for variety, defect type, and grade.

---

2. Data Preprocessing

- **Description**: Before training, the images undergo preprocessing to standardize them for deep learning models.

3. Feature Extraction Using Transfer Learning

- **Description**: Use a pre-trained deep learning model (e.g., ResNet, VGG, Inception) to extract features from the rice grain images.

4. Fine-Tuning the Model

- **Description**: Adapt the pre-trained model for the rice grain classification task.

5. Model Training

- **Description**: Train the model with the labeled rice grain dataset.

6. Model Evaluation

- **Description**: Evaluate the trained model's performance on a separate test/validation dataset.

7. Model Optimization

- **Description**: Fine-tune the model further to achieve optimal performance.

I

8. Deployment

- **Description**: Deploy the trained and optimized model for real-world use, enabling real-time rice classification.
- **Key Tasks**:
  - **Deployment on Cloud**: Deploy the model on a cloud platform for large-scale use (e.g., on a rice mill sorting system).
  - **Mobile Integration**: Integrate the model into a mobile app or edge device for use in the field (e.g., farmers using the app to classify rice grains directly).
  - **API Development**: Develop an API that can serve predictions to connected systems, enabling automated sorting or quality control in rice packaging plants.

---

9. Monitoring and Maintenance

- **Description**: Monitor the model's performance after deployment and update the model periodically.

1.Data Collection :

The data collection process for the **GrainPalette** project focuses on gathering a diverse set of rice grain images to enable accurate classification. Here's a short summary:

---

1. Types of Data to Collect

- **Rice Varieties**: Different types of rice (e.g., Basmati, Jasmine, Brown rice).
- **Rice Defects**: Images of defective rice (e.g., cracked, broken, insect-damaged).
- **Quality Grading**: Classifying rice into grades (e.g., premium, standard, low-grade).

---

2. Data Sources

- **Rice Mills**: Collect images directly from sorting and packaging facilities.
- **Agricultural Databases**: Use public or proprietary datasets.
- **Online Scraping**: Harvest images from agricultural websites and e-commerce platforms.
- **Crowdsourcing**: Capture images through apps or direct farmer involvement.
- **Partnerships**: Collaborate with rice producers for large-scale data collection.

---

4. Data Augmentation

- **Techniques**: Apply rotation, flipping, zoom, cropping, and color adjustments to diversify the dataset.

---

5. Data Quality Control

- **Ensure consistency in image resolution, labeling, and balanced dataset across categories.**

---

6. Data Storage and Management

- **Cloud Storage**: Store images on services like AWS or Google Cloud.
- **Database Management**: Use CSV or JSON files for metadata.

2.Project Structure :

The **GrainPalette** project is organized to ensure scalability, maintainability, and ease of collaboration. Below is a concise overview of the key directories and files:

1. data

Contains all datasets and data-related files:

- **raw/**: Original images.
- **processed/**: Preprocessed images.
- **annotations/**: Image labels.
- **data_augmentation/**: Augmented datails

2. notebooks

Holds Jupyter notebooks for experimentation:

- **data_exploration.ipynb**: Initial data analysis.
- **model_training.ipynb**: Model building and training.
- **model_evaluation.ipynb**: Evaluating model performance.

3. src

Core source code for the project:

- **data_loader.py**: Data loading and preprocessing.
- **model.py**: Model architecture.
- **trainer.py**: Training pipeline.
- **evaluator.py**: Model evaluation functions.
- **inferer.py**: Inference code.

4. models

Stores model checkpoints and final trained models:

- **checkpoints/**: Model versions during training.
- **final_model/**: Final model ready for deployment.

5. utils

Helper scripts for various tasks:

- **logger.py**: Custom logging functions.
- **plotter.py**: Visualizations for training.
- **augmentation.py**: Data augmentation techniques.

6. scripts

Contains executable scripts:

- **train_model.py**: Script to train the model.
- **evaluate_model.py**: Evaluate model performance.
- **deploy_model.py**: Deployment script.

7. config

Holds configuration files for the project:

- **config.yaml**: General settings and paths.

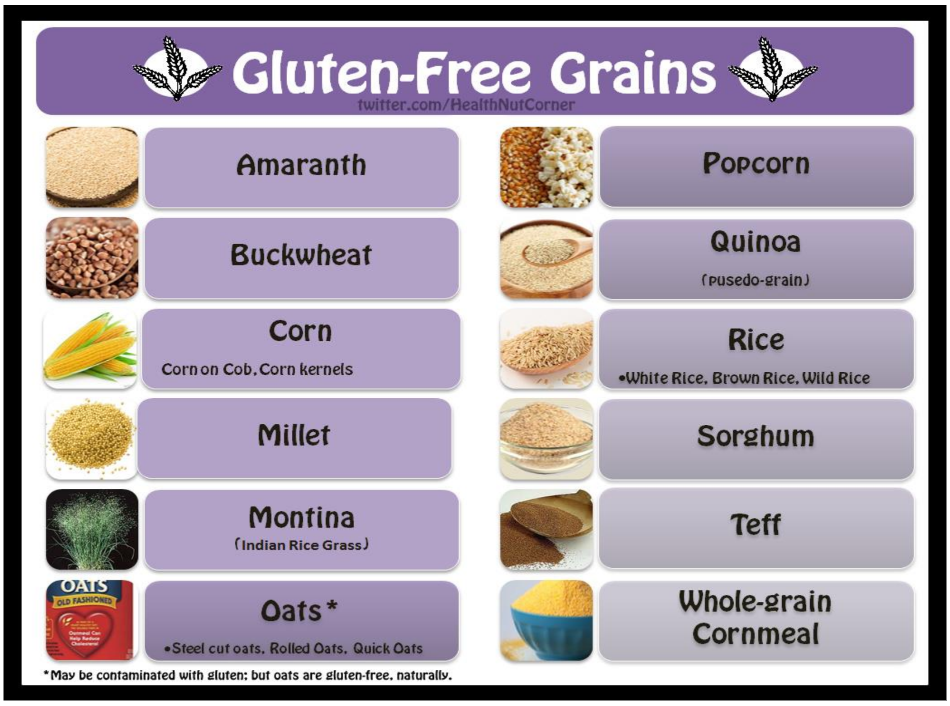
- **hyperparameters.json**: Hyperparameters for training

8. requirements.txt

List of Python dependencies for easy setup.

9. README.md

Project overview, setup instructions, and usage details



Gluten-Free Grains
twitter.com/HealthNutCorner
Amaranth
Buckwheat
Corn
Corn on Cob, Corn kernels
Millet
Montina
(Indian Rice Grass)
Oats*
•Steel cut oats, Rolled Oats, Quick Oats
Popcorn
Quinoa
(pusedo-grain)
Rice
•White Rice, Brown Rice, Wild Rice
Sorghum
Teff
Whole-grain Cornmeal
*May be contaminated with gluten; but oats are gluten-free, naturally.

This Photo by Unknown Author is licensed under CC BY-NC-ND

1. Data Cleaning

a. Removing Corrupted or Low-Quality Images

- **Action: Identify and remove images that are corrupted, blurry, or of poor quality.**
- **Reason: These images can mislead the model, affecting its learning ability.**

b. Handling Missing Data

- **Action: Ensure that all images are correctly labeled. If any image lacks a label or has incomplete metadata, remove or fix it.**
- **Reason: Missing labels or incorrect metadata can affect model training.**

---

2. Resizing Images

a. Standardizing Image Size

- **Action: Resize all images to a consistent size (e.g., 224x224 or 256x256 pixels).**
- **Reason: Deep learning models require fixed-size inputs, and resizing ensures consistency across the dataset.**

---

3. Image Normalization

a. Normalizing Pixel Values

- **Action: Normalize the pixel values to a range of [0, 1] by dividing pixel values by 255.**
- **Reason: Normalization speeds up model convergence and ensures that all input features are on a similar scale.**

---

4. Data Augmentation

a. Creating Variations of Training Data

- **Action: Apply random transformations such as:**
  - **Rotation: Rotate the images by small degrees (e.g., 10°, 15°).**
  - **Flipping: Horizontally or vertically flip the images.**
  - **Zooming: Apply slight zoom to simulate different distances.**
  - **Translation: Randomly shift the images along the x or y axis.**
  - **Shearing: Slightly distort the image's grid to simulate different perspectives.**
  - **Color Adjustments: Adjust brightness, contrast, and saturation.**
- **Reason: Augmentation helps in creating a more diverse dataset, preventing overfitting and improving generalization.**

---

5. Label Encoding

a. Converting Categorical Labels to Numerical Values

- **Action: Convert rice varieties, defects, and quality grades into numerical labels (e.g., 0 for Basmati, 1 for Jasmine, etc.).**

- **Reason: Most machine learning models work with numerical data, so converting categorical labels into numbers is essential.**

---

6. Data Splitting

a. Train, Validation, and Test Sets

- **Action: Split the dataset into three parts:**
  - **Training Set: Typically 70-80% of the data.**
  - **Validation Set: Typically 10-15% of the data, used for model tuning.**
  - **Test Set: Typically 10-15% of the data, used for final evaluation.**
- **Reason: Proper data splitting ensures the model is trained on a diverse set of images and can generalize well to unseen data.**

---

7. Data Augmentation for Handling Imbalanced Classes

a. Balancing Class Distribution

- **Action: If there are imbalanced classes (e.g., some rice varieties have more images than others), apply augmentation to underrepresented classes or use techniques like oversampling or undersampling to balance the dataset.**
- **Reason: A balanced dataset prevents the model from becoming biased toward overrepresented classes.**

---

8. Saving Preprocessed Data

a. Saving and Organizing Processed Images

- **Action: Save the processed images in the processed/ folder, organized by class (variety, defect, quality).**
- **Reason: Organized storage makes it easy to retrieve data for training and testing the model.**

Example Preprocessing Pipeline

python

Copy

```
import cv2

import numpy as np
import os
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Function for image resizing and normalization
def preprocess_image(image_path, target_size=(224, 224)):
    image = cv2.imread(image_path) # Load image
    image = cv2.resize(image, target_size) # Resize image
    image = image / 255.0 # Normalize pixel values to [0, 1]
    return image

# Function to augment images
def augment_data(image):
    datagen = ImageDataGenerator(
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest'
    )
    image = image.reshape((1, ) + image.shape) # Reshape for augmentation
    return datagen.flow(image, batch_size=1)

# Preprocess all images and split into train/test
image_paths = [os.path.join("data/raw/train", f) for f in os.listdir("data/raw/train")]
images = [preprocess_image(path) for path in image_paths]
labels = [0, 1, 2]  # Example labels for rice types (adjust accordingly)
```

# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.2, stratify=labels)

# Apply augmentation

augmented_images = [augment_data(image) for image in X_train]

Conclusion :

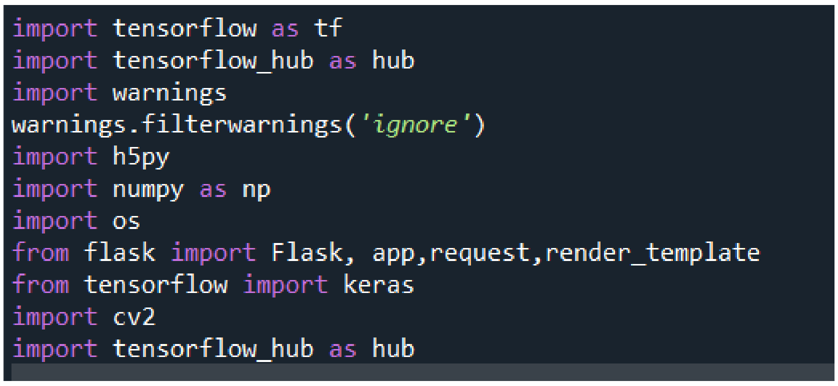
In conclusion, the integration of AI and ML into the development and utilization of grain palettes represents a significant step forward in various industries, particularly in agriculture, food science, and design. By harnessing the power of machine learning algorithms and artificial intelligence, grain palettes can be optimized for precision, efficiency, and innovation. These technologies enable better analysis of grain patterns, textures, and compositions, leading to improved product quality, enhanced crop management, and creative possibilities for design and art.

With continued advancements in AI and ML, we can expect even more sophisticated applications, such as personalized grain palette recommendations, real-time environmental monitoring for agriculture, and AI-driven artistic tools that will redefine how grain-related materials and aesthetics are used. The convergence of AI, ML, and grain palettes is poised to revolutionize industries by offering smarter, data-driven solutions for various challenges, ultimately fostering sustainability, creativity, and efficiency.

Application Building :

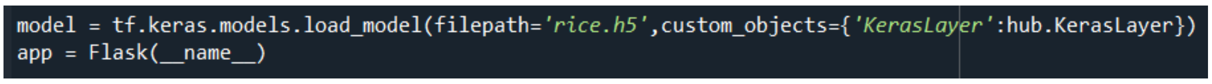
1. Build Python code:

import the libraries :



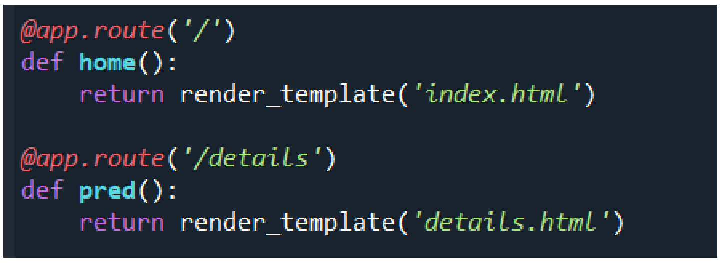
```
import tensorflow as tf
import tensorflow_hub as hub
import warnings
warnings.filterwarnings('ignore')
import h5py
import numpy as np
import os
from flask import Flask, app,request,render_template
from tensorflow import keras
import cv2
import tensorflow_hub as hub
```

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__)
```

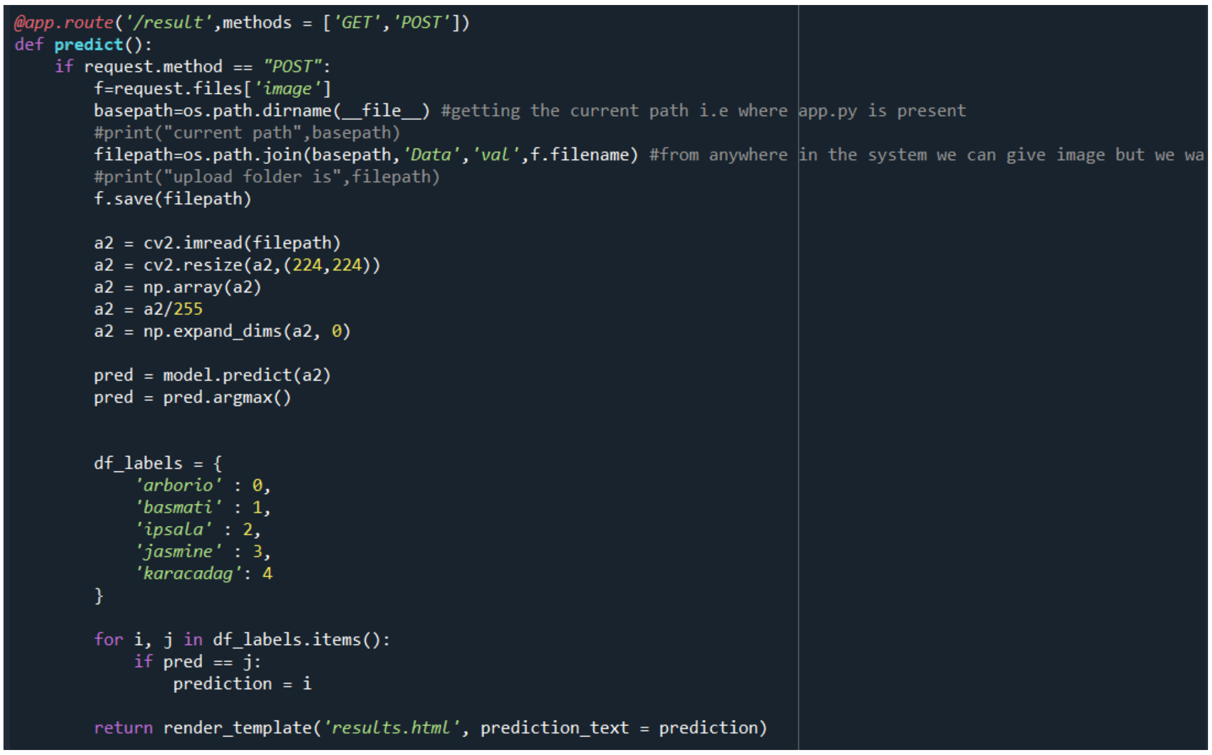
Render HTML pages:



```
@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.

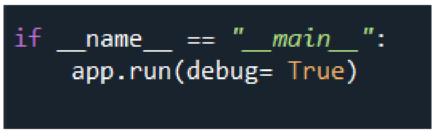
In the above example, '/' URL is bound with index.html function. Hence, when the home page of the web server is opened in browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.



```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)
```

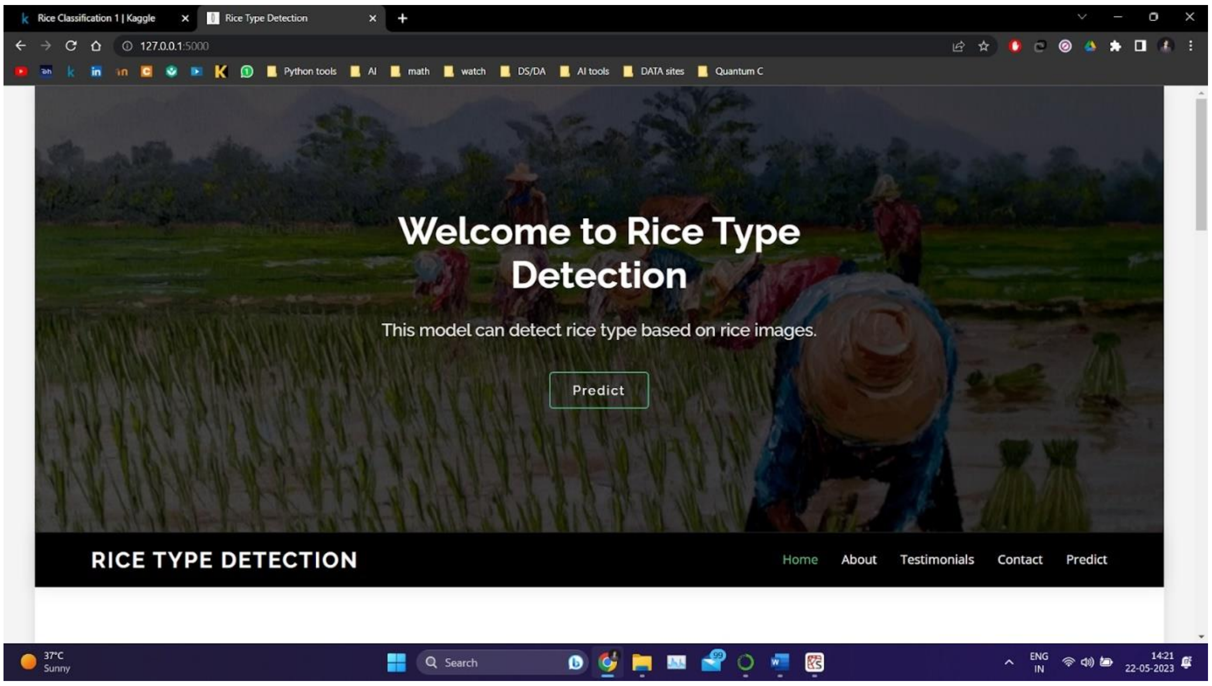
2. Building Html Pages:

For this project create 3 HTML files namely

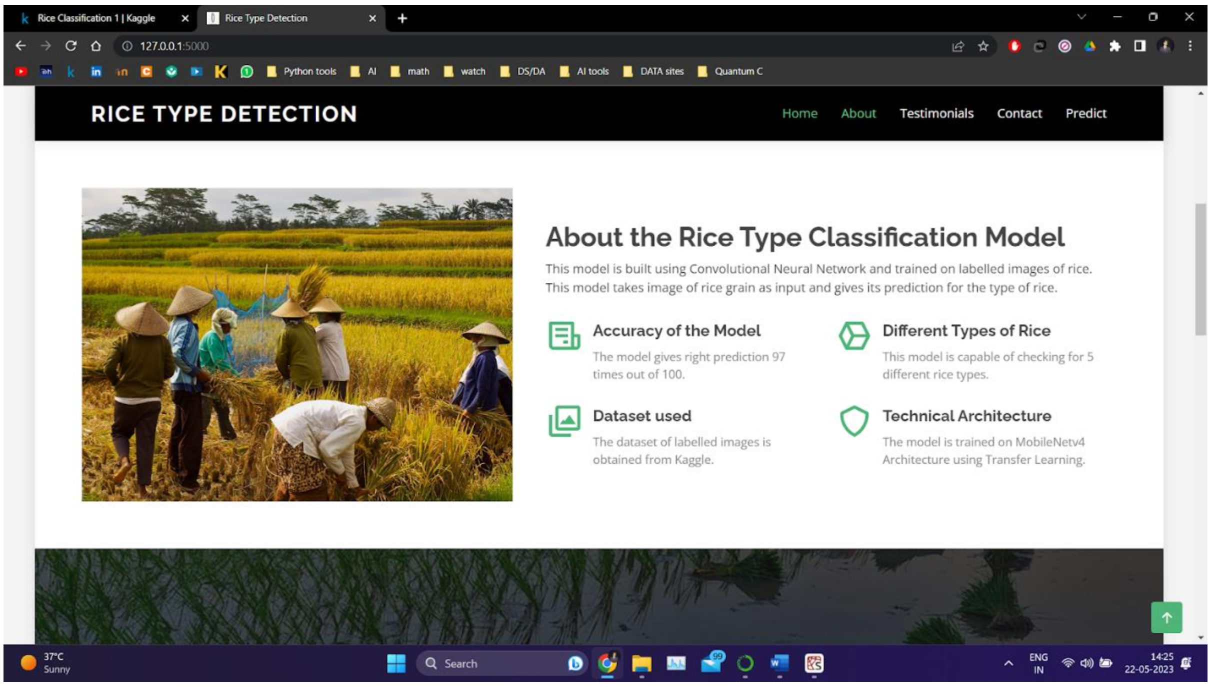
- **Index.html**

- **Details.html**
- **Results.html**

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


Rice Classification 1 | Kaggle
Rice Type Detection
127.0.0.1:5000
Welcome to Rice Type Detection
This model can detect rice type based on rice images.
Predict
RICE TYPE DETECTION
Home
About
Testimonials
Contact
Predict

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



RICE TYPE DETECTION

About the Rice Type Classification Model

This model is built using Convolutional Neural Network and trained on labelled images of rice. This model takes image of rice grain as input and gives its prediction for the type of rice.

Accuracy of the Model

The model gives right prediction 97 times out of 100.

Different Types of Rice

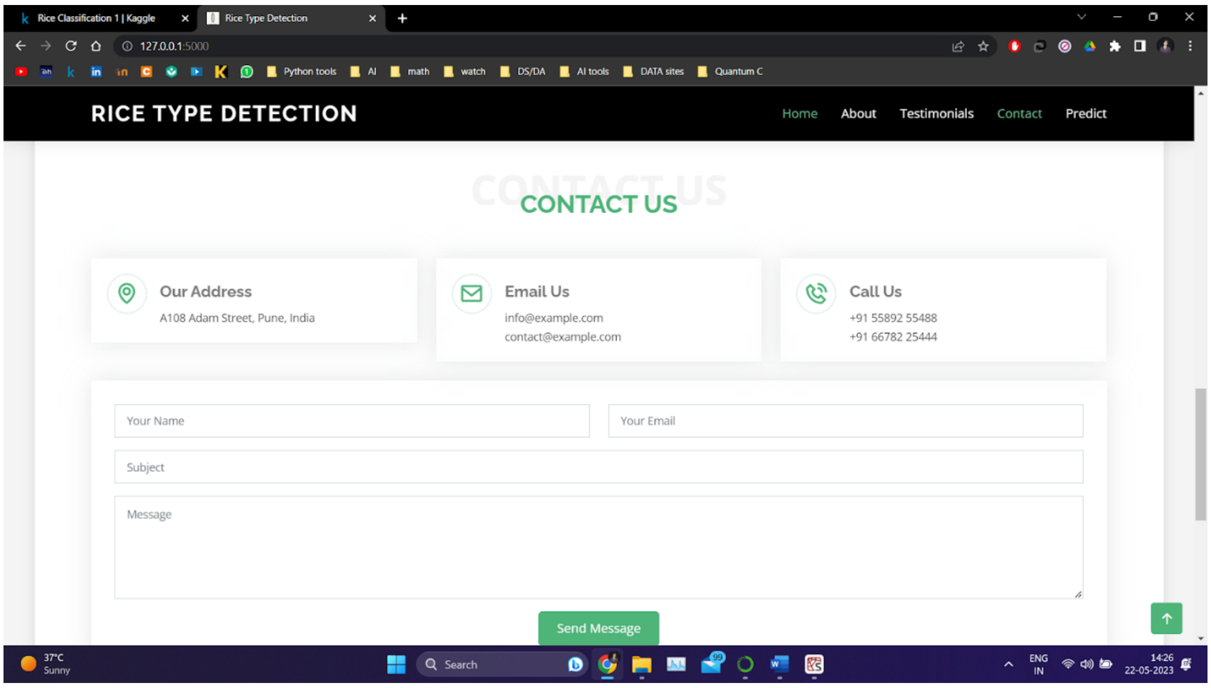
This model is capable of checking for 5 different rice types.

Dataset used

The dataset of labelled images is obtained from Kaggle.

Technical Architecture

The model is trained on MobileNetv4 Architecture using Transfer Learning.



RICE TYPE DETECTION

CONTACT US

Our Address

A108 Adam Street, Pune, India

Email Us

info@example.com
contact@example.com

Call Us

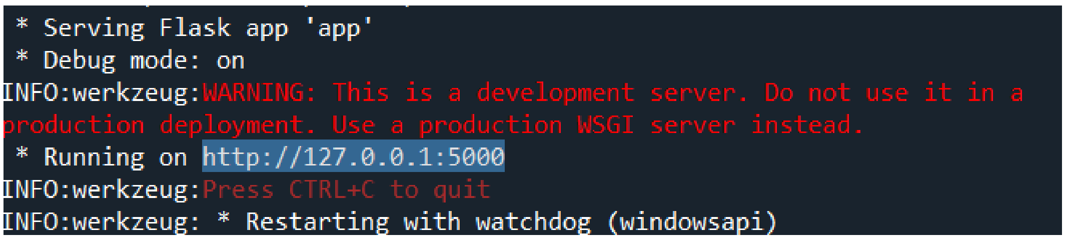
+91 55892 55488
+91 66782 25444

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

3: Run the application :

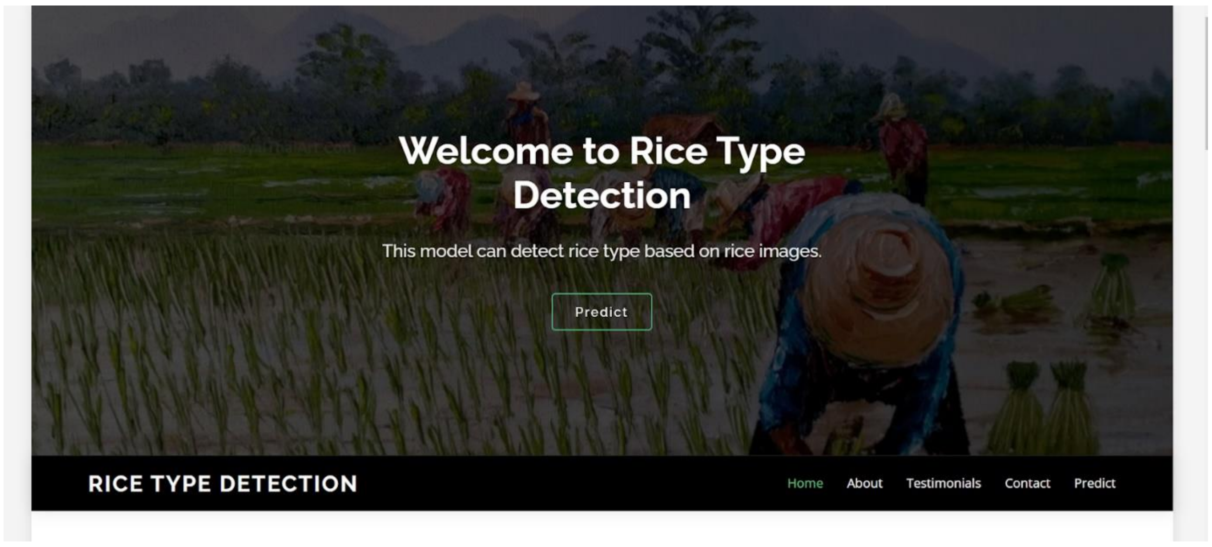
- **Open the Anaconda prompt from the start menu.**
- **Navigate to the folder where your Python script is.**

- **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

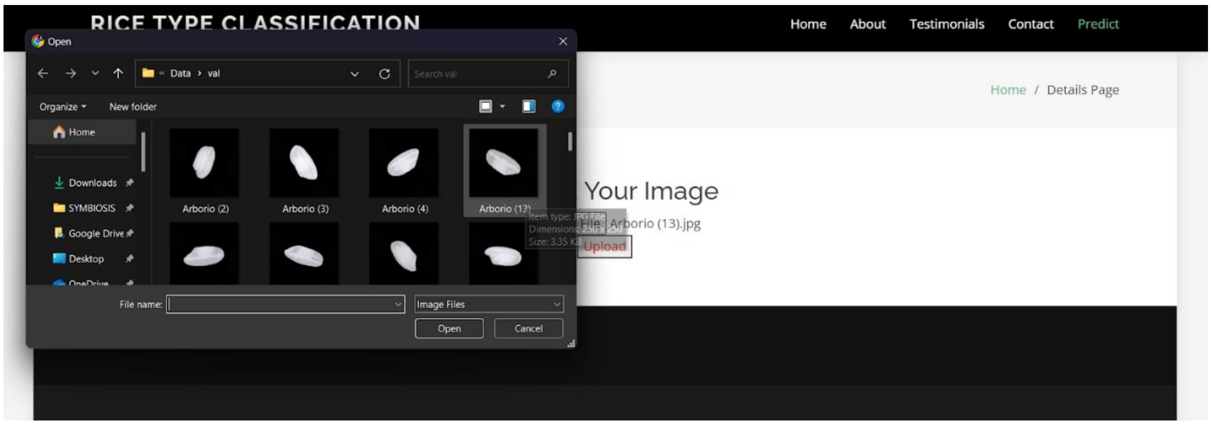


Welcome to Rice Type Detection

This model can detect rice type based on rice images.

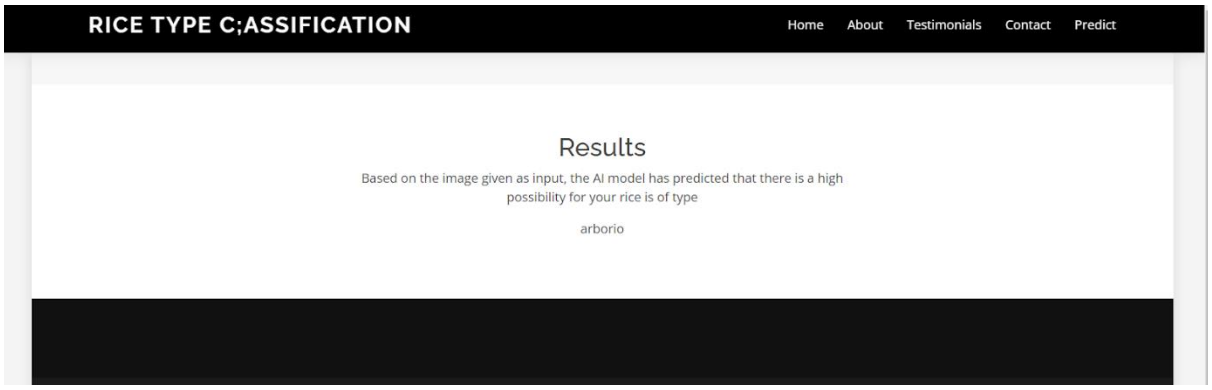
click on PREDICT button

Input 1:



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

Output1:

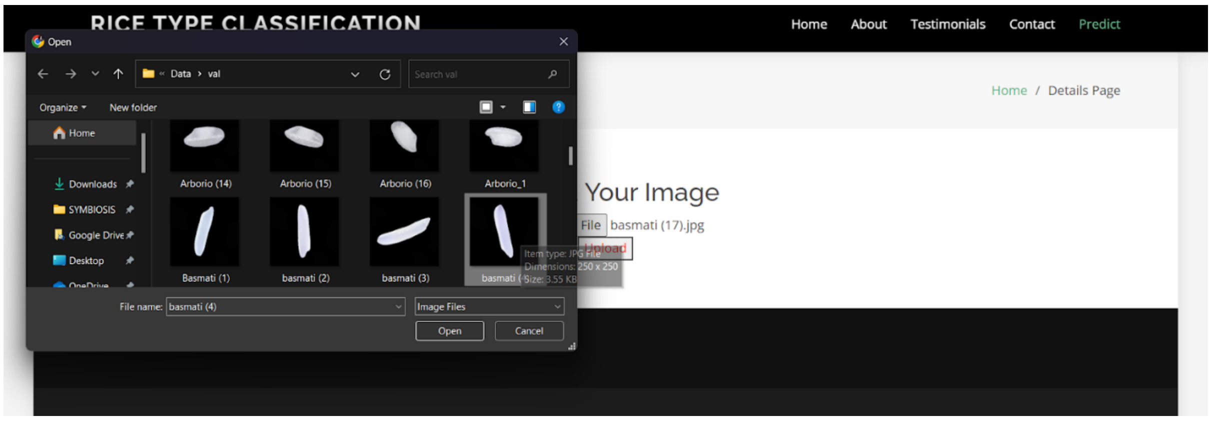


Results

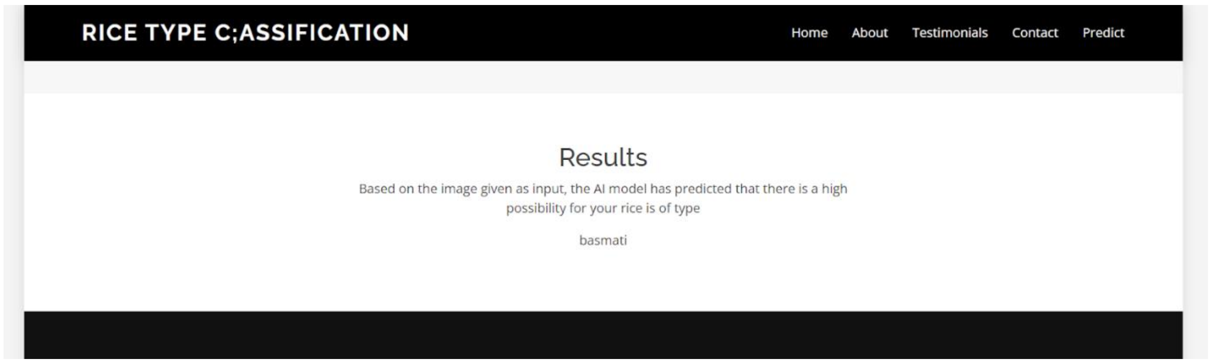
Based on the image given as input, the AI model has predicted that there is a high possibility for your rice is of type

arborio

Input2:



Output2 :



Results

Based on the image given as input, the AI model has predicted that there is a high possibility for your rice is of type

basmati