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

1. Introduction

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

Team ID: LTVIP2025TMID41438

Team Size: 5

- **Team Leader**: Nimmala Sravanthi – Project Coordination, Milestone Tracking, Data Collection, Cleaning.
- **Team Member**: Agraharam Amrutha– preprocessing pipeline, Model Building, Testing
- **Team Member**: Marpuri Likitha – UI/UX Design, Documentation, Descriptive Statistics
- **Team Member**: Mukkamalla Venkata Bhavana– ML Model Development & Flask Integration
- **Team Member**: B Mounika– Define Problem /Problem Understanding, Model Evaluation

2.Project Overview

2.1. Purpose

Rice is a staple food globally, and differentiating between various rice types is essential for quality control, pricing, and market classification. This project, *GrainPalette*, aims to build a deep learning model using transfer learning (MobileNetV2) to classify rice grains into distinct types based on image features. The goal is to offer a fast, accurate, and accessible solution through a web interface that can assist in automating rice type classification in agricultural and industrial sectors.

2.2. Features

- User interface to upload rice grain images.
- Prediction using a trained **MobileNetV2** deep learning model.
- Real-time result display on a web interface.
- Error handling for unsupported formats and invalid images.
- Instantly returns a label – e.g., "Jasmine", "Basmati", "Arborio", etc. – after image submission.

3. Architecture

3.1. Frontend

- Developed Using: HTML5 and CSS3.
- HTML Pages (in templates/ folder):
  - ➢ index.html: Provides the form for uploading rice images.
  - ➢ results.html: Displays the prediction output after model inference.
  - ➢ details.html: (Optional) Shows rice variety details or additional visual information.
- Static Files: Located in static/uploads/, containing sample images such as Ipasala_1.jpg.
- User Experience:
  - ➢ Clean, minimal interface with intuitive design.
  - ➢ Responsive layout with gradient background and prediction results rendered dynamically.
  - ➢ Conditional result display: predicted rice type shown along with an image and label.

3.2. Backend

- **Framework**: Python with Flask microframework.
- **Prediction Workflow**:
  - ➢ User uploads image via the form (index.html).
  - ➢ Image is saved to the uploads/ folder.
  - ➢ Flask loads the trained model (rice.h5).
  - ➢ Image is preprocessed to match MobileNetV2 input size and format.
  - ➢ The model returns the rice type label (e.g., "Arborio", "Ipsala").
  - ➢ Prediction is rendered back on the results.html page.
- **Model Training**:
  - ➢ Done using **Transfer Learning** with MobileNetV2 in **Google Colab** or **Jupyter Notebook** (train2.ipynb, train22.py).
  - ➢ Training images from Rice_Image_Dataset/ are resized and normalized.
  - ➢ The model is exported as rice.h5 after achieving satisfactory accuracy.
- **Files Used**:
  - ➢ app.py: Manages routing, file handling, image preprocessing, and invoking the model.
  - ➢ rice.h5: Trained deep learning model used for real-time predictions.

3.3. Data Preprocessing:

- All images are: Resized to 224x224 pixels (MobileNetV2 default input size), normalized to a 0–1 scale and converted to NumPy arrays and reshaped to (1, 224, 224, 3) before being passed to the model.
- Class indices are handled using internal logic or optionally stored in a mapping dictionary.

3.4. Database

- No Database is integrated.
- The app processes the uploaded image in real-time and returns the result immediately.
- Uploaded files are stored temporarily in the uploads/ folder during session runtime

4. Setup Instructions

4.1. Prerequisites

- Python 3.10+
- Flask
- TensorFlow / Keras
- Pillow, NumPy, pickle
- Google Colab (for training)

4.2 Installation

- pip install requirements.txt
- python app.py

5.Folder Structure

5.1 Client (Flask Frontend)

```
rice_type_detection/
├── Rice_Image_Dataset/          # Dataset used for training the model
├── static/
│   └── uploads/
│       └── Ipasala_1.jpg        # Sample static image for UI testing
├── templates/
│   ├── index.html               # Upload form for rice image input
│   ├── details.html             # (Optional) Additional rice info page
│   └── results.html             # Displays prediction result
├── uploads/
│   ├── Arborio.jpg              # User-uploaded or test image
│   └── Ipsala.jpg               # Another sample input image
├── app.py                       # Flask application script
├── req.txt                      # Requirements file for dependencies
├── rice.h5                      # Trained MobileNetV2 model file
├── train2.ipynb                 # Jupyter notebook for model training and evaluation
└── train22.py                   # Python script for training (Colab/local)
```

5.2 File/Directory Roles

- **Rice_Image_Dataset/:** Contains the categorized training images of different rice types.
- **static/uploads/**: Holds demo images served on the frontend or used in testing.
- **templates/**: Stores HTML templates used by Flask:
  - ➢ index.html: Main page to upload a rice image.
  - ➢ results.html: Shows prediction result with styled output.
  - ➢ details.html: Optional page for showing extra rice classification information or visual summary.

6 Running the Application

- Start the app:
  ```
  cd Flask
  python app.py
  ```
- Open in browser at: http://127.0.0.1:5000/

7 API Document

Route: /

- Method: POST (via HTML form)
- Input: Uploaded image file of rice grain
- Processing:
  - Image is resized and normalized using utility functions.
  - Model (rice_model.h5) is loaded and prediction is made.
  - Result mapped using class_indices.pkl.
- Output:
  - Rendered HTML with predicted rice type and visual cues.
- Error Handling:
  - Handles missing file or unsupported image formats gracefully.

8 Authentication

Not applicable – no user login is implemented. Prototype for classification use only.

9 User Interface

- Simple form to upload a rice grain image.
- Background image featuring grains.
- Output shows classified rice type with related image and message.

10 Testing

- Model evaluated using training/validation split in Colab.
- Accuracy validated using confusion matrix and classification report.
- Manual testing with sample rice images from dataset.

11 Screenshots or Demo

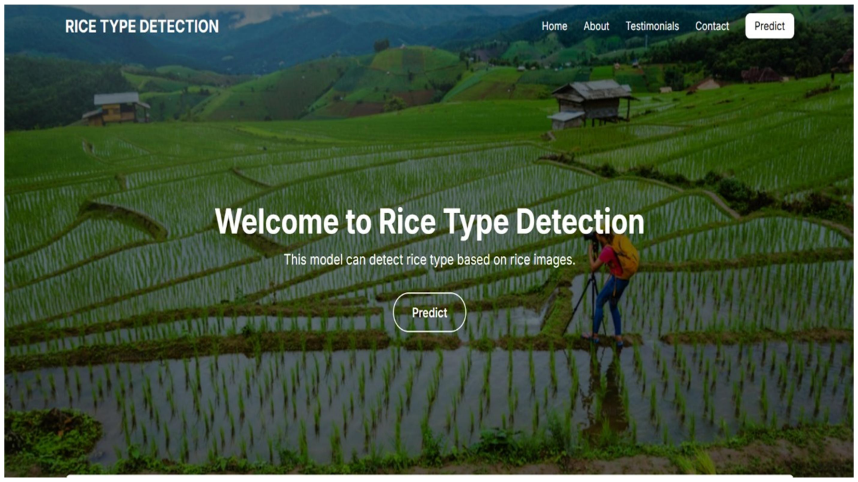

RICE TYPE DETECTION
Home
About
Testimonials
Contact
Predict
Welcome to Rice Type Detection
This model can detect rice type based on rice images.
Predict

Figure 11.1

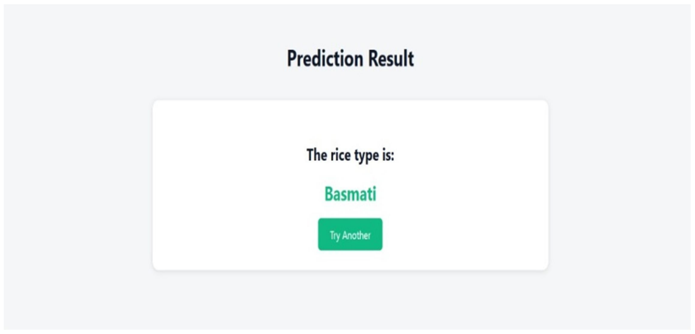
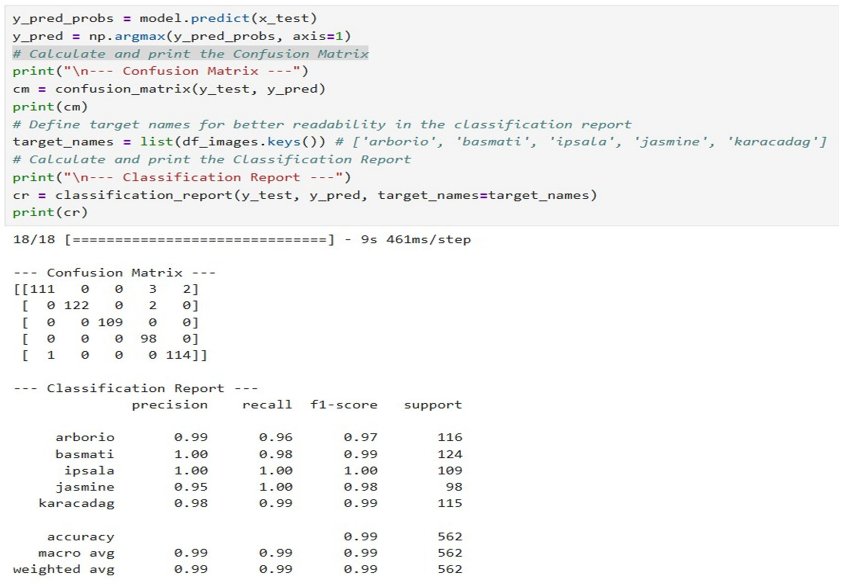


Figure 11.2



```
y_pred_probs = model.predict(x_test)
y_pred = np.argmax(y_pred_probs, axis=1)
# Calculate and print the Confusion Matrix
print("\n--- Confusion Matrix ---")
cm = confusion_matrix(y_test, y_pred)
print(cm)
# Define target names for better readability in the classification report
target_names = list(df_images.keys()) # ['arborio', 'basmati', 'ipsala', 'jasmine', 'karacadag']
# Calculate and print the Classification Report
print("\n--- Classification Report ---")
cr = classification_report(y_test, y_pred, target_names=target_names)
print(cr)
```

```
18/18 [==============================] - 9s 461ms/step

--- Confusion Matrix ---
[[111   0   0   3   2]
 [  0 122   0   2   0]
 [  0   0 109   0   0]
 [  0   0   0  98   0]
 [  1   0   0   0 114]]

--- Classification Report ---
              precision    recall  f1-score   support

     arborio       0.99      0.96      0.97       116
     basmati       1.00      0.98      0.99       124
      ipsala       1.00      1.00      1.00       109
     jasmine       0.95      1.00      0.98        98
   karacadag       0.98      0.99      0.99       115

    accuracy                           0.99       562
   macro avg       0.99      0.99      0.99       562
weighted avg       0.99      0.99      0.99       562
```

Figure 11.3: Classification Report

- **Demo Link:**
  https://drive.google.com/drive/folders/1LqE2MawknsaI9-gKlDe54_8XvgJB8Juw

12 Known Issues

- Model may misclassify if image is blurry or background is noisy.
- Currently supports limited rice classes as per training data.
- No database integration yet.

13 Future Enhancements

- Add support for more rice types with a larger dataset.
- Implement a database for logging predictions and user data.
- Add authentication and downloadable reports.
- Integrate with supply-chain or agricultural platforms for real-time utility.