Project Report Format

1. **INTRODUCTION**
   1.1 **Project Overview**

Rice is a staple food for more than half the world's population, with various types offering distinct nutritional values, tastes, and commercial uses. Manual classification of rice types is time-consuming and error-prone, especially in large-scale agricultural and supply chain environments.

GrainPalette is a deep learning–driven solution designed to automatically classify rice types using image data. Leveraging transfer learning with MobileNetV2, this project combines speed, accuracy, and scalability to enhance grain quality control, streamline processing, and support agricultural research..

   1.2 **Purpose**

The core purpose of the GrainPalette project is to harness the power of deep learning to automate and improve the classification of rice varieties through image-based analysis. In many agricultural, food processing, and export industries, the ability to accurately distinguish between rice types—such as Basmati, Jasmine, Brown, and White rice—is critical for quality control, pricing, packaging, and regulatory compliance.

This project aims to:

- Reducing dependency on manual inspection.
- Enhancing accuracy and consistency in rice type classification.
- Enabling real-time classification in agricultural industries and marketplaces.
- Providing a user-friendly interface for interactive testing and usage.

Ultimately, GrainPalette aims to merge the domains of artificial intelligence and agriculture to create a future-ready, efficient, and intelligent system for rice type classification that is accessible, explainable, and impactful.

2. **IDEATION PHASE**
   2.1 Problem Statement
   2.2 Empathy Map Canvas
   2.3 Brainstorming
3. **REQUIREMENT ANALYSIS**
   3.1 Solution Requirement
   3.2 Data Flow Diagram
   3.3 Technology Stack
4. **PROJECT DESIGN**
   4.1 Problem Solution Fit
   4.2 Proposed Solution
   4.3 Solution Architecture

5. **PROJECT PLANNING & SCHEDULING**

5.1 Project Planning

6. **FUNCTIONAL AND PERFORMANCE TESTING**

6.1 Performance Testing

7. **RESULTS**

7.1 Output Screenshots

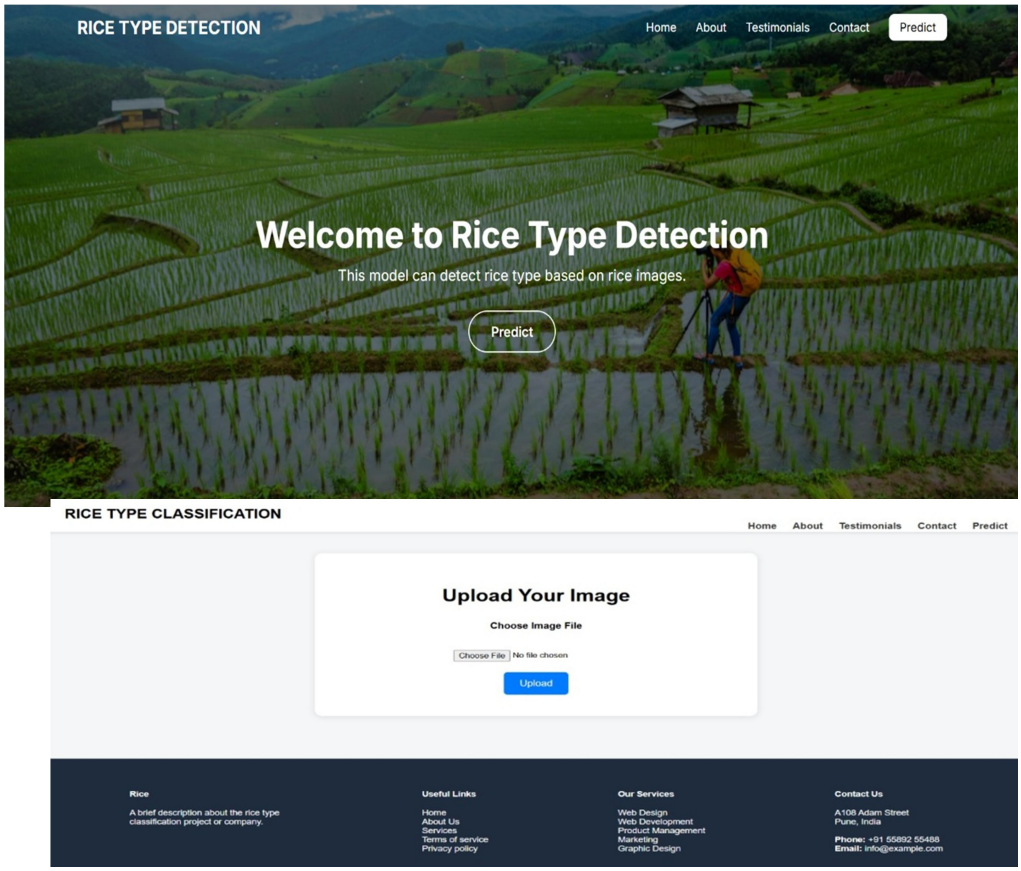

Figure 7.1

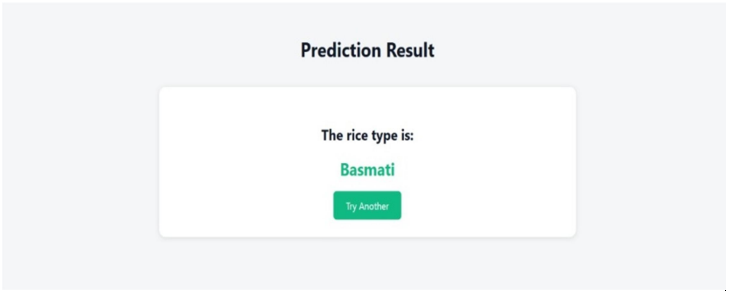

Figure 7.2

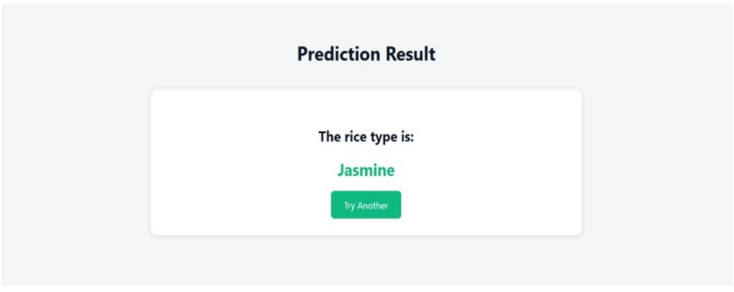

Figure 7.3

```
history=model.fit(x_train,y_train,epochs=10,validation_data=(x_val,y_val))
```

```
Epoch 1/10
71/71 [==============================] - 24s 287ms/step - loss: 0.5110 - accuracy: 0.8671 - val_loss: 0.1798 - val_accuracy: 0.9787
Epoch 2/10
71/71 [==============================] - 38s 535ms/step - loss: 0.1426 - accuracy: 0.9698 - val_loss: 0.1135 - val_accuracy: 0.9840
Epoch 3/10
71/71 [==============================] - 31s 431ms/step - loss: 0.1000 - accuracy: 0.9800 - val_loss: 0.0860 - val_accuracy: 0.9840
Epoch 4/10
71/71 [==============================] - 30s 430ms/step - loss: 0.0798 - accuracy: 0.9836 - val_loss: 0.0703 - val_accuracy: 0.9840
Epoch 5/10
71/71 [==============================] - 32s 446ms/step - loss: 0.0656 - accuracy: 0.9862 - val_loss: 0.0637 - val_accuracy: 0.9840
Epoch 6/10
71/71 [==============================] - 31s 432ms/step - loss: 0.0564 - accuracy: 0.9898 - val_loss: 0.0571 - val_accuracy: 0.9840
Epoch 7/10
71/71 [==============================] - 39s 546ms/step - loss: 0.0526 - accuracy: 0.9893 - val_loss: 0.0548 - val_accuracy: 0.9894
Epoch 8/10
71/71 [==============================] - 37s 526ms/step - loss: 0.0458 - accuracy: 0.9920 - val_loss: 0.0482 - val_accuracy: 0.9840
Epoch 9/10
71/71 [==============================] - 38s 529ms/step - loss: 0.0408 - accuracy: 0.9924 - val_loss: 0.0503 - val_accuracy: 0.9840
Epoch 10/10
71/71 [==============================] - 38s 537ms/step - loss: 0.0363 - accuracy: 0.9933 - val_loss: 0.0471 - val_accuracy: 0.9840
```

Figure 7.4

```
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 7.5

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
mobilenetv2_1.00_224 (Functional)	(None, 4, 4, 1280)	2,257,984
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0
dense (Dense)	(None, 128)	163,968
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 5)	645

Figure 7.6

8. ADVANTAGES & DISADVANTAGES

8.1 Advantages

- Fast and Accurate: Uses optimized deep learning architecture.
- User-Friendly: Simple UI with drag-and-drop image upload.
- Scalable: Can be extended to other grains or food items.
- Cost-Efficient: Reduces manual inspection costs.

8.2 Disadvantages

- Limited Generalization: Performance may drop on unseen grain types.
- Requires Quality Input: Low-quality or blurry images reduce accuracy.

Dependence on Pretrained Weights: Transfer learning limitations if rice features vary drastically.

9. CONCLUSION

The project "GrainPalette - A Deep Learning Odyssey In Rice Type Classification Through Transfer Learning" demonstrates how transfer learning can be applied effectively to agricultural problems. With MobileNetV2 at its core, this solution provides a reliable, scalable, and efficient tool for rice classification. It not only speeds up the process but also ensures consistency and accuracy—essential for modern food supply chains.

This project bridges the gap between traditional agricultural inspection and AI-powered automation, contributing to smarter and more sustainable food systems

10. FUTURE SCOPE

Real-Time Video Classification: Analyze rice streams via webcam or video input.

Mobile App Deployment: Android/iOS version with offline capabilities.

More Classes Support: Extend model for more rice varieties or sub-types.

Explainable AI (XAI): Use Grad-CAM for visual explanation of decisions.

Multi-language Interface: Interface localization for broader adoption.

11. APPENDIX

Source Code

This appendix contains key parts of the source code used in the project " GrainPalette - A Deep Learning Odyssey In Rice Type Classification Through Transfer Learning ". It includes the backend logic, model training script, and frontend form design.

A. app.py – Flask backend handling image input, prediction, and output rendering.

This script handles the core backend logic including:

- Route definitions and handling form input
- Loading the trained model, scaler, and label encoders
- Handling form input
- Preprocessing and prediction

```
ice > app.py > ...
1   from flask import Flask, render_template, request, url_for, redirect
2   from tensorflow.keras.models import load_model
3   import tensorflow_hub as hub
4   import numpy as np
5   import cv2
6   import os
7   from werkzeug.utils import secure_filename
8
9   app = Flask(__name__)
10
11  # --- CHANGE 1: Point UPLOAD_FOLDER to be inside the 'static' directory ---
12  UPLOAD_FOLDER = 'static/uploads'
13  app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
14
15  # Create the folder if it doesn't exist
16  if not os.path.exists(app.config['UPLOAD_FOLDER']):
17      os.makedirs(app.config['UPLOAD_FOLDER'])
18
19  # Load your trained model
20  model = load_model('rice.h5', custom_objects={'KerasLayer': hub.KerasLayer})
21
22  # Map the model's output indices to the actual rice type names
23  label_map = {0: "Arborio", 1: "Basmati", 2: "Ipsala", 3: "Jasmine", 4: "Karacada
24
25  @app.route('/')
26  def index():
27      """Renders the home page."""
28      return render_template('index.html')
29
30  @app.route('/details')
31  def details():
32      """Renders the page for uploading an image."""
33      return render_template('details.html')
34
35  @app.route('/results', methods=['POST'])
36  def results():
```

B. train22.py – Model Training Scrip-Includes dataset preprocessing, model building with MobileNetV2, training loop, and saving of weights.

rice > train22.py > ...

```python
import pathlib
import cv2
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
import plotly.offline as iplot
import plotly.express as px
import pandas as pd
import tensorflow_hub as hub
import plotly
import matplotlib.pyplot as plt

data_dir = pathlib.Path('Rice_Image_Dataset')

# Limiting to 600 images per class for demonstration.
# For a more robust model, consider using the full dataset.
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]

# Display example images
fig, ax = plt.subplots(ncols=5, figsize=(20,5))
fig.suptitle('Rice Category Examples')

# Read and display the first image of each category
arborio_image = cv2.imread(str(arborio[0]))
basmati_image = cv2.imread(str(basmati[0]))
ipsala_image = cv2.imread(str(ipsala[0]))
jasmine_image = cv2.imread(str(jasmine[0]))
karacadag_image = cv2.imread(str(karacadag[0]))

ax[0].set_title('Arborio')
ax[1].set_title('Basmati')
ax[2].set_title('Ipsala')
ax[3].set_title('Jasmine')
```

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

GitHub Link: https://github.com/Amrutha242005/Grainpallete-A-Deep-learning-odyssey-in-rice-type-classification-through-transfer-learning

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