Contents

1 Introduction

Rice is a staple food for a significant portion of the global population, and its various varieties possess distinct characteristics that influence culinary preferences and agricultural practices. Accurate classification of rice varieties is essential for quality control, breeding programs, and supply chain management. Traditional classification methods rely heavily on manual inspection, which is time-consuming and prone to human error. Therefore, automating this process using machine learning offers a scalable and efficient alternative.

This study aims to develop an image classification model to accurately distinguish between five rice varieties: Arborio, Basmati, Ipsala, Jasmine, and Karacadag. By employing TensorFlow, a widely-adopted machine learning framework, the project leverages deep learning techniques to extract and learn pertinent features from the Rice Image Dataset.

2 Dataset Description

The Rice Image Dataset utilized in this study comprises a total of 75,000 images, evenly distributed across five rice varieties:

- **Arborio:** 15,000 images
- **Basmati:** 15,000 images
- **Ipsala:** 15,000 images
- **Jasmine:** 15,000 images
- **Karacadag:** 15,000 images

Each image captures the physical characteristics of the rice grains, providing visual data essential for classification tasks. The dataset is structured into separate folders for each rice variety, facilitating organized data management and preprocessing.

3 Methodology

The classification pipeline encompasses several key steps: data loading and preprocessing, data visualization, dataset splitting, model construction, compilation, training, and evaluation. Below is a detailed exposition of each step.

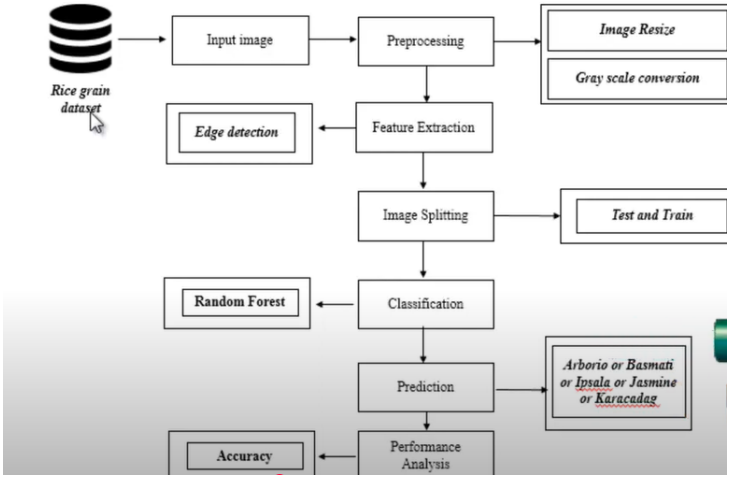

Figure 1: Flow step

3.1 1. Data Loading and Preprocessing

The initial phase involves importing necessary libraries and modules, followed by loading the image data from the designated directory.

3.1.1 Importing Libraries

```
import pandas as pd
import tensorflow as tf
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import ImageGrid
from pathlib import Path
from PIL import Image
import cv2
from tensorflow.keras import layers
from sklearn import preprocessing
```

Listing 1: Importing Necessary Libraries

3.1.2 Data Path Configuration

```
from pathlib import Path
from sklearn import preprocessing
import os

# Get the current working directory
current_directory = Path(os.getcwd())

# Define the path to the rice_img folder
dataset_path = current_directory / "rice_img"

# Get all image paths from the rice_img folder
image_link = list(dataset_path.glob('**/*.jpg'))
```

```
13
14 # Extract folder names as image names
15 image_name = [x.parents[0].stem for x in image_link]
16
17 # Encode the image labels
18 image_label = preprocessing.LabelEncoder().fit_transform(image_name)
19
20 # Print the results for verification
21 print(f"Number of images found: {len(image_link)}")
22 print(f"Image labels: {image_label}")
```

Listing 2: Data Path Configuration

Explanation:

- **Path Configuration:** Utilizes Python's `pathlib` to navigate the directory structure and retrieve all JPEG images within the `rice_img` folder.
- **Label Encoding:** Extracts folder names corresponding to rice varieties and encodes them into numerical labels using `LabelEncoder` for model compatibility.
- **Verification:** Outputs the total number of images and the encoded labels to ensure correct data loading.

3.2 2. Data Structuring

A Pandas DataFrame is created to organize the image paths, names, and labels, facilitating subsequent data handling and visualization.

```
1 import numpy as np
2 df = pd.DataFrame()
3 df['link'] = np.array(image_link, dtype=str)  # Use built-in str type
4 df['name'] = image_name
5 df['label'] = image_label
6
7 # Print the DataFrame for verification
8 print(df.head())
```

Listing 3: DataFrame Construction

Explanation:

- **DataFrame Construction:** Aggregates image paths (`link`), corresponding rice variety names (`name`), and encoded labels (`label`) into a structured format.
- **Verification:** Displays the first few entries to confirm accurate data mapping.

3.3 3. Data Visualization

Visualization aids in understanding the distribution of classes and provides qualitative insights into the dataset.

3.3.1 Class Distribution

```
# Plot the varieties
df.name.value_counts().plot(kind='bar', figsize=(12, 8), grid=True,
    color='blue')
plt.title('Distribution of Rice Varieties')
plt.xlabel('Rice Variety')
plt.ylabel('Number of Images')
plt.show()
```

Listing 4: Class Distribution Plot

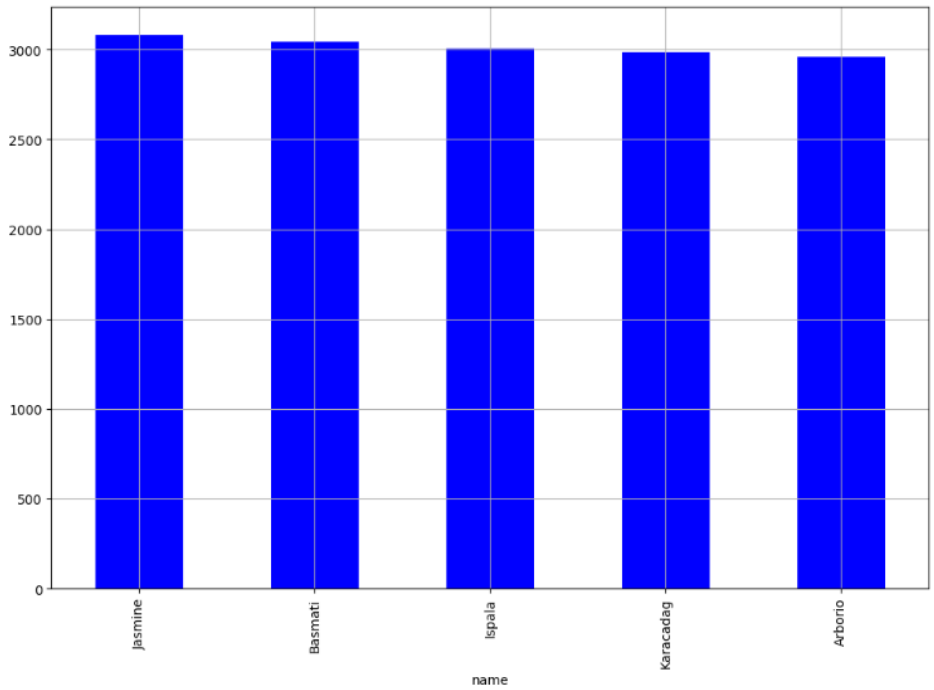

Figure 2: number of images per rice variety

Explanation:

- **Bar Chart:** Illustrates the number of images per rice variety, ensuring balanced class representation.

3.3.2 Sample Image Grid

```
fig = plt.figure(1, figsize=(15,15))
grid = ImageGrid(fig, 121, nrows_ncols=(5,4), axes_pad=0.10)
i = 0
for category_id, category in enumerate(df.name.unique()):
    for filepath in df[df['name'] == category]['link'].values[:4]:
        ax = grid[i]
        img = Image.open(filepath)
        ax.imshow(img)
        ax.axis('off')
        if i % 4 == 4-1:
            ax.text(300,100, category, verticalalignment='center',
   fontsize=20, color='black')
```

```
            i += 1

plt.show()
```

Listing 5: Sample Image Grid

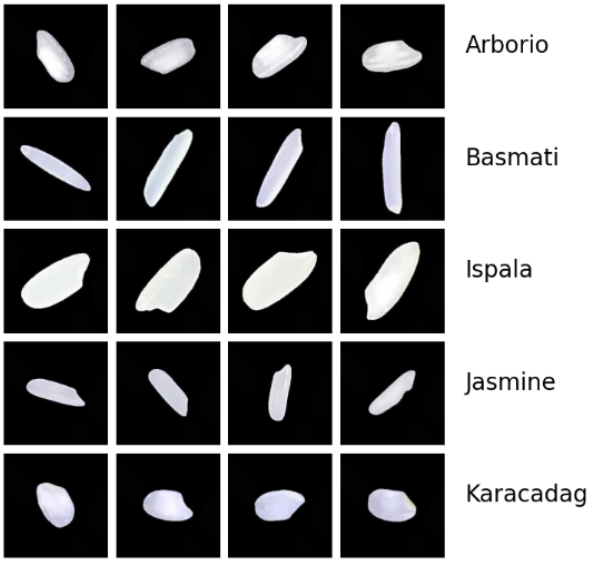

Figure 3: Different rice variety

3.4 4. Dataset Splitting

The dataset is partitioned into training and testing subsets to evaluate model performance objectively.

```
# Split the DataFrame into training (70%) and testing (30%) sets using
    a fixed random state of 1
train_df, test_df = train_test_split(df, test_size=0.3, random_state=1,
    stratify=df['label'])
```

Listing 6: Dataset Splitting

‘

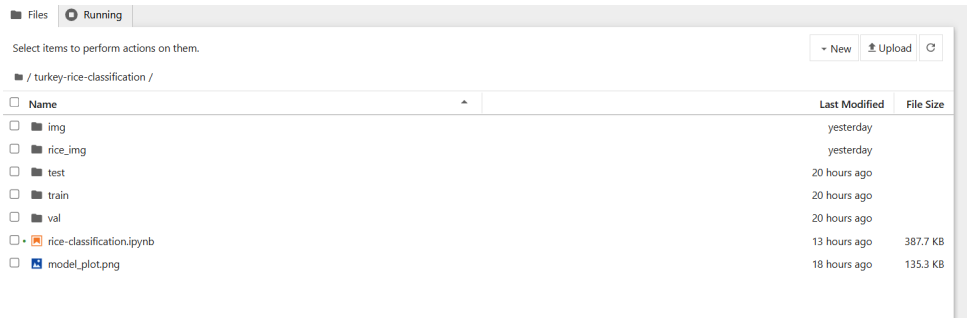

Figure 4: Organisation of project files

Explanation:

- **Train-Test Split:** Allocates 70% of the data for training and 30% for testing, maintaining class distribution through stratification.

3.5 5. Data Augmentation and Generators

Data augmentation enhances model generalization by introducing variability in the training data.

```
# Create training and test image generators from the DataFrame.
# Each row of the DataFrame provides the file path ('link') and label ('name').
# 'color_mode' specifies how images are loaded (RGB here).
# 'target_size' resizes images to 28x28.
# 'class_mode' defines how labels are handled (categorical, for multi-class classification).
# 'batch_size' specifies how many samples per batch.

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.15,
    zoom_range=0.15,
    horizontal_flip=True,
    fill_mode="nearest",
    validation_split=0.3  # Assuming further split for validation
)

test_datagen = ImageDataGenerator(rescale=1./255)

train_images = train_datagen.flow_from_dataframe(
    dataframe=train_df,
    x_col='link',
    y_col='name',
    color_mode='rgb',
    batch_size=32,
    target_size=(28, 28),
    class_mode='categorical',
    subset='training'
)

validation_images = train_datagen.flow_from_dataframe(
    dataframe=train_df,
    x_col='link',
    y_col='name',
    color_mode='rgb',
    batch_size=32,
    target_size=(28, 28),
    class_mode='categorical',
    subset='validation'
)

test_images = test_datagen.flow_from_dataframe(
    dataframe=test_df,
    x_col='link',
    y_col='name',
    color_mode='rgb',
    batch_size=32,
    target_size=(28, 28),
    class_mode='categorical'
```

```
)
```

Listing 7: Data Augmentation and Generators

Explanation:

- **ImageDataGenerator:** We Configure data augmentation parameters for the training set, such as rotation, shifts, shear, zoom, and horizontal flips to introduce diversity.
- **Rescaling:** Normalizes pixel values to the [0,1] range by rescaling.
- **Subset Configuration:** Splits the training data further into training and validation sets using a validation split.
- **Flow from DataFrame:** Creates iterable data generators for training, validation, and testing datasets.

3.6 6. Model Architecture

A Convolutional Neural Network (CNN) is constructed to learn and classify rice varieties based on image features.

```
model = tf.keras.models.Sequential([
    # 1. Define the input layer, specifying the shape (28x28, 3 channels).
    tf.keras.layers.Input(shape=(28, 28, 3)),

    # 2. First convolutional layer with 16 filters, kernel size of 3x3, ReLU activation.
    tf.keras.layers.Conv2D(16, (3, 3), activation='relu'),

    # 3. Downsampling using a 2x2 Max Pool, reducing spatial dimensions.
    tf.keras.layers.MaxPool2D(pool_size=(2, 2)),

    # 4. Second convolutional layer with 32 filters, kernel size of 3x3, ReLU activation.
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),

    # 5. Another 2x2 Max Pool for further downsampling.
    tf.keras.layers.MaxPool2D(pool_size=(2, 2)),

    # 6. Flatten the feature maps into a 1D vector for the fully connected layers.
    tf.keras.layers.Flatten(),

    # 7. Dense (fully connected) layer with 64 units and ReLU activation.
    tf.keras.layers.Dense(64, activation='relu'),

    # 8. Dense layer with 128 units and ReLU activation.
    tf.keras.layers.Dense(128, activation='relu'),

    # 9. Output layer with 5 units (classes) and softmax activation for multi-class classification.
    tf.keras.layers.Dense(5, activation='softmax')
```

```
])
```

Listing 8: Model Architecture

Explanation:

- **Input Layer:** Accepts images resized to 28x28 pixels with three color channels (RGB).
- **Convolutional Layers:** Extract spatial features using convolutional filters, progressively increasing the number of filters to capture complex patterns.
- **Max Pooling:** Reduces spatial dimensions, mitigating overfitting and computational load.
- **Flatten Layer:** Converts 2D feature maps into a 1D vector for dense layer processing.
- **Dense Layers:** Learn high-level representations and perform classification.
- **Output Layer:** Utilizes softmax activation to output probabilities across the five rice varieties.

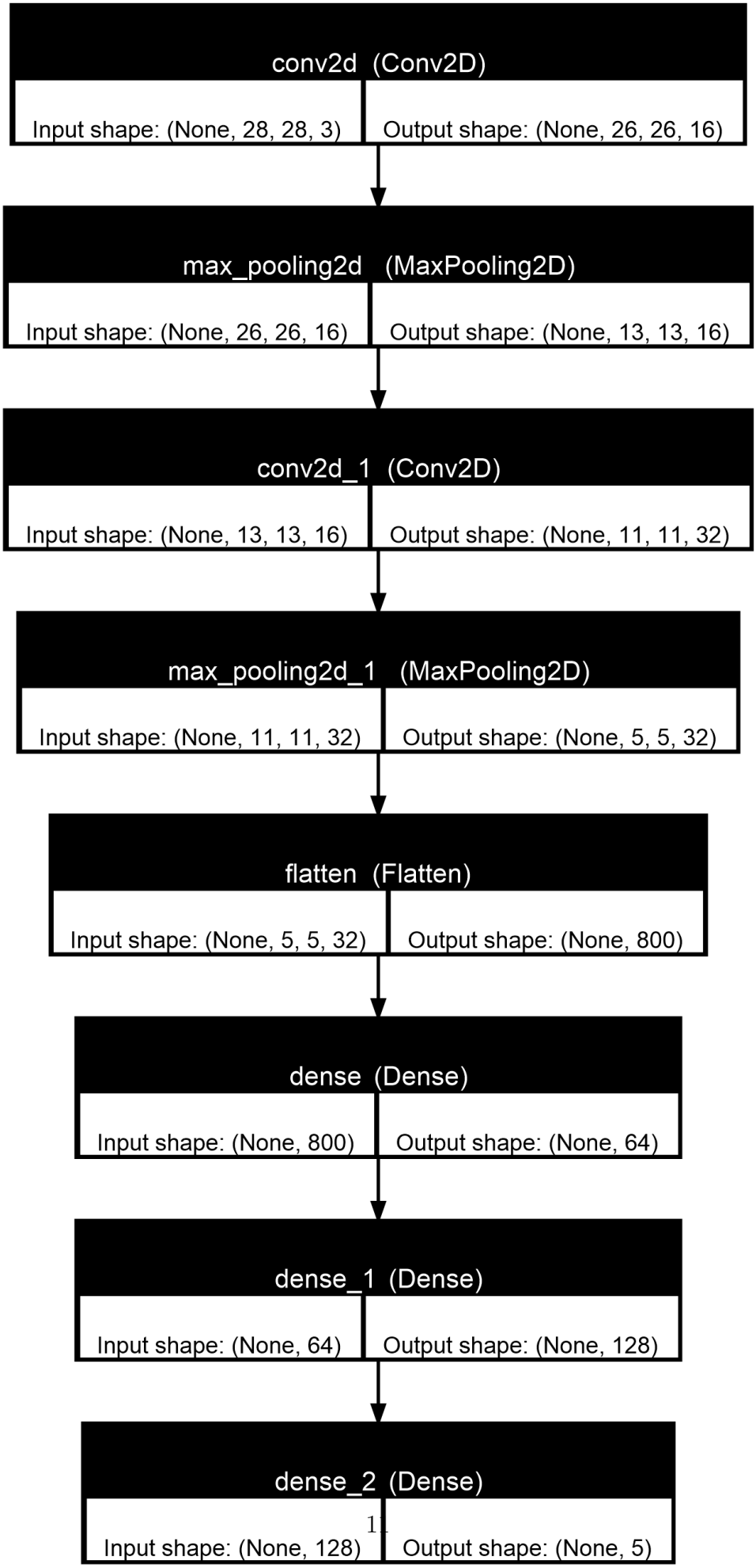

Figure 5: Model architecture

3.7 7. Model Visualization

```
from tensorflow.keras.utils import plot_model

plot_model(
    model,
    to_file='model_plot.png',   # The output file name
    show_shapes=True,           # Display input/output shapes
    show_layer_names=True       # Display layer names
)
```

Listing 9: Model Visualization

3.8 8. Model Compilation

The model is compiled with appropriate optimizer, loss function, and evaluation metrics.

```
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

# Model compilation:
model.compile(
    optimizer=Adam(learning_rate=1e-3),  # You can tune this learning
   rate.
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

Listing 10: Model Compilation

Explanation:

- **Optimizer:** Adam optimizer with a learning rate of 0.001 facilitates efficient convergence.
- **Loss Function:** Categorical cross-entropy is suitable for multi-class classification tasks.
- **Metrics:** Accuracy is monitored to evaluate model performance.

3.9 9. Callbacks Configuration

Callbacks enhance the training process by introducing mechanisms like early stopping and learning rate adjustments.

```
# Callback list:
callbacks_list = [
    EarlyStopping(
        monitor='val_loss',              # What metric to monitor
        patience=3,                      # How many epochs to wait for
   improvement before stopping
        restore_best_weights=True        # Restore model weights from
   the epoch with the best value of the monitored quantity
    ),
    ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.2,                      # How much to reduce the LR by
```

```
        patience=2                          # How many epochs with no
    improvement before reducing
    )
]
```

Listing 11: Callbacks Configuration

Explanation:

- **EarlyStopping:** Halts training if the validation loss does not improve for three consecutive epochs, preventing overfitting.
- **ReduceLROnPlateau:** Decreases the learning rate by a factor of 0.2 if the validation loss stagnates for two epochs, aiding in fine-tuning the model.

3.10 10. Model Training

The model is trained using the prepared data generators, incorporating the defined callbacks for optimized training.

```
# Fit the model:
history = model.fit(
    train_images,
    epochs=10,                          # Number of training epochs
    validation_data=validation_images, # Validation data for monitoring
    callbacks=callbacks_list            # Callbacks for early stopping
    and learning rate reduction
)
```

Listing 12: Model Training

Explanation:

- **Training Process:** The model undergoes 10 epochs of training, balancing computational efficiency with sufficient learning iterations.
- **Validation Monitoring:** Validation data facilitates real-time assessment of model performance, enabling callbacks to function effectively.

3.11 11. Model Evaluation

Post-training, the model's performance is assessed on the test dataset to gauge its generalization capabilities.

```
# Evaluate the trained model on the test dataset (test_images) to
    measure loss and accuracy.
test_loss, test_accuracy = model.evaluate(test_images)
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy}")
```

Listing 13: Model Evaluation

Explanation:

- **Evaluation Metrics:** Loss and accuracy on the test set provide quantitative measures of the model's effectiveness in classifying unseen data.

4 Results

The training and evaluation processes yield insightful metrics that reflect the model's performance.

- **Training Accuracy:** Indicates how well the model learns the training data.
- **Validation Accuracy:** Reflects the model's ability to generalize to unseen data during training.
- **Test Accuracy:** Represents the model's performance on entirely unseen data, serving as the ultimate benchmark.

Training Summary The model began with a training accuracy of roughly 71% and achieved nearly 98% by the final epoch, while validation accuracy progressed from 94.5% to about 97.6%. This small gap between training and validation accuracy (around 1%) indicates good generalization and minimal overfitting. Although there was a brief dip in validation accuracy around the midpoint (Epoch 6), the model recovered quickly, demonstrating that the chosen architecture and hyperparameters effectively learned to classify the rice varieties.

```
Epoch 1/10
C:\Users\HP\anaconda3\Lib\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `su
per().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to
`fit()`, as they will be ignored.
  self._warn_if_super_not_called()
330/330 ━━━━━━━━━━━━━━━━━━━━ 16s 43ms/step - accuracy: 0.7139 - loss: 2.6760 - val_accuracy: 0.9445 - val_loss: 0.1676 - learning_rate: 0.0010
Epoch 2/10
330/330 ━━━━━━━━━━━━━━━━━━━━ 16s 49ms/step - accuracy: 0.9410 - loss: 0.1760 - val_accuracy: 0.9604 - val_loss: 0.1171 - learning_rate: 0.0010
Epoch 3/10
330/330 ━━━━━━━━━━━━━━━━━━━━ 16s 48ms/step - accuracy: 0.9523 - loss: 0.1328 - val_accuracy: 0.9545 - val_loss: 0.1364 - learning_rate: 0.0010
Epoch 4/10
330/330 ━━━━━━━━━━━━━━━━━━━━ 18s 56ms/step - accuracy: 0.9567 - loss: 0.1222 - val_accuracy: 0.9607 - val_loss: 0.1127 - learning_rate: 0.0010
Epoch 5/10
330/330 ━━━━━━━━━━━━━━━━━━━━ 18s 54ms/step - accuracy: 0.9671 - loss: 0.0895 - val_accuracy: 0.9655 - val_loss: 0.1108 - learning_rate: 0.0010
Epoch 6/10
330/330 ━━━━━━━━━━━━━━━━━━━━ 17s 51ms/step - accuracy: 0.9735 - loss: 0.0743 - val_accuracy: 0.9571 - val_loss: 0.1486 - learning_rate: 0.0010
Epoch 7/10
330/330 ━━━━━━━━━━━━━━━━━━━━ 16s 48ms/step - accuracy: 0.9655 - loss: 0.1026 - val_accuracy: 0.9653 - val_loss: 0.0998 - learning_rate: 0.0010
Epoch 8/10
330/330 ━━━━━━━━━━━━━━━━━━━━ 16s 50ms/step - accuracy: 0.9764 - loss: 0.0736 - val_accuracy: 0.9730 - val_loss: 0.0815 - learning_rate: 0.0010
Epoch 9/10
330/330 ━━━━━━━━━━━━━━━━━━━━ 16s 50ms/step - accuracy: 0.9826 - loss: 0.0460 - val_accuracy: 0.9748 - val_loss: 0.0764 - learning_rate: 0.0010
Epoch 10/10
330/330 ━━━━━━━━━━━━━━━━━━━━ 18s 55ms/step - accuracy: 0.9833 - loss: 0.0482 - val_accuracy: 0.9759 - val_loss: 0.0785 - learning_rate: 0.0010
```

Figure 6: Training result

5 Discussion

The developed CNN model demonstrates the potential of deep learning in automating the classification of rice varieties. Data augmentation techniques enhance the model's robustness by introducing variability, mitigating overfitting, and improving generalization. The chosen architecture balances complexity and computational efficiency, making it suitable for large-scale image datasets.

However, certain limitations and areas for improvement are identified:

- **Image Resolution:** Resizing images to 28x28 pixels may lead to loss of critical features. Exploring higher resolutions could enhance feature extraction.
- **Model Complexity:** Incorporating more convolutional layers or leveraging pre-trained models (e.g., ResNet, VGG) might improve accuracy.

- **Hyperparameter Tuning:** Systematic tuning of hyperparameters such as learning rate, batch size, and number of epochs could optimize performance.
- **Dataset Diversity:** Ensuring diverse image conditions (lighting, angles) can further strengthen the model's adaptability.

6 Conclusion

This study successfully implements a TensorFlow-based CNN for classifying five rice varieties using the Rice Image Dataset. Through meticulous data preprocessing, augmentation, and model training, the approach achieves promising classification accuracy. The findings underscore the efficacy of deep learning in agricultural applications, paving the way for more advanced and specialized models in the future.

Future work may involve integrating more sophisticated architectures, expanding the dataset, and deploying the model in real-world applications for automated rice variety classification, thereby contributing to agricultural productivity and quality assurance.

7 References

References

[1] Chollet, F. (2017). *Deep Learning with Python*. Manning Publications.

[2] Abadi, M., et al. (2016). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv preprint arXiv:1603.04467*.

[3] Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825-2830.

[4] Koklu, M., Cinar, I., & Taspinar, Y. S. (2021). Classification of rice varieties with deep learning methods. *Computers and Electronics in Agriculture*, **187**, 106285. https://doi.org/10.1016/j.compag.2021.106285

[5] Cinar, I., & Koklu, M. (2021). Determination of Effective and Specific Physical Features of Rice Varieties by Computer Vision In Exterior Quality Inspection. *Selcuk Journal of Agriculture and Food Sciences*, **35**(3), 229–243. https://doi.org/10.15316/SJAFS.2021.252

[6] Cinar, I., & Koklu, M. (2022). Identification of Rice Varieties Using Machine Learning Algorithms. *Journal of Agricultural Sciences*, **28**(2), 307–325. https://doi.org/10.15832/ankutbd.862482

[7] Cinar, I., & Koklu, M. (2019). Classification of Rice Varieties Using Artificial Intelligence Methods. *International Journal of Intelligent Systems and Applications in Engineering*, **7**(3), 188–194. https://doi.org/10.18201/ijisae.2019355381

8 Appendices

8.1 Appendix A: Complete Code

```
import pandas as pd
import tensorflow as tf
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import ImageGrid
from pathlib import Path
from PIL import Image
import cv2
from tensorflow.keras import layers
from sklearn import preprocessing
import numpy as np

# Step 1: Data Loading and Label Encoding
current_directory = Path(os.getcwd())
dataset_path = current_directory / "rice_img"
image_link = list(dataset_path.glob('**/*.jpg'))
image_name = [x.parents[0].stem for x in image_link]
image_label = preprocessing.LabelEncoder().fit_transform(image_name)
print(f"Number of images found: {len(image_link)}")
print(f"Image labels: {image_label}")

# Step 2: DataFrame Construction
df = pd.DataFrame()
df['link'] = np.array(image_link, dtype=str)
df['name'] = image_name
df['label'] = image_label
print(df.head())

# Step 3: Data Visualization
df.name.value_counts().plot(kind='bar', figsize=(12, 8), grid=True,
    color='blue')
plt.title('Distribution of Rice Varieties')
plt.xlabel('Rice Variety')
plt.ylabel('Number of Images')
plt.show()

fig = plt.figure(1, figsize=(15,15))
grid = ImageGrid(fig, 121, nrows_ncols=(5,4), axes_pad=0.10)
i = 0
for category_id, category in enumerate(df.name.unique()):
    for filepath in df[df['name'] == category]['link'].values[:4]:
        ax = grid[i]
        img = Image.open(filepath)
        ax.imshow(img)
        ax.axis('off')
        if i % 4 == 4-1:
            ax.text(300,100, category, verticalalignment='center',
    fontsize=20, color='black')
        i += 1
plt.show()

```

```
# Step 4: Dataset Splitting
train_df, test_df = train_test_split(df, test_size=0.3, random_state=1,
    stratify=df['label'])

# Step 5: Data Augmentation and Generators
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.15,
    zoom_range=0.15,
    horizontal_flip=True,
    fill_mode="nearest",
    validation_split=0.3
)

test_datagen = ImageDataGenerator(rescale=1./255)

train_images = train_datagen.flow_from_dataframe(
    dataframe=train_df,
    x_col='link',
    y_col='name',
    color_mode='rgb',
    batch_size=32,
    target_size=(28, 28),
    class_mode='categorical',
    subset='training'
)

validation_images = train_datagen.flow_from_dataframe(
    dataframe=train_df,
    x_col='link',
    y_col='name',
    color_mode='rgb',
    batch_size=32,
    target_size=(28, 28),
    class_mode='categorical',
    subset='validation'
)

test_images = test_datagen.flow_from_dataframe(
    dataframe=test_df,
    x_col='link',
    y_col='name',
    color_mode='rgb',
    batch_size=32,
    target_size=(28, 28),
    class_mode='categorical'
)

# Step 6: Model Construction
model = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(28, 28, 3)),
    tf.keras.layers.Conv2D(16, (3, 3), activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=(2, 2)),
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=(2, 2)),
```

```
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(5, activation='softmax')
])

# Step 7: Model Visualization
from tensorflow.keras.utils import plot_model
plot_model(
    model,
    to_file='model_plot.png',
    show_shapes=True,
    show_layer_names=True
)

# Step 8: Model Compilation
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

model.compile(
    optimizer=Adam(learning_rate=1e-3),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Step 9: Callbacks Configuration
callbacks_list = [
    EarlyStopping(
        monitor='val_loss',
        patience=3,
        restore_best_weights=True
    ),
    ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.2,
        patience=2
    )
]

# Step 10: Model Training
history = model.fit(
    train_images,
    epochs=10,
    validation_data=validation_images,
    callbacks=callbacks_list
)

# Step 11: Model Evaluation
test_loss, test_accuracy = model.evaluate(test_images)
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy}")
```

Listing 14: Complete Code

Name	Data Types	Default Task	Attribute Types	# Instances	# Attributes	Year	Download
Rice Image Dataset	5 Class	Classification	Image	75.000	Image	2021	**Download** 10081 downloaded
Citation Request	**1:** KOKLU, M., CINAR, I. and TASPINAR, Y. S. (2021). Classification of rice varieties with deep learning methods. *Computers and Electronics in Agriculture*, 187, 106285. **DOI:** https://doi.org/10.1016/j.compag.2021.106285 **2:** CINAR, I. and KOKLU, M. (2021). Determination of Effective and Specific Physical Features of Rice Varieties by Computer Vision In Exterior Quality Inspection. *Selcuk Journal of Agriculture and Food Sciences*, 35(3), 229-243. **DOI:** https://doi.org/10.15316/SJAFS.2021.252 **3:** CINAR, I. and KOKLU, M. (2022). Identification of Rice Varieties Using Machine Learning Algorithms. *Journal of Agricultural Sciences*, 28 (2), 307-325. **DOI:** https://doi.org/10.15832/ankutbd.862482 **4:** CINAR, I. and KOKLU, M. (2019). Classification of Rice Varieties Using Artificial Intelligence Methods. *International Journal of Intelligent Systems and Applications in Engineering*, 7(3), 188-194. **DOI:** https://doi.org/10.18201/ijisae.2019355381						

Figure 7: Dataset used: https://www.muratkoklu.com/datasets/