**FRONT END: CODING**

import streamlit as st

import tensorflow as tf

from tensorflow.keras.preprocessing import image

import numpy as np

from PIL import Image

import os

# Function to load models

def load\_model(model\_name):

if model\_name == 'CNN':

return tf.keras.models.load\_model('RICE\_CNN\_model.keras')

elif model\_name == 'VGG16':

return tf.keras.models.load\_model('RICE\_VGG\_model.keras')

# Function to preprocess the uploaded image

def preprocess\_image(img):

img = img.resize((224, 224)) # Resize the image to 224x224 (input size for the model)

img = np.array(img) / 255.0 # Normalize pixel values to [0, 1]

img = np.expand\_dims(img, axis=0) # Add batch dimension

return img

# Login Page

def login\_page():

st.title("Login Page")

username = st.text\_input("Username", "")

password = st.text\_input("Password", "", type="password")

if st.button("Login"):

if username == "Admin" and password == "123":

st.session\_state.logged\_in = True

st.success("Login successful!")

else:

st.error("Invalid username or password!")

# Home Page with Enhanced Disease Information

def home\_page():

st.title("Rice Plant Disease Information")

st.write(

"This section provides detailed information about various rice plant diseases. "

"Learn about their symptoms, causes, and control measures to keep your crops healthy."

)

# Healthy Rice Plant - Information in columns

st.subheader("1. Healthy Rice Plant")

st.write(

"A healthy rice plant is characterized by vibrant green leaves and stems, "

"healthy root systems, and no visible lesions, discoloration, or wilting. "

"Healthy plants are resistant to many diseases and show no yellowing or stunting."

)

col1, col2 = st.columns(2)

with col1:

st.write("### Key Features of Healthy Rice Plants:")

st.write("- Strong, upright stems")

st.write("- Green leaves without yellowing")

st.write("- Well-developed root system")

st.write("- Even growth throughout the plant")

with col2:

st.image("healthy\_plant.jpg", caption="Healthy Rice Plant")

# Bacterial Blight - Information in columns

st.subheader("2. Bacterial Blight")

st.write(

"Bacterial blight is a serious disease caused by the bacterium \*Xanthomonas oryzae\*."

" It affects rice plants by causing yellow lesions and wilting of the leaves, particularly "

"during the reproductive stage."

)

col1, col2 = st.columns(2)

with col1:

st.write("### Symptoms of Bacterial Blight:")

st.write("- Yellow lesions on leaves and stems")

st.write("- Leaf tips turning brown and curling")

st.write("- Water-soaked lesions on infected plants")

st.write("- Wilting and stunted growth")

st.write("### Causes and Control:")

st.write("- Caused by the bacterium \*Xanthomonas oryzae\*")

st.write("- Spread through infected seeds and water")

st.write("- Control: Use resistant varieties, crop rotation, proper irrigation management")

with col2:

st.image("bacterial\_blight.jpg", caption="Bacterial Blight")

# Rice Blast - Information in columns

st.subheader("3. Rice Blast")

st.write(

"Rice blast, caused by the fungus \*Magnaporthe oryzae\*, is one of the most destructive "

"diseases of rice. It causes irregular lesions on leaves, stems, and panicles, reducing yields."

)

col1, col2 = st.columns(2)

with col1:

st.write("### Symptoms of Rice Blast:")

st.write("- Irregular lesions on leaves, often with gray centers")

st.write("- Panicles and stems may show dark lesions")

st.write("- Affected plants can show reduced grain formation")

st.write("### Causes and Control:")

st.write("- Caused by the fungal pathogen \*Magnaporthe oryzae\*")

st.write("- Spread by wind, water, and infected seeds")

st.write("- Control: Use resistant varieties, fungicides, and good field sanitation")

with col2:

st.image("rice\_blast.jpg", caption="Rice Blast")

# Sheath Blight - Information in columns

st.subheader("4. Sheath Blight")

st.write(

"Sheath blight is caused by the fungus \*Rhizoctonia solani\* and affects the rice plant's "

"lower leaves and stems. It causes lesions that result in poor grain filling and yield losses."

)

col1, col2 = st.columns(2)

with col1:

st.write("### Symptoms of Sheath Blight:")

st.write("- Lesions appear on lower stems and leaves")

st.write("- Brown, irregular lesions with a water-soaked appearance")

st.write("- Decayed tissue at the base of the plant")

st.write("### Causes and Control:")

st.write("- Caused by the fungal pathogen \*Rhizoctonia solani\*")

st.write("- Spread by infected debris and wet conditions")

st.write("- Control: Use resistant varieties, manage irrigation, avoid excessive nitrogen use")

with col2:

st.image("sheath\_blight.jpg", caption="Sheath Blight")

# Prediction Page

def prediction\_page():

st.title("Rice Plant Disease Prediction")

# Upload image for prediction

uploaded\_file = st.file\_uploader("Choose an image...", type=["jpg", "png", "jpeg"])

if uploaded\_file is not None:

# Display the uploaded image

st.image(uploaded\_file, caption="Uploaded Image")

# Model selection dropdown

model\_choice = st.selectbox("Select Model", ['CNN', 'VGG16'])

# Preprocess and predict on button click

if st.button("Predict"):

# Load the chosen model

model = load\_model(model\_choice)

# Preprocess the uploaded image

img = Image.open(uploaded\_file)

img = preprocess\_image(img)

# Predict the class

predictions = model.predict(img)

class\_idx = np.argmax(predictions, axis=1)[0]

# Map class index to disease name

class\_names = ['Bacterial Blight', 'Healthy Rice Plant', 'Rice Blast', 'Sheath Blight']

prediction = class\_names[class\_idx]

st.success(f"Prediction: {prediction}")

def background():

# CSS code to set the background image

st.markdown(

f"""

<style>

.stApp {{

background-image: url("https://images.pexels.com/photos/1438516/pexels-photo-1438516.jpeg?cs=srgb&dl=pexels-thanh-nguy-n-637271-1438516.jpg&fm=jpg");

background-size: cover;

background-position: center center;

background-repeat: no-repeat;

height: 100vh;

}}

</style>

""",

unsafe\_allow\_html=True

)

# Main function to control page flow

def main():

background()

# Check if the user is logged in

if "logged\_in" not in st.session\_state or not st.session\_state.logged\_in:

login\_page()

else:

# Sidebar for navigation

st.sidebar.title("Navigation")

page = st.sidebar.radio("Go to", ["Home", "Prediction"])

# Show pages based on user selection

if page == "Home":

home\_page()

elif page == "Prediction":

prediction\_page()

# Run the app

if \_\_name\_\_ == "\_\_main\_\_":

main()

**BACKEND: CODING**

import os

import pandas as pd

from glob import glob

import matplotlib.pyplot as plt

import seaborn as sns

from PIL import Image

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import confusion\_matrix, classification\_report

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.applications import VGG16

from tensorflow.keras.preprocessing.image import load\_img, img\_to\_array, ImageDataGenerator

from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

# Define the root folder where the dataset is stored

root\_folder = r'C:\Naveen\PLANT DISEASE RICE (CYNTHIA.M)\DATASET\resized\_raw\_images\resized\_raw\_images'

# Initialize an empty list to store image paths and their labels

data = []

# Iterate through each class (folder) inside the root folder

for class\_name in os.listdir(root\_folder):

class\_folder = os.path.join(root\_folder, class\_name)

# Check if it's a directory (folder)

if os.path.isdir(class\_folder):

# Get all image paths in the class folder (assuming image format is .jpg)

image\_paths = glob(os.path.join(class\_folder, "\*.jpg")) # You can adjust the extension if needed

# For each image, append the image path and the class label (folder name)

for image\_path in image\_paths:

data.append([image\_path, class\_name])

# Create a DataFrame with columns 'image\_path' and 'label'

df = pd.DataFrame(data, columns=['image\_path', 'label'])

# Display the DataFrame

print(df.head())

image\_path label

0 C:\Naveen\PLANT DISEASE RICE (CYNTHIA.M)\DATAS... Bacterial Blight

1 C:\Naveen\PLANT DISEASE RICE (CYNTHIA.M)\DATAS... Bacterial Blight

2 C:\Naveen\PLANT DISEASE RICE (CYNTHIA.M)\DATAS... Bacterial Blight

3 C:\Naveen\PLANT DISEASE RICE (CYNTHIA.M)\DATAS... Bacterial Blight

4 C:\Naveen\PLANT DISEASE RICE (CYNTHIA.M)\DATAS... Bacterial Blight

# Shuffle the DataFrame

df = df.sample(frac=1, random\_state=42).reset\_index(drop=True)

# Display the DataFrame

print(df.head())

image\_path label

0 C:\Naveen\PLANT DISEASE RICE (CYNTHIA.M)\DATAS... Bacterial Blight

1 C:\Naveen\PLANT DISEASE RICE (CYNTHIA.M)\DATAS... Rice Blast

2 C:\Naveen\PLANT DISEASE RICE (CYNTHIA.M)\DATAS... Rice Blast

3 C:\Naveen\PLANT DISEASE RICE (CYNTHIA.M)\DATAS... Bacterial Blight

4 C:\Naveen\PLANT DISEASE RICE (CYNTHIA.M)\DATAS... Sheath Blight

# Check the class distribution

class\_counts = df['label'].value\_counts()

# Plot the distribution of labels (class counts)

plt.figure(figsize=(8, 6))

class\_counts.plot(kind='bar', color='skyblue')

plt.title('Distribution of Classes in Dataset')

plt.xlabel('Class')

plt.ylabel('Number of Images')

plt.xticks(rotation=45)

plt.show()

# Check basic dataframe info

print("\nDataframe Info:")

print(df.info())

Dataframe Info:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 393 entries, 0 to 392

Data columns (total 2 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 image\_path 393 non-null object

1 label 393 non-null object

dtypes: object(2)

memory usage: 6.3+ KB

None

# Check for missing values

missing\_values = df.isnull().sum()

print("\nMissing Values:")

print(missing\_values)

Missing Values:

image\_path 0

label 0

dtype: int64

# Sample some images from each class (example images)

sample\_per\_class = 3

unique\_labels = df['label'].unique()

# Create a plot with sample images from each class

plt.figure(figsize=(12, 12))

for i, label in enumerate(unique\_labels):

sample\_images = df[df['label'] == label].sample(sample\_per\_class)['image\_path'].values

for j, img\_path in enumerate(sample\_images):

plt.subplot(len(unique\_labels), sample\_per\_class, i \* sample\_per\_class + j + 1)

img = Image.open(img\_path)

plt.imshow(img)

plt.axis('off')

plt.title(f"{label} Sample {j+1}")

plt.tight\_layout()

plt.show()

df['label'].value\_counts()

label

Healthy Rice Plant 100

Rice Blast 98

Sheath Blight 98

Bacterial Blight 97

Name: count, dtype: int64

# Resampling: For each class, we will downsample or upsample to 1000 samples

df\_resampled = df.groupby('label').apply(lambda x: x.sample(n=100, random\_state=42, replace=True) if len(x) < 100 else x.sample(n=100, random\_state=42))

# Reset the index after resampling

df\_resampled = df\_resampled.reset\_index(drop=True)

# Check the new class distribution

print("\nNew Class Distribution After Resampling:")

print(df\_resampled['label'].value\_counts())

New Class Distribution After Resampling:

label

Bacterial Blight 100

Healthy Rice Plant 100

Rice Blast 100

Sheath Blight 100

Name: count, dtype: int64

# Optionally, plot the class distribution again after resampling

plt.figure(figsize=(8, 6))

df\_resampled['label'].value\_counts().plot(kind='bar', color='skyblue')

plt.title('Class Distribution After Resampling')

plt.xlabel('Class')

plt.ylabel('Number of Images')

plt.xticks(rotation=45)

plt.show()

# Split the data into features (X) and labels (y)

X = df\_resampled['image\_path'] # Image file paths (features)

y = df\_resampled['label'] # Labels (target)

# Split the data into training and testing sets (80% train, 20% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42, stratify=y)

# Create DataFrames for train and test sets

train\_df = pd.DataFrame({'image\_path': X\_train, 'label': y\_train})

test\_df = pd.DataFrame({'image\_path': X\_test, 'label': y\_test})

# Verify the distribution of classes in the train and test sets

print("\nClass Distribution in Train Set:")

print(train\_df['label'].value\_counts())

print("\nClass Distribution in Test Set:")

print(test\_df['label'].value\_counts())

Class Distribution in Train Set:

label

Healthy Rice Plant 80

Rice Blast 80

Bacterial Blight 80

Sheath Blight 80

Name: count, dtype: int64

Class Distribution in Test Set:

label

Rice Blast 20

Healthy Rice Plant 20

Bacterial Blight 20

Sheath Blight 20

Name: count, dtype: int64

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

train\_df['label'].value\_counts().plot(kind='bar', color='skyblue')

plt.title('Class Distribution in Train Set')

plt.xlabel('Class')

plt.ylabel('Number of Images')

plt.subplot(1, 2, 2)

test\_df['label'].value\_counts().plot(kind='bar', color='lightcoral')

plt.title('Class Distribution in Test Set')

plt.xlabel('Class')

plt.ylabel('Number of Images')

plt.tight\_layout()

plt.show()

# Set image size and batch size

image\_size = (224, 224) # Updated image size

batch\_size = 32

# Create ImageDataGenerators for data augmentation and scaling

train\_datagen = ImageDataGenerator(

rescale=1./255, # Normalize pixel values to [0, 1]

rotation\_range=20, # Random rotations

width\_shift\_range=0.2, # Horizontal shift

height\_shift\_range=0.2, # Vertical shift

shear\_range=0.2, # Shear transformation

zoom\_range=0.2, # Zoom in/out

horizontal\_flip=True, # Random horizontal flip

fill\_mode='nearest' # Fill mode for missing pixels

)

test\_datagen = ImageDataGenerator(rescale=1./255) # Only rescaling for test data

# Flow from dataframe to load images for training

train\_generator = train\_datagen.flow\_from\_dataframe(

dataframe=train\_df,

x\_col='image\_path', # Column with image paths

y\_col='label', # Column with labels

target\_size=image\_size, # Resize images to 224x224

batch\_size=batch\_size,

class\_mode='sparse', # Integer labels

shuffle=True,

seed=42

)

# Flow from dataframe to load images for testing

test\_generator = test\_datagen.flow\_from\_dataframe(

dataframe=test\_df,

x\_col='image\_path', # Column with image paths

y\_col='label', # Column with labels

target\_size=image\_size, # Resize images to 224x224

batch\_size=batch\_size,

class\_mode='sparse', # Integer labels

shuffle=False, # No shuffling for evaluation

seed=42

)

Found 320 validated image filenames belonging to 4 classes.

Found 80 validated image filenames belonging to 4 classes.

# Build the CNN model

model = models.Sequential([

# First Convolutional Block

layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(224, 224, 3)),

layers.MaxPooling2D((2, 2)),

# Second Convolutional Block

layers.Conv2D(64, (3, 3), activation='relu'),

layers.MaxPooling2D((2, 2)),

# Third Convolutional Block

layers.Conv2D(128, (3, 3), activation='relu'),

layers.MaxPooling2D((2, 2)),

# Flatten the output

layers.Flatten(),

# Fully Connected (Dense) Layers

layers.Dense(128, activation='relu'),

layers.Dropout(0.5), # Dropout layer to prevent overfitting

layers.Dense(4, activation='softmax') # 3 output classes: EarlyBlight, LateBlight, Healthy

])

# Compile the model

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy', # Sparse labels (integer-based loss)

metrics=['accuracy'])

# Model summary to check architecture

model.summary()

**Model: "sequential\_1"**

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓

┃ **Layer (type)** ┃ **Output Shape** ┃ **Param #** ┃

┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩

│ conv2d\_3 (Conv2D) │ (None, 222, 222, 32) │ 896 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ max\_pooling2d\_3 (MaxPooling2D) │ (None, 111, 111, 32) │ 0 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ conv2d\_4 (Conv2D) │ (None, 109, 109, 64) │ 18,496 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ max\_pooling2d\_4 (MaxPooling2D) │ (None, 54, 54, 64) │ 0 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ conv2d\_5 (Conv2D) │ (None, 52, 52, 128) │ 73,856 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ max\_pooling2d\_5 (MaxPooling2D) │ (None, 26, 26, 128) │ 0 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ flatten\_1 (Flatten) │ (None, 86528) │ 0 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ dense\_2 (Dense) │ (None, 128) │ 11,075,712 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ dropout\_1 (Dropout) │ (None, 128) │ 0 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ dense\_3 (Dense) │ (None, 4) │ 516 │

└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘

**Total params:** 11,169,476 (42.61 MB)

**Trainable params:** 11,169,476 (42.61 MB)

**Non-trainable params:** 0 (0.00 B)

*# Learning Rate Scheduler (Reduce learning rate when validation loss plateaus)*

lr\_scheduler **=** ReduceLROnPlateau(monitor**=**'val\_loss',

factor**=**0.5, *# Reduce LR by 50%*

patience**=**3, *# If no improvement for 3 epochs*

min\_lr**=**1e-6, *# Minimum learning rate*

verbose**=**1)

*# Train the model*

history **=** model.fit(

train\_generator,

epochs**=**50, *# You can adjust the number of epochs*

validation\_data**=**test\_generator,

callbacks**=**[lr\_scheduler] *# Include the callbacks*

)

Epoch 1/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **12s** 702ms/step - accuracy: 0.2761 - loss: 2.1062 - val\_accuracy: 0.5125 - val\_loss: 1.2748 - learning\_rate: 0.0010

Epoch 2/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **10s** 685ms/step - accuracy: 0.4405 - loss: 1.2426 - val\_accuracy: 0.4125 - val\_loss: 1.0993 - learning\_rate: 0.0010

Epoch 3/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **10s** 646ms/step - accuracy: 0.5105 - loss: 1.1630 - val\_accuracy: 0.6375 - val\_loss: 0.9754 - learning\_rate: 0.0010

Epoch 4/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 676ms/step - accuracy: 0.6023 - loss: 1.0293 - val\_accuracy: 0.6125 - val\_loss: 0.8924 - learning\_rate: 0.0010

Epoch 5/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **10s** 686ms/step - accuracy: 0.6067 - loss: 0.9613 - val\_accuracy: 0.6125 - val\_loss: 0.8728 - learning\_rate: 0.0010

Epoch 6/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **10s** 665ms/step - accuracy: 0.5688 - loss: 1.0146 - val\_accuracy: 0.7000 - val\_loss: 0.7707 - learning\_rate: 0.0010

Epoch 7/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 671ms/step - accuracy: 0.6787 - loss: 0.8138 - val\_accuracy: 0.6125 - val\_loss: 0.8286 - learning\_rate: 0.0010

Epoch 8/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **10s** 681ms/step - accuracy: 0.5902 - loss: 0.9071 - val\_accuracy: 0.7250 - val\_loss: 0.7240 - learning\_rate: 0.0010

Epoch 9/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 651ms/step - accuracy: 0.6245 - loss: 0.8981 - val\_accuracy: 0.7000 - val\_loss: 0.6700 - learning\_rate: 0.0010

Epoch 10/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 689ms/step - accuracy: 0.7140 - loss: 0.7461 - val\_accuracy: 0.7625 - val\_loss: 0.6523 - learning\_rate: 0.0010

Epoch 11/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **11s** 829ms/step - accuracy: 0.7066 - loss: 0.8024 - val\_accuracy: 0.7625 - val\_loss: 0.6308 - learning\_rate: 0.0010

Epoch 12/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 637ms/step - accuracy: 0.7518 - loss: 0.6428 - val\_accuracy: 0.7500 - val\_loss: 0.6403 - learning\_rate: 0.0010

Epoch 13/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 638ms/step - accuracy: 0.7119 - loss: 0.6508 - val\_accuracy: 0.7500 - val\_loss: 0.6680 - learning\_rate: 0.0010

Epoch 14/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **0s** 610ms/step - accuracy: 0.7407 - loss: 0.7056

Epoch 14: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 655ms/step - accuracy: 0.7426 - loss: 0.7032 - val\_accuracy: 0.7750 - val\_loss: 0.6410 - learning\_rate: 0.0010

Epoch 15/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 638ms/step - accuracy: 0.7831 - loss: 0.6417 - val\_accuracy: 0.7250 - val\_loss: 0.5635 - learning\_rate: 5.0000e-04

Epoch 16/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 637ms/step - accuracy: 0.7964 - loss: 0.5314 - val\_accuracy: 0.7875 - val\_loss: 0.5233 - learning\_rate: 5.0000e-04

Epoch 17/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 644ms/step - accuracy: 0.7707 - loss: 0.5725 - val\_accuracy: 0.7625 - val\_loss: 0.4938 - learning\_rate: 5.0000e-04

Epoch 18/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 639ms/step - accuracy: 0.8093 - loss: 0.4846 - val\_accuracy: 0.7375 - val\_loss: 0.6341 - learning\_rate: 5.0000e-04

Epoch 19/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 643ms/step - accuracy: 0.7871 - loss: 0.5080 - val\_accuracy: 0.7625 - val\_loss: 0.4506 - learning\_rate: 5.0000e-04

Epoch 20/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 644ms/step - accuracy: 0.8416 - loss: 0.4602 - val\_accuracy: 0.7750 - val\_loss: 0.4404 - learning\_rate: 5.0000e-04

Epoch 21/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 644ms/step - accuracy: 0.7748 - loss: 0.5374 - val\_accuracy: 0.8125 - val\_loss: 0.5713 - learning\_rate: 5.0000e-04

Epoch 22/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 640ms/step - accuracy: 0.7933 - loss: 0.5363 - val\_accuracy: 0.8000 - val\_loss: 0.4980 - learning\_rate: 5.0000e-04

Epoch 23/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **0s** 590ms/step - accuracy: 0.7605 - loss: 0.5493

Epoch 23: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 635ms/step - accuracy: 0.7643 - loss: 0.5439 - val\_accuracy: 0.7625 - val\_loss: 0.4772 - learning\_rate: 5.0000e-04

Epoch 24/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 644ms/step - accuracy: 0.8545 - loss: 0.4666 - val\_accuracy: 0.7750 - val\_loss: 0.5864 - learning\_rate: 2.5000e-04

Epoch 25/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 638ms/step - accuracy: 0.8446 - loss: 0.4409 - val\_accuracy: 0.8125 - val\_loss: 0.4196 - learning\_rate: 2.5000e-04

Epoch 26/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 636ms/step - accuracy: 0.8249 - loss: 0.4146 - val\_accuracy: 0.7875 - val\_loss: 0.4529 - learning\_rate: 2.5000e-04

Epoch 27/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 636ms/step - accuracy: 0.8480 - loss: 0.3877 - val\_accuracy: 0.7625 - val\_loss: 0.4314 - learning\_rate: 2.5000e-04

Epoch 28/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **0s** 602ms/step - accuracy: 0.8445 - loss: 0.3977

Epoch 28: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 646ms/step - accuracy: 0.8444 - loss: 0.3994 - val\_accuracy: 0.7625 - val\_loss: 0.5887 - learning\_rate: 2.5000e-04

Epoch 29/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 636ms/step - accuracy: 0.8145 - loss: 0.4974 - val\_accuracy: 0.8125 - val\_loss: 0.3940 - learning\_rate: 1.2500e-04

Epoch 30/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 634ms/step - accuracy: 0.8619 - loss: 0.3856 - val\_accuracy: 0.7750 - val\_loss: 0.4516 - learning\_rate: 1.2500e-04

Epoch 31/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 646ms/step - accuracy: 0.8682 - loss: 0.3670 - val\_accuracy: 0.8625 - val\_loss: 0.3885 - learning\_rate: 1.2500e-04

Epoch 32/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 638ms/step - accuracy: 0.8579 - loss: 0.3619 - val\_accuracy: 0.8000 - val\_loss: 0.4245 - learning\_rate: 1.2500e-04

Epoch 33/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 649ms/step - accuracy: 0.8680 - loss: 0.3224 - val\_accuracy: 0.8625 - val\_loss: 0.3535 - learning\_rate: 1.2500e-04

Epoch 34/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 628ms/step - accuracy: 0.8628 - loss: 0.3314 - val\_accuracy: 0.7875 - val\_loss: 0.4317 - learning\_rate: 1.2500e-04

Epoch 35/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 645ms/step - accuracy: 0.8558 - loss: 0.3684 - val\_accuracy: 0.8000 - val\_loss: 0.4272 - learning\_rate: 1.2500e-04

Epoch 36/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **0s** 597ms/step - accuracy: 0.8683 - loss: 0.3387

Epoch 36: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 641ms/step - accuracy: 0.8683 - loss: 0.3381 - val\_accuracy: 0.7875 - val\_loss: 0.4233 - learning\_rate: 1.2500e-04

Epoch 37/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 652ms/step - accuracy: 0.9023 - loss: 0.2964 - val\_accuracy: 0.8000 - val\_loss: 0.4265 - learning\_rate: 6.2500e-05

Epoch 38/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 645ms/step - accuracy: 0.8784 - loss: 0.2950 - val\_accuracy: 0.8000 - val\_loss: 0.4284 - learning\_rate: 6.2500e-05

Epoch 39/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **0s** 593ms/step - accuracy: 0.9126 - loss: 0.3057

Epoch 39: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 637ms/step - accuracy: 0.9120 - loss: 0.3077 - val\_accuracy: 0.7875 - val\_loss: 0.4829 - learning\_rate: 6.2500e-05

Epoch 40/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 639ms/step - accuracy: 0.8875 - loss: 0.3209 - val\_accuracy: 0.8000 - val\_loss: 0.4468 - learning\_rate: 3.1250e-05

Epoch 41/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 644ms/step - accuracy: 0.9041 - loss: 0.3186 - val\_accuracy: 0.8000 - val\_loss: 0.4080 - learning\_rate: 3.1250e-05

Epoch 42/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **0s** 593ms/step - accuracy: 0.8747 - loss: 0.3481

Epoch 42: ReduceLROnPlateau reducing learning rate to 1.5625000742147677e-05.

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 637ms/step - accuracy: 0.8742 - loss: 0.3481 - val\_accuracy: 0.8000 - val\_loss: 0.4232 - learning\_rate: 3.1250e-05

Epoch 43/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 648ms/step - accuracy: 0.8362 - loss: 0.3652 - val\_accuracy: 0.8000 - val\_loss: 0.4257 - learning\_rate: 1.5625e-05

Epoch 44/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 652ms/step - accuracy: 0.8808 - loss: 0.3565 - val\_accuracy: 0.8000 - val\_loss: 0.4230 - learning\_rate: 1.5625e-05

Epoch 45/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **0s** 590ms/step - accuracy: 0.8845 - loss: 0.3435

Epoch 45: ReduceLROnPlateau reducing learning rate to 7.812500371073838e-06.

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 640ms/step - accuracy: 0.8845 - loss: 0.3425 - val\_accuracy: 0.8000 - val\_loss: 0.4112 - learning\_rate: 1.5625e-05

Epoch 46/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 640ms/step - accuracy: 0.8823 - loss: 0.3062 - val\_accuracy: 0.8000 - val\_loss: 0.4182 - learning\_rate: 7.8125e-06

Epoch 47/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 650ms/step - accuracy: 0.8809 - loss: 0.2963 - val\_accuracy: 0.8000 - val\_loss: 0.4250 - learning\_rate: 7.8125e-06

Epoch 48/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **0s** 595ms/step - accuracy: 0.8924 - loss: 0.2897

Epoch 48: ReduceLROnPlateau reducing learning rate to 3.906250185536919e-06.

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 640ms/step - accuracy: 0.8908 - loss: 0.2914 - val\_accuracy: 0.8000 - val\_loss: 0.4267 - learning\_rate: 7.8125e-06

Epoch 49/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 644ms/step - accuracy: 0.8604 - loss: 0.3703 - val\_accuracy: 0.8000 - val\_loss: 0.4220 - learning\_rate: 3.9063e-06

Epoch 50/50

**10/10** ━━━━━━━━━━━━━━━━━━━━ **9s** 639ms/step - accuracy: 0.8795 - loss: 0.2916 - val\_accuracy: 0.8000 - val\_loss: 0.4225 - learning\_rate: 3.9063e-06

*# Plot accuracy and loss graphs*

plt.figure(figsize**=**(12, 6))

​

*# Accuracy plot*

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'], label**=**'Training Accuracy')

plt.plot(history.history['val\_accuracy'], label**=**'Validation Accuracy')

plt.title('Training and Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

​

*# Loss plot*

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'], label**=**'Training Loss')

plt.plot(history.history['val\_loss'], label**=**'Validation Loss')

plt.title('Training and Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

​

plt.tight\_layout()

plt.show()

y\_true **=** test\_generator.labels *# True labels from the test set*

y\_pred\_prob **=** model.predict(test\_generator, verbose**=**1)

**3/3** ━━━━━━━━━━━━━━━━━━━━ **0s** 103ms/step

*# Convert probabilities to class labels*

y\_pred **=** np.argmax(y\_pred\_prob, axis**=**1) *# Get the index of the highest probability*

*# Generate the Confusion Matrix*

cm **=** confusion\_matrix(y\_true, y\_pred)

*# Plot the Confusion Matrix using Seaborn Heatmap*

plt.figure(figsize**=**(8, 6))

sns.heatmap(cm, annot**=True**, fmt**=**'d', cmap**=**'Blues', xticklabels**=**test\_generator.class\_indices.keys(),

yticklabels**=**test\_generator.class\_indices.keys(), cbar**=False**, annot\_kws**=**{"size": 16})

plt.title('Confusion Matrix')

plt.xlabel('Predicted Labels')

plt.ylabel('True Labels')

plt.show()

*# Generate the Classification Report*

report **=** classification\_report(y\_true, y\_pred, target\_names**=**test\_generator.class\_indices.keys())

print(report)

precision recall f1-score support

Bacterial Blight 0.62 1.00 0.77 20

Healthy Rice Plant 0.71 0.50 0.59 20

Rice Blast 1.00 0.70 0.82 20

Sheath Blight 1.00 1.00 1.00 20

accuracy 0.80 80

macro avg 0.83 0.80 0.80 80

weighted avg 0.83 0.80 0.80 80

model.save('RICE\_CNN\_model.keras')

*# Load VGG16 pre-trained model with weights from ImageNet*

base\_model **=** VGG16(weights**=**'imagenet', include\_top**=False**, input\_shape**=**(224, 224, 3))

​

*# Freeze the layers of the VGG model*

base\_model.trainable **=** **False**

​

*# Add custom layers on top of the VGG base model*

model\_vgg **=** models.Sequential([

base\_model, *# Add VGG16 base model*

layers.Flatten(), *# Global average pooling layer to reduce dimensionality*

layers.Dense(128, activation**=**'relu'), *# Fully connected layer*

layers.Dropout(0.5), *# Dropout layer to prevent overfitting*

layers.Dense(4, activation**=**'softmax') *# Output layer for 4 classes*

])

*# Compile the model*

model\_vgg.compile(optimizer**=**'adam',

loss**=**'sparse\_categorical\_crossentropy', *# Sparse labels (integer-based loss)*

metrics**=**['accuracy'])

​

*# Model summary to check architecture*

model\_vgg.summary()

**Model: "sequential\_3"**

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓

┃ **Layer (type)** ┃ **Output Shape** ┃ **Param #** ┃

┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩

│ vgg16 (Functional) │ (None, 7, 7, 512) │ 14,714,688 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ flatten\_3 (Flatten) │ (None, 25088) │ 0 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ dense\_6 (Dense) │ (None, 128) │ 3,211,392 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ dropout\_3 (Dropout) │ (None, 128) │ 0 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ dense\_7 (Dense) │ (None, 4) │ 516 │

└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘

**Total params:** 17,926,596 (68.38 MB)

**Trainable params:** 3,211,908 (12.25 MB)

**Non-trainable params:** 14,714,688 (56.13 MB)

*# Train the model with the callbacks*

history **=** model\_vgg.fit(

train\_generator,

epochs**=**100, *# You can increase the number of epochs*

validation\_data**=**test\_generator,

callbacks**=**[lr\_scheduler] *# Include the callbacks*

)

Epoch 1/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **44s** 4s/step - accuracy: 0.2610 - loss: 3.0962 - val\_accuracy: 0.6125 - val\_loss: 0.9774 - learning\_rate: 0.0010

Epoch 2/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **41s** 4s/step - accuracy: 0.5191 - loss: 1.1224 - val\_accuracy: 0.6375 - val\_loss: 0.8828 - learning\_rate: 0.0010

Epoch 3/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **42s** 4s/step - accuracy: 0.6028 - loss: 0.9935 - val\_accuracy: 0.7125 - val\_loss: 0.7441 - learning\_rate: 0.0010

Epoch 4/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **44s** 4s/step - accuracy: 0.6241 - loss: 0.8652 - val\_accuracy: 0.6750 - val\_loss: 0.7278 - learning\_rate: 0.0010

Epoch 5/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **46s** 4s/step - accuracy: 0.6763 - loss: 0.8223 - val\_accuracy: 0.7250 - val\_loss: 0.7130 - learning\_rate: 0.0010

Epoch 6/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **44s** 4s/step - accuracy: 0.6785 - loss: 0.8162 - val\_accuracy: 0.7000 - val\_loss: 0.6533 - learning\_rate: 0.0010

Epoch 7/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **46s** 4s/step - accuracy: 0.6906 - loss: 0.7794 - val\_accuracy: 0.7375 - val\_loss: 0.6762 - learning\_rate: 0.0010

Epoch 8/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **44s** 4s/step - accuracy: 0.7094 - loss: 0.7193 - val\_accuracy: 0.8000 - val\_loss: 0.5726 - learning\_rate: 0.0010

Epoch 9/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **45s** 4s/step - accuracy: 0.7120 - loss: 0.7015 - val\_accuracy: 0.7750 - val\_loss: 0.5808 - learning\_rate: 0.0010

Epoch 10/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **44s** 4s/step - accuracy: 0.7065 - loss: 0.7072 - val\_accuracy: 0.7375 - val\_loss: 0.5722 - learning\_rate: 0.0010

Epoch 11/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **45s** 4s/step - accuracy: 0.7285 - loss: 0.6201 - val\_accuracy: 0.8000 - val\_loss: 0.5458 - learning\_rate: 0.0010

Epoch 12/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **44s** 4s/step - accuracy: 0.7335 - loss: 0.7033 - val\_accuracy: 0.8375 - val\_loss: 0.4945 - learning\_rate: 0.0010

Epoch 13/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **44s** 4s/step - accuracy: 0.7862 - loss: 0.5587 - val\_accuracy: 0.8250 - val\_loss: 0.4926 - learning\_rate: 0.0010

Epoch 14/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **45s** 4s/step - accuracy: 0.7534 - loss: 0.6269 - val\_accuracy: 0.8000 - val\_loss: 0.5299 - learning\_rate: 0.0010

Epoch 15/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **45s** 4s/step - accuracy: 0.7769 - loss: 0.6174 - val\_accuracy: 0.8625 - val\_loss: 0.4921 - learning\_rate: 0.0010

Epoch 16/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **43s** 4s/step - accuracy: 0.7289 - loss: 0.6366 - val\_accuracy: 0.7875 - val\_loss: 0.5111 - learning\_rate: 0.0010

Epoch 17/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **43s** 4s/step - accuracy: 0.7418 - loss: 0.5701 - val\_accuracy: 0.8375 - val\_loss: 0.4737 - learning\_rate: 0.0010

Epoch 18/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **43s** 4s/step - accuracy: 0.7420 - loss: 0.6218 - val\_accuracy: 0.8250 - val\_loss: 0.4488 - learning\_rate: 0.0010

Epoch 19/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **45s** 4s/step - accuracy: 0.7778 - loss: 0.5489 - val\_accuracy: 0.8375 - val\_loss: 0.4633 - learning\_rate: 0.0010

Epoch 20/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **47s** 4s/step - accuracy: 0.7760 - loss: 0.5314 - val\_accuracy: 0.8250 - val\_loss: 0.4317 - learning\_rate: 0.0010

Epoch 21/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **46s** 4s/step - accuracy: 0.8197 - loss: 0.4825 - val\_accuracy: 0.8375 - val\_loss: 0.4190 - learning\_rate: 0.0010

Epoch 22/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **42s** 4s/step - accuracy: 0.7630 - loss: 0.5427 - val\_accuracy: 0.8375 - val\_loss: 0.4328 - learning\_rate: 0.0010

Epoch 23/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **43s** 4s/step - accuracy: 0.8104 - loss: 0.4923 - val\_accuracy: 0.8750 - val\_loss: 0.4038 - learning\_rate: 0.0010

Epoch 24/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **45s** 4s/step - accuracy: 0.8135 - loss: 0.4767 - val\_accuracy: 0.8500 - val\_loss: 0.3969 - learning\_rate: 0.0010

Epoch 25/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **46s** 4s/step - accuracy: 0.7910 - loss: 0.5283 - val\_accuracy: 0.9125 - val\_loss: 0.3696 - learning\_rate: 0.0010

Epoch 26/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **46s** 4s/step - accuracy: 0.7825 - loss: 0.4867 - val\_accuracy: 0.9000 - val\_loss: 0.3786 - learning\_rate: 0.0010

Epoch 27/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **45s** 4s/step - accuracy: 0.8106 - loss: 0.5209 - val\_accuracy: 0.8375 - val\_loss: 0.4504 - learning\_rate: 0.0010

Epoch 28/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **0s** 3s/step - accuracy: 0.7562 - loss: 0.5238

Epoch 28: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.

**10/10** ━━━━━━━━━━━━━━━━━━━━ **44s** 4s/step - accuracy: 0.7553 - loss: 0.5278 - val\_accuracy: 0.8875 - val\_loss: 0.3903 - learning\_rate: 0.0010

Epoch 29/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **46s** 4s/step - accuracy: 0.8282 - loss: 0.4779 - val\_accuracy: 0.9000 - val\_loss: 0.3593 - learning\_rate: 5.0000e-04

Epoch 30/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **45s** 4s/step - accuracy: 0.8444 - loss: 0.4048 - val\_accuracy: 0.9000 - val\_loss: 0.4067 - learning\_rate: 5.0000e-04

Epoch 31/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **43s** 4s/step - accuracy: 0.7841 - loss: 0.4847 - val\_accuracy: 0.8875 - val\_loss: 0.4027 - learning\_rate: 5.0000e-04

Epoch 32/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **0s** 3s/step - accuracy: 0.8115 - loss: 0.4562

Epoch 32: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.

**10/10** ━━━━━━━━━━━━━━━━━━━━ **41s** 4s/step - accuracy: 0.8116 - loss: 0.4563 - val\_accuracy: 0.9125 - val\_loss: 0.3623 - learning\_rate: 5.0000e-04

Epoch 33/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **43s** 4s/step - accuracy: 0.7934 - loss: 0.5090 - val\_accuracy: 0.9125 - val\_loss: 0.3675 - learning\_rate: 2.5000e-04

Epoch 34/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **49s** 5s/step - accuracy: 0.8425 - loss: 0.4053 - val\_accuracy: 0.9125 - val\_loss: 0.3649 - learning\_rate: 2.5000e-04

Epoch 35/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **0s** 3s/step - accuracy: 0.8416 - loss: 0.3789

Epoch 35: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.

**10/10** ━━━━━━━━━━━━━━━━━━━━ **46s** 4s/step - accuracy: 0.8421 - loss: 0.3789 - val\_accuracy: 0.9250 - val\_loss: 0.3709 - learning\_rate: 2.5000e-04

Epoch 36/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **42s** 4s/step - accuracy: 0.8201 - loss: 0.4822 - val\_accuracy: 0.9250 - val\_loss: 0.3684 - learning\_rate: 1.2500e-04

Epoch 37/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **45s** 4s/step - accuracy: 0.8815 - loss: 0.3376 - val\_accuracy: 0.9250 - val\_loss: 0.3653 - learning\_rate: 1.2500e-04

Epoch 38/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **41s** 4s/step - accuracy: 0.7955 - loss: 0.4076 - val\_accuracy: 0.9250 - val\_loss: 0.3516 - learning\_rate: 1.2500e-04

Epoch 39/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **44s** 4s/step - accuracy: 0.8144 - loss: 0.4519 - val\_accuracy: 0.9250 - val\_loss: 0.3416 - learning\_rate: 1.2500e-04

Epoch 40/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **47s** 4s/step - accuracy: 0.8440 - loss: 0.4206 - val\_accuracy: 0.9125 - val\_loss: 0.3434 - learning\_rate: 1.2500e-04

Epoch 41/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **46s** 4s/step - accuracy: 0.8432 - loss: 0.4317 - val\_accuracy: 0.9250 - val\_loss: 0.3508 - learning\_rate: 1.2500e-04

Epoch 42/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **0s** 3s/step - accuracy: 0.8748 - loss: 0.3403

Epoch 42: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.

**10/10** ━━━━━━━━━━━━━━━━━━━━ **44s** 4s/step - accuracy: 0.8740 - loss: 0.3409 - val\_accuracy: 0.9250 - val\_loss: 0.3512 - learning\_rate: 1.2500e-04

Epoch 43/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **41s** 4s/step - accuracy: 0.8758 - loss: 0.3124 - val\_accuracy: 0.9250 - val\_loss: 0.3456 - learning\_rate: 6.2500e-05

Epoch 44/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **44s** 4s/step - accuracy: 0.8638 - loss: 0.3616 - val\_accuracy: 0.9250 - val\_loss: 0.3395 - learning\_rate: 6.2500e-05

Epoch 45/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **45s** 4s/step - accuracy: 0.8637 - loss: 0.3626 - val\_accuracy: 0.9250 - val\_loss: 0.3411 - learning\_rate: 6.2500e-05

Epoch 46/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **48s** 5s/step - accuracy: 0.8719 - loss: 0.3478 - val\_accuracy: 0.9250 - val\_loss: 0.3465 - learning\_rate: 6.2500e-05

Epoch 47/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **0s** 3s/step - accuracy: 0.8490 - loss: 0.3958

Epoch 47: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.

**10/10** ━━━━━━━━━━━━━━━━━━━━ **44s** 4s/step - accuracy: 0.8483 - loss: 0.3964 - val\_accuracy: 0.9250 - val\_loss: 0.3450 - learning\_rate: 6.2500e-05

Epoch 48/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **46s** 4s/step - accuracy: 0.8348 - loss: 0.3742 - val\_accuracy: 0.9250 - val\_loss: 0.3465 - learning\_rate: 3.1250e-05

Epoch 49/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **43s** 4s/step - accuracy: 0.8329 - loss: 0.3807 - val\_accuracy: 0.9250 - val\_loss: 0.3472 - learning\_rate: 3.1250e-05

Epoch 50/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **0s** 3s/step - accuracy: 0.8602 - loss: 0.3563

Epoch 50: ReduceLROnPlateau reducing learning rate to 1.5625000742147677e-05.

**10/10** ━━━━━━━━━━━━━━━━━━━━ **44s** 4s/step - accuracy: 0.8601 - loss: 0.3552 - val\_accuracy: 0.9250 - val\_loss: 0.3476 - learning\_rate: 3.1250e-05

Epoch 51/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **43s** 4s/step - accuracy: 0.8545 - loss: 0.3671 - val\_accuracy: 0.9250 - val\_loss: 0.3482 - learning\_rate: 1.5625e-05

Epoch 52/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **47s** 5s/step - accuracy: 0.8775 - loss: 0.3772 - val\_accuracy: 0.9250 - val\_loss: 0.3471 - learning\_rate: 1.5625e-05

Epoch 53/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **0s** 3s/step - accuracy: 0.8510 - loss: 0.3793

Epoch 53: ReduceLROnPlateau reducing learning rate to 7.812500371073838e-06.

**10/10** ━━━━━━━━━━━━━━━━━━━━ **43s** 4s/step - accuracy: 0.8500 - loss: 0.3806 - val\_accuracy: 0.9250 - val\_loss: 0.3479 - learning\_rate: 1.5625e-05

Epoch 54/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **45s** 4s/step - accuracy: 0.8795 - loss: 0.3355 - val\_accuracy: 0.9250 - val\_loss: 0.3475 - learning\_rate: 7.8125e-06

Epoch 55/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **42s** 4s/step - accuracy: 0.8639 - loss: 0.3713 - val\_accuracy: 0.9250 - val\_loss: 0.3467 - learning\_rate: 7.8125e-06

Epoch 56/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **0s** 3s/step - accuracy: 0.8487 - loss: 0.3403

Epoch 56: ReduceLROnPlateau reducing learning rate to 3.906250185536919e-06.

**10/10** ━━━━━━━━━━━━━━━━━━━━ **42s** 4s/step - accuracy: 0.8488 - loss: 0.3423 - val\_accuracy: 0.9250 - val\_loss: 0.3462 - learning\_rate: 7.8125e-06

Epoch 57/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **41s** 4s/step - accuracy: 0.8501 - loss: 0.3559 - val\_accuracy: 0.9250 - val\_loss: 0.3462 - learning\_rate: 3.9063e-06

Epoch 58/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **41s** 4s/step - accuracy: 0.8740 - loss: 0.3371 - val\_accuracy: 0.9250 - val\_loss: 0.3466 - learning\_rate: 3.9063e-06

Epoch 59/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **0s** 3s/step - accuracy: 0.8979 - loss: 0.3742

Epoch 59: ReduceLROnPlateau reducing learning rate to 1.9531250927684596e-06.

**10/10** ━━━━━━━━━━━━━━━━━━━━ **41s** 4s/step - accuracy: 0.8964 - loss: 0.3749 - val\_accuracy: 0.9250 - val\_loss: 0.3464 - learning\_rate: 3.9063e-06

Epoch 60/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **41s** 4s/step - accuracy: 0.8829 - loss: 0.3201 - val\_accuracy: 0.9250 - val\_loss: 0.3464 - learning\_rate: 1.9531e-06

Epoch 61/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **42s** 4s/step - accuracy: 0.8533 - loss: 0.3740 - val\_accuracy: 0.9250 - val\_loss: 0.3462 - learning\_rate: 1.9531e-06

Epoch 62/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **0s** 3s/step - accuracy: 0.8400 - loss: 0.3687

Epoch 62: ReduceLROnPlateau reducing learning rate to 1e-06.

**10/10** ━━━━━━━━━━━━━━━━━━━━ **41s** 4s/step - accuracy: 0.8400 - loss: 0.3699 - val\_accuracy: 0.9250 - val\_loss: 0.3461 - learning\_rate: 1.9531e-06

Epoch 63/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **42s** 4s/step - accuracy: 0.8732 - loss: 0.3633 - val\_accuracy: 0.9250 - val\_loss: 0.3461 - learning\_rate: 1.0000e-06

Epoch 64/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **41s** 4s/step - accuracy: 0.8592 - loss: 0.3873 - val\_accuracy: 0.9250 - val\_loss: 0.3460 - learning\_rate: 1.0000e-06

Epoch 65/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **42s** 4s/step - accuracy: 0.8856 - loss: 0.3095 - val\_accuracy: 0.9250 - val\_loss: 0.3459 - learning\_rate: 1.0000e-06

Epoch 66/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **42s** 4s/step - accuracy: 0.8715 - loss: 0.3564 - val\_accuracy: 0.9250 - val\_loss: 0.3458 - learning\_rate: 1.0000e-06

Epoch 67/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **42s** 4s/step - accuracy: 0.8498 - loss: 0.3842 - val\_accuracy: 0.9250 - val\_loss: 0.3458 - learning\_rate: 1.0000e-06

Epoch 68/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **42s** 4s/step - accuracy: 0.8841 - loss: 0.3249 - val\_accuracy: 0.9250 - val\_loss: 0.3457 - learning\_rate: 1.0000e-06

Epoch 69/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **42s** 4s/step - accuracy: 0.8603 - loss: 0.3507 - val\_accuracy: 0.9250 - val\_loss: 0.3457 - learning\_rate: 1.0000e-06

Epoch 70/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **43s** 4s/step - accuracy: 0.8499 - loss: 0.3788 - val\_accuracy: 0.9250 - val\_loss: 0.3456 - learning\_rate: 1.0000e-06

Epoch 71/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **44s** 4s/step - accuracy: 0.8605 - loss: 0.3664 - val\_accuracy: 0.9250 - val\_loss: 0.3457 - learning\_rate: 1.0000e-06

Epoch 72/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **43s** 4s/step - accuracy: 0.8321 - loss: 0.3811 - val\_accuracy: 0.9250 - val\_loss: 0.3457 - learning\_rate: 1.0000e-06

Epoch 73/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **42s** 4s/step - accuracy: 0.8362 - loss: 0.3887 - val\_accuracy: 0.9250 - val\_loss: 0.3456 - learning\_rate: 1.0000e-06

Epoch 74/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **43s** 4s/step - accuracy: 0.8855 - loss: 0.3742 - val\_accuracy: 0.9250 - val\_loss: 0.3456 - learning\_rate: 1.0000e-06

Epoch 75/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **42s** 4s/step - accuracy: 0.8391 - loss: 0.3817 - val\_accuracy: 0.9250 - val\_loss: 0.3455 - learning\_rate: 1.0000e-06

Epoch 76/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **41s** 4s/step - accuracy: 0.8532 - loss: 0.3975 - val\_accuracy: 0.9250 - val\_loss: 0.3454 - learning\_rate: 1.0000e-06

Epoch 77/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **42s** 4s/step - accuracy: 0.8869 - loss: 0.3277 - val\_accuracy: 0.9250 - val\_loss: 0.3452 - learning\_rate: 1.0000e-06

Epoch 78/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **44s** 4s/step - accuracy: 0.8620 - loss: 0.3608 - val\_accuracy: 0.9250 - val\_loss: 0.3450 - learning\_rate: 1.0000e-06

Epoch 79/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **43s** 4s/step - accuracy: 0.8400 - loss: 0.3974 - val\_accuracy: 0.9250 - val\_loss: 0.3448 - learning\_rate: 1.0000e-06

Epoch 80/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **42s** 4s/step - accuracy: 0.8214 - loss: 0.4195 - val\_accuracy: 0.9250 - val\_loss: 0.3448 - learning\_rate: 1.0000e-06

Epoch 81/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **41s** 4s/step - accuracy: 0.8838 - loss: 0.3212 - val\_accuracy: 0.9250 - val\_loss: 0.3446 - learning\_rate: 1.0000e-06

Epoch 82/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **41s** 4s/step - accuracy: 0.8448 - loss: 0.3808 - val\_accuracy: 0.9250 - val\_loss: 0.3445 - learning\_rate: 1.0000e-06

Epoch 83/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **41s** 4s/step - accuracy: 0.8612 - loss: 0.4015 - val\_accuracy: 0.9250 - val\_loss: 0.3444 - learning\_rate: 1.0000e-06

Epoch 84/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **41s** 4s/step - accuracy: 0.8744 - loss: 0.3394 - val\_accuracy: 0.9250 - val\_loss: 0.3443 - learning\_rate: 1.0000e-06

Epoch 85/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **41s** 4s/step - accuracy: 0.8903 - loss: 0.3296 - val\_accuracy: 0.9250 - val\_loss: 0.3442 - learning\_rate: 1.0000e-06

Epoch 86/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **42s** 4s/step - accuracy: 0.8463 - loss: 0.4096 - val\_accuracy: 0.9250 - val\_loss: 0.3440 - learning\_rate: 1.0000e-06

Epoch 87/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **43s** 4s/step - accuracy: 0.8890 - loss: 0.3418 - val\_accuracy: 0.9250 - val\_loss: 0.3440 - learning\_rate: 1.0000e-06

Epoch 88/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **42s** 4s/step - accuracy: 0.8522 - loss: 0.3610 - val\_accuracy: 0.9250 - val\_loss: 0.3439 - learning\_rate: 1.0000e-06

Epoch 89/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **42s** 4s/step - accuracy: 0.8595 - loss: 0.3755 - val\_accuracy: 0.9250 - val\_loss: 0.3438 - learning\_rate: 1.0000e-06

Epoch 90/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **42s** 4s/step - accuracy: 0.8589 - loss: 0.3811 - val\_accuracy: 0.9250 - val\_loss: 0.3437 - learning\_rate: 1.0000e-06

Epoch 91/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **42s** 4s/step - accuracy: 0.8966 - loss: 0.3191 - val\_accuracy: 0.9250 - val\_loss: 0.3436 - learning\_rate: 1.0000e-06

Epoch 92/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **42s** 4s/step - accuracy: 0.8411 - loss: 0.3694 - val\_accuracy: 0.9250 - val\_loss: 0.3435 - learning\_rate: 1.0000e-06

Epoch 93/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **44s** 4s/step - accuracy: 0.8508 - loss: 0.4138 - val\_accuracy: 0.9250 - val\_loss: 0.3435 - learning\_rate: 1.0000e-06

Epoch 94/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **42s** 4s/step - accuracy: 0.8596 - loss: 0.3830 - val\_accuracy: 0.9250 - val\_loss: 0.3435 - learning\_rate: 1.0000e-06

Epoch 95/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **42s** 4s/step - accuracy: 0.8162 - loss: 0.4617 - val\_accuracy: 0.9250 - val\_loss: 0.3436 - learning\_rate: 1.0000e-06

Epoch 96/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **42s** 4s/step - accuracy: 0.8799 - loss: 0.3630 - val\_accuracy: 0.9250 - val\_loss: 0.3438 - learning\_rate: 1.0000e-06

Epoch 97/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **42s** 4s/step - accuracy: 0.8950 - loss: 0.3334 - val\_accuracy: 0.9250 - val\_loss: 0.3439 - learning\_rate: 1.0000e-06

Epoch 98/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **42s** 4s/step - accuracy: 0.8515 - loss: 0.3599 - val\_accuracy: 0.9250 - val\_loss: 0.3438 - learning\_rate: 1.0000e-06

Epoch 99/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **42s** 4s/step - accuracy: 0.8452 - loss: 0.3963 - val\_accuracy: 0.9250 - val\_loss: 0.3437 - learning\_rate: 1.0000e-06

Epoch 100/100

**10/10** ━━━━━━━━━━━━━━━━━━━━ **42s** 4s/step - accuracy: 0.8541 - loss: 0.3351 - val\_accuracy: 0.9250 - val\_loss: 0.3437 - learning\_rate: 1.0000e-06

*# Plot accuracy and loss graphs*

plt.figure(figsize**=**(12, 6))

​

*# Accuracy plot*

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'], label**=**'Training Accuracy')

plt.plot(history.history['val\_accuracy'], label**=**'Validation Accuracy')

plt.title('Training and Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

​

*# Loss plot*

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'], label**=**'Training Loss')

plt.plot(history.history['val\_loss'], label**=**'Validation Loss')

plt.title('Training and Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

​

plt.tight\_layout()

plt.show()

*# Generate Predictions on Test Data*

y\_true **=** test\_generator.labels *# True labels from the test set*

y\_pred\_prob **=** model\_vgg.predict(test\_generator, verbose**=**1)

​

*# Convert probabilities to class labels*

y\_pred **=** np.argmax(y\_pred\_prob, axis**=**1) *# Get the index of the highest probability*

​

*# Generate the Confusion Matrix*

cm **=** confusion\_matrix(y\_true, y\_pred)

​

*# Plot the Confusion Matrix using Seaborn Heatmap*

plt.figure(figsize**=**(8, 6))

sns.heatmap(cm, annot**=True**, fmt**=**'d', cmap**=**'Blues', xticklabels**=**test\_generator.class\_indices.keys(),

yticklabels**=**test\_generator.class\_indices.keys(), cbar**=False**, annot\_kws**=**{"size": 16})

plt.title('Confusion Matrix')

plt.xlabel('Predicted Labels')

plt.ylabel('True Labels')

plt.show()

**3/3** ━━━━━━━━━━━━━━━━━━━━ **8s** 2s/step

In [132]:

*# Generate the Classification Report*

report **=** classification\_report(y\_true, y\_pred, target\_names**=**test\_generator.class\_indices.keys())

print(report)

precision recall f1-score support

Bacterial Blight 0.90 0.95 0.93 20

Healthy Rice Plant 0.87 1.00 0.93 20

Rice Blast 0.94 0.75 0.83 20

Sheath Blight 1.00 1.00 1.00 20

accuracy 0.93 80

macro avg 0.93 0.93 0.92 80

weighted avg 0.93 0.93 0.92 80

model\_vgg.save('RICE\_VGG\_model.keras')

​