#Business Understanding

Smallholder Commercial Wheat Farmers are in need a diagnostic tool to detect, diagnose and treat wheat crop pests and diseases to prevent yield reduction. and the main goal of this project is to develop an accessible,accurate and easy to use diagnostic tool that empowers smallholder wheat farmers to detect,diagnose and treat pests and diseases early,therby minimizing and improving productivity

#Data Understanding

This dataset is designed to empower researchers and developers in creating robust machine learning models for classifying various wheat plant diseases. It offers a collection of high-resolution images showcasing real-world wheat diseases without the use of artificial augmentation techniques.

#Problem Statement

Smallholder wheat farmers face significant challenges in identifying and managing pests and diseases due to limited access to timely and accurate diagnostic tools. This results in delayed interventions, reduced crop yields, and economic losses. There is a need for an affordable, user-friendly solution that provides real-time diagnosis and actionable treatment recommendations to help farmers mitigate these issues effectively

#Main Objectives

Early detection devlops a diagnostic tool that identifies wheat crop pests and diseases at an early stage to prevent significant damage

Accurate diagnosis leverage image classification technology to ensure high accuracy in identifying specific pests and diseases affecting wheat crops

Actionable treatment recommendations provides tailored, practical, and easy-to-implement treatment suggestions to farmers.

Accessibility designs a user-friendly web platform that is affordable and usable in low-resource settings, including areas with limited internet connectivity.

Farmer empowerment will help equip smallholder farmers with technology to make informed decisions, reducing dependency on external experts.

Increased productivity may minimize yield losses by enabling timely interventions, leading to improved crop performance and farmer incomes.

Sustainability promotes targeted pest and disease management to reduce the overuse of chemicals, fostering environmentally sustainable farming practices

#Import the relevant libraries

```
import pandas as pd
import numpy as np

import tensorflow as tf
import pathlib

import os

import matplotlib.pyplot as plt
import matplotlib.image as mpimg

from PIL import Image

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout


import datetime

import tensorflow as tf
```

#Load the dataset

```
import kagglehub

path = kagglehub.dataset_download("kushagra3204/wheat-plant-diseases")

print("Path to dataset files:", path)

Downloading from
https://www.kaggle.com/api/v1/datasets/download/kushagra3204/wheat-
plant-diseases?dataset_version_number=6...
```

```
100%|██████████| 6.09G/6.09G [00:56<00:00, 115MB/s]

Extracting files...



Path to dataset files:
/root/.cache/kagglehub/datasets/kushagra3204/wheat-plant-diseases/
versions/6
```

#Understanding the dataset

This dataset is designed to empower researchers and developers in creating robust machine learning models for classifying various wheat plant diseases. It offers a collection of high-resolution images showcasing real-world wheat diseases without the use of artificial augmentation techniques.

```
def get_class_distribution(dataset_path):
    class_distribution = {}
    for class_name in os.listdir(dataset_path):
        class_dir = os.path.join(dataset_path, class_name)
        if os.path.isdir(class_dir):
            class_distribution[class_name] =
len(os.listdir(class_dir))
    return class_distribution

def plot_class_distribution(class_distribution):
    classes = list(class_distribution.keys())
    counts = list(class_distribution.values())

    plt.figure(figsize=(10, 5))
    plt.bar(classes, counts, color="skyblue")
    plt.xlabel("Classes")
    plt.ylabel("Number of Images")
    plt.title("Dataset Class Distribution")
    plt.xticks(rotation=45)
    plt.show()

def display_sample_images(dataset_path, n_samples=3):
    classes = os.listdir(dataset_path)
    plt.figure(figsize=(15, 10))

    for i, class_name in enumerate(classes):
        class_dir = os.path.join(dataset_path, class_name)
        if os.path.isdir(class_dir):
            images = os.listdir(class_dir)[:n_samples]
            for j, img_name in enumerate(images):
                img_path = os.path.join(class_dir, img_name)
                img = Image.open(img_path)
                plt.subplot(len(classes), n_samples, i * n_samples + j
+ 1)
```

```
                plt.imshow(img)
                plt.title(class_name)
                plt.axis("off")

    plt.tight_layout()
    plt.show()
```

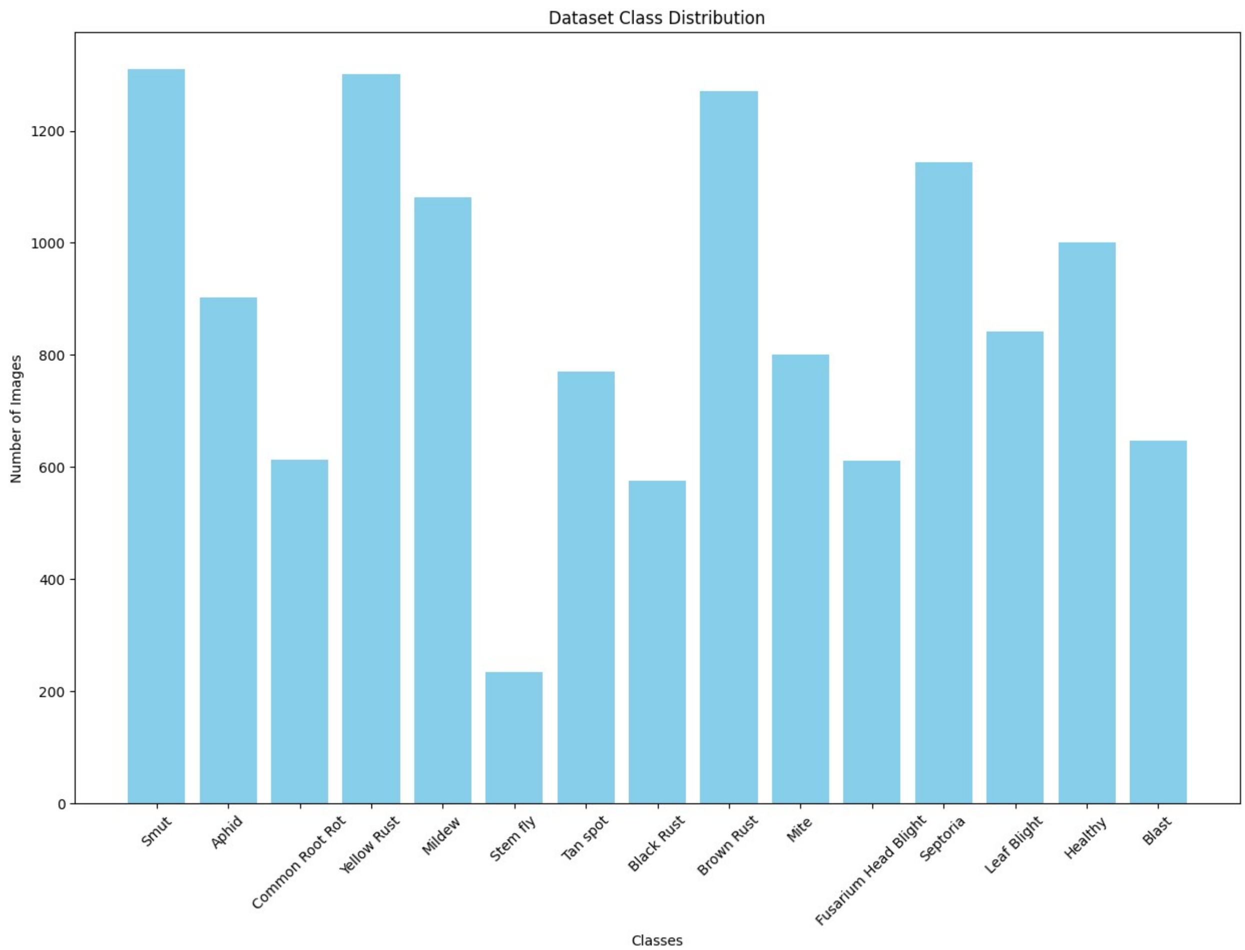
#Exploring the dataset and coming up with visualizations

By exploring the dataset we gain insights into its structure, sample images, class distribution, and image properties. This understanding helpes in making informed decisions when preprocessing the data and training our model.

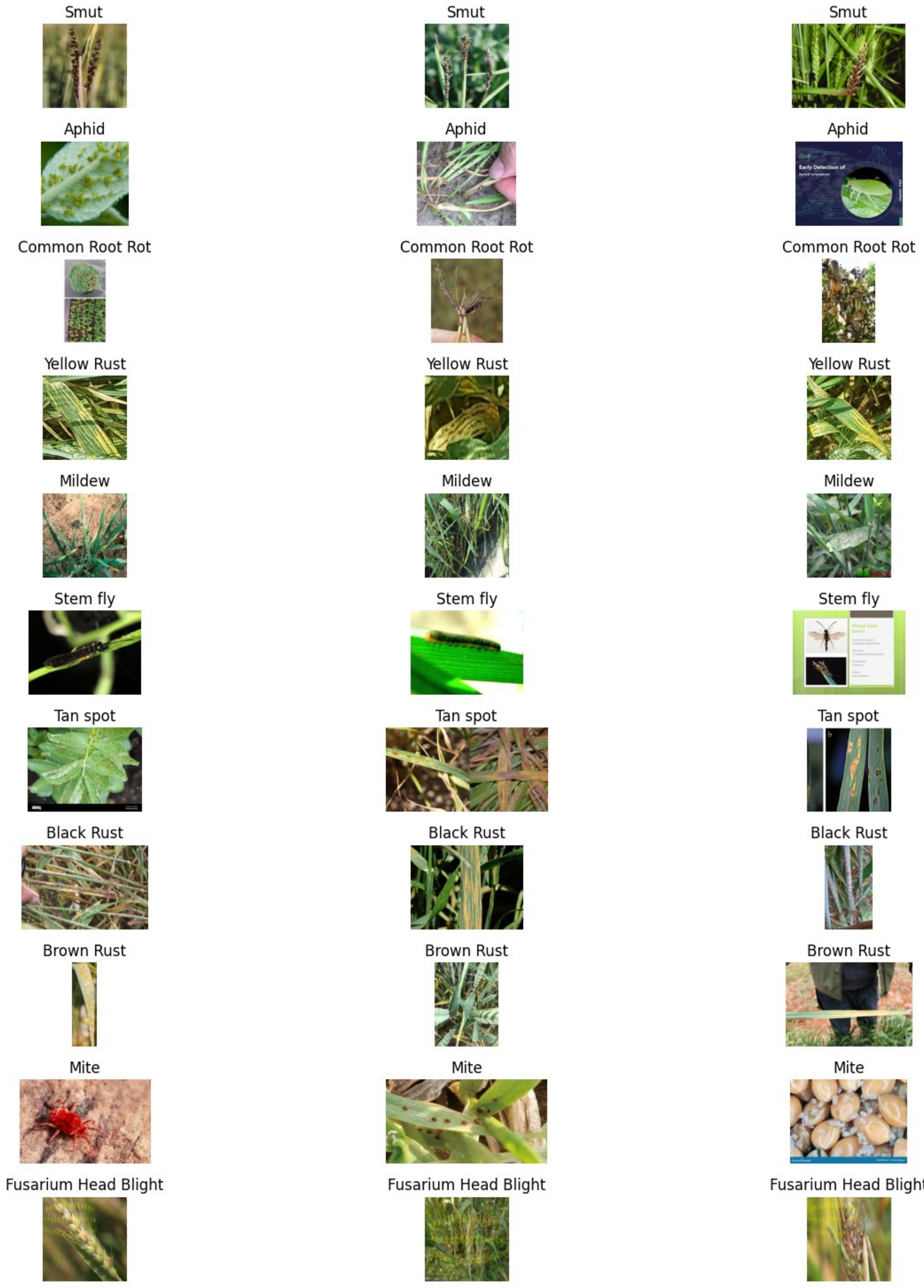
```
# Update dataset_path to point to the directory containing the class
folders
dataset_path = os.path.join(path, "data", "train")

print("Path to dataset files:", dataset_path)

def get_class_distribution(dataset_path):
    class_distribution = {}
    for class_name in os.listdir(dataset_path):
        class_dir = os.path.join(dataset_path, class_name)
        if os.path.isdir(class_dir):
            class_distribution[class_name] =
len(os.listdir(class_dir))
    return class_distribution

def plot_class_distribution(class_distribution):
    classes = list(class_distribution.keys())
    counts = list(class_distribution.values())

    plt.figure(figsize=(15, 10))
    plt.bar(classes, counts, color="skyblue")
    plt.xlabel("Classes")
    plt.ylabel("Number of Images")
    plt.title("Dataset Class Distribution")
    plt.xticks(rotation=45)
    plt.show()

def display_sample_images(dataset_path, n_samples=3):
    classes = os.listdir(dataset_path)
    plt.figure(figsize=(15, 20))

    for i, class_name in enumerate(classes):
        class_dir = os.path.join(dataset_path, class_name)
        if os.path.isdir(class_dir):
            images = os.listdir(class_dir)[:n_samples]
            for j, img_name in enumerate(images):
```

```
                img_path = os.path.join(class_dir, img_name)
                img = Image.open(img_path)
                plt.subplot(len(classes), n_samples, i * n_samples + j
+ 1)
                plt.imshow(img)
                plt.title(class_name)
                plt.axis("off")

    plt.tight_layout()
    plt.show()

if __name__ == "__main__":
    # Check class distribution
    class_distribution = get_class_distribution(dataset_path)
    print("Class Distribution:", class_distribution)

    # Plot class distribution
    plot_class_distribution(class_distribution)

    # Display sample images
    display_sample_images(dataset_path)
```

```
Path to dataset files:
/root/.cache/kagglehub/datasets/kushagra3204/wheat-plant-diseases/
versions/6/data/train
Class Distribution: {'Smut': 1310, 'Aphid': 903, 'Common Root Rot':
614, 'Yellow Rust': 1301, 'Mildew': 1081, 'Stem fly': 234, 'Tan spot':
770, 'Black Rust': 576, 'Brown Rust': 1271, 'Mite': 800, 'Fusarium
Head Blight': 611, 'Septoria': 1144, 'Leaf Blight': 842, 'Healthy':
1000, 'Blast': 647}
```



Dataset Class Distribution
Number of Images
0
200
400
600
800
1000
1200
Smut
Aphid
Common Root Rot
Yellow Rust
Mildew
Stem fly
Tan spot
Black Rust
Brown Rust
Mite
Fusarium Head Blight
Septoria
Leaf Blight
Healthy
Blast
Classes


Smut
Smut
Smut
Aphid
Aphid
Aphid
Common Root Rot
Common Root Rot
Common Root Rot
Yellow Rust
Yellow Rust
Yellow Rust
Mildew
Mildew
Mildew
Stem fly
Stem fly
Stem fly
Tan spot
Tan spot
Tan spot
Black Rust
Black Rust
Black Rust
Brown Rust
Brown Rust
Brown Rust
Mite
Mite
Mite
Fusarium Head Blight
Fusarium Head Blight
Fusarium Head Blight

#Data Preprocessing This step involves preparing the data for the model an also includes tasks like resizing images, normalizing pixel values, data augmentation, and splitting the data into training and validation.

```
#resize and normalize images
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow

#split into training,validation and test sets
train_dir = os.path.join(path, "data", "train")
test_dir = os.path.join(path, "data", "test")
val_dir = os.path.join(path, "data", "val")

train_dir

{"type":"string"}
```

#Model Architecture

Building a Convolutional Neural Network (CNN) model. A simple model might consist of several convolutional layers followed by pooling layers and a few dense layers.

Evaluate the model on the validation and test sets to assess its performance. Monitor metrics such as accuracy, loss, precision, recall, and F1-score

```
#compiling the model
def train_and_evaluate_model(model, train_generator,
validation_generator, test_generator, epochs=30):
    """
    Train the model, evaluate on validation and test sets, plot
training history (loss and accuracy),
    and plot train vs test accuracy and train vs test loss.
    """
    start = datetime.datetime.now()

    # Train the model
    history = model.fit(
        train_generator,
        epochs=epochs,
        validation_data=validation_generator,
        verbose=1
    )

    # Record the training duration
    end = datetime.datetime.now()
    training_duration = end - start
    print(f"Training completed in: {training_duration}")

    # Evaluate on validation set
```

```
    validation_loss, validation_accuracy =
model.evaluate(validation_generator)
    print("Validation Loss:", validation_loss)
    print("Validation Accuracy:", validation_accuracy)

    # Evaluate on test set
    test_loss, test_accuracy = model.evaluate(test_generator)
    print("Test Loss:", test_loss)
    print("Test Accuracy:", test_accuracy)

    # Plot training history (loss and accuracy)
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title('Training and Validation Loss')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation
Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.title('Training and Validation Accuracy')
    plt.legend()

    plt.show()

    # Plot train vs test accuracy
    plt.figure(figsize=(8, 6))
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation
Accuracy')
    plt.axhline(y=test_accuracy, color='r', linestyle='--',
label='Test Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.title('Train vs Test Accuracy')
    plt.legend()
    plt.show()

    # Plot train vs test loss
    plt.figure(figsize=(8, 6))
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.axhline(y=test_loss, color='r', linestyle='--', label='Test
Loss')
```

```
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title('Train vs Test Loss')
    plt.legend()
    plt.show()

    return history

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout

def cnn_model(input_shape):
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu',
input_shape=input_shape),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])
    return model

# Define input shape based on your image dimensions (e.g., (height,
width, channels))
input_shape = (224, 224, 3)

# Create the baseline CNN model
baseline_model = cnn_model(input_shape)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/
convolutional/base_conv.py:107: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)
```

```
baseline_model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

baseline_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_3 (Dense)	(None, 128)	11,075,712
dropout_2 (Dropout)	(None, 128)	0
dense_4 (Dense)	(None, 1)	129

Total params: 11,169,089 (42.61 MB)

Trainable params: 11,169,089 (42.61 MB)

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

```python
import os

# Assuming your dataset is in a directory named 'wheat-plant-diseases'
# Modify this path to reflect your actual dataset location
dataset_dir = '/root/.cache/kagglehub/datasets/kushagra3204/wheat-
plant-diseases/versions/6/data/'

train_dir = os.path.join(dataset_dir, 'train')
test_dir = os.path.join(dataset_dir, 'test')
val_dir = os.path.join(dataset_dir, 'val')


train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=32)
```

```
Found 13104 images belonging to 15 classes.
```

```python
import tensorflow as tf

# num_classes should be 15 for your wheat disease dataset
num_classes = 15  # Set the number of classes to 15
baseline_model.add(tf.keras.layers.Dense(num_classes,
activation='softmax'))

# Compile the model with an appropriate loss function for multi-class
classification
baseline_model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])


baseline_model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical')

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical')
```

```
Found 13104 images belonging to 15 classes.
Found 750 images belonging to 15 classes.
```

```python
def train_and_evaluate_model(model, train_generator, test_generator,
epochs):
    """
    Trains and evaluates a Keras model.

    Args:
        model: The Keras model to train.
        train_generator: The training data generator.
        test_generator: The test data generator. # Added documentation
for test_generator
        epochs: The number of training epochs.

    Returns:
        A history object containing the training and validation
metrics.
    """

    history = model.fit(
        train_generator,
        epochs=epochs,
        validation_data=test_generator
    )

    return history

#train model
history = train_and_evaluate_model(baseline_model, train_generator,
test_generator, epochs=30)
```

```
Epoch 1/30
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/
data_adapters/py_dataset_adapter.py:122: UserWarning: Your `PyDataset`
class should call `super().__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()
```

```
410/410 ━━━━━━━━━━━━━━━━━━━━ 1705s 4s/step - accuracy: 0.0927 - loss:
2.6957 - val_accuracy: 0.0667 - val_loss: 2.7185
Epoch 2/30
410/410 ━━━━━━━━━━━━━━━━━━━━ 1780s 4s/step - accuracy: 0.0967 - loss:
2.6616 - val_accuracy: 0.0667 - val_loss: 2.7353
Epoch 3/30
410/410 ━━━━━━━━━━━━━━━━━━━━ 1749s 4s/step - accuracy: 0.0932 - loss:
2.6525 - val_accuracy: 0.0667 - val_loss: 2.7500
Epoch 4/30
```

```
410/410 ━━━━━━━━━━━━━━━━━━━━ 1745s 4s/step - accuracy: 0.0960 - loss:
2.6472 - val_accuracy: 0.0667 - val_loss: 2.7605
Epoch 5/30
410/410 ━━━━━━━━━━━━━━━━━━━━ 1667s 4s/step - accuracy: 0.0959 - loss:
2.6453 - val_accuracy: 0.0667 - val_loss: 2.7682
Epoch 6/30
410/410 ━━━━━━━━━━━━━━━━━━━━ 1746s 4s/step - accuracy: 0.1045 - loss:
2.6403 - val_accuracy: 0.0667 - val_loss: 2.7732
Epoch 7/30
410/410 ━━━━━━━━━━━━━━━━━━━━ 1666s 4s/step - accuracy: 0.0981 - loss:
2.6434 - val_accuracy: 0.0667 - val_loss: 2.7765
Epoch 8/30
410/410 ━━━━━━━━━━━━━━━━━━━━ 1673s 4s/step - accuracy: 0.0996 - loss:
2.6416 - val_accuracy: 0.0667 - val_loss: 2.7796
Epoch 9/30
410/410 ━━━━━━━━━━━━━━━━━━━━ 1666s 4s/step - accuracy: 0.0988 - loss:
2.6428 - val_accuracy: 0.0667 - val_loss: 2.7813
Epoch 10/30
 61/410 ━━━━━━━━━━━━━━━━━━━━ 23:36 4s/step - accuracy: 0.1010 - loss:
2.6447
```

Conclusion:from the above output the validation accuracy was 0.067 and the test accuracy 0.0984 we tried to improve the model preformance by reducing overfitting

#Building CNN TUNED Baseline Model

```
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt

def train_and_evaluate_model_es(model, train_generator,
validation_generator, test_generator, epochs=30):
    """
    Train the model, evaluate on validation and test sets, plot
training history (loss and accuracy),
    and plot train vs test accuracy and train vs test loss.
    """
    # Define early stopping callback
    early_stopping = EarlyStopping(monitor='val_loss', patience=3,
restore_best_weights=True)

    # Train the model
    history = model.fit(
        train_generator,
        epochs=epochs,
        validation_data=validation_generator,
        callbacks=[early_stopping],  # Pass the early stopping
callback
        verbose=1
    )
```

```
    # Evaluate on validation set
    validation_loss, validation_accuracy =
model.evaluate(validation_generator, verbose=0)
    print("Validation Loss:", validation_loss)
    print("Validation Accuracy:", validation_accuracy)

    # Evaluate on test set
    test_loss, test_accuracy = model.evaluate(test_generator,
verbose=0)
    print("Test Loss:", test_loss)
    print("Test Accuracy:", test_accuracy)

    # Plot training history (loss and accuracy)
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title('Training and Validation Loss')

    plt.subplot(1, 2, 2)
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation
Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.title('Training and Validation Accuracy')
    plt.legend()

    plt.show()

    # Plot train vs test accuracy
    plt.figure(figsize=(8, 6))
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation
Accuracy')
    plt.axhline(y=test_accuracy, color='r', linestyle='--',
label='Test Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.title('Train vs Test Accuracy')
    plt.legend()
    plt.show()

    # Plot train vs test loss
    plt.figure(figsize=(8, 6))
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
```

```
    plt.axhline(y=test_loss, color='r', linestyle='--', label='Test
Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title('Train vs Test Loss')
    plt.legend()
    plt.show()

    return history

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout

def new_cnn_model(input_shape):
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu',
input_shape=input_shape),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(256, activation='relu'),
        Dropout(0.5),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])
    return model

# Define input shape based on your image dimensions (e.g., (height,
width, channels))
input_shape = (224, 224, 3)

# Create the new CNN model
new_model = new_cnn_model(input_shape)

# Compile the model
new_model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# Print model summary
new_model.summary()
```

```
Model: "sequential_2"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┓
━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                          ┃ Output Shape                 ┃
```

```
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇
━━━━━━━━━━━━━━━━┩
│ conv2d_6 (Conv2D)                     │ (None, 222, 222, 32)         │
896 │
├───────────────────────────────────────┼──────────────────────────────┼
────────────────┤
│ max_pooling2d_6 (MaxPooling2D)        │ (None, 111, 111, 32)         │
0 │
├───────────────────────────────────────┼──────────────────────────────┼
────────────────┤
│ conv2d_7 (Conv2D)                     │ (None, 109, 109, 64)         │
18,496 │
├───────────────────────────────────────┼──────────────────────────────┼
────────────────┤
│ max_pooling2d_7 (MaxPooling2D)        │ (None, 54, 54, 64)           │
0 │
├───────────────────────────────────────┼──────────────────────────────┼
────────────────┤
│ conv2d_8 (Conv2D)                     │ (None, 52, 52, 128)          │
73,856 │
├───────────────────────────────────────┼──────────────────────────────┼
────────────────┤
│ max_pooling2d_8 (MaxPooling2D)        │ (None, 26, 26, 128)          │
0 │
├───────────────────────────────────────┼──────────────────────────────┼
────────────────┤
│ flatten_2 (Flatten)                   │ (None, 86528)                │
0 │
├───────────────────────────────────────┼──────────────────────────────┼
────────────────┤
│ dense_6 (Dense)                       │ (None, 256)                  │
22,151,424 │
├───────────────────────────────────────┼──────────────────────────────┼
────────────────┤
│ dropout_3 (Dropout)                   │ (None, 256)                  │
0 │
├───────────────────────────────────────┼──────────────────────────────┼
────────────────┤
│ dense_7 (Dense)                       │ (None, 128)                  │
32,896 │
├───────────────────────────────────────┼──────────────────────────────┼
────────────────┤
│ dropout_4 (Dropout)                   │ (None, 128)                  │
0 │
├───────────────────────────────────────┼──────────────────────────────┼
────────────────┤
│ dense_8 (Dense)                       │ (None, 1)                    │
129 │
```

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

 Total params: 22,277,697 (84.98 MB)

 Trainable params: 22,277,697 (84.98 MB)

 Non-trainable params: 0 (0.00 B)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout

def new_cnn_model(input_shape, num_classes): # Add num_classes as an
argument
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu',
input_shape=input_shape),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(256, activation='relu'),
        Dropout(0.5),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')  # Change to softmax
and use num_classes
    ])
    return model

# Assuming you have 15 classes
num_classes = 15

# Define input shape based on your image dimensions (e.g., (height,
width, channels))
input_shape = (224, 224, 3)

# Create the new CNN model
new_model = new_cnn_model(input_shape, num_classes) # Pass num_classes
to the function

# Compile the model
# Use sparse_categorical_crossentropy if your labels are integers
new_model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Print model summary
new_model.summary()
```

```
Model: "sequential_3"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳
━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                 ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇
━━━━━━━━━━━━━━━━━┩
│ conv2d_9 (Conv2D)                    │ (None, 222, 222, 32)         │
896 │
├──────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ max_pooling2d_9 (MaxPooling2D)       │ (None, 111, 111, 32)         │
0 │
├──────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ conv2d_10 (Conv2D)                   │ (None, 109, 109, 64)         │
18,496 │
├──────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ max_pooling2d_10 (MaxPooling2D)      │ (None, 54, 54, 64)           │
0 │
├──────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ conv2d_11 (Conv2D)                   │ (None, 52, 52, 128)          │
73,856 │
├──────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ max_pooling2d_11 (MaxPooling2D)      │ (None, 26, 26, 128)          │
0 │
├──────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ flatten_3 (Flatten)                  │ (None, 86528)                │
0 │
├──────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ dense_9 (Dense)                      │ (None, 256)                  │
22,151,424 │
├──────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ dropout_5 (Dropout)                  │ (None, 256)                  │
0 │
├──────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ dense_10 (Dense)                     │ (None, 128)                  │
32,896 │
├──────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ dropout_6 (Dropout)                  │ (None, 128)                  │
0 │
```

```
├──────────────────────────────────────┼───────────────────────────────┼
───────────────────┤
│ dense_11 (Dense)                     │ (None, 15)                    │
1,935 │
└──────────────────────────────────────┴───────────────────────────────┴
───────────────────┘

 Total params: 22,279,503 (84.99 MB)

 Trainable params: 22,279,503 (84.99 MB)

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

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator


# Define the path to your validation data directory
validation_data_dir =
'/root/.cache/kagglehub/datasets/kushagra3204/wheat-plant-diseases/
versions/6/data/'

# Create a validation ImageDataGenerator (assuming no augmentations
for validation)
validation_datagen = ImageDataGenerator(rescale=1./255)

# Create the validation generator
validation_generator = validation_datagen.flow_from_directory(
    validation_data_dir,
    target_size=(224, 224),
    class_mode='categorical'
)
```

```
Found 14154 images belonging to 3 classes.
```

```
#pre trained resnet50v2 architecture
from tensorflow.keras.applications import ResNet50V2
from tensorflow.keras.layers import Dense, Dropout,
GlobalAveragePooling2D
from tensorflow.keras.models import Sequential

# Define the function to create the ResNet50V2 model
def resnet_model(input_shape):
    # Load pre-trained ResNet50V2 model without the top (fully
connected) layers
    base_model = ResNet50V2(weights='imagenet', include_top=False,
input_shape=input_shape)

    # Freeze all layers of the pre-trained model
    for layer in base_model.layers:
        layer.trainable = False
```

```
    # Create a Sequential model
    model = Sequential(name='ResNet50V2')

    # Add the pre-trained ResNet50V2 model to the Sequential model
    model.add(base_model)

    # Add global average pooling layer to reduce parameters
    model.add(GlobalAveragePooling2D())

    # Add a fully connected layer with fewer neurons
    model.add(Dense(64, activation='relu'))

    # Add dropout layer
    model.add(Dropout(0.5))

    # Add output layer for binary classification
    model.add(Dense(1, activation='sigmoid'))

    return model

# Define input shape based on your image dimensions (e.g., (height,
width, channels))
input_shape = (224, 224, 3)

# Create the ResNet50V2 model
resnet_model_instance = resnet_model(input_shape)

# Compile the model
resnet_model_instance.compile(optimizer='adam',
loss='binary_crossentropy', metrics=['accuracy'])

# Print model summary
resnet_model_instance.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/resnet/
resnet50v2_weights_tf_dim_ordering_tf_kernels_notop.h5
94668760/94668760 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step

Model: "ResNet50V2"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┓
━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┩
━━━━━━━━━━━━━━━━━┩
│ resnet50v2 (Functional)              │ (None, 7, 7, 2048)          │
23,564,800 │
├──────────────────────────────────────┼─────────────────────────────┤
─────────────────┤
```

```
│ global_average_pooling2d                    │ (None, 2048)               │
0 │
│ (GlobalAveragePooling2D)                    │                            │
│
├─────────────────────────────────────────────┼────────────────────────────┼
─────────────────┤
│ dense_12 (Dense)                            │ (None, 64)                 │
131,136 │
├─────────────────────────────────────────────┼────────────────────────────┼
─────────────────┤
│ dropout_7 (Dropout)                         │ (None, 64)                 │
0 │
├─────────────────────────────────────────────┼────────────────────────────┼
─────────────────┤
│ dense_13 (Dense)                            │ (None, 1)                  │
65 │
└─────────────────────────────────────────────┴────────────────────────────┴
─────────────────┘

 Total params: 23,696,001 (90.39 MB)

 Trainable params: 131,201 (512.50 KB)

 Non-trainable params: 23,564,800 (89.89 MB)

def train_and_evaluate_model(model, train_gen, val_gen, test_gen,
epochs=10):
    history = model.fit(
        train_gen,
        validation_data=val_gen,
        epochs=epochs,
        verbose=1
    )
    return history

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define the path to your validation data directory
validation_data_dir =
'/root/.cache/kagglehub/datasets/kushagra3204/wheat-plant-diseases/
versions/6/data/'

# Create a validation ImageDataGenerator (assuming no augmentations
for validation)
validation_datagen = ImageDataGenerator(rescale=1./255)

# Create the validation generator
validation_generator = validation_datagen.flow_from_directory(
    validation_data_dir,
    target_size=(224, 224),
```

```
    class_mode='categorical'
)
```

```
Found 14154 images belonging to 3 classes.
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define image data generator with rescaling and validation split
datagen = ImageDataGenerator(rescale=1.0/255, validation_split=0.2)

# Training generator
train_generator = datagen.flow_from_directory(
    '/root/.cache/kagglehub/datasets/kushagra3204/wheat-plant-
diseases/versions/6/data/',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='training'
)

# Validation generator
val_generator = datagen.flow_from_directory(
    '/root/.cache/kagglehub/datasets/kushagra3204/wheat-plant-
diseases/versions/6/data/',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='validation'
)

# Test generator (if you have a separate test dataset)
test_datagen = ImageDataGenerator(rescale=1.0/255)
test_generator = test_datagen.flow_from_directory(
    '/root/.cache/kagglehub/datasets/kushagra3204/wheat-plant-
diseases/versions/6/data/',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)
```

```
Found 11324 images belonging to 3 classes.
Found 2830 images belonging to 3 classes.
Found 14154 images belonging to 3 classes.
```

```
def train_and_evaluate_model(model, train_gen, val_gen, test_gen,
epochs=10):
    # Ensure the model is compiled before fitting
    model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

    history = model.fit(
```

```
        train_gen,
        validation_data=val_gen,
        epochs=epochs,
        verbose=1
    )

    # Explicitly return the history object
    return history

#train model
history = train_and_evaluate_model(resnet_model_instance,
train_generator, validation_generator, test_generator, epochs=10)
```

```
Epoch 1/10
354/354 ━━━━━━━━━━━━━━━━━━━━ 0s 5s/step - accuracy: 0.1749 - loss:
1.2544

/usr/local/lib/python3.10/dist-packages/keras/src/trainers/
data_adapters/py_dataset_adapter.py:122: UserWarning: Your `PyDataset`
class should call `super().__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()

 354/354 ━━━━━━━━━━━━━━━━━━━━ 4068s 11s/step - accuracy: 0.1758 -
loss: 1.2538 - val_accuracy: 0.9258 - val_loss: 0.6903
Epoch 2/10
354/354 ━━━━━━━━━━━━━━━━━━━━ 0s 5s/step - accuracy: 0.9281 - loss:
0.5918
```

```
#pre trained resnet50v2 architecture
from tensorflow.keras.applications import ResNet50V2
from tensorflow.keras.layers import Dense, Dropout,
GlobalAveragePooling2D
from tensorflow.keras.models import Sequential

# Define the function to create the ResNet50V2 model
def resnet_model(input_shape, num_classes): # Add num_classes argument
    # Load pre-trained ResNet50V2 model without the top (fully
connected) layers
    base_model = ResNet50V2(weights='imagenet', include_top=False,
input_shape=input_shape)

    # Freeze all layers of the pre-trained model
    for layer in base_model.layers:
        layer.trainable = False
    # Create a Sequential model
    model = Sequential(name='ResNet50V2')
```

```python
# Add the pre-trained ResNet50V2 model to the Sequential model
model.add(base_model)
```

#Conclusion

Based on the execution time which decreases each time a new model is executed and the accuracy becomes better each time hence boosting performance.

The best model is the ResNET50V2, which has nalidation accuracy of 0.9281 and took the shortest execution time of 1hr 30min 10 secs.

The least was Baseline model and the second improved model was complex architecutre.

Early Stoping model helped in minimizing overfitting hence boosting performance.

#Recomendations

Based on the evaluation results we recommend ResNET50V2 as the primary model of wheat plant diseases.This model has exhibited the best overall performance,striking a balance between high accuracy,precision, and low loss.Its intergration the specific wheat diseases and imaging analysis systems can significantly enhance the speed and accuracy of the diseases,ultimately leading to improved yeald outcomes and more efficient produce