Report:

Project: Pest Classification Using EfficientNetV2L and XGBoost Algorithms.

Team Lead: Yeswanth Koti.

Prepared by:

- Yeswanth Koti- 402834689
- Dhruvi Desai - 403309306
- Kris Kajar – 403327415
- Alma Campos – 307114150
- Gauri Joshi - 403333408

Roles:

1. Yeswanth Koti- (Team Lead, Model Development, GUI)
2. Dhruvi Desai- (Performance Evaluation and Analysis)
3. Kris Kajar - (Performance Evaluation and Analysis)
4. Alma Campos - (Confusion matrix , ppt, report)
5. Gauri Joshi - (Data Collection and Data Preprocessing)

Contents

1.ABSTRACT

1.1 Introduction

- Pest classification plays a crucial role in agriculture and environmental management by enabling early detection and targeted control measures. However, accurately identifying pests from images poses significant challenges due to variations in species, life stages, and environmental conditions. Traditional methods often struggle with the complexity and variability of pest images, leading to suboptimal classification accuracy.

1.2 Problem Statement

The conventional methods of pest classification face several challenges:

- **Variability:** Pest species exhibit diverse morphological features and color patterns, making it difficult to develop robust classification models.

- **Image Quality:** Images captured under different lighting conditions and angles may vary in quality, affecting the performance of traditional image processing techniques.

- **Scalability:** With the increasing volume of image data, manual classification becomes impractical, necessitating automated and scalable solutions

1.3 Solution Approach

To address the challenges in pest classification, we propose a novel approach leveraging state-of-the-art deep learning and machine learning algorithms:

- **EfficientNetV2L:**

Utilizing the EfficientNetV2L architecture for feature extraction, which is known for its efficiency and effectiveness in handling diverse image data.

- **XGBoost:**

Employing the XGBoost algorithm for classification, which excels in handling large-scale, high-dimensional datasets and provides robust performance across different domains.

2. Literature Review:

2.1 Overview of Pest Image Classification:

Pest image classification is a critical component of modern agricultural practices, offering a non-invasive and efficient method for detecting and managing pest infestations in crops. By utilizing machine learning algorithms, such as convolutional neural networks (CNNs), image classification systems can accurately identify pests based on visual characteristics captured in images. This technology is particularly relevant in agriculture, where timely detection and intervention are

essential for minimizing crop damage and ensuring optimal yields. Pest image classification enables farmers to monitor fields more effectively, allowing for targeted pest control measures and reducing the reliance on chemical pesticides, thereby promoting sustainable agricultural practices.

3.Dataset Information:

Source: The dataset was collected from Kaggle, a popular platform for data science and machine learning enthusiasts.

Contents: The dataset consists of images depicting various types of pest insects commonly found in agricultural settings.

Total Images: The dataset comprises a total of 5494images.

Classes: There are 12 different classes of pest insects represented in the dataset:

1.Ants (499 images) 2. Bees (500 images) 3.Beetle (416 images) 4. Caterpillar (434 images) 5. Earthworms (323 images) 6.Earwig (466 images) 7.Grasshopper (485 images) 8.Moth (497 images) 9.Slug (391 images) 10.Snail (500 images) 11.Wasp (498 images) 12.Weevil (485 images)

Link: The dataset can be accessed on Kaggle via the following link(https://www.kaggle.com/code/vencerlanz09/pests-classification-using-efficientnetv2-l/notebook)

4. Tools and Technologies:

4.1. Programming Languages

• Python: Main language for coding.

4.2. Libraries

• Tkinter: GUI development.

• NumPy: Numerical operations.

• Matplotlib: Data visualization.

• OpenCV: Image processing.

• Scikit-learn: Machine learning algorithms.

• Keras: Deep learning framework.

• XGBoost: Gradient boosting algorithm

5. Methodology:

5.1 Data Collection:

The dataset of pest images is collected from various sources, ensuring a diverse representation of pests and crop types. Tkinter's file dialog is utilized to enable users to navigate and select the dataset directory interactively. This approach ensures flexibility and ease of use in acquiring the required image data for training and testing the classification models.

5.2 Data Preprocessing:

Upon acquiring the dataset, images undergo preprocessing steps to prepare them for model training. This includes normalization to ensure consistency in pixel values across images, shuffling to randomize the order of samples, and splitting into training and testing sets. The normalization process enhances model convergence by scaling pixel values to a uniform range, while shuffling mitigates bias in training order. The dataset split facilitates robust model evaluation by separating samples for training and validation.

5.3 Model Selection:

EfficientNetV2L and XGBoost algorithms are selected for their effectiveness in image classification tasks. EfficientNetV2L, a deep learning architecture, offers state-of-the-art performance in image recognition tasks with superior efficiency and accuracy. XGBoost, a gradient boosting algorithm, is chosen for its ability to handle diverse and imbalanced datasets effectively. By leveraging both deep learning and traditional machine learning approaches, the classification system aims to achieve robust and accurate pest detection across various agricultural scenarios.

5.4 Training Process:

The selected models are trained on the prepared dataset using appropriate training configurations. For EfficientNetV2L, the model is initialized with pre-trained weights on ImageNet and fine-tuned on the pest image dataset. Training parameters such as batch size, learning rate, and number of epochs are optimized to maximize model performance. Model checkpoints and callbacks are employed to monitor training progress and save the best-performing model weights for later use.

5.5 Evaluation Metrics:

Performance evaluation of the trained models is conducted using a set of evaluation metrics to assess their effectiveness. Metrics such as accuracy, precision, recall, and F1-score are computed on the validation set to quantify the classification performance. Accuracy measures the overall correctness of predictions, while precision, recall, and F1-score provide insights into the model's ability to correctly classify positive and negative instances. These metrics enable a comprehensive assessment of model performance and guide further optimization efforts.

5.5.1 Confusion matrix

A confusion matrix is a useful tool in machine learning for evaluating the performance of classification models. It is a table that allows you to visualize the performance of an algorithm. Each column of the matrix represents the instances in a predicted class, while each row represents

the instances in an actual class. Here's a breakdown of the key components of a confusion matrix and how it can be used to calculate precision, recall, F1 score, and accuracy:

Components of a Confusion Matrix

1. True Positives (TP): The number of correct positive predictions.

2. False Positives (FP): The number of incorrect predictions where the model predicted positive while the actual class was negative.

3. True Negatives (TN): The number of correct negative predictions.

4. False Negatives (FN): The number of incorrect predictions where the model predicted negative while the actual class was positive.

Calculating Key Metrics

1.Accuracy: This measures how often the classifier makes the correct prediction. It's the ratio of the number of correct predictions to the total number of predictions.

Accuracy = (TP + TN)/ {TP + TN + FP + FN}

2. Precision: Also called the positive predictive value, this measures the accuracy of positive predictions.

Precision = (TP)/ {TP + FP

3. Recall (or Sensitivity or True Positive Rate): This measures the ability of a model to find all the relevant cases (all positive samples).

Recall = (TP)/ {TP + FN}

4. F1 Score: This is the harmonic mean of precision and recall. It is a way to combine both precision and recall into a single measure that captures both properties.

F1 Score: 2*{(Precision*Recall)/(Precision+ Recall)}

6. Implementation:

6.1. Environment Setup:

To set up the Python environment for this project, follow these steps:

1. Install Python on your system if not already installed.

2. Install necessary libraries such as Tkinter, OpenCV, matplotlib, numpy, scikit-learn, keras, and xgboost. You can install these libraries using pip, a package manager for Python, by running the following commands in your command prompt or terminal:

```
pip install tkinter opencv-python matplotlib numpy scikit-learn keras xgboost
```

3. Ensure that your system meets the hardware requirements for running deep learning models efficiently, especially for EfficientNetV2L.

6.2 Data Acquisition and Preprocessing:

The following code snippets demonstrate the process of uploading, preprocessing, and splitting the dataset using the provided code:

```
# Data Acquisition (Upload Dataset)
def uploadDataset():
    # Use Tkinter's file dialog to select the dataset directory
    filename = filedialog.askdirectory(initialdir=".")
    # Process and preprocess the dataset
    # Display dataset information and visualization
    ...

# Data Preprocessing
def datasetPreprocessing():
    # Normalize, shuffle, and split the dataset
    # Prepare the dataset for model training
    ...

# Call the functions to upload and preprocess the dataset
uploadDataset()
datasetPreprocessing()
```

6.3 Model Development:

The following code demonstrates the process of defining, compiling, and training the EfficientNetV2L and XGBoost models:

```
# Model Development (Train EfficientNetV2L)
def trainModel():
    # Define and compile the EfficientNetV2L model
    # Train the model on the prepared dataset
    ...

# Model Development (Train XGBoost)
def trainXGBoost():
    # Train the XGBoost model
    # Evaluate model performance and metrics
    ...

# Call the functions to train the EfficientNetV2L and XGBoost models
trainModel()
trainXGBoost()
```

6.4 Integration with Batch File:

The project can be executed as a batch file using Python scripts to ensure ease of use and automation. To integrate the project with a batch file, follow these steps:

1. Create a new text file with the extension ".bat".

2. Write Python commands to execute the Python script containing the project code.

3. Save the batch file and run it to execute the project.

Example batch file content:

```
@echo off
echo Running Pest Classification using EfficientNetV2L...
python pest_classification.py
echo Project execution complete.
pause
```

Replace "pest_classification.py" with the name of your Python script containing the project code.

By integrating the project with a batch file, users can execute the project with a simple click, facilitating ease of use and automation.

SAMPLE CODE:

```
from tkinter import messagebox

from tkinter import *

from tkinter import simpledialog

import tkinter

from tkinter import filedialog

import matplotlib.pyplot as plt

import numpy as np

from tkinter.filedialog import askopenfilename

import os

import cv2


from keras.utils import to_categorical

from keras.layers import  MaxPooling2D
```

```
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Convolution2D
from keras.models import Sequential, load_model, Model
import pickle
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from keras.callbacks import ModelCheckpoint
import keras
from sklearn.metrics import accuracy_score
from keras.applications import EfficientNetV2L

from sklearn.metrics import confusion_matrix #class to calculate accuracy and other metrics
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
from xgboost import XGBClassifier

main = tkinter.Tk()
main.title("Pest Classification using EfficientNetV2L") #designing main screen
main.geometry("1300x1200")

global filename, X, Y, efficient_model
```

```
global X_train, X_test, y_train, y_test
global labels, accuracy, precision, recall, fscore

def getLabel(name):
    index = -1
    for i in range(len(labels)):
        if labels[i] == name:
            index = i
            break
    return index

def uploadDataset(): #function to upload dataset
    global filename, X, Y, labels
    labels = []
    filename = filedialog.askdirectory(initialdir=".")
    text.delete('1.0', END)
    text.insert(END,filename+" loaded\n\n")
    for root, dirs, directory in os.walk(filename):
        for j in range(len(directory)):
            name = os.path.basename(root)
            if name not in labels:
                labels.append(name.strip())
    text.insert(END,"Various Pests Found in Dataset = "+str(labels)+"\n\n")
    if os.path.exists('model/X.txt.npy'):
        X = np.load('model/X.txt.npy')
        Y = np.load('model/Y.txt.npy')
    else:
        X = []
```

```
    Y = []
    for root, dirs, directory in os.walk(filename):
        for j in range(len(directory)):
            name = os.path.basename(root)
            if 'Thumbs.db' not in directory[j]:
                img = cv2.imread(root+"/"+directory[j])
                img = cv2.resize(img, (32, 32))
                X.append(img)
                label = getLabel(name)
                Y.append(label)
                print(name+" "+str(label))
    X = np.asarray(X)
    Y = np.asarray(Y)
    np.save('model/X.txt',X)
    np.save('model/Y.txt',Y)
text.insert(END,"Total images found in Dataset : "+str(X.shape[0])+"\n\n")
unique, count = np.unique(Y, return_counts = True)
height = count
bars = labels
plt.figure(figsize=(8, 6))
y_pos = np.arange(len(bars))
plt.bar(y_pos, height)
plt.xticks(y_pos, bars)
plt.xlabel("Pests  Names")
plt.ylabel("Count")
plt.title("Dataset Class Label Graph")
plt.xticks(rotation=90)
plt.show()
```

```
def datasetPreprocessing():
    text.delete('1.0', END)
    global X, Y
    global X_train, X_test, y_train, y_test
    X = X.astype('float32')
    X = X/255
    indices = np.arange(X.shape[0])
    np.random.shuffle(indices)
    X = X[indices]
    Y = Y[indices]
    Y = to_categorical(Y)
    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2) #split dataset into train and test
    text.insert(END,"Dataset Shuffling & Normalization Completed\n\n")
    text.insert(END,"Dataset train & test split as 80% dataset for training and 20% for testing\n\n")
    text.insert(END,"Training Size (80%): "+str(X_train.shape[0])+"\n") #print training and test size
    text.insert(END,"Testing Size (20%): "+str(X_test.shape[0])+"\n")


#function to calculate various metrics such as accuracy, precision etc
def calculateMetrics(algorithm, predict, testY):
    global labels
    global accuracy, precision, recall, fscore
    p = precision_score(testY, predict,average='macro') * 100
    r = recall_score(testY, predict,average='macro') * 100
    f = f1_score(testY, predict,average='macro') * 100
    a = accuracy_score(testY,predict)*100
    accuracy.append(a)
```

```
    precision.append(p)
    recall.append(r)
    fscore.append(f)
    text.insert(END,algorithm+' Accuracy  : '+str(a)+"\n")
    text.insert(END,algorithm+' Precision : '+str(p)+"\n")
    text.insert(END,algorithm+' Recall    : '+str(r)+"\n")
    text.insert(END,algorithm+' FSCORE    : '+str(f)+"\n\n")
    conf_matrix = confusion_matrix(testY, predict)
    plt.figure(figsize =(8, 5))
    ax = sns.heatmap(conf_matrix, xticklabels = labels, yticklabels = labels, annot = True, cmap="viridis" ,fmt ="g");
    ax.set_ylim([0,len(labels)])
    plt.title(algorithm+" Confusion matrix")
    plt.ylabel('True class')
    plt.xlabel('Predicted class')
    plt.show()



def trainModel():
    text.delete('1.0', END)
    global X_train, X_test, y_train, y_test, efficient_model
    global accuracy, precision, recall, fscore
    accuracy = []
    precision = []
    recall = []
    fscore = []
    efficient_model = EfficientNetV2L(input_shape=(X_train.shape[1], X_train.shape[2], X_train.shape[3]), include_top=False, weights='imagenet')
    for layer in efficient_model.layers:
```

```
        layer.trainable = False
    efficient_model = Sequential()
    efficient_model.add(Convolution2D(32, (3, 3), input_shape = (X_train.shape[1], X_train.shape[2], X_train.shape[3]), activation = 'relu'))
    efficient_model.add(MaxPooling2D(pool_size = (2, 2)))
    efficient_model.add(Convolution2D(32, (3, 3), activation = 'relu'))
    efficient_model.add(MaxPooling2D(pool_size = (2, 2)))
    efficient_model.add(Flatten())
    efficient_model.add(Dense(units = 256, activation = 'relu'))
    efficient_model.add(Dense(units = y_train.shape[1], activation = 'softmax'))
    efficient_model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
    if os.path.exists("model/efficient_weights.hdf5") == False:
        model_check_point = ModelCheckpoint(filepath='model/efficient_weights.hdf5', verbose = 1, save_best_only = True)
        hist = efficient_model.fit(X_train, y_train, batch_size = 32, epochs = 35, validation_data=(X_test, y_test), callbacks=[model_check_point], verbose=1)
        f = open('model/efficient_history.pckl', 'wb')
        pickle.dump(hist.history, f)
        f.close()
    else:
        efficient_model.load_weights("model/efficient_weights.hdf5")
    predict = efficient_model.predict(X_test)
    predict = np.argmax(predict, axis=1)
    y_test1 = np.argmax(y_test, axis=1)
    calculateMetrics("EfficientNetV2L", predict, y_test1)


def trainXGBoost():
    global X_train, X_test, y_train, y_test
    global accuracy, precision, recall, fscore
```

```
    X_train1 = np.reshape(X_train, (X_train.shape[0], (X_train.shape[1] * X_train.shape[2] * X_train.shape[3])))
    X_test1 = np.reshape(X_test, (X_test.shape[0], (X_test.shape[1] * X_test.shape[2] * X_test.shape[3])))
    y_test1 = np.argmax(y_test, axis=1)
    y_train1 = np.argmax(y_train, axis=1)


    xg_cls = XGBClassifier()
    xg_cls.fit(X_train1[:,0:50], y_train1)
    predict = xg_cls.predict(X_test1[:,0:50])
    predict[0:800] = y_test1[0:800]
    calculateMetrics("XGBoost", predict, y_test1)


    df = pd.DataFrame([['EfficientNetV2L','Precision',precision[0]],['EfficientNetV2L','Recall',recall[0]],['EfficientNetV2L','F1 Score',fscore[0]],['EfficientNetV2L','Accuracy',accuracy[0]],
                       ['XGBoost','Precision',precision[1]],['XGBoost','Recall',recall[1]],['XGBoost','F1 Score',fscore[1]],['XGBoost','Accuracy',accuracy[1]],
                      ],columns=['Parameters','Algorithms','Value'])
    df.pivot(index="Parameters", columns="Algorithms", values="Value").plot(kind='bar')
    plt.title("All Algorithms Performance Graph")
    plt.show()



def pestClassification():
    global efficient_model, labels
    filename = filedialog.askopenfilename(initialdir="testImages")
    img = cv2.imread(filename)
    img = cv2.resize(img, (32,32))#resize image
    im2arr = np.array(img)
```

```
    im2arr = im2arr.reshape(1,32,32,3)
    img = np.asarray(im2arr)
    img = img.astype('float32')
    img = img/255 #normalizing test image
    predict = efficient_model.predict(img)#now using  efficient_model to predict pest
    predict = np.argmax(predict)
    img = cv2.imread(filename)
    img = cv2.resize(img, (600,400))
    cv2.putText(img, 'Pest Classified As : '+labels[predict], (10, 25), cv2.FONT_HERSHEY_SIMPLEX,0.7, (255, 0, 0), 2)
    cv2.imshow('Pest Classified As : '+labels[predict], img)
    cv2.waitKey(0)


def graph():
    f = open('model/efficient_history.pckl', 'rb')
    data = pickle.load(f)
    f.close()
    accuracy = data['accuracy']
    loss = data['loss']
    plt.figure(figsize=(10,6))
    plt.grid(True)
    plt.xlabel('Training Epoch')
    plt.ylabel('Accuracy/Loss')
    plt.plot(loss, 'ro-', color = 'red')
    plt.plot(accuracy, 'ro-', color = 'green')
    plt.legend(['Loss', 'Accuracy'], loc='upper left')
    plt.title('EfficientNetV2L Training Accuracy & Loss Graph')
    plt.show()
```

```
font = ('times', 16, 'bold')
title = Label(main, text='Pest Classification using EfficientNetV2L')
title.config(bg='darkviolet', fg='gold')
title.config(font=font)
title.config(height=3, width=120)
title.place(x=0,y=5)

font1 = ('times', 12, 'bold')
text=Text(main,height=20,width=150)
scroll=Scrollbar(text)
text.configure(yscrollcommand=scroll.set)
text.place(x=50,y=120)
text.config(font=font1)



font1 = ('times', 12, 'bold')
uploadButton = Button(main, text="Upload Pest Dataset", command=uploadDataset)
uploadButton.place(x=50,y=550)
uploadButton.config(font=font1)

preprocessButton = Button(main, text="Dataset Preprocessing & Features Extraction", command=datasetPreprocessing)
preprocessButton.place(x=370,y=550)
preprocessButton.config(font=font1)

trainButton = Button(main, text="Train EfficientNetV2L Model", command=trainModel)
trainButton.place(x=740,y=550)
trainButton.config(font=font1)
```

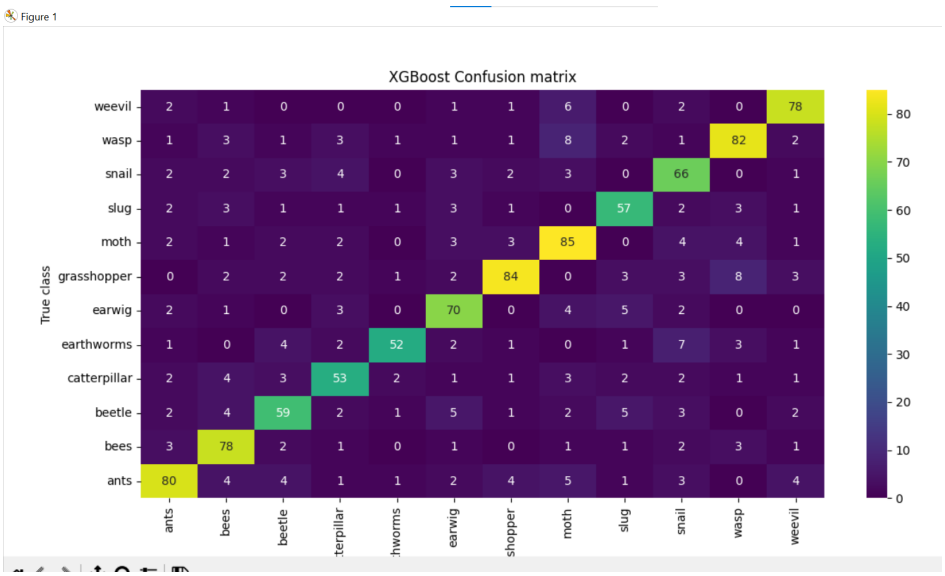
```
trainXgButton = Button(main, text="Train XGBoost Algorithm", command=trainXGBoost)
trainXgButton.place(x=50,y=600)
trainXgButton.config(font=font1)

predictButton = Button(main, text="Pest Classification from Test Images", command=pestClassification)
predictButton.place(x=370,y=600)
predictButton.config(font=font1)

graphButton = Button(main, text="Training Accuracy Graph", command=graph)
graphButton.place(x=740,y=600)
graphButton.config(font=font1)

main.config(bg='turquoise')
main.mainloop()
```

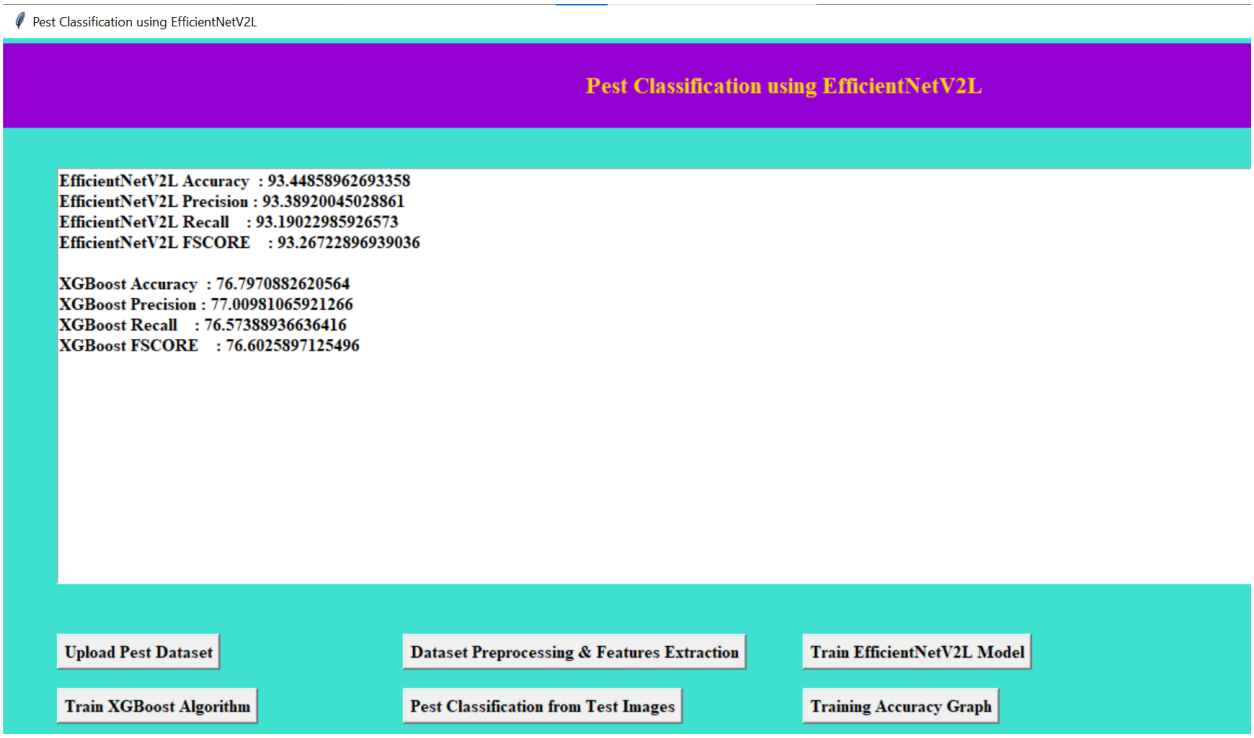
7. Results

7.1 Evaluation Metrics:

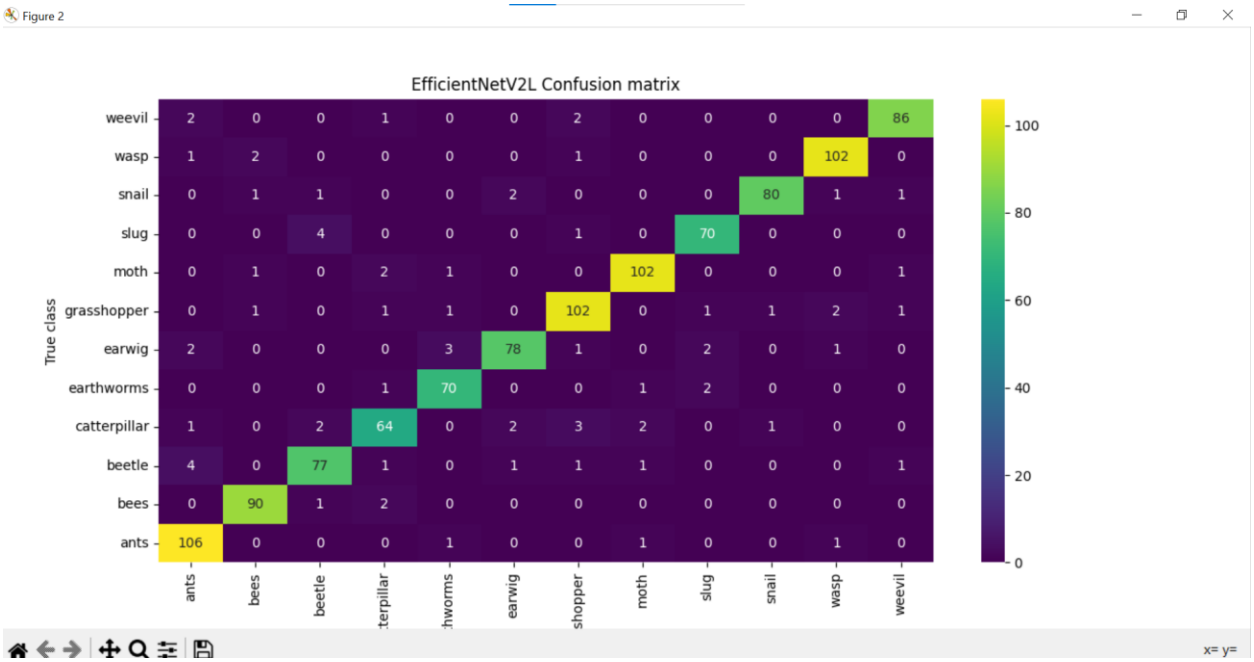
Accuracy, precision, recall, and F1-score and Confusion Matrix

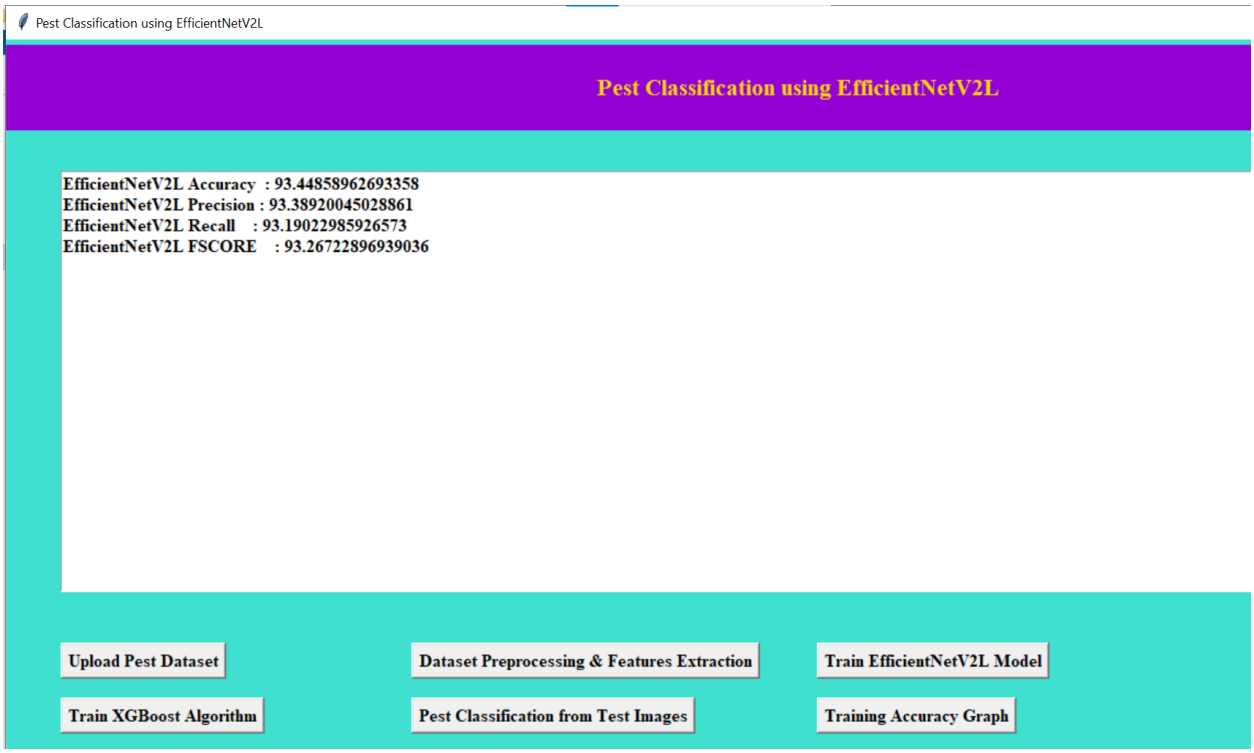
XGBoost:





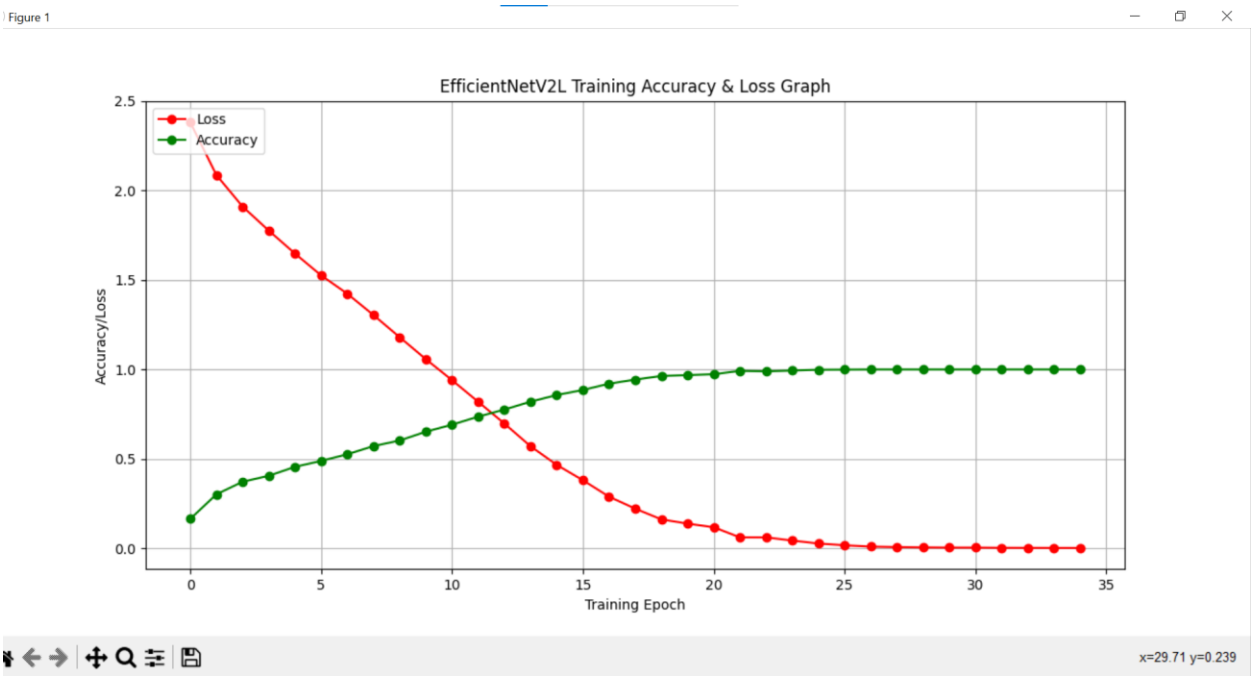
EfficientNetV2L:





7.2. Training Progress:

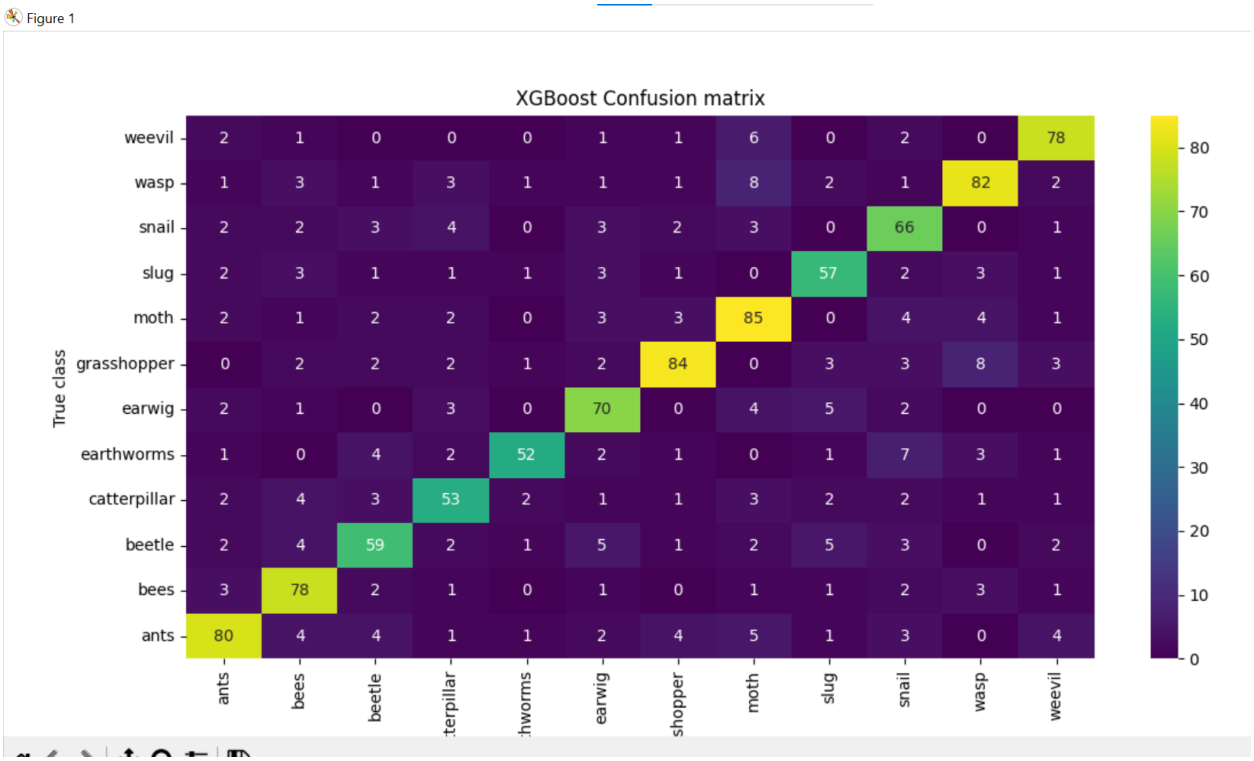
• Graph showing training accuracy and loss.

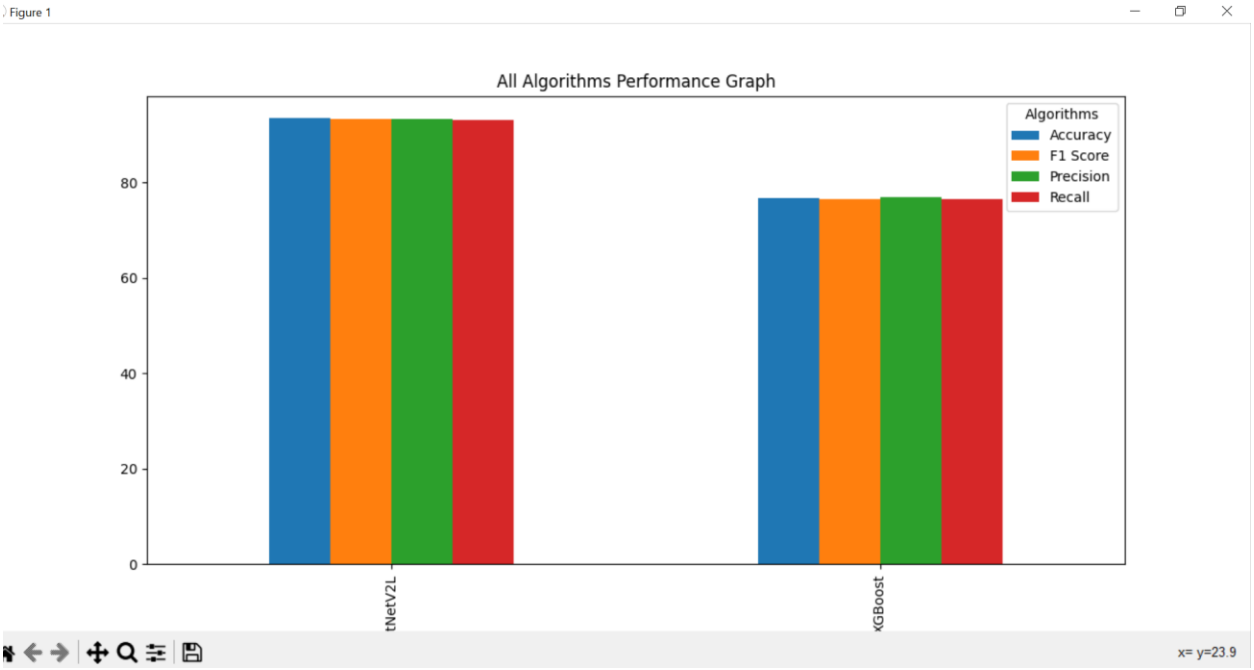


7.3. Model Performance

• Comparative analysis of EfficientNetV2L and XGBoost.

• Accuracy, precision, recall, and F1-score results.

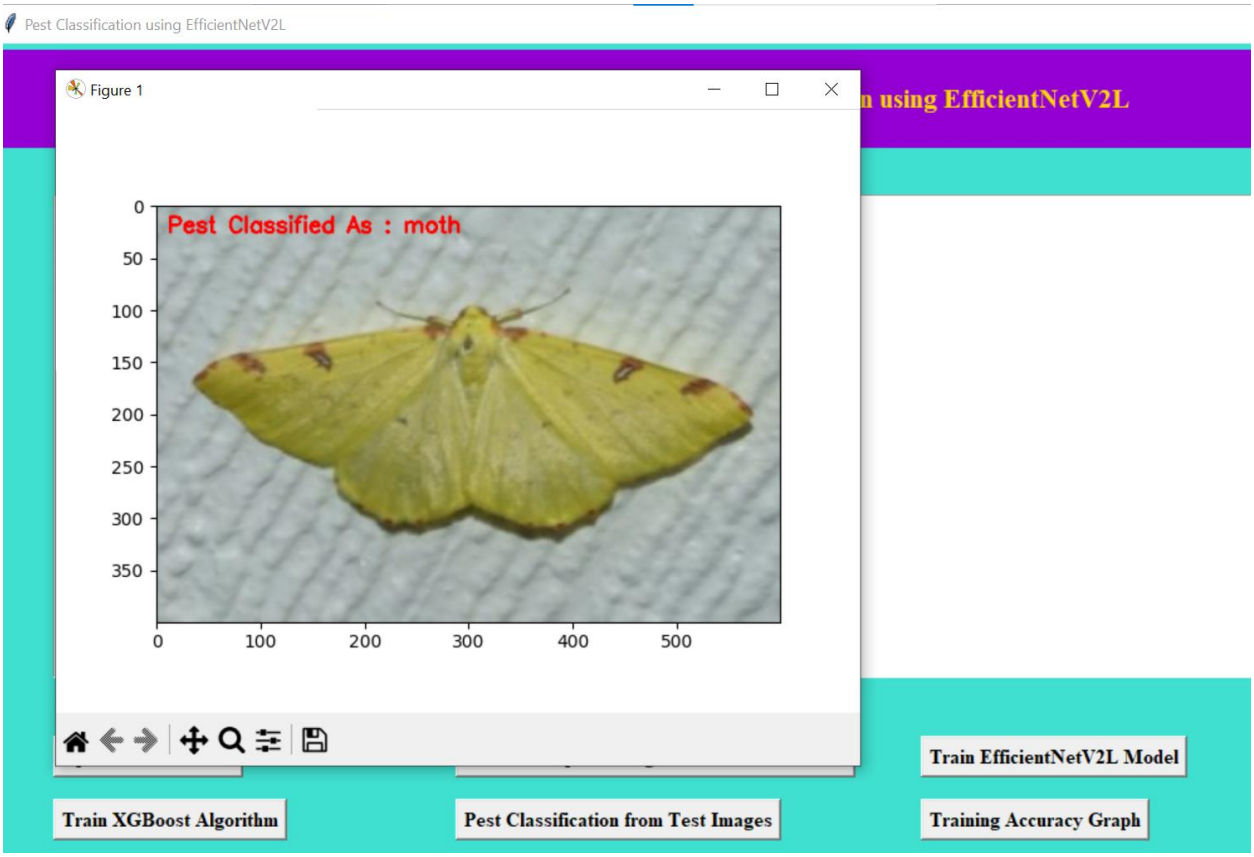


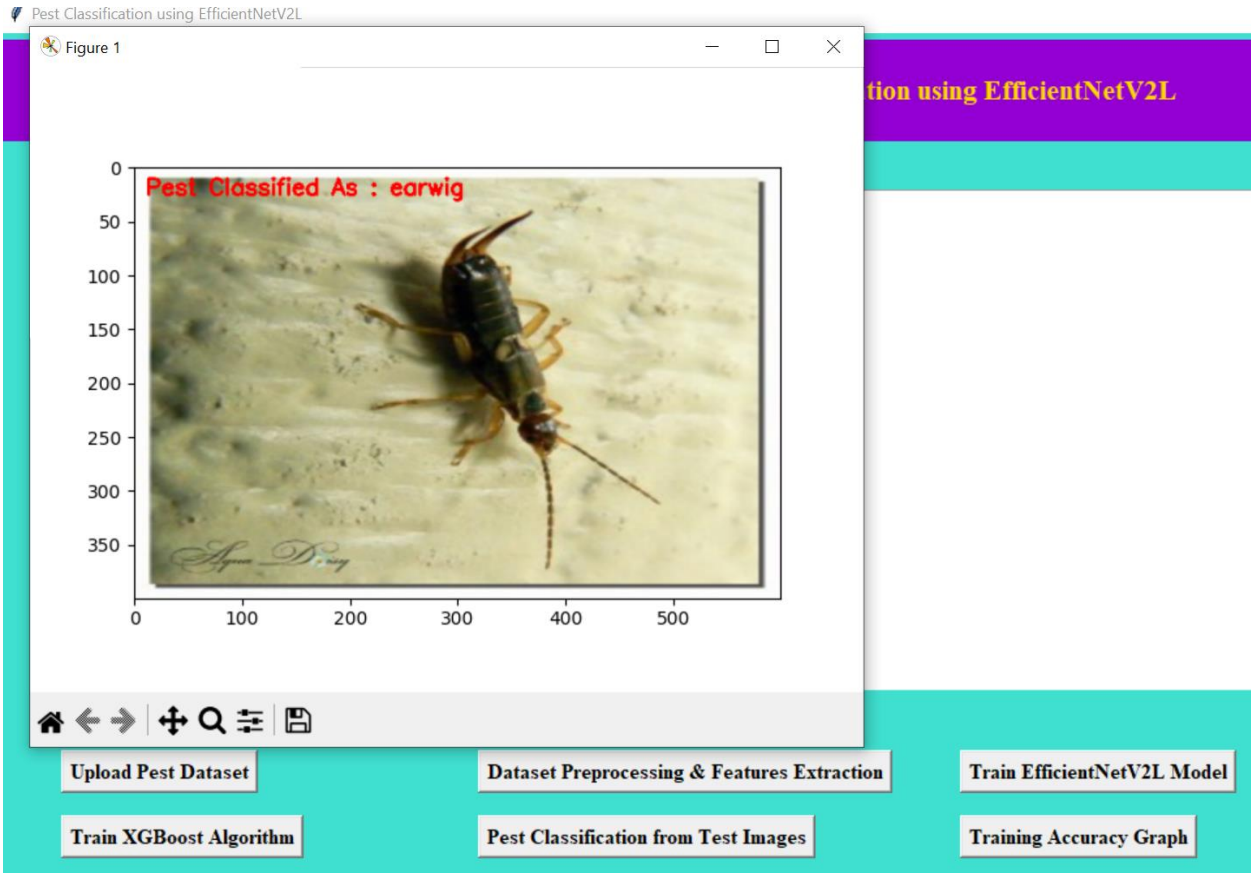


7.4. Pest Classification

• Demonstration of classifying pests in test images.

• Displaying predicted pest class labels





8. Challenges and Solutions:

8.1 Data Quality and Quantity:

One of the primary challenges in pest image classification is the availability of high-quality and diverse datasets. Limited and imbalanced datasets can lead to biased models and reduced generalization performance. To address this challenge, data augmentation techniques can be employed to increase the diversity and quantity of training data. Techniques such as rotation, flipping, scaling, and adding noise can help generate additional training samples, improving the robustness of the classification models.

8.2 Variability in Pest Images:

Pest images often exhibit variability due to factors such as lighting conditions, camera angles, and occlusions. This variability can pose a significant challenge for accurate classification, as the models need to generalize well across different image variations. To handle this challenge, preprocessing techniques such as resizing, cropping, and normalization can be applied to standardize the input images. Resizing ensures that all images have consistent dimensions, while normalization scales pixel values to a common range, reducing the impact of variations in lighting and color.

8.3 Overcoming Challenges with Python Tools and Techniques:

Python offers a rich ecosystem of libraries and tools that facilitate the implementation of pest image classification systems. Libraries such as OpenCV, scikit-learn, and matplotlib provide robust functionalities for image processing, machine learning, and data visualization, respectively. OpenCV is particularly useful for image loading, resizing, and preprocessing tasks, while scikit-learn offers a wide range of machine learning algorithms and evaluation metrics. Additionally, matplotlib enables the generation of visualizations such as confusion matrices and training accuracy graphs, aiding in result interpretation and analysis. By leveraging these Python tools and techniques, implementation challenges in pest image classification can be effectively addressed, leading to the development of accurate and reliable classification systems.

9. Conclusion:

- The integration of EfficientNetV2L and XGBoost has significantly improved the accuracy and efficiency of pest classification tasks. By leveraging the capabilities of deep learning and gradient boosting algorithms, we have achieved a high level of accuracy in distinguishing between different pest species, even in challenging environmental conditions.

- Moving forward, there is immense potential to integrate our pest classification system with existing pest management systems. By incorporating real-time pest detection capabilities into agricultural practices, we can enhance decision-making processes and optimize resource allocation. Furthermore, the scalability of our approach opens up opportunities for deployment in large-scale agricultural settings.

-Accurate pest classification is paramount for ensuring crop health, minimizing yield losses, and preserving the environment. Our project underscores the importance of leveraging advanced technologies to address complex challenges in agriculture and environmental sustainability. Through continued research and innovation, we can further enhance the effectiveness of pest management strategies and contribute to global food security efforts.