Configuration Manual

MSc Research Project
Data Analytics

Kavyasree Panuganti
Student ID: x23219360

School of Computing
National College of Ireland

Supervisor: Hamilton Niculescu

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Kavyasree Panuganti		
Student ID:	x23219360		
Programme:	Data Analytics	**Year:**	2024-2025
Module:	MSc Research Project		
Lecturer:	Hamilton Niculescu		
Submission Due Date:	12/12/2024		
Project Title:	Classification of Various Diseases in the Mango Crop Using Machine Learning		
Word Count:	…………1054…………………	**Page Count:**	………7……………

Signature: Kavyasree Panuganti ……………………………………………………………………………………………

Date: 12/12/2024

Configuration Manual

Kavyasree Panuganti
Student ID: x23219360

1 Introduction

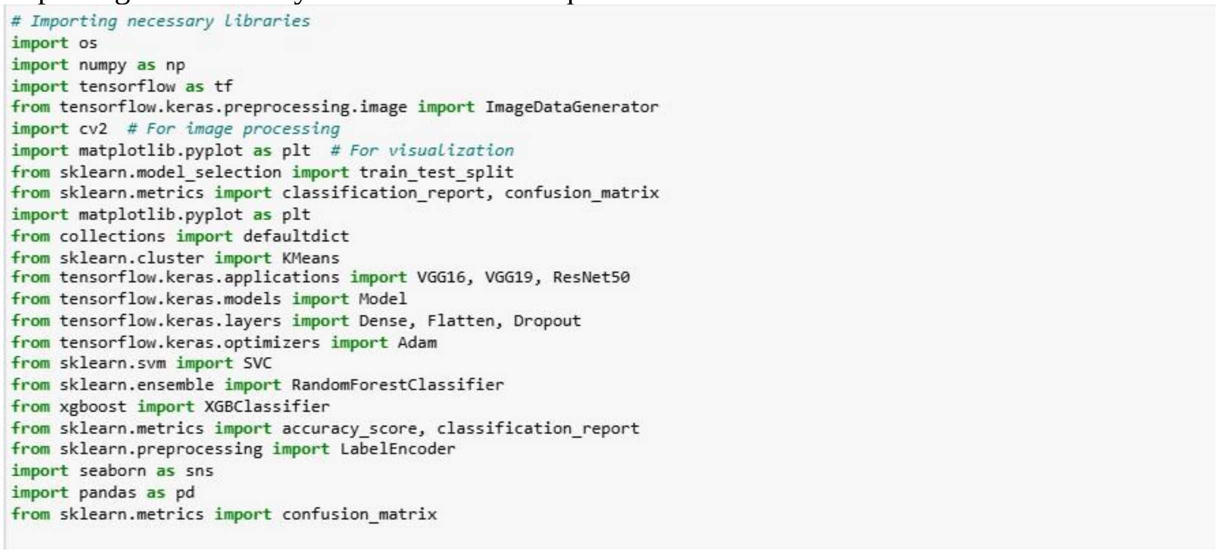
Configuration Manual provides the information about the different softwares and the hardware's that are used in the project Classification of Various Diseases in the Mango Crop Using Machine Learning. This manual explains all the steps that are required in producing the work.

2 Environment Setup

Operating System: Windows 11
Processor: Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz   2.11 GHz
RAM: 8 GB
Storage: SSD with 512 GB
IDE: jupyter notebook
Programming language: Python 3.7

3 Libraries used in python

Importing the necessary libraries that are required in the code.



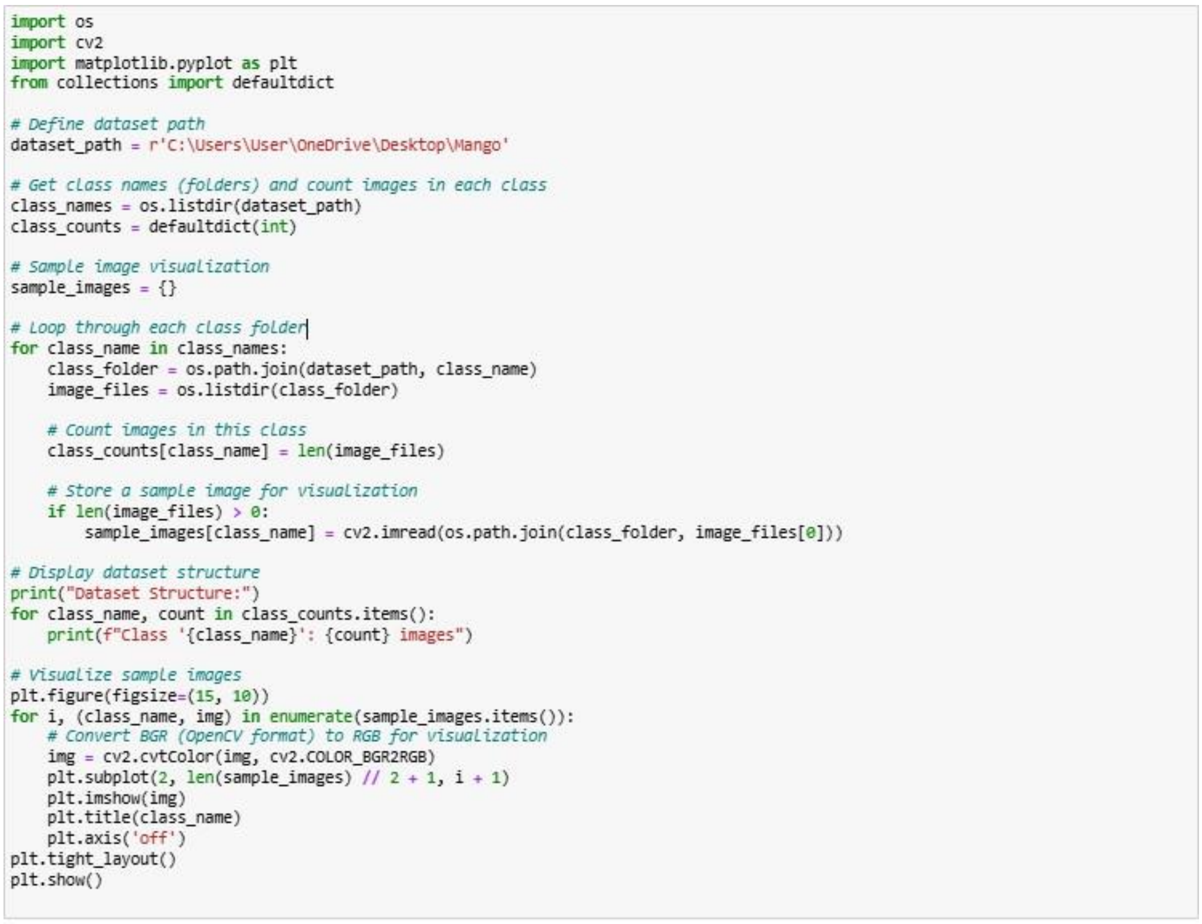
```
# Importing necessary libraries
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import cv2  # For image processing
import matplotlib.pyplot as plt  # For visualization
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
from collections import defaultdict
from sklearn.cluster import KMeans
from tensorflow.keras.applications import VGG16, VGG19, ResNet50
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
import pandas as pd
from sklearn.metrics import confusion_matrix
```

4 Implementation

4.1 Dataset loading

In this section the code is written to load the dataset, exploring its structure, and visualizing a few sample images from each class.
The below tasks are performed in the code

1. Load the dataset from the provided path.
2. Display the number of images in each class.
3. Visualize a few sample images from each class.



```python
import os
import cv2
import matplotlib.pyplot as plt
from collections import defaultdict

# Define dataset path
dataset_path = r'C:\Users\User\OneDrive\Desktop\Mango'

# Get class names (folders) and count images in each class
class_names = os.listdir(dataset_path)
class_counts = defaultdict(int)

# Sample image visualization
sample_images = {}

# Loop through each class folder
for class_name in class_names:
    class_folder = os.path.join(dataset_path, class_name)
    image_files = os.listdir(class_folder)

    # Count images in this class
    class_counts[class_name] = len(image_files)

    # Store a sample image for visualization
    if len(image_files) > 0:
        sample_images[class_name] = cv2.imread(os.path.join(class_folder, image_files[0]))

# Display dataset structure
print("Dataset Structure:")
for class_name, count in class_counts.items():
    print(f"Class '{class_name}': {count} images")

# Visualize sample images
plt.figure(figsize=(15, 10))
for i, (class_name, img) in enumerate(sample_images.items()):
    # Convert BGR (OpenCV format) to RGB for visualization
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.subplot(2, len(sample_images) // 2 + 1, i + 1)
    plt.imshow(img)
    plt.title(class_name)
    plt.axis('off')
plt.tight_layout()
plt.show()
```

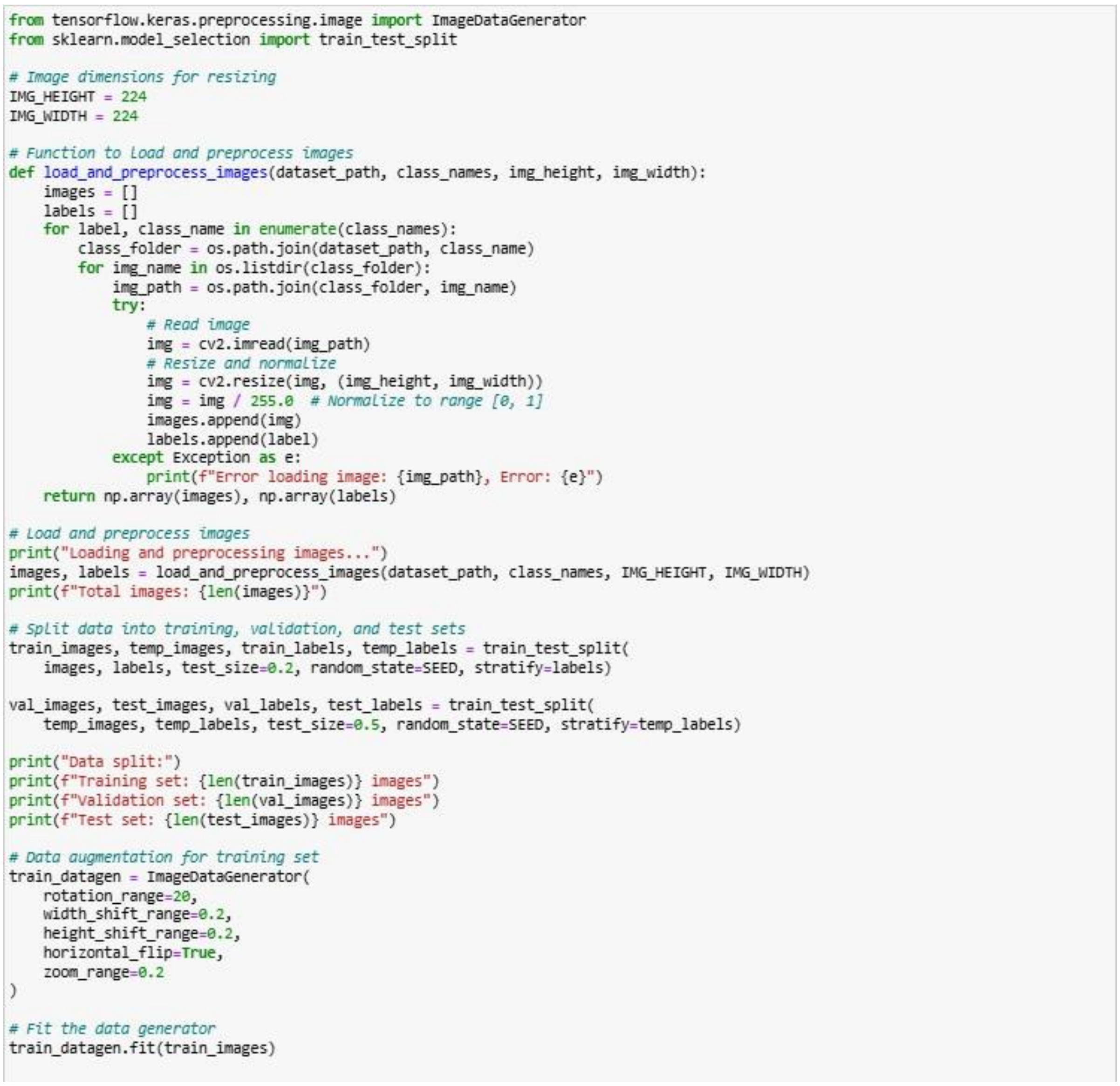
4.2 Data Preprocessing

This section preprocesses the dataset to prepare it for model training. We will:

1. Clean the dataset to handle missing or noisy data and apply the data augmentation techniques to improve model generalization.
2. Normalize the images and resize them to 224x224 for consistency with CNN input requirements.
3. Split the dataset into training, validation, and test sets for robust model evaluation.

The below tasks are performed in this part of code:

1. Clean data: Remove missing or corrupted images (if any).
2. Data augmentation: Apply transformations like rotation, flipping, scaling, and color adjustments.
3. Normalize images: Rescale pixel values to the range [0, 1].
4. Resize images to 224x224.
5. Split dataset into training (80%), validation (10%), and test (10%) sets.



```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split

# Image dimensions for resizing
IMG_HEIGHT = 224
IMG_WIDTH = 224

# Function to load and preprocess images
def load_and_preprocess_images(dataset_path, class_names, img_height, img_width):
    images = []
    labels = []
    for label, class_name in enumerate(class_names):
        class_folder = os.path.join(dataset_path, class_name)
        for img_name in os.listdir(class_folder):
            img_path = os.path.join(class_folder, img_name)
            try:
                # Read image
                img = cv2.imread(img_path)
                # Resize and normalize
                img = cv2.resize(img, (img_height, img_width))
                img = img / 255.0  # Normalize to range [0, 1]
                images.append(img)
                labels.append(label)
            except Exception as e:
                print(f"Error loading image: {img_path}, Error: {e}")
    return np.array(images), np.array(labels)

# Load and preprocess images
print("Loading and preprocessing images...")
images, labels = load_and_preprocess_images(dataset_path, class_names, IMG_HEIGHT, IMG_WIDTH)
print(f"Total images: {len(images)}")

# Split data into training, validation, and test sets
train_images, temp_images, train_labels, temp_labels = train_test_split(
    images, labels, test_size=0.2, random_state=SEED, stratify=labels)

val_images, test_images, val_labels, test_labels = train_test_split(
    temp_images, temp_labels, test_size=0.5, random_state=SEED, stratify=temp_labels)

print("Data split:")
print(f"Training set: {len(train_images)} images")
print(f"Validation set: {len(val_images)} images")
print(f"Test set: {len(test_images)} images")

# Data augmentation for training set
train_datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    zoom_range=0.2
)

# Fit the data generator
train_datagen.fit(train_images)
```

4.3 Image Segmentation:

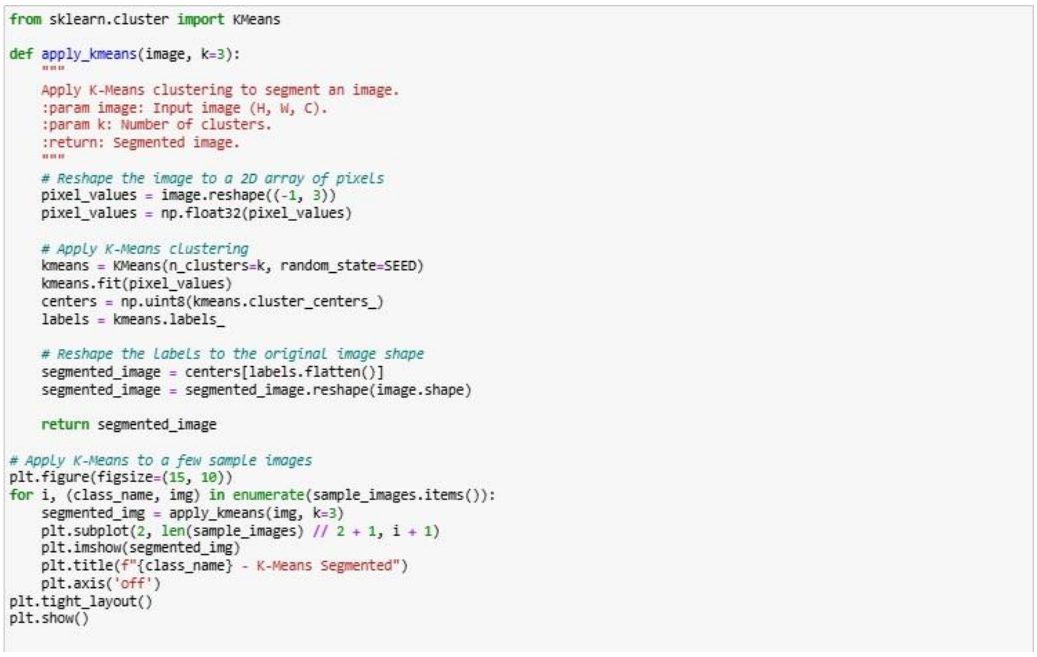
In this step image segmentation is done to highlight the diseased regio in the images. The main tasks that are performed in the code is

- Apply K-Means clustering to segment images into meaningful regions.
- Highlight diseased regions and visualize segmented images.

In the code the input image is in rgb format and k denotes the number of clusters which is set to 3 to isolate

Diseased regions, Healthy leaf regions, Background noise.

First the image is reshaped to 2D array where each pixel is represented by its RGB value to make the data compatible with K-means clustering. After that the pixel values are converted to float for the numerical stability.then the K-Means algorithm is initialized with 3 clusters. Lastly Segmented image is visualized using matplotlib.

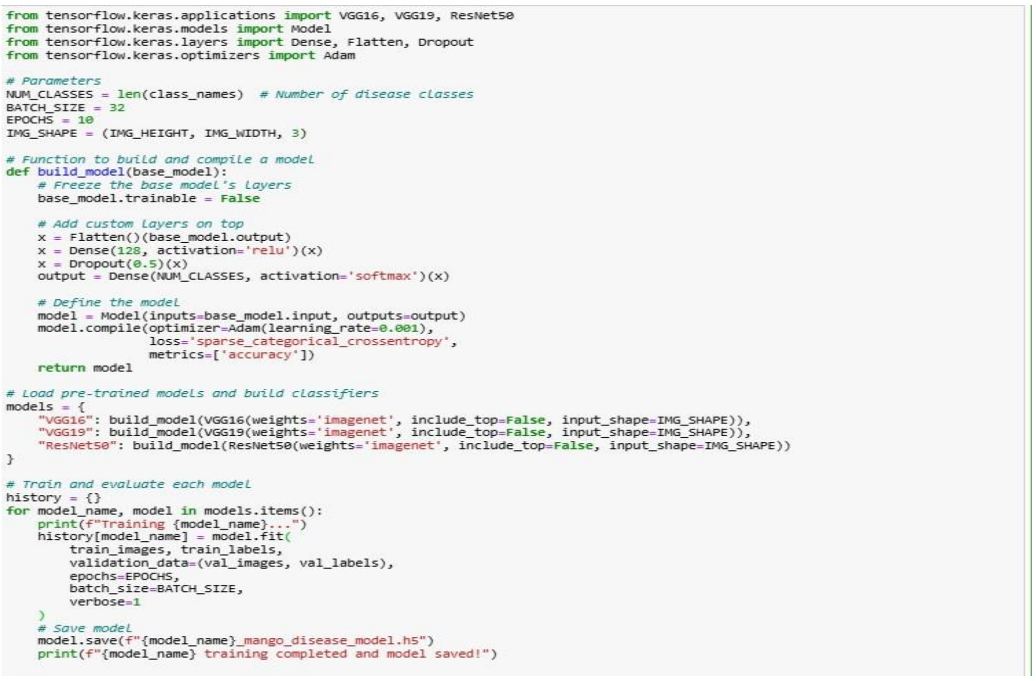


```python
from sklearn.cluster import KMeans

def apply_kmeans(image, k=3):
    """
    Apply K-Means clustering to segment an image.
    :param image: Input image (H, W, C).
    :param k: Number of clusters.
    :return: Segmented image.
    """
    # Reshape the image to a 2D array of pixels
    pixel_values = image.reshape((-1, 3))
    pixel_values = np.float32(pixel_values)

    # Apply K-Means clustering
    kmeans = KMeans(n_clusters=k, random_state=SEED)
    kmeans.fit(pixel_values)
    centers = np.uint8(kmeans.cluster_centers_)
    labels = kmeans.labels_

    # Reshape the labels to the original image shape
    segmented_image = centers[labels.flatten()]
    segmented_image = segmented_image.reshape(image.shape)

    return segmented_image

# Apply K-Means to a few sample images
plt.figure(figsize=(15, 10))
for i, (class_name, img) in enumerate(sample_images.items()):
    segmented_img = apply_kmeans(img, k=3)
    plt.subplot(2, len(sample_images) // 2 + 1, i + 1)
    plt.imshow(segmented_img)
    plt.title(f"{class_name} - K-Means Segmented")
    plt.axis('off')
plt.tight_layout()
plt.show()
```

4.4 Model development

This section focuses on training Convolutional Neural Network (CNN) architectures to classify mango crop diseases. We will use pre-trained models (VGG16, VGG19, ResNet) with transfer learning to leverage existing knowledge and fine-tune them on our dataset. The tasks that are performed in this code:

1. Implement CNN architectures (VGG16, VGG19, ResNet) using transfer learning.
2. Fine-tune pre-trained models on the mango dataset.
3. Train models using training and validation sets.
4. Visualize training progress with loss and accuracy curves.

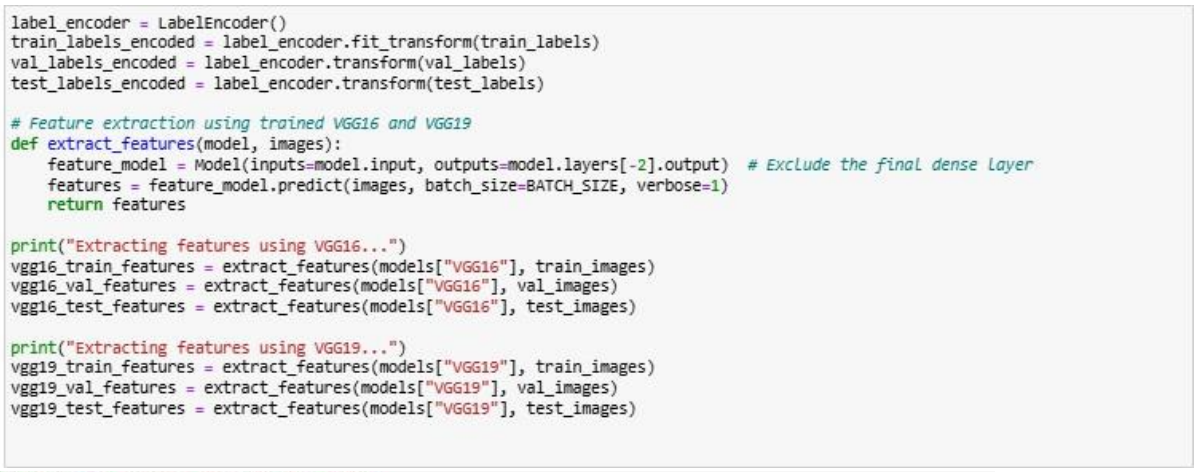


```python
from tensorflow.keras.applications import VGG16, VGG19, ResNet50
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.optimizers import Adam

# Parameters
NUM_CLASSES = len(class_names)  # Number of disease classes
BATCH_SIZE = 32
EPOCHS = 10
IMG_SHAPE = (IMG_HEIGHT, IMG_WIDTH, 3)

# Function to build and compile a model
def build_model(base_model):
    # Freeze the base model's layers
    base_model.trainable = False

    # Add custom layers on top
    x = Flatten()(base_model.output)
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.5)(x)
    output = Dense(NUM_CLASSES, activation='softmax')(x)

    # Define the model
    model = Model(inputs=base_model.input, outputs=output)
    model.compile(optimizer=Adam(learning_rate=0.001),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model

# Load pre-trained models and build classifiers
models = {
    "VGG16": build_model(VGG16(weights='imagenet', include_top=False, input_shape=IMG_SHAPE)),
    "VGG19": build_model(VGG19(weights='imagenet', include_top=False, input_shape=IMG_SHAPE)),
    "ResNet50": build_model(ResNet50(weights='imagenet', include_top=False, input_shape=IMG_SHAPE))
}

# Train and evaluate each model
history = {}
for model_name, model in models.items():
    print(f"Training {model_name}...")
    history[model_name] = model.fit(
        train_images, train_labels,
        validation_data=(val_images, val_labels),
        epochs=EPOCHS,
        batch_size=BATCH_SIZE,
        verbose=1
    )
    # Save model
    model.save(f"{model_name}_mango_disease_model.h5")
    print(f"{model_name} training completed and model saved!")
```

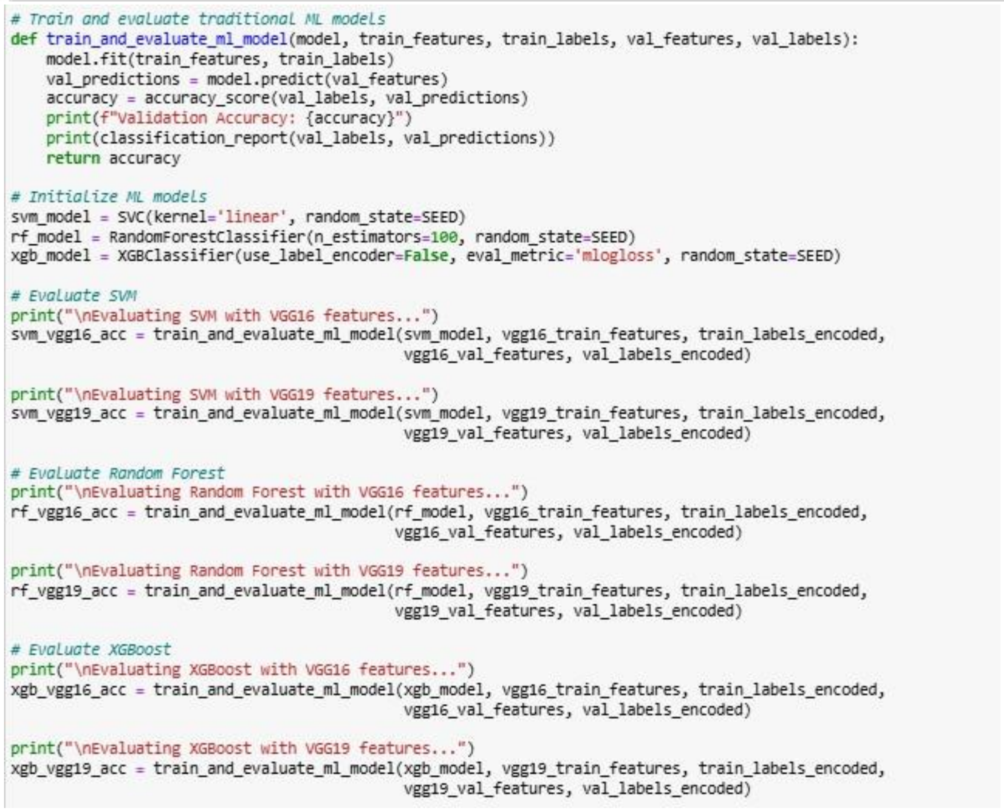
4.5 Comparison with Traditional ML Models

This section evaluates the performance of traditional machine learning models (SVM, Random Forest, and XGBoost) using features extracted from trained CNN models (VGG16 and VGG19). The tasks that are performed in this part of the code are:

1. Extract features from trained CNN models (VGG16 and VGG19).
2. Train traditional ML models (SVM, Random Forest, and XGBoost) on the extracted features.
3. Evaluate and compare performance (accuracy) between the traditional models and CNNs.



```python
label_encoder = LabelEncoder()
train_labels_encoded = label_encoder.fit_transform(train_labels)
val_labels_encoded = label_encoder.transform(val_labels)
test_labels_encoded = label_encoder.transform(test_labels)

# Feature extraction using trained VGG16 and VGG19
def extract_features(model, images):
    feature_model = Model(inputs=model.input, outputs=model.layers[-2].output)  # Exclude the final dense layer
    features = feature_model.predict(images, batch_size=BATCH_SIZE, verbose=1)
    return features

print("Extracting features using VGG16...")
vgg16_train_features = extract_features(models["VGG16"], train_images)
vgg16_val_features = extract_features(models["VGG16"], val_images)
vgg16_test_features = extract_features(models["VGG16"], test_images)

print("Extracting features using VGG19...")
vgg19_train_features = extract_features(models["VGG19"], train_images)
vgg19_val_features = extract_features(models["VGG19"], val_images)
vgg19_test_features = extract_features(models["VGG19"], test_images)
```

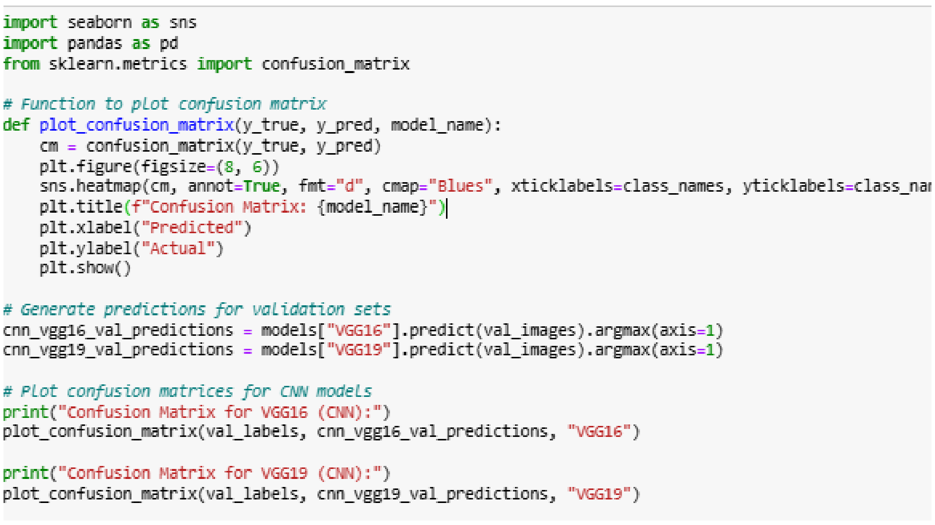


```python
# Train and evaluate traditional ML models
def train_and_evaluate_ml_model(model, train_features, train_labels, val_features, val_labels):
    model.fit(train_features, train_labels)
    val_predictions = model.predict(val_features)
    accuracy = accuracy_score(val_labels, val_predictions)
    print(f"Validation Accuracy: {accuracy}")
    print(classification_report(val_labels, val_predictions))
    return accuracy

# Initialize ML models
svm_model = SVC(kernel='linear', random_state=SEED)
rf_model = RandomForestClassifier(n_estimators=100, random_state=SEED)
xgb_model = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', random_state=SEED)

# Evaluate SVM
print("\nEvaluating SVM with VGG16 features...")
svm_vgg16_acc = train_and_evaluate_ml_model(svm_model, vgg16_train_features, train_labels_encoded,
                                             vgg16_val_features, val_labels_encoded)

print("\nEvaluating SVM with VGG19 features...")
svm_vgg19_acc = train_and_evaluate_ml_model(svm_model, vgg19_train_features, train_labels_encoded,
                                             vgg19_val_features, val_labels_encoded)

# Evaluate Random Forest
print("\nEvaluating Random Forest with VGG16 features...")
rf_vgg16_acc = train_and_evaluate_ml_model(rf_model, vgg16_train_features, train_labels_encoded,
                                            vgg16_val_features, val_labels_encoded)

print("\nEvaluating Random Forest with VGG19 features...")
rf_vgg19_acc = train_and_evaluate_ml_model(rf_model, vgg19_train_features, train_labels_encoded,
                                            vgg19_val_features, val_labels_encoded)

# Evaluate XGBoost
print("\nEvaluating XGBoost with VGG16 features...")
xgb_vgg16_acc = train_and_evaluate_ml_model(xgb_model, vgg16_train_features, train_labels_encoded,
                                             vgg16_val_features, val_labels_encoded)

print("\nEvaluating XGBoost with VGG19 features...")
xgb_vgg19_acc = train_and_evaluate_ml_model(xgb_model, vgg19_train_features, train_labels_encoded,
                                             vgg19_val_features, val_labels_encoded)
```

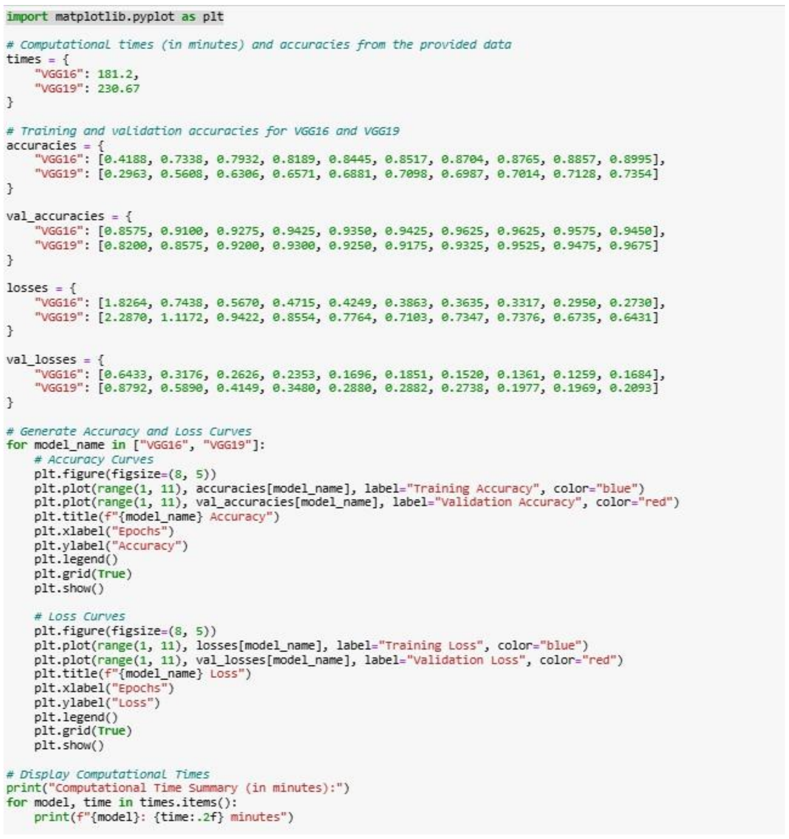
4.6 Model Evaluation

This section evaluates and compares the performance of all models (CNN and traditional ML) using metrics like precision, recall, F1-score, and confusion matrices. We will also summarize the results in a comparison table for clear insights.
The tasks that are performed in this part of the code are:

1. Compute evaluation metrics (precision, recall, F1-score) for CNN and traditional ML models.
2. Generate confusion matrices for each model.
3. Generate the model accuracy curves and computational time for CNN Models

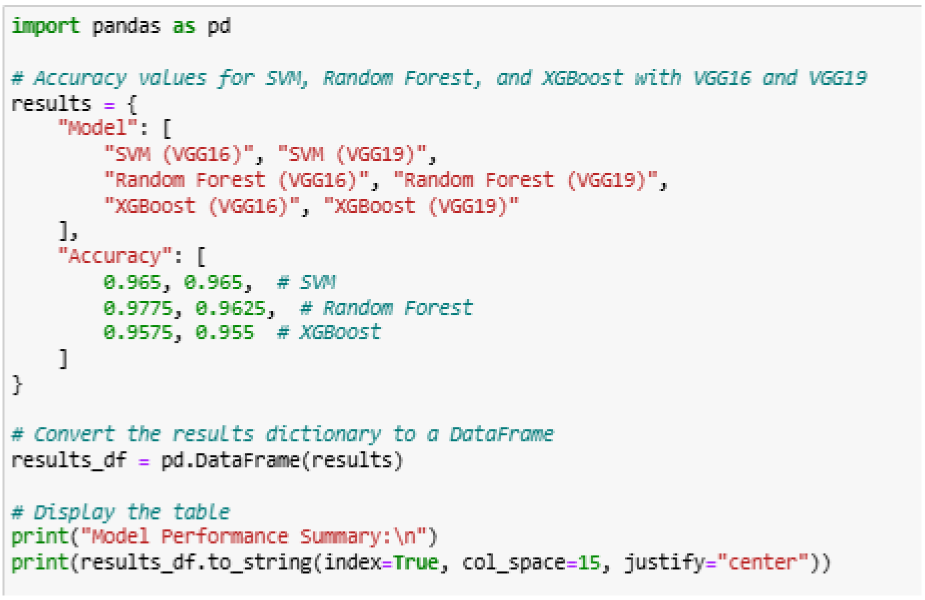


```python
import seaborn as sns
import pandas as pd
from sklearn.metrics import confusion_matrix

# Function to plot confusion matrix
def plot_confusion_matrix(y_true, y_pred, model_name):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_names, yticklabels=class_nar
    plt.title(f"Confusion Matrix: {model_name}")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()

# Generate predictions for validation sets
cnn_vgg16_val_predictions = models["VGG16"].predict(val_images).argmax(axis=1)
cnn_vgg19_val_predictions = models["VGG19"].predict(val_images).argmax(axis=1)

# Plot confusion matrices for CNN models
print("Confusion Matrix for VGG16 (CNN):")
plot_confusion_matrix(val_labels, cnn_vgg16_val_predictions, "VGG16")

print("Confusion Matrix for VGG19 (CNN):")
plot_confusion_matrix(val_labels, cnn_vgg19_val_predictions, "VGG19")
```



```python
import matplotlib.pyplot as plt

# Computational times (in minutes) and accuracies from the provided data
times = {
    "VGG16": 181.2,
    "VGG19": 230.67
}

# Training and validation accuracies for VGG16 and VGG19
accuracies = {
    "VGG16": [0.4188, 0.7338, 0.7932, 0.8189, 0.8445, 0.8517, 0.8704, 0.8765, 0.8857, 0.8995],
    "VGG19": [0.2963, 0.5608, 0.6306, 0.6571, 0.6881, 0.7098, 0.6987, 0.7014, 0.7128, 0.7354]
}

val_accuracies = {
    "VGG16": [0.8575, 0.9100, 0.9275, 0.9425, 0.9350, 0.9425, 0.9625, 0.9625, 0.9575, 0.9450],
    "VGG19": [0.8200, 0.8575, 0.9200, 0.9300, 0.9250, 0.9175, 0.9325, 0.9525, 0.9475, 0.9675]
}

losses = {
    "VGG16": [1.8264, 0.7438, 0.5670, 0.4715, 0.4249, 0.3863, 0.3635, 0.3317, 0.2950, 0.2730],
    "VGG19": [2.2870, 1.1172, 0.9422, 0.8554, 0.7764, 0.7103, 0.7347, 0.7376, 0.6735, 0.6431]
}

val_losses = {
    "VGG16": [0.6433, 0.3176, 0.2626, 0.2353, 0.1696, 0.1851, 0.1520, 0.1361, 0.1259, 0.1684],
    "VGG19": [0.8792, 0.5890, 0.4149, 0.3480, 0.2880, 0.2882, 0.2738, 0.1977, 0.1969, 0.2093]
}

# Generate Accuracy and Loss Curves
for model_name in ["VGG16", "VGG19"]:
    # Accuracy Curves
    plt.figure(figsize=(8, 5))
    plt.plot(range(1, 11), accuracies[model_name], label="Training Accuracy", color="blue")
    plt.plot(range(1, 11), val_accuracies[model_name], label="Validation Accuracy", color="red")
    plt.title(f"{model_name} Accuracy")
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.legend()
    plt.grid(True)
    plt.show()

    # Loss Curves
    plt.figure(figsize=(8, 5))
    plt.plot(range(1, 11), losses[model_name], label="Training Loss", color="blue")
    plt.plot(range(1, 11), val_losses[model_name], label="Validation Loss", color="red")
    plt.title(f"{model_name} Loss")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.legend()
    plt.grid(True)
    plt.show()

# Display Computational Times
print("Computational Time Summary (in minutes):")
for model, time in times.items():
    print(f"{model}: {time:.2f} minutes")
```

1. In the below part of the code models' performances of all traditional models with CNN are summarized and compared in a table.



```
import pandas as pd

# Accuracy values for SVM, Random Forest, and XGBoost with VGG16 and VGG19
results = {
    "Model": [
        "SVM (VGG16)", "SVM (VGG19)",
        "Random Forest (VGG16)", "Random Forest (VGG19)",
        "XGBoost (VGG16)", "XGBoost (VGG19)"
    ],
    "Accuracy": [
        0.965, 0.965,  # SVM
        0.9775, 0.9625,  # Random Forest
        0.9575, 0.955  # XGBoost
    ]
}

# Convert the results dictionary to a DataFrame
results_df = pd.DataFrame(results)

# Display the table
print("Model Performance Summary:\n")
print(results_df.to_string(index=True, col_space=15, justify="center"))
```

References

Ari, N., & Ustazhanov, M. (2014). Matplotlib in Python. In *2014 11th International Conference on Electronics, Computer and Computation (ICECCO)* (pp. 1–6). Abuja, Nigeria. https://doi.org/10.1109/ICECCO.2014.6997585

Navlani, A., Fandango, A., & Idris, I. (2021). *Python Data Analysis: Perform data collection, data processing, wrangling, visualization, and model building using Python.* Packt Publishing.

Stančin, I., & Jović, A. (2019). An overview and comparison of free Python libraries for data mining and big data analysis. In *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)* (pp. 977–982). Opatija, Croatia. https://doi.org/10.23919/MIPRO.2019.8757088

Lindstrom, G. (2005). Programming with Python. *IT Professional*, *7*(5), 10–16. https://doi.org/10.1109/MITP.2005.120