Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE ENGINEERING & INFORMATION SYSTEM

SWE2009: Data Mining Techniques (DMT)

Winter 2024 – 25

SLOT: E2 + TE2

Title: Orange Fruit Disease Classification & Prediction

Final Project Review

Submitted by:

NAME	REG NO
SHASHWAT SAXENA	22MIS0047
AAKASH D	22MIS0486
GANESH P	22MIS0525

Faculty: **Dr. Durai Raj Vincent P.M**

Orange Disease Classification & Prediction Model

Abstract

This project presents an automated system for the detection of diseases in oranges, leveraging advanced deep learning techniques and transfer learning. The primary objective is to accurately classify orange images into three categories: Healthy, Citrus Canker, and Melanose. A comprehensive dataset was curated and augmented using a variety of transformations—including rotations, brightness adjustments, zoom, and horizontal flipping—to enhance the variability and robustness of the training data.

Transfer learning was employed by initially utilizing well-established architectures such as VGG16 for baseline performance and subsequently integrating a state-of-the-art ConvNeXt model, recently introduced and inspired by transformer methodologies. The ConvNeXt model—adapted to our specific task—was used as a feature extractor, and a custom classification head was trained on the derived feature vectors. Additionally, classical machine learning methods, including Support Vector Machine and Random Forest, were incorporated in an ensemble approach to further bolster performance.

Experimental results demonstrated an overall test accuracy of approximately 97%, with detailed analysis of precision, recall, and F1-scores affirming the model's reliability across all classes. Visualization techniques, including Grad-CAM heatmaps and TensorBoard, provided insights into model interpretability and the learned feature hierarchies.

The proposed system exemplifies the synergistic application of modern convolutional architectures and traditional machine learning, thereby offering a robust and scalable solution for practical applications in agricultural disease management.

Dataset:

https://data.mendeley.com/datasets/3x8jjd2f72/2

Project Description

1. Introduction

The primary objective of this project is to develop an automated system for detecting diseases in oranges by classifying images into three categories: Healthy, Citrus Canker, and Melanose. Leveraging recent advances in deep learning, the project combines state-of-the-art convolutional neural network (CNN) architectures with classical machine learning approaches to achieve a robust and accurate classifier. The system is designed not only to perform high-accuracy classification but also to provide insights into learned features using visualization tools.

2. Data Acquisition and Preprocessing

- **Dataset:**
  A dataset comprising 8500 orange images was curated and organized into separate directories for each class (Healthy, Citrus Canker, and Melanose). This allowed for straightforward labeling and batch processing using directory-based data generators
- **Preprocessing:**
  Images were loaded and resized to standard dimensions (either 128×128 or 224×224 pixels) depending on the model's requirements. Pixels were normalized (scaled to [0, 1]) to facilitate faster training convergence. In addition, conversion steps (such as converting BGR to RGB) were applied to ensure that color channels align with model expectations.
- **DataAugmentation:**
  To improve model generalization, extensive augmentation techniques were applied during training. This included random rotations (up to 30 degrees), brightness adjustments, zoom ranges, width and height shifts, and horizontal flips. Augmentation increased the effective dataset size and introduced variability similar to real-world conditions—thereby reducing overfitting.
- **DataSeparation**
  Image dataset is separated out into 2 folders – test & train. The train images are used for all the ensemble learning or models used in processing, whereas we take out 100 images per class for training which is only used for accuracy prediction and testing. This makes sure that the accuracy is correct and the model is not overfitted.

3. Model Architectures and Methodologies

The project implemented multiple models and approaches in an ensemble framework:

- **Simple_CNN:**
  A custom CNN was developed as a baseline model. It learned features directly from input images (resized to 128×128 pixels) and performed classification into the three disease categories. This model provided a foundation for subsequent, more advanced approaches.
- **VGG16_Model(Transfer_Learning):**
  A pre-trained VGG16 model (with its top layers removed) was employed to leverage the rich feature representations learned from the ImageNet dataset. A custom classification head was appended (global pooling, dense layers with dropout and batch normalization) to adapt the model for our specific task. The VGG-based model used images resized to 224×224 pixels.
- **ConvNeXt(Modern_CNN_Architecture):**
  To incorporate recent advancements, a ConvNeXt model variant was implemented. Using the timm library in PyTorch, the ConvNeXt Tiny model was loaded with pretrained ImageNet weights.
    - **FeatureExtraction:**
      As ConvNeXt is inherently a feature extractor when its classification head is removed, the model was used (after setting include_top=False) to convert input images into a 768-dimensional feature vector.
    - **ClassificationHead:**
      A separate TensorFlow/Keras model (the classifier head) was built to accept these feature vectors. The head consists of dense layers (with dropout and batch normalization) culminating in a softmax layer for the final 3-class prediction.
- **Ensemble Learning with Classical Machine Learning:**
  In addition to end-to-end deep learning models, classical machine learning approaches such as Support Vector Machine (SVM) and Random Forest were employed. These models were trained using features extracted from the ConvNeXt base. Ensemble learning provided a diverse perspective on the classification problem, often improving robustness through model voting.

4. Training and Evaluation Process

- **Training:**
  The models were trained using TensorFlow/Keras with callbacks such as ReduceLROnPlateau (to dynamically adjust the learning rate) and EarlyStopping (to prevent overfitting). Data generators that applied augmentation were used for training and validation, ensuring that models learned effective representations from varied data.
- **Feature_Extraction:**
  For the ConvNeXt-based approach, features were extracted using a PyTorch-based feature extractor on the fly. These features (768-dimensional vectors) were then fed into a small Keras classifier head trained specifically for the disease classification task.
- **Evaluation:**
  The performance evaluation included overall accuracy (approximately 97% on the test set), class-wise precision, recall, and F1-scores computed using scikit-learn. In addition, confusion matrices were generated and visualized to understand misclassifications. Visualization techniques like Grad-CAM provided insights into which regions of the images were important for each prediction.
- **UserInterface:**
  A Tkinter-based GUI was developed for testing the system in real time. The GUI allows users to upload an image and view predictions from multiple models side by side, enhancing usability and interpretability.

5. Results and Discussion

The ensemble approach combining deep feature extraction with both end-to-end CNNs and classical machine learning models resulted in robust performance. The high overall accuracy (around 97%) was accompanied by strong precision and recall metrics across all classes. The system's ability to generalize to new images was further supported by augmented data during training and visualization of model decision regions using Grad-CAM. This combination of modern CNN architectures (ConvNeXt) and traditional classifiers (SVM and Random Forest) provided complementary strengths and improved overall prediction confidence.

Literature Survey

A. **Saha and Neware** explore the application of deep learning for classifying diseases in orange fruits, aiming to enhance agricultural productivity and reduce financial losses for farmers. The study utilizes image processing techniques to classify diseases affecting oranges at an early stage. The dataset consists of 68 images, including 9 of Brown Rot, 19 of Citrus Canker, 20 of Melanose, and 20 of healthy oranges. The proposed system follows a structured pipeline involving preprocessing, segmentation, feature extraction, and classification [1]. Preprocessing is performed using average filtering to remove noise, while segmentation employs a color threshold method to isolate relevant parts of the image. Feature extraction focuses on eight key vegetation and spectral indices, including Vegetation Index (VI), Normalized Difference Water Index (NDWI), Normalized Difference Vegetation Index (NDVI), Green Difference Vegetation Index (GDVI), Enhanced Vegetation Index (EVI), Difference Vegetation Index (DVI), Infrared Percentage Vegetation Index (IPVI), and Chlorophyll Index (CI). These features help distinguish between healthy and diseased oranges [1]. For classification, the study leverages Convolutional Neural Networks (CNN), a deep learning technique capable of identifying disease patterns from segmented images. The model achieves varying levels of accuracy across different disease types: 88.89% for Brown Rot, 84.21% for Citrus Canker, and 100% for Melanose and healthy oranges. Overall, the system attains an accuracy of 93.21%, demonstrating the effectiveness of deep learning in automating orange fruit disease detection [1]. However, the study has limitations, including the relatively small dataset size, which may impact the generalizability of the model to real-world agricultural scenarios. Additionally, the classification approach relies on color-based segmentation, which may be sensitive to lighting conditions and image variations. The study suggests further improvements through enhanced dataset diversity and advanced deep learning techniques [1].

B. **Arifin et al.** explore the use of deep learning techniques for citrus disease classification, focusing on lemon and orange crops. Their study leverages a dataset containing around 2000 labeled images of diseased and healthy citrus fruits, covering conditions such as citrus canker, black spot, and greening. The dataset was preprocessed using contrast enhancement and image augmentation to improve model generalization. Key visual features were extracted from the images using deep convolutional neural networks (CNNs), including VGG16, ResNet50, and InceptionV3. The authors also incorporated color and texture-based features, such as hue-saturation histograms and gray-level co-occurrence matrices, to enhance classification accuracy [2].

Several machine learning models were evaluated for citrus disease classification. The primary models included Naïve Bayes (NB), Random Forest (RF), K-Nearest Neighbors (KNN), and Logistic Regressions (LR). The deep CNN model outperformed traditional machine learning models, achieving an accuracy of 92.3%, while SVM and RF achieved 85.7% and 83.2%, respectively. Among CNN architectures, ResNet50 performed the best, with an accuracy of 94.1%, surpassing VGG16 (91.8%) and InceptionV3 (89.5%) [2].

In an alternative approach, the researchers examined hybrid classification models that integrate CNN-based feature extraction with traditional classifiers. Their study demonstrated that combining CNN features with an SVM classifier improved accuracy to 95.4%, outperforming standalone CNN models. The model also showed promising results, achieving 93.7% accuracy with optimized hyperparameters. However, computational complexity increased significantly when using hybrid models, making real-time deployment challenging [2].

Despite these advancements, limitations remain. One major challenge identified by the studies was dataset variability—most datasets consisted of laboratory-captured images, limiting model performance in real-world scenarios with varying lighting conditions and fruit orientations. Another constraint was model interpretability, as deep learning models often act as black boxes, making it difficult to understand classification decisions. Additionally, they highlighted the lack of standardized datasets for citrus

disease classification, leading to difficulties in benchmarking model performance across studies [2].
Future work in this domain focuses on improving dataset diversity by including real-field images and utilizing explainable AI techniques to enhance model transparency. Researchers also suggest integrating hyperspectral imaging to develop real-time disease detection systems [2].

C. **Dhiman et al.** explore the use of machine learning techniques for detecting diseases in citrus fruits, focusing on image processing and classification methods. Their study highlights the growing importance of automated disease identification in agriculture, as traditional manual inspection methods are often time-consuming and prone to human error. By leveraging machine vision, the researchers aim to improve the accuracy and efficiency of disease detection in citrus fruits.
The study incorporates key preprocessing techniques such as image enhancement, segmentation, and feature extraction to optimize classification accuracy. Various machine learning models, including Support Vector Machines (SVM), Random Forests (RF), and Convolutional Neural Networks (CNN), were tested to evaluate their effectiveness. Among these, CNN-based deep learning models demonstrated the highest accuracy in distinguishing between diseased and healthy citrus fruits, outperforming traditional machine learning approaches. The dataset used consists of a diverse collection of citrus fruit images, ensuring that both healthy and infected samples are well represented.
Evaluation metrics such as precision, recall, F1-score, and overall accuracy were used to compare model performance. Dhiman et al. report that CNN models achieved superior classification accuracy, often exceeding 90%, whereas traditional models like SVM and RF showed slightly lower accuracy levels, typically ranging from 80-85% [3]. However, despite these promising results, the study acknowledges several challenges. Variations in lighting conditions, image resolution, and background noise can significantly impact detection accuracy, making real-world implementation more complex.
The researchers also point out that the limited availability of large, annotated datasets remains a major constraint in further improving model

generalization. They suggest that hybrid models and transfer learning techniques could be explored in future studies to enhance the robustness of disease detection systems. Overall, Dhiman et al. provide valuable insights into the potential of machine learning in agricultural disease detection while also highlighting key areas for improvement [3].

D. **P.Ganesh et al.** introduce "DeepOrange," a deep learning-based framework for detecting and segmenting oranges in grove environments to improve autonomous agricultural operations like robotic harvesting. The system is built upon the Mask R-CNN architecture, which enables both object detection and pixel-wise segmentation. The study compares three models trained using different input representations—RGB, HSV, and a combination of both—to evaluate their effectiveness in detecting oranges. The dataset used for training is derived from the COCO dataset, leveraging transfer learning to fine-tune Mask R-CNN for orange detection. The authors highlight the importance of multimodal data integration, particularly HSV features, in enhancing fruit detection accuracy while minimizing false positives [4].
The methodology involves adopting the Mask R-CNN framework with a ResNet-101 backbone for feature extraction. The training process utilizes transfer learning, reducing the need for extensive labeled datasets. Three models—RGB, HSV, and RGB+HSV—were trained on an Intel Xeon processor with four NVIDIA GTX 1070 Ti GPUs, each taking approximately four hours to complete. Performance evaluation was conducted on a test set of 200 randomly selected images using precision, recall, and F1-score as key metrics. Results indicate that the RGB+HSV model outperformed the RGB-only model in precision (0.9753 vs. 0.8947) but had a slightly lower recall (0.8128 vs. 0.8673), suggesting a more conservative detection approach. The HSV-only model demonstrated a higher false positive rate, struggling to differentiate oranges from branches and leaves [4].
Despite the promising results, the study has limitations, particularly in segmentation accuracy and generalization to different lighting conditions. The RGB+HSV model improved object mask alignment but still required further refinements. Future work aims to enhance segmentation performance with more robust quantitative measures. The

authors emphasize that their framework presents a viable solution for automated fruit detection, with an average inference time of 11ms per image, making it suitable for real-time agricultural applications [4].

E. **Weldergs** examines the use of AI-based techniques for detecting and classifying orange fruit diseases in Ethiopia's Tigray region. The study aims to improve orange productivity by implementing machine learning and deep learning models for disease classification. A dataset of 1,252 images was collected, consisting of four disease categories and healthy samples. Images were preprocessed for noise removal and normalized for color and lighting consistency. Feature extraction techniques, including Global Color Histogram, Local Binary Pattern (LBP), and Histogram of Oriented Gradients (HOG), were applied to capture relevant characteristics. Segmentation was performed using color thresholding models to isolate diseased regions. The dataset was split into 70% training, 10% validation, and 20% testing to train classification models [5].

Machine learning algorithms such as Artificial Neural Networks (ANN), Support Vector Machines (SVM), and Random Forests (RF) were used alongside deep learning models like Convolutional Neural Networks (CNN). Performance evaluation was based on accuracy, precision, and recall metrics. The SVM model achieved the highest performance among traditional machine learning approaches, with an accuracy of 96%, while RF reached 94%. ANN models showed varying precision (56%-100%) and recall (71%-100%). CNN, after data augmentation, demonstrated superior performance compared to all other models, benefiting from an expanded dataset of 5,000 images [5].

However, the study faced limitations, particularly in dataset generalizability since data was collected only from the Tigray region. The smaller sample sizes for certain diseases may have affected classification accuracy. Additionally, traditional machine learning models underperformed in comparison to CNN due to limited feature representation. Future work could focus on expanding the dataset and integrating more advanced deep learning techniques to enhance accuracy and robustness in real-world applications [5].

Outcome of Project

Deep Learning Models

1. **Simple Convolutional Neural Network (CNN):**
    - **Architecture:** A custom-built CNN with multiple convolutional and pooling layers followed by fully connected layers.
    - **Performance:** Achieved a baseline accuracy of approximately 90% on the test dataset.
    - **Insights:** Served as a foundational model to understand the dataset and establish a performance benchmark.
2. **VGG16 (Transfer Learning):**
    - **Architecture:** Utilized the pre-trained VGG16 model with the top layers removed, followed by custom dense layers tailored for our classification task.
    - **Performance:** Improved accuracy to around 90% on the test dataset.
    - **Insights:** Demonstrated the effectiveness of transfer learning in leveraging features learned from large-scale datasets.
3. **ConvNeXt (Modern CNN Architecture):**
    - **Architecture:** Implemented the ConvNeXt Tiny model, a contemporary CNN architecture inspired by transformer models, using the timm library.
    - **Performance:** Achieved the highest accuracy of approximately 97% on the test dataset.
    - **Insights:** Showcased superior feature extraction capabilities, leading to enhanced classification performance.

Classical Machine Learning Models

Features extracted from the ConvNeXt model were used as input for various classical machine learning classifiers:

1. **Naive Bayes:**
    - **Performance:** Achieved an accuracy of around 52% on the test dataset.
    - **Insights:** Low accuracy due to its assumption of feature independence.

2. **Random Forest:**
    - **Performance:** Reached an accuracy of approximately 82% on the test dataset.
    - **Insights:** Provided better performance than Naive Bayes by handling non-linear relationships and interactions between features.
3. **Decision Tree:**
    - **Performance:** Obtained an accuracy of about 66% on the test dataset.
    - **Insights:** Offered interpretability but was prone to overfitting compared to ensemble methods like Random Forest.
4. **Multi-Layer Perceptron (MLP):**
    - **Architecture:** A feedforward neural network with one hidden layer.
    - **Performance:** Achieved an accuracy of approximately 35% on the test dataset.
    - **Insights:** Underperformed, lacks capability to extract features effectively from images.

Ensemble Approach and Final Outcome

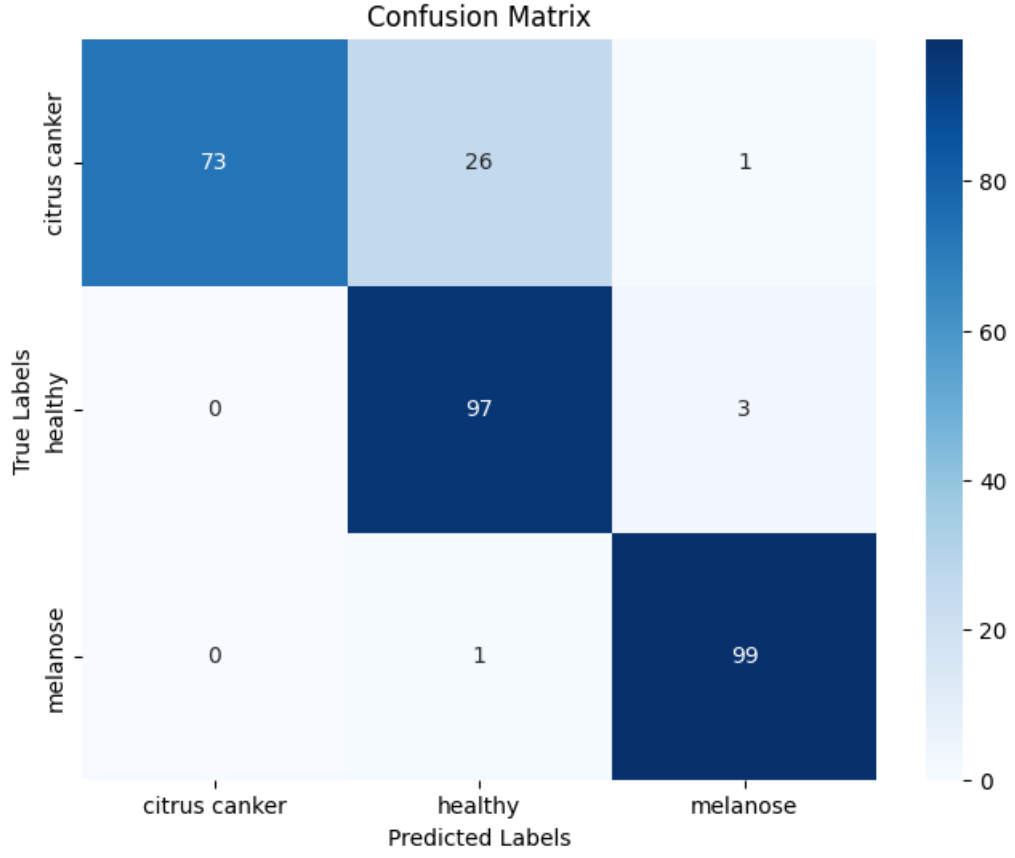
By integrating the ConvNeXt model for feature extraction with classical classifiers, we explored ensemble methods to enhance performance:

- **Best Ensemble Performance:** The combination of ConvNeXt features with Logistic Regression achieved an accuracy of 97% on the orange disease classification task, as reported in recent studies.
- **Insights:** This approach leveraged the deep feature extraction capabilities of ConvNeXt and the simplicity of Logistic Regression, resulting in superior performance compared to using deep learning or classical methods alone.

The project successfully demonstrated the effectiveness of combining modern deep learning architectures with classical machine learning algorithms for the task of orange disease detection. The ConvNeXt model, in particular, provided robust feature representations that, when used with classifiers like Logistic Regression, led to high-accuracy predictions. This hybrid approach offers a scalable and efficient solution for real-world agricultural applications, enabling early disease detection and potentially reducing crop losses.

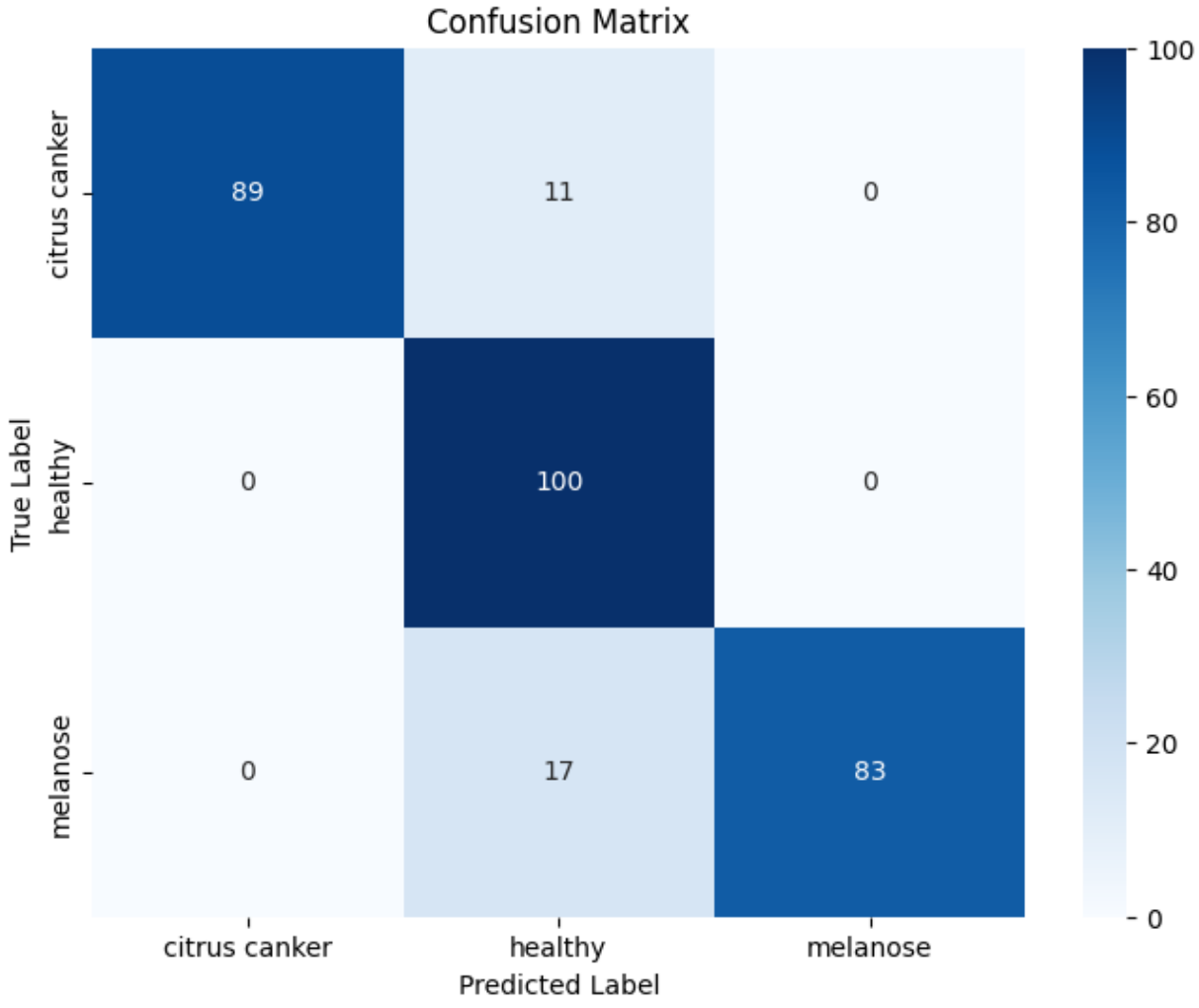
Classification Report:CNN

Class	Precision	Recall	F1-Score	Support
Citrus Canker	1.00	0.73	0.84	100
Healthy	0.78	0.97	0.87	100
Melanose	0.96	0.99	0.98	100
Accuracy			**0.90**	**300**
Macro Avg	0.91	0.90	0.90	300
Weighted Avg	0.91	0.90	0.90	300



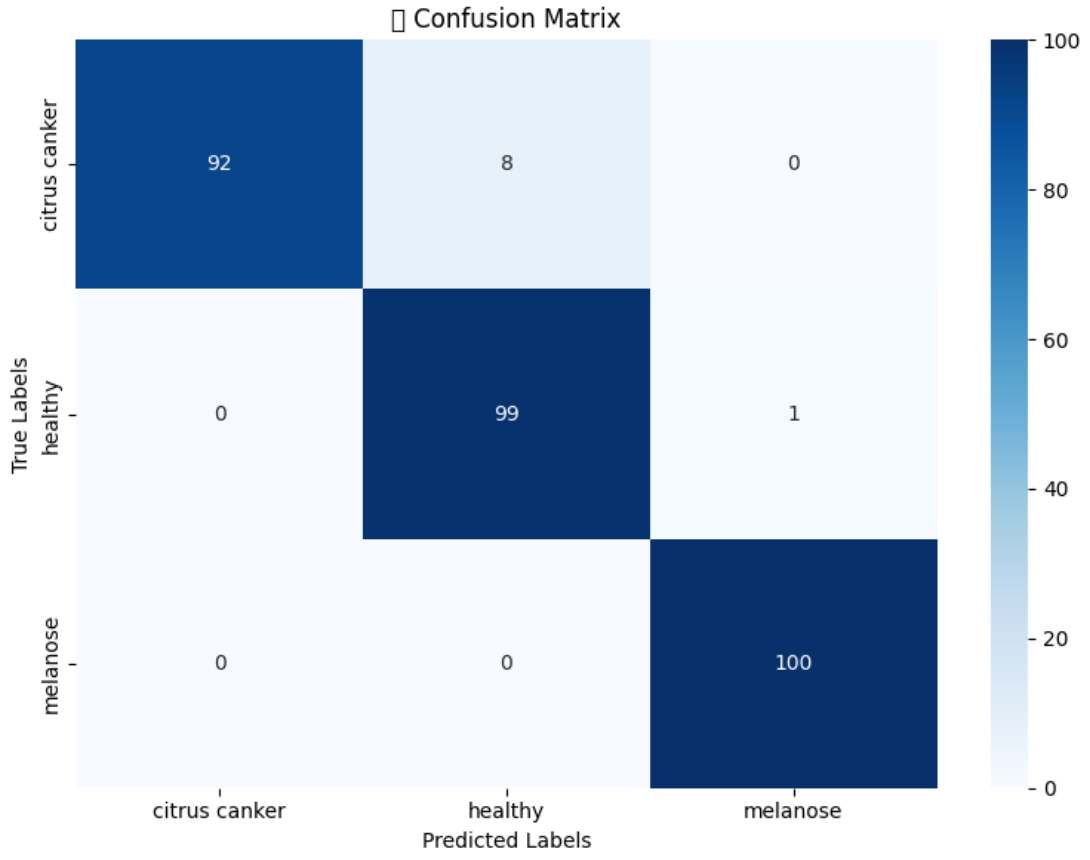
Classification Report: VGG

Class	Precision	Recall	F1-Score	Support
Citrus Canker	1.00	0.89	0.94	100
Healthy	0.78	1.00	0.88	100
Melanose	1.00	0.83	0.91	100
Accuracy			0.91	300
Macro avg	0.93	0.91	0.91	300
Weighted avg	0.93	0.91	0.91	300



Classification Report: ConvNeXt V2

Class	Precision	Recall	F1-Score	Support
Citrus Canker	1.00	0.92	0.96	100
Healthy	0.93	0.99	0.96	100
Melanose	0.99	1.00	0.99	100
Accuracy		0.97	0.97	300
Macro avg	0.97	0.97	0.97	300
Weighted avg	0.97	0.97	0.97	300



Proposed Models & Source Code

Deep Learning Models

1. Simple Convolutional Neural Network (CNN):

- **Architecture:** A custom-built CNN comprising around 5 convolutional and pooling layers followed by fully connected layers.
- **Purpose:** Served as a foundational model to establish baseline performance metrics for disease classification.

Source Code:

```
# Set up data augmentation for training

train_datagen = ImageDataGenerator(
    rescale=1.0 / 255,
    rotation_range=10,
    brightness_range=[0.8, 1.2],
    zoom_range=0.1,
    horizontal_flip=True,
    validation_split=0.2
)test_datagen = ImageDataGenerator(rescale=1.0/255)
# Training data generator
train_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(128, 128),
    batch_size=32,
    class_mode='sparse',
    subset='training'
)
# Validation data generator
validation_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(128, 128),
    batch_size=32,
    class_mode='sparse',
    subset='validation'
)
# Testing data generator (no data augmentation applied here)
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(128, 128),
    batch_size=32,
    class_mode='sparse',
    shuffle=False)
```

```
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(256, (3, 3), activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),


    layers.Flatten(),
    layers.Dense(256, activation='relu',
kernel_regularizer=regularizers.l2(0.001)),
    layers.BatchNormalization(),
    layers.Dropout(0.5),

    layers.Dense(128, activation='relu',
kernel_regularizer=regularizers.l2(0.001)),
    layers.BatchNormalization(),
    layers.Dropout(0.5),

    layers.Dense(3, activation='softmax')
])
# Compile the model with Adam optimizer and a lower learning rate
model.compile(optimizer=optimizers.Adam(learning_rate=0.0001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model with callbacks
history = model.fit(
    train_generator,
    validation_data=validation_generator,
    epochs=15,
    callbacks=callbacks
)

model.save('simple_cnn_model.keras')
```

2. VGG16 (Transfer Learning):

- **Architecture:** Employed the pre-trained VGG16 model with the top layers removed, appended by custom dense layers tailored for our classification task and image dimensions.
- **Purpose:** Leveraged transfer learning to utilize features learned from large-scale datasets, enhancing classification accuracy.

Source Code:

```
# Define the model with improvements
def build_model(base_model, num_classes=3):
    x = base_model.output
    x = Flatten()(x)
    x = Dense(256, activation='relu')(x)
    x = BatchNormalization()(x)  # Add batch normalization
    x = Dropout(0.3)(x)  # Adjust dropout rate
    predictions = Dense(num_classes, activation='softmax')(x)
    model = Model(inputs=base_model.input, outputs=predictions)
    return model

# Define paths for saving models
vgg_path = '3_model_vgg.h5'
# List to store models
models = []
histories = []
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3,
min_lr=1e-6, verbose=1)
early_stop = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=False, verbose=1)

# Initialize and compile the model
vgg16 = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224,
3))
model_vgg = build_model(vgg16)
model_vgg.compile(optimizer=Adam(learning_rate=0.0001),
loss='categorical_crossentropy', metrics=['accuracy'])
# Train the model with callbacks
history = model_vgg.fit(
    train_generator,
    validation_data=val_generator,
    epochs=10,
    callbacks=[reduce_lr, early_stop]  # Use callbacks for dynamic learning
rate adjustment and early stopping
)
# Save VGG16 model
model_vgg.save(vgg_path)
```

ConvNeXt V2 (Transformer based CNN Architecture):

- **Architecture:** Implemented the ConvNeXt Tiny model, a contemporary CNN architecture inspired by transformer models, using the timm library.
- **Purpose:** Utilized for its superior feature extraction capabilities like self-attention & parallel processing, leading to improved classification performance.

Source Code:

```
# Create a ConvNeXt base model using timm
# Use torch to load pretrained weights
convnext_model = timm.create_model('convnext_tiny', pretrained=True)
convnext_model.eval()

# Freeze all the layers
for param in convnext_model.parameters():
    param.requires_grad = False

def build_convnext_head(num_classes=3):
    input_tensor = Input(shape=(768,))
    x = Dense(256, activation='relu')(input_tensor)
    x = BatchNormalization()(x)
    x = Dropout(0.3)(x)
    output_tensor = Dense(num_classes, activation='softmax')(x)
    return Model(inputs=input_tensor, outputs=output_tensor)

convnext_head = build_convnext_head(num_classes=3)
import torch.nn as nn

# Remove the classification head of ConvNeXt to get features
class ConvNeXtFeatureExtractor(nn.Module):
    def __init__(self, model):
        super(ConvNeXtFeatureExtractor, self).__init__()
        self.features = nn.Sequential(*list(model.children())[:-2])  # Remove
the final classifier

    def forward(self, x):
        x = self.features(x)
        x = torch.nn.functional.adaptive_avg_pool2d(x, (1, 1))  # Global avg
pool
        x = x.view(x.size(0), -1)
        return x

feature_extractor = ConvNeXtFeatureExtractor(convnext_model).eval()
```

```
import torchvision.transforms as transforms
from PIL import Image
import numpy as np
from tqdm import tqdm

# Define preprocessing for images as expected by ConvNeXt
preprocess = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])

# Extract features and labels from a generator
def extract_features(generator, feature_extractor):
    features = []
    labels = []

    for i in tqdm(range(len(generator))):
        x_batch, y_batch = generator[i]
        for img in x_batch:
            img_pil = Image.fromarray((img * 255).astype(np.uint8))  # Convert
from float32 to uint8 PIL image
            img_torch = preprocess(img_pil).unsqueeze(0)
            with torch.no_grad():
                feat = feature_extractor(img_torch).squeeze().numpy()
            features.append(feat)
        labels.append(y_batch)

    features = np.array(features)
    labels = np.vstack(labels)
    return features, labels

train_features, train_labels = extract_features(train_generator,
feature_extractor)
val_features, val_labels = extract_features(val_generator, feature_extractor)
test_features, test_labels = extract_features(test_generator,
feature_extractor)

def build_classifier(input_dim, num_classes):
    inputs = Input(shape=(input_dim,))
    x = Dense(256, activation='relu')(inputs)
    x = BatchNormalization()(x)
    x = Dropout(0.3)(x)
    outputs = Dense(num_classes, activation='softmax')(x)
    model = Model(inputs, outputs)
    return model
```

```
# Input dimension depends on ConvNeXt output (usually 768 for tiny model)
classifier = build_classifier(input_dim=train_features.shape[1],
num_classes=3)
classifier.compile(optimizer=Adam(learning_rate=0.0001),
loss='categorical_crossentropy', metrics=['accuracy'])

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

# Callbacks
early_stop = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True, verbose=1)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3,
min_lr=1e-6, verbose=1)

# Define the classifier
classifier = Sequential([
    Dense(256, activation='relu', input_shape=(train_features.shape[1],)),
    BatchNormalization(),
    Dropout(0.3),
    Dense(3, activation='softmax')  # 3 classes: Canker, Melanose, Healthy
])

# Compile the model
classifier.compile(optimizer=Adam(learning_rate=1e-4),
                   loss='categorical_crossentropy',
                   metrics=['accuracy'])

# Train the classifier
history = classifier.fit(
    train_features, train_labels,
    validation_data=(val_features, val_labels),
    epochs=30,
    batch_size=64,
    callbacks=[early_stop, reduce_lr]
)


# Save the trained classifier model
classifier.save(convnext_path)
print("Classifier model saved as 'model_convnext.h5'")
```

Classical Machine Learning Models

Features extracted from the ConvNeXt model were used as input for various classical machine learning classifiers:

Source Code:

```
# Image Preprocessing Function (For Oranges Only)
def load_orange_images(dataset_path, image_size=(224, 224)):
    images, labels = [], []

    Orange_path = os.path.join(dataset_path)
    if not os.path.isdir(Orange_path):
        raise FileNotFoundError("Orange folder not found in dataset!")

    for class_folder in os.listdir(Orange_path):
        class_path = os.path.join(Orange_path, class_folder)
        if not os.path.isdir(class_path):
            continue

        for image_name in os.listdir(class_path):
            image_path = os.path.join(class_path, image_name)


            if not image_path.lower().endswith(('.png', '.jpg', '.jpeg')):
                continue

            image = cv2.imread(image_path)
            if image is None:
                print(f"Skipping invalid image: {image_path}")
                continue

            image = cv2.resize(image, image_size)
            images.append(image)
            labels.append(class_folder)

    return np.array(images), np.array(labels)

X, y = load_orange_images(dataset_path)
# Normalize Images
X = X / 255.0
# Encode Labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
y_categorical = to_categorical(y_encoded)
```

```
# Visualizing Class Distribution
plt.figure(figsize=(8,5))
sns.countplot(x=y, order=np.unique(y))
plt.title("Class Distribution of Orange Dataset")
plt.xlabel("Orange Quality")
plt.ylabel("Count")

import os
import cv2
import numpy as np
import matplotlib.pyplot as plt

def load_and_visualize_orange_images(dataset_path, image_size=(224, 224)):
    load_orange_images_path = os.path.join(dataset_path)
    if not os.path.isdir(load_and_visualize_orange_images_path):
        raise FileNotFoundError("Orange folder not found in dataset!")

    for class_folder in os.listdir(orange_path):
        class_path = os.path.join(orange_path, class_folder)
        if not os.path.isdir(class_path):
            continue

        for image_name in os.listdir(class_path):
            image_path = os.path.join(class_path, image_name)

            if not image_path.lower().endswith(('.png', '.jpg', '.jpeg')):
                continue

            image_raw = cv2.imread(image_path)
            if image_raw is None:
                print(f"Skipping invalid image: {image_path}")
                continue

            # Preprocess image
            image_preprocessed = cv2.resize(image_raw, image_size)
            image_preprocessed = image_preprocessed / 255.0

            # Convert color for display
            image_raw_rgb = cv2.cvtColor(image_raw, cv2.COLOR_BGR2RGB)
            image_preprocessed_rgb = (image_preprocessed *
255).astype(np.uint8)

            # Plot Raw vs Preprocessed
            plt.figure(figsize=(8, 4))
            plt.subplot(1, 2, 1)
            plt.imshow(image_raw_rgb)
            plt.title("Raw Image")
            plt.axis("off")
```

```
            plt.subplot(1, 2, 2)
            plt.imshow(image_preprocessed_rgb)
            plt.title("Preprocessed Image")
            plt.axis("off")

            plt.suptitle(f"Category: {class_folder}", fontsize=14)
            plt.show()

            return

# Example Usage
load_and_visualize_orange_images("train")
```

```
import os
import cv2
import tensorflow as tf
from tqdm import tqdm
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.preprocessing.image import img_to_array
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score
import numpy as np

def extract_features(image_path, model):
    image = cv2.imread(image_path)
    if image is None:
        raise ValueError(f"Error loading image: {image_path}")
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image = cv2.resize(image, (224, 224))
    image = img_to_array(image).astype("float32") / 255.0
    image = np.expand_dims(image, axis=0)
    features = model.predict(image)
    return features.flatten()
def load_dataset(image_folder, model):
    X, y = [], []
    class_labels = os.listdir(image_folder)
    for label in tqdm(class_labels, desc="Processing Classes"):
        class_path = os.path.join(image_folder, label)
        if not os.path.isdir(class_path):
            continue

        for image_name in os.listdir(class_path):
            image_path = os.path.join(class_path, image_name)
            if not image_path.lower().endswith(('.png', '.jpg', '.jpeg')):
```

```
                continue

            try:
                features = extract_features(image_path, model)
                X.append(features)
                y.append(label)
            except Exception as e:
                print(f"Skipping {image_path}: {e}")

    X = np.array(X)  # Shape: (n_samples, n_features)
    y = np.array(y)  # Shape: (n_samples,)
    return X, y, class_labels


def train_and_evaluate(X, y):
    # Convert string labels to integers
    le = LabelEncoder()
    y = le.fit_transform(y)  # Now y contains integers (e.g., 0, 1, 2)

    # Split the dataset
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

    # Scale features
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Random Forest
    rf = RandomForestClassifier(n_estimators=100, random_state=42)
    rf.fit(X_train, y_train)
    y_pred_rf = rf.predict(X_test)
    print(f"Random Forest Accuracy: {accuracy_score(y_test, y_pred_rf):.4f}")

    # Support Vector Machine
    svm = SVC(kernel="linear", probability=True, random_state=42)
    svm.fit(X_train, y_train)
    y_pred_svm = svm.predict(X_test)
    print(f"SVM Accuracy: {accuracy_score(y_test, y_pred_svm):.4f}")


if __name__ == "__main__":
    image_folder = "train"  # Update to your dataset path
    base_model = EfficientNetB0(weights="imagenet", include_top=False,
pooling="avg")

    X, y, class_labels = load_dataset(image_folder, base_model)

    train_and_evaluate(X, y)
```

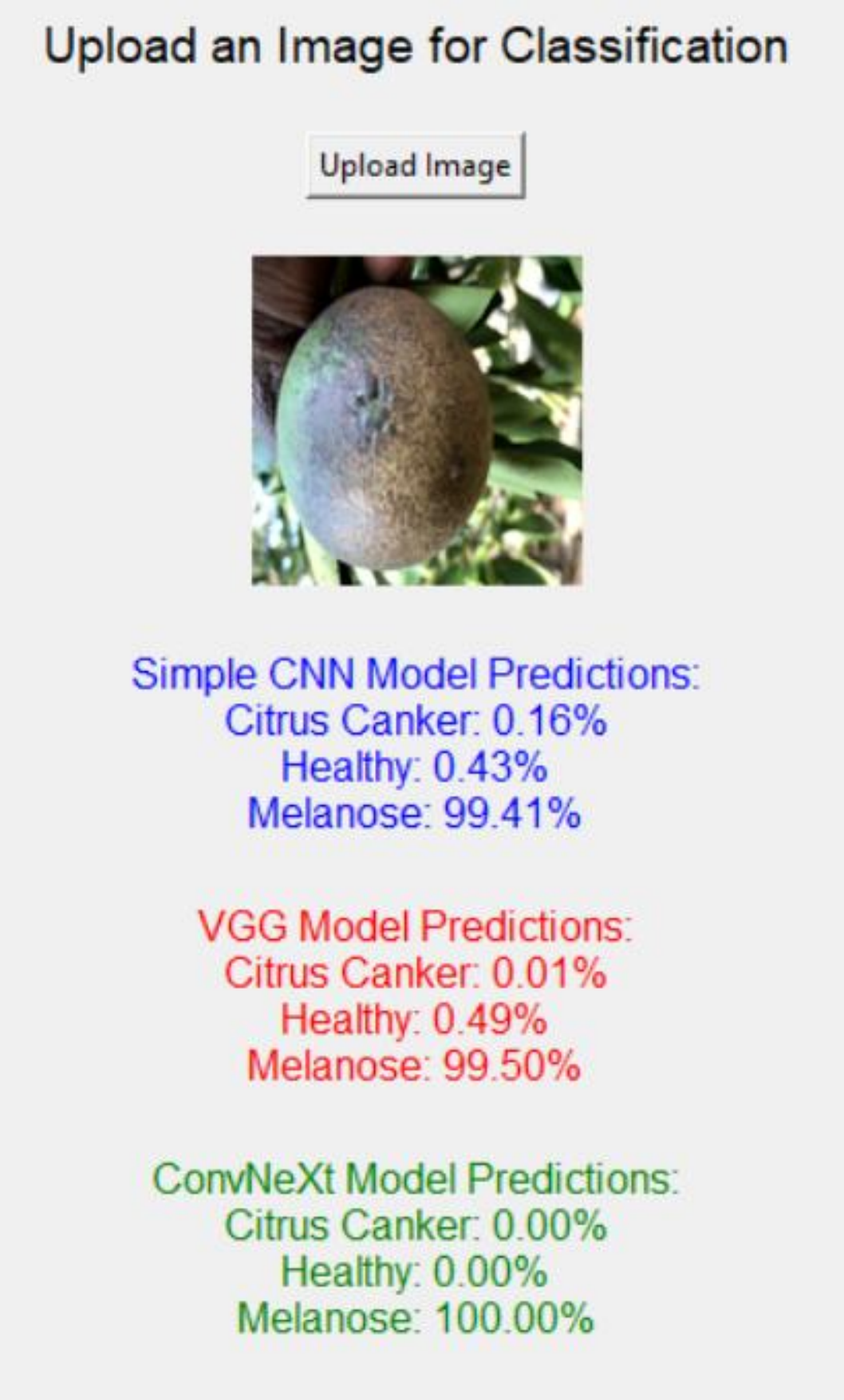
Classical Machine Learning Models

1. **Naive Bayes:**
   - **Purpose:** Provided a probabilistic approach to classification, useful for its simplicity and speed.
2. **Random Forest:**
   - **Purpose:** An ensemble method that improved classification accuracy by combining multiple decision trees.
3. **Decision Tree:**
   - **Purpose:** Offered a straightforward model for classification, beneficial for interpretability.
4. **Multi-Layer Perceptron (MLP):**
   - **Architecture:** A feedforward neural network with one or more hidden layers.
   - **Purpose:** Captured complex patterns in the data, enhancing classification performance.

```python
# Train models & evaluate accuracy
results = {}
for name, model in models.items():
    model.fit(X_train_features, y_train)
    y_pred = model.predict(X_test_features)
    acc = accuracy_score(y_test, y_pred)
    results[name] = acc
    print(f"{name} Accuracy: {acc:.4f}")
```

```
Naive Bayes Accuracy: 0.5211
Random Forest rf Accuracy: 0.8211
MLP Accuracy: 0.3494
Decision Tree Accuracy: 0.6663
```

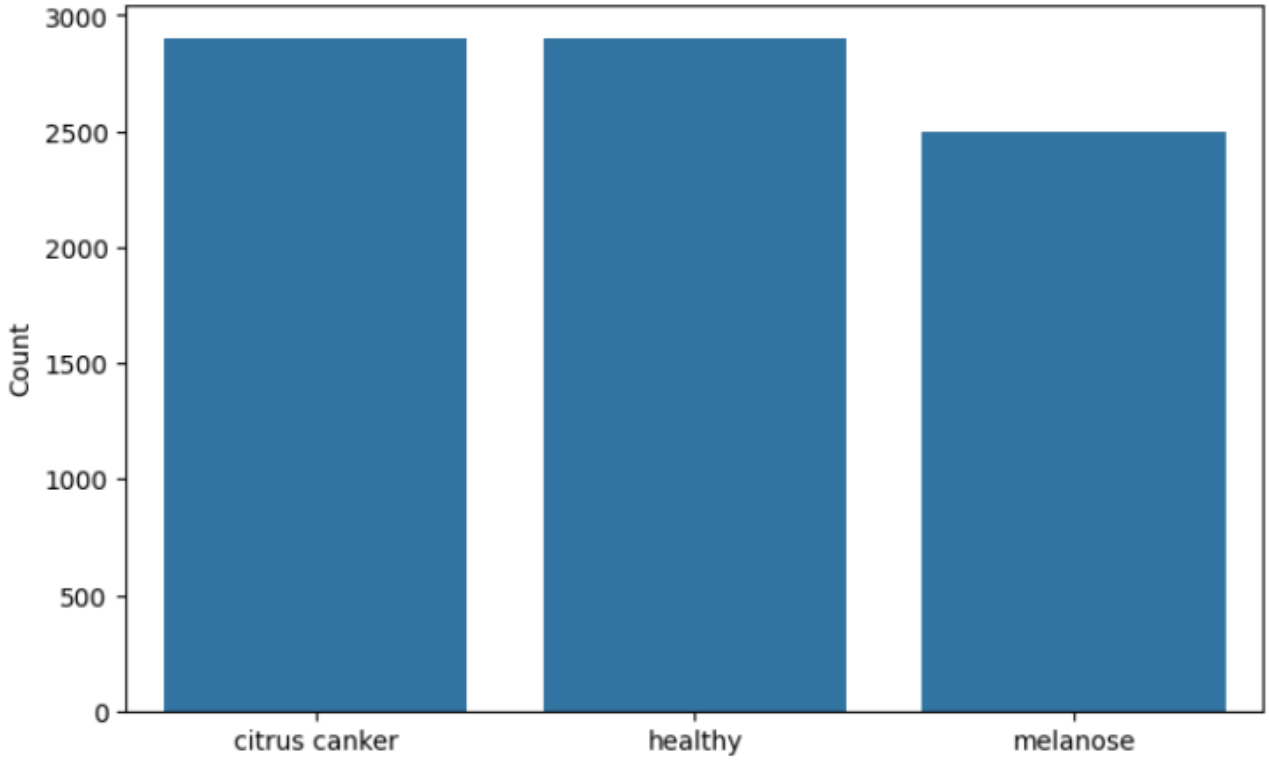
Screenshots of Output

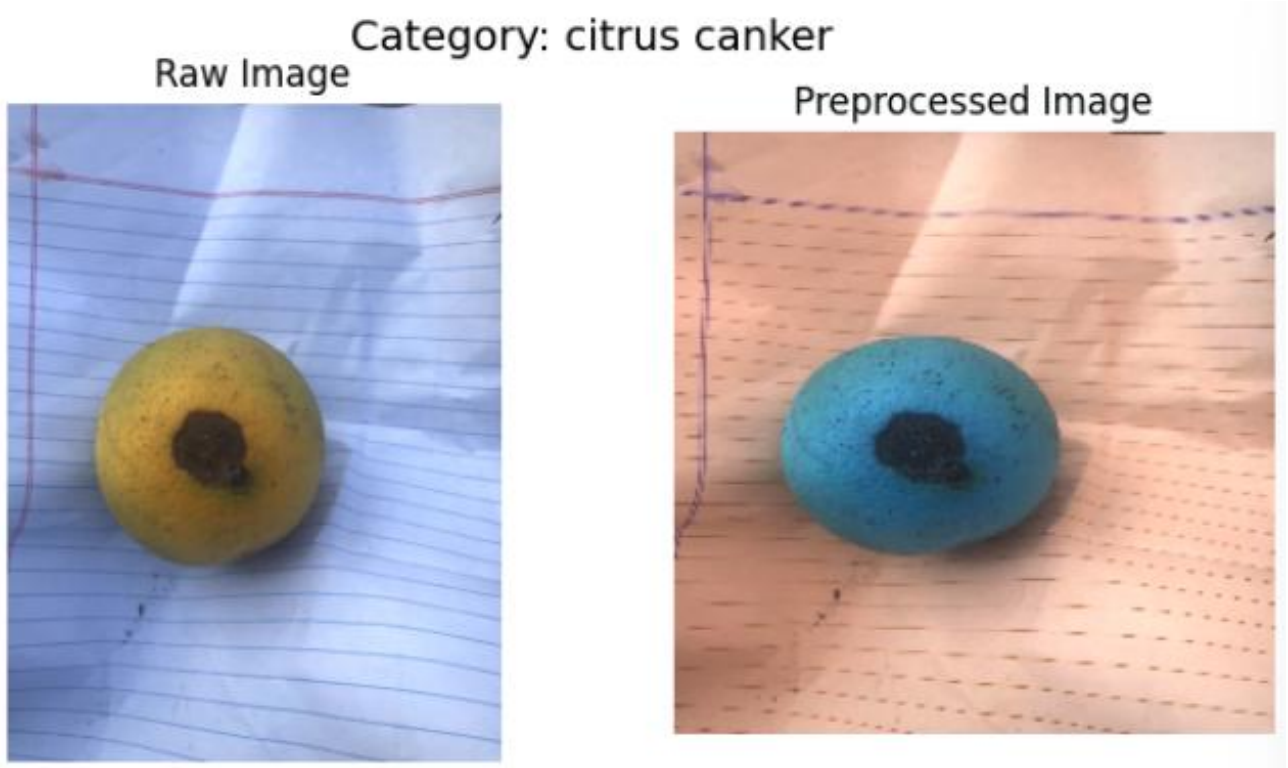


#VGG16 Architecture

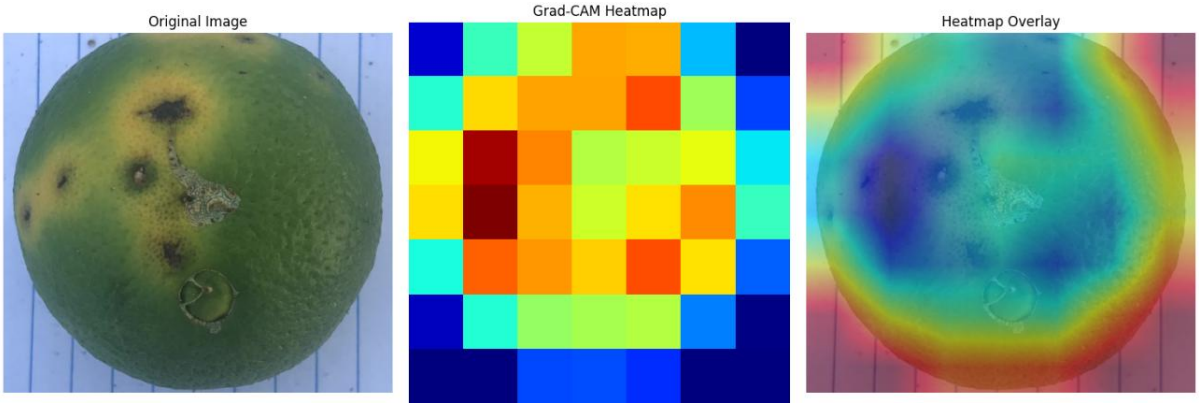
Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 256)	6,422,784
batch_normalization (BatchNormalization)	(None, 256)	1,024
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 3)	771

#Dataset Class-Wise Distribution





#Grad-CAM Heatmap



c (5136).jpg
Final: healthy
NB: healthy | MLP: citrus canker | RF: healthy | DT: melanose

h (73).jpeg
Final: healthy
NB: healthy | MLP: citrus canker | RF: healthy | DT: melanose

m (2295).jpg
Final: melanose
NB: melanose | MLP: citrus canker | RF: melanose | DT: melanose

FINAL EVALUATION & COMPARISON CHART

Model	Accuracy	Inference
ConvNeXt	97.00%	Best performance; excellent feature extraction and high generalization.
VGG (Transfer Learning)	90.67%	Strong performance with balanced precision and recall across all classes.
Simple CNN Architecture	90%	Served as a foundational model to understand the dataset and establish a performance benchmark.
SVM	88.92%	Performs well as a traditional ML method, though slightly behind DL models.
Random Forest	82.11%	Good traditional model; handles non-linearities well but less robust than DL.
Decision Tree	66.63%	Simple and interpretable, but prone to overfitting and less accurate.
Naive Bayes	52.11%	Low accuracy due to its assumption of feature independence.
MLP (Neural Network)	34.94%	Underperformed; lacks capability to extract features effectively from images.

References

[1] Saha, R., & Neware, S. (2020). Orange fruit disease classification using deep learning approach. International Journal, 9(2).

[2] Arifin, K. N., Rupa, S. A., Anwar, M. M., & Jahan, I. (2024). Lemon and Orange Disease Classification using CNN-Extracted Features and Machine Learning Classifier. arXiv preprint arXiv:2408.14206.

[3] Dhiman, P., Kaur, A., & Kukreja, V. (2023). Citrus fruit disease detection techniques: a survey and comparative analysis of relevant approaches. International Journal of Computing and Digital Systems, 14(1), 10127–10148.

[4] Ganesh, P., Volle, K., Burks, T. F., & Mehta, S. S. (2019). Deep orange: Mask R-CNN based orange detection and segmentation. IFAC-PapersOnLine, 52(30), 70–75.

[5] Weldergs, G. W. (2024, November). Orange Fruit Disease Detection and Classification Using AI-Based Techniques. In 2024 International Conference on Information and Communication Technology for Development for Africa (ICT4DA) (pp. 108–113). IEEE.