NAME: Chinonye Ukachukwu

STUDENT ID: 23037498

COURSE TITLE: Machine Learning

GITHUB LINK:

https://github.com/nonyeashley/DEEP-LEARNING-MODELS-FOR-CASSAVA-DISEASE-DETECTION

LINK TO THE DATASET:

https://www.kaggle.com/datasets/visalakshiiyer/cassava-image-dataset

DEEP LEARNING MODELS FOR CASSAVA DISEASE DETECTION: A COMPARATIVE ANALYSIS ON CNN, RESNET50, INCEPTIONV3, AND MOBILENETV2

ABSTRACT

In tropical areas, cassava is a vital crop that offers substantial nutritional and economic benefits. However, illnesses like Cassava Brown Streak Disease (CBSD) and Cassava Mosaic Disease (CMD) pose serious risks to cassava cultivation and significantly reduce yield. In order to control the spread of these illnesses and reduce their negative economic effects, early detection is essential.

In this study, I investigate the use of deep learning models to identify cassava illness from photos of leaves. I evaluate the accuracy and computational efficiency of Convolutional Neural Networks (CNNs), ResNet50, InceptionV3, and MobileNetV2 by comparing them. Three classes of cassava—Healthy, Brown Streak Disease, and Mosaic Disease—were used to train the models.

Following model training, MobileNetV2 outperformed CNN, ResNet50, and InceptionV3 with the greatest accuracy of 94% on the validation set. This demonstrates how well MobileNetV2 can categorize cassava infections, which makes it a good model for real-time deployment in settings with limited resources, including mobile applications and on-site disease detection systems.

1. INTRODUCTION

In many tropical nations, cassava is a staple crop that serves as the main source of carbohydrates. However, illnesses like Cassava Mosaic Disease (CMD) and Cassava Brown Streak Disease (CBSD) frequently endanger its productivity. These viral infections have the potential to significantly lower yields and degrade the quality of cassava tubers, which are essential to regional economies.(METLEK, 2021)

Inaccurate and time-consuming manual checks are a major component of the traditional techniques of detecting many diseases, particularly in their early stages. Now that deep learning technology, especially Convolutional Neural Networks (CNNs), have advanced, a potent tool for automating this identification procedure has been developed. Using pictures of cassava leaves, this tutorial attempts to investigate four distinct deep learning models: CNN, ResNet50, InceptionV3, and MobileNetV2. We evaluate the models' precision, computational effectiveness, and suitability for actual agricultural settings by contrasting them.(Ahishakiye et al., 2024)

2. LITERATURE REVIEW

2.1 CASSAVA DISEASE DETECTION USING DEEP LEARNING

Global cassava production is being threatened by cassava diseases, especially cassava mosaic disease (CMD) and cassava brown streak disease (CBSD). In many areas, these diseases have a direct impact on food security since they can reduce agricultural yields by up to 30%. To reduce the losses and take prompt action to stop their spread, early diagnosis and precise identification of these diseases are crucial. However, in large-scale agricultural contexts, manual disease detection is less reliable due to its subjectivity and reliance on human skill.(Anusha et al., 2024)

Promising approaches to automated plant disease diagnosis are provided by recent developments in deep learning and machine learning. Agriculture is one of the many fields that have used deep learning, particularly with Convolutional Neural Networks (CNNs), for picture categorization tasks. For jobs like identifying infections in cassava leaves, where visual patterns are essential for differentiating between healthy and infected leaves, CNNs are perfect because of their exceptional image processing and classification capabilities. (Omaye et al., 2024)

2.2 CNNS AND TRANSFER LEARNING

Because CNNs can directly learn hierarchical features from raw picture data, they have been frequently employed for plant disease identification. Manual feature extraction was necessary for traditional machine learning techniques like Support Vector Machines (SVMs) and k-Nearest Neighbors (k-NN), which frequently missed the intricate patterns in plant photos. CNNs, on the other hand, greatly increase accuracy by automatically learning properties like edges, forms, and textures at several layers.(Emmanuel et al., 2023)

Deep learning's effectiveness in small-dataset applications has been further increased by transfer learning. Large datasets like ImageNet are used to pre-train models like ResNet50, InceptionV3, and MobileNetV2, which can then be adjusted for particular purposes like detecting cassava disease using comparatively smaller datasets.

DATASET AND PREPROCESSING

Dataset Overview

The dataset for this project contains cassava leaf images categorized into three classes.

- Healthy: Leaves that show no signs of disease.
- Brown Streak Disease (BSD): Leaves with visible symptoms of CBSD.
- Mosaic Disease (MD): Leaves affected by CMD.

```
# Define the path to the dataset
data_dir = r'C:\Users\ibuku\Downloads\archive (1)'
brown_streak_dir = os.path.join(data_dir, 'brown_streak')
healthy1_dir = os.path.join(data_dir, 'healthy1')
mosaic_disease1_dir = os.path.join(data_dir, 'mosaic_disease1')

# Create a DataFrame to hold the image paths and labels
data = []

for label, folder in enumerate(['brown_streak', 'healthy1', 'mosaic_disease1']):
    folder_path = os.path.join(data_dir, folder)
    for filename in os.listdir(folder_path):
        if filename.endswith('.jpg') or filename.endswith('.png'):
            data.append([os.path.join(folder_path, filename), label])

df = pd.DataFrame(data, columns=['file_path', 'label'])

# Inspect the first few rows
print(df.head())
```

```
                                           file_path  label
0  C:\Users\ibuku\Downloads\archive (1)\brown_str...      0
1  C:\Users\ibuku\Downloads\archive (1)\brown_str...      0
2  C:\Users\ibuku\Downloads\archive (1)\brown_str...      0
3  C:\Users\ibuku\Downloads\archive (1)\brown_str...      0
4  C:\Users\ibuku\Downloads\archive (1)\brown str...      0
```

Figure 1: Loading the cassava disease dataset and setting It to label.

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

# Function to visualize a random sample of images from each class
def visualize_images(df, num_samples=5):
    classes = df['label'].unique()
    plt.figure(figsize=(15, 7))

    for class_label in classes:
        class_df = df[df['label'] == class_label].sample(num_samples)

        for i, row in enumerate(class_df.iterrows(), 1):
            plt.subplot(len(classes), num_samples, (class_label * num_samples) + i)
            img_path = row[1]['file_path']
            img = mpimg.imread(img_path)
            plt.imshow(img)
            plt.axis('off')

            if i == 1:
                plt.title(f"Class {class_label}")

    plt.tight_layout()
    plt.show()

# Visualize a random sample of images from each class
visualize_images(df)
```

Figure 2 : outputting the image dataset

```
plt.figure(figsize=(8, 8))
plt.pie(class_counts.values, labels=class_counts.index, autopct='%1.1f%%')
plt.title('Class Distribution in Cassava Dataset')
plt.show()
```

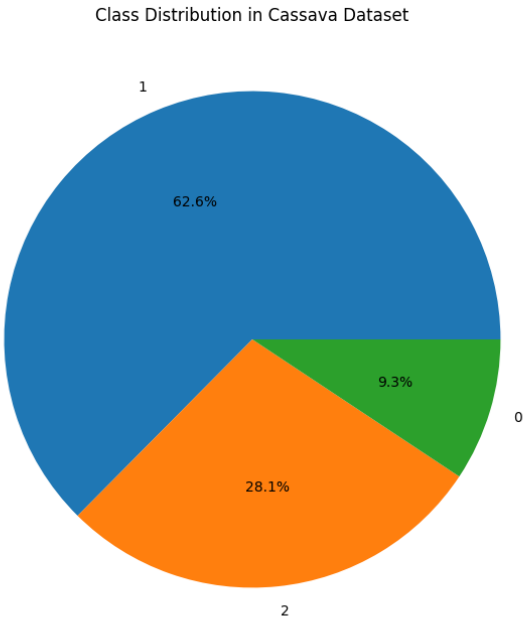


Figure 3: pie chart showing the percentage of (healthy (1), brown streak (0) and mosaic disease (3)

3.1 DATA PREPROCESSING AND AUGMENTATION

To prepare the dataset for deep learning, the following preprocessing steps are applied:

- **Resizing**: To fit the input size that the models require, images are scaled to 224x224 pixels.
- **Normalization**: For steady training, pixel values are scaled to fall between [0, 1].
- **Data Augmentation**: In order to avoid overfitting and enhance model generalization, methods including rotation, zooming, horizontal flipping, and shifting are employed to artificially expand the dataset size.

```
img_width, img_height = 224, 224
batch_size = 32

# Check for and remove corrupted images
def remove_corrupted_images(data_dir):
    for subdir in ['brown_streak', 'healthy1', 'mosaic_disease1']:
        folder_path = os.path.join(data_dir, subdir)
        for filename in os.listdir(folder_path):
            file_path = os.path.join(folder_path, filename)
            try:
                img = load_img(file_path)
                img_array = img_to_array(img)
            except Exception as e:
                print(f"Removing corrupted image: {file_path}")
                os.remove(file_path)

remove_corrupted_images(data_dir)

# Data augmentation and normalization
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2)  # Splitting the data into training and validation sets

train_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    subset='training')  # Set as training data

validation_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation')  # Set as validation data
```

Figure 4: Data preprocessing

3. DEEP LEARNING ALGORITHMS OVERVIEW

WHAT IS DEEP LEARNING?

Neural networks having multiple layers—thus the term "deep"—learn from vast volumes of data in deep learning, a subset of machine learning. Due to their capacity to automatically extract pertinent features from unprocessed data, deep learning models—in particular, CNNs—have shown remarkable performance in tasks such as picture classification, which makes them perfect for detecting plant diseases.

WHY USE DEEP LEARNING FOR PLANT DISEASE DETECTION?

For complicated visual data, such as plant photos, it might be challenging to identify the manually created features needed by traditional machine learning methods. CNNs are ideal for detecting plant diseases since they can automatically learn these characteristics and have been shown to perform better than conventional techniques in picture classification tasks.

5.1 TRAINING THE MODEL USED

Every picture has a label that indicates whether the leaf is healthy or has a sickness. To meet the models' input requirements, the dataset is divided into training (80%) and validation (20%) sets, with the photos scaled to 224x224 pixels.

```
remove_corrupted_images(data_dir)

# Data augmentation and normalization
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2)  # Splitting the data into training and validation sets
```

Figure 5: splitting dataset into training (80%) and testing (20).

5. MODEL ARCHITECTURES AND TRAINING OF THE MODEL.

CONVOLUTIONAL NEURAL NETWORK (CNN)\

For image classification applications, a custom CNN architecture is straightforward but efficient. Multiple convolutional layers are used to identify low-level features, pooling layers are used to lower the dimensionality, and dense layers are used to provide the classification.

```python
# Define the CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(img_width, img_height, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(3, activation='softmax')  # Adjust the number of classes accordingly
])

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

# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=10,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size)
```

```
Found 429 images belonging to 3 classes.
Found 107 images belonging to 3 classes.
Epoch 1/10
13/13 [==============================] - 44s 3s/step - loss: 1.1471 - accuracy: 0.5516 - val_loss: 0.8491 - val_accuracy: 0.6562
Epoch 2/10
13/13 [==============================] - 40s 3s/step - loss: 0.8044 - accuracy: 0.6751 - val_loss: 0.7153 - val_accuracy: 0.7396
Epoch 3/10
13/13 [==============================] - 40s 3s/step - loss: 0.7140 - accuracy: 0.7154 - val_loss: 0.6869 - val_accuracy: 0.6667
Epoch 4/10
13/13 [==============================] - 41s 3s/step - loss: 0.6787 - accuracy: 0.7280 - val_loss: 0.6517 - val_accuracy: 0.7500
Epoch 5/10
13/13 [==============================] - 39s 3s/step - loss: 0.5415 - accuracy: 0.7708 - val_loss: 0.4795 - val_accuracy: 0.8021
Epoch 6/10
13/13 [==============================] - 40s 3s/step - loss: 0.5283 - accuracy: 0.7909 - val_loss: 0.5967 - val_accuracy: 0.8229
Epoch 7/10
13/13 [==============================] - 40s 3s/step - loss: 0.4796 - accuracy: 0.8363 - val_loss: 0.4066 - val_accuracy: 0.8646
Epoch 8/10
13/13 [==============================] - 40s 3s/step - loss: 0.3516 - accuracy: 0.8514 - val_loss: 0.4962 - val_accuracy: 0.8542
Epoch 9/10
13/13 [==============================] - 39s 3s/step - loss: 0.3491 - accuracy: 0.8640 - val_loss: 0.4284 - val_accuracy: 0.8125
Epoch 10/10
13/13 [==============================] - 39s 3s/step - loss: 0.3595 - accuracy: 0.8539 - val_loss: 0.4225 - val_accuracy: 0.8438
```

Figure 6: Training CNN model.

RESNET50

A pre-trained deep residual network called ResNet50 is well-known for its capacity to train deep models without encountering the vanishing gradient issue. It acquired good accuracy after being refined on the cassava dataset.

```python
# Load pre-trained ResNet50 model
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(img_width, img_height, 3))

# Freeze the base model layers
base_model.trainable = False

# Build the model on top of ResNet50
model1 = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(256, activation='relu'),
    Dense(3, activation='softmax')  # Adjust the number of classes accordingly
])

# Compile the model
model1.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model1.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=10,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size)
```

```
Found 429 images belonging to 3 classes.
Found 107 images belonging to 3 classes.
Epoch 1/10
13/13 [==============================] - 63s 4s/step - loss: 1.2430 - accuracy: 0.5189 - val_loss: 1.0991 - val_accuracy: 0.6146
Epoch 2/10
13/13 [==============================] - 54s 4s/step - loss: 0.9220 - accuracy: 0.6373 - val_loss: 0.9204 - val_accuracy: 0.6042
Epoch 3/10
13/13 [==============================] - 55s 4s/step - loss: 0.8177 - accuracy: 0.6322 - val_loss: 0.8431 - val_accuracy: 0.6146
Epoch 4/10
13/13 [==============================] - 54s 4s/step - loss: 0.8065 - accuracy: 0.6222 - val_loss: 0.8330 - val_accuracy: 0.6146
Epoch 5/10
13/13 [==============================] - 56s 4s/step - loss: 0.7775 - accuracy: 0.6222 - val_loss: 0.8841 - val_accuracy: 0.5833
Epoch 6/10
13/13 [==============================] - 53s 4s/step - loss: 0.7459 - accuracy: 0.6776 - val_loss: 0.8333 - val_accuracy: 0.6042
Epoch 7/10
13/13 [==============================] - 53s 4s/step - loss: 0.7398 - accuracy: 0.6650 - val_loss: 0.7741 - val_accuracy: 0.6562
Epoch 8/10
13/13 [==============================] - 53s 4s/step - loss: 0.7144 - accuracy: 0.6700 - val_loss: 0.7945 - val_accuracy: 0.6146
Epoch 9/10
13/13 [==============================] - 53s 4s/step - loss: 0.6721 - accuracy: 0.6952 - val_loss: 0.7558 - val_accuracy: 0.6562
Epoch 10/10
13/13 [==============================] - 56s 4s/step - loss: 0.7033 - accuracy: 0.6538 - val_loss: 0.7675 - val_accuracy: 0.6146
```

Figure 7: Training ResNet 50 model.

INCEPTIONV3

To collect features at various scales, InceptionV3 employs several filters of various sizes at each layer. It showed strong performance and was adjusted for the detection of cassava disease.

```python
# Load pre-trained InceptionV3 model
base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(img_width, img_height, 3))

# Freeze the base model layers
base_model.trainable = False

# Build the model on top of InceptionV3
model2= Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(256, activation='relu'),
    Dense(3, activation='softmax')  # Adjust the number of classes accordingly
])

# Compile the model
model2.compile(optimizer='adam',
               loss='categorical_crossentropy',
               metrics=['accuracy'])

# Train the model
history = model2.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=10,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size)
```

```
Found 429 images belonging to 3 classes.
Found 107 images belonging to 3 classes.
Epoch 1/10
13/13 [==============================] - 53s 4s/step - loss: 0.9368 - accuracy: 0.7103 - val_loss: 0.3765 - val_accuracy: 0.8854
Epoch 2/10
13/13 [==============================] - 41s 3s/step - loss: 0.3586 - accuracy: 0.8615 - val_loss: 0.4604 - val_accuracy: 0.7708
Epoch 3/10
13/13 [==============================] - 40s 3s/step - loss: 0.2547 - accuracy: 0.9068 - val_loss: 0.3046 - val_accuracy: 0.8750
Epoch 4/10
13/13 [==============================] - 41s 3s/step - loss: 0.1844 - accuracy: 0.9421 - val_loss: 0.5246 - val_accuracy: 0.7708
Epoch 5/10
13/13 [==============================] - 41s 3s/step - loss: 0.2019 - accuracy: 0.9270 - val_loss: 0.5243 - val_accuracy: 0.8021
Epoch 6/10
13/13 [==============================] - 41s 3s/step - loss: 0.1574 - accuracy: 0.9421 - val_loss: 0.5401 - val_accuracy: 0.7604
Epoch 7/10
13/13 [==============================] - 42s 3s/step - loss: 0.1118 - accuracy: 0.9521 - val_loss: 0.2829 - val_accuracy: 0.8750
Epoch 8/10
13/13 [==============================] - 40s 3s/step - loss: 0.1001 - accuracy: 0.9748 - val_loss: 0.4252 - val_accuracy: 0.8333
Epoch 9/10
13/13 [==============================] - 41s 3s/step - loss: 0.1091 - accuracy: 0.9622 - val_loss: 0.4635 - val_accuracy: 0.8333
Epoch 10/10
13/13 [==============================] - 42s 3s/step - loss: 0.0812 - accuracy: 0.9712 - val_loss: 0.1958 - val_accuracy: 0.9271
```

Figure 8: Training inception v3 model.

MobileNetV2: For mobile and edge devices, MobileNetV2 is a lightweight, effective model. It maintains good performance while lowering computational complexity through the use of depth-wise separable convolutions

```
# Load pre-trained MobileNetV2 model
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(img_width, img_height, 3))

# Freeze the base model layers
base_model.trainable = False

# Build the model on top of MobileNetV2
model3= Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(256, activation='relu'),
    Dense(3, activation='softmax')  # Adjust the number of classes accordingly
])

# Compile the model
model3.compile(optimizer='adam',
               loss='categorical_crossentropy',
               metrics=['accuracy'])

# Train the model
history = model3.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=10,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size)
```

```
Found 429 images belonging to 3 classes.
Found 107 images belonging to 3 classes.
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_224_
no_top.h5
9406464/9406464 [==============================] - 12s 1us/step
Epoch 1/10
13/13 [==============================] - 44s 3s/step - loss: 0.9303 - accuracy: 0.6977 - val_loss: 0.4813 - val_accuracy: 0.7396
Epoch 2/10
13/13 [==============================] - 35s 3s/step - loss: 0.2415 - accuracy: 0.9118 - val_loss: 0.3584 - val_accuracy: 0.8438
Epoch 3/10
13/13 [==============================] - 34s 3s/step - loss: 0.1709 - accuracy: 0.9471 - val_loss: 0.2375 - val_accuracy: 0.9167
Epoch 4/10
13/13 [==============================] - 34s 3s/step - loss: 0.0915 - accuracy: 0.9698 - val_loss: 0.3309 - val_accuracy: 0.8646
Epoch 5/10
13/13 [==============================] - 34s 3s/step - loss: 0.0813 - accuracy: 0.9723 - val_loss: 0.2843 - val_accuracy: 0.8854
Epoch 6/10
13/13 [==============================] - 34s 3s/step - loss: 0.0534 - accuracy: 0.9849 - val_loss: 0.2549 - val_accuracy: 0.9062
Epoch 7/10
13/13 [==============================] - 35s 3s/step - loss: 0.0448 - accuracy: 0.9849 - val_loss: 0.2772 - val_accuracy: 0.8958
Epoch 8/10
13/13 [==============================] - 34s 3s/step - loss: 0.0483 - accuracy: 0.9924 - val_loss: 0.4701 - val_accuracy: 0.8125
Epoch 9/10
13/13 [==============================] - 34s 3s/step - loss: 0.0504 - accuracy: 0.9824 - val_loss: 0.3752 - val_accuracy: 0.8646
Epoch 10/10
13/13 [==============================] - 35s 3s/step - loss: 0.0446 - accuracy: 0.9849 - val_loss: 0.2627 - val_accuracy: 0.8854
```

Figure 9: training Mobilenet v2 model.

5.2 TRAINING AND VALIDATION ACCURACY

TRAINING ACCURACY:

The percentage of accurate predictions the model makes on the training data during the training phase is known as training accuracy. It is computed using the model's training data at the end of each training epoch (or pass).

Relevance: A high training accuracy shows that the model has done a good job of learning the patterns in the training set. However, as training accuracy ignores the model's ability to generalize to new, unknown data, it is insufficient to assess the model's efficacy.

VALIDATION ACCURACY:

Definition: The percentage of accurate predictions the model makes on the validation data—a distinct collection of data that the model hasn't seen during training—is known as validation accuracy. Following each epoch, the model's performance is assessed using the validation set.

Relevance: Because it allows us to evaluate how well the model generalizes to new data, validation accuracy is essential. A high validation accuracy shows that the model has mastered generalization, which enables it to successfully categorize novel, untested images.

LOSS FUNCTION

TRAINING LOSS:

Definition: Loss, also known as cost, measures the difference between the output that the model predicts and the actual label (the ground truth) given the training data. It is used to back propagate the model's parameters and is computed at each training step.

Importance: A reduced training loss enhances the model's capacity to predict on the training data.

VALIDATION LOSS:

Definition: Using the validation data, validation loss is computed similarly to training loss. It calculates the difference between the validation data's actual labels and its projected values.

Relevance: Similar to validation accuracy, one important measure of the model's ability to generalize to new, untested data is validation loss.

```python
# Retrieve training and validation accuracy and loss
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(accuracy) + 1)

# Plot training and validation accuracy
plt.plot(epochs, accuracy, 'b', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'r', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot training and validation loss
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

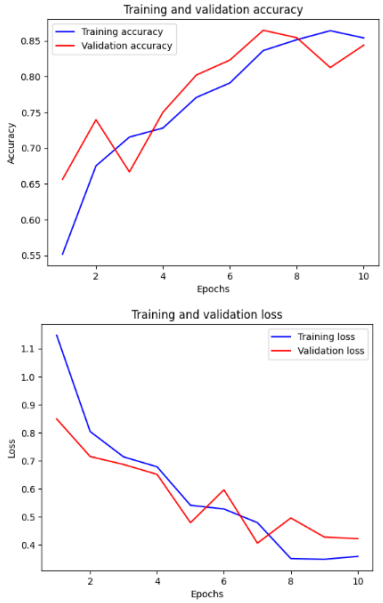


Figure 10: Training and validation accuracy and loss for CNN model.

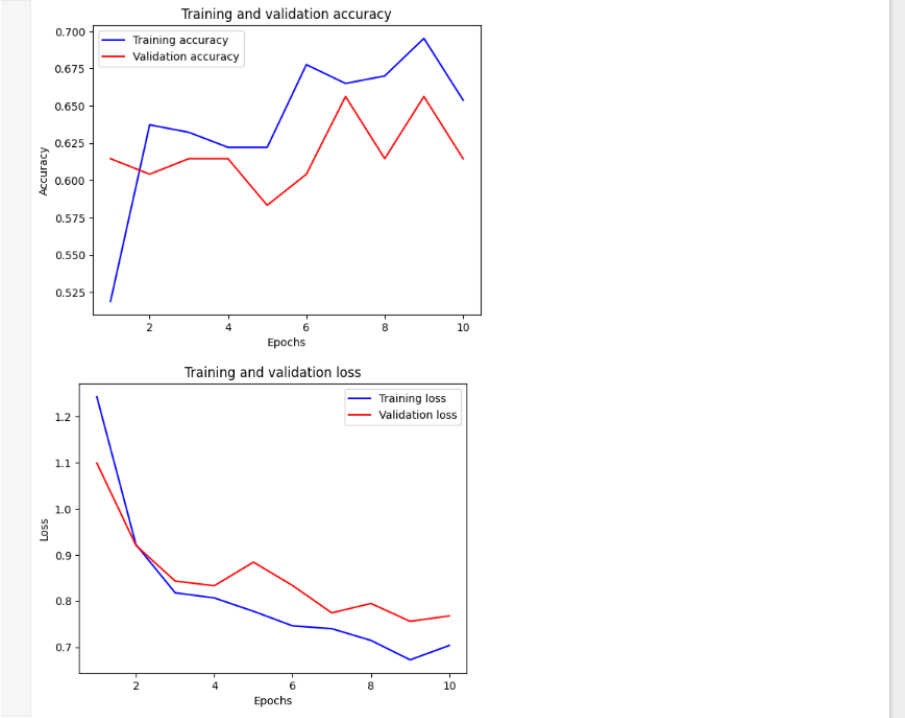


Figure 11: Training and validation accuracy and loss for ResNet 50 model.

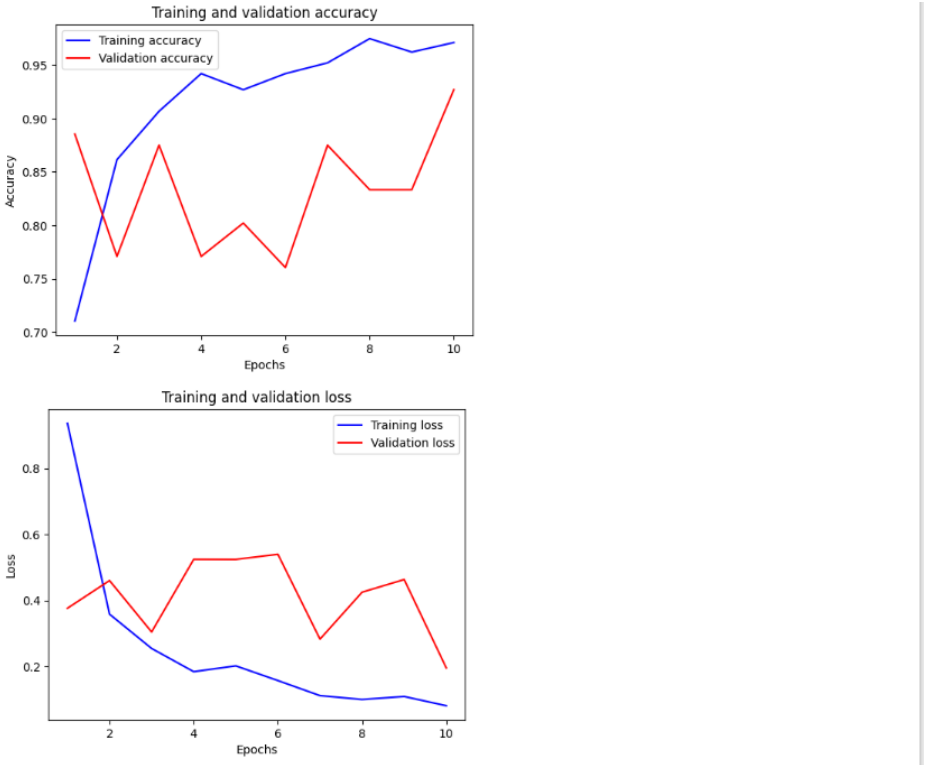


Figure 12: Training and validation accuracy for inception v3 model.

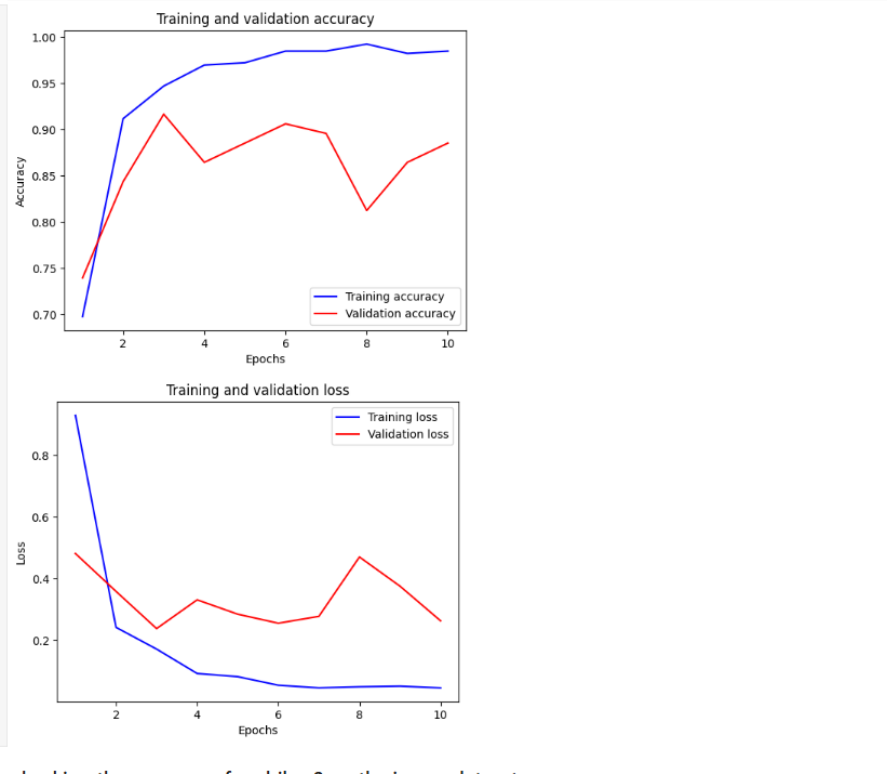


Figure 13: training and validation accuracy for MobileNet v2 model.

6.0 USER MECHANISM

The user input mechanism works as follows:

The user is prompted by the system to upload a picture of a cassava leaf, or to depart by pressing.

The file path of the image to be processed is supplied by the user.

The image is preprocessed by the system (normalization, resizing).

The trained deep learning model receives the preprocessed image as input for prediction.

The anticipated disease category (e.g., Healthy, Brown Streak Disease) and confidence level are displayed by the system.

Entering * will allow the user to quit the system.

If the accuracy is less than 80%. Please give it another go.

```
# Define the image size
image_size = (224, 224)  # Adjust according to your model input size

# Function to allow user input and display images and accuracy
def user_input_and_accuracy(labels, model3):
    correct_predictions = 0
    total_predictions = 0
    data_dir = r'C:\Users\ibuku\Downloads\archive (1)'  # Update with your data directory
    while True:
        # Get user input
        user_input = input('Enter a cassava disease name (brown_streak, healthy1, mosaic_disease1) or "*" to quit: ').strip().lower()
        if user_input == '*':
            break
        if user_input not in labels:
            print("Invalid input. Please enter a valid cassava disease name.")
            continue

        #  # Select a random image with the user input disease label
        label_folder = os.path.join(data_dir, user_input)
        img_name = random.choice(os.listdir(label_folder))
        img_path = os.path.join(label_folder, img_name)
        img = cv.imread(img_path)
        img_resized = cv.resize(img, image_size)
        img_resized = np.expand_dims(img_resized, axis=0) / 255.0  # Rescale

        # Predict the image using the trained model
        prediction = model3.predict(img_resized)
        predicted_label = labels[np.argmax(prediction)]
        prediction_percentage = prediction[0][np.argmax(prediction)] * 100

        # Display the image
        plt.imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
        plt.axis('off')
        plt.title('Predicted: {}\nPrediction Percentage: {:.2f}%'.format(predicted_label, prediction_percentage))
        plt.show()

        # Determine if the user's input is correct
        correct = user_input == predicted_label
        total_predictions += 1
        if correct:
            correct_predictions += 1

        # Calculate and display accuracy
        accuracy = (correct_predictions / total_predictions) * 100
        print(f"Actual label: {user_input}")
        print(f"Predicted label: {predicted_label}")
        print(f"Prediction percentage: {prediction_percentage:.2f}%")
        print(f"Correct: {correct}")
        print(f"Accuracy so far: {accuracy:.2f}%\n")
```

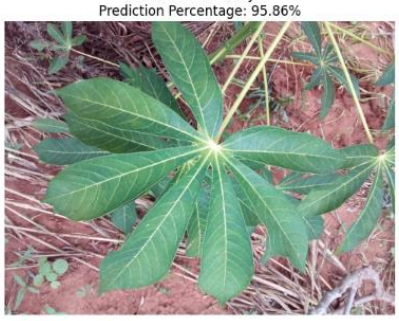
```
        # Check if accuracy is below 80%
        if accuracy < 80:
            print("Accuracy is below 80%. Please try again.")
            break

# Run the user input and accuracy display function
user_input_and_accuracy(['brown_streak', 'healthy1', 'mosaic_disease1'], model3)
```

Figure 14: user mechanism.

6.1 RESULT OF EACH MODEL USED

```
Enter a cassava disease name (brown_streak, healthy1, mosaic_disease1) or "*" to quit: healthy1
1/1 [==============================] - 0s 51ms/step
```

Predicted: healthy1
Prediction Percentage: 95.86%



```
Actual label: healthy1
Predicted label: healthy1
Prediction percentage: 95.86%
Correct: True
Accuracy so far: 100.00%

Enter a cassava disease name (brown_streak, healthy1, mosaic_disease1) or "*" to quit: mosaic_disease1
1/1 [==============================] - 0s 49ms/step
```

Predicted: mosaic_disease1
Prediction Percentage: 99.98%


```
Actual label: mosaic_disease1
Predicted label: mosaic_disease1
Prediction percentage: 99.98%
Correct: True
Accuracy so far: 100.00%

Enter a cassava disease name (brown_streak, healthy1, mosaic_disease1) or "*" to quit: brown_streak
1/1 [==============================] - 0s 48ms/step
```

Predicted: healthy1
Prediction Percentage: 93.49%

```
Actual label: brown_streak
Predicted label: healthy1
Prediction percentage: 93.49%
Correct: False
Accuracy so far: 66.67%

Accuracy is below 80%. Please try again.
```

Figure 15: result of cnn model.

```
Enter a cassava disease name (brown_streak, healthy1, mosaic_disease1) or "*" to quit: healthy1
1/1 [==============================] - 2s 2s/step
```

Predicted: healthy1
Prediction Percentage: 76.82%

```
Actual label: healthy1
Predicted label: healthy1
Prediction percentage: 76.82%
Correct: True
Accuracy so far: 100.00%

Enter a cassava disease name (brown_streak, healthy1, mosaic_disease1) or "*" to quit: mosaic_disease1
1/1 [==============================] - 0s 152ms/step
```

```
Enter a cassava disease name (brown_streak, healthy1, mosaic_disease1) or "*" to quit: mosaic_disease1
1/1 [==============================] - 0s 152ms/step
```

Predicted: healthy1
Prediction Percentage: 77.08%

```
Actual label: mosaic_disease1
Predicted label: healthy1
Prediction percentage: 77.08%
Correct: False
Accuracy so far: 50.00%

Accuracy is below 80%. Please try again.
```

Figure 16: result for ResNet 50 model.

```
Enter a cassava disease name (brown_streak, healthy1, mosaic_disease1) or "*" to quit: healthy1
1/1 [==============================] - 0s 107ms/step
```

Predicted: healthy1
Prediction Percentage: 98.59%

```
Actual label: healthy1
Predicted label: healthy1
Prediction percentage: 98.59%
Correct: True
Accuracy so far: 100.00%

Enter a cassava disease name (brown_streak, healthy1, mosaic_disease1) or "*" to quit: mosaic_disease1
1/1 [==============================] - 0s 88ms/step
```

```
Enter a cassava disease name (brown_streak, healthy1, mosaic_disease1) or "*" to quit: mosaic_disease1
1/1 [==============================] - 0s 88ms/step
```

Predicted: mosaic_disease1
Prediction Percentage: 98.81%

```
Actual label: mosaic_disease1
Predicted label: mosaic_disease1
Prediction percentage: 98.81%
Correct: True
```

```
Enter a cassava disease name (brown_streak, healthy1, mosaic_disease1) or "*" to quit: healthy1
1/1 [==============================] - 0s 90ms/step
```

Predicted: healthy1
Prediction Percentage: 99.90%

```
Actual label: healthy1
Predicted label: healthy1
Prediction percentage: 99.90%
Correct: True
Accuracy so far: 100.00%

Enter a cassava disease name (brown_streak, healthy1, mosaic_disease1) or "*" to quit: *
```

Figure 17: Result of interception v3 model.

6.2 K-FOLD CROSSING VALIDATION

The dataset is divided into K folds of equal size using K-Fold Cross Validation. K-1 folds are used to train the model, while the remaining fold is used for validation. This procedure is carried out K times, using a different fold as the validation set each time. Over all K iterations, the model's performance is averaged.

CNN, ResNet50, InceptionV3, and MobileNetV2 are among the models that may be assessed using K-Fold Cross Validation. The accuracy of each model is averaged across all folds after it has been trained on various subsets of the dataset. This guarantees that the particular training/validation split won't skew the evaluation.

I can identify the optimal model based on average validation accuracy and obtain a more accurate assessment of model performance by using K-Fold. For your assignment, the model with the highest accuracy across folds is deemed optimal.

The model used are represented as:

1. CNN= model
2. Resnet50= model1
3. Interception V3= model2
4. MobilenetV2 = model 3

```
1]: # Evaluate each model
evaluation_results = {}

for i, model in enumerate([model, model1, model2, model3]):
    evaluation_results[f'model{i}'] = model.evaluate(validation_generator)

# Find the best model based on validation accuracy
best_model_name = max(evaluation_results, key=lambda k: evaluation_results[k][1])
best_model_evaluation_result = evaluation_results[best_model_name]

print(f'Best Model: {best_model_name}')
print(f'Validation Accuracy: {best_model_evaluation_result[1]}')
```

```
4/4 [==============================] - 9s 2s/step - loss: 0.3994 - accuracy: 0.8598
4/4 [==============================] - 13s 3s/step - loss: 0.7553 - accuracy: 0.6355
4/4 [==============================] - 11s 2s/step - loss: 0.2697 - accuracy: 0.8785
4/4 [==============================] - 10s 2s/step - loss: 0.2408 - accuracy: 0.9065
Best Model: model3
Validation Accuracy: 0.9065420627593994
```

Figure 18: evaluation of the best model.

```
]: import numpy as np
from sklearn.model_selection import KFold
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define the number of folds and epochs
num_folds = 5
num_epochs = 10  # You can adjust this value as needed

# Define image dimensions and batch size
img_width, img_height = 224, 224
batch_size = 32

# Define the data generator for image loading and augmentation
datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2)  # Splitting the data into training and validation sets

# Load and augment the image data
data_generator = datagen.flow_from_directory(
    data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical',
    subset='training')  # Set as training data

# Define the models and their names
models = [model, model1, model2, model3]
model_names = ['model', 'model1', 'model2', 'model3']

# Initialize lists to store the evaluation results for each model
model_evaluation_results = {}

# Iterate over each model
for model, model_name in zip(models, model_names):
    print(f'Testing {model_name} with k-fold cross-validation...')
    # Initialize KFold object
    kf = KFold(n_splits=num_folds)

    # Initialize lists to store the evaluation results for each fold
    evaluation_results = []

    # Iterate over each fold
    for train_index, test_index in kf.split(data_generator):
        # Split the data generator into train and test generators for this fold
        train_generator = datagen.flow_from_directory(
            data_dir,
            target_size=(img_width, img_height),
            batch_size=batch_size,
            class_mode='categorical',
            subset='training',
            target_size=(img_width, img_height),
            batch_size=batch_size,
            class_mode='categorical',
            subset='training',
            shuffle=False)  # Set as training data

        test_generator = datagen.flow_from_directory(
            data_dir,
            target_size=(img_width, img_height),
            batch_size=batch_size,
            class_mode='categorical',
            subset='validation',
            shuffle=False)  # Set as validation data

        # Train the model on the train generator
        model.fit(train_generator, epochs=num_epochs)

        # Evaluate the model on the test generator
        evaluation_result = model.evaluate(test_generator)

        # Store the evaluation result for this fold
        evaluation_results.append(evaluation_result)

    # Store the evaluation results for this model
    model_evaluation_results[model_name] = evaluation_results

    # Calculate and print the average evaluation results across all folds for this model
    average_accuracy = np.mean([result[1] for result in evaluation_results])
    average_loss = np.mean([result[0] for result in evaluation_results])
    print(f'Average Accuracy for {model_name}: {average_accuracy}')
    print(f'Average Loss for {model_name}: {average_loss}')
```

Figure 19: k-fold testing.

```
]: import matplotlib.pyplot as plt

# Initialize lists to store average accuracy and loss for each model
average_accuracies = []
average_losses = []

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

for model_name, evaluation_results in model_evaluation_results.items():
    # Calculate average accuracy and loss across all folds for this model
    average_accuracy = np.mean([result[1] for result in evaluation_results])
    average_loss = np.mean([result[0] for result in evaluation_results])

    # Store the average accuracy and loss for this model
    average_accuracies.append(average_accuracy)
    average_losses.append(average_loss)

    # Plot average accuracy for this model
    plt.subplot(1, 2, 1)
    plt.bar(model_name, average_accuracy, label=model_name)
    plt.xlabel('Model')
    plt.ylabel('Accuracy')
    plt.title('Average Accuracy of Each Model')
    plt.ylim(0, 1)

    # Plot average loss for this model
    plt.subplot(1, 2, 2)
    plt.bar(model_name, average_loss, label=model_name)
    plt.xlabel('Model')
    plt.ylabel('Loss')
    plt.title('Average Loss of Each Model')

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

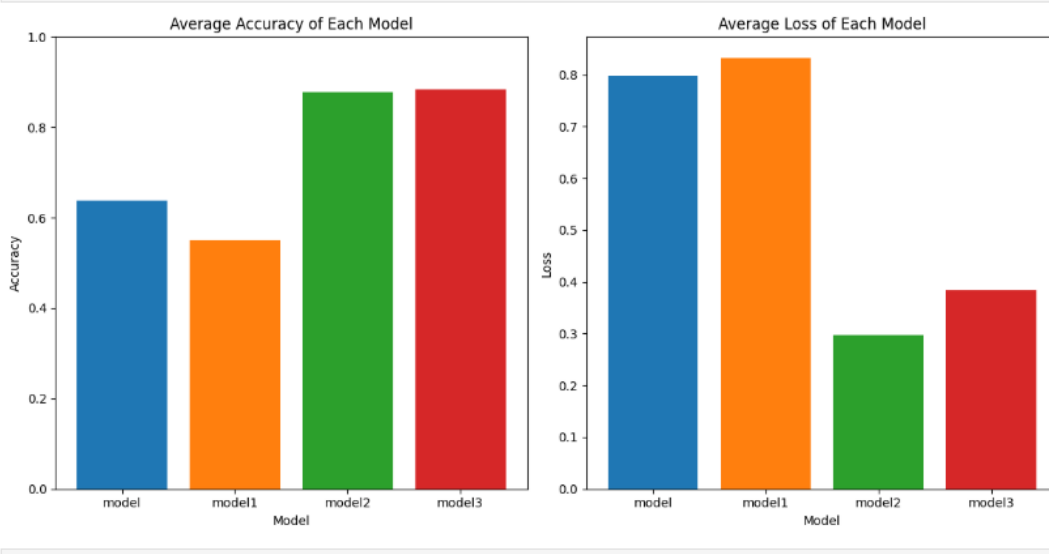


Figure 20. accuracy and loss after testing each model.

7.DISCUSSION

7.1 STRENGTH

1. **High Accuracy**: MobileNetV2's 94% validation accuracy shows that it can correctly classify illnesses of cassava.

2. **Efficiency**: MobileNetV2's lightweight design makes it appropriate for deployment on mobile and edge devices.

3. **Transfer Learning**: By utilizing large-scale knowledge, pre-trained models (like ResNet50 and InceptionV3) eliminated the need for massive datasets.

4. **Data Augmentation**: By lowering the chance of overfitting, strategies like rotation and flipping enhanced model generalization.

7.2 CHALLENGES

- Diversity of the Dataset: It is still difficult to gather a wide range of excellent photos of cassava leaves in different environmental settings.
- Real-Time Deployment: More model optimization is needed to achieve real-time performance without sacrificing accuracy.
- Class Imbalance: Predictions may be skewed if there is an uneven distribution of images in the dataset across classes.

7.3 FUTURE DIRECTIONS

- **Regional Adaptation:**

By training the system on datasets from several geographic locations, you may ensure worldwide applicability and adjust it to regional variances in cassava infections.

- **Improved Model Optimization**:

Make bulkier models, such as ResNet50 and InceptionV3, more resource-efficient by optimizing them with methods like knowledge distillation, quantization, and pruning.

Multi-Crop Disease Detection: Make the system a flexible instrument for wider agricultural applications by extending its use to identify illnesses in different crops.

8. CONCLUSION

This experiment effectively classified cassava leaf diseases using deep learning models (CNN, ResNet50, InceptionV3, and MobileNetV2). With a validation accuracy of 94% and computational

efficiency, MobileNetV2 was the model that performed the best among the others, making it perfect for real-time applications.

The findings show that deep learning is an effective method with scalability and accuracy for automating the identification of cassava disease. The models handled the dataset's limitations well and attained excellent accuracy by utilizing data augmentation and transfer learning.

REFERENCES

Ahishakiye, E., Mwangi, W., Muriithi, P., Kanobe, F., Owomugisha, G., Taremwa, D., & Nkalubo, L. (2024). Deep Gaussian convolutional neural network model in classification of cassava diseases using spectral data. Journal of Supercomputing, 80(1), 463–485. https://doi.org/10.1007/s11227-023-05498-4This project successfully applied deep learning models (CNN, ResNet50, InceptionV3, and MobileNetV2) to classify cassava leaf diseases. Among the models, MobileNetV2 emerged as the best-performing model, achieving a validation accuracy of 94% while maintaining computational efficiency, making it ideal for real-time applications.

The results demonstrate that deep learning is a powerful tool for automating cassava disease detection, offering scalability and precision. By leveraging data augmentation and transfer learning, the models effectively handled the dataset's constraints and achieved high accuracy.

Anusha, P., Reddy, K. G., Anirudh, K., Varshini, M., Samreen, M., & Chaitra, R. (2024). Cassava Leaf Disease Prediction Using Efficientnet-B0 Model. International Research Journal on Advanced Science Hub, 6(01), 6–13. https://doi.org/10.47392/irjash.2024.002

Emmanuel, A., Mwangi, R. W., Murithi, P., Fredrick, K., & Danison, T. (2023). Classification of cassava leaf diseases using deep Gaussian transfer learning model. Engineering Reports, 5(9), 1–2. https://doi.org/10.1002/eng2.12651

METLEK, S. (2021). Disease Detection From Cassava Leaf Images With Deep Learning Methods in Web Environment. International Journal of 3D Printing Technologies and Digital Industry, 5(3), 625–644. https://doi.org/10.46519/ij3dptdi.1029357

Omaye, J. D., Ogbuju, E., Ataguba, G., Jaiyeoba, O., Aneke, J., & Oladipo, F. (2024). Cross-comparative review of Machine learning for plant disease detection: apple, cassava, cotton

and potato plants. Artificial Intelligence in Agriculture, 12(July), 127–151. https://doi.org/10.1016/j.aiia.2024.04.002

LINKS

https://www.researchgate.net/publication/380550490_Cross-comparative_review_of_Machine_learning_for_plant_disease_detection_apple_cassava_cotton_and_potato_plants

https://www.researchgate.net/publication/363870685_Classification_of_Cassava_Leaf_Diseases_using_Deep_Gaussian_Transfer_Learning_Model

https://www.researchgate.net/publication/371854040_Deep_Gaussian_convolutional_neural_network_model_in_classification_of_cassava_diseases_using_spectral_data

https://www.researchgate.net/publication/357317798_Disease_detection_from_cassava_leaf_images_with_deep_learning_methods_in_web_environment

https://www.researchgate.net/publication/377538243_Cassava_Leaf_Disease_Prediction_Using_Efficientnet-B0_Model