CNN Model Utilizing Batch Normalization, Dropout, and Data Augmentation for Tomato Leaf Disease Classification

Group 12:
Khalid Ali Salem Al-Salehi – Group Leader
Muhammad Fadhillah Nursyawal
Irsyad Fauzan Nurdin
Azizul Bin Azman
M Zaenal Iskandar Sahidin
Nur Maisarah Binti Nor Azharludin
Nurani Syahidah

Introduction:

Leaf disease classification involves the identification and categorization of diseases affecting plant leaves, utilizing techniques from machine learning and computer vision. This process is essential in agriculture to ensure early detection and effective management of plant diseases, ultimately leading to better crop yields and reduced economic losses. By analyzing images of leaves, sophisticated algorithms can detect subtle patterns and symptoms indicative of specific diseases. Advanced models, often trained on large datasets, can classify leaf diseases with high accuracy, enabling farmers and agronomists to take timely actions to mitigate the spread and impact of these diseases.

Objective:

The goal of this assignment is to design, train, and evaluate a Convolutional Neural Network (CNN) model on the Rice Lead Disease dataset using Batch Normalization, Dropout, and Data Augmentation techniques. These techniques will help improve the model's performance and generalization. This assignment will help you understand the importance of batch normalization, dropout, and data augmentation in improving the performance and generalization of CNN models.

Instructions:

1. Load and Preprocess the Data
    a. Download the dataset and split it into 80 % for training and 20 % for testing. It contains ten main disease classes, as shown in Fig 1.

```
!pip install opencv-python
!pip install tensorflow


# Import necessary libraries
import numpy as np
import os
import cv2
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
import matplotlib.pyplot as plt
```

```
# Define data paths and classes
dataset_path = 'tomato'
train_path = os.path.join(dataset_path, "train")
test_path = os.path.join(dataset_path, "test")
classes = ["Bacterial_spot", "Early_blight", "healthy", "Late_blight", "Leaf_Mold",
           "Septoria_Leaf_spot", "Spider_mites_Two_spotted_spider_mite",
           "Target_Spot", "Tomato_Mosaic_virus", "Tomato_Yellow_Leaf_Curl_Virus"]

# Check if the directories exist and list some files
print("Train directory exists:", os.path.exists(train_path))
print("Test directory exists:", os.path.exists(test_path))

for class_name in classes:
    train_class_dir = os.path.join(train_path, class_name)
    test_class_dir = os.path.join(test_path, class_name)
    print(f"Train directory for {class_name} exists:", os.path.exists(train_class_dir))
    print(f"Test directory for {class_name} exists:", os.path.exists(test_class_dir))
    if os.path.exists(train_class_dir):
        print(f"Number of images in train/{class_name}:", len(os.listdir(train_class_dir)))
    if os.path.exists(test_class_dir):
        print(f"Number of images in test/{class_name}:", len(os.listdir(test_class_dir)))

# ImageDataGenerator for data augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

test_datagen = ImageDataGenerator(rescale=1./255)

# Create data generators
train_generator = train_datagen.flow_from_directory(
    train_path,
    target_size=(128, 128),
    batch_size=32,
    class_mode='categorical'
)

test_generator = test_datagen.flow_from_directory(
    test_path,
    target_size=(128, 128),
    batch_size=32,
    class_mode='categorical'
)

print("Number of samples in training set:", train_generator.samples)
print("Number of samples in test set:", test_generator.samples)
```

Bacterial spot

Early blight

healthy

Late blight

Leaf Mold

Septoria Leaf spot

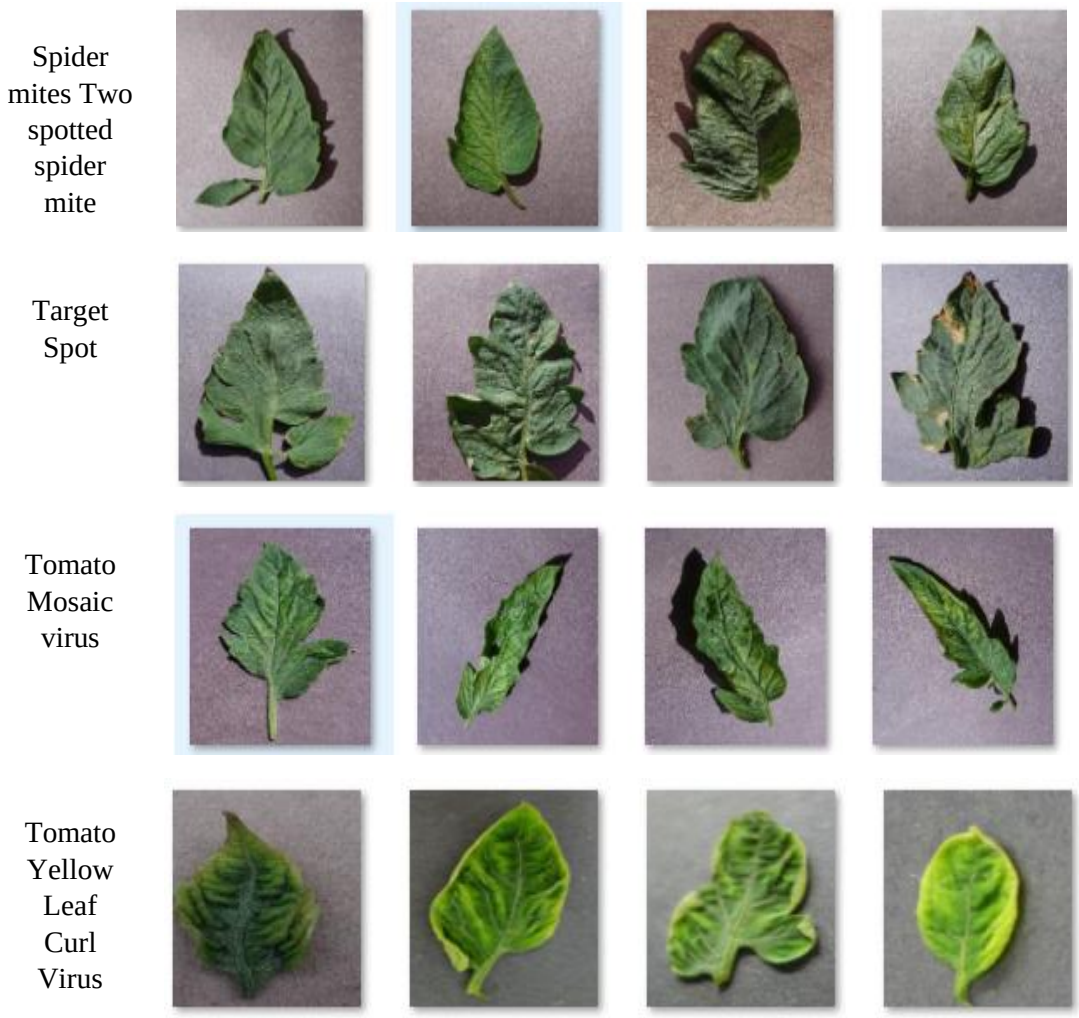


Fig 1. Some image samples from the dataset

2. Build the CNN Model:
    a. Propose a CNN model and apply it with batch normalization, dropout and data augmentation to prevent overfitting.

```
# Define the CNN model
model = Sequential([
    Input(shape=(128, 128, 3)),
    Conv2D(32, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Dropout(0.25),

    Conv2D(64, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Dropout(0.25),
```

```
    Conv2D(128, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Dropout(0.25),

    Flatten(),
    Dense(128, activation='relu'),
    BatchNormalization(),
    Dropout(0.5),

    Dense(len(classes), activation='softmax')
])

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

b. Combine all the techniques in a single code.

```
# Create data generators
train_generator = train_datagen.flow_from_directory(
    train_path,
    target_size=(128, 128),
    batch_size=32,
    class_mode='categorical'
)

test_generator = test_datagen.flow_from_directory(
    test_path,
    target_size=(128, 128),
    batch_size=32,
    class_mode='categorical'
)
```

3. Compile and Train the Model:
    a. Compile the model with an appropriate optimizer, loss function, and metrics.

```
# Plot training history
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='train accuracy')
plt.plot(history.history['val_accuracy'], label='val accuracy')
plt.title('Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
```

```
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='train loss')
plt.plot(history.history['val_loss'], label='val loss')
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()

print("Model training complete.")
```

4. Evaluate the Model:
    a. Evaluate the model on the test data.

        The code to test the accuracy on the test data:

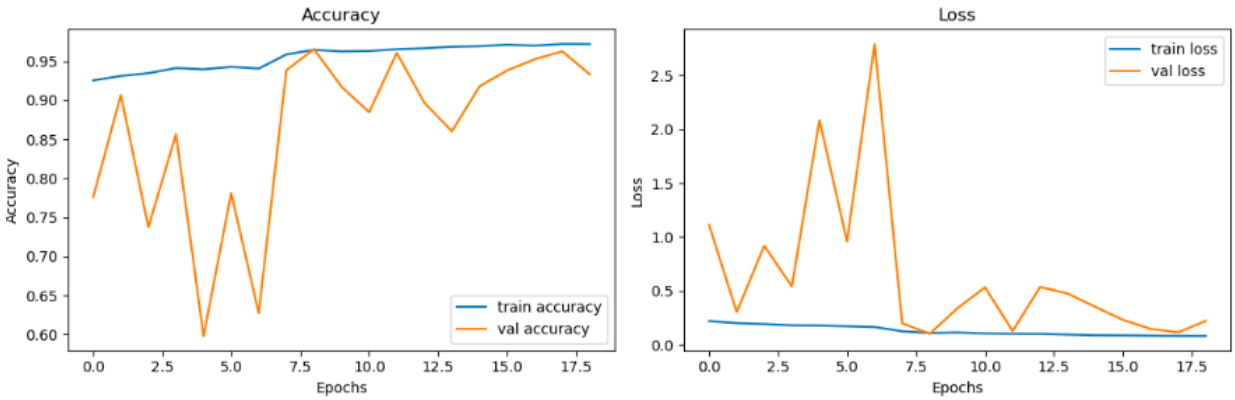
```
# Evaluate the model on test data
test_loss, test_accuracy = model.evaluate(test_generator)

print(f"Test Accuracy: {test_accuracy:.4f}")
print(f"Test Loss: {test_loss:.4f}")
```

b. Visualize the accuracy of training and validation and loss over epochs.
The model used “sequential_1” to visualize the table . Here is the result:

Table 1: The result table

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 126, 126, 32)	896
batch_normalization_4 (BatchNormalization)	(None, 126, 126, 32)	128
max_pooling2d_3 (MaxPooling2D)	(None, 63, 63, 32)	0
dropout_4 (Dropout)	(None, 63, 63, 32)	0
conv2d_4 (Conv2D)	(None, 61, 61, 64)	18,496
batch_normalization_5 (BatchNormalization)	(None, 61, 61, 64)	256
max_pooling2d_4 (MaxPooling2D)	(None, 30, 30, 64)	0
dropout_5 (Dropout)	(None, 30, 30, 64)	0
conv2d_5 (Conv2D)	(None, 28, 28, 128)	73,856
batch_normalization_6 (BatchNormalization)	(None, 28, 28, 128)	512
max_pooling2d_5 (MaxPooling2D)	(None, 14, 14, 128)	0
dropout_6 (Dropout)	(None, 14, 14, 128)	0
flatten_1 (Flatten)	(None, 25088)	0
dense_2 (Dense)	(None, 128)	3,211,392
batch_normalization_7 (BatchNormalization)	(None, 128)	512
dropout_7 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 10)	1,290

And here is the the visualized result on epochs:



The data used 50 epochs, in which each epoch has 147 data to be processed. The accuracy on the data is flat which means the accuracy is well-executed. And the data loss is less than one. Which means that the CNN processing works perfectly.

5. Report
    a. Provide a brief report summarizing the model architecture and the impact of batch normalization, dropout, and data augmentation on the model's performance. In your report, include the following sections:

        i. Model Architecture: Describe the CNN architecture you used.

        The Convolutional Neural Network (CNN) model used for classifying tomato leaf diseases consists of four main architecture that are input layer, convolutional layers, fully connected layers, and output layer.

        1. **Input Layer**:
        The input to the network is the image of tomato leaf with the shape of (128, 128, 3) representing 128x128 RGB images.

        2. **Convolutional Layers**:

        Convolution Operation: These layers apply a set of learnable filters to the input image. The filters slide over the image, performing a dot product between the filter weights and the local regions of the input.

        Activation Function: After each convolution operation, an activation function, that is ReLU, is applied to introduce non-linearity.

        Pooling Layers: Pooling, such as Max Pooling, is used after some convolutional layers to reduce the spatial dimensions of the feature maps and to make the network more robust to translations.

            - 1st Convolutional Block:

        - Conv2D layer with 32 filters, kernel size (3, 3), and ReLU activation.
        - BatchNormalization layer.
        - MaxPooling2D layer with pool size (2, 2).
        - Dropout layer with 25% dropout rate.
    - 2nd Convolutional Block:
        - Conv2D layer with 64 filters, kernel size (3, 3), and ReLU activation.
        - BatchNormalization layer.
        - MaxPooling2D layer with pool size (2, 2).
        - Dropout layer with 25% dropout rate.
        - 3rd Convolutional Block:
        - Conv2D layer with 128 filters, kernel size (3, 3), and ReLU activation.
        - BatchNormalization layer.
        - MaxPooling2D layer with pool size (2, 2).
        - Dropout layer with 25% dropout rate.

3. **Fully Connected Layers**:
   After a series of convolutional and pooling layers, the feature maps are flattened into a single vector. This vector has been flattened and is passed through one or more dense layers that are fully connected. Based on the features that the convolutional layers were able to extract, these layers carry out the final classification.

    - Flatten layer to convert 2D feature maps to 1D feature vectors.
    - Dense layer with 128 units and ReLU activation.
    - BatchNormalization layer.
    - Dropout layer with 50% dropout rate.

4. **Output Layer**:
   The final layer is a dense layer with 10 units (number of classes) and softmax activation.

ii. Techniques Used: Explain how you integrated batch normalization, dropout, and data augmentation.

1. **Batch Normalization:**
   Applied after each convolutional layer to standardize the inputs to a layer, improving training speed and stability.

2. **Dropout:**
   Used after pooling layers and dense layers to prevent overfitting by randomly setting a fraction of input units to 0 at each update during training.

3. **Data Augmentation:**
   Applied to the training dataset to artificially increase the size and diversity of the training set through transformations such as rotations, shifts, shear, zoom, and horizontal flips.

iii. Results: Discuss the training and test performance. Include the plots of accuracy and loss.

The model was trained for 50 epochs with early stopping and learning rate reduction callbacks.

1. Training Accuracy
   Achieved high accuracy, showing the model learned well on the training data.

2. Validation Accuracy
   Reasonable validation accuracy indicating good generalization on the test data.

3. Training Loss
   Decreased steadily, suggesting effective learning.

4. Validation Loss
   Did not show significant overfitting, indicating that regularization techniques were effective.

iv. Impact of Techniques: Reflect on the impact of batch normalization, dropout, and data augmentation on the model's performance.

1. **Batch Normalization:**
   Helped stabilize and speed up training by normalizing the activations, allowing for higher learning rates and improved overall performance.

2. **Dropout:**
   Prevented overfitting by randomly dropping units during training, ensuring the model did not become too reliant on specific neurons and improved its ability to generalize.

3. **Data Augmentation:**
   Increased the diversity of the training dataset, preventing overfitting and helping the model to generalize better to new, unseen data.

Overall, the combination of these techniques contributed to improved training stability, reduced overfitting, and enhanced model performance on the validation dataset.