**TAITA TAVETA UNIVERSITY**

**HBT: 2403**

**SYSTEM PROJECT: PROJECT REPORT**

**GROUP 1**

GROUP MEMBERS INCLUDE

**NAME REG. NO**

Michael Otieno Omondi TU01-BE213-0406/2017

Augustine Kotut TU01-BE213-0405/2016

Joseph Njuguna TU01-BE213-0397/2017

Rose Ndinda TU01-BE213-0366/2016

George Wanderi TU01-BE213-0384/2017

**PLANT LEAF DISEASE DETECTION USING IMAGE CLASSIFICATION**

**Step 1: We imported the necessary Libraries.**

This include:

Tensorflow library which is used to generate models, supplement data, and so on. It's famous for being a part of a deep learning library. (G Kuikel, 2020) It comes with the Keras API, which we'll utilize to build our model. We also added the OS library, which is used to extract a directory's path from the user's path on the machine. We also imported the Numpy library, which is used to return images in the same shape as arr, which sorts the array. ImageDataGenerator is used to augment data. Matplotlib is used to plot the pictures. For our optimizer we opted to use Adam. “It is a replacement optimization algorithm for stochastic gradient desent for training deap learning models.”

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from keras.preprocessing import image

from keras.models import load\_model

from tensorflow.keras.optimizers import Adam

import tensorflow as tf

from matplotlib import pyplot as plt

import cv2

import os

import numpy as np

#### Step 2: Prepare Dataset for Training

Crop disease-related photos are included in this dataset. Crop diseases are divided into three categories and organized into three folders (according to disease type). Aphid, Powdery mildew, and plant parasite Spider mites are the disease kinds. The dataset is separated into three sections: training, validation, and testing.

We'll assign routes and create categories (labels) after splitting the dataset for training and resize our image to 100,100 pixels. When we train the model, we must then reshape our dataset inputs to the structure that our model anticipates. Rescale 1/255 is used to transform every pixel value in the range of (0,255) - because 255 is the maximum pixel value (0,1).  
The ImageDataGenerator function extracts images from a large NumPy array and image directories.

In [2]: train = ImageDataGenerator(rescale=1/255)

validation = ImageDataGenerator(rescale=1/255)

The train. flow\_from\_directory() is a method of ImageDataGenerator class that reads the images from a big NumPy array. There is need of parameter “target\_ size” to make all images of same shape, also we need to specify the batch size with “batch\_size = 0”

In [3]: train\_dataset = train.flow\_from\_directory(r"C:\Users\HP\Desktop\pictures\plantdisease\_dataset\train",

target\_size= (100,100),

batch\_size= 20,

class\_mode= "categorical")

validation\_dataset = train.flow\_from\_directory(r"C:\Users\HP\Desktop\pictures\plantdisease\_dataset\validation",

target\_size= (100,100),

batch\_size= 20,

class\_mode= "categorical")

Out [3]:

Found 4368 images belonging to 3 classes.

Found 3737 images belonging to 3 classes.

Our target variable must be one-hot-encoded. This means that for each output category, a column will be generated and a binary variable will be entered. To do so, we divide the label into three integers, each of which indicates a class.

In [4]: train\_dataset.class\_indices

Out [4]: 'aphids': 0, 'powdery mildew': 1, 'spider mites': 2}

**Step 3: Build the Model**

#### To begin, we must first create the convolutional layers. To allow the network to learn key features, you apply various filters. You choose the kernel size and the number of filters.

The architecture of our CNN model:

1. Conv2D layer – we will add 3 convolutional layers of 3\*3 filter size, and activation function as ReLu.
2. Max Pooling – MaxPool2D with 2\*2 polling size and stride 2.
3. 2nd Convolutional layer of 3\*3 filter size and stride 2
4. Max Pooling – MaxPool2D with 2\*2 polling size and stride 2.
5. 3nd Convolutional layer of 3\*3 filter size and stride 2
6. Max Pooling – MaxPool2D with 2\*2 polling size and stride 2.
7. One Flatten layer
8. Dense, feed-forward neural network (activation =” ReLu”)
9. Two Dense layer (activation =” softmax”)

Maximum pooling layer (MaxPool2D) is used to minimize the size of images.  
Flattening is the process of transforming a dataset into a one-dimensional array for use in the next layer, which is the fully connected layer. Dense layer – aids in determining which category an image belongs to in a feed-forward neural network. For multi-class classification, the final layer will use a softmax activation function.

model = tf.keras.models.Sequential([ tf.keras.layers.Conv2D(16,(3,3),activation = 'relu',input\_shape =(100,100,3)),

tf.keras.layers.MaxPool2D(2,2),

#

tf.keras.layers.Conv2D(32,(3,3),activation = 'relu'),

tf.keras.layers.MaxPool2D(2,2),

#

tf.keras.layers.Conv2D(64,(3,3),activation = 'relu'),

tf.keras.layers.MaxPool2D(2,2),

#

tf.keras.layers.Conv2D(128,(3,3),activation = 'relu'),

tf.keras.layers.MaxPool2D(2,2),

##

tf.keras.layers.Flatten(),

##

tf.keras.layers.Dense(128,activation= 'relu'),

##

tf.keras.layers.Dense(3,activation= 'softmax')

])

**Step 4: Compile and Train the Model**

We must first configure the learning process, which is done using the compile technique, before we can begin the training process. It is addressed with three arguments:  
A loss function, which is the objective that the model will attempt to minimize. We utilize categorical\_crossntropy as a string identifier for an existing loss function.  
Adamis our string identifier, and we use an optimizer with a learning rate of 0.001.  
We use 'accuracy' as a metric for any categorization problem.

In [6]: model.compile(loss= 'categorical\_crossentropy',

optimizer = Adam(lr=0.001),

metrics =['accuracy'])

For training we use the model.fit. Epoch is the number of times the model will be trained. We used 20 epochs and 40 step\_per\_epoch which define the total number of steps(batches of samples) to yield the generator before declaring one epoch finished and starting the next epoch.

model\_fit = model.fit(train\_dataset,

step\_per\_epoch = 40,

epochs = 20,

validation\_data = validation\_dataset)

We got a 99 percent accuracy after training and saved our model as (leafdiseasedetection) using model. save.

Epoch 20/20

187/187 [==============================] - 39s 210ms/step - loss: 0.0431 - acc: 0.9930

219/219 [==============================] - 130s 596ms/step - loss: 0.0370 - acc: 0.9876 - val\_loss: 0.0431 - val\_acc: 0.9930

In [8]: model.save('leafdiseasedetection')

## **Step 6: Make Prediction**

First, we import the necessary libralies then we import our model using the load\_model function. To see predictions that our model has made for the test data, we use the predict function. The predict function will give an array with 3 numbers. These numbers are the probabilities that the input image represents each digit (0–2). If the number is 0 the image will be classified as a APHIDS, 1 for POWDERY MILDEW and 2 for SPIDER MITES. We used cv2 to load a sample Leaf with aphids from our leaf disease detect folder, then resized it to 100 by 100 and used NumPy to reshape the image so as to feed it into the model. The predict function predicts the image as a certificate

In [9]:from tensorflow.keras.models import load\_model

from keras.preprocessing import image

import cv2

import numpy as np

In [10]: model = load\_model('leafdiseasedetection')

In [11]: img = cv2.imread(r"C:\Users\HP\Desktop\pictures\plantdisease\_dataset\test\image7.jpg") #image path

img = cv2.resize(img, (100,100)) #resizing the image into

img = np.reshape(img,[1,100,100,3])

In [12]:classes = model.predict\_classes(img)

if classes == 0:

print("0: APHIDS")

elif classes == 1:

print("1: POWDERY\_MILDEW")

else:

print("2: SPIDER\_MITES")

Out [12]:

0: APHIDS