

ARTIFICIAL INTELLIGENCE PROJECT REPORT.

PLANT DISEASE DETECTION.

**Group members:**

Zawat Masta (1912330)

Umer Ali Usmani (1912327)

Danish Aslam sheikh (1912300)

Syed Yousuf Fatmi (1912325)

**INTRODUCTION:**

Getting impacted by a disease is exceptionally normal in plants because of different factors, for example, composts, social practices followed, ecological circumstances, and so on. These sicknesses hurt horticultural yield and in the end the economy in view of it.

Any procedure or strategy to conquer this issue and getting a warning before the plants are harmed would help Farmers to proficiently develop yields or plants, both subjectively and quantitatively. Hence, disease discovery in plants assumes a vital part in farming.

**DATA SET:**

We use a publicly available and quite famous, the Plant Village Dataset. The dataset was published by crowd AI during the "Plant Village Disease Classification Challenge".

The dataset consists of about 54,305 images of plant leaves collected under controlled environmental conditions. The plant images span the following 14 species:

Apple, Blueberry, Cherry, Corn, Grape, Orange, Peach, Bell Pepper, Potato, Raspberry, Soybean, Squash, Strawberry, and Tomato.

The dataset contains a total of 38 classes of plant disease and 1 class of background images listed below:

Apple Scab

Apple Black Rot

Apple Cedar Rust

Apple healthy

Blueberry healthy

Cherry healthy

Cherry Powdery Mildew

Corn Gray Leaf Spot

Corn Common Rust

Corn healthy

Corn Northern Leaf Blight

Grape Black Rot

Grape Black Measles

Grape Leaf Blight

Grape healthy

Orange Huanglongbing

Peach Bacterial Spot

Peach healthy

Bell Pepper Bacterial Spot

Bell Pepper healthy

Potato Early Blight

Potato healthy

Potato Late Blight

Raspberry healthy

Soybean healthy

Squash Powdery Mildew

Strawberry Healthy

Strawberry Leaf Scorch

Tomato Bacterial Spot

Tomato Early Blight

Tomato Late Blight

Tomato Leaf Mold

Tomato Septoria Leaf Spot

Tomato Two Spotted Spider Mite

Tomato Target Spot

Tomato Mosaic Virus

Tomato Yellow Leaf Curl Virus

Tomato healthy

**RESEARCH PAPER (ARTICLE):**

Plant diseases and pests detection is a very important research content in the field of machine vision. It is a technology that uses machine vision equipment to acquire images to judge whether there are diseases and pests in the collected plant images.

**OBJECTIVES:**

Thus the main objectives are:

1) To design such system that can detect crop disease and pest accurately.

2) Create database of insecticides for respective pest and disease.

3) To provide remedy for the disease that is detected.

Human society needs to increase food production by an estimated 70% by 2050 to feed an expected population size that is predicted to be over 9 billion people. Currently, infectious diseases reduce the potential yield by an average of 40% with many farmers in the developing world experiencing yield losses as high as 100%. The widespread distribution of smartphones among crop growers around the world with an expected 5 billion smartphones by 2020 offers the potential of turning the smartphone into a valuable tool for diverse communities growing food. One potential application is the development of mobile disease diagnostics through machine learning and crowdsourcing.

We describe both the data and the models. These data are the beginning of an on-going, crowdsourcing effort to enable computer vision approaches to help solve the problem of yield losses in crop plants due to infectious diseases.

We use to create by these implementations:

* Introduction
* Dataset
* Libraries
* Data Preprocessing
* Data Augmentation
* Model
* Training
* Evaluation
* Testing
* Reuse

1. [**NumPy**](http://www.numpy.org/)**:** A library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
2. [**Pickle**](https://docs.python.org/3/library/pickle.html)**:** Any object in Python can be pickled so that it can be saved on disk. Pickling is a way to convert a python object (list, dict, etc.) into a character stream. The idea is that this character stream contains all the information necessary to reconstruct the object in another python script.
3. [**Cv2 (OpenCV)**](https://pypi.org/project/opencv-python/)**:**OpenCV is a library of bindings designed to solve computer vision problems.
4. [**Os**](https://docs.python.org/3/library/os.html)**:** The OS module in Python provides functions for creating and removing a directory (folder), fetching its contents, changing and identifying the current directory, etc. It is also possible to automatically perform many operating system tasks.
5. [**Sklearn**](https://scikit-learn.org/stable/)**:** A free software machine learning library for the Python programming language. It features various [classification](https://en.wikipedia.org/wiki/Statistical_classification), [regression](https://en.wikipedia.org/wiki/Regression_analysis) and [clustering](https://en.wikipedia.org/wiki/Cluster_analysis) algorithms including [support vector machines](https://en.wikipedia.org/wiki/Support_vector_machine), [random forests](https://en.wikipedia.org/wiki/Random_forests), [gradient boosting](https://en.wikipedia.org/wiki/Gradient_boosting), [*k*-means](https://en.wikipedia.org/wiki/K-means_clustering), and [DBSCAN](https://en.wikipedia.org/wiki/DBSCAN), and is designed to interoperate with the Python numerical and scientific libraries [NumPy](https://en.wikipedia.org/wiki/NumPy) and [SciPy](https://en.wikipedia.org/wiki/SciPy" \t "_blank).
6. [**Keras**](https://keras.io/)**:** Keras is an open-source neural network library written in Python. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible.
7. [**Matplotlib**](https://matplotlib.org/)**:** A plotting library for the Python programming language and its numerical mathematics extension.

Due to the limited computational power, it is difficult to train the classification model locally on a majority of normal machines. Therefore, we use the processing power offered by vs code as it connects us to a free instance quickly and effortlessly.

**ALGORITHM:**

**PSUEDOCODE:**

* LIBRARIES:

We import all the necessary libraries required to process the data and build the classification model.

import numpy as np

import pickle

import cv2

import os

from zipfile import ZipFile

import gdown

import tensorflow

import requests

import matplotlib.pyplot as plt

from os import listdir

from os import path

from sklearn.preprocessing import LabelBinarizer

from keras.models import Sequential

# from tensorflow import BatchNormalization

from keras.layers import BatchNormalization

from keras.layers import Conv2D

from keras.layers import MaxPooling2D

from keras.layers import Activation, Flatten, Dropout, Dense

from keras import backend as K

from keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.optimizers import Adam

from keras.preprocessing import image

from tensorflow.keras.utils import img\_to\_array

from sklearn.preprocessing import MultiLabelBinarizer

from sklearn.model\_selection import train\_test\_split

from scipy.sparse import extract

# Download a file based on its file ID.

if path.exists('PlantVillage') == False:

if path.exists('PlantVillage.zip') == False:

# if path.exists('PlantVillage.zip') == False:

URL = "https://drive.google.com/uc?id=18DbC6Xj4NP-hLzI14WuMaAEyq482vNfn"

output= "PlantVillage.zip"

gdown.download(URL,output,quiet=False)

file\_name = "PlantVillage.zip"

with ZipFile('PlantVillage.zip', 'r') as zf:

# printing all the contents of the zip file

zf.extractall()

else:

with ZipFile('PlantVillage.zip', 'r') as zf:

# printing all the contents of the zip file

zf.extractall()

* LOAD DATASET:

Initializing a few parameters required for the image dataset processing.

We define a couple of variables required to perform operations on the dataset images.

# Dimension of resized image

DEFAULT\_IMAGE\_SIZE = tuple((256, 256))

# Number of images used to train the model

N\_IMAGES = 100

# Path to the dataset folder

root\_dir = './PlantVillage'

train\_dir = os.path.join(root\_dir, 'train')

val\_dir = os.path.join(root\_dir, 'val')

We use the function `convert\_image\_to\_array` to resize an image to the size `DEFAULT\_IMAGE\_SIZE` we defined above.

def convert\_image\_to\_array(image\_dir):

try:

image = cv2.imread(image\_dir)

if image is not None:

image = cv2.resize(image, DEFAULT\_IMAGE\_SIZE)

return img\_to\_array(image)

else:

return np.array([])

except Exception as e:

print(f"Error : {e}")

return None

Here, we load the training data images by traversing through all the folders and converting all the images and labels into separate lists respectively.

image\_list, label\_list = [], []

try:

print("[INFO] Loading images ...")

plant\_disease\_folder\_list = listdir(train\_dir)

for plant\_disease\_folder in plant\_disease\_folder\_list:

print(f"[INFO] Processing {plant\_disease\_folder} ...")

plant\_disease\_image\_list = listdir(f"{train\_dir}/{plant\_disease\_folder}/")

for image in plant\_disease\_image\_list[:N\_IMAGES]:

image\_directory = f"{train\_dir}/{plant\_disease\_folder}/{image}"

if image\_directory.endswith(".jpg")==True or image\_directory.endswith(".JPG")==True:

image\_list.append(convert\_image\_to\_array(image\_directory))

label\_list.append(plant\_disease\_folder)

print("[INFO] Image loading completed")

except Exception as e:

print(f"Error : {e}")

# Transform the loaded training image data into numpy array

np\_image\_list = np.array(image\_list, dtype=np.float16) / 225.0

print()

# Check the number of images loaded for training

image\_len = len(image\_list)

print(f"Total number of images: {image\_len}")

OUTPUT:

Output exceeds the size limit. Open the full output data in a text editor

[INFO] Loading images ...

[INFO] Processing Apple\_\_\_Apple\_scab ...

[INFO] Processing Apple\_\_\_Black\_rot ...

[INFO] Processing Apple\_\_\_Cedar\_apple\_rust ...

[INFO] Processing Apple\_\_\_healthy ...

[INFO] Processing background ...

[INFO] Processing Blueberry\_\_\_healthy ...

[INFO] Processing Cherry\_(including\_sour)\_\_\_healthy ...

[INFO] Processing Cherry\_(including\_sour)\_\_\_Powdery\_mildew ...

[INFO] Processing Corn\_(maize)\_\_\_Cercospora\_leaf\_spot Gray\_leaf\_spot ...

[INFO] Processing Corn\_(maize)\_\_\_Common\_rust\_ ...

[INFO] Processing Corn\_(maize)\_\_\_healthy ...

[INFO] Processing Corn\_(maize)\_\_\_Northern\_Leaf\_Blight ...

[INFO] Processing Grape\_\_\_Black\_rot ...

[INFO] Processing Grape\_\_\_Esca\_(Black\_Measles) ...

[INFO] Processing Grape\_\_\_healthy ...

[INFO] Processing Grape\_\_\_Leaf\_blight\_(Isariopsis\_Leaf\_Spot) ...

[INFO] Processing Orange\_\_\_Haunglongbing\_(Citrus\_greening) ...

[INFO] Processing Peach\_\_\_Bacterial\_spot ...

[INFO] Processing Peach\_\_\_healthy ...

[INFO] Processing Pepper,\_bell\_\_\_Bacterial\_spot ...

[INFO] Processing Pepper,\_bell\_\_\_healthy ...

[INFO] Processing Potato\_\_\_Early\_blight ...

[INFO] Processing Potato\_\_\_healthy ...

[INFO] Processing Potato\_\_\_Late\_blight ...

...

[INFO] Processing Tomato\_\_\_Tomato\_Yellow\_Leaf\_Curl\_Virus ...

[INFO] Image loading completed

Total number of images: 3900

After loading the image dataset, we map each label or class of each plant disease to a unique value for the training task. Also, saving this transform to a pickle file will help us later in predicting a label or class of plant disease from the output of the classification model.

Examine the labels/classes in the training dataset.

label\_binarizer = LabelBinarizer()

image\_labels = label\_binarizer.fit\_transform(label\_list)

pickle.dump(label\_binarizer,open('plant\_disease\_label\_transform.pkl', 'wb'))

n\_classes = len(label\_binarizer.classes\_)

print("Total number of classes: ", n\_classes)

Total no of classes: 39.

Finally, we split the loaded image dataset into two sets, namely train and test sets. Train set to train the classification model and test set to validate the model while training.

* AUGMENT AND SPLIT DATASET:

Using `ImageDataGenerator` to augment data by performing various operations on the training images.

The data augmentation technique is used to significantly increase the number of images in a dataset. We perform various operations such as shift, rotation, height, zoom, and flip on the image dataset to diversify our dataset. Providing augmented images to a model helps it efficiently learn features from different areas of the same image and thus perform better on unseen image data.

augment = ImageDataGenerator(rotation\_range=25, width\_shift\_range=0.1,

height\_shift\_range=0.1, shear\_range=0.2,

zoom\_range=0.2, horizontal\_flip=True,

fill\_mode="nearest")

Splitting the data into training and test sets for validation purpose.

print("[INFO] Splitting data to train and test...")

x\_train, x\_test, y\_train, y\_test = train\_test\_split(np\_image\_list, image\_labels, test\_size=0.2, random\_state = 42)

* BUILD MODEL:

Defining the hyper parameters of the plant disease classification model.

EPOCHS = 20 #, 12 workable

STEPS = 100 #97 default

LR = 1e-3

BATCH\_SIZE = 32

WIDTH = 256

HEIGHT = 256

DEPTH = 3Creating a sequential model and adding Convolutional, Normalization, Pooling, Dropout and Activation layers at the appropriate positions.

model = Sequential()

inputShape = (HEIGHT, WIDTH, DEPTH)

chanDim = -1

if K.image\_data\_format() == "channels\_first":

inputShape = (DEPTH, HEIGHT, WIDTH)

chanDim = 1

model.add(Conv2D(32, (3, 3), padding="same",input\_shape=inputShape))

model.add(Activation("relu"))

model.add(BatchNormalization(axis=chanDim))

model.add(MaxPooling2D(pool\_size=(3, 3)))

model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding="same"))

model.add(Activation("relu"))

model.add(BatchNormalization(axis=chanDim))

model.add(Conv2D(64, (3, 3), padding="same"))

model.add(Activation("relu"))

model.add(BatchNormalization(axis=chanDim))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3), padding="same"))

model.add(Activation("relu"))

model.add(BatchNormalization(axis=chanDim))

model.add(Conv2D(128, (3, 3), padding="same"))

model.add(Activation("relu"))

model.add(BatchNormalization(axis=chanDim))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(1024))

model.add(Activation("relu"))

model.add(BatchNormalization())

model.add(Dropout(0.5))

model.add(Dense(n\_classes))

model.add(Activation("softmax"))

model.summary()

OUTPUT:

Output exceeds the size limit. Open the full output data in a text editor

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d (Conv2D) (None, 256, 256, 32) 896

activation (Activation) (None, 256, 256, 32) 0

batch\_normalization (BatchN (None, 256, 256, 32) 128

ormalization)

max\_pooling2d (MaxPooling2D (None, 85, 85, 32) 0

)

dropout (Dropout) (None, 85, 85, 32) 0

conv2d\_1 (Conv2D) (None, 85, 85, 64) 18496

activation\_1 (Activation) (None, 85, 85, 64) 0

batch\_normalization\_1 (Batc (None, 85, 85, 64) 256

hNormalization)

conv2d\_2 (Conv2D) (None, 85, 85, 64) 36928

...

Total params: 58,127,271

Trainable params: 58,124,391

Non-trainable params: 2,880

* TRAIN MODEL:

We initialize Adam optimizer with learning rate and decay parameters.

Also, we choose the type of loss and metrics for the model and compile it for training.

opt = Adam(lr=LR, decay=LR / EPOCHS)

# Compile model

model.compile(loss="binary\_crossentropy", optimizer=opt, metrics=["accuracy"])

# Train model

print(" Training network...")

history = model.fit(augment.flow(x\_train, y\_train, batch\_size=BATCH\_SIZE),

validation\_data=(x\_test, y\_test),

steps\_per\_epoch=len(x\_train) // BATCH\_SIZE,

epochs=EPOCHS,

verbose=1)



* EVALUATE MODEL:

Comparing the accuracy and loss by plotting the graph for training and validation.

Evaluating model accuracy by using the `evaluate` method

import matplotlib.pyplot as plt

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy']

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs = range(1, len(acc) + 1)

# Train and validation accuracy

plt.plot(epochs, acc, 'b', label='Training accurarcy')

plt.plot(epochs, val\_acc, 'r', label='Validation accurarcy')

plt.title('Training and Validation accurarcy')

plt.legend()

plt.figure()

# Train 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.legend()

plt.show()

* SAVE MODEL:

model.save('C:\\Users\\zawat\\OneDrive\\Desktop\\python project\\Model')

* TEST MODEL:

We write the following `predict\_disease` function to predict the class or disease of a plant image.

We just need to provide the complete path to the image and it displays the image along with its prediction class or plant disease.

For testing purposes, we randomly choose images from the dataset and try predicting class or disease of the plant image.

def predict\_disease(image\_path):

image\_array = convert\_image\_to\_array(image\_path)

np\_image = (np.array(image\_array, dtype=np.float16) / 225.0)

np\_image = np.expand\_dims(np\_image,0)

plt.imshow(plt.imread(image\_path))

result = np.argmax(model.predict(np\_image),axis=-1)

print((image\_labels.classes\_[result]))

predict\_disease('C:\\Users\\zawat\\onedrive\\desktop\\python project\\PlantVillage\\val\\Corn\_(maize)\_\_\_Northern\_Leaf\_Blight\\00a14441-7a62-4034-bc40-b196aeab2785\_\_\_RS\_NLB 3932.jpg')

* REUSE MODEL:

model = keras.models.load\_model('C:\\Users\\zawat\\OneDrive\\Desktop\\python project\\Model.pkl')

