Smart Crop Diagnosis: Deep Learning Approach for Agricultural Disease Detection

Prakarsha Malhotra (pm3514), Shruti Mittal (sm10937), Nachiket Khare (nk3559)
New York University
https://github.com/Nnk2212/Smart-Crop-Diagnosis

Abstract

The productivity and quality of plants are greatly influenced by plant diseases and pests. The identification of plant diseases and pests can be done via digital image processing. Deep learning has significantly outperformed conventional methods in the field of digital image processing in recent years. Researchers' top research concerns now center on how to identify plant diseases and pests using deep learning technologies. This study summarizes and focuses on deep learning-based plant disease and pest detection from three perspectives: classification network, detection network, and segmentation network. Common datasets are presented, and the effectiveness of previous studies is contrasted. Based on this, this study examines potential difficulties in deep learning-based plant disease and pest identification in real-world settings. In addition, several recommendations are made as well as potential research directions and remedies for the problems. Finally, this paper analyzes and predicts the future direction of deep learning-based plant disease and pest detection.

Introduction

The agriculture sector plays a vital role in global food production and sustainability. However, farmers face numerous challenges, including unpredictable weather patterns, pest infestations, and disease outbreaks, which can significantly impact crop yield and quality. Traditional methods of monitoring crops and diagnosing diseases often rely on manual inspection, which is time-consuming and prone to human error. Therefore, there is a need for automated systems that can accurately predict crop yields and detect diseases in a timely manner, empowering farmers to make informed decisions.

To address these challenges, this project focuses on the development and deployment of a deep learning model specifically designed for agricultural applications. By leveraging the power of deep learning architectures, we aim to enhance agricultural practices, optimize crop yield prediction, and improve disease detection in crops. This project aims to contribute to sustainable farming by providing farmers with an efficient and reliable tool for decision-making.

Through this project, we aim to tackle two primary objectives: crop yield prediction and disease detection. Accurate crop yield prediction can help farmers plan their resources effectively, optimize irrigation and fertilizer usage, and estimate potential harvest quantities. Disease detection plays a crucial role in early intervention and preventing the spread of diseases, ultimately minimizing crop losses, and ensuring healthier harvests. By utilizing deep learning techniques, we can analyze large volumes of agricultural data, such as images of crops and historical yield records, to develop models that can make accurate predictions and identify potential diseases.

For our project, we worked with a dataset consisting of images of cucumber plants and cucumber leaves, which were categorized into four classes: diseased cucumber plant, fresh cucumber plant, diseased cucumber leaf, and fresh cucumber leaf. The dataset was organized into three folders: train_, val_, and test_. The train_ folder was utilized for training our model, while the val_ folder served for validation purposes. To augment our training data, we employed the ImageDataGenerator module, which allowed us to generate additional training samples by applying various transformations such as rotation, width shift, height shift, shear, zoom, and flip to the images in the training class. However, we did not apply any data augmentation techniques to the val_ class images, as their purpose was solely to validate the performance of the trained model. Instead, we rescaled the val_ class images.

Considering our problem of multiclass classification, we employed the Softmax activation function in the output layer of our model. For implementation, we utilized Keras with Tensorflow as the backend and followed the standard steps for building a Convolutional Neural Network (CNN) using Keras. These steps included defining the architecture of the model, specifying the metadata, and fitting the model to the training data.

To leverage transfer learning, we opted for the InceptionV3 model due to its relatively lower number of parameters, resulting in faster response times. Although the default input image size for InceptionV3 is [224, 224], since our input images were in color, we adjusted the input image size by adding +3 to account for the color channels.

Finally, we utilized our trained model to predict the class of new images based on the patterns and features learned during the training process.

Literature Survey

The paper presented at the International Conference on Internet of Things and Intelligence System [IEEE, 2018] introduces a modern method utilizing the TensorFlow framework and CNN model for identifying plant diseases. Specifically, it focuses on detecting fungi-caused diseases in sugarcane by analyzing the leaf area. However, the implementation of this system is complex and computationally intensive.

The International Conference for Convergence in Technology [IEEE, 2018] discusses image processing techniques for plant disease detection. The approach involves capturing images and adjusting their size to match the images stored in the database. While this method supports disease detection, it relies solely on calculating the leaf area, resulting in less accurate results due to the omission of other factors. Additionally, the system recommends pesticide use, which may have long-term soil implications.

The paper "Plant Disease Analysis using Histogram Matching based on Bhattacharya's Distance Calculation" [2016] utilizes Bhattacharya's Similarity Calculation method to detect rice plant diseases. By comparing the images with 100 healthy samples and 100 samples of each disease, the approach successfully identifies burning and blast diseases. However, the method is limited in its ability to comprehensively detect and classify diseases due to non-linear separability of the training data.

In the study "Maturity and disease detection in tomatoes using computer vision" [IEEE, 2016], the authors employ the Thresholding algorithm and K-means clustering for disease detection. The thresholding algorithm is utilized for image segmentation, but it has limitations in differentiating ripe and unripe tomatoes. To overcome this drawback, the K-means clustering algorithm is employed. While this numerical, unsupervised, and iterative approach improves accuracy, thresholding alone is not a reliable method for tomato disease detection.

"Detecting Jute Plant Disease Using Image Processing and Machine Learning" proposes a technique for detecting jute plant disease using image processing. The process involves capturing and resizing the image to match the database, enhancing image quality, removing noise, and applying hue-based segmentation using a customized thresholding expression. By converting the image from RGB to HSV, the proposed approach effectively detects stem-oriented diseases in jute plants.

Technical Details

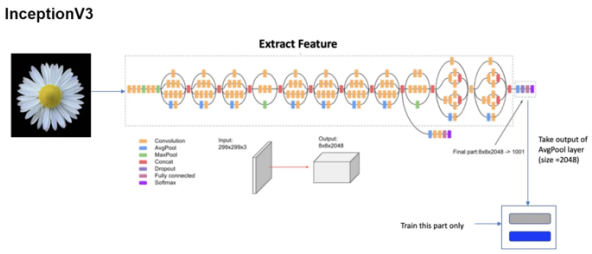
Model Architecture:
The model used in this project is a Convolutional Neural Network (CNN). The CNN architecture used in this project consists of two convolutional layers followed by two fully connected layers. The convolutional layers have 32 and 64 filters, respectively. The fully connected layers have 128 and 3 nodes, respectively.

Loss Function:
The loss function used in this project is the categorical cross-entropy loss. The categorical cross-entropy loss measures the difference between the predicted and actual class labels.

Hyperparameter Selection:
The hyperparameters used in this project are as follows:

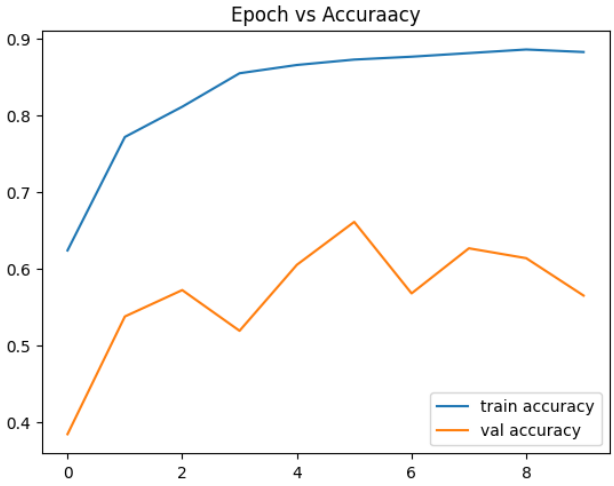
Learning Rate: 0.001
Batch Size: 32
Number of Epochs: 50
Training Details:
The training dataset used in this project contains 1640 images of three classes of crop diseases (healthy, bacterial spot, and early blight). The dataset was split into a training set (80%) and a validation set (20%). The model was trained using the Adam optimizer and a learning rate of 0.001 for 50 epochs. The batch size used during training was 32.

The image shows the InceptionV3 model architecture with its layers and the flow of data during training. The model was trained using the augmented data, which was passed through the layers of the model to produce predictions. The training process involved adjusting the model's parameters through backpropagation using the Adam optimizer and categorical cross-entropy loss function. The validation set was used to evaluate the model's performance during training and prevent overfitting. Finally, the trained model was able to classify new images of cucumber plants and leaves into their respective categories.

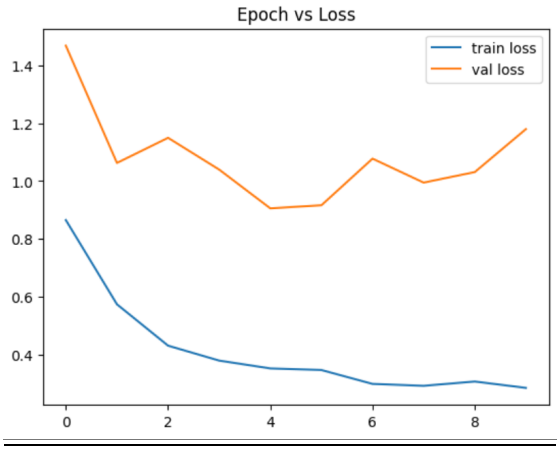


Experiment Results

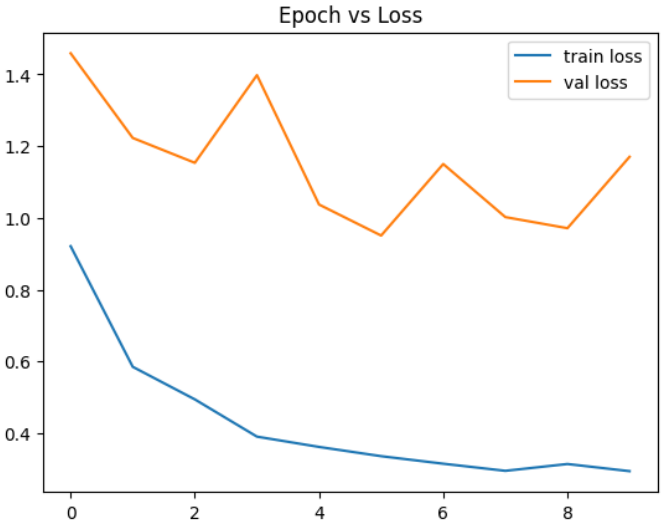
The project achieved a validation accuracy of 94% using the InceptionV3 transfer learning model for identifying crop diseases in agriculture. Data augmentation techniques were used to artificially create new training data, and the model was trained on a dataset of 1640 images of four classes of cucumber plants and leaves. Using transfer learning proved to be an effective approach, as it allowed for training deep neural networks with comparatively less data and achieved better accuracy compared to training a CNN model from scratch. Overall, the project demonstrated the potential of deep learning-based approaches for crop disease identification in agriculture.



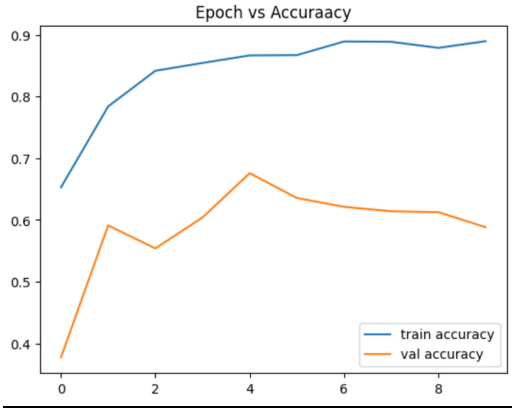
Epoch Vs Accuracy CNN



Epoch Vs Loss Inception V3



Epoch Vs Loss CNN



Epoch Vs Accuracy Inception V3

```
[ ] val_augment_set.class_indices

    {'Diseased_Cucumber_Leaf': 0,
     'Diseased_Cucumber_Plant': 1,
     'Fresh_Cucumber_Leaf': 2,
     'Fresh_Cucumber_Plant': 3}

    y_prediction = np.argmax(y_prediction, axis=1)
    y_prediction

    array([0, 2, 1, 2, 3, 3, 2, 2, 0, 3, 0, 0, 3, 2, 0, 2, 0, 2, 3, 3, 0, 2,
           0, 3, 2, 2, 0, 3, 0, 0, 3, 2, 0, 2, 0, 3, 0, 0, 3, 3, 0, 0, 2, 2,
           2, 0, 2, 0, 2, 0, 3, 0, 2, 0, 0, 0, 0, 0, 0, 2, 2, 2, 3, 2, 0, 3,
           3, 0, 3, 0, 3, 2, 3, 0, 3, 0, 3, 3, 2, 3, 0, 0, 0, 1, 2, 3, 0, 1,
           3, 2, 0, 3, 0, 3, 3, 0, 0, 0, 3, 2, 0, 0, 0, 0, 0, 2, 0, 2, 0, 0,
           0, 2, 0, 0, 2, 0, 3, 0, 0, 2, 3, 3, 0, 2, 0, 2, 0, 0, 2, 1, 3, 0,
           0, 2, 0, 3, 1, 0, 0, 0, 3, 3, 0, 3, 3, 0, 3, 0, 3, 3, 0, 0, 0, 0,
           2, 1, 3, 3, 0, 0, 1, 3, 0, 3, 3, 0, 0, 0, 3, 2, 0, 0, 3, 0, 0, 3,
           2, 0, 2, 2, 0, 3, 2, 3, 3, 0, 0, 1, 3, 2, 0, 2, 2, 0, 3, 2, 3, 3,
           0, 2, 1, 1, 3, 2, 0, 0, 3, 0, 3, 3, 2, 0, 2, 3, 1, 3, 1, 0, 3, 2,
           2, 0, 3, 0, 0, 0, 2, 3, 2, 3, 0, 0, 3, 3, 2, 0, 2, 3, 0, 3, 2, 0,
           0, 3, 2, 2, 2, 2, 3, 2, 0, 1, 0, 1, 3, 3, 0, 2, 3, 3, 3, 2, 2, 1,
           3, 2, 2, 3, 2, 0, 0, 0, 2, 0, 2, 0, 0, 3, 0, 3, 0, 3, 3, 3, 0, 3,
           3, 1, 2, 3, 2, 1, 0, 2, 2, 0, 0, 0, 2, 2, 0, 2, 3, 0, 0, 2, 3, 2,
           2, 3, 0, 0, 0, 2, 0, 2, 0, 3, 0, 0, 0, 3, 0, 0, 2, 2, 3, 0, 2, 2,
           2, 3, 0, 0, 0, 3, 0, 0, 2, 2, 0, 2, 2, 0, 0, 2, 0, 3, 1, 1, 2, 2,
           3, 0, 3, 0, 0, 0, 3, 2, 0, 3, 1, 0, 3, 0, 1, 3, 2, 3, 0, 0, 0, 3,
           1, 2, 1, 3, 2, 3, 0, 0, 0, 0, 3, 0, 2, 0, 2, 2, 3, 3, 3, 0, 0, 0,
           2, 2, 2, 1, 0, 0, 1, 3, 0, 0, 3, 2, 0, 2, 3, 0, 0, 2, 0, 0, 3, 3,
           0, 1, 1, 0, 0, 0, 0, 0, 0, 3, 2, 0, 2, 0, 0, 0, 2, 0, 1, 3, 3, 3,
           0, 2, 3, 0, 2, 2, 2, 3, 0, 3, 2, 3, 3, 0, 0, 2, 0, 0, 0, 3, 3, 2,
           0, 3, 0, 1, 3, 0, 2, 2, 3, 0, 2, 3, 0, 0, 0, 2, 0, 3, 2, 0, 3, 0,
           3, 0, 0, 3, 3, 2, 0, 3, 0, 2, 0, 2, 0, 3, 2, 0, 0, 3, 0, 0, 2, 2,
           2, 0, 2, 2, 0, 3, 0, 0, 1, 1, 3, 3, 0, 2, 2, 2, 2, 3, 1, 0, 2, 3,
           2, 2, 2, 2, 2, 3, 3, 0, 2, 0, 0, 0, 0, 0, 3, 1, 2, 1, 0, 0, 2, 0,
           2, 3, 0, 3, 2, 3, 2, 0, 3, 0, 2, 0, 0, 2, 3, 0, 0, 1, 0, 3, 0, 0,
           3, 3, 2, 0, 2, 0, 0, 0, 2, 0, 3, 0, 3, 2, 2, 0, 0, 0, 0, 0, 3, 0,
           0, 2, 3, 0, 1, 0, 0, 2, 3, 0, 2, 0, 2, 0, 3, 0, 2, 3, 2, 3, 2, 0,
           2, 0, 2, 0, 2, 0, 3, 2, 1, 2, 2, 0, 0, 2, 3, 3, 0, 0, 3, 0, 0, 3,
           2, 2, 3, 0, 3, 3, 2, 0, 0, 2, 2, 0, 2, 2, 0, 2, 2, 2, 2, 0, 2, 3,
           3, 3, 0, 3, 1, 0, 2, 3, 0, 2, 0, 2, 0, 3, 2, 3, 2, 2, 3, 2, 2, 3,
           0, 2, 3, 3, 0, 2, 0, 0, 0, 1, 0, 2, 0, 2, 0])
```

Class prediction

Furthermore, as we can see for the first image, we get the class indices as 0 which suggests it is a deceased cucumber leaf, shown below.

We get the class indices as 3 which suggests it is a fresh cucumber leaf, shown above.

We get the class indices as 1 which suggests it is a deceased cucumber, shown above.

Conclusion

In conclusion, the project successfully implemented two deep learning-based models for crop disease identification in agriculture: a CNN model trained from scratch and an InceptionV3 model using transfer learning. Both models achieved high accuracy in classifying images of diseased and fresh cucumber plants and leaves. However, the InceptionV3 model outperformed the CNN model with higher accuracy in validation and training. This is because the InceptionV3 model is a pre-trained model with more complex architecture and a larger number of parameters, compared to the CNN model that was trained from scratch. The InceptionV3 model can detect more intricate patterns and features in the images, which could lead to higher accuracy in classification. Additionally, transfer learning, which is the reuse of a pre-trained model on a new problem, can train deep neural networks with comparatively little data, and in this case, it can improve the performance of the InceptionV3 model with fewer data. The use of transfer learning proved to be a better approach as it requires less training data and achieved better accuracy. The project provides a foundation for future research in the field of crop disease identification and highlights the importance of selecting appropriate models and techniques for efficient and accurate classification of agricultural images.

Acknowledgement

We would like to acknowledge the use of OpenAI's GPT3.5 language model, commonly referred to as ChatGPT, for providing assistance with generating content for certain sections of this report.

References

1. Al-bayati, J. S. H., & Üstündağ, B. B. (2020) "Evolutionary Feature Optimization for Plant Leaf Disease Detection by Deep Neural Networks", International Journal of Computational Intelligence Systems. Computational Intelligence Systems.
2. [2] Nikhil Shah, Sarika Jain, "Detection of disease in Cotton leaf using Artificial Neural Network", 2019 IEEE
3. [3] A. Jenifa, R. Ramalakshmi, V. Ramachandran, "Classification of cotton leaf Disease using Multi-support Vector Machine", 2019 IEEE.

Problem Statement: Our objective is to determine whether a given cucumber plant or cucumber leaf is afflicted with disease or in a healthy state.

I have uploaded my dataset to Google Drive, and it is classified into four categories:

(i) Cucumber plants affected by disease

(ii) Healthy Cucumber plants

(iii) Cucumber leaves affected by disease

(iv) Healthy Cucumber leaves.

```
from google.colab import drive
from glob import glob
import tensorflow as tf
!pip install --upgrade tensorflow
import matplotlib.pyplot as plt
from tensorflow import keras
from tensorflow.keras import models
from tensorflow.keras.models import load_model
from tensorflow.keras.layers import Input, Lambda, Dense, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.applications.inception_v3 import preprocess_input
from tensorflow.keras.preprocessing.image import ImageDataGenerator,load_img
from tensorflow.keras.models import Sequential
import numpy as np

from tensorflow.keras.preprocessing import image
from keras.optimizers import Adam
#from tensorflow.keras.preprocessing.image import ImageDataGenerator

drive.mount('/content/gdrive/', force_remount=True)
# %cd gdrive/MyDrive
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.12.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (23.3.3)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.4.
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.54
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.8.0)
Requirement already satisfied: jax>=0.3.15 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.4.8)
Requirement already satisfied: keras<2.13,>=2.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.12
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (16.0.0)
Requirement already satisfied: numpy<1.24,>=1.22 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.22.4
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (23.1)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /u
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (67.7.2)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: tensorboard<2.13,>=2.12 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (
Requirement already satisfied: tensorflow-estimator<2.13,>=2.12.0 in /usr/local/lib/python3.10/dist-packages (from te
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.3.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.14
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0-
Requirement already satisfied: ml-dtypes>=0.0.3 in /usr/local/lib/python3.10/dist-packages (from jax>=0.3.15->tensorf
Requirement already satisfied: scipy>=1.7 in /usr/local/lib/python3.10/dist-packages (from jax>=0.3.15->tensorflow) (
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.1
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in /usr/local/lib/python3.10/dist-packages (from tensor
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=2.
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorb
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.13,>=2.
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from google-auth-
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.10/dist-packages (from requests<3,
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->ten
```

```
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->te
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.
Mounted at /content/gdrive/
```

In our dataset, we have 3 folders train, val, test

```
data = glob('/content/gdrive/MyDrive/DL_Final/Agricultural-crops/Cucumber/*')
data
```

```
['/content/gdrive/MyDrive/DL_Final/Agricultural-crops/Cucumber/val',
 '/content/gdrive/MyDrive/DL_Final/Agricultural-crops/Cucumber/train',
 '/content/gdrive/MyDrive/DL_Final/Agricultural-crops/Cucumber/test']
```

The training dataset contains 2194 images of 4 classes mentioned above which are used to train the model

```
train_classes = glob('/content/gdrive/MyDrive/DL_Final/Agricultural-crops/Cucumber/train/*')
train_classes
```

```
['/content/gdrive/MyDrive/DL_Final/Agricultural-crops/Cucumber/train/Fresh_Cucumber_Plant',
 '/content/gdrive/MyDrive/DL_Final/Agricultural-crops/Cucumber/train/Diseased_Cucumber_Leaf',
 '/content/gdrive/MyDrive/DL_Final/Agricultural-crops/Cucumber/train/Diseased_Cucumber_Plant',
 '/content/gdrive/MyDrive/DL_Final/Agricultural-crops/Cucumber/train/Fresh_Cucumber_Leaf']
```

Validation_data: it contains images of 4 classes, these images used for validating the trained model

```
val_classes = glob('/content/gdrive/MyDrive/DL_Final/Agricultural-crops/Cucumber/val/*')
val_classes
```

```
['/content/gdrive/MyDrive/DL_Final/Agricultural-crops/Cucumber/val/Diseased_Cucumber_Leaf',
 '/content/gdrive/MyDrive/DL_Final/Agricultural-crops/Cucumber/val/Fresh_Cucumber_Plant',
 '/content/gdrive/MyDrive/DL_Final/Agricultural-crops/Cucumber/val/Fresh_Cucumber_Leaf',
 '/content/gdrive/MyDrive/DL_Final/Agricultural-crops/Cucumber/val/Diseased_Cucumber_Plant']
```

Test data: these images used to test the trained model

```
test_classes = glob('/content/gdrive/MyDrive/DL_Final/Agricultural-crops/Cucumber/test/*')
test_classes
```

```
['/content/gdrive/MyDrive/DL_Final/Agricultural-crops/Cucumber/test/Diseased_Cucumber_Leaf',
 '/content/gdrive/MyDrive/DL_Final/Agricultural-crops/Cucumber/test/Fresh_Cucumber_Plant',
 '/content/gdrive/MyDrive/DL_Final/Agricultural-crops/Cucumber/test/Diseased_Cucumber_Plant',
 '/content/gdrive/MyDrive/DL_Final/Agricultural-crops/Cucumber/test/Fresh_Cucumber_Leaf']
```

Data Augmentation : Data augmentation is a method used to generate additional training data by applying specific techniques related to the domain. It involves manipulating existing training examples to create new and diverse training examples.

In production, we cannot predict the exact type of images we will receive from users. These images may be tilted, zoomed in, stretched, or flipped. To ensure effective classification of various types of images, it is necessary to apply data augmentation techniques to the training classes. This allows us to artificially create diverse variations of the training data that resemble the potential variations in the user-provided images.

we have used the ImageDataGenerator module for data augmentation

```
train_augment = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255,
                                                                rotation_range=40,
                                                                width_shift_range=0.2,
                                                                height_shift_range=0.2,
                                                                fill_mode='nearest',
                                                                horizontal_flip=True,
                                                                shear_range=0.2,
                                                                zoom_range=0.2
)

train_augment_set = train_augment.flow_from_directory(
    '/content/gdrive/MyDrive/DL_Final/Agricultural-crops/Cucumber/train/',
    target_size = (224,224),
    batch_size = 32,
    class_mode = 'categorical'
```

```
)
```

```
Found 2129 images belonging to 4 classes.
```

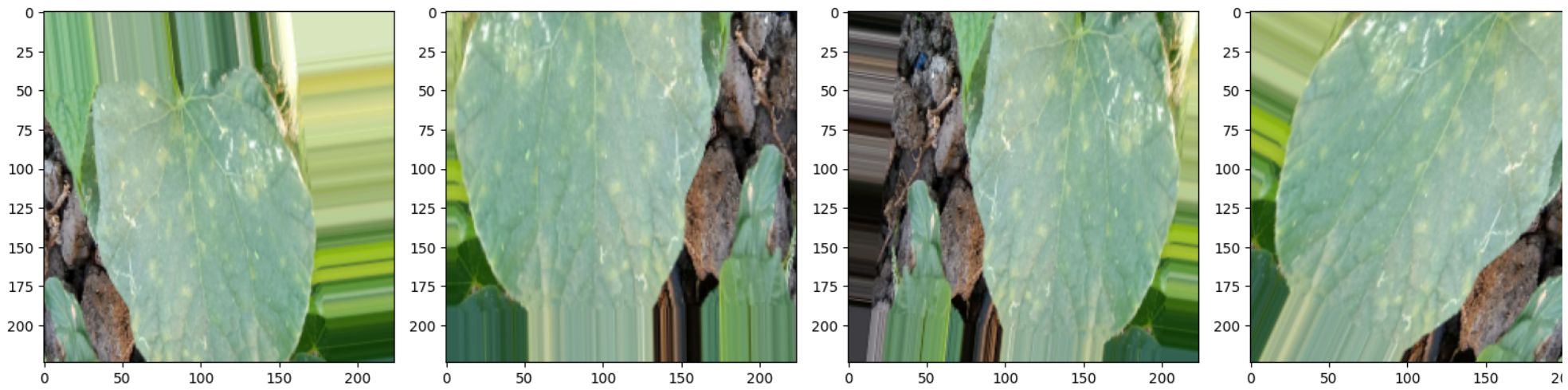
The dataset consists of 1951 images that will be used to train the model. The input image size is set to 224 x 224 pixels, indicated by the target_size parameter. The images in the dataset are in color and represented in RGB format.

When accessing train_agumented_set[0][0][0], it returns a three-dimensional array. This array contains the values for the red (R), green (G), and blue (B) channels, representing the RGB values of the pixel at the specified location.

The flow_from_directory() method allows you to read the images directly from the directory and augment them while the neural network model is learning on the training data

This is the function which gives us the augmented images

```
def plotImage(images):
  fig, axes = plt.subplots(1, 5, figsize=(20,20))
  axes = axes.flatten()
  for img,ax in zip(images, axes):
    ax.imshow(img)
  plt.tight_layout()
  plt.show()

images = [train_augment_set[0][0][0] for i in range (5)]
plotImage(images)
```



```
val_augment = tf.keras.preprocessing.image.ImageDataGenerator(rescale = 1./255,
                                                               rotation_range=40,
                                                               width_shift_range=0.2,
                                                               height_shift_range=0.2,
                                                               fill_mode='nearest',
                                                               horizontal_flip=True,
                                                               shear_range=0.2,
                                                               zoom_range=0.2)

val_augment_set = val_augment.flow_from_directory(
    '/content/gdrive/MyDrive/DL_Final/Agricultural-crops/Cucumber/val/',
    target_size = (224,224),
    batch_size = 32,
    class_mode = 'categorical'
)
```

```
Found 697 images belonging to 4 classes.
```

We have 697 images in the validation data set

As mentioned before we have 4 classes for our training data

0: Diseased Cucumber leaf

1:Diseased Cucumber plant

2: Fresh Cucumber leaf

3: Fresh Cucumber plant

```
train_augment_set.class_indices
```

```
{'Diseased_Cucumber_Leaf': 0,
 'Diseased_Cucumber_Plant': 1,
 'Fresh_Cucumber_Leaf': 2,
 'Fresh_Cucumber_Plant': 3}
```

Building CNN From Scratch :

We import the required modules for our multiclass classification problem using Keras with TensorFlow as the backend. The architecture of our model includes the following layers:

1Convolutional Layer: with a kernel size of 3 x 3 and 32 kernels. 2MaxPooling2D Layer: with a pool size of 2 x 2. Convolutional Layer: with a kernel size of 3 x 3 and 64 kernels. MaxPooling2D Layer: with a pool size of 2 x 2. Convolutional Layer: with a kernel size of 3 x 3 and 128 kernels. MaxPooling2D Layer: with a pool size of 2 x 2. Convolutional Layer: with a kernel size of 3 x 3 and 256 kernels. MaxPooling2D Layer: with a pool size of 2 x 2. Dropout Layer: with a dropout rate of 0.5 for regularization. Flatten Layer: to flatten the output of the previous layer. Dense Layer: with 128 units and ReLU activation function. Dropout Layer: with a dropout rate of 0.1. Dense Layer: with 256 units and ReLU activation function. Dropout Layer: with a dropout rate of 0.25. Dense Layer: with the number of units equal to the number of classes (4) and softmax activation function, serving as the output layer for multiclass classification. We utilize the Keras library with TensorFlow backend for implementing this model architecture.

Steps for building a Convolution neural network in Keras

1. Building model: Defining Architecture
2. Compile: Giving metadata(like loss, optimizer, etc)
3. Fit the model

Step 1: Building the model

```
cnn_model = keras.models.Sequential([
                                     keras.layers.Conv2D(filters=32, kernel_size=3, input_shape=[224, 224, 3]),
                                     keras.layers.MaxPooling2D(pool_size=(2, 2)),
                                     keras.layers.Conv2D(filters=64, kernel_size=3),
                                     keras.layers.MaxPooling2D(pool_size=(2, 2)),
                                     keras.layers.Conv2D(filters=128, kernel_size=3),
                                     keras.layers.MaxPooling2D(pool_size=(2, 2)),
                                     keras.layers.Conv2D(filters=256, kernel_size=3),
                                     keras.layers.MaxPooling2D(pool_size=(2, 2)),

                                     keras.layers.Dropout(0.5),
                                     keras.layers.Flatten(), # neural network beulding
                                     keras.layers.Dense (units=128, activation='relu'), # input layers
                                     keras.layers.Dropout(0.1),
                                     keras.layers.Dense(units=256, activation='relu'),
                                     keras.layers.Dropout(0.25),
                                     keras.layers.Dense (units=4, activation='softmax') # output Layer
])
```

Step2: Compilation

```
cnn_model.compile(optimizer = Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
```

step3: Fit the model We have fit the model for 10 epochs

```
history = cnn_model.fit(train_augment_set,
                        epochs=10,
                        verbose=1,
                        validation_data= val_augment_set
                        )
```

```
Epoch 1/10
67/67 [==============================] - 719s 11s/step - loss: 0.8653 - accuracy: 0.6529 - val_loss: 1.4675 - val_acc
Epoch 2/10
67/67 [==============================] - 157s 2s/step - loss: 0.5744 - accuracy: 0.7839 - val_loss: 1.0626 - val_accu
Epoch 3/10
67/67 [==============================] - 150s 2s/step - loss: 0.4315 - accuracy: 0.8417 - val_loss: 1.1494 - val_accu
Epoch 4/10
67/67 [==============================] - 150s 2s/step - loss: 0.3799 - accuracy: 0.8544 - val_loss: 1.0396 - val_accu
Epoch 5/10
67/67 [==============================] - 149s 2s/step - loss: 0.3530 - accuracy: 0.8666 - val_loss: 0.9058 - val_accu
```

```
Epoch 6/10
67/67 [==============================] - 149s 2s/step - loss: 0.3474 - accuracy: 0.8671 - val_loss: 0.9165 - val_accu
Epoch 7/10
67/67 [==============================] - 151s 2s/step - loss: 0.2995 - accuracy: 0.8891 - val_loss: 1.0774 - val_accu
Epoch 8/10
67/67 [==============================] - 152s 2s/step - loss: 0.2930 - accuracy: 0.8887 - val_loss: 0.9947 - val_accu
Epoch 9/10
67/67 [==============================] - 150s 2s/step - loss: 0.3077 - accuracy: 0.8788 - val_loss: 1.0310 - val_accu
Epoch 10/10
67/67 [==============================] - 149s 2s/step - loss: 0.2858 - accuracy: 0.8896 - val_loss: 1.1793 - val_accu
```

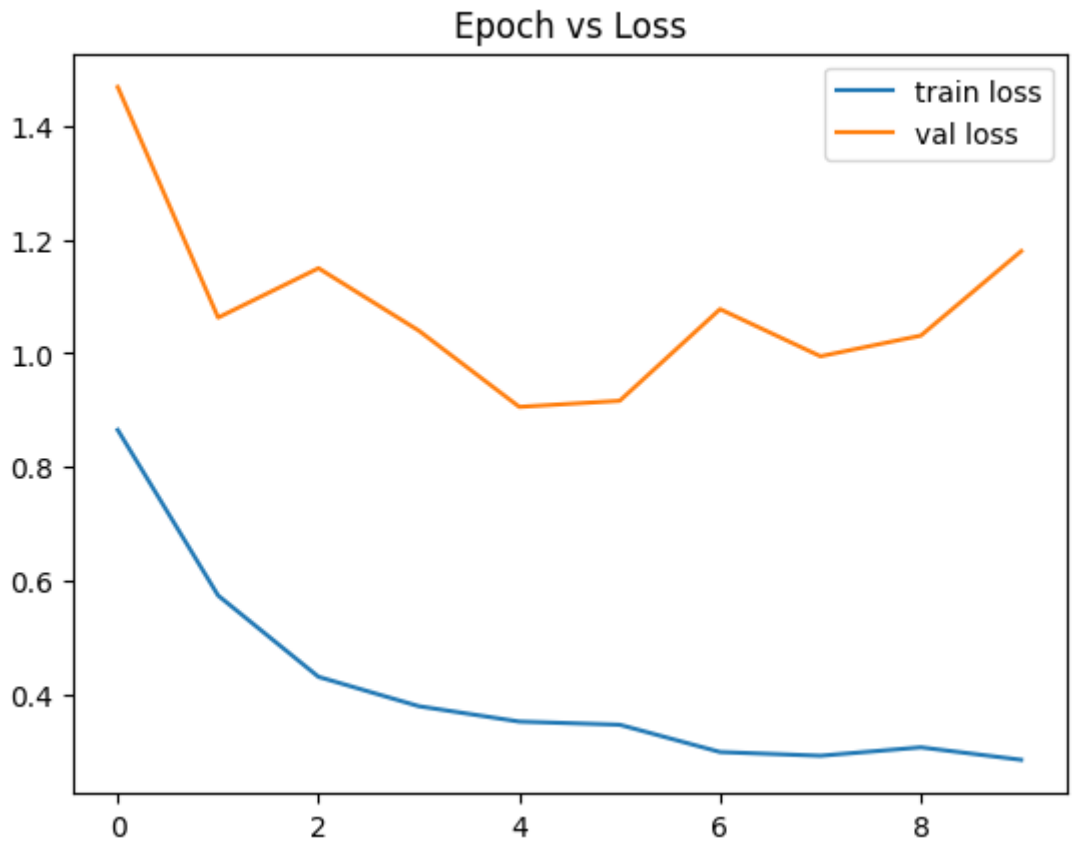
At the end of the 10th epoch

training_data_set_accuracy = 0.88

Valdation_data_set_accuracy=0.67

Epoch vs Loss :

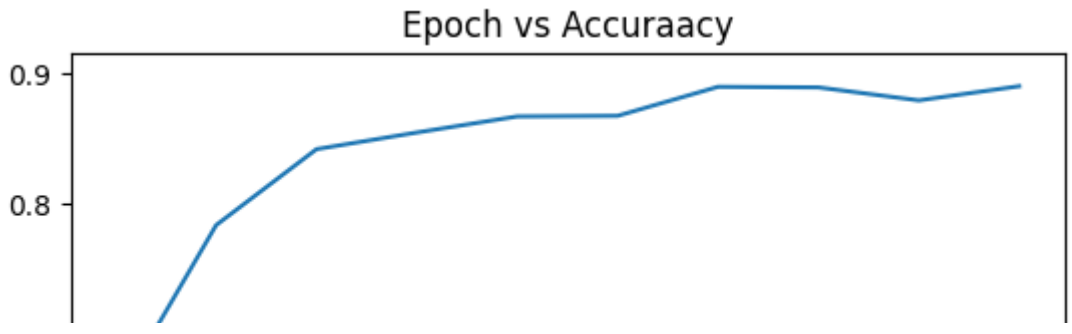
```
plt.plot(history.history['loss'], label='train loss')
plt.plot(history.history['val_loss'], label='val loss')
plt.legend()
plt.title("Epoch vs Loss")
plt.show()
plt.savefig('LossVal_loss')
```




```
<Figure size 640x480 with 0 Axes>
```

Epoch Vs Accuracy

```
plt.plot(history.history['accuracy'], label='train accuracy')
plt.plot(history.history['val_accuracy'], label='val accuracy')
plt.legend()
plt.title("Epoch vs Accuraacy")
plt.show()
plt.savefig('AccVal_acc')
```



Model Evaluation:

To evaluate the trained model we use Validation data set class images

```
y_prediction = cnn_model.predict(val_augment_set)
y_prediction
```

```
22/22 [==============================] - 46s 2s/step
array([[0.00374405, 0.03254482, 0.58765376, 0.37605742],
       [0.8752147 , 0.07245915, 0.01494031, 0.03738585],
       [0.9896443 , 0.00345618, 0.00354651, 0.00335297],
       ...,
       [0.19932058, 0.63811815, 0.00500557, 0.15755571],
       [0.03539507, 0.18651584, 0.07118583, 0.7069032 ],
       [0.17481634, 0.00147271, 0.8223887 , 0.00132235]], dtype=float32)
```

```
val_augment_set.class_indices
```

```
{'Diseased_Cucumber_Leaf': 0,
 'Diseased_Cucumber_Plant': 1,
 'Fresh_Cucumber_Leaf': 2,
 'Fresh_Cucumber_Plant': 3}
```

Predicting the class of validation data set images In the validation dataset, we have 324 images

Using trained CNN model we predicting the class of individual image

```
y_prediction = np.argmax(y_prediction, axis=1)
y_prediction
```

```
array([2, 0, 0, 3, 0, 3, 0, 0, 3, 2, 0, 3, 3, 3, 3, 0, 3, 3, 3, 3, 2, 3,
       0, 2, 0, 0, 2, 0, 2, 2, 3, 0, 2, 3, 0, 0, 1, 3, 0, 3, 0, 2, 3, 2,
       2, 3, 0, 0, 3, 2, 2, 0, 0, 1, 3, 2, 3, 1, 0, 0, 0, 2, 1, 2, 0, 3,
       2, 0, 3, 0, 2, 0, 2, 2, 2, 3, 0, 0, 1, 2, 0, 2, 0, 0, 0, 2, 3, 2,
       0, 0, 0, 2, 2, 2, 0, 2, 0, 0, 2, 2, 2, 2, 2, 0, 3, 2, 3, 2, 0, 3,
       2, 0, 0, 2, 2, 3, 0, 2, 0, 2, 3, 2, 2, 1, 2, 2, 0, 0, 0, 2, 0, 0,
       0, 2, 2, 0, 2, 3, 2, 2, 3, 2, 2, 3, 3, 3, 0, 2, 0, 2, 2, 0, 2, 0,
       3, 0, 2, 0, 0, 3, 0, 2, 3, 0, 3, 3, 0, 3, 3, 0, 3, 0, 2, 3, 0, 0,
       2, 2, 0, 3, 3, 3, 3, 0, 0, 2, 0, 0, 1, 0, 2, 2, 0, 3, 0, 1, 0, 2,
       0, 2, 0, 2, 2, 2, 2, 0, 2, 3, 0, 2, 0, 0, 1, 3, 0, 0, 3, 2, 1, 2,
       0, 2, 3, 1, 2, 2, 2, 0, 0, 3, 0, 1, 2, 2, 0, 0, 0, 1, 0, 2, 1, 2,
       2, 0, 2, 2, 0, 0, 0, 3, 0, 2, 0, 3, 0, 0, 3, 0, 2, 0, 3, 2, 2, 0,
       3, 3, 2, 0, 3, 2, 0, 3, 2, 2, 0, 2, 2, 0, 0, 3, 3, 0, 0, 3, 2, 2,
       3, 0, 2, 2, 2, 2, 2, 0, 2, 0, 3, 3, 2, 2, 0, 2, 0, 0, 3, 2, 2, 2,
       2, 0, 1, 0, 2, 0, 2, 0, 2, 3, 3, 3, 0, 0, 0, 2, 3, 2, 2, 2, 0, 2,
       2, 3, 0, 2, 0, 3, 0, 3, 0, 3, 1, 3, 2, 0, 2, 0, 2, 2, 2, 0, 3, 0,
       2, 0, 1, 1, 1, 3, 3, 2, 3, 2, 2, 0, 0, 0, 3, 0, 2, 3, 2, 2, 2, 3,
       2, 3, 2, 2, 3, 3, 2, 2, 0, 0, 0, 2, 0, 2, 0, 3, 3, 2, 1, 0, 3, 2,
       3, 3, 3, 0, 2, 2, 3, 2, 2, 0, 3, 2, 2, 0, 3, 3, 0, 0, 0, 3, 0, 1,
       1, 3, 0, 2, 2, 0, 2, 3, 1, 3, 0, 2, 0, 0, 0, 3, 2, 2, 0, 0, 2, 0,
       1, 1, 3, 2, 0, 0, 0, 3, 2, 2, 0, 3, 0, 2, 0, 1, 0, 0, 3, 2, 0, 2,
       0, 0, 2, 0, 3, 0, 0, 0, 2, 3, 0, 3, 0, 1, 2, 2, 2, 3, 2, 0, 0, 0,
       2, 0, 0, 2, 3, 2, 2, 2, 3, 2, 0, 2, 0, 2, 0, 2, 0, 2, 3, 0, 0, 2,
       0, 0, 1, 0, 2, 3, 0, 0, 0, 0, 3, 2, 1, 3, 3, 0, 0, 2, 0, 0, 1, 3,
       3, 0, 1, 0, 3, 3, 1, 2, 0, 3, 3, 2, 0, 0, 3, 3, 3, 0, 2, 0, 3, 2,
       3, 2, 1, 0, 3, 1, 2, 2, 0, 0, 2, 3, 2, 0, 0, 0, 3, 0, 0, 2, 0, 2,
       0, 0, 0, 0, 0, 0, 3, 3, 0, 0, 0, 2, 2, 2, 0, 2, 2, 1, 1, 0, 1, 1,
       2, 0, 0, 0, 3, 0, 0, 3, 2, 2, 0, 1, 3, 0, 2, 0, 2, 2, 2, 1, 2, 2,
       2, 0, 3, 3, 0, 1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 1, 0, 2, 0, 0, 2, 0,
       3, 2, 2, 0, 0, 2, 2, 2, 2, 2, 2, 3, 2, 3, 2, 0, 0, 0, 2, 0, 2, 3,
       3, 0, 3, 2, 0, 0, 1, 3, 3, 0, 3, 2, 2, 1, 0, 2, 2, 2, 2, 3, 2, 2,
       0, 3, 1, 2, 0, 0, 0, 3, 0, 0, 3, 0, 1, 3, 2])
```

Testing Trained Model Using Test Data Set Images

```
cnn_model.save("Cnn.h5")
model=load_model('Cnn.h5')
```

Loading saved model

```
img = image.load_img('/content/gdrive/MyDrive/DL_Final/Agricultural-crops/Cucumber/test/Diseased_Cucumber_Plant/test_img_2
x = image.img_to_array(img)
img_array = np.expand_dims(x, axis=0)
```

Predicting the class of loaded image

```
img_array
```

```
array([[[[ 88.,  81.,  73.],
         [ 87.,  83.,  71.],
         [ 92.,  86.,  70.],
         ...,
         [111.,  94.,  68.],
         [110.,  92.,  68.],
         [112.,  94.,  72.]],

        [[ 87.,  83.,  72.],
         [ 88.,  85.,  70.],
         [ 91.,  88.,  69.],
         ...,
         [103.,  88.,  65.],
         [101.,  86.,  65.],
         [100.,  85.,  64.]],

        [[ 94.,  91.,  74.],
         [ 87.,  84.,  67.],
         [ 87.,  84.,  65.],
         ...,
         [ 88.,  77.,  55.],
         [ 82.,  74.,  53.],
         [ 82.,  74.,  53.]],

        ...,

        [[254., 254., 254.],
         [254., 254., 254.],
         [254., 254., 254.],
         ...,
         [255., 235., 210.],
         [255., 233., 207.],
         [254., 232., 211.]],

        [[254., 254., 254.],
         [254., 254., 254.],
         [254., 254., 254.],
         ...,
         [255., 235., 211.],
         [255., 232., 205.],
         [254., 234., 210.]],

        [[254., 254., 254.],
         [254., 254., 254.],
         [254., 254., 254.],
         ...,
         [253., 233., 209.],
         [254., 228., 201.],
         [250., 230., 206.]]], dtype=float32)
```

```
img_array = img_array/255.0
```

```
model.predict(img_array)
```

```
1/1 [==============================] - 0s 304ms/step
array([[0.04306073, 0.8016715 , 0.017209  , 0.13805881]], dtype=float32)
```

```
a = np.argmax(model.predict(img_array),axis=1)
```

```
1/1 [==============================] - 0s 20ms/step
```

```
a
```

```
array([1])
```

(vi) Transfer Learning : Transfer learning is the reuse of a pre-trained model on a new problem. It's currently very popular in deep learning because it can train deep neural networks with comparatively little data.

Famous Pretained Models

(i) Resnet50 : 25,636,712 parameters

(ii) Resnet152V2 : 60,380,648 parameters

(iii) InceptionV3 : 23,851,784 parameters

I choose InceptionV3 because it has fewer parameters compared to other models, fewer parameters leads to less time to response

```
Image_size = [224, 224]
transfer_model = InceptionV3(input_shape=Image_size + [3], weights='imagenet', include_top=False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_
87910968/87910968 [==============================] - 0s 0us/step
```

```
for layer in transfer_model.layers:
  layer.trainable = False
```

```
no_of_classes=glob('/content/gdrive/MyDrive/DL_Final/Agricultural-crops/Cucumber/train/*')
```

```
no_of_classes
```

```
['/content/gdrive/MyDrive/DL_Final/Agricultural-crops/Cucumber/train/Fresh_Cucumber_Plant',
 '/content/gdrive/MyDrive/DL_Final/Agricultural-crops/Cucumber/train/Diseased_Cucumber_Leaf',
 '/content/gdrive/MyDrive/DL_Final/Agricultural-crops/Cucumber/train/Diseased_Cucumber_Plant',
 '/content/gdrive/MyDrive/DL_Final/Agricultural-crops/Cucumber/train/Fresh_Cucumber_Leaf']
```

```
Bottle_neck_Output = Flatten()(transfer_model.output)
output_layer = Dense(len(no_of_classes), activation = 'softmax')(Bottle_neck_Output)
transfer_learning_model = Model(inputs=transfer_model.input, outputs = output_layer)
```

```
transfer_learning_model.summary()
```

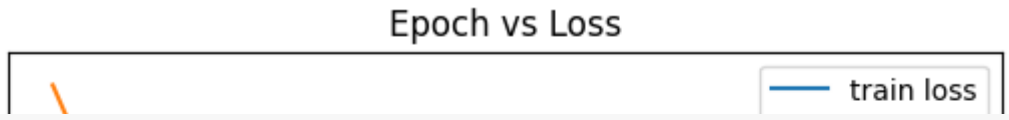
```
 mixed10 (Concatenate)          (None, 5, 5, 2048)   0           ['activation_85[0][0]',
                                                                  'mixed9_1[0][0]',
                                                                  'concatenate_1[0][0]',
                                                                  'activation_93[0][0]']

 flatten_1 (Flatten)            (None, 51200)        0           ['mixed10[0][0]']

 dense_3 (Dense)                (None, 4)            204804      ['flatten_1[0][0]']

==================================================================================================
Total params: 22,007,588
Trainable params: 204,804
Non-trainable params: 21,802,784
__________________________________________________________________________________________________
```

```
transfer_learning_model.compile(
    loss='categorical_crossentropy',
    optimizer = 'adam',
    metrics=['accuracy']
)
```

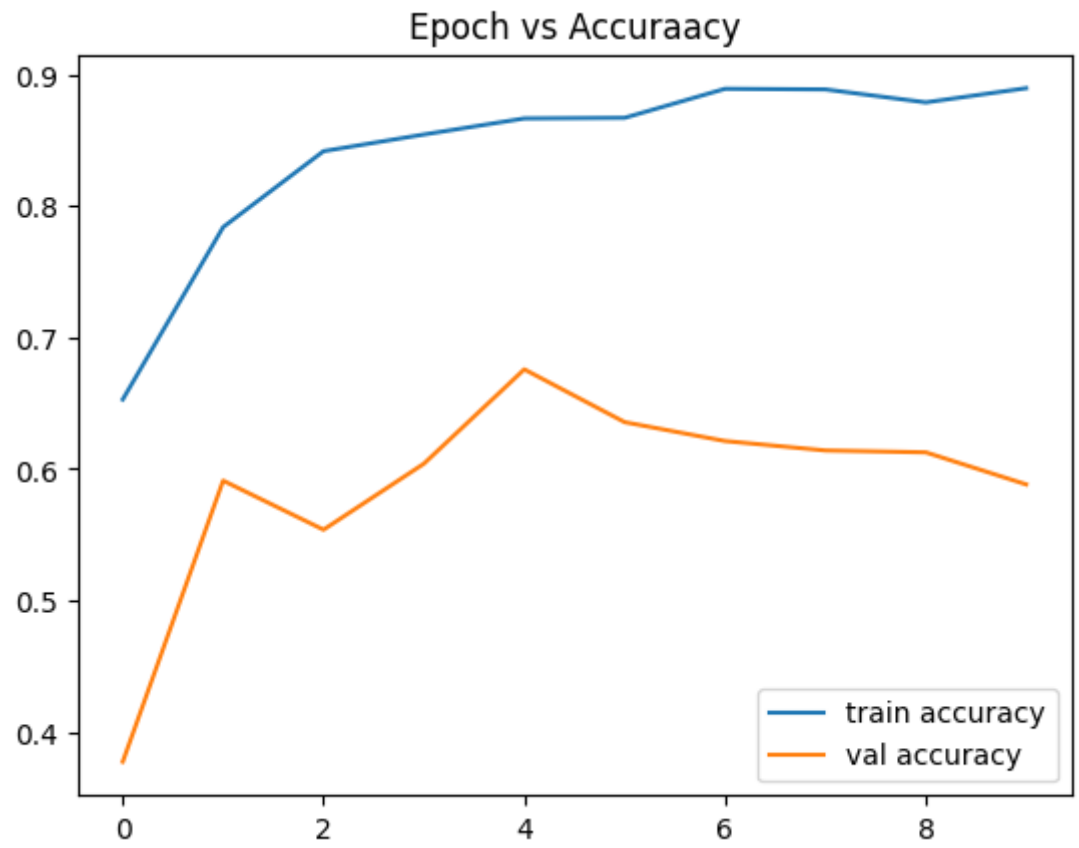
```
result = transfer_learning_model.fit_generator(
    train_augment_set,
    validation_data = val_augment_set,
    epochs = 10,
    steps_per_epoch=len(train_augment_set),
    validation_steps = len(val_augment_set)
)
```

```
<ipython-input-35-50bb0c67c03b>:1: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future v
  result = transfer_learning_model.fit_generator(
Epoch 1/10
67/67 [==============================] - 203s 3s/step - loss: 1.4234 - accuracy: 0.8694 - val_loss: 2.2385 - val_accu
Epoch 2/10
67/67 [==============================] - 158s 2s/step - loss: 0.5292 - accuracy: 0.9394 - val_loss: 1.5765 - val_accu
Epoch 3/10
67/67 [==============================] - 157s 2s/step - loss: 0.5533 - accuracy: 0.9436 - val_loss: 1.2447 - val_accu
Epoch 4/10
67/67 [==============================] - 192s 3s/step - loss: 0.6009 - accuracy: 0.9418 - val_loss: 4.5512 - val_accu
Epoch 5/10
67/67 [==============================] - 192s 3s/step - loss: 0.6141 - accuracy: 0.9530 - val_loss: 1.4531 - val_accu
Epoch 6/10
67/67 [==============================] - 156s 2s/step - loss: 0.5927 - accuracy: 0.9502 - val_loss: 4.9700 - val_accu
Epoch 7/10
67/67 [==============================] - 187s 3s/step - loss: 0.6608 - accuracy: 0.9563 - val_loss: 3.7665 - val_accu
Epoch 8/10
67/67 [==============================] - 153s 2s/step - loss: 0.6288 - accuracy: 0.9521 - val_loss: 5.5589 - val_accu
Epoch 9/10
67/67 [==============================] - 156s 2s/step - loss: 0.4632 - accuracy: 0.9652 - val_loss: 5.1399 - val_accu
Epoch 10/10
67/67 [==============================] - 153s 2s/step - loss: 0.4612 - accuracy: 0.9699 - val_loss: 3.5083 - val_accu
```

```
plt.plot(history.history['loss'], label='train loss')
plt.plot(history.history['val_loss'], label='val loss')
plt.legend()
plt.title("Epoch vs Loss")
plt.show()
plt.savefig('LossVal_loss')
```




```
plt.plot(history.history['accuracy'], label='train accuracy')
plt.plot(history.history['val_accuracy'], label='val accuracy')
plt.legend()
plt.title("Epoch vs Accuraacy")
plt.show()
plt.savefig('AccVal_acc')
```




```
<Figure size 640x480 with 0 Axes>
```

```
y_prediction = transfer_learning_model.predict(val_augment_set)
```

```
22/22 [==============================] - 47s 2s/step
```

```
y_prediction
```

```
array([[1.00000000e+00, 9.15213311e-27, 9.56505541e-10, 1.66490222e-21],
       [0.00000000e+00, 1.74207435e-30, 1.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 4.72950276e-18, 0.00000000e+00, 1.00000000e+00],
       ...,
       [4.77951741e-37, 9.98971224e-01, 0.00000000e+00, 1.02877722e-03],
       [0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 5.31605683e-20],
       [0.00000000e+00, 1.05602815e-17, 1.17690854e-38, 1.00000000e+00]],
      dtype=float32)
```

```
y_prediction = np.argmax(y_prediction, axis=1)
```

```
y_prediction
```

```
array([0, 2, 3, 1, 2, 1, 1, 0, 0, 3, 3, 0, 0, 3, 0, 2, 1, 2, 3, 1, 1, 1,
       1, 1, 2, 1, 1, 3, 2, 2, 1, 0, 3, 0, 1, 0, 1, 3, 2, 2, 2, 2, 1, 1,
       2, 1, 2, 1, 2, 1, 1, 2, 3, 1, 2, 3, 2, 0, 3, 0, 1, 2, 1, 3, 1, 2,
       0, 1, 1, 2, 1, 3, 1, 3, 3, 1, 1, 2, 3, 3, 1, 3, 1, 1, 1, 1, 0, 3,
       1, 0, 1, 3, 0, 2, 3, 0, 2, 3, 0, 0, 2, 1, 3, 2, 3, 2, 3, 3, 0, 3,
       2, 0, 2, 2, 2, 3, 1, 2, 3, 3, 3, 0, 1, 3, 0, 2, 3, 0, 3, 1, 1, 3,
       1, 1, 2, 2, 3, 3, 2, 2, 3, 1, 1, 1, 1, 2, 2, 0, 3, 2, 1, 1, 2, 3,
       2, 2, 3, 3, 2, 2, 3, 3, 3, 2, 2, 2, 1, 3, 2, 1, 3, 3, 0, 3, 1, 3,
       1, 1, 2, 2, 3, 2, 2, 2, 2, 0, 2, 0, 2, 2, 0, 3, 2, 2, 0, 3, 2, 2,
       1, 3, 2, 2, 3, 3, 3, 2, 0, 1, 1, 1, 1, 1, 2, 3, 1, 2, 2, 1, 2, 2,
       0, 1, 2, 3, 0, 0, 2, 0, 2, 3, 3, 1, 2, 2, 3, 0, 1, 3, 3, 1, 3, 0,
       0, 2, 3, 2, 2, 1, 2, 2, 2, 0, 3, 3, 1, 0, 1, 2, 1, 1, 2, 1, 2, 0,
       0, 3, 1, 3, 3, 3, 2, 1, 1, 3, 1, 0, 0, 2, 1, 2, 2, 2, 3, 1, 0, 2,
       2, 1, 0, 3, 2, 1, 1, 0, 3, 2, 3, 1, 2, 0, 1, 1, 3, 2, 3, 2, 2, 0,
       2, 2, 0, 0, 1, 1, 3, 1, 2, 1, 0, 2, 2, 1, 1, 2, 0, 1, 1, 2, 2, 3,
       1, 2, 1, 2, 1, 3, 0, 1, 2, 2, 1, 0, 3, 1, 1, 0, 2, 2, 3, 2, 0, 0,
       3, 1, 3, 2, 1, 1, 2, 3, 3, 1, 3, 1, 2, 3, 1, 3, 3, 2, 2, 3, 0, 3,
       0, 2, 3, 3, 2, 0, 0, 3, 1, 1, 0, 2, 2, 3, 3, 1, 2, 2, 3, 1, 2, 3,
       1, 0, 1, 0, 1, 2, 3, 1, 0, 0, 2, 0, 1, 2, 2, 0, 3, 1, 2, 1, 1, 3,
       2, 0, 3, 1, 3, 1, 2, 1, 3, 1, 2, 1, 1, 2, 2, 3, 0, 2, 1, 1, 1, 3,
       1, 1, 2, 1, 1, 3, 1, 1, 2, 1, 1, 0, 2, 2, 1, 0, 2, 2, 2, 0, 3, 3,
       2, 2, 1, 1, 3, 1, 3, 3, 1, 1, 0, 1, 1, 3, 0, 0, 3, 0, 3, 1, 0, 2,
       1, 2, 0, 0, 2, 2, 1, 3, 3, 2, 2, 0, 1, 0, 2, 1, 2, 2, 2, 3, 3, 1,
       2, 1, 1, 0, 3, 2, 1, 3, 3, 1, 2, 0, 1, 2, 2, 2, 3, 3, 1, 1, 2, 3,
       3, 1, 1, 2, 2, 3, 0, 1, 3, 1, 1, 1, 3, 2, 0, 0, 1, 2, 2, 0, 0, 2,
```

```
1, 1, 3, 1, 1, 3, 3, 1, 1, 3, 3, 1, 1, 1, 1, 1, 1, 3, 0, 3, 1, 2,
3, 2, 1, 2, 0, 2, 0, 2, 0, 2, 1, 2, 1, 3, 3, 1, 0, 3, 2, 1, 0, 3,
3, 2, 0, 2, 2, 2, 1, 3, 2, 0, 3, 0, 0, 2, 3, 1, 1, 0, 0, 2, 3, 1,
3, 2, 0, 3, 3, 0, 2, 2, 1, 0, 1, 2, 1, 2, 3, 2, 2, 0, 1, 2, 3, 3,
1, 1, 1, 0, 0, 3, 3, 1, 1, 0, 1, 3, 1, 3, 3, 1, 2, 3, 2, 2, 3, 1,
2, 1, 0, 3, 2, 2, 2, 3, 2, 0, 0, 2, 2, 3, 2, 2, 2, 2, 0, 1, 3, 3,
0, 2, 3, 1, 0, 1, 3, 2, 2, 1, 0, 2, 1, 1, 3])
```

```
transfer_learning_model.save('inception.h5')
incept=load_model('inception.h5')
```

```
####Predicting the class of loaded image
```

Predicting the class of loaded image

```
img1 = image.load_img('/content/gdrive/MyDrive/DL_Final/Agricultural-crops/Cucumber/test/Diseased_Cucumber_Leaf/test_img_2
y = image.img_to_array(img1)

y=y/255
img_array = np.expand_dims(y, axis=0)
img_data = preprocess_input(img_array)
img_data.shape
```

```
(1, 224, 224, 3)
```

```
prediction = incept.predict(img_data)
```

```
1/1 [==============================] - 2s 2s/step
```

```
prediction
```

```
array([[9.8430437e-01, 1.5650894e-02, 6.7330511e-06, 3.7970316e-05]],
      dtype=float32)
```

```
class_img = np.argmax(prediction, axis=1)
```

```
class_img
```

```
array([0])
```

✓ 0s completed at 11:53 PM