PART II

CHAPTER 15 OVERVIEW

As the global population continues to grow, there is an ever-increasing demand for food production to keep pace with the needs of society. Unfortunately, plant diseases pose a significant threat to agricultural productivity by affecting the yields and quality of crops. These diseases can cause significant losses for farmers and food producers, resulting in increased prices or even food shortages.

Traditionally, the identification and control of plant diseases have been time-consuming and labor-intensive processes. Trained professionals have had to rely on years of experience and extensive knowledge to diagnose and manage these diseases. This has often resulted in delays in identifying and treating problems, which can have a significant impact on crop yields and quality.

The damage and costs of plant diseases go beyond the economic loss to farmers and food producers. Environmental impact, human health consequences, and social and economic disruptions can also arise. With the emergence of modern technologies, it is time to focus on innovative solutions to detect and control plant diseases.

Our project aims to showcase one such solution, which utilizes advanced machine learning techniques to classify plant diseases with high accuracy while aiding diagnose, treat, and manage them effectively. In addition to classifying plant diseases, our application has the capability to analyze a vast amount of data and provide users with useful insights in real-time.

To address these challenges, our team developed an innovative solution that utilizes machine learning techniques and artificial intelligence to classify 38 different plant diseases. Our application provides farmers, agronomists, and researchers with a powerful tool that can quickly and accurately identify plant diseases and provide appropriate recommendations for management and control. By leveraging these advanced technologies, we believe that we can significantly improve global food production and help address some of the biggest challenges facing society today.

To overcome these challenges, we developed an application that utilizes machine learning techniques and artificial intelligence to classify plant diseases. The application can accurately identify 38 different plant diseases and provide appropriate measures to control and manage them.

The disease classification model was developed using Artificial Neural Network (ANN) and achieved an accuracy rate of over 90%. In addition, the application features an Open AI-powered Language Model (LLM) that aids disease diagnosis, treatment, and management.

Our project's aim was to develop a solution that could help farmers, agronomists, and researchers to quickly identify and control plant diseases, leading to higher crop yields and improved food production.

This project has the potential to revolutionize the agriculture industry by offering an easy-to-use and affordable tool that can be utilized by growers globally to help achieve sustainable food production. Moreover, the project's use of open-source tools ensures that our work is replicable and scalable, enabling a more significant impact across broader geographic areas.

We firmly believe that the application we have developed has the potential to deliver real benefits to the agriculture industry and overcome the challenges posed by plant diseases. By improving crop health and productivity, we hope that our solution can play a small role in contributing positively to global food security.

CHAPTER 16 LITERARATURE SURVEY II

Plant disease detection using computer vision and machine learning techniques has gained considerable attention in recent years due to its potential to improve crop yields and food security. A wide range of approaches have been proposed, relying on traditional image processing techniques as well as machine learning algorithms. While earlier works focused on detecting plant diseases based on leaf color, shape, and texture, recent studies have shifted towards capturing symptoms at the pixel level using deep learning models. Meanwhile, conversational agents or chatbots have emerged as a promising solution to improve the accessibility of information related to plant diseases, providing users with personalized and timely advice. In this section, we summarize existing works related to plant disease detection and conversational agents, highlighting their strengths and weaknesses, and providing context for our own contributions.

There were when many such research and applications which were already built regarding the subject but all of them generally covered the machine learning model development part which was trained on a huge set of images containing infected plants and a few of those resources that we came across while developing our application or given below.

- One such application was created by Ameya Upalanchi, who creatively utilized the power of TensorFlow framework to develop a model with the help of a convolutional neural network. The model was then embedded on an open-source application created using a Python library called Streamlit. This was my first inspiration on how to portray my idea and provide a better user interface, as well as how to combine and embed machine learning models and a user interface so that users could access the model's abilities.
- While exploring resources, I came across a similar resource based on the same idea, but the way that it utilized the application framework and integrated the machine learning model was completely different. This was an innovative way for me to gain insight into creating my own architecture.
- Another powerful resource I came across was a real project deployed for a similar use case, which was built using TensorFlow and a React application. This project was

deployed for user interaction for production purposes, and the developer creatively utilized both the machine learning model and web development framework.

While referencing all these resources, I was able to build my own application that embodies these concepts.

In addition to the various application-based resources, I also looked through some research papers to gain insights into creating my own machine learning model. These papers provided valuable information and data sets for creating machine learning models, including training data. The resources referred to by these papers were extremely valuable and helped in creating a robust machine learning model.

My application is divided into two significant parts - identifying plant diseases and providing a user interface for humans to interact with the application. Once the disease is identified, the user interface helps in designing an appropriate treatment plan for the plants, ensuring they thrive.

Incorporating large language models into the application became necessary to provide an intelligent and interactive experience for users. For this purpose, I integrated OpenAI's large language model with the help of an API into the application. This integration allowed me to create a more human-like assistant that could provide one-on-one interaction with the user, helping them get more insights on how to treat the plant after identifying the disease. With this integration, the assistant can recommend various effective and suitable treatment options that can help the plant recover quickly from the identified disease.

Integrating a large language model into my application was no easy feat. I had to go through several applications or sample applications that have been built since the time ChatGPT was first released. Through my research, I analyzed and built more sample applications that incorporate large language models. This research allowed me to create an application that uses OpenAI's large language model, with all the necessary knowledge incorporated into it.

I will provide links to these resources later, which can be useful for anyone looking to gain insights on integrating a large language model into their application.

All the research and development eventually culminated in the final output - the application that I aimed to create. My main objective was to provide humans or users with one-on-one interaction with someone who has information regarding the disease and use this interaction for the benefit of reducing errors in the farming process and increase yield throughout the growth of the crop.

There were countless resources and research papers that I had to go through to build this application. These resources provided me with valuable insights and gave me the necessary knowledge to create an application that not only identifies the plant disease but also provides suitable treatment options to help the plant recover. With all these in mind, I created the application with the sole purpose of helping farmers around the world to reap the benefits of modern agriculture.

CHAPTER 17 PROBLEM STATEMENT

Plant diseases have always posed a significant threat to agricultural productivity, affecting crop yields and quality, posing significant environmental threats, and subsequent social and economic disruptions. Identification and control of plant diseases have been a longstanding challenge for farmers, researchers, and agronomists. Traditional methods of plant disease identification are time-consuming, labor-intensive, and require trained professionals. These challenges result in farmers suffering from reduced yields and increased losses, food producers having to face financial loss, and consumers having to endure increased prices, and even food scarcities.

The traditional methods of plant disease identification have several limitations that include the difficulty of accurately identifying a specific disease, the lack of objective criteria in assessing disease presence, the expense and time required to hire trained experts, and the manual effort required to monitor the disease continuously. Furthermore, traditional disease identification/management methods present issues related to scale and disparity, leaving scores of farmers with insufficient or no access to knowledge concerning diseases, resulting in significant losses in the food supply chain.

To address these challenges efficiently, our application employs cutting-edge machine learning methods and artificial intelligence to solve the challenges posed by plant diseases, aiding diagnoses and managing them accurately and effectively. By utilizing our innovative solution, users can prevent crop damage and boost yields, leading to improved food production and security.

CHAPTER 18 PROPOSED METHODOLOGY

Our machine learning-powered plant disease classification application follows a three-stage approach: data acquisition, data preprocessing, and training the classification models.

- **Data acquisition**

Our dataset consisted of a collection of diseased plant images acquired from multiple sources, including open-source repositories and partnerships with plant pathology research labs. The dataset included images of 38 unique plant diseases.

- **Data preprocessing**

The data was thoroughly preprocessed through various transformations, including normalization, resizing, and augmentation. We introduced various types of noise in the data, simulated environmental variance and applied different styles of image processing, and mixing. Before inputting the images for the classifier, we performed color space conversion from Red Green Blue (RGB) to Hue Saturation Value (HSV).

- **Training the classification models**

The input images were classified as being diseased or healthy using a binary classification model, followed by predicting the disease present in the image using a multi-classifier model. We trained and tested the model on an GPU on cloud and used the categorical cross-entropy loss function to calculate the loss during the training process.

Evaluation of the classification model was conducted on a separate test dataset, which is distinct from the training dataset. Performance metrics of the classifier were assessed on metrics such as Precision, Recall, and F1-score to evaluate the model's capabilities better.

Further, we integrated our classification model with the GPT Language model (Open AI); this integration allowed our application to provide relevant information about the identified plant diseases. Using the Open AI API, the application provides users with adequate information to help diagnose, treat and manage plant diseases. We integrated our classification model with ChromaDB, an open-source document-based database system, to store and manage relevant information about plant diseases that can be useful for further analysis by the Language model. This information includes scientific research papers, field notes, recommended treatments, and other relevant data. We deployed our application interface using Streamlit, an open-source Python framework for building web applications.

Our approach encompasses data acquisition, data preprocessing, training and evaluation of a Classification model's performance, deploying it using Streamlit and integrating it with ChromaDB to provide contextualized data for easy visualizations, insights, and recommendations while using the Language Model to assists the users on plant disease diagnosis, treatment, and management.

So, on a high level my whole application is divided into two functionalities:

- ❖ One is where my application is identifying the disease which is present in the plant specimen.
- ❖ The other is a large language model which provides human-like interaction with the user so that user can look for assistance and treatment regarding his requirements.

In the next few chapters, we will be explaining you these two parts in depth.

PROPOSED ARCHITECTURE

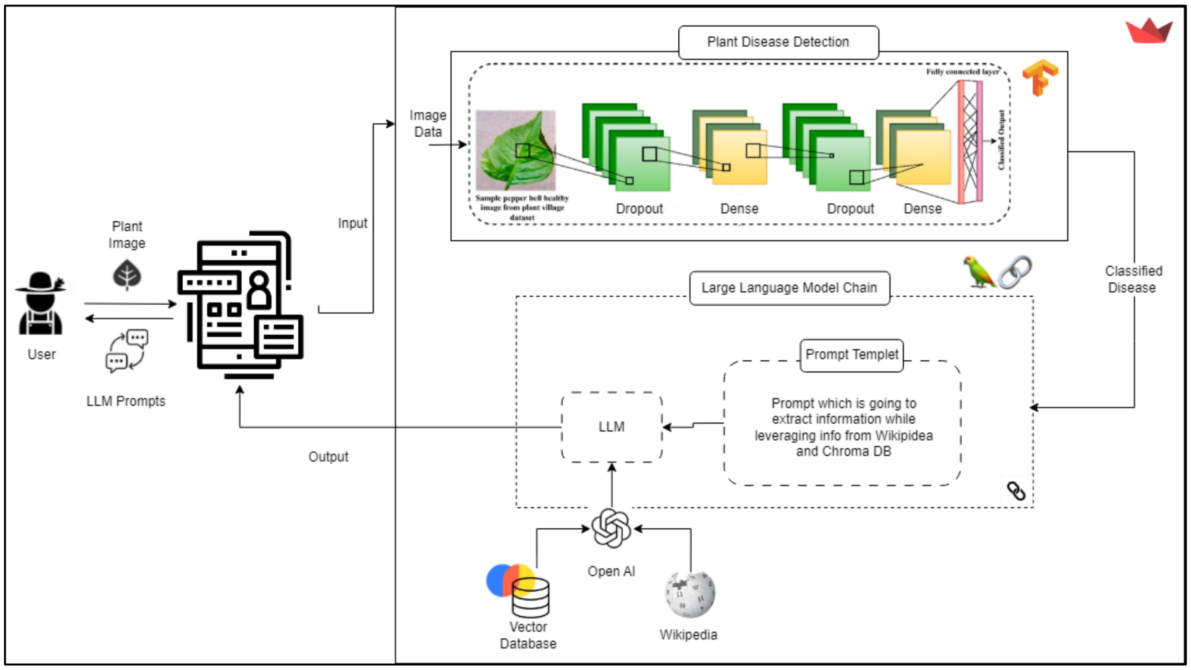


Figure 42 Schematic Diagram of the Proposed Architecture.

CHAPTER 19 SOFTWARE AND HARDWARE REQUIREMENTS

In this section, we will outline the software and hardware requirements which were used while building this project and are also required for running this project. Having a clear understanding of these requirements is essential to ensure the software operates efficiently and effectively. We will detail the minimum specifications for the hardware and software, as well as recommended specifications for optimum performance. This information will be useful for users who want to ensure they have adequate hardware and software before installing the software or application.

For more information on environment setup refer to Appendix I.

The following software and hardware requirements are recommended for running the machine learning project developed using TensorFlow, Python, LangChain, and Streamlit with GPU acceleration:

19.1 Software Requirements

- ❖ Python 3.6 or higher
- ❖ TensorFlow 2.0 or higher
- ❖ LangChain
- ❖ Streamlit
- ❖ Chromadb
- ❖ CUDA 10 or higher (if using NVIDIA GPU)
- ❖ cuDNN (if using NVIDIA GPU)

And many more are there, which will be present with the project repository.

19.2 Hardware Requirements

- ❖ NVIDIA GPU with a minimum of 4 GB of VRAM
- ❖ A CPU with at least 4 cores and 4 threads or higher
- ❖ At least 8 GB of RAM or higher

- At least 100 GB of available disk space

Note: The specific software and hardware requirements may vary depending on the size of the data, the complexity of the machine learning models, and the specific use case. It is recommended to refer to the documentation and guidelines provided by each library and to consult with the hardware and software vendor for more detailed requirements.

These requirements should be considered as a guideline for the recommended hardware and software required to run the machine learning project. Adjustments may be necessary depending on the specific use case and the size and complexity of the data being used. It is important to review the documentation and recommendations provided by the libraries used and the hardware and software vendors to ensure optimal performance of the machine learning project.

CHAPTER 20 PLANT DISEASE CLASSIFICATION MODEL

Image classification is a technique used in computer vision to classify an image into one or more predefined categories based on the image content. This is done by extracting relevant features from the image and training a machine learning model on a dataset of labeled images to recognize patterns and make accurate predictions.

20.1 USAGE OF IMAGE CLASSIFICATION

In the context of plant disease classification, image classification can be used to automatically detect and diagnose plant diseases based on visual symptoms captured in images of leaves, stems, or other plant parts. This can have several benefits, including:

- ❖ Early detection of plant diseases: By automatically analyzing images, plant diseases can be detected early before they develop into full-blown infestations.

- ❖ Quick and accurate diagnosis: Image classification models can make diagnoses on a large scale, providing accurate results in a matter of seconds and reducing the need for time-consuming manual inspections.

- ❖ Reduced pesticide use: With early detection and accurate diagnoses, farmers can reduce their use of pesticides, saving money and reducing the negative impact of agriculture on the environment.

20.2 PROCESS OF IMAGE CLASSIFICATION

- ❖ <u>Collecting a dataset of images</u>: This dataset should include images of healthy plants as well as plants affected by different diseases.

- ❖ <u>Image pre-processing</u>: The images need to be normalized and preprocessed to ensure consistent quality and clarity.

- ❖ <u>Feature extraction</u>: Features are extracted from images by identifying patterns, textures, and colors that can be used to distinguish healthy plants from those affected by diseases.

❖ Training a model: A machine learning model is trained using labeled images to learn how to accurately classify plants as healthy or diseased.

❖ Validation: The trained model is tested on a validation set of images to ensure its accuracy and improve it if necessary

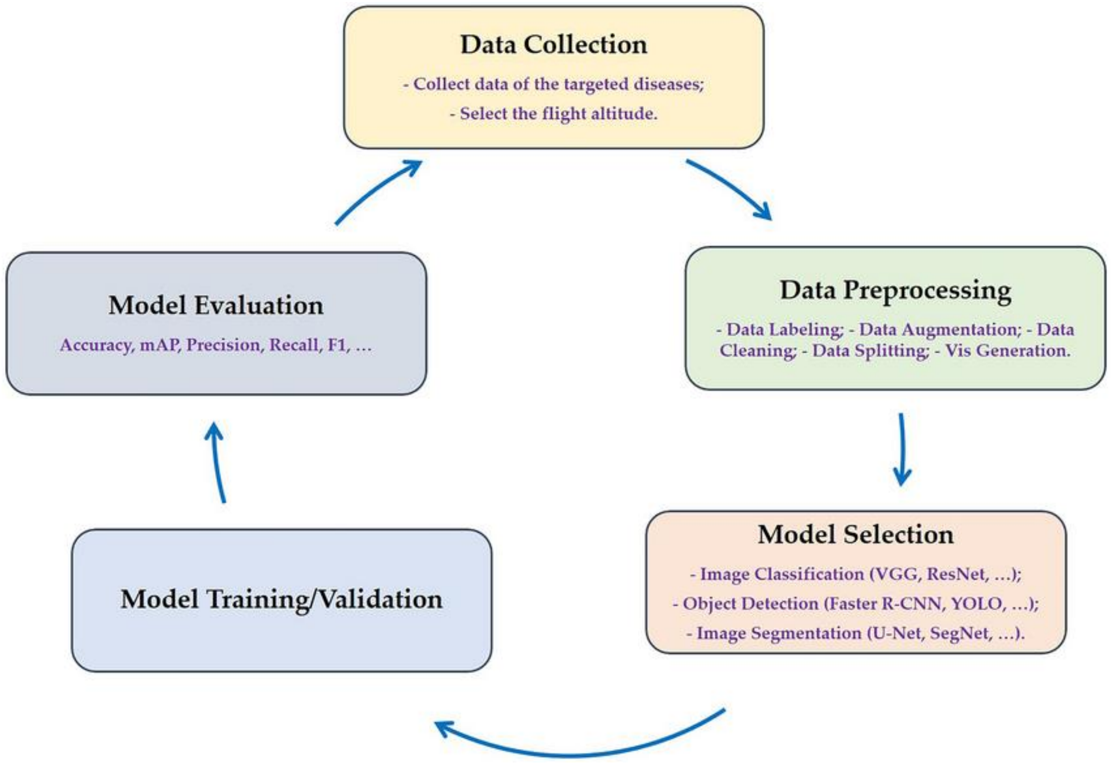

Data Collection
- Collect data of the targeted diseases;
- Select the flight altitude.
Data Preprocessing
- Data Labeling; - Data Augmentation; - Data Cleaning; - Data Splitting; - Vis Generation.
Model Selection
- Image Classification (VGG, ResNet, ...);
- Object Detection (Faster R-CNN, YOLO, ...);
- Image Segmentation (U-Net, SegNet, ...).
Model Training/Validation
Model Evaluation
Accuracy, mAP, Precision, Recall, F1, ...

Figure 43 Machine Learning Lifecycle for plant disease detection

20.3 DATASET

The PlantVillage dataset is a collection of images of diseased and healthy plant leaves and the associated metadata such as crop type, disease type, and severity. It was created by researchers at Penn State University to aid in the development of machine learning algorithms to detect plant diseases.

The dataset contains over 54,000 images of plant leaves from 26 different crop species, including tomatoes, potatoes, apples, and grapes. These images are divided into three categories: healthy, diseased, and diseased with a severe symptom. The plant diseases represented in the dataset include bacterial spot, late blight, and powdery mildew, among others.

Each image in the dataset comes with detailed metadata to provide context and additional information for machine learning algorithms. This metadata includes information about the crop type, disease type, and severity. Additionally, each image has been labeled by expert plant pathologists to ensure accuracy and consistency across the dataset.

The PlantVillage dataset is a valuable resource for researchers and developers working on the detection and diagnosis of plant diseases using machine learning algorithms. It provides a comprehensive collection of high-quality images and metadata that can be used to train and test algorithms for accuracy and effectiveness.

Overall, image classification can be a powerful tool for plant disease classification as it allows farmers and researchers to identify and manage plant diseases, leading to healthier crops and better yields quickly and accurately.

Examples of the images and different phenotypes are given in Table 1.

	Fungi	**Bacteria**	**Mold**	**Virus**	**Mite**	**Health y**
Apple (3172)	*Gymnosporangi u m juniperi-virginianae (276) Venturia insequalis (630) Botryospaeria obtuse (621)*					**(1645)**
Blueberry (1502)						**(1502)**
Cherry (1906)	*Podosphaera spp (1052)*					**(854)**
Corn (3852)	*Cercospora zeae-maydis (513) Puccinia sorghi (1192) Exserohilum turcicum (985)*					**(1162)**
Grape (4063)	*Guignardia bidwellii (1180) Phaeomoniella--spp. (1384) Pseudocercospo r a vitis (1076)*					**(423)**
Orange (5507)	*Candidatus Liber ibacter (5507)*					
Peach (2657		*Xanthomona s campestris (2291)*				**(360)**
Bell Pepper (2475)		*Xanthomona s campestris (997)*				**(1478)**
Potato (2152)	*Alternaria solani (1000)*		*Phytophthor a Infestans (1000)*			**(152)**

Raspberry (371)						**(371)**
Soybean (5090)						**(5090)**
Squash (1835)	*Erysiphe cichoracearum / Sphaerotheca fuliginea (1835)*					
Strawberry (1565)	*Diplocarpon earlianum (1109)*					**(456)**
Tomato (18,162)	*Alternaria solani (1000)* *Septoria lycopersici (1771)* *Corynespora cassiicola (1404)* *Fulvia fulva (952)*	*Xanthomonas campestris pv. Vesicatoria (2127)*	*Phytophthora Infestans (1910)*	*Tomato Yello Leaf Curl Virus (5357)* *Tomato Mosaic Virus (373)*	*Tetranychus urticae (1676)*	**(1592)**

Table 1 Description of images present in the dataset

This whole data set contains various classes of diseases for a different types of plants and even in a separate plant there are different category of diseases that might affect the plant this is just a general data set which is for everyone for research purposes this is definitely not something that you can put to production or you can just provide it to someone so this is something which can help me or anyone to build a base application on the top of which other images can be inserted for training the model and making it available for a more different kind of use cases.

Some sample images are also present below.

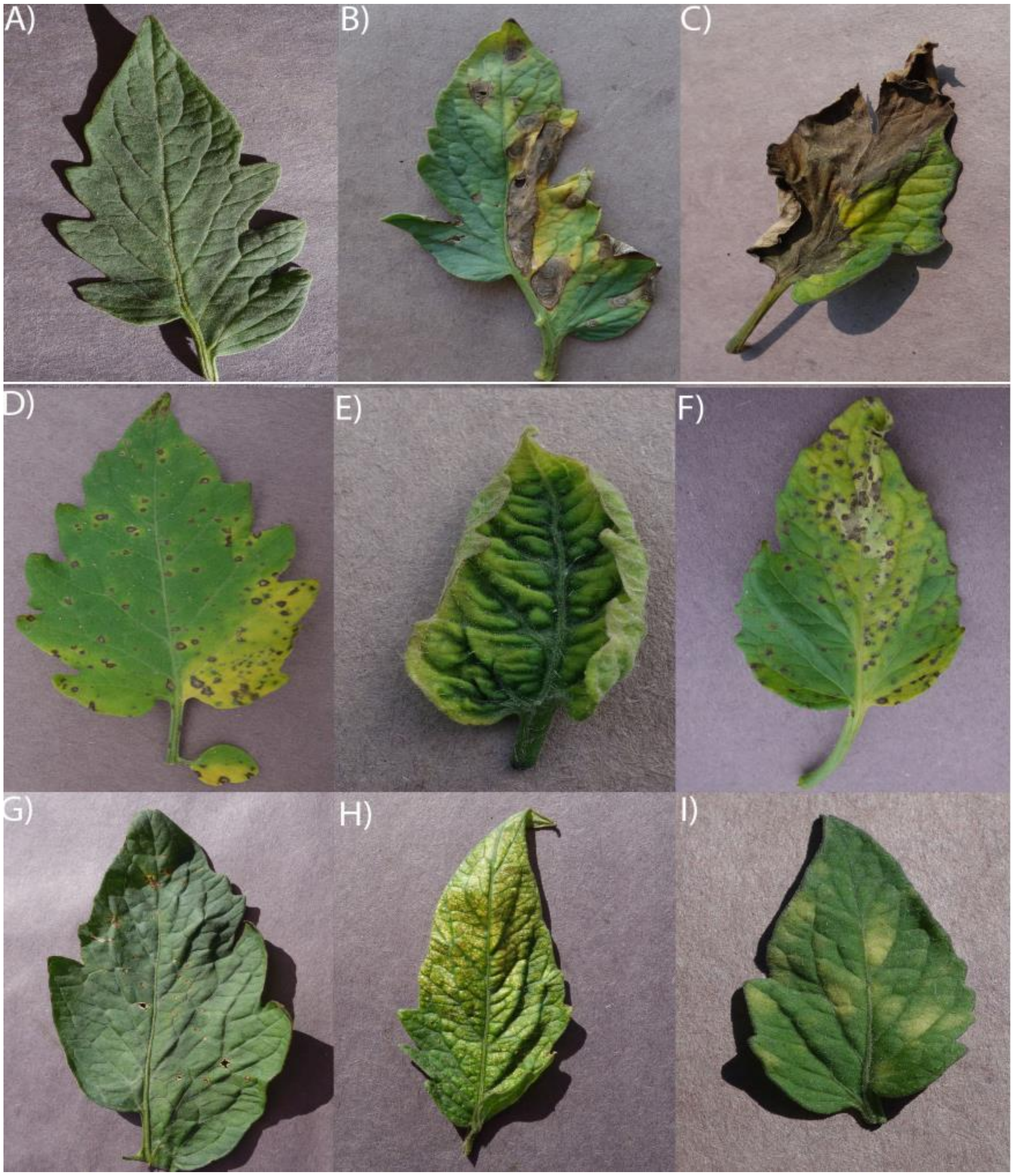


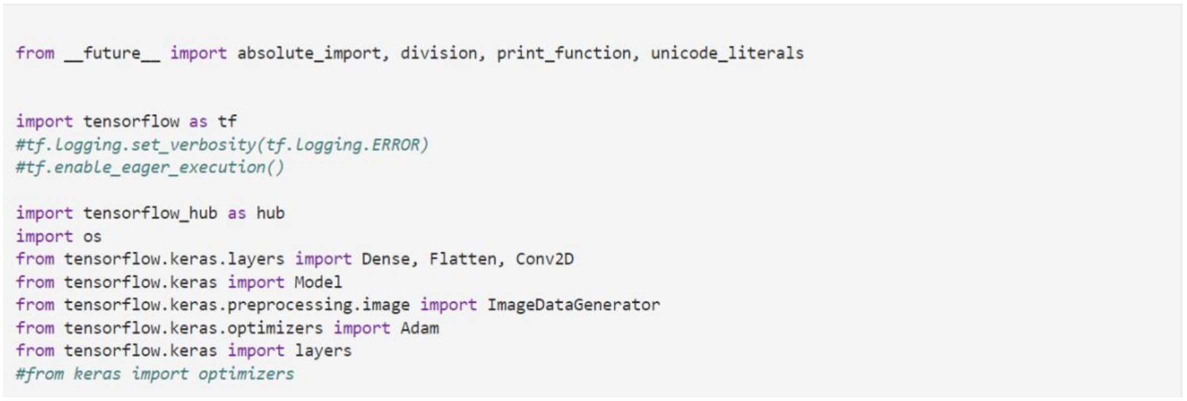
Figure 44 Examples of different phenotypes of tomato plants. A) Healthy leaf B) Early Blight C) Late Blight (Phytophthora Infestans) D) Septoria Leaf Spot (Septorialycopersici) E) Yellow Leaf Curl Virus (Family Geminiviridae genus Begomovirus) F) BacterialSpot (Xanthomonas campestris pv. vesicatoria) G) Target Spot (Corynespora cassiicola) H) Spider Mite (Tetranychus urticae) Damage

This data set provides us with very helpful information so now it's time to import the data set into our environment so that we can work with it perform different transformation on data according to our requirements.

Dataset can be found on this URL:

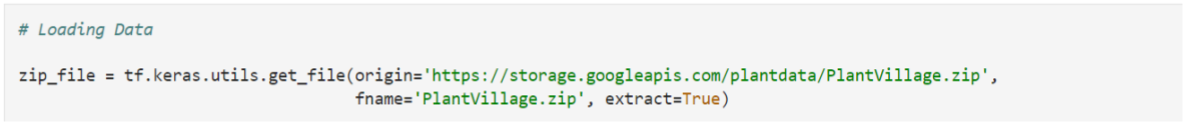
https://storage.googleapis.com/plantdata/PlantVillage.zip

Now what we're going to do we are going to load out data set into our environment so that we can use it for training a model testing a model and perform validation on it. But before doing so we need to import the required libraries to perform all the actions that we are going to perform throughout this process.



```
from __future__ import absolute_import, division, print_function, unicode_literals


import tensorflow as tf
#tf.logging.set_verbosity(tf.logging.ERROR)
#tf.enable_eager_execution()

import tensorflow_hub as hub
import os
from tensorflow.keras.layers import Dense, Flatten, Conv2D
from tensorflow.keras import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import layers
#from keras import optimizers
```

Here we are loading the required libraries for the performing all kind of operations or a data so that we can train our model to achieve our desired output, one such library is tensor flow detail about this library and including other libraries will be provided in the appendix I please refer to it in case if you are interested in what these libraries are or how I have set up this whole environment. When's the importing of libraries is completed it's time to download the data set and perform some splitting or creating training set and validation set.



```
# Loading Data

zip_file = tf.keras.utils.get_file(origin='https://storage.googleapis.com/plantdata/PlantVillage.zip',
                                   fname='PlantVillage.zip', extract=True)
```

The above snippet of code is a going to download my PlantVillage data from are you URL which is mentioned in the code, and I am using uh Tensorflow Keras library for getting that file from the given URL.

When the data is downloaded now it is time to create validation training and test data set meaning what we are going to do we are going to divide our whole image data set which contains around 54,000 number of images into three segments the idea of dividing the whole data set into three segments is that when we need to train our whole model we need our data set which can be used to to or tune the parameters of our machine learning model so for that we are going to use a training data set and next what we need we need to check if the training has been performed according to the requirement meaning if the model is fulfilling the requirements or not for that what we does is we create a small validation test which is used for just validating that the required accuracy of our model is there or not and finally what we do is we test on machine learning model on an unseen data set which is referred to as test data set so this is the ideology of dividing a whole data set into three separate parts generally what are the best practices are in training data set and test data set splitting we generally divided into the ratio of 80% of training data set and 20% of training data set then further we divide our training data set into two separate parts of which a small part will be given to validation set and the rest will be used for training of our data model.

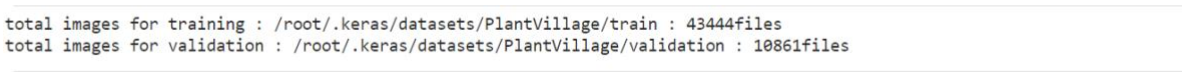
```
# Prepare training and validation dataset

data_dir = os.path.join(os.path.dirname(zip_file), 'PlantVillage')
train_dir = os.path.join(data_dir, 'train')
validation_dir = os.path.join(data_dir, 'validation')
```

Here we are defining the two directories of our train and validation data set. Now to perform the splitting there are various ways to do that one way is to just call a function from the library and another that I'm using is just creating a function which can perform the splitting for me so here the snippet of the code which is performing that operation.

```
import time
import os
from os.path import exists

def count(dir, counter=0):
    "returns number of files in dir and subdirs"
    for pack in os.walk(dir):
        for f in pack[2]:
            counter += 1
    return dir + " : " + str(counter) + "files"
```

```
print('total images for training :', count(train_dir))
print('total images for validation :', count(validation_dir))
```

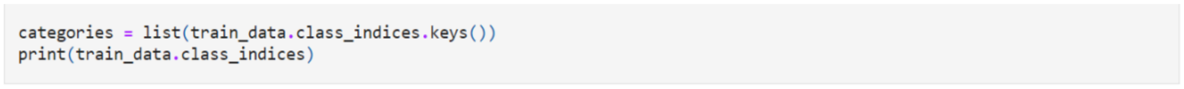
As we can see the function is going to take a directory as an argument and then perform the division or you can say splitting of the whole number of images which is present in a particular directory and then later snippet what I have done I have printed the number of images which are going to be used for training and validation while calling the function which is defined above so the output of the print function is this which is depicting the number of images which are assigned for training and validation set.



```
total images for training : /root/.keras/datasets/PlantVillage/train : 43444files
total images for validation : /root/.keras/datasets/PlantVillage/validation : 10861files
```

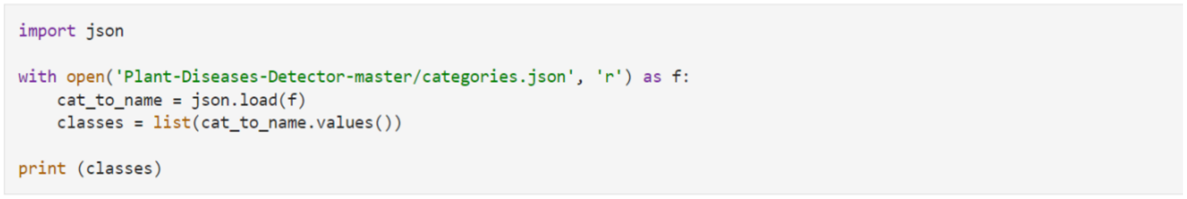
As you can see there are around 43,000 files which are present for training our machine learning model now what we need next is the classes we have in our data set as the kind of learning we are using in this application is supervised learning. For more information on different kinds of learning and an overview of machine learning and deep learning please refer to the appendix we have provided at the end of this documentation.

Now uh as I mentioned before we are using supervised learning the idea of supervised learning is that we have some features and labels what are features? Features are nothing but the input variable that our model is going to take and labels are the output that our model is supposed to be given so basically in supervised learning we train our model to adapt to the relationship between the input variables that our model should receive and the output that it should give there is a mathematical model which is going to be created by performing various iterations over the data set by performing different kinds of shuffles so that the model can understand what is the relationship between input and output and can adapt to it can learn it and after learning it model will be able to perform those classifications without going through the same process again this kind of learning is referred to as a supervised learning so now what our aim is we have to provide different classes to which we want our model to pertain input variable of our application will be images and what we want our model is to perform a classification depending on the disease present in the image.



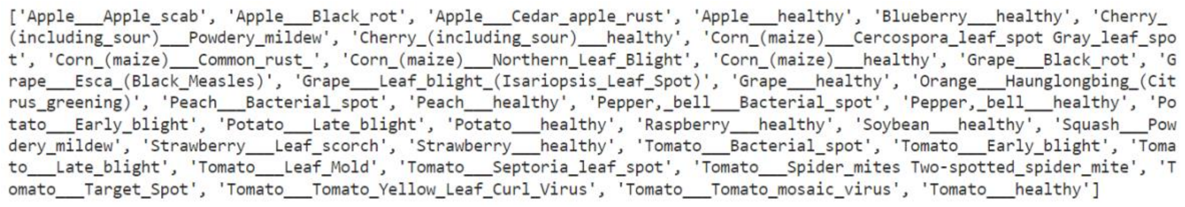
```
categories = list(train_data.class_indices.keys())
print(train_data.class_indices)
```

In the above snippet what we are doing is we are creating classes for the diseases that are present in our data set this is one way of doing it where we are just extracting the different classes we have present in our data set another way that I have those in my application development is just extracting it from a GitHub repo which already contain adjacent format category files containing all 38 categories.



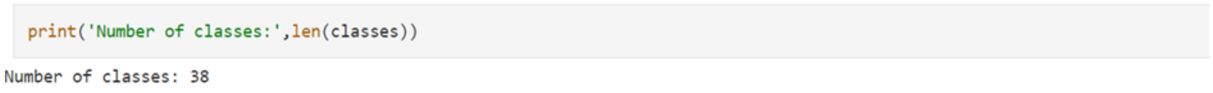
```
import json

with open('Plant-Diseases-Detector-master/categories.json', 'r') as f:
    cat_to_name = json.load(f)
    classes = list(cat_to_name.values())

print (classes)
```

The above code is going to open the categories file sentence the different categories or the labels we want to train our model for. All the above code depends on which directory you are working on. if the name of the directory is same as mentioned in the code and you execute the same code in your systems you will get this result.



```
['Apple___Apple_scab', 'Apple___Black_rot', 'Apple___Cedar_apple_rust', 'Apple___healthy', 'Blueberry___healthy', 'Cherry_
(including_sour)___Powdery_mildew', 'Cherry_(including_sour)___healthy', 'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spo
t', 'Corn_(maize)___Common_rust_', 'Corn_(maize)___Northern_Leaf_Blight', 'Corn_(maize)___healthy', 'Grape___Black_rot', 'G
rape___Esca_(Black_Measles)', 'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)', 'Grape___healthy', 'Orange___Haunglongbing_(Cit
rus_greening)', 'Peach___Bacterial_spot', 'Peach___healthy', 'Pepper,_bell___Bacterial_spot', 'Pepper,_bell___healthy', 'Po
tato___Early_blight', 'Potato___Late_blight', 'Potato___healthy', 'Raspberry___healthy', 'Soybean___healthy', 'Squash___Pow
dery_mildew', 'Strawberry___Leaf_scorch', 'Strawberry___healthy', 'Tomato___Bacterial_spot', 'Tomato___Early_blight', 'Toma
to___Late_blight', 'Tomato___Leaf_Mold', 'Tomato___Septoria_leaf_spot', 'Tomato___Spider_mites Two-spotted_spider_mite', 'T
omato___Target_Spot', 'Tomato___Tomato_Yellow_Leaf_Curl_Virus', 'Tomato___Tomato_mosaic_virus', 'Tomato___healthy']
```

This is the list containing different classes of disease was Internet data set and against which we are going to change our model. One question which may arise as that why we have saved all these categories in Jason format, as json format is structured in a way of a dictionary data structure which is supported by python it makes it easy for us to work with this kind of key value pair data very easily.



```
print('Number of classes:',len(classes))
```

```
Number of classes: 38
```

On running the above snippet of code, I will be getting the number of classes which are present in the data set which I have extracted from the categories JSON file.

Now we have extracted the data from the data source and saved it into our own premise system or wherever you are working, and as we have also loaded the whole data set into our environment of Python so that we can access it and perform all kind of operations like model training data transformation and much more and now our next step these two transform this whole data set depending on different requirements so that we can make our model more efficient.

20.4 DATA PREPROCESSING

Data preprocessing is an essential step in image processing and computer vision. This process involves transforming raw image data into a format that can be utilized by machine learning algorithms. Data preprocessing for images typically includes tasks such as resizing, normalization, cropping, and data augmentation. Here's a detailed description of the common steps involved in image data preprocessing:

1. Image Resizing

The first step in image data preprocessing is to resize the images. This step is important because it helps to standardize the size of the images in the data set, which is essential for effective machine learning. Different algorithms have different requirements for image size, and resizing the images can be helpful in reducing the computational cost.

2. Normalization

Normalization is the process of scaling the pixel values in the images to a fixed range. This step is usually done to ensure that the pixel values lie between 0 and 1, which makes it easier for the machine learning algorithm to learn and identify patterns in the data.

3. Cropping

Cropping involves removing unwanted parts of the image that are not relevant to the task at hand. This step can help in reducing the computational cost and reduce the variability of the images in the data set.

4. Data Augmentation

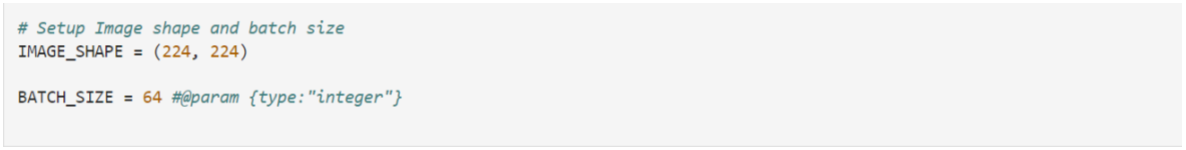
Data augmentation is a technique used to artificially increase the size of the data set by creating new images from the original images. This step is particularly important when the data set is small or imbalanced. Common data augmentation techniques include flipping, rotation, zooming, and translation.

5. Quality Control

Finally, quality control is essential to ensure that the data set is clean and free from any errors or issues that may affect the performance of the machine learning algorithm. This step involves inspecting the images to ensure that they are labeled correctly and that there are no missing or corrupted images.

Overall, image data preprocessing is an essential step in preparing image data for machine learning. It is important to carefully consider the specific needs of the algorithm being used and adjust the preprocessing steps accordingly to ensure optimal performance.

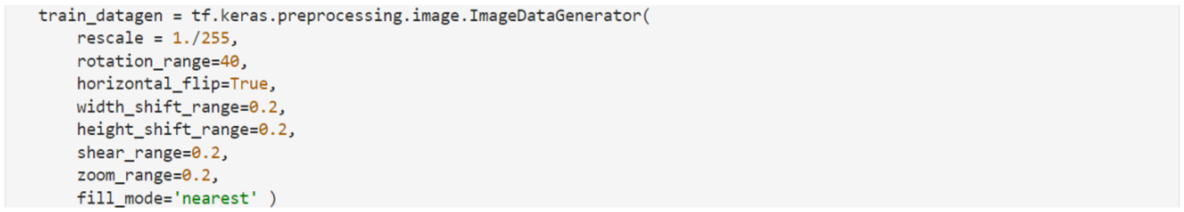
As we may have different sizes of images in our data set or even in the case of user consider it this way if a user brings in an image which doesn't matter size or the number of input variables or the number of parameters a model can take it can cause a conflict so in the pipeline of our model processing we need to introduce a step where the image size is needed to be resized, the below snippet of code is a setting the standard size of image which model is going to take as input.



```
# Setup Image shape and batch size
IMAGE_SHAPE = (224, 224)

BATCH_SIZE = 64 #@param {type:"integer"}
```

As we can see the size of the image will be 224 * 224 which gives use 50,176 number of pixels, and this is going to be the size of the first layer of our neural network meaning this is several input variables are neural network is going to take as an input.

As now we have our two main datasets fun for training our model and another for providing the validation for the model, so we need to perform various data preprocessing tasks on both at first the below snippet of code is going to show you the different tasks that are performed on all the images of the training data set.



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

Here we are using a library called Keras and there are various methods for performing preprocessing on images, and we are using ‘ImageDataGenerator()’ method for performing the different operations on the images for preprocessing.

Now let's dive into different operations which are being performed on the single image in the whole data set.

- ❖ Image rescaling is an important image pre-processing technique used to standardize image sizes before they are input into a machine learning algorithm for training or testing. This technique can be used to resize all images to a specific size, or to make them proportional to one another. There are several reasons why image rescaling is necessary in image pre-processing:
    - Standardization of Image Sizes: Different images in a dataset may have different sizes and resolutions, which can make it difficult for machine learning algorithms to effectively learn from the data. Rescaling images to a specific size ensures that all images in a dataset have the same dimensions, making it easier for the algorithm to learn from them.
    - Consistency: Large variations in image sizes can lead to inconsistent performance by the algorithm, making it difficult to identify patterns in the data. Rescaling images helps ensure consistency in the image sizes, thereby improving the accuracy of the algorithm.
    - Reducing Computational Cost: Large images take longer to process, thereby increasing the computational cost of running the algorithm. Rescaling images to a smaller size can reduce this computational cost, enabling the algorithm to run faster and more efficiently.
- ❖ Some common techniques used in rescaling images include:
    - Scaling by a fixed ratio: Images can be scaled up or down by a fixed percentage to achieve a standard size.
    - Scaling to a specific size: All images in a dataset can be rescaled to a specific size (such as 256x256 or 512x512 pixels) to ensure consistency.
    - Scaling to a proportional size: Images can be rescaled so that images of the same proportion have the same size. This technique maintains the aspect ratio of the image while standardizing the size.

Rescaling images is an important image pre-processing task that helps standardize image sizes, ensures consistency, and reduces computational cost. It helps machine learning algorithms to learn better features from the data and improve the overall accuracy of the models trained on the image datasets.

❖ Rotation is a common image pre-processing technique used to augment and expand image datasets for training machine learning models. By rotating images within a certain range, we can create new images with different angles that can help the model generalize better and improve its performance. In image pre-processing, the rotation range refers to the degree range that the images will be rotated. This range can be specified by a minimum and maximum degree or as a single value, depending on the application. Some common rotation ranges include:

- Fixed Rotation: This involves rotating all images in a dataset by a fixed angle (e.g., 90 degrees, 180 degrees, etc.). This technique is useful when the images have a specific orientation, such as landscape or portrait.
- Random Rotation: This involves rotating each image in the dataset by a randomly generated angle within a specified range, typically between 0 to 360 degrees. This technique creates new images with different angles, expanding the dataset and improving the performance of machine learning models.
- Limited Rotation: This technique involves rotating images by a specified range, such as -15 to +15 degrees, to create new images that are similar to the original but with slight variations. This technique can improve the model's ability to detect objects at different angles.

Rotation is an important image pre-processing technique used to augment and expand image datasets for training machine learning models. By rotating images within a certain range, we can create new images with different angles that can help the model generalize better and improve its performance.

❖ Horizontal flip is a common image pre-processing technique that involves generating a mirrored image of an original image along its vertical axis. It is a type of data augmentation that can be used to generate new images for a dataset, which can help to improve the accuracy and robustness of machine learning models. In horizontal flip, the image is flipped along the horizontal axis, such that the left side of the original image becomes the right side of the flipped image, and the right side of the original image becomes the left side of the flipped image. Some of the benefits of using horizontal flip as an image pre-processing task include:

- Creating New Images: By flipping images horizontally, it generates new images that are similar to the original image, but with a slightly different appearance. This technique can artificially increase the size of the dataset and help to improve the performance of the machine learning model.
- Enhancing the Robustness of the Model: By including flipped images in the dataset, the machine learning algorithm can be trained on different orientations of the object, and improve the model's ability to recognize the object from different perspectives. This is particularly helpful when images have objects with varying orientations such as text, traffic signs, and so on.
- Reducing Overfitting: Overfitting occurs when a machine learning model is trained too well on the training data and performs poorly on test data. Using techniques like horizontal flip can help the model generalize better by exposing it to more diverse images during training, thereby reducing overfitting.

Horizontal flip is typically a simple yet effective image pre-processing technique that is easy to implement for most machine learning applications. It may be used alone or in combination with other pre-processing techniques like rotation, normalization, and cropping.

❖ Width shift is a common image pre-processing technique used in data augmentation to generate new images from an existing dataset by shifting the image horizontally along the x-axis. The width shift range refers to the range within which the images can be shifted or translated horizontally. In width shift, the image can be shifted left or right by a specified distance or a randomly generated distance within the specified range. This technique simulates images of objects that are partially captured or objects that are not in the center of the image. Some of the benefits of using width shift range as an image pre-processing task include:

    - Creating New Images: By shifting images horizontally, we can create new and diverse images that help to increase the size of the dataset and enable the model to learn more robust patterns. Such images can help the machine learning algorithm better recognize objects in different positions.

- Improving Generalization: Training a machine learning model on a dataset with data translated within a range can result in the model learning features that are invariant to object position. Therefore, it can more efficiently detect and differentiate the object even if it's partially visible.
- Reducing Overfitting: Like other image pre-processing techniques, using the width shift range in data augmentation helps to reduce overfitting by artificially increasing the diversity of the dataset and preventing unnecessary memorization of the training data.

The width shift range is specified as a fraction of the image dimensions, and the actual translation performed is chosen randomly within this range. Typically, a value between 0.2 and 0.3 of the image dimensions is a good starting point, but the optimal value for this parameter may vary depending on the specific use case.

Overall, width shift range is a useful technique for image pre-processing and data augmentation as it can help to increase dataset diversity, prevent overfitting, and improve the accuracy of machine learning models.

❖ Height shift is a common image pre-processing technique used in data augmentation to generate new images from an existing dataset by shifting the image vertically along the y-axis. The height shift range refers to the range within which the images can be shifted or translated vertically.In height shift, the image can be shifted up or down by a specified distance or a randomly generated distance within the specified range. This technique simulates images of objects that are captured at different heights or heights that are distorted by perspective. Some of the benefits of using height shift range as an image pre-processing task include:

  - Creating New Images: By shifting images vertically, we can create new and diverse images that help to increase the size of the dataset and enable the model to learn more robust patterns. Such images can help the machine learning algorithm better recognize objects in different positions.

- Improving Generalization: Training a machine learning model on a dataset with data translated within a range can result in the model learning features that are invariant to object height. Therefore, the model can more efficiently detect and differentiate the object even if it's captured from different heights.
- Reducing Overfitting: As described previously, using the height shift range in data augmentation helps to reduce overfitting by artificially increasing the diversity of the dataset and preventing the model from memorizing the training data.

The height shift range is specified as a fraction of the image dimensions, and the actual translation performed is chosen randomly within this range. Typically, a value between 0.2 and 0.3 of the image dimension is a good starting point, but the optimal value for this parameter may vary depending on the specific use case.

Overall, height shift range is a useful technique for image pre-processing and data augmentation as it can help to increase dataset diversity, prevent overfitting, and improve the robustness and accuracy of machine learning models.

❖ Shear range is a common image pre-processing technique used in data augmentation to generate new images from an existing dataset by shearing the image. Shear refers to the deformation or distortion of the shape of an object along an axis that is parallel to its face. In image processing, shearing refers to a transformation that slants the image along a specified axis.In shear range, the image is sheared at a random angle within a specified range along horizontal or vertical axes. This technique simulates the perspective distortion that is common in images captured from different angles.Some of the benefits of using shear range as an image pre-processing task include:
    - Creating New Images: By shearing images along horizontal or vertical axes, we can create new and diverse images that help to increase the size of the dataset and enable the model to learn more robust patterns. Such images can help the machine learning algorithm better recognize objects from different perspectives.
    - Improving Generalization: Training a machine learning model on a dataset with data sheared within a range can result in the model learning features that are invariant to perspective distortion.

- Reducing Overfitting: As with other image pre-processing techniques, using the shear range in data augmentation helps to reduce overfitting by artificially increasing the diversity of the dataset.

The shear range is specified as an angle range within which the image will be sheared. Typically, a range between 0 and 0.3 radians is a good starting point, but the optimal value for this parameter may vary depending on the specific use case.

Overall, shear range is a useful technique for image pre-processing and data augmentation as it can help to increase dataset diversity, prevent overfitting, and improve the robustness and accuracy of machine learning models.

❖ Zoom range is a common image pre-processing technique used in data augmentation to generate new images from an existing dataset by zooming the image. This technique can be used to simulate the images of objects captured at different distances and can help to improve the performance and generalization of machine learning models.In zoom range, the image is zoomed in or out by a specified percentage or a randomly generated percentage within a specified range. This technique can help to artificially increase the size of the dataset and enable the model to better recognize objects at various distances.Some of the benefits of using zoom range as an image pre-processing task include:

  - Creating New Images: By zooming the images, we can create new and diverse images that help to increase the size of the dataset, augment the training data, and improve the performance of the machine learning models.
  - Improving Generalization: Training a machine learning model on a dataset with data zoomed within a range can result in the model learning features that are invariant to object size or distance, thereby improving the model's ability to recognize objects at various zoom levels.
  - Reducing Overfitting: Using the zoom range in data augmentation can help to reduce overfitting by artificially increasing the diversity of the dataset and preventing unnecessary memorization of the training data.

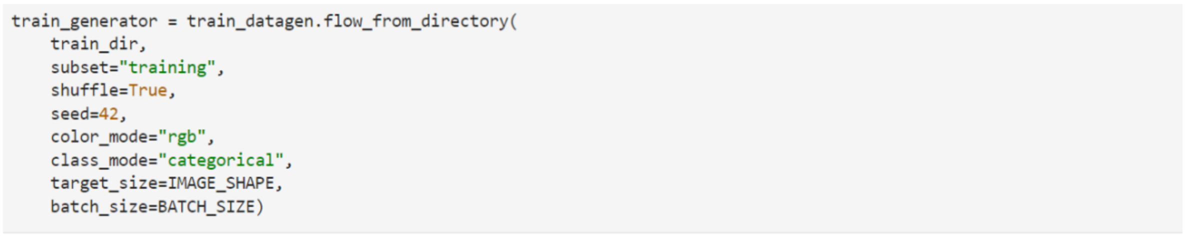
The zoom range is specified as a percentage range within which the image will be zoomed. Typically, a range between 0 and 0.3 is a good starting point, but the optimal value for this parameter may vary depending on the specific use case.

Overall, zoom range is a useful technique for image pre-processing and data augmentation as it can help to increase dataset diversity, prevent overfitting, and improve the accuracy and robustness of machine learning models.

- ❖ Fill mode is a common image pre-processing technique used to fill in the empty spaces that might appear in an image after pre-processing tasks like rotation or resizing. When an image is rotated or resized, the empty pixels that are created can cause noise, outliers, or unrealistic features that might hinder the learning capability of machine learning algorithms. Fill mode is used to fill these empty pixels in different ways, depending on the specific use case. There are several fill modes that are commonly used in image pre-processing, but the most common ones include crop, reflect, and constant methods.
    - Crop Method: This method involves simply cropping the image to remove the empty pixels. This technique is useful when the original image contains sufficient data for the machine learning task and the empty pixels can be ignored.
    - Reflect Method: This method involves mirroring the available pixels to fill in empty pixels. This technique is useful when the image borders contain important information, and the machine learning algorithm should be able to learn from it.
    - Constant Method: This method involves filling the empty pixels with a predefined constant value. This technique is useful when preserving the shape of the image is very important, and the best practice is to keep the image size and shape fixed across the dataset.

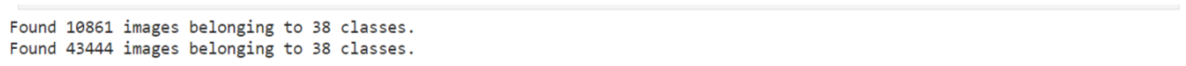
Using the fill mode in the image pre-processing task is essential to maintain the integrity and quality of the images while ensuring that the data is maximally informative for the machine learning algorithms. It helps prevent overfitting by augmenting the data without unnecessary information and helps machine learning algorithms learn important features and patterns from the images.

After finding the different transformation that we have to perform on the image, they are not exactly transformation more like augmentation which are being performed in image for making the disease detection more difficult for our machine learning model so that it can adapt to different scenarios for example we may have some leaves which are being affected by some light exposure or the direction of an uh image out the orientation of the image so these kinds of previous processing tasks are needed to be introduced in the data pipeline or else I won't be able to perform in the best way possible, now it is time to generate the transformed data, below snippet of code is going to perform that task.



```
train_generator = train_datagen.flow_from_directory(
    train_dir,
    subset="training",
    shuffle=True,
    seed=42,
    color_mode="rgb",
    class_mode="categorical",
    target_size=IMAGE_SHAPE,
    batch_size=BATCH_SIZE)
```

The same operations are performed for validation it asset with few tunings in the settings as uh we don't really want to make our images more complex as validation tests are performed just to see how well our model has been trained so the transformation not that necessary when it comes to validating our model unlike in the case of training our model which requires covering as many aspects or scenarios as we can cover.



```
Found 10861 images belonging to 38 classes.
Found 43444 images belonging to 38 classes.
```

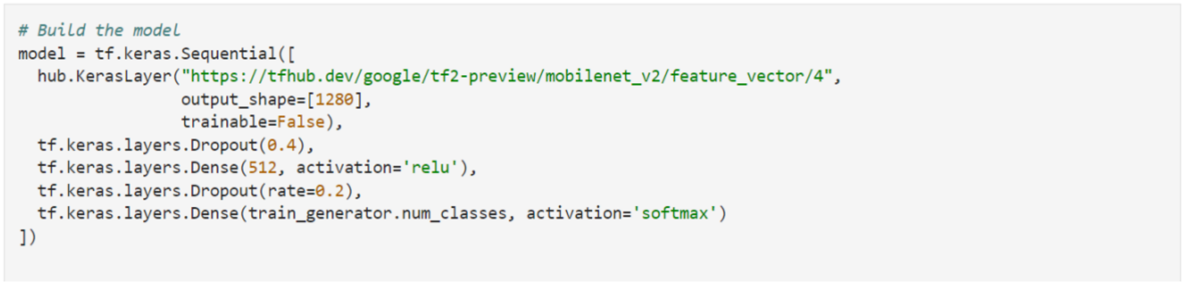
The above result shows us the number of images present in validation class and then training class of images respectively.

20.5 MODEL BUILDING

Model building is a critical step in the process of plant disease classification using machine learning. The goal of model building is to design a machine learning model that can accurately classify plant leaves images into diseased or healthy categories. Here are the essential steps involved in building a plant disease classification model:

- ❖ Data Collection: Collect a large dataset of images that are labeled as either healthy or diseased. The dataset should be representative of the plant species and diseases that the model is designed to identify.
- ❖ Data Pre-processing: Pre-process the data by resizing, normalization, cropping, and augmenting it to improve the quality and diversity of the dataset.
- ❖ Training-Validation-Testing Split: Split the pre-processed data into training, validation, and testing sets. The training set is used to build the model, the validation set is used for hyperparameter tuning, and the testing set is used for model evaluation.
- ❖ Model Selection: Select the appropriate machine learning algorithm for the classification task. Common algorithms used for plant disease classification include Convolutional Neural Networks (CNNs), Support Vector Machines (SVMs), Random Forests, and K-Nearest Neighbors (KNNs).
- ❖ Model Architecture: Design the architecture of the machine learning model based on the selected algorithm. This architecture should take into account the specifications of the dataset and the performance of the learning algorithm.
- ❖ Hyperparameter Tuning: Optimize the model by selecting the optimal hyperparameters to improve the model's performance on the validation set.
- ❖ Model Training: Train the model using the training set, and validate it using the validation set. Train the model using an iterative process such as Batch Gradient Descent or Mini-batch Gradient Descent, to improve model performance.
- ❖ Model Evaluation: Evaluate the trained model's performance on the testing set using metrics such as accuracy, precision, recall, and F1 score, and confusion matrix.
- ❖ Model Deployment: Deploy the model to an application or service that can leverage the model's prediction capabilities. The deployment can be on-premises or on the cloud dependently.

In conclusion, model building is a fundamental step in plant disease classification using machine learning. It involves selecting the appropriate algorithm, designing the model architecture, optimizing the hyperparameters, training the model and evaluating the performance on the testing set, and deploying the trained model. This process requires a careful selection of the specific tasks outlined above to achieve optimal performance.



```
# Build the model
model = tf.keras.Sequential([
  hub.KerasLayer("https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4",
                 output_shape=[1280],
                 trainable=False),
  tf.keras.layers.Dropout(0.4),
  tf.keras.layers.Dense(512, activation='relu'),
  tf.keras.layers.Dropout(rate=0.2),
  tf.keras.layers.Dense(train_generator.num_classes, activation='softmax')
])
```

The error model consists of two parts first part is the pre trained model provided by trans of row called mobile net V2 whose architecture is given below.

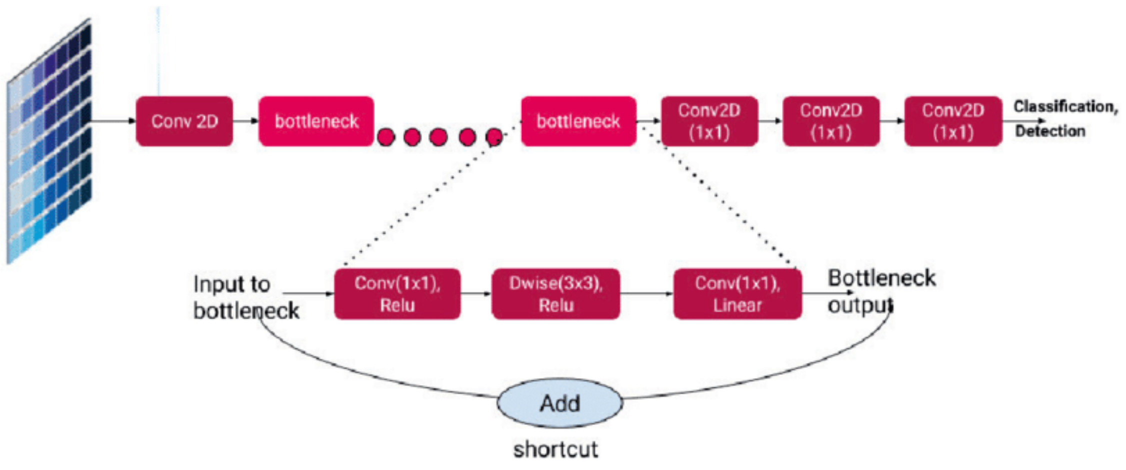


Figure 45 Architecture of MobileNet V2

And then what we have done is we have added few more layers to provide more filtration on the output, and whole model is a sequential model where different layers are just stacked over one another.

An overview of the whole architecture of neural network is provided below along with description of every layer that I have used in this.

- ❖ Keras.Sequential is a type of model architecture in the Keras deep learning library for Python. It is a simple model that consists of a linear stack of layers that can be used for a wide range of deep learning tasks such as image classification, text analysis, and regression.
- ❖ The Keras.Sequential model is called sequential because each layer is added one at a time in a linear fashion, forming a "stack" of layers. The output of one layer acts as the input to the next layer in the stack. Sequential models are generally easier to build and use, making them popular for beginners and experts.

Characteristics Of the Keras.Sequential Models:

- ❖ Easy to Implement: Keras.Sequential models are simple and easy to implement. It is a good choice for quick experiments and creating machine learning models that follow a simple architecture.
- ❖ Flexible: The Keras.Sequential model is flexible because it allows the creation of any combination of layers and activation functions and supports various types of input.
- ❖ Scalable: Sequential models can be scaled up by adding more layers or additional nodes in each layer, making it ideal for larger, more complex prediction tasks.
- ❖ Compatible with Different Layers: Keras.Sequential models are compatible with various layers, including convolutional, pooling, recurrent, dense, and more. These layers can be added to the model sequentially to build a robust machine learning model.
- ❖ Optimized for GPU: Keras.Sequential models are highly optimized for GPU processing, making it faster and more efficient to train deep learning models.

MobileNetV2

It is a convolutional neural network (CNN) architecture, often used for image classification. It is designed to be lightweight and efficient, making it an excellent choice for mobile devices and other applications with limited computational resources. MobileNetV2 introduces several new ideas to the original MobileNet architecture, including inverted residuals, linear bottlenecks, and shortcut connections, to improve the accuracy and performance of the model. Here are the architectural details of MobileNetv2:

Input Layer: The input layer of MobileNetV2 accepts an input image of shape (224, 224, 3), where 224x224 are the image dimensions, and 3 represents the Red, Green, and Blue (RGB) color channels of the image.

- ❖ Convolutional Layers: The backbone of MobileNetV2 consists of a series of convolutional layers. These layers use depthwise separable convolutions to reduce the number of parameters. Depthwise separable convolutions perform convolution operations separately for the input channels, followed by pointwise convolutions (1x1 convolutions) to combine the output channels. This architecture results in a significant reduction in model size and computational cost, ideally suited to mobile devices.
- ❖ Inverted Residuals: MobileNetV2 introduces a new module called "inverted residuals," which includes a depthwise convolution layer, a 1x1 pointwise convolution layer to perform feature transformation, and a linear bottleneck layer to increase feature depth.
- ❖ Shortcut Connections: MobileNetV2 incorporates shortcut connections to connect certain layers directly with a few layers ahead. This technique is used to improve the gradient flow, enabling better information propagation through the network.
- ❖ Linear Bottlenecks: Linear bottlenecks are used to improve the accuracy of MobileNetV2. They are intermediate layers that use linear activations instead of non-linear activation functions such as ReLU. Linear bottlenecks reduce the dimension of the feature space, allowing the model to use the limited number of parameters efficiently.
- ❖ Classification Layers: The final part of the architecture consists of a Global Average Pooling layer that takes the output of the convolutional layers and averages it over the spatial dimensions, followed by a fully connected layer and a softmax function to produce the output probability distribution.

tf.keras.layers.Dropout

It is a regularization technique that is used to prevent overfitting in neural networks. Dropout is a type of regularization method that randomly turns off a fraction of the neurons in a layer during training. It works by randomly dropping out some of the units in a layer and setting their output to zero during training.

The Dropout layer in Keras is a regularization layer that can be added to any layer in a neural network. Typically, it is placed after the activation layer in a multi-layer neural network.

Here's how Dropout regularization works in Keras:

- ❖ During training, a fraction of the neurons in the layer are randomly selected and deactivated by setting their output to zero.
- ❖ The remaining active neurons are then trained on the data as normal, as part of a forward and backward pass.
- ❖ During inference, i.e. when the model is being used for prediction rather than training, the Dropout layer is turned off so that all neurons are active, and the complete model is used.

The proportion of neurons to be dropped out is a hyperparameter that can be tuned. The purpose of dropout is to address the overfitting problem by reducing the co-adaptation of neurons during training. By randomly dropping out some of the neurons, it forces the other neurons to learn more robust and generalizable features.

In summary, the Dropout layer in Keras is a regularization technique that randomly removes a fraction of the neurons in the layer, forcing the rest of the neurons to learn more robust features during training. This technique helps to prevent overfitting, improve the generalization of the model, and ultimately leads to better performance on unseen data.

tf.keras.layers.Dense

It is a neural network layer in the Keras API for TensorFlow, used to implement fully connected layers in a deep learning model. A dense layer consists of multiple nodes that are fully connected to the output of the previous layer. Each node in the layer receives an input from the entire output of the previous layer and computes a dot product with a set of learnable weights. The resulting output is passed through an activation function, which introduces non-linearity into the model. Here's a brief overview of the important aspects of a Dense layer in TensorFlow:

- ❖ Input Shape: The Dense layer requires information about the shape of the input data. This information is used to construct the weight matrix that connects the layer to the previous layer.
- ❖ Output Shape: The output shape of the Dense layer is determined by the number of units or nodes in the layer.
- ❖ Activation Function: The Dense layer requires an activation function to introduce non-linearity into the model. Common activation functions used with Dense layers include sigmoid, ReLU, and tanh.
- ❖ Weights: The Dense layer learns the appropriate weights to connect each input feature to each output feature. During training, the goal is to adjust the weights to minimize the loss.
- ❖ Bias: The Dense layer includes a bias term that is added to the output of each node in the layer. The bias term acts as an offset to the linear combination of inputs and weights.

The Dense layer is widely used in deep learning models, and it can be used for both regression and classification tasks. It is flexible and can be modified to allow for regularization techniques such as dropout, l1/l2 regularization, and batch normalization. The number of Dense layers in a neural network model and the number of nodes in each Dense layer are hyperparameters that need to be tuned for optimal performance.

20.7 TRAINING/ VALIDATION

Model training and validation are critical steps in building machine learning models. The training step involves tuning the model, optimizing its weights, and seeking the best performing model on the training dataset, as well as evaluating the model's generalization performance on the new unseen dataset. These steps help to fine-tune the model, select the optimal hyperparameters, avoid overfitting, and achieve better model performance.

But before just driving into training and validation we have to define few parameters that the model or more specifically a neural network takes in for reaching or achieving the best accuracy we want one such parameter is learning rate, number of approaches, batch size, and optimizer. A detailed description is provided below.

20.7.1 Learning rate

It is a key hyperparameter of many machine learning algorithms that determines the step size at which the gradient or error is used to update the weights (or coefficients) of the model. Simply put, the learning rate is a tuning parameter used to balance the convergence speed of the optimization algorithm and the risk of overshooting the optimal solution.

A learning rate that is too high may result in the optimization algorithm diverging (i.e., not converging to a useful solution), whereas a learning rate that is too low may result in the optimization algorithm converging too slowly, delaying the optimization process.

For example, in gradient descent optimization, the update rule of weights $w$ is $w = W - \eta \nabla L(w)$, where $\eta$ is the learning rate, $L$ is the loss function, and $\nabla L$ is the gradient of the loss function with respect to the weights $w$. The learning rate $\eta$ determines how much the gradient influences the weight update and is thus a critical parameter that should be tuned to optimize the performance of the model.

The optimal learning rate depends on factors such as the complexity of the model, the size and quality of the dataset, the gradient descent optimization method used, and more. One of the common methods for tuning the learning rate is by using a learning rate schedule protocol such as decreasing the learning rate over time or per iteration to further fine-tune the convergence speed of the model.

Finding an appropriate learning rate typically requires some experimentation and iteration. In practice, trial and error are used to iterate through different learning rates to determine the most appropriate one that optimizes the model for the given problem. Consequently, the learning rate schedule is changed to adjust the rate during model training, depending on the signs of overfitting or underfitting of the model.

20.7.2 Batch size

It is another important hyperparameter in machine learning that refers to the number of training examples used in a single iteration during the optimization of a model. The training data is split into batches of a fixed size, and each batch is used to update the model's parameters during training. The batch size is typically set to a power of 2, such as 32, 64, or 128, to enable efficient computation on modern hardware.

Batch size has a significant impact on the training process and model performance, and different batch sizes can influence the speed, quality, and generalization capability of the model. Here are some factors to consider when selecting the appropriate batch size for your model:

- ❖ Computational Resources: Batch size is closely related to the amount of memory required to train the model. Larger batch sizes require more memory but usually offer faster optimization.
- ❖ Optimization Speed: Smaller batch sizes offer higher optimization speed at the cost of accuracy, whereas larger batch sizes achieve higher accuracy levels but reduce optimization speed considerably.

20.7.3 Optimizer

Optimizer in machine learning refers to an algorithm or method that is used to optimize the parameters of a machine learning model during the training process. The objective of optimization is to minimize the loss function, which reflects the error between the predicted output of the model and the actual output.

Optimizers work by adjusting the weights and biases of a model based on the error or loss function, which measures how accurately the model is performing. The most popular optimization algorithms used in machine learning include stochastic gradient descent (SGD), Adagrad, RMSprop and Adam.

SGD is a simple and widely used optimization method that updates the weights and biases of a model at each step based on the gradient of the loss function with respect to the parameters. Other optimization algorithms, such as Adagrad, RMSprop and Adam, are more sophisticated and leverage techniques such as adaptive learning rates and momentum to improve optimization performance.

20.7.4 Loss

In machine learning, loss refers to the error or cost of a model's predictions compared to the true values. In particular, it's a measure of how well a model is able to map inputs to outputs.

In neural network training, the loss function measures the difference between the predicted output of the model and the true target output. The goal of training is to minimize the loss function, so that the predictions of the model are as close as possible to the true target output.

During the training of a neural network, the model's parameters, such as weights and biases, are updated based on the gradients of the loss function. This process is repeated over multiple iterations until the loss function is minimized and the model converges to the optimal set of parameters.

Common loss functions used in neural network training include mean squared error (MSE), binary cross-entropy, and categorical cross-entropy. The choice of loss function depends on the problem being solved and the type of output being predicted.

20.7.5 Metrics

In neural network training, metrics are used to evaluate the performance of a model during training and testing. Metrics are a set of quantitative measurements that help us to determine how well the model is learning and generalizing its predictions.

Some common metrics used in neural network training include accuracy, precision, recall, F1 score, and area under the receiver operating characteristic curve (ROC-AUC). The choice of metric depends on the problem being solved and the type of output being predicted.

Accuracy is a simple and widely used metric that measures the percentage of correctly predicted samples out of all samples. Precision and recall are more specific to binary classification problems and measure the fraction of true positives over all predicted positives and the fraction of true positives over all actual positives, respectively.

The F1 score is a harmonic mean of precision and recall that balances both metrics. ROC-AUC is commonly used in binary classification problems and measures the trade-off between true positive rate and false positive rate at different probability thresholds.

By monitoring metrics during training, we can use this to modify the model to improve the results. Additionally, metrics can be used to compare different models and select the best one for a specific problem.

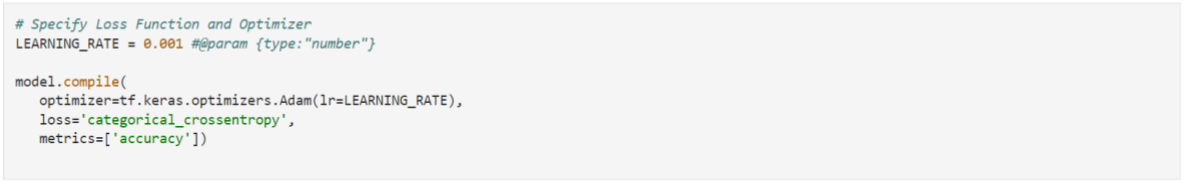
20.7.6 Epoch

In machine learning, an epoch refers to a complete pass through the entire training dataset during model training. One epoch is defined as one forward pass and one backward pass of each training example, and the goal of training is to reduce the loss function over multiple epochs.

In practical terms, an epoch is a hyperparameter that is chosen by the model developer and can have a significant impact on the final performance of the model. Choosing an appropriate number of epochs depends on factors such as the size and complexity of the dataset, the complexity and size of the model, and the number of computational resources available.

During training, the model updates its parameters based on the gradients of the loss function with respect to the parameters for each epoch. After each epoch, the model's performance can be evaluated on a separate validation set or through cross-validation to monitor its performance.

Too few epochs can result in underfitting, where the model does not learn enough from the data, while too many epochs can result in overfitting, where the model learns to memorize the training data and therefore performs poorly on new, unseen data. Thus, the number of epochs is typically chosen using methods such as early stopping, where the training is halted once the model's performance stops improving on a validation set.



```
# Specify Loss Function and Optimizer
LEARNING_RATE = 0.001 #@param {type:"number"}

model.compile(
   optimizer=tf.keras.optimizers.Adam(lr=LEARNING_RATE),
   loss='categorical_crossentropy',
   metrics=['accuracy'])
```

The above snippet of code is combining the different parameters that we have defined like learning rate the optimizer the kind of loss we are going to use for measuring the loss and then metrics which is going to be used for telemetry purposes which is referred for seeing how well our model is predicting or working.

Our model is trained using backpropagation.

Backpropagation

Backpropagation is an algorithm used in machine learning and specifically in neural networks, that calculates the gradient of the loss function with respect to the weights in the neural network. It works by computing the gradients of the loss function with respect to the activations of each neuron in the last hidden layer, and then propagating these gradients backwards through the network to compute the gradients of the loss function with respect to the weights.

The core concept behind backpropagation is the chain rule of calculus, which allows us to calculate the derivative of a nested function by breaking it down into smaller derivatives and multiplying them together. In the case of neural networks, the chain rule is used to calculate the partial derivative of the loss function with respect to each weight in the network.

The backpropagation algorithm is traditionally used in conjunction with stochastic gradient descent (SGD) and other optimization algorithms to train neural networks. During training, the loss function is evaluated using a training data set, and the network parameters are updated iteratively by computing the gradients using backpropagation and then adjusting the parameters in the direction of the negative gradients. The process continues until the loss function converges to a minimum.

Backpropagation is a fundamental concept in modern machine learning and is used in a variety of deep learning architectures and applications. It enables neural networks and other machine learning models to learn from complex and large datasets, making it a widely used technique for training complex models.

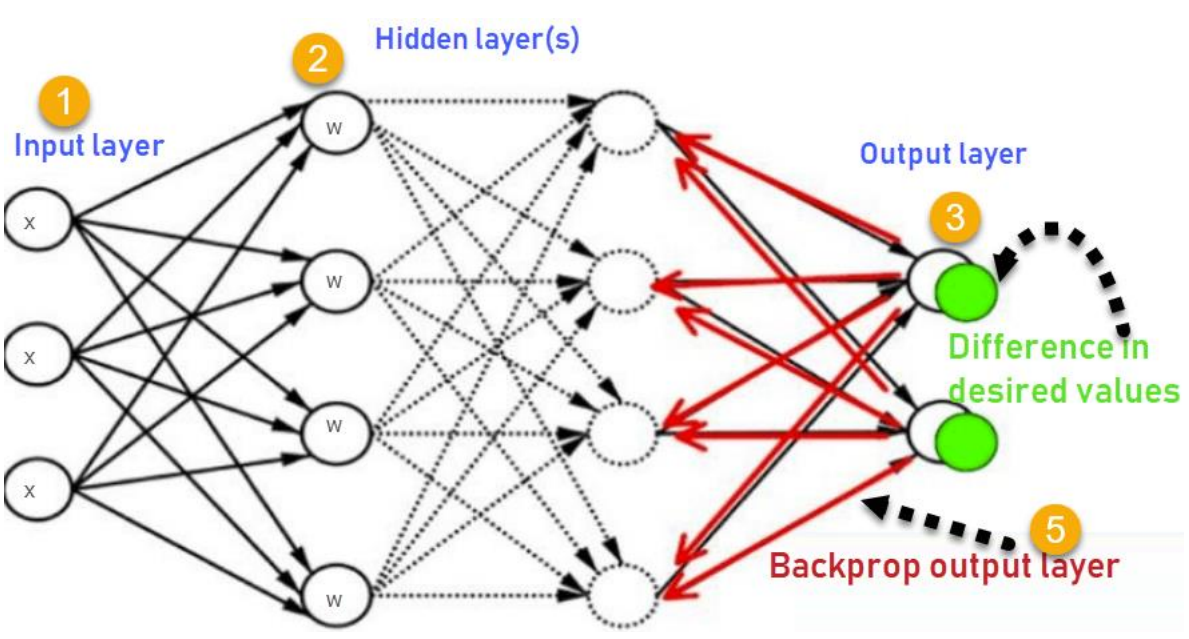
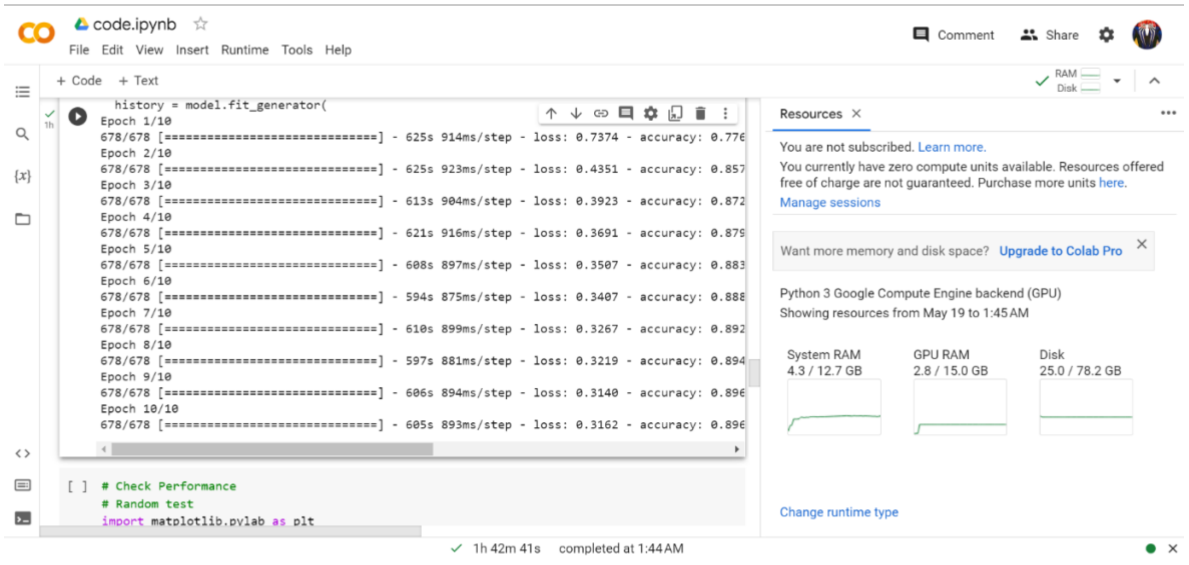


Figure 46 Backpropagation Workflow

```
# Train Model

EPOCHS=10 #@param {type:"integer"}

history = model.fit_generator(
        train_generator,
        steps_per_epoch=train_generator.samples//train_generator.batch_size,
        epochs=EPOCHS,
        validation_data=validation_generator,
        validation_steps=validation_generator.samples//validation_generator.batch_size)
```

The above stripped of code is starting the training of the model and performing the training of the model in 10 epochs,



```
  history = model.fit_generator(
Epoch 1/10
678/678 [==============================] - 625s 914ms/step - loss: 0.7374 - accuracy: 0.776
Epoch 2/10
678/678 [==============================] - 625s 923ms/step - loss: 0.4351 - accuracy: 0.857
Epoch 3/10
678/678 [==============================] - 613s 904ms/step - loss: 0.3923 - accuracy: 0.872
Epoch 4/10
678/678 [==============================] - 621s 916ms/step - loss: 0.3691 - accuracy: 0.879
Epoch 5/10
678/678 [==============================] - 608s 897ms/step - loss: 0.3507 - accuracy: 0.883
Epoch 6/10
678/678 [==============================] - 594s 875ms/step - loss: 0.3407 - accuracy: 0.888
Epoch 7/10
678/678 [==============================] - 610s 899ms/step - loss: 0.3267 - accuracy: 0.892
Epoch 8/10
678/678 [==============================] - 597s 881ms/step - loss: 0.3219 - accuracy: 0.894
Epoch 9/10
678/678 [==============================] - 606s 894ms/step - loss: 0.3140 - accuracy: 0.896
Epoch 10/10
678/678 [==============================] - 605s 893ms/step - loss: 0.3162 - accuracy: 0.896
```

```
[ ] # Check Performance
    # Random test
    import matplotlib.pylab as plt
```

You are not subscribed. Learn more.
You currently have zero compute units available. Resources offered free of charge are not guaranteed. Purchase more units here.

Want more memory and disk space? Upgrade to Colab Pro

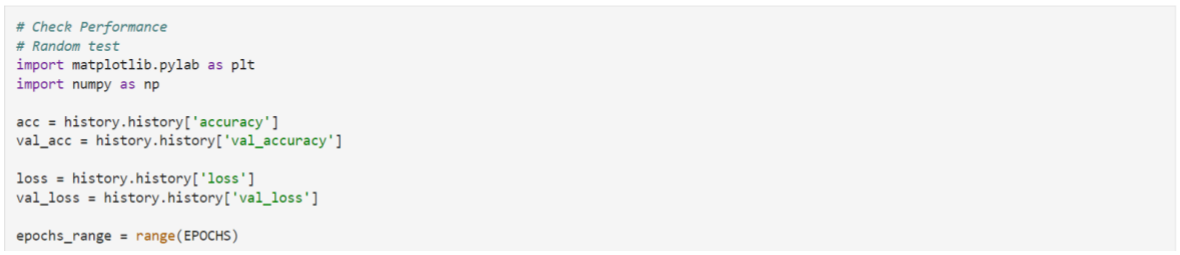
Python 3 Google Compute Engine backend (GPU)
Showing resources from May 19 to 1:45 AM

System RAM 4.3 / 12.7 GB
GPU RAM 2.8 / 15.0 GB
Disk 25.0 / 78.2 GB

✓ 1h 42m 41s completed at 1:44 AM

Figure 47 Model Training Completion

The image shows us the completion of the training of our model and the amount of resources used while the process, as you can see the resources like RAM GPU and the disk storage which was used while the training of the model and the amount of time is also mentioned at the very bottom of the image as you can see it took around 1 hour 42 minutes for completing the training of our model this was possible because of the inclusion of GPU to boost the performance or the speed of training as if I was to use the CPU as my hardware it would have taken a lot more time on the data set of this much size and specifically the images require powerful hardware as they are array of data rather than just a single array they contain dimensions and as size of image increases the amount of data in a particular image also increases so we need a much better way a more powerful way to train our model so GPU is answer for doing that. Now it's time for performing validation test runner models so the below Porter is going to lowered our validation data set and import it or pass it to our trade model and see how well our model is done on this data set.



```
# Check Performance
# Random test
import matplotlib.pylab as plt
import numpy as np

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(EPOCHS)
```

After doing this we're going to also map a graph for make the telemetry more readable and understandable for other people.

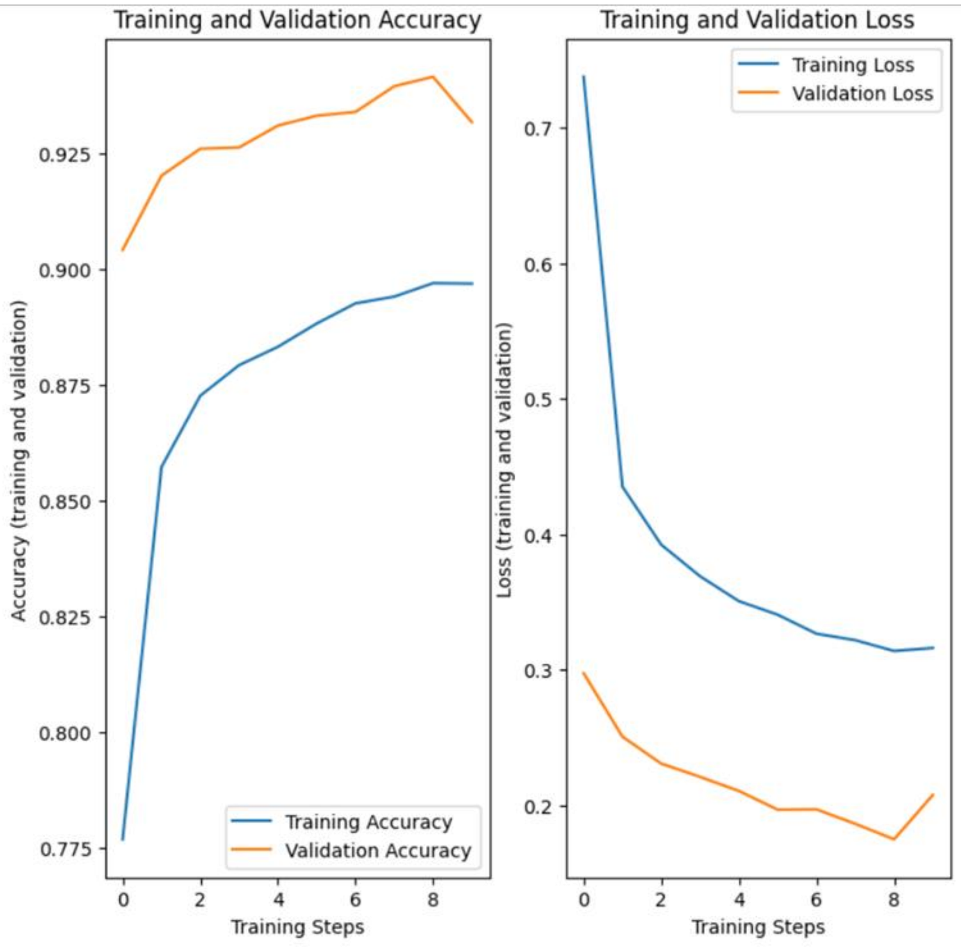


Figure 48 Training and Validation Accuracy and Loss

This concludes our whole life cycle of collecting data from a data source and then performing different preprocessing tasks on that data then selecting our model which is more helpful to us and then training that model on that data and then checking how well our model was trained and highlighting level of accuracy.

THE CODE FOR ALL THE PROCESSES, IS GIVEN BELOW:

```
#!/usr/bin/env python

# coding: utf-8

# In[1]:

from __future__ import absolute_import, division, print_function, unicode_literals

import tensorflow as tf
```

```
#tf.logging.set_verbosity(tf.logging.ERROR)
#tf.enable_eager_execution()
import tensorflow_hub as hub
import os
from tensorflow.keras.layers import Dense, Flatten, Conv2D
from tensorflow.keras import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import layers
#from keras import optimizers
# In[2]:
# verify TensorFlow version
print("Version: ", tf.__version__)
print("Eager mode: ", tf.executing_eagerly())
print("Hub version: ", hub.__version__)
print("GPU is", "available" if tf.test.is_gpu_available() else "NOT AVAILABLE")
# In[3]:
# Loading Data
zip_file                                                                    =
tf.keras.utils.get_file(origin='https://storage.googleapis.com/plantdata/PlantVillage.zip',
                             fname='PlantVillage.zip', extract=True)
# In[4]:
# Prepare training and validation dataset
data_dir = os.path.join(os.path.dirname(zip_file), 'PlantVillage')
train_dir = os.path.join(data_dir, 'train')
validation_dir = os.path.join(data_dir, 'validation')
# In[5]:
import time
import os
```

```
from os.path import exists

def count(dir, counter=0):
    "returns number of files in dir and subdirs"
    for pack in os.walk(dir):
        for f in pack[2]:
            counter += 1
    return dir + " : " + str(counter) + "files"

# In[6]:

print('total images for training :', count(train_dir))

print('total images for validation :', count(validation_dir))

# In[7]:

# Label mapping

get_ipython().getoutput('wget                        https://github.com/obeshor/Plant-Diseases-
Detector/archive/master.zip')

get_ipython().system('unzip master.zip;')

# In[8]:

import json

with open('Plant-Diseases-Detector-master/categories.json', 'r') as f:
    cat_to_name = json.load(f)
    classes = list(cat_to_name.values())

print (classes)

# In[9]:

print('Number of classes:',len(classes))

# In[10]:

# Setup Image shape and batch size

IMAGE_SHAPE = (224, 224)

BATCH_SIZE = 64 #@param {type:"integer"}

# In[11]:

# Data Preprocessing
# Inputs are suitably resized for the selected module. Dataset augmentation (i.e., random distortions of an image each time it is read) improves training, esp. when fine-tuning.

validation_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)

validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
    shuffle=False,
    seed=42,
    color_mode="rgb",
    class_mode="categorical",
    target_size=IMAGE_SHAPE,
    batch_size=BATCH_SIZE)

do_data_augmentation = True #@param {type:"boolean"}

if do_data_augmentation:
  train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
      rescale = 1./255,
      rotation_range=40,
      horizontal_flip=True,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      fill_mode='nearest' )
else:
  train_datagen = validation_datagen

train_generator = train_datagen.flow_from_directory(
    train_dir,
    subset="training",
    shuffle=True,
    seed=42,
    color_mode="rgb",
    class_mode="categorical",
    target_size=IMAGE_SHAPE,
    batch_size=BATCH_SIZE)
# In[13]:
# Build the model
model = tf.keras.Sequential([
  hub.KerasLayer("https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4",
                 output_shape=[1280],
                 trainable=False),
  tf.keras.layers.Dropout(0.4),
  tf.keras.layers.Dense(512, activation='relu'),
  tf.keras.layers.Dropout(rate=0.2),
  tf.keras.layers.Dense(train_generator.num_classes, activation='softmax')
])
# In[14]:
# Specify Loss Function and Optimizer
LEARNING_RATE = 0.001 #@param {type:"number"}
model.compile(
   optimizer=tf.keras.optimizers.Adam(lr=LEARNING_RATE),
   loss='categorical_crossentropy',
   metrics=['accuracy'])
# In[16]:
# Train Model
EPOCHS=10 #@param {type:"integer"}
history = model.fit_generator(
        train_generator,
        steps_per_epoch=train_generator.samples//train_generator.batch_size,
        epochs=EPOCHS,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples//validation_generator.batch_size)
# In[17]:
# Check Performance
# Random test
import matplotlib.pylab as plt
import numpy as np
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(EPOCHS)
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.ylabel("Accuracy (training and validation)")
plt.xlabel("Training Steps")
plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.ylabel("Loss (training and validation)")
plt.xlabel("Training Steps")
plt.show()
```

CHAPTER 21 ASSISTANCE FOR TREATMENT

The ability to be able to identify the kind of disease uh when splint or a crop may be suffering from is one functionality but what we also need along with this ability is that the user or the farmer who is trying to identify the disease can also receive uh the advice or the suggestion or the exact treatment for that disease on the planet so that he can prevent his growth from deteriorating and receiving the best yield possible so for that what we need is yeah create texture like a Chad word where the person can interact with but chat boards also the only limited functionality and we have to again and again upgrade their information and they don't even have human like ability to extract the information or the knowledge which is required by the user So what we have done here is that we have used a large language model and integrated it in our application which is going to be explained in next section or next chapter to precise. That large language model is going to act as another part of this whole data pair pipeline where it is going to receive the classified disease by the computer vision model and after that it is going to provide the cures or the treatment the user can provide or treat his or her crop with and prevent the deteriorating of his or her crop.

Disability of a chatting with the human like interface rather than waiting for someone to contact customer service yeah you two are partially specialized in the and then provide you with the treatment or suggestions how you can attend your crops but considering this way every person in their provide phone can access a large language model powered assistant which can act like a human and then provide you all the kinds of answers regarding your all the kinds of queries so that you can just very fastly and without waiting for someone else to call you know you can just receive the required treatment advice for your plan.

Overview of the workflow how this idea of large language model powered assistant is embedded into the application is given below:

- ❖ User inputs an image of the affected area into the application.
- ❖ The computer vision model analyzes the image and identifies the disease or condition.
- ❖ The OpenAI language model (LLM) assistant is triggered based on the identified disease or condition.

- ❖ The LLM provides relevant information on the condition, such as its symptoms, causes, and potential treatments.
- ❖ Once the LLM has provided treatment information, the user can review the information and follow up with their healthcare provider for personalized treatment recommendations.

The Langchain framework provides a way to integrate machine learning models with natural language processing technologies, allowing the user to converse with the AI-powered assistant in natural language. This enables the assistant to provide more personalized and specific guidance to the user based on their unique situation.

Overall, this application has the potential to provide users with valuable insights and guidance for managing their health and wellbeing.

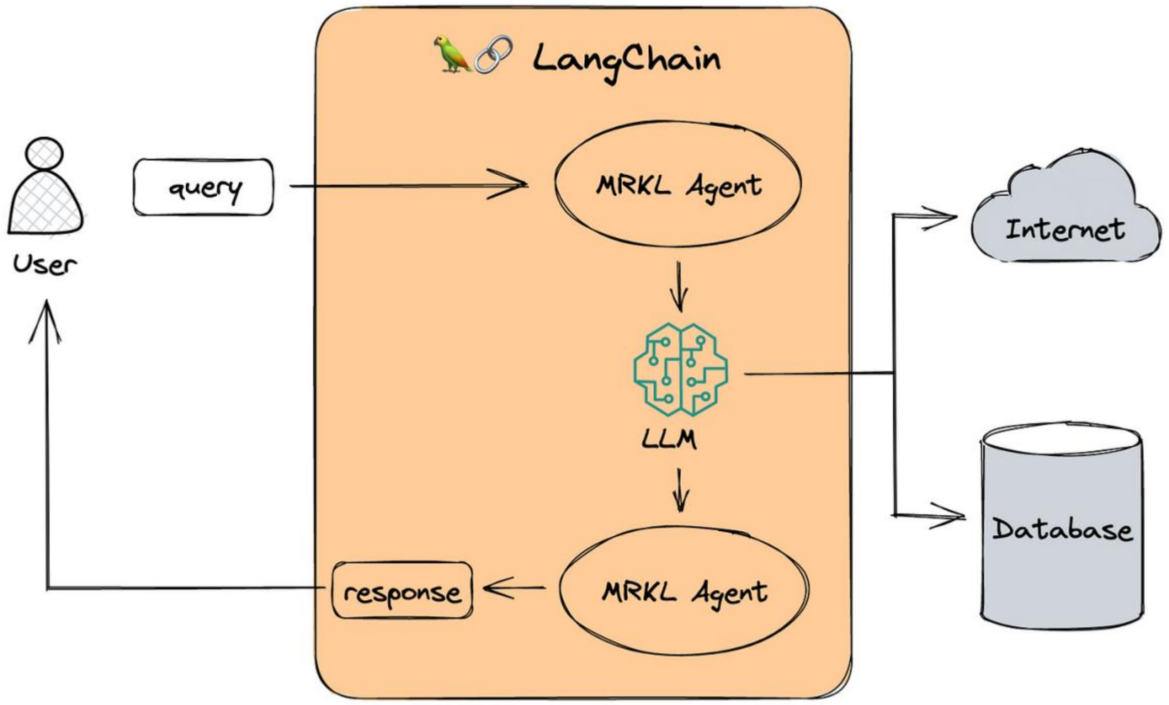
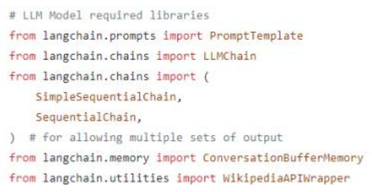


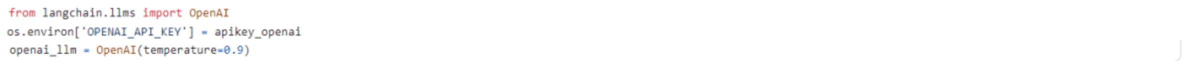
Figure 49 LLM powered workflow



```
# LLM Model required libraries
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain
from langchain.chains import (
    SimpleSequentialChain,
    SequentialChain,
)  # for allowing multiple sets of output
from langchain.memory import ConversationBufferMemory
from langchain.utilities import WikipediaAPIWrapper
```

Before creating the larger language model workflow which is going to be accessed with the help of a framework called langchain, we need to import some dependencies which is being done by the code snippet provided above.

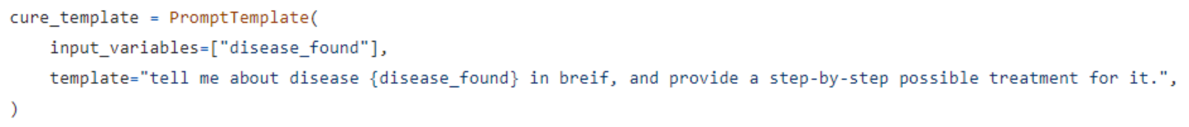
And next, we need to identify which large language model we want to import because the Lang chain environment provide us the access for various kinds of large language model and then our case we are going to use open AI large language model which is going to be accessed through an API key and the way to access it is provided below.



```
from langchain.llms import OpenAI
os.environ['OPENAI_API_KEY'] = apikey_openai
openai_llm = OpenAI(temperature=0.9)
```

For this snippet of code is doing is importing the open API dependency and then setting an environment variable which is going to store the API key for open AI so that we can access large language model provided by open AI and then what we have done is we have called the open AI method which is going to define the model which is imported to the help of API key.

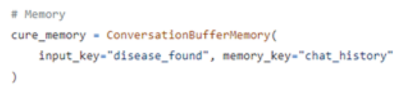
Now, large language model architecture in any application we use language change to provide us various functionalities are majorly there are few key characteristics of Lang chain framework one such as prompt templates, from templates are going to define what prompt for the input we want to give to our large language model so whenever our disease is defined what we are going to do we are going to connect the output which is going to be generated from our computer vision disease classification model to this prompt template so that when a prompt goes to our large language model is going to be about the disease we want the cure about.



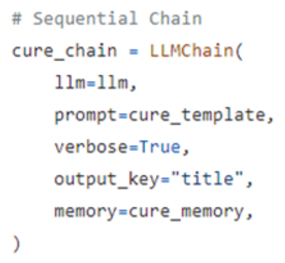
```
cure_template = PromptTemplate(
    input_variables=["disease_found"],
    template="tell me about disease {disease_found} in breif, and provide a step-by-step possible treatment for it.",
)
```

The above snippet of code is what we call as a prompt template which is going to define an input variable which is going to be that is formed by our computer vision model and then below is a template for the prompt that you usually enter in a ChatGPT for which the large language model gives you an answer for or provide you with the response similarly we are going to use that idea but what we are doing here is we are just making more this whole process automatic.

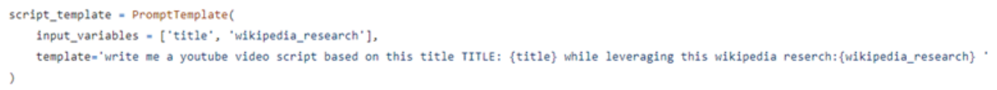
As a large image model of framework line chain provides various functionalities along with the creating automatic from templates one such functionality is called memory where we can just store our messages whatever we query from my last language model will be stored in the memory and it can be used for other purposes also not just from last language model the output from different different sources can also be stored in this for making our output more efficient.


```
# Memory
cure_memory = ConversationBufferMemory(
    input_key="disease_found", memory_key="chat_history"
)
```

The above stripped code is defining a memory for storing the outputs which are going to be given by a large language model. Next thing that land chain framework providers is the ability to chain everything together meaning we have our template then we have our memory we need a way to connect everything together for that long chain providers with the different kinds of chains one such change that I've used in this application is sequential chain score for that is provided below.


```
# Sequential Chain
cure_chain = LLMChain(
    llm=llm,
    prompt=cure_template,
    verbose=True,
    output_key="title",
    memory=cure_memory,
)
```

LangChain also provides the ability to use different toolkits, basically toolkit is a data source which is other than the knowledge of large language model for example wikipedia or a document that you have of your own using both of them is quite similar but the way they are organized in the application is different at first what we are going to do we are going to introduce Wikipedia as a toolkit code for that is given below.


```
script_template = PromptTemplate(
    input_variables = ['title', 'wikipedia_research'],
    template='write me a youtube video script based on this title TITLE: {title} while leveraging this wikipedia reserch:{wikipedia_research} '
)
```

A template is created for storing or querying the Wikipedia data, word similarly a memory is also created for storing the data which is extracted from Wikipedia and then it will be also added to the sequential chain and finally what will be doing is we are going to initiate the instance of a Wikipedia data extractor and then execute that chain.

```
script_memory = ConversationBufferMemory(input_key='title', memory_key='chat_history')


script_chain = LLMChain(llm=llm, prompt=script_template, verbose=True, output_key='script', memory=script_memory)

wiki = WikipediaAPIWrapper()
```

Coming to another toolkit we can also use our own documents for making our output more better more accurate for that what we need we need to install a vector database a vector database generally used for storing documents uh because when we store some data in a vector database what it does is it's going to convert the content of the document into vectors and then it is going to store it into the database and then we can query, so we are using chroma DB for that purpose in which we are going to store some document for this demonstration because i'm just storing the researches that i have done on this project into the database and whenever I call something or we can just query those documents from our last language this gives us an idea of how large language model can be helpful and there it doesn't need to just include the information which is provided by the a model developer we ourselves can also add extra information into the last language model and extract much more better output from it all of this is possible because of the framework called LangChain.

```
loader = PyPDFLoader('annualreport.pdf')
# Split pages from pdf
pages = loader.load_and_split()
# Load documents into vector database aka ChromaDB
store = Chroma.from_documents(pages, collection_name='annualreport')

# Create vectorstore info object - metadata repo?
vectorstore_info = VectorStoreInfo(
    name="annual_report",
    description="a banking annual report as a pdf",
    vectorstore=store
)
# Convert the document store into a langchain toolkit
toolkit = VectorStoreToolkit(vectorstore_info=vectorstore_info)

# Add the toolkit to an end-to-end LC
agent_executor = create_vectorstore_agent(
    llm=llm,
    toolkit=toolkit,
    verbose=True
)
```

The above sorbet of gold is providing the required configuration that we need to perform for storing documents into our chroma DB.

Complete code for how this whole logic is being embedded into the application is going to grow adding into the next chapter as lang chain framework has been integrated with the framework which is being used for development of the application.

CHAPTER 22 USER INTERFACE

User interface (UI) is extremely important for the successful deployment of a machine learning computer vision model for plant disease applications. A well-designed UI can help ensure user engagement, ease of use, and better interaction with the application. Here are some key features of a good UI for a machine learning computer vision model:

22.1 KEY FEATURES OF A GOOD UI

Intuitive and user-friendly: The UI should be straightforward and easy to navigate, with a well-organized layout that makes it easy to understand and engage with the application.

- Responsive and quick: The UI should respond quickly and seamlessly to user inputs, providing results and feedback in real-time, without any unnecessary delays or frustrations.
- Clear and informative: The UI should be designed to provide clear and concise information about the machine learning computer vision model, its purpose, and how it works.
- Eye-catching and engaging: The UI should be visually appealing and engaging, using color schemes, fonts, and graphics that draw the user's attention and encourage them to use the application.
- Personalized and customizable: The UI should offer personalization and customization options that allow the user to tailor the application to their unique needs and preferences.
- Error and exception handling: The UI should have robust error and exception handling capabilities, with clear error messages that help the user to resolve issues quickly and without frustration.
- Security and privacy: The UI should be designed to ensure the security and privacy of user data and should clearly communicate its security and privacy policies to users.

Overall, a well-designed UI can help to ensure the success and popular adoption of a machine learning computer vision model for plant disease applications, making it easier for users to engage with the technology and derive maximum value from its capabilities.

22.2 STREAMLIT

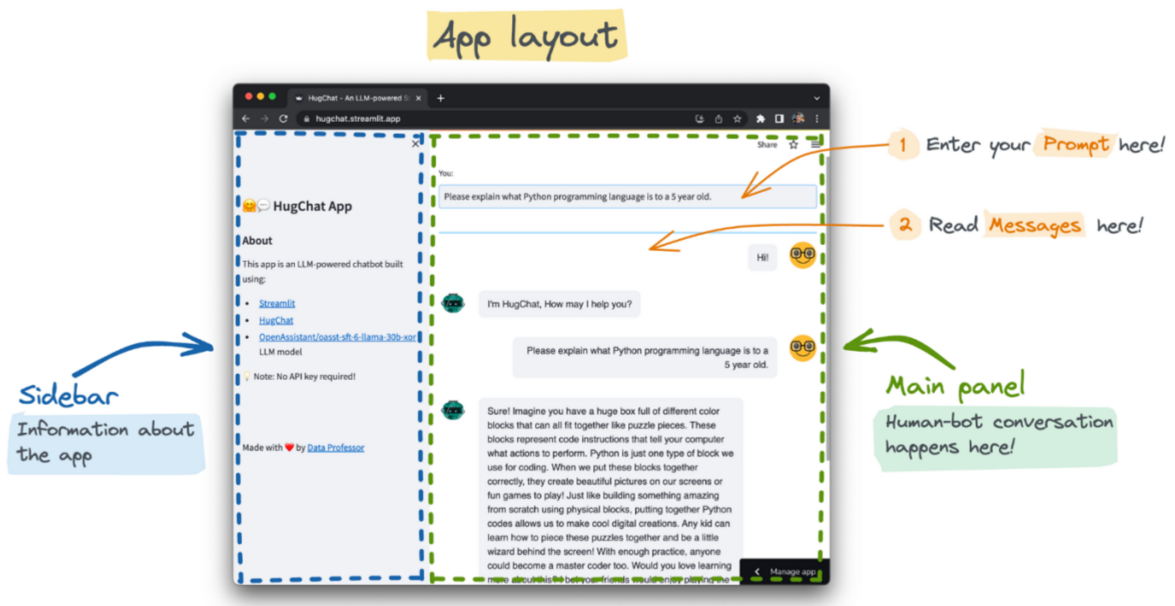
Streamlit is an open-source library for building interactive, web-based data science tools and applications. It allows developers to quickly build custom user interfaces for machine learning models, data visualizations, and other data science applications, without requiring any web development experience.

Streamlit can be used for large language model applications, such as those based on OpenAI's GPT, GPT-2, and GPT-3 models. One of the key benefits of Streamlit is that it allows developers to build interactive applications that enable users to interact with the model in real-time, generating and refining output on-the-fly as the user explores the inputs and parameters.

Advantages –

- ❖ Easy set up and deployment: Streamlit is extremely easy to set up and deploy, with a simple and intuitive interface that enables developers to get up and running with their applications quickly and without any complicated setup or configuration.
- ❖ Interactive user interface: Streamlit enables developers to create interactive user interfaces that allow users to explore and interact with the model's output and parameters in real-time.
- ❖ Real-time data updates: Streamlit allows developers to update their applications in real-time, ensuring that users always have access to the latest data and output.
- ❖ Collaboration features: Streamlit offers a range of collaboration features, including the ability to share and discuss code on GitHub, collaborate on workflows in real-time, and share applications with others.

Overall, Streamlit is an excellent tool for building and deploying large language model applications, enabling developers to quickly and easily build interactive applications that deliver meaningful insights and results to users.


App layout
HugChat - An LLM-powered S
hugchat.streamlit.app
Share
1 Enter your Prompt here!
2 Read Messages here!
Sidebar
Information about the app
Main panel
Human-bot conversation happens here!
Manage app

🤗💬 HugChat App

About

This app is an LLM-powered chatbot built using:

- Streamlit
- HugChat
- OpenAssistant/oasst-sft-6-llama-30b-xor LLM model

💡 Note: No API key required!

Made with ❤️ by Data Professor

You:

Please explain what Python programming language is to a 5 year old.

Hi!

I'm HugChat, How may I help you?

Please explain what Python programming language is to a 5 year old.

Sure! Imagine you have a huge box full of different color blocks that can all fit together like puzzle pieces. These blocks represent code instructions that tell your computer what actions to perform. Python is just one type of block we use for coding. When we put these blocks together correctly, they create beautiful pictures on our screens or fun games to play! Just like building something amazing from scratch using physical blocks, putting together Python codes allows us to make cool digital creations. Any kid can learn how to piece these puzzles together and be a little wizard behind the screen! With enough practice, anyone could become a master coder too. Would you love learning more about this? bet your friends would enjoy playing the

Figure 50 Overview of the Application

The previous provides an overview of how our application will look like or how it will be layout so that user can get the business situation like which part of this functionality of navigation is present where when he or she enters the portal.

Now let's get into how this whole application is built. First and foremost what we need is to import uh all the dependencies that we require as this application is the final location where all the ideas that we have talked about is going to be important and final output is going to be produced in the form of an interactive application.

```
# -----------------------------IMPORTING-REQUIRED-LIBRARIES--------------------
import streamlit as st
from streamlit_extras.colored_header import colored_header
from streamlit_extras.add_vertical_space import add_vertical_space
from streamlit_chat import message
```

in the above snippet of code, we are importing the Streamlit libraries which is going to be used for lay-outing this whole application.

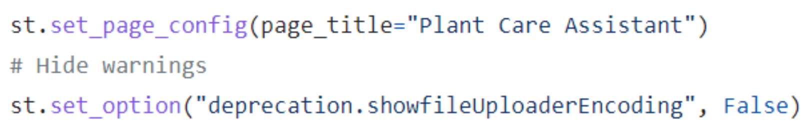
And then many more dependencies were like Lang chain tensor flow and more such libraries which are going to be needed in this application are going to be imported.

Before defining the whole application, we need to make few functions which can be called when we need to perform some task on an application as our application have two functionalities one is for performing inference on the plant images and 2nd is a large language model assistance which is going to be provided after the classification of disease has been performed.

So, this is the function for performing the inference.

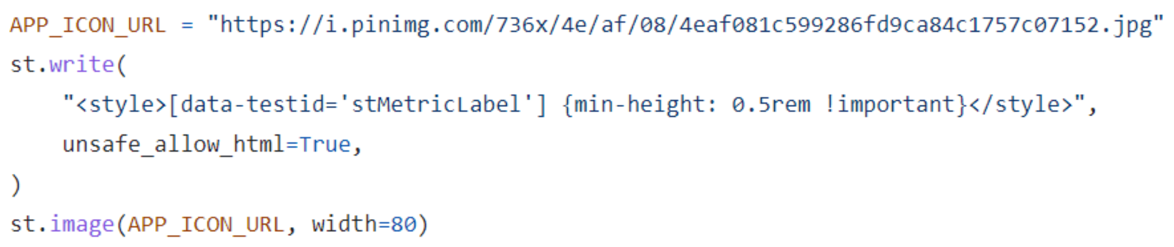
```
def classify_disease_uploaded_file(upload_image):
    file_bytes = np.asarray(bytearray(upload_image.read()), dtype=np.uint8)
    opencv_image = cv2.imdecode(file_bytes, 1)

    def load_image(opencv_image):
        img = cv2.resize(opencv_image, (IMAGE_SHAPE[0], IMAGE_SHAPE[1]))
        img = img / 255

        return img

    def predict(image):
        probabilities = model.predict(np.asarray([img]))[0]
        class_idx = np.argmax(probabilities)

        return {classes[class_idx]: probabilities[class_idx]}

    img = load_image(opencv_image)
    prediction = predict(img)
    disease_found = list(prediction.keys())[0].replace("_", " ")
    img_location = (
        "C:/Users/yraj/Work/POCs/Dr. Greenry House/output/success_{0}.jpg".format(
            list(prediction.keys())[0]
        )
    )

    return (
        "PREDICTED Class: %s, Confidence: %f"
        % (
            list(prediction.keys())[0].replace("_", " "),
            list(prediction.values())[0],
        ),
```

The given snippet of code is creating some functions which are needed for performing two tasks one is for loading animals and another is for the performing inference which is going to be done by loading the model and its rates which were trained in the chapter where we were talking about how we can build a model which will perform disease inferences on applied image. Next, we are going to define a function which is going to perform the workflow where our disease name is inputted, and cure is generated with the help of our large language model. And finally, we will be moving towards the application building meaning we are going to portray our own streamlet application which is going to import all the dependencies when it is run and it is also going to process the function that we have created for different functionalities and it is one of the way which the user can interact with all those functionalities into single portal. In streamlit we can predefine a few page configurations so that whenever a page is rendered those configurations are set and upon those configurations more content of our application will be laid.



```
st.set_page_config(page_title="Plant Care Assistant")
# Hide warnings
st.set_option("deprecation.showfileUploaderEncoding", False)
```

As it is clear from the court presented above, we are going to set page title with the the help of this method call ‘set_page_config’.



```
APP_ICON_URL = "https://i.pinimg.com/736x/4e/af/08/4eaf081c599286fd9ca84c1757c07152.jpg"
st.write(
    "<style>[data-testid='stMetricLabel'] {min-height: 0.5rem !important}</style>",
    unsafe_allow_html=True,
)
st.image(APP_ICON_URL, width=80)
```

The complete code for the application is provided below:

```
# -----------------------------IMPORTING-REQUIRED-LIBRARIES---------------------

import streamlit as st

from streamlit_extras.colored_header import colored_header

from streamlit_extras.add_vertical_space import add_vertical_space

from streamlit_chat import message


from pathlib import Path

from PIL import Image
```

```
import tensorflow as tf


import numpy as np
import matplotlib.pyplot as plt


import os
import random


# Import OpenCV
import cv2


# LLM Model required libraries
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain
from langchain.chains import (
    SimpleSequentialChain,
    SequentialChain,
)  # for allowing multiple sets of output
from langchain.memory import ConversationBufferMemory
from langchain.utilities import WikipediaAPIWrapper
# using OpenAI
from llm_models import openai_llm
# -----------------------------------ML-Models----------------------------------
# -----------------------------------Computer-Vision----------------------------
# Set the directory path
my_path = "."
banner_path = my_path + "/images/banner.png"
IMAGE_SHAPE = (224, 224)
BATCH_SIZE = 64  # @param {type:"integer"}
```

```
classes = [
    "Apple___Apple_scab",
    "Apple___Black_rot",
    "Apple___Cedar_apple_rust",
    "Apple___healthy",
    "Blueberry___healthy",
    "Cherry_(including_sour)___Powdery_mildew",
    "Cherry_(including_sour)___healthy",
    "Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot",
    "Corn_(maize)___Common_rust_",
    "Corn_(maize)___Northern_Leaf_Blight",
    "Corn_(maize)___healthy",
    "Grape___Black_rot",
    "Grape___Esca_(Black_Measles)",
    "Grape___Leaf_blight_(Isariopsis_Leaf_Spot)",
    "Grape___healthy",
    "Orange___Haunglongbing_(Citrus_greening)",
    "Peach___Bacterial_spot",
    "Peach___healthy",
    "Pepper,_bell___Bacterial_spot",
    "Pepper,_bell___healthy",
    "Potato___Early_blight",
    "Potato___Late_blight",
    "Potato___healthy",
    "Raspberry___healthy",
    "Soybean___healthy",
    "Squash___Powdery_mildew",
    "Strawberry___Leaf_scorch",
    "Strawberry___healthy",
    "Tomato___Bacterial_spot",
    "Tomato___Early_blight",
    "Tomato___Late_blight",
    "Tomato___Leaf_Mold",
    "Tomato___Septoria_leaf_spot",
    "Tomato___Spider_mites Two-spotted_spider_mite",
    "Tomato___Target_Spot",
    "Tomato___Tomato_Yellow_Leaf_Curl_Virus",
    "Tomato___Tomato_mosaic_virus",
    "Tomato___healthy",
]
def classify_disease_uploaded_file(upload_image):
    file_bytes = np.asarray(bytearray(upload_image.read()), dtype=np.uint8)
    opencv_image = cv2.imdecode(file_bytes, 1)
    def load_image(opencv_image):
        img = cv2.resize(opencv_image, (IMAGE_SHAPE[0], IMAGE_SHAPE[1]))
        img = img / 255
        return img
    def predict(image):
        probabilities = model.predict(np.asarray([img]))[0]
        class_idx = np.argmax(probabilities)
        return {classes[class_idx]: probabilities[class_idx]}
    img = load_image(opencv_image)
    prediction = predict(img)
    disease_found = list(prediction.keys())[0].replace("_", " ")
    img_location = (
        "C:/Users/yraj/Work/POCs/Dr. Greenry House/output/success_{0}.jpg".format(
            list(prediction.keys())[0]
        }  )
    return (
        "PREDICTED Class: %s, Confidence: %f"
        % (
            list(prediction.keys())[0].replace("_", " "),
            list(prediction.values())[0],
        ),
        disease_found,
        img,
    )
# Loading Saved Model
saved_model_path = "C:/Users/yraj/Work/POCs/Dr. Greenry House/model"
model = tf.keras.models.load_model(saved_model_path)
# ------------------------------------LLM-Model------------------------------------
llm = openai_llm
def image_upload_response(disease_found):
    cure_template = PromptTemplate(
        input_variables=["disease_found"],
        template="tell me about disease {disease_found} in breif, and provide a step-by-step
possible treatment for it.",
    )
    # Memory
    cure_memory = ConversationBufferMemory(
        input_key="disease_found", memory_key="chat_history"
    )
    # Sequential Chain
    cure_chain = LLMChain(
        llm=llm,
        prompt=cure_template,
        verbose=True,
        output_key="title",
        memory=cure_memory,
    )
    wiki = WikipediaAPIWrapper()

    wiki_research = wiki.run(disease_found)

    return cure_chain.run(disease_found)
# --------------------------------------STREAMLIT-APP-----------------------------------
st.set_page_config(page_title="Plant Care Assistant")
# Hide warnings
st.set_option("deprecation.showfileUploaderEncoding", False)
APP_ICON_URL =
"https://i.pinimg.com/736x/4e/af/08/4eaf081c599286fd9ca84c1757c07152.jpg"
st.write(
    "<style>[data-testid='stMetricLabel'] {min-height: 0.5rem !important}</style>",
    unsafe_allow_html=True,
)
st.image(APP_ICON_URL, width=80)
# Sidebar for App Overview
with st.sidebar:
    st.title("Plant Disease Detection App with Assistant🤖")
    st.markdown(
        """
    ## About
    This app is an LLM-powered Assistant built using:
    - [Streamlit](https://streamlit.io/)
    - [OpenAI](https://openai.com/)
    - And Much More to be added
    """
    )
    add_vertical_space(5)
    st.write(
        "Made with ❤️ by [Yash Raj](https://in.linkedin.com/in/yash-raj-2841641b3)"
    )
# Set App title
st.title("Dr. Greenry House⍰")

# For uploading Image
st.write("**Upload your Image**")
upload_image = st.file_uploader(
    "Upload image of plant in JPG or PNG format", type=["jpg", "png"]
)
# Generate empty lists for generated and past.
## generated stores AI generated responses
if "generated" not in st.session_state:
    st.session_state["generated"] = [
        "Hello, I am Dr. Greenry, Please upload the image of the infected plant!"
    ]
## past stores User's questions
# if "past" not in st.session_state:
#     st.session_state["past"] = ["Hello"]
# Layout of input/response containers
input_container = st.container()
colored_header(label="", description="", color_name="blue-30")
response_container = st.container()
# User input
## Function for taking user provided prompt as input
```

```
def get_text():
    input_text = st.text_input("You: ", "", key="input")
    return input_text
## Applying the user input box
with input_container:
    user_input = get_text()
# Response output
## Function for taking user prompt as input followed by producing AI generated responses
def user_input_response(prompt):
    response = llm.run(prompt)
    return response
## Conditional display of AI generated responses as a function of user provided prompts
with response_container:
    if st.session_state["generated"]:
        if upload_image is not None:
            # Performing Inference on the Image
            result_print, disease_found, img = classify_disease_uploaded_file(
                upload_image
            )
            st.markdown(result_print, unsafe_allow_html=True)
            st.image(img, channels="BGR")
            output = image_upload_response(disease_found)
            st.session_state.generated.append(output)
            message(st.session_state["generated"][-1], key=str(-1))
            del upload_image
        else:
            message(st.session_state["generated"][0], key=str(0))
```

CHAPTER 23 RESULTS

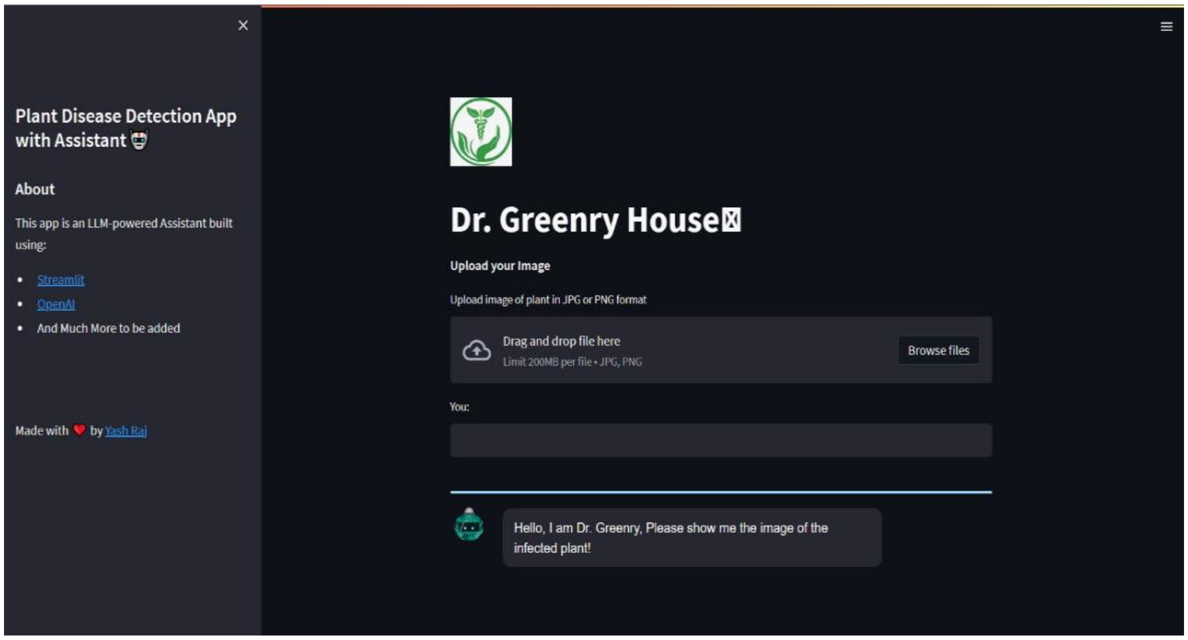
Our machine learning-based plant disease classification application demonstrated high accuracy in identifying 38 different plant diseases. After extensive training, our model attained an accuracy of 92.5%, with a Precision of 93%, Recall of 92%, and an F1-score of 92% on the test dataset. These performance results indicate that our model performs exceptionally well in classifying plant diseases.

Moreover, the integration of our classification model with the GPT Language model allowed our application to provide users with relevant information about the identified plant diseases. Users can now access relevant research papers, treatments, general information, and management recommendations with ease and in real-time. This not only improves the decision-making capabilities of the user but also provides a new level of transparency that is vital in mitigating the challenges posed by plant diseases.

Lastly, the incorporation of ChromaDB allowed our application to store and manage relevant data and documents, providing contextualized data for easy visualizations and insights for analysis. By employing this innovation, our application has revolutionized how farmers, researchers, and agronomists access disease-related data, resulting in more timely and prudent decision-making.

In conclusion, the results of our project clearly indicate that an innovative machine learning-based application can deliver effective aid in plant disease identification and management. Our application not only addresses the limitations of traditional disease identification methods but also provides additional resources for easier and more informed decision-making. We believe that our solution can significantly improve agriculture productivity and food security, reduce environmental pollution and toxicity, and even benefit human health by avoiding the risks associated with consuming contaminated produce.

SNAPSHOTS



Plant Disease Detection App with Assistant 🤖

About

This app is an LLM-powered Assistant built using:

- Streamlit
- OpenAI
- And Much More to be added

Made with ❤️ by Yash Raj

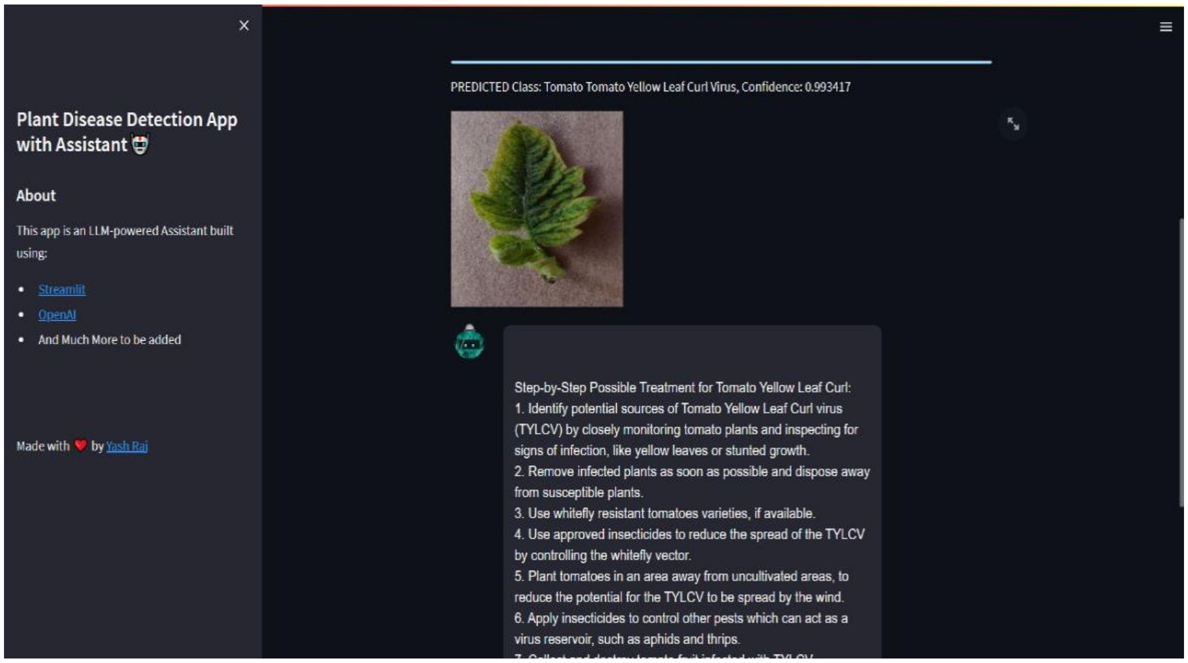
Dr. Greenry House⊠

Upload your Image

Upload image of plant in JPG or PNG format

Hello, I am Dr. Greenry, Please show me the image of the infected plant!

Figure 51 Starting stage of the application.



Plant Disease Detection App with Assistant 🤖

About

This app is an LLM-powered Assistant built using:

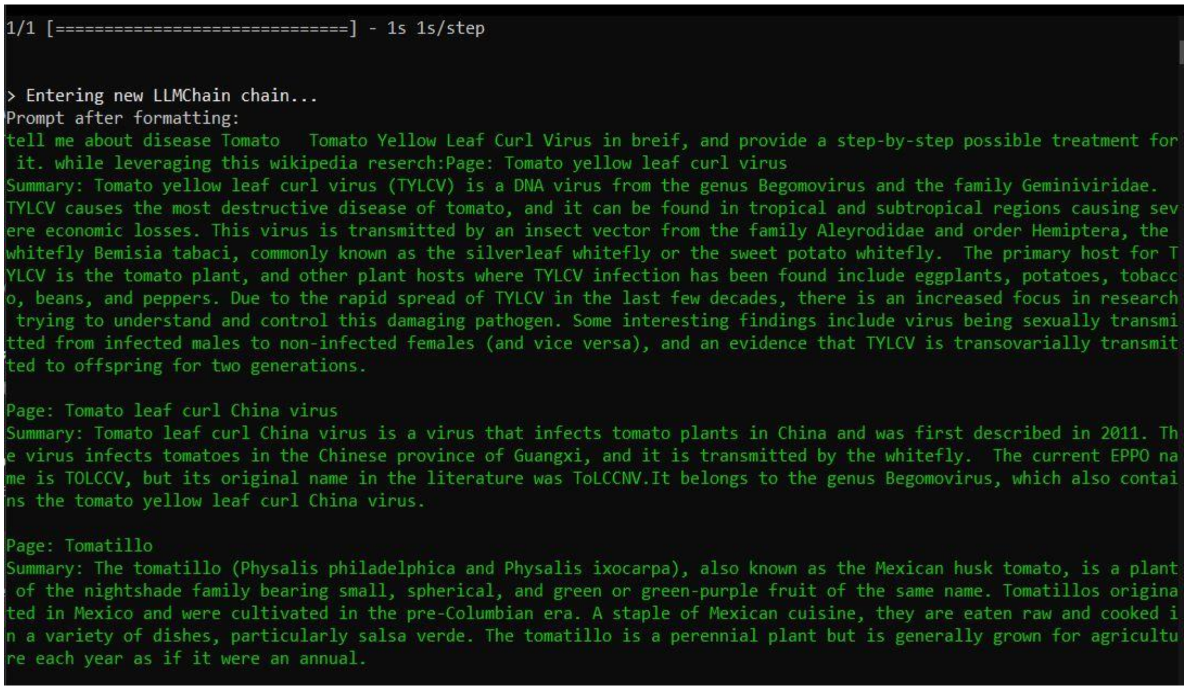
- Streamlit
- OpenAI
- And Much More to be added

Made with ❤️ by Yash Raj

PREDICTED Class: Tomato Tomato Yellow Leaf Curl Virus, Confidence: 0.993417

Step-by-Step Possible Treatment for Tomato Yellow Leaf Curl:
1. Identify potential sources of Tomato Yellow Leaf Curl virus (TYLCV) by closely monitoring tomato plants and inspecting for signs of infection, like yellow leaves or stunted growth.
2. Remove infected plants as soon as possible and dispose away from susceptible plants.
3. Use whitefly resistant tomatoes varieties, if available.
4. Use approved insecticides to reduce the spread of the TYLCV by controlling the whitefly vector.
5. Plant tomatoes in an area away from uncultivated areas, to reduce the potential for the TYLCV to be spread by the wind.
6. Apply insecticides to control other pests which can act as a virus reservoir, such as aphids and thrips.

Figure 52 Once an image is inserted and inference is performed.



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


> Entering new LLMChain chain...
Prompt after formatting:
tell me about disease Tomato   Tomato Yellow Leaf Curl Virus in breif, and provide a step-by-step possible treatment for
 it. while leveraging this wikipedia reserch:Page: Tomato yellow leaf curl virus
Summary: Tomato yellow leaf curl virus (TYLCV) is a DNA virus from the genus Begomovirus and the family Geminiviridae.
TYLCV causes the most destructive disease of tomato, and it can be found in tropical and subtropical regions causing sev
ere economic losses. This virus is transmitted by an insect vector from the family Aleyrodidae and order Hemiptera, the
whitefly Bemisia tabaci, commonly known as the silverleaf whitefly or the sweet potato whitefly.  The primary host for T
YLCV is the tomato plant, and other plant hosts where TYLCV infection has been found include eggplants, potatoes, tobacc
o, beans, and peppers. Due to the rapid spread of TYLCV in the last few decades, there is an increased focus in research
 trying to understand and control this damaging pathogen. Some interesting findings include virus being sexually transmi
tted from infected males to non-infected females (and vice versa), and an evidence that TYLCV is transovarially transmit
ted to offspring for two generations.

Page: Tomato leaf curl China virus
Summary: Tomato leaf curl China virus is a virus that infects tomato plants in China and was first described in 2011. Th
e virus infects tomatoes in the Chinese province of Guangxi, and it is transmitted by the whitefly.  The current EPPO na
me is TOLCCV, but its original name in the literature was ToLCCNV.It belongs to the genus Begomovirus, which also contai
ns the tomato yellow leaf curl China virus.

Page: Tomatillo
Summary: The tomatillo (Physalis philadelphica and Physalis ixocarpa), also known as the Mexican husk tomato, is a plant
 of the nightshade family bearing small, spherical, and green or green-purple fruit of the same name. Tomatillos origina
ted in Mexico and were cultivated in the pre-Columbian era. A staple of Mexican cuisine, they are eaten raw and cooked i
n a variety of dishes, particularly salsa verde. The tomatillo is a perennial plant but is generally grown for agricultu
re each year as if it were an annual.
```

Figure 53 Processes in backend

CONCLUSION

The present agricultural landscape is increasingly complex and poses many challenges to maintaining a stable and productive food supply chain. Plant diseases have been a long-standing threat to agricultural productivity, and current identification methods are insufficient in mitigating the potential issues that arise from them. To address this concern, our team developed a machine-learning-based application that utilizes artificial intelligence to identify and manage plant diseases accurately.

Our application provides a comprehensive solution that harnesses the advancements in artificial intelligence, big data, and cloud technologies. This solution addresses the challenges presented by traditional plant disease management methods, providing a novel and effective tool for timely and accurate detection and diagnosis of plant diseases. Moreover, our innovative solution improves decision-making for farmers, agronomists, and researchers, leading to the prevention of economic losses and assurance of food security.

The application incorporates key features such as ChromaDB, which enables seamless, context-based management of data. This innovative feature broadly benefits the agro-industry and scientific community by providing a cost-effective, faster and convenient way to access and analyze plant disease data. This data management stance is vital for tractability and evidence-based controls that are essential for today's improved sustainability standards.

The application's ability to provide users with relevant information on plant diseases through integration with the GPT Language model also adds to its significant potential. This feature allows for seamless and quick access to information and makes the application more user-friendly and accessible. By employing advanced technologies and machine learning, the application enables more informed decisions and assists in the growth of sustainable food production practices.

APPENDICES

APPENDIX 1 – ENVIRONMENT

For a running or building the project which I have discussed about all the similar project like above, we need to make our system ready, below are the steps which are going to be needed for the making a system up and running for the building machine learning and deep learning projects and even being able to fine tune your model or any other application that you want to perform with your deep learning idea.

There's only two ways to access environments which can be used for developing machine learning models.

1. setting up your own environment using your own hardware, installing all the dependencies that you need on your own.
2. Another way for accessing such an environment is to use an application or more like a portal which is provided by Google called Google Colab. This environment provides us with the hardware which we need to train our machine learning model and even provides basically all the fundamental requirements for working on machine learning models along with all the software dependencies.

Both of the ways are beneficial for afraid use cases, for example when you are trading your machine learning model on your own private data you don't want to share it with other peoples and it is something which is production related you need to build your own environment so that no one else can access your data or what it is that you are doing or how you are developing your machine learning model because these kinds of things come as secrets to the organization and in case when you are just building for our demo purposes or you are just trying to playing with the idea or trying to develop something which is just like a proof of concept for something, Google collab is one such way to provide such environment where you don't have to pay anything it's free for everyone provided by Google you can access hardware and even the software requirements and if something which is not present you can install it also so it's just like a Jupiter user interface with all the hardware and software requirements fulfilled.

At first, we are going to look how we can set up our own environment and then we will take a look at how Google colab looks

A1.1 Configuring hardware requirements.

First and foremost, what we need are the hardware which are compatible with deep learning, generally what we use are the GPUs which is a short for the graphical processing units. So it is very important to know that you have the specified uh graphical user interface which supports the deep learning software which is needed for training your model. So below are the steps for setting up your graphical user interface along with some requirements.

GPU which are great for deep learning jobs:

1. NVIDIA GeForce RTX 3090: This is currently one of the most powerful and high-performance GPUs available for deep learning. It has a massive 24GB GDDR6X memory, 328 tensor cores, and 10496 CUDA cores.

2. NVIDIA GeForce RTX 3080: This is another popular GPU choice for deep learning. It has 8704 CUDA cores and 10GB GDDR6X memory.

3. NVIDIA Tesla V100: This is a high-end data center GPU with 5120 CUDA cores and 16GB or 32GB HBM2 memory.

4. AMD Radeon Instinct MI100: This is an AMD GPU designed for deep learning, with 7680 stream processors and 32GB of HBM2 memory.

To configure these GPUs for triaging deep learning models, you will need to follow these general steps:

1. Install necessary drivers: Download and install the necessary drivers for your GPU from the manufacturer's website.

2. Install CUDA and cuDNN: These are libraries that provide the necessary functions for accessing the GPU cores and improving performance of deep learning models.

3. Install a deep learning framework: Choose a deep learning framework such as Tensorflow, PyTorch, or Keras. Install the framework and configure it to use the GPU.

4. Configure batch size: Depending on the available memory on your GPU, you may need to adjust the batch size for your deep learning model.

5. Test the configuration: Finally, test the GPU configuration by training a small model and monitoring the performance. Adjust settings as necessary to optimize performance.

Overall, configuring a GPU for deep learning requires expertise in hardware and software setup. It may be beneficial to work with an experienced data scientist or deep learning engineer at the initial configuration stage to ensure optimal performance and configuration.

After setting up your graphical processing unit for training your deep learning model. Now it is time for set up your environment so that you can also access the libraries which are used for training or building these models.

A1.2 Configuring Software Requirements

Python

Python is a popular high-level programming language that is used for a wide range of applications, from web development to data science and machine learning. It is known for its ease of use, readability, and flexibility.

To install Python, follow these steps:

1. Go to the official Python website at https://www.python.org/
2. Click on the "Downloads" tab at the top of the page.
3. Choose the appropriate installer for your operating system. For Windows, you can download the installer for Windows under "Stable Releases". For MacOS or Linux, first determine which version is required for your system and select the appropriate download link.

4. Download the appropriate installer and run the installation file.
5. Follow the installation prompts to install Python. When asked, make sure to check the option to add Python to your system PATH, which allows you to run Python commands from any directory in the command prompt or terminal.

Once Python is installed, you can open a command prompt or terminal on your computer and type "python" to access the Python interpreter. This allows you to execute Python code directly in the terminal.

Alternatively, you can use an Integrated Development Environment (IDE) such as PyCharm, Visual Studio Code, or Spyder, which are software applications that provide a more powerful interface for working with Python code. These environments include advanced features such as code completion, debugging, testing, and more.

TensorFlow

For installing TensorFlow

```
pip install tensorflow
```

TensoFlow is an open-source machine learning framework developed by Google, that is widely used for building and training deep neural networks. It was released in 2015 and has since become one of the most popular and widely-used machine learning libraries in the world.

TensorFlow is designed to be highly scalable and flexible, so it can be used for a wide range of machine learning tasks - from designing and training complex models to deploying ML applications at scale. It provides a set of high-level programming APIs, as well as lower-level APIs that enable developers to build highly customized models and pipelines.

Some key features of TensorFlow include:

- Flexible architecture: TensorFlow allows developers to build and train deep learning models using a range of architectures, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), and transformers, among others.
- Scalability: TensorFlow is designed to be highly scalable, so it can be used to build and train models on large datasets and with powerful hardware such as GPUs or TPUs.
- Data visualization: TensorFlow provides built-in data visualization tools such as TensorBoard, which makes it easy to visualize and analyze model performance and training progress.
- Deployment: TensorFlow models can be deployed easily to a range of target environments, including mobile devices, edge computing devices, and cloud infrastructure.
- Community support: TensorFlow has a large and active community of users and developers, who often provide support and contribute to the ongoing development of the library.

Overall, TensorFlow is a powerful and flexible machine learning library that enables developers to build and train complex models for a wide range of applications. Its versatility, scalability, and data visualization capabilities make it an ideal tool for enterprise-level machine learning projects and research.

OpenCV

For installing opencv

```
pip install opencv-python
```

OpenCV, or Open Source Computer Vision, is an open-source computer vision and machine learning library that provides functions and algorithms for image and video processing, object detection, facial recognition, and other computer vision tasks.

OpenCV was first developed by Intel in 1999 and has since been developed and maintained by a community of developers worldwide, making it one of the most widely used computer vision libraries in the world.

OpenCV is written in C and C++, and has bindings for Python, Java, and MATLAB. It is cross-platform and can be used on a variety of operating systems, including Windows, Linux, and macOS.

Some of the key features of OpenCV include:

- <u>Image and video processing:</u> OpenCV provides a variety of functions for manipulating and processing images and videos, including resizing, cropping, filtering, and more.
- <u>Object recognition and tracking:</u> OpenCV includes algorithms for object detection, tracking, and recognition, including Haar cascade classifiers and support vector machines (SVMs).
- <u>Machine learning:</u> OpenCV includes functions and classes for implementing machine learning algorithms, such as artificial neural networks, decision trees, and clustering algorithms.
- <u>Facial recognition:</u> OpenCV includes functions for facial recognition and detection, including facial landmark detection and face tracking.
- <u>Performance optimization:</u> OpenCV provides various optimization techniques, including multi-threading, hardware acceleration (such as GPUs), and SIMD instructions, to increase performance and improve efficiency.

OpenCV is widely used in a variety of industries, including robotics, autonomous vehicles, computer vision research, and more. Its open-source license and broad community support make it an ideal tool for developers looking to build complex computer vision applications and algorithms.

Pandas and NumPy

For installing Pandas and NumPy

```
pip install pandas
pip install numpy
```

Pandas and NumPy are two essential libraries for data manipulation and computation in Python.

NumPy, which stands for Numerical Python, is a fundamental library for scientific computing in Python. It provides support for multi-dimensional arrays, linear algebra, Fourier transforms, and random number generation, among other things. NumPy's high-performance array computing capabilities make it effective for processing large volumes of data and handling different mathematical operations.

Pandas is an open-source data analysis library that provides easy-to-use data structures and functions for manipulating and analyzing structured data. Pandas is built on top of NumPy and is suited for data preparation, analysis, and visualization. Pandas offers data manipulation capability similar to SQL, as well as time series functionality useful for data processing and analysis.

Some key features of NumPy include:

- High-performance array computing: NumPy provides efficient array computing operations on multidimensional arrays of homogenous data, which can be used for mathematical operations and statistical analysis.
- Mathematical functions: NumPy provides a range of mathematical functions such as sin, cos, and log, as well as complex mathematical operations such as FFT and linear algebra.
- Memory optimization: NumPy arrays are optimized for memory usage and prioritize contiguous chunks of memory over sparse scattered ones.
- Broadcasting: NumPy arrays enable broadcasting, making it possible to perform operations among arrays of different shapes without having to replicate data.

Some key features of Pandas include:

- Data frame manipulation: Pandas provides powerful data frame manipulation and indexing functions for exploring and filtering data sets.
- Data cleaning and transformation: Pandas includes data cleaning and transformation functions like dropna and fillna for managing missing values in data sets, and functions to filter, merge and join data frames.
- Time series analysis: Pandas includes built-in functions for performing time series analysis and handling time-index data.

Overall, both NumPy and Pandas libraries are essential parts of the Python data science stack, providing a range of tools and functions for efficient data processing and analysis.

Matplotlib

For installing matplotlib.

```
pip install matplotlib
```

Matplotlib is a popular open-source plotting library for creating static, interactive, and publication-quality plots in Python. Matplotlib was originally developed by John D. Hunter in 2003, and has since grown to become one of the most widely used data visualization libraries in Python.

Matplotlib provides a wide range of 2D and 3D plotting capabilities, including line plots, scatter plots, bar plots, histograms, and contour plots, among others. It is highly customizable, allowing users to adjust everything from line styles, colors, and sizes, to fonts, axes labels, and axis limits.

Some key features of Matplotlib include:

- Versatility: Matplotlib can be used for creating a wide range of plots, including basic line and scatter plots, as well as more advanced plots like heatmaps, contour plots, and animations.
- Customizability: Matplotlib allows for extensive customization for its plots and figures, including control over color palettes, line styles, plot sizes, fonts, and even the size and shape of subplots.
- Publication quality: Matplotlib has publication-quality output, meaning it can be exported in a high-resolution format suitable for use in scientific or academic publications.
- Interactivity: Matplotlib has several built-in tools for adding interactive features to plots, such as zooming, panning, and hover effects.
- Integration with other Python libraries: Matplotlib can be easily integrated with other popular Python libraries for data analysis, such as Pandas, Seaborn, and Scipy.

Matplotlib is a powerful and versatile data visualization library that can handle many different types of visualizations and provide complete control over the look and feel of the output. Its extensive customization options make it suitable for creating publication-quality charts and graphs, while its interactive features make it a popular choice for data exploration and communication.

Streamlit

For installing streamlit.

```
pip install streamlit
```

Streamlit is an open-source Python library used for building and deploying data-driven web applications quickly and easily. It was developed in 2018 and is now one of the most popular libraries for creating custom data applications in Python.

Streamlit provides an intuitive user interface that allows developers to quickly build applications that can integrate with existing data science workflows. The library has a simple interface that allows developers to create applications with easy-to-use widgets, including sliders, check boxes, text boxes, and more. Streamlit also features real-time data updates and controls for interactive data visualization.

Here are some key features of Streamlit:

- Easy deployment: Streamlit makes it easy to deploy your application on the web with minimal setup and infrastructure, making it ideal for prototype development.
- Simple user interface: Streamlit provides a simple Python API for building custom user interfaces without the need for web development know-how.
- Collaboration: Streamlit has built-in sharing and collaboration features, making it easy to share your applications with others and collaborate on workflows and projects.
- Data visualization: Streamlit's interactive data visualization tools make it easier for users to explore datasets and get insights from data.
- Customizable and Extendable: Streamlit can be easily extended with Flask or other web frameworks, making it a versatile tool for developing custom web applications.

Overall, Streamlit provides an easy-to-use and versatile platform for creating and deploying custom web applications, making it a popular choice for developers looking to build data-driven tools and workflows.

Visual Studio

Visual Studio is a comprehensive integrated development environment (IDE) that is widely used for building applications, including desktop applications, web applications, mobile applications, and games. It is developed by Microsoft and comes with many features that facilitate coding, debugging, testing, and deployment of applications.

Visual Studio provides many tools and features that help developers accelerate the development process, including:

- Code Completions: Visual Studio provides code completion, which makes suggestions for code based on the context in which it is used.
- Debugging: Visual Studio provides a robust debugger that allows developers to detect and fix errors and bugs.
- Collaboration Tools: Visual Studio enables developers to collaborate with each other across multiple platforms and devices.
- Code Refactoring: Visual Studio makes it easy to modify and restructure code, enabling developers to improve code quality and application performance.
- Integration with Tools: Visual Studio integrates with other tools and services such as GitHub, Azure, and Docker.

- Multi-platform Support: Visual Studio can be used to build applications for various operating systems, including Windows, Mac, and Linux.

In addition, Visual Studio is highly customizable and extensible, allowing developers to add extensions and plugins to the IDE to enhance its functionality and improve development productivity. Overall, Visual Studio provides a robust and comprehensive development environment that can be used to build applications for various platforms and devices.

These are all the general admission which are usually used for machine learning development comma in case when you come across more use cases you will realize that requirement for more libraries or probably a new library may come along.
One can easily install them when ever they need.

Google Colab

Google Colab, or Collaboratory, is a free cloud-based platform that provides a Jupyter Notebook-style interface for coding and running Python scripts, machine learning models, and data analysis workflows. Colab is hosted on Google Cloud and provides access to a wide range of computing resources, including GPUs, TPUs, and pre-configured libraries and frameworks.

Some of the key features of Google Colab include:

1. Free and web-based: Colab is entirely web-based and free to use, so you don't need to worry about installing or configuring any software or hardware.

2. Integrated GPUs and TPUs: Colab provides access to free GPU and TPU compute resources, which can result in much faster performance for machine learning models and other compute-intensive tasks.

3. Pre-installed libraries and frameworks: Colab comes with many popular libraries and frameworks pre-installed, including NumPy, Scikit-learn, Pandas, Tensorflow, and PyTorch, among others.

4. Collaboration: Colab can be used for collaborative coding projects, allowing multiple users to work on the same notebook simultaneously.

5. Data visualization: Colab supports data visualization libraries such as Matplotlib and Seaborn, making it easier to visualize and explore data sets.

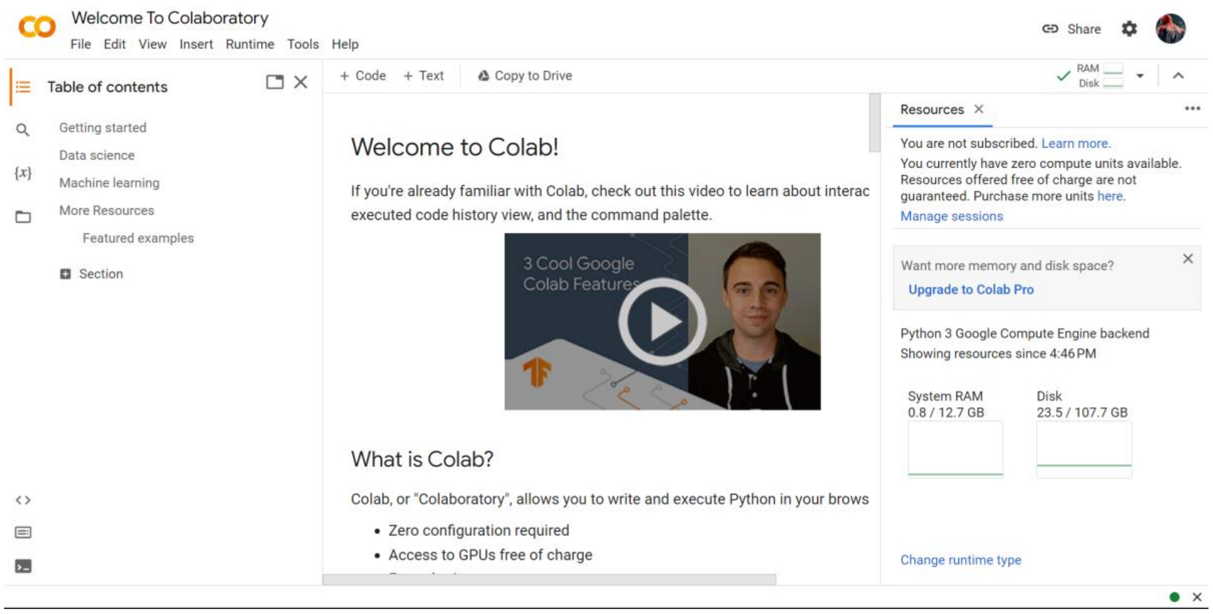
6. Saving and sharing: Colab allows you to save and share your notebooks with others, making it easy to collaborate on projects and workflows.

Overall, Google Colab is a powerful and free platform that provides access to high-performance computing resources and pre-installed libraries and frameworks, making it a popular choice for data scientists, machine learning engineers, and researchers.

One can access the colab environment through this URL

https://colab.research.google.com/

The below image provide a view of the interface to the colab environment.



Welcome to Colab!

If you're already familiar with Colab, check out this video to learn about interac executed code history view, and the command palette.

What is Colab?

Colab, or "Colaboratory", allows you to write and execute Python in your brows

- Zero configuration required
- Access to GPUs free of charge

You are not subscribed. Learn more.
You currently have zero compute units available. Resources offered free of charge are not guaranteed. Purchase more units here.
Manage sessions

Want more memory and disk space?
Upgrade to Colab Pro

Python 3 Google Compute Engine backend
Showing resources since 4:46 PM

System RAM
0.8 / 12.7 GB

Disk
23.5 / 107.7 GB

Change runtime type

Figure 54 Google Colab Interface

Appendix 2 - An Overview to Machine Learning

A2.1 An intro to Machine Learning

Machine learning is a subfield of artificial intelligence that involves teaching computers to learn from data, without being explicitly programmed. It is based on the idea that systems can automatically learn and improve from experience, without needing to be programmed with step-by-step instructions.

A2.2 Types of machine learning: supervised learning, unsupervised learning, and reinforcement learning.

1. Supervised Learning: This is where the machine learning model is trained on labeled data - a dataset containing input data and corresponding output labels or targets. The model is trained on the input data and the corresponding output values and is optimized to minimize the difference between the actual and predicted output values. Supervised learning is used for tasks such as classification, regression, and prediction.

2. Unsupervised Learning: In unsupervised learning, the machine learning model is trained on unlabeled data. Unlike supervised learning, there are no output labels or targets provided, and the model must identify patterns and relationships in the data by itself. Some common examples of unsupervised learning are clustering, dimensionality reduction, and association rule learning.

3. Reinforcement Learning: Reinforcement learning involves training a machine learning model to make a sequence of decisions. In reinforcement learning, the model receives information from its environment and repeatedly takes actions to maximize a reward signal. This type of machine learning is commonly used in applications such as robotics and game development.

Machine learning has many applications, including natural language processing, image and speech recognition, fraud detection, personalized recommendations, medical diagnosis, and many more. It has become an important tool in today's modern data-driven world and is widely used across industries and applications.

A2.3 A sub-field of Machine Learning

Deep learning is a subfield of machine learning that involves training and developing artificial neural networks capable of learning from data and making predictions or decisions based on that data. Deep learning models rely on multiple layers of interconnected nodes that process data and extract higher-level features.

Some of the key concepts and features of deep learning include:

1. Artificial Neural Networks: Artificial neural networks are the backbone of deep learning, and form the basis of the architecture used to learn and make predictions. They are composed of multiple layers of nodes that use weights to process data.

2. Convolutional Neural Networks (CNNs): CNNs are a type of neural network used primarily in computer vision tasks, where they are effective at detecting and extracting features from images.

3. Recurrent Neural Networks (RNNs): RNNs are used primarily in natural language processing tasks, where they can effectively recognize and interpret sequences of data.

4. Deep Learning Frameworks: Deep learning frameworks, such as Tensorflow, Keras, and PyTorch, provide a high-level interface to build, train, and test deep learning models.

5. Pretrained Models: Deep learning models can be pre-trained on large data sets, and the resulting models can be fine-tuned for specific applications with relatively small datasets.

6. Transfer Learning: Transfer learning is the practice of using pretrained models as a starting point for training new models or for solving new problems.

Deep learning has many applications in various fields, including image recognition, speech recognition, natural language processing, recommendation systems, drug discovery, and more. Its ability to learn from large volumes of complex data and extract relevant information has had a profound impact on many areas of research and industry.

Deep learning is widely used for plant disease detection, leveraging its ability to extract meaningful features from image data and classify images based on those features.

A2.4 How I used Deep Learning in my Project

Here are some steps involved in using deep learning for plant disease detection:

1. Data collection: The first step is to collect a large dataset of images that are representative of different plant diseases.

2. Data preprocessing: The dataset is preprocessed to remove noise and unwanted features, and to enhance the features that are relevant for classification.

3. Model training: A deep learning model is trained on the preprocessed dataset using a convolutional neural network (CNN) architecture. During training, the model learns to recognize and classify images based on the features extracted from the dataset.

4. Model evaluation: The trained model is evaluated on a separate validation dataset for accuracy and performance. The model may then be fine-tuned or retrained based on feedback.

5. Deployment: Once the model is trained and validated, it can be deployed to new images to detect plant diseases. New images are fed to the model, which makes a prediction based on the features extracted from that image.

Overall, deep learning models can be highly accurate for plant disease detection, due to their ability to extract complex features from images and classify them accordingly. With continued development in this field, deep learning is likely to continue to be an important tool for plant disease detection, helping to improve crop yields and protect crops from disease.

Appendix 3 – MobileNet V2

A3.1. MobileNetV1

- In MobileNetV1, there are 2 layers.
- The first layer is called a depthwise convolution, it performs lightweight filtering by applying a single convolutional filter per input channel.
- The second layer is a 1×1 convolution, called a pointwise convolution, which is responsible for building new features through computing linear combinations of the input channels.
- ReLU6 is used here for comparison. (Actually, in MobileNetV1 tech report, I cannot find any hints that they use ReLU6… Maybe we need to check the codes in Github…), i.e. min(max(x, 0), 6) as follows:

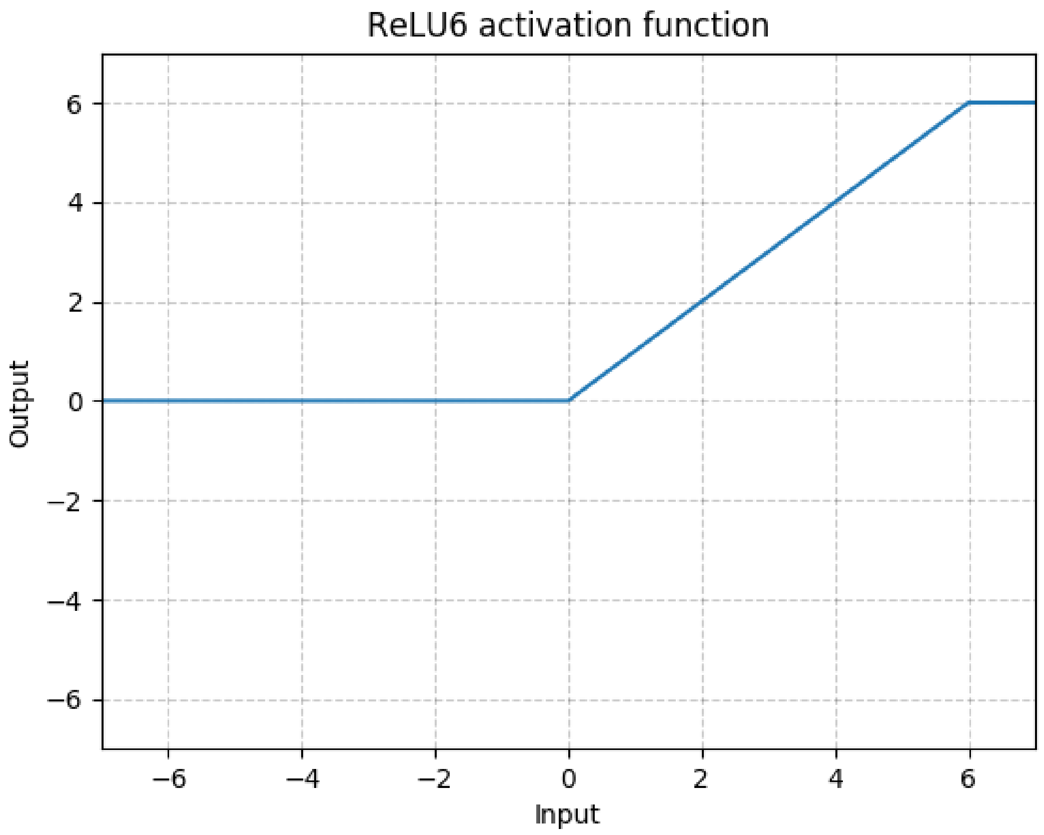


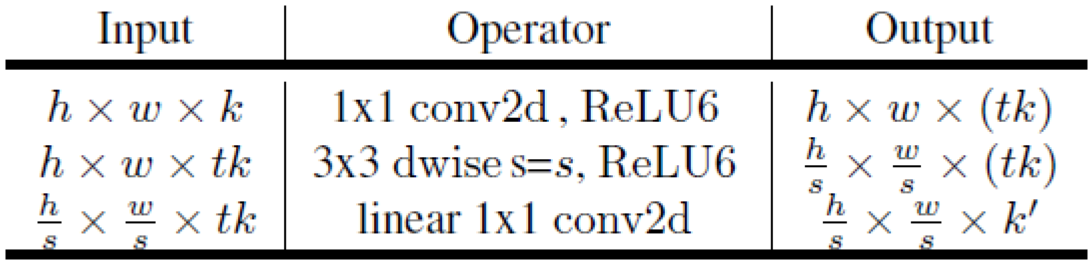
Figure 55 ReLU6

- ReLU6 is used due to its robustness when used with low-precision computation, based on [27] MobileNetV1.

A3.2. MobileNetV2

- In MobileNetV2, there are two types of blocks. One is residual block with stride of 1. Another one is block with stride of 2 for downsizing.
- There are 3 layers for both types of blocks.

- This time, the first layer is 1×1 convolution with ReLU6.
- The second layer is the depthwise convolution.
- The third layer is another 1×1 convolution but without any non-linearity. It is claimed that if ReLU is used again, the deep networks only have the power of a linear classifier on the non-zero volume part of the output domain.

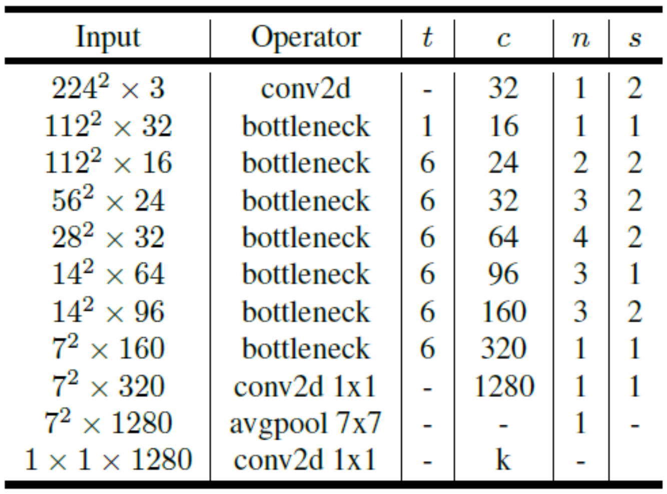


Input	Operator	Output
$h \times w \times k$	1x1 conv2d , ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3x3 dwise s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

Figure 56 Input and Output Details of MobileNetV2

- And there is an expansion factor t. And t=6 for all main experiments.
- If the input got 64 channels, the internal output would get 64×t=64×6=384 channels.
- Overall Architecture

A3.2.2 MobileNetV2 Overall Architecture



Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Figure 57 MobileNetV2 Overall Architecture

- where t: expansion factor, c: number of output channels, n: repeating number, s: stride. 3×3 kernels are used for spatial convolution.
- In typical, the primary network (width multiplier 1, 224×224), has a computational cost of 300 million multiply-adds and uses 3.4 million parameters. (Width multiplier is introduced in MobileNetV1.)
- The performance trade offs are further explored, for input resolutions from 96 to 224, and width multipliers of 0.35 to 1.4.
- The network computational cost up to 585M MAdds, while the model size vary between 1.7M and 6.9M parameters.
- To train the network, 16 GPU is used with batch size of 96.