Agricultural Crop Disease Detection

PROBLEM STATEMENT: Create a model to classify and identify plant diseases from images of crops, providing farmers with timely information to take preventive measures and improve crop health.

1.ABSTRACT: This work presents a machine learning model, in the form of a Kaggle Notebook. The detection of crop diseases is a crucial aspect of modern agriculture, significantly impacting crop yield and food security. This report explores the application of **convolutional neural networks (CNNs)** for automated plant disease recognition, leveraging deep learning techniques to enhance the accuracy and efficiency of disease diagnosis.

Through rigorous data preprocessing, model training, and evaluation, the CNN achieved a commendable accuracy of approximately **98%** on training datasets and **95%** on validation dataset. This indicates the model's potential as a valuable tool for farmers and agronomists, facilitating early detection of crop diseases. The findings suggest that integrating such automated systems into agricultural practices can lead to timely interventions, ultimately reducing losses and improving crop management strategies.

Future work will focus on addressing challenges such as class imbalance and enhancing model generalization to diverse environmental conditions.

2.DATASET:
https://www.kaggle.com/datasets/vipoooool/new-plant-diseases-dataset

The study utilizes a publicly available dataset consists of about **87K** rgb images of healthy and diseased crop leaves which is categorized into 38 different classes. The total dataset is divided into 80/20 ratio of training and validation set preserving the directory structure. A new directory containing 33 test images is created later for prediction purpose**.**

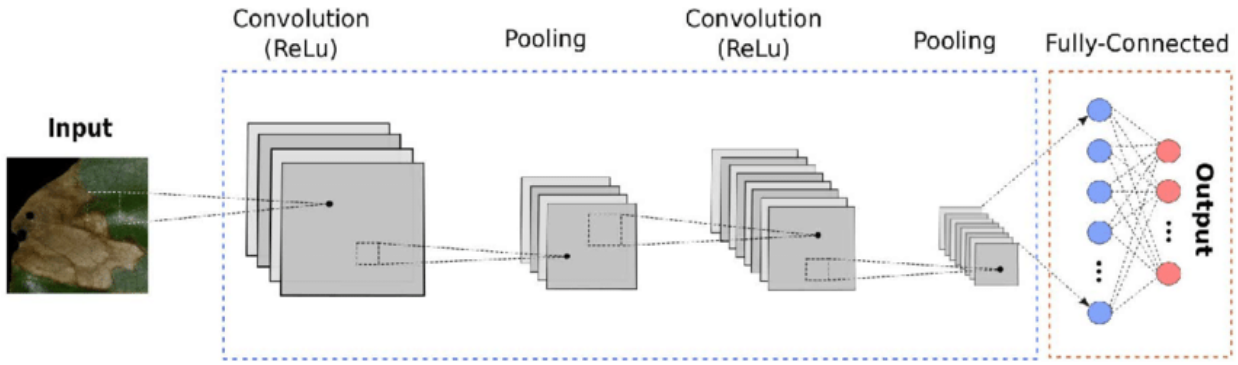
3.KEYWORDS: Deep Learning, Convolutional Neural Networks (CNNs), Image classification, Feature extraction etc.

4.INTRODUCTION: Plant diseases pose a significant threat to global agriculture, accounting for a substantial percentage of crop loss each year.

This project aims to develop a robust CNN model capable of classifying various plant diseases based on image input. By utilizing a comprehensive dataset of plant images, the model seeks to achieve high accuracy rates in disease detection. The importance of this research lies not only in improving disease diagnosis but also in promoting sustainable agricultural practices through timely and informed decision-making. The results of this study will contribute to the ongoing efforts in precision agriculture, where technology plays a pivotal role in enhancing productivity and minimizing resource wastage.

4.1EARLY METHODS: Traditional methods of disease diagnosis rely heavily on visual inspection by trained experts, which can be time-consuming, subjective, and prone to human error. As the demand for food continues to rise due to population growth, the need for efficient and reliable disease detection systems becomes increasingly critical.

4.2 CNN IN DISEASE DETECTION:



A **Convolutional Neural Network (CNN)** is a specialized type of **deep learning** model designed for processing structured grid data, such as images or video frames. CNNs are a subset of **artificial neural networks (ANNs)** that have shown exceptional performance in tasks like image recognition, object detection, time-series analysis, text classification and in our case **Crop disease detection**.

CNN ARCHITECTURE IN DISEASE DETECTION:

- **INPUT IMAGE**: The image is passed through the network. the input is a 128x128 image of a plant leaf.

- **CONVOLUTION**: A filter (3x3 matrix) is applied to the image. The filter detects basic features like edges or textures. After sliding over the image with strider of 1 pixel we get a feature map.

- **POOLING**: The feature map is passed through a pooling layer (2x2 max pooling), which reduces the size of the feature map by retaining only the most important features.

- **MULTIPLE CONVOLUTIONS**: The process is repeated with multiple convolutional layers. As you move deeper into the network, the model begins to recognize more complex features, like parts of the plant or disease patterns.

- **FLATTENING AND FULLY CONNECTED LAYER**: After several convolution and pooling layers, the feature map is "flattened" into a 1D array. This flattened data is then passed to a fully connected layer, where neurons are fully connected, just like in a traditional neural network.

- **OUTPUT LAYER**: The output layer generates a probability distribution over the possible classes (e.g., "Healthy", "Diseased" with a confidence level for each). The class with the highest probability is the model's prediction.

ADVANTAGES:

- **AUTOMATIC FEATURE EXTRACTION**: CNNs automatically learn and extract relevant features (e.g., texture, colour, and shape) from raw image data, eliminating the need for manual feature engineering. This makes CNNs highly efficient in recognizing complex patterns in plant disease symptoms.

- **SCALABILITY WITH LARGE DATASETS**: CNNs can handle large-scale image datasets (e.g., 87k of plant images) and scale well for real-time processing. They can be deployed across large agricultural areas making them ideal for large-scale field monitoring.

- **HIGH GENERALIZATION CAPABILITY**: CNNs can generalize well across various plant species and environmental conditions. Once trained on a diverse dataset, they can accurately detect diseases across different crops and regions, even with variations in lighting, angle, or background.

- **END-TO-END LEARNING PROCESS**: CNNs provide an end-to-end learning process, meaning they directly learn from raw images to final predictions without needing intermediate steps. This simplifies the workflow, streamlining the development of disease detection models and making them easier to deploy in practical scenarios.

CONCLUSION:

Convolutional Neural Networks (CNNs) are powerful tools for plant disease detection, offering automated, efficient, and highly accurate ways to identify diseases from images. With advances in deep learning, CNNs have proven to be indispensable in agricultural applications, helping farmers researchers, and agronomists detect diseases early, reduce crop losses, and improve yield predictions.

5.PROCEDURE INVOLVED IN TRAINING :

5.1 IMPORTING LIBRARIES:

```
import tensorflow as tf
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

- **TENSORFLOW:** An open-source library for numerical computation and machine learning, primarily used for building and training neural networks.
- **MATPLOTLIB**: A plotting library in Python that provides an object-oriented API for embedding plots into applications and creating static, interactive, and animated visualizations**.**
- **NUMPY:** A core scientific computing library for Python, providing support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on them.
- **PANDAS:** A powerful data manipulation and analysis library for Python, offering data structures like dataframes for handling structured data efficiently.

5.2 DATA PREPROCESSING:

TRAINING IMAGE PREPROCESSING:

```
training_set = tf.keras.utils.image_dataset_from_directory(
    'train',
    labels="inferred",
    label_mode="categorical",
    class_names=None,
    color_mode="rgb",
    batch_size=32,
    image_size=(128, 128),
    shuffle=True,
    seed=None,
    validation_split=None,
    subset=None,
    interpolation="bilinear",
    follow_links=False,
    crop_to_aspect_ratio=False
)
```

Found 70295 files belonging to 38 classes.

VALIDATION IMAGE PREPROCESSING

```
validation_set = tf.keras.utils.image_dataset_from_directory(
    'valid',
    labels="inferred",
    label_mode="categorical",
    class_names=None,
    color_mode="rgb",
    batch_size=32,
    image_size=(128, 128),
    shuffle=True,
    seed=None,
    validation_split=None,
    subset=None,
    interpolation="bilinear",
    follow_links=False,
    crop_to_aspect_ratio=False
)
```

Found 17572 files belonging to 38 classes.

For Image data loading we use **(image_dataset_from_directory)** function

SOME IMPORTANT PARAMETERS:

- **LABELS**: If labels "inferred" it should contain subdirectories, each containing images for a class. (labels are generated from the directory structure).

- **LABEL_MODE**: String describing the encoding of labels. "categorical" means that the labels are encoded as a categorical vector (e.g. for categorical_crossentropy loss).

- **IMAGE_SIZE**: Size to resize images to after they are read from disk, specified as (height, width). Since the pipeline processes batches of images that must all have the same size, this must be provided. Here it is (128 x 128).

- **COLOR_MODE**: colour mode is "rgb", there are 3 channels in the image tensors.

- **BATCH SIZE**: batch size controls how many samples are processed at once and influences the model's performance, training time, and memory usage. Using batches instead of the entire dataset helps make training more efficient, especially for large datasets. Here batch size=32 means 32 images are processed at once.

5.3 BUILDING MODEL:

```
cnn = tf.keras.models.Sequential()
```

This creates a **Sequential** model in Keras, which is a linear stack of layers where each layer has exactly one input and one output.

5.4 BUILDING CONVOLUTION LAYER:

EXTRACTING FEATURES:

```
cnn.add(tf.keras.layers.Conv2D(filters=32,kernel_size=3,padding='same',activation='relu',input_shape=[128,128,3]))
cnn.add(tf.keras.layers.Conv2D(filters=32,kernel_size=3,activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2))
```

```
cnn.add(tf.keras.layers.Conv2D(filters=64,kernel_size=3,padding='same',activation='relu'))
cnn.add(tf.keras.layers.Conv2D(filters=64,kernel_size=3,activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2))
```

```
cnn.add(tf.keras.layers.Conv2D(filters=128,kernel_size=3,padding='same',activation='relu'))
cnn.add(tf.keras.layers.Conv2D(filters=128,kernel_size=3,activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2))
```

```
cnn.add(tf.keras.layers.Conv2D(filters=256,kernel_size=3,padding='same',activation='relu'))
cnn.add(tf.keras.layers.Conv2D(filters=256,kernel_size=3,activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2))
```

```
cnn.add(tf.keras.layers.Conv2D(filters=512,kernel_size=3,padding='same',activation='relu'))
cnn.add(tf.keras.layers.Conv2D(filters=512,kernel_size=3,activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2))
```

Here with the help of Convolutional Layers and MaxPooling layers we are extracting some important features that will help in recognising the disease.

FILTERS:

As the model goes deeper, the number of filters increases. This is a common practice, as deeper layers are meant to learn more complex, abstract features.

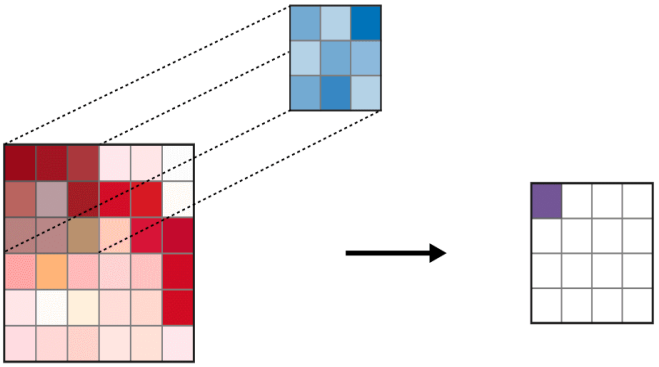
Hence, Filters increases by 32,64,128,256,512

KERNEL SIZE:

Here kernel size is of 3 x 3 matrix that will slides over the input image or feature map to detect specific patterns, such as edges, textures, or more complex features in deeper layers of the network.

STRIDES:

Here the strides=2 it means the filter moves two pixel at a time both horizontally and vertically.



ACTIVATION=ReLU

The **ReLU (Rectified Linear Unit)** activation function is one of the most commonly used activation functions in deep learning models, especially in convolutional neural networks (CNNs). It introduces non-linearity into the model, which allows the network to learn complex patterns in the data.

$g(z)=\max(0,z)$

FLATTENING AND FULLY CONNECTED LAYER:

```
cnn.add(tf.keras.layers.Dropout(0.25))
```

```
cnn.add(tf.keras.layers.Flatten())
```

```
cnn.add(tf.keras.layers.Dense(units=1500,activation='relu'))
```

```
cnn.add(tf.keras.layers.Dropout(0.4)) #To avoid overfitting
```

```
#Output Layer
cnn.add(tf.keras.layers.Dense(units=38,activation='softmax'))
```

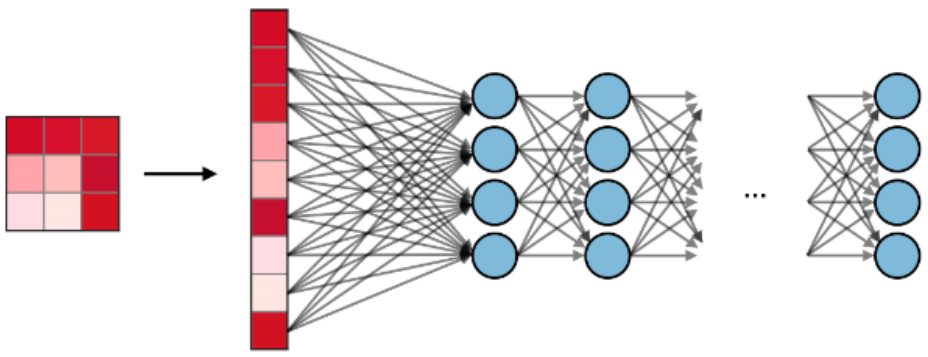
In fully connected layer here comes the distribution of probability, the maximum probability leads to prediction.

DROPUT:

The primary goal of dropout is to prevent the model from **overfitting** to the training data. Overfitting occurs when the model learns the training data too well, including the noise and irrelevant details, causing it to perform poorly on new, unseen data.

Flatten:

Converts the 2D feature map (from CNN layers) into a 1D vector to pass to fully connected layers.



DENSE LAYER:

It connects every neuron in one layer to every neuron in the next layer. This makes it a powerful tool for learning complex, high-level features**.(As shown in above figure)**

Units: 1500 neurons in intermediate hidden layer

Units: 38 output neurons (for a multi-class classification problem with 38 classes).

ACTIVATION=SOFTMAX:

Softmax activation, suitable for multi-class classification, outputs probabilities summing to 1.

5.5 COMPILING AND TRAINING PHASE:

```
cnn.compile(optimizer=tf.keras.optimizers.legacy.Adam(
    learning_rate=0.0001),loss='categorical_crossentropy',metrics=['accuracy'])
```

Compile is how we define and configure the necessary components (optimizer, loss function, and metrics) to train our model effectively.

It tells the framework how to learn from the data (via the optimizer and loss function) and how to evaluate its progress (via metrics).

LOSS FUNCTION: CATEGORICAL_CROSSENTROPY

categorical_crossentropy is appropriate for multi-class classification problems.

LEARNING RATE:

The learning rate in machine learning is a hyperparameter that controls the step size at each iteration while moving toward a minimum of the loss function. It determines how quickly or slowly a model learns. A high learning rate speeds up learning but risks overshooting the minimum, while a low rate ensures more precise convergence but can lead to longer training times or getting stuck in local minima**.**

By default, the learning rate for the Adam optimizer in TensorFlow/ Keras is 0.001.

But,with this learning rate our loss is gradually increasing in each epochs which leads to overshooting problem so here we have taken 0.0001 learning rate.

CONVOLUTIONAL LAYER SUMMARY:

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 128, 128, 32)	896
conv2d_1 (Conv2D)	(None, 126, 126, 32)	9,248
max_pooling2d (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_2 (Conv2D)	(None, 63, 63, 64)	18,496
conv2d_3 (Conv2D)	(None, 61, 61, 64)	36,928
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_4 (Conv2D)	(None, 30, 30, 128)	73,856
conv2d_5 (Conv2D)	(None, 28, 28, 128)	147,584
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 128)	0
conv2d_6 (Conv2D)	(None, 14, 14, 256)	295,168
conv2d_7 (Conv2D)	(None, 12, 12, 256)	590,080
max_pooling2d_3 (MaxPooling2D)	(None, 6, 6, 256)	0
conv2d_8 (Conv2D)	(None, 6, 6, 512)	1,180,160
conv2d_9 (Conv2D)	(None, 4, 4, 512)	2,359,808
max_pooling2d_4 (MaxPooling2D)	(None, 2, 2, 512)	0
dropout (Dropout)	(None, 2, 2, 512)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 1500)	3,073,500
dropout_1 (Dropout)	(None, 1500)	0
dense_1 (Dense)	(None, 38)	57,038

Total params: 7,842,762 (29.92 MB)

Trainable params: 7,842,762 (29.92 MB)

Non-trainable params: 0 (0.00 B)

5.6 TRAINING MODEL:

```
training_history = cnn.fit(x=training_set,validation_data=validation_set,epochs=10)
```

```
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 155s 65ms/step - accuracy: 0.3878 - loss: 2.1742 - val_accuracy: 0.8534 - val_loss: 0.4612
Epoch 2/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 65s 29ms/step - accuracy: 0.8292 - loss: 0.5440 - val_accuracy: 0.8945 - val_loss: 0.3337
Epoch 3/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 65s 29ms/step - accuracy: 0.9029 - loss: 0.3004 - val_accuracy: 0.9158 - val_loss: 0.2677
Epoch 4/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 65s 30ms/step - accuracy: 0.9329 - loss: 0.2051 - val_accuracy: 0.9402 - val_loss: 0.1874
Epoch 5/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 68s 31ms/step - accuracy: 0.9474 - loss: 0.1553 - val_accuracy: 0.9479 - val_loss: 0.1745
Epoch 6/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 72s 33ms/step - accuracy: 0.9610 - loss: 0.1176 - val_accuracy: 0.9537 - val_loss: 0.1561
Epoch 7/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 67s 31ms/step - accuracy: 0.9699 - loss: 0.0933 - val_accuracy: 0.9557 - val_loss: 0.1500
Epoch 8/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 65s 30ms/step - accuracy: 0.9737 - loss: 0.0800 - val_accuracy: 0.9635 - val_loss: 0.1223
Epoch 9/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 64s 29ms/step - accuracy: 0.9778 - loss: 0.0666 - val_accuracy: 0.9582 - val_loss: 0.1352
Epoch 10/10
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 65s 29ms/step - accuracy: 0.9807 - loss: 0.0594 - val_accuracy: 0.9586 - val_loss: 0.1456
```

By doing various changes in parameter i.e (decreasing learning rate ,increases number of neurons and Added more Convolutional Layer to extract more feature from images there may be possibility that model unable to capture relevant feature or model is confusing due to lack of feature so feed with more feature) finally our model accuracy increases by 98%.

5.7 EVALUATING MODEL:

Evaluating a model is a critical step in the machine learning and deep learning pipeline. It helps to assess how well our model is performing after training and provides insights into how it might generalize to new, unseen data.

EVALUATING TRAINING SET ACCURACY:

```
train_loss, train_acc = cnn.evaluate(training_set)
print('Training accuracy:', train_acc)
```

```
2197/2197 ━━━━━━━━━━━━━━━━━━━━ 97s 44ms/step - accuracy: 0.9830 - loss: 0.0530
Training accuracy: 0.984749972820282
```

EVALUATING VALIDATION SET ACCURACY:

```
val_loss, val_acc = cnn.evaluate(validation_set)
print('Validation accuracy:', val_acc)
```

```
550/550 ━━━━━━━━━━━━━━━━━━━━ 23s 42ms/step - accuracy: 0.9571 - loss: 0.1529
Validation accuracy: 0.9586273431777954
```

While evaluating our model it can be seen that our model performs well with Training Accuracy of 98% and Validation accuracy of 95%.

TRAINING HISTORY:

training_history.history It basically gives the history of all epochs in dictionary format .

further, we dump our history in **(training_hist.json)** Json format for any further use.

```
import json
with open('training_hist.json','w') as f:
  json.dump(training_history.history,f)
```

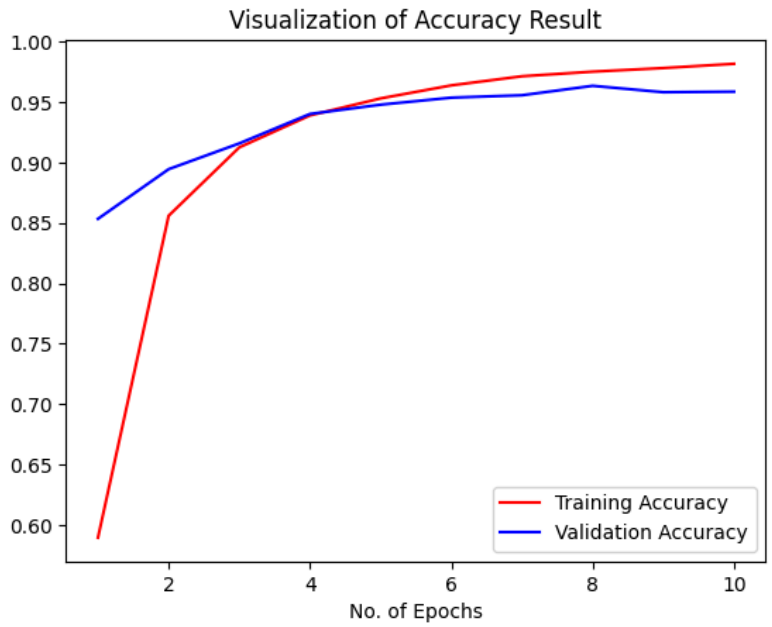
```
{'accuracy': [0.5896578431129456,
  0.8559641242027283,
  0.9125257730484009,
  0.9389572739601135,
  0.9530549645423889,
  0.9638381004333496,
  0.9713777899742126,
  0.9751760363578796,
  0.978163480758667,
  0.981677234172821],
 'loss': [1.3936800956726074,
  0.4562781751155853,
  0.2691739797592163,
  0.1854141354560852,
  0.1412469744682312,
  0.10765228420495987,
  0.08697693794965744,
  0.07674852013587952,
  0.06610886752605438,
  0.0570187047123909],
 'val_accuracy': [0.8534031510353088,
  0.8944912552833557,
  0.915831983089447,
  0.9401889443397522,
  0.9478716254234314,
  0.9536762833595276,
  0.9556680917739868,
  0.963464617729187,
  0.958172082901001,
  0.9586273431777954],
 'val_loss': [0.46121159195899963,
  0.3337039649486542,
  0.267711877822876,
  0.18735376000404358,
  0.1744774729013443,
  0.1560913324356079,
  0.14999143779277802,
  0.12226346135139465,
  0.13521425426006317,
  0.14562000334262848]}
```


The above values depicts training history.

ACCURACY VISUALIZATION:

With the help of training history we access the history for accuracy and validation and plot it using matplotlib library**.**

```
epochs = [i for i in range(1,11)]
plt.plot(epochs,training_history.history['accuracy'],color='red',label='Training Accuracy')
plt.plot(epochs,training_history.history['val_accuracy'],color='blue',label='Validation Accuracy')
plt.xlabel('No. of Epochs')
plt.title('Visualization of Accuracy Result')
plt.legend()
plt.show()
```



5.8 SAVING MODEL:

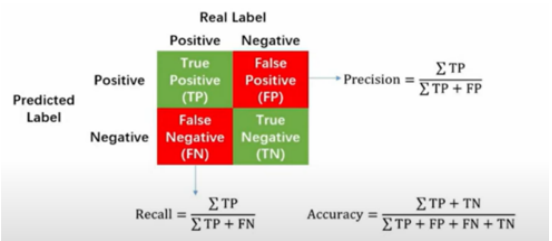
```
cnn.save('trained_plant_disease_model.keras')
```

Once We have trained a model and achieved satisfactory performance, we don't want to retrain it every time we need to make predictions or deploy it into production. Saving the model ensures that we can persist the model, its architecture, its weights, and other components, so we can reuse it later without having to retrain it from scratch.

Here for saving we use keras because it need only 30 mb of storage which makes it friendly to use without using so much storage**.**

5.9 SOME OTHER METRICS FOR MODEL EVALUATION:

Some other metrics like precision, recall, f1-score are to be find for evaluating.



$$\text{F1 Score} = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

```
class_name = validation_set.class_names
```

```
test_set = tf.keras.utils.image_dataset_from_directory(
    '/kaggle/input/new-plant-diseases-dataset/New Plant Diseases Dataset(Augmented)/New Plant Diseases Dataset(Augmented)/va
    labels="inferred",
    label_mode="categorical",
    class_names=None,
    color_mode="rgb",
    batch_size=1,
    image_size=(128, 128),
    shuffle=False,
    seed=None,
    validation_split=None,
    subset=None,
    interpolation="bilinear",
    follow_links=False,
    crop_to_aspect_ratio=False
)
```

```
Found 17572 files belonging to 38 classes.
```

```
y_pred = cnn.predict(test_set)
predicted_categories = tf.argmax(y_pred, axis=1)
```

```
17572/17572 ━━━━━━━━━━━━━━━━━━━━ 34s 2ms/step
```

```
true_categories = tf.concat([y for x, y in test_set], axis=0)
Y_true = tf.argmax(true_categories, axis=1)
```

```
Y_true
```

```
<tf.Tensor: shape=(17572,), dtype=int64, numpy=array([ 0,  0,  0, ..., 37, 37, 37])>
```

```
predicted_categories
```

```
<tf.Tensor: shape=(17572,), dtype=int64, numpy=array([ 0,  0,  0, ..., 37, 37, 37])>
```

Here we are using validation set for testing because in testing directory the class_names are all shuffled up and not in the proper directory which hampers our evaluation.

We predicted on test_set and find out all 38 possible probability for all 17572 images then with argmax columned up all the highest probability for all images .and hence we get predicted_categories.

Then access true_categories which are basically the y from test_Set it also lead to all 38 possible probability for all 17572 images then with argmax columned up all the highest probability for all images and hence we get y_true.

Then with the help of predicted_categories and y_true we will visualize our performance with the help of confusion matrix and classification report.

5.10 METRICS VISUALIZATION:

CLASSIFICATION_REPORT: The classification_report function from the sklearn.metrics module generates a detailed report with precision, recall, F1 score, and support for each class.

```
print(classification_report(Y_true,predicted_categories,target_names=class_name))
```

Classification report result:

	precision	recall	f1-score	support
Apple___Apple_scab	0.94	0.97	0.96	504
Apple___Black_rot	0.91	1.00	0.95	497
Apple___Cedar_apple_rust	0.94	0.97	0.96	440
Apple___healthy	0.99	0.91	0.95	502
Blueberry___healthy	0.99	0.96	0.97	454
Cherry_(including_sour)___Powdery_mildew	0.99	0.97	0.98	421
Cherry_(including_sour)___healthy	0.94	1.00	0.97	456
Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot	0.96	0.91	0.93	410
Corn_(maize)___Common_rust_	1.00	0.98	0.99	477
Corn_(maize)___Northern_Leaf_Blight	0.95	0.96	0.96	477
Corn_(maize)___healthy	0.99	1.00	1.00	465
Grape___Black_rot	0.97	0.98	0.98	472
Grape___Esca_(Black_Measles)	1.00	0.97	0.99	480
Grape___Leaf_blight_(Isariopsis_Leaf_Spot)	0.99	1.00	0.99	430
Grape___healthy	1.00	1.00	1.00	423
Orange___Haunglongbing_(Citrus_greening)	0.96	1.00	0.98	503
Peach___Bacterial_spot	0.97	0.97	0.97	459
Peach___healthy	0.99	0.98	0.98	432
Pepper,_bell___Bacterial_spot	0.98	0.76	0.85	478
Pepper,_bell___healthy	0.89	0.97	0.93	497
Potato___Early_blight	0.99	0.98	0.98	485
Potato___Late_blight	0.97	0.95	0.96	485
Potato___healthy	0.98	0.95	0.97	456
Raspberry___healthy	1.00	0.96	0.98	445
Soybean___healthy	0.97	0.95	0.96	505
Squash___Powdery_mildew	0.97	0.99	0.98	434
Strawberry___Leaf_scorch	1.00	0.95	0.97	444
Strawberry___healthy	0.98	0.98	0.98	456
Tomato___Bacterial_spot	0.99	0.93	0.96	425
Tomato___Early_blight	0.92	0.92	0.92	480
Tomato___Late_blight	0.88	0.93	0.91	463
Tomato___Leaf_Mold	0.90	0.99	0.94	470
Tomato___Septoria_leaf_spot	0.83	0.93	0.88	436
Tomato___Spider_mites Two-spotted_spider_mite	0.95	0.94	0.95	435
Tomato___Target_Spot	0.96	0.86	0.91	457
Tomato___Tomato_Yellow_Leaf_Curl_Virus	0.91	1.00	0.95	490
Tomato___Tomato_mosaic_virus	0.96	1.00	0.98	448
Tomato___healthy	0.99	0.97	0.98	481
accuracy			0.96	17572
macro avg	0.96	0.96	0.96	17572
weighted avg	0.96	0.96	0.96	17572

CONFUSION MATRIX REPORT: A confusion matrix is a fundamental tool used in classification tasks to evaluate the performance of a classification model. It provides a summary of the prediction results compared to the actual (true) values with the help of seaborn heatmap we visualize our confusion matrix.

```
from sklearn.metrics import confusion_matrix,classification_report
cm = confusion_matrix(Y_true,predicted_categories)
```

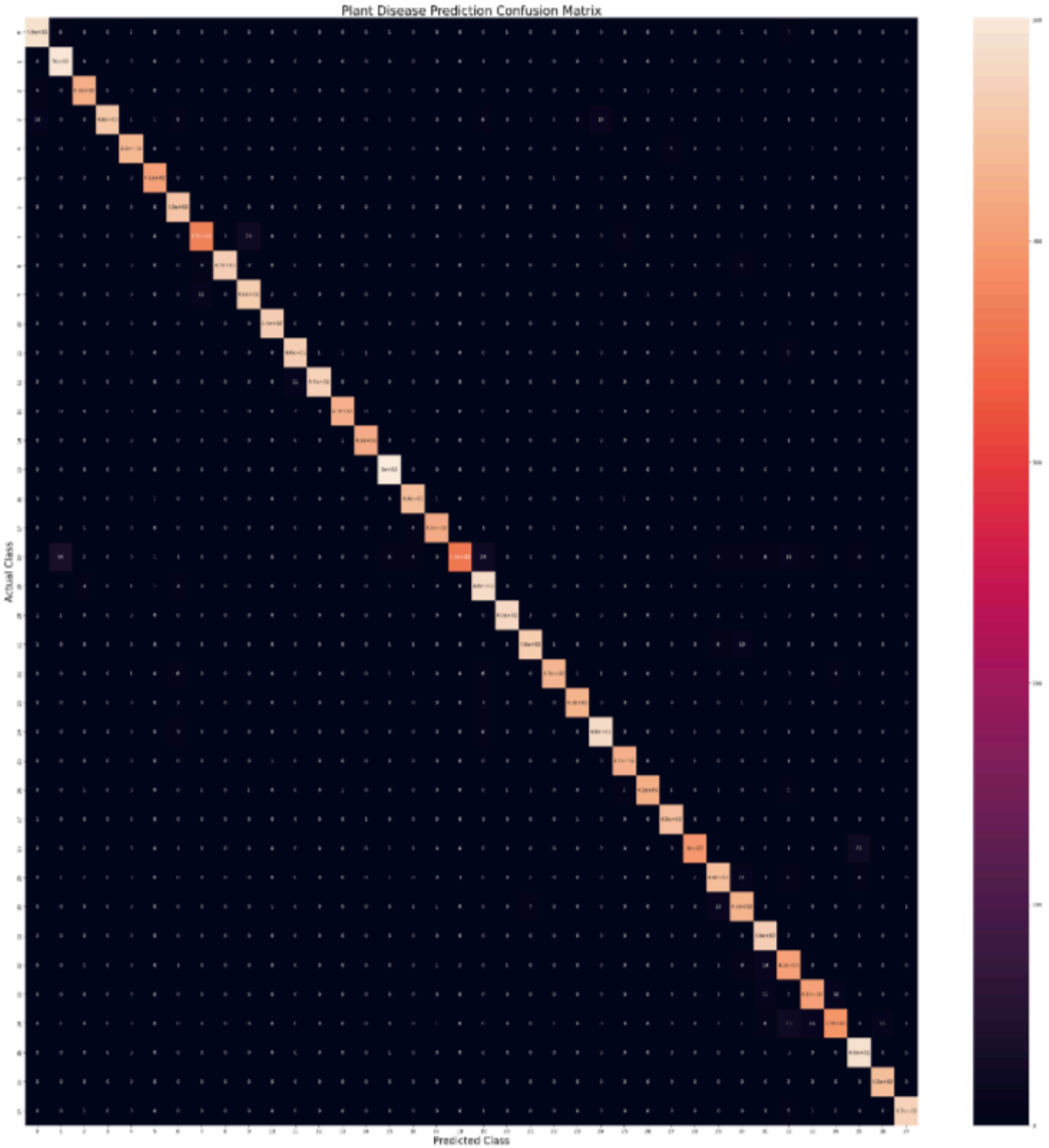
```
plt.figure(figsize=(40, 40))
sns.heatmap(cm,annot=True,annot_kws={"size": 10})

plt.xlabel('Predicted Class',fontsize = 20)
plt.ylabel('Actual Class',fontsize = 20)
plt.title('Plant Disease Prediction Confusion Matrix',fontsize = 25)
plt.show()
```

Annot=True: it means each class value will show in each cell

Annot_kws={size=10}: Basically used for the size of each cell value.

And, in matrix if our diagonal element is bright than our prediction is good



6.PROCEDURE INVOLVES IN TESTING:

6.1 TEST SET IMAGE PREPROCESSING:

```
validation_set = tf.keras.utils.image_dataset_from_directory(
    'valid',
    labels="inferred",
    label_mode="categorical",
    class_names=None,
    color_mode="rgb",
    batch_size=32,
    image_size=(128, 128),
    shuffle=True,
    seed=None,
    validation_split=None,
    subset=None,
    interpolation="bilinear",
    follow_links=False,
    crop_to_aspect_ratio=False
)
class_name = validation_set.class_names
print(class_name)
```

For the test set preprocessing we are using validation set because in valid folder the images are in proper directory and validation set is only used during training so just fetching out all the directories and keep it in class_name so that while prediction the index should be according to class_name.

```
Found 17572 files belonging to 38 classes.
['Apple___Apple_scab', 'Apple___Black_rot', 'Apple___Cedar_apple_rust', 'Apple___healthy', 'Blueberry___healthy', 'Cherry_(in
cluding_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_gree
ning)', 'Peach___Bacterial_spot', 'Peach___healthy', 'Pepper,_bell___Bacterial_spot', 'Pepper,_bell___healthy', 'Potato___Ear
ly_blight', 'Potato___Late_blight', 'Potato___healthy', 'Raspberry___healthy', 'Soybean___healthy', 'Squash___Powdery_milde
w', 'Strawberry___Leaf_scorch', 'Strawberry___healthy', 'Tomato___Bacterial_spot', 'Tomato___Early_blight', 'Tomato___Late_bl
ight', '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']
```

The above figure illustrates all 38 classes of crop in array format and the label can be easily accessed with its index.

6.2 Loading Model:

Loads a model saved via model.save(),so that we can access all the content that had been saved.

```
cnn = tf.keras.models.load_model('trained_plant_disease_model.keras')
```

6.3 VISUALISING AND PERFORMING PREDICTION ON SINGLE IMAGE:

```
import cv2
image_path = '/kaggle/input/new-plant-diseases-dataset/test/test/CornCommonRust2.JPG'
# Reading an image in default mode
img = cv2.imread(image_path)
img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB) #Converting BGR to RGB
# Displaying the image
plt.imshow(img)
plt.title('Test Image')
plt.xticks([])
plt.yticks([])
plt.show()
```

CV2 is the OpenCV module in Python that provides powerful tools for image and video processing.

IMAGE_PATH is the path of image taken from test folder.

IMREAD is the class of open cv that reads image in BGR format by default which hampers the effect of image.so to handle this variations we need to convert BGR to RGB format.

CVTCOLOR is also the class of open cv that convert the effect of image from BGR to RGB.

6.4 TESTING MODEL:

```
image = tf.keras.preprocessing.image.load_img(image_path,target_size=(128,128))
input_arr = tf.keras.preprocessing.image.img_to_array(image)
input_arr = np.array([input_arr])  # Convert single image to a batch.
predictions = cnn.predict(input_arr)
```

The image to be tested should be in same format like we have trained our model.

LOAD_IMG loads the image with the help of its path.

The target size of image should be same as trained image i.e 128,128

IMG_TO_ARRAY convert the image into array format because training is also done in array format image.

During training we are processing image in batch so here also we need to convert in batch and then only the shape of image will be (1,128,128,3). otherwise it will be like(128,128,3) where the batch format is excluded.

And then with the help of our model we are predicting the output.

PRINT(PREDICTIONS):it helps to print the probability of each class in array format and in total we will get 38 predictions because we have 38 classes. And the prediction whose value will be larger will be the actual index.

```
[[1.9345169e-32 8.9753608e-33 2.2701035e-43 4.2038954e-45 0.0000000e+00
  4.4863576e-38 6.1504972e-36 1.1606238e-35 1.0000000e+00 4.8138409e-37
  1.6030854e-42 5.6051939e-45 0.0000000e+00 4.2038954e-45 0.0000000e+00
  9.8375356e-41 6.5608794e-42 1.2191297e-43 2.0262035e-33 8.3906411e-38
  1.2973485e-30 0.0000000e+00 1.1842827e-35 0.0000000e+00 8.9683102e-44
  0.0000000e+00 1.2013028e-38 0.0000000e+00 0.0000000e+00 4.2446865e-36
  3.2521737e-33 0.0000000e+00 7.8192454e-42 0.0000000e+00 0.0000000e+00
  0.0000000e+00 0.0000000e+00 0.0000000e+00]]
```

As in this array our highest value lies in index 8 which is 1.000000e+00 and rest are small**.**

The other probabilities are in negative exponential format which can be expressed as 1.9345169e-32 represents the number 1.9345169 X 10−32 which is 0.00000000000000000000000000000019345169 which is extremely close to zero.

```
result_index = np.argmax(predictions) #Return index of max element
print(result_index)
```

```
8
```

With the help of argmax we find the highest probability value and stored in index form

6.5 DISPLAYING THE DISEASE PREDICTION:

```
# Displaying the disease prediction
model_prediction = class_name[result_index]
plt.imshow(img)
plt.title(f"Disease Name: {model_prediction}")
plt.xticks([])
plt.yticks([])
plt.show()
```

And Hence, the predictions comes out to be totally correct and by overcoming various challenges such as overshooting problem(loss function gradually increasing), training bottleneck(problem of underutilization of CPU),model bloat(where model utilize too much storage) and colour distortion(where image is affected by BGR colour) our model completely justified.

7.CONCLUSION

The study successfully demonstrates the potential of **Convolutional Neural Networks (CNNs)** for **crop disease detection** through **image classification**, achieving promising results with a deep learning-based approach. By utilizing CNNs, the model effectively identifies crop diseases from crop images, providing a reliable and automated solution for farmers. The accuracy rates achieved in this study, including metrics such as **precision**, **recall**, and **F1 score**, indicate a significant improvement over traditional plant disease detection methods, which often rely on manual inspection or simpler algorithms with lower diagnostic accuracy.

This research contributes to the growing field of **precision agriculture**, where artificial intelligence and machine learning are becoming central to improving agricultural productivity and sustainability. By integrating AI-based tools like the proposed **CNN model**, farmers can make data-driven decisions that improve crop yields, reduce the use of chemicals, and minimize environmental impact. With the agricultural sector facing increasing challenges due to climate change, pests, and limited resources, AI technologies hold the potential to revolutionize farming practices. The successful deployment of this model could be part of a larger **precision farming system**, where multiple sensors, drones, and satellite data integrate to provide a comprehensive solution for monitoring crop health.

As agriculture continues to evolve, tools like the proposed CNN model will be crucial in addressing the challenges of **food security** in an increasingly complex global landscape. The integration of **real-time disease prediction systems**, powered by AI and computer vision, will not only help reduce crop losses but also enable sustainable farming by optimizing resource usage and minimizing waste

In conclusion, this study underscores the importance of **AI technologies** in fostering **sustainable farming practices** and tackling the growing concerns of food security. As the agricultural sector continues to face pressures from climate change, population growth, and resource limitations, AI-powered tools like crop disease detection models will be indispensable in shaping the future of agriculture, enabling farmers to make informed decisions that support both environmental health and economic stability.

8.REFERENCES

1. S. D. Khirade and A. B. Patil, "Crop Disease Detection Using Image Processing," 2015 International Conference on Computing Communication Control and Automation, 2015, pp. 768-771, doi: 10.1109/ICCUBEA.2015.153.

2. S. C. Madiwalar and M. V. Wyawahare, "crop disease identification: A comparative study," 2017 International Conference on Data Management, Analytics and Innovation (ICDMAI), 2017, pp. 13-18, doi: 10.1109/ICDMAI.2017.8073478.

3. P. Moghadam, D. Ward, E. Goan, S. Jayawardena, P. Sikka and E. Hernandez, "crop Disease Detection Using Hyperspectral Imaging," 2017 International Conference on Digital Image Computing: Techniques and Applications (DICTA), 2017, pp. 1-8, doi: 10.1109/DICTA.2017.8227476.

4. S. D.M., Akhilesh, S. A. Kumar, R. M.G. and P. C., "Image based crop Disease Detection in Pomegranate Plant for Bacterial Blight," 2019 International Conference on Communication and Signal Processing (ICCSP), 2019, pp. 0645-0649, doi: 10.1109/ICCSP.2019.8698007.

5. G. Shrestha, Deepsikha, M. Das and N. Dey, "crop Disease Detection Using CNN," 2020 IEEE Applied Signal Processing Conference (ASPCON), 2020, pp. 109-113, doi: 10.1109/ASPCON49795.2020.9276722.

6. Mohanty SP, Hughes DP and Salathé M (2016) Using Deep Learning for Image-Based Plant

Disease Detection. Front. Plant Sci. 7:1419. doi: 10.3389/fpls.2016.01419

7. R. M. Haralick, K. Shanmugam and I. Dinstein, "Textural Features for Image Classification," in IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-3, no. 6, pp. 610-621, Nov. 1973, doi: 10.1109/TSMC.1973.4309314.

8. Breiman, L. Random Forests. Machine Learning 45, 5–32 (2001). https://doi.org/10.1023/A:1010933404324