AI BASED LEAF DISEASE IDENTIFICATION AND CLASSIFICATION SYSTEM USING DEEP LEARNING

A PROJECT REPORT

Submitted by

ARUN V	**(952420104001)**
GANESH PANDIYAN M	**(952420104006)**
KALEESWARAN V	**(952420104012)**
RAMKUMAR P	**(952420104023)**

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

PSN INSTITUTE OF TECHNOLOGY AND SCIENCE

TIRUNELVELI- 627 152

ANNA UNIVERSITY : CHENNAI 600 025

MAY 2024

ANNA UNIVERSITY : CHENNAI

BONAFIDE CERTIFICATE

Certified that this project report **"AI BASED LEAF DISEASE IDENTIFICATION AND CLASSIFICATION SYSTEM USING DEEP LEARNING"** is the bonafide work of **"ARUN V (952420104001), GANESH PANDIYAN M (952420104006), KALEESWARAN V (952420104012), RAMKUMAR P (952420104023)"** who carried out the project work under my supervision.

HEAD OF THE DEPARTMENT	**SUPERVISOR**
Mr. K. BALA KARTHIK M.Tech., (Ph.D)	**Mr. V. SOLAI RAJA M.E.,**
Head of the Department	Assistant Professor
Department of CSE	Department of CSE
9524 – PSN ITS	9524 – PSN ITS

Submitted for the Anna University Project Viva Voce Examination held on ……………………

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ABSTRACT

Tomato cultivation is a vital component of global agriculture, contributing significantly to both food security and economic development. However, the occurrence of diseases in tomato plants poses a significant threat to crop yield and quality. Early detection and diagnosis of these diseases are crucial for effective disease management. In recent years, Artificial Intelligence (AI) techniques, particularly Convolutional Neural Networks (CNNs) such as DCNN, have shown promise in automating the detection and classification of plant diseases. This research explores the application of DCNN in the context of tomato leaf disease detection. The study involves the collection of a diverse dataset of tomato leaf images representing various diseases and healthy states. The dataset is pre-processed to enhance the model's ability to generalize across different conditions. DCNN, a type of deep neural network architecture known for its dense connectivity and feature reuse, is employed for the classification task. The model is trained on the prepared dataset, utilizing transfer learning techniques to leverage knowledge gained from pre-trained models on large image datasets. The training process involves optimizing model parameters through back propagation, enabling the network to learn discriminative features for distinguishing between different tomato leaf conditions. The effectiveness of the model is assessed through rigorous testing on an independent dataset, and performance metrics such as accuracy, precision, recall, and F1-score are computed. The results demonstrate the potential of DCNN in accurately identifying and classifying tomato leaf diseases. The AI model exhibits robust performance even in the presence of variations in lighting conditions, leaf orientations, and disease severity. Moreover, the ability to detect diseases at an early stage holds promise for timely intervention and disease management strategies.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER-I

INTRODUCTION

Plant diseases have become more prevalent in recent years due to globalization, trade, and climate change. These issues have reached pandemic proportions in several nations, which increased the likelihood of crop damage and, in turn, created a threat to people's access to adequate food and nutrition. Specialists should ensure to safeguard agricultural plants. Parasitic organisms such as bacteria, fungi, viruses, roundworms (nematodes), and other plants can cause illnesses. Environmental conditions such as winter frosts or summer dryness and lack or excess of nutrients in the soil can be the potential causes of plant diseases. Phytopathology is the scientific study of plant disease that focuses on ways to treat and avoid the conditions responsible for plant illnesses. To overcome plant illness, plants have to be diagnosed precisely. There are several methods available for diagnosing. Local plant clinics and agricultural groups have traditionally helped in disease detection. Still, the technique could be more effective as there are more possibilities of human errors, and it is difficult for humans to access plants across a wide area. Moreover, using software using machine learning techniques can improve the efficiency of classifying diseased plants.

Smartphones are being developed using different tools and technology. Modern plant illness classification approaches can be adopted using smartphones due to their incorporation of high computational techniques, high-resolution screens, and built-in accessories such as HD cameras, which will be more effective for plant disease detection. Plant diseases are effectively diagnosed using machine learning techniques as accurate diagnostic tests can be carried out, which benefits in preserving the resource. Farmers can upload field images recognized by smartphones, and distinct software can be used for analysing, diagnosing, and developing action plans.

Recently, image processing techniques with deep neural networks have been effectively used diversely and proven to be highly effective approaches in constantly monitoring the health of plants and identifying signs of diseases in their early stages . A neural network considers an image of a diseased plant as an input and processes the image to produce a crop disease pair. Creating a deep network in such a manner that the network topology, functions (nodes), and edge weights accurately map the input to output is challenging. While training deep neural networks, the network parameters are adjusted in such a manner that the mapping is enhanced better over the training period. This complex computational process has recently seen several conceptual and practical advancements that have dramatically increased its performance . One of these approaches is Vision Transformer, which takes the whole image and extracts its features by splitting the image into multi patches, and from each patch, the transformer encoder will extract the features. Extracting the features from the whole image may take much more time and extract unnecessary features.

Developed Technologies have provided the ability to produce sufficient food to meet the demand of society. But still, the safety and security of the food or crops remained unattained. Factors like change in climate, the decline in pollinators, Plant disease, and others are challenging to the farmers. An important foundation for these factors needs to be attained on a priority basis [1,2]. Making use of analysis and detection processes using present technology helps the farmers to get rid of such problems. During pandemic situations like COVID 19 the nation is dependent on the recent technologies to prevent address the issues to reduce the transmission of the diseases. As plant diseases are a significant threat to human life as they may lead to droughts and famines. In turn it results causing substantial losses, where farming is accompanying in commercial purpose. The use of technologies like Computer vision and Machine Learning (ML) helps to fight against diseases. In this project, we are using ML to give a solution to Plant Diseases. In this method, we have divided the process into three stages Identity, Analyse and Verify with the Available database .

The key issues and challenges are identified by the researchers and the scientists, while analysing the leaf diseases of plant. Some of them are as follows The quality of the leaf image must be high. Publicly available Dataset requirement. Noisy data affecting the leaf samples. Through the process of segmentation, diseases may be identified but the samples must undergo training and testing. Classification is one more challenge, in the stage of detecting the leaf diseases. Colour of the leaves may be varied due to environmental effect. Variety of diseases can be seen in various kinds of plants, so detection of disease is quite difficult. Based on the challenges discussed above and combined techniques using image processing (IP) and ML, the proposed model provide better accuracy. Keeping all these things in mind, in this paper an algorithm based on ML and IP tools to automatically detect leaf diseases is proposed.

CHAPTER-II

LITERATURE SURVEY

1. A. K. Pradhan, S. Swain, and J. K. Rout, ‘‘Role of machine learning and cloud-driven platform in IoT-based smart farming,’’2022.

Technological advances in agriculture are important for better growth and sustainability in today’s market, which is characterized by intense competition and risk. Crops, their production, and quality are all vital to a farmer’s livelihood. Agriculture is our country’s most basic and important occupation, as it balances food demand while also supplying vital raw materials for a variety of industries. The amalgamation of machine learning (ML) and Internet of Things (IoT) based advanced smart farming tools are turning the face of agricultural production day after day not only by improving it but also by rendering it cost-effective and reducing waste. The IoT produces vast amounts of data depending on position and time with various features. To increase agricultural productivity by intelligent farming systems, data must be thoroughly collected and interpreted. As the volume of data collected expands, higher processing capabilities in ML enable new possibilities for data-intensive scientific research, and these could be used to boost application intelligence and flexibility. This chapter looks at the current approaches to smart agriculture integrating ML and IoT in such applications using Cloud-driven platforms.

2. A. Bhargava and A. Bansal, ‘‘Fruits and vegetables quality evaluation using computer vision: A review,’’2021.

In agriculture science, automation increases the quality, economic growth and productivity of the country. The export market and quality evaluation are affected by assorting of fruits and vegetables. The crucial sensory characteristic of fruits and vegetables is appearance that impacts their market value, the consumer’s preference and choice. Although, the sorting and grading can be done by human but it is inconsistent, time consuming, variable, subjective, onerous, expensive and easily influenced by surrounding. Hence, an astute fruit grading system is needed. In recent years, various algorithms for sorting and grading are done by various researchers using computer vision. This paper presents a detailed overview of various methods i.e. pre-processing, segmentation, feature extraction, classification which addressed fruits and vegetables quality based on colour, texture, size, shape and defects. In this paper, a critical comparison of different algorithm proposed by researchers for quality inspection of fruits and vegetables has been carried out.

3. A. Ahmad, D. Saraswat, and A. El Gamal, ‘‘A survey on using deep learning techniques for plant disease diagnosis and recommendations for development of appropriate tools,’’ 2023.

Several factors associated with disease diagnosis in plants using deep learning techniques must be considered to develop a robust system for accurate disease management. A considerable number of studies have investigated the potential of deep learning techniques for precision agriculture in the last decade. However, despite the range of applications, several gaps within plant disease research are yet to be addressed to support disease management on farms. Thus, there is a need to establish a knowledge base of existing applications and identify the challenges and opportunities to help advance the development of tools that address farmers' needs. This study presents a comprehensive overview of 70 studies on deep learning applications and the trends associated with their use for disease diagnosis and management in agriculture. The studies were sourced from four indexing services, namely Scopus, IEEE Xplore, Science Direct, and Google Scholar, and 11 main keywords used were Plant Diseases, Precision Agriculture, Unmanned Aerial System (UAS), Imagery Datasets, Image Processing, Machine Learning, Deep Learning, Transfer Learning, Image Classification, Object Detection, and Semantic Segmentation.

4. R. Karthik, M. Hariharan, S. Anand, P. Mathikshara, A. Johnson, and R. Menaka, ‘‘Attention embedded residual CNN for disease detection in tomato leaves,’’ 2020.

Automation in plant disease detection and diagnosis is one of the challenging research areas that has gained significant attention in the agricultural sector. Traditional disease detection methods rely on extracting handcrafted features from the acquired images to identify the type of infection. Also, the performance of these works solely depends on the nature of the handcrafted features selected. This can be addressed by learning the features automatically with the help of Convolutional Neural Networks (CNN). This research presents two different deep architectures for detecting the type of infection in tomato leaves. The first architecture applies residual learning to learn significant features for classification. The second architecture applies attention mechanism on top of the residual deep network. Experiments were conducted using Plant Village Dataset comprising of three diseases namely early blight, late blight, and leaf mold. The proposed work exploited the features learned by the CNN at various processing hierarchy using the attention mechanism and achieved an overall accuracy of 98% on the validation sets in the 5-fold cross-validation.

5. O. O. Abayomi-Alli, R. Damasevicius, S. Misra, and R. Maskeliunas, ‘‘Cassava disease recognition from low-quality images using enhanced data augmentation model and deep learning,’’ 2021.

Improvement of deep learning algorithms in smart agriculture is important to support the early detection of plant diseases, thereby improving crop yields. Data acquisition for machine learning applications is an expensive task due to the requirements of expert knowledge and professional equipment. The usability of any application in a real-world setting is often limited by unskilled users and the limitations of devices used for acquiring images for classification. We aim to improve the accuracy of deep learning models on low-quality test images using data augmentation techniques for neural network training. We generate synthetic images with a modified colour value distribution to expand the trainable image colour space and to train the neural network to recognize important colour-based features, which are less sensitive to the deficiencies of low-quality images such as those affected by blurring or motion. This paper introduces a novel image colour histogram transformation technique for generating synthetic images for data augmentation in image classification tasks. The approach is based on the convolution of the Chebyshev orthogonal functions with the probability distribution functions of image colour histograms. To validate our proposed model, we used four methods (resolution down-sampling, Gaussian blurring, motion blur, and overexposure) for reducing image quality from the Cassava leaf disease dataset. The results based on the modified MobileNetV2 neural network showed a statistically significant improvement of cassava leaf disease recognition accuracy on lower-quality testing images when compared with the baseline network. The model can be easily deployed for recognizing and detecting cassava leaf diseases in lower quality images, which is a major factor in practical data acquisition.

6. H.-T. Thai, N.-Y. Tran-Van, and K.-H. Le, ‘‘Artificial cognition for early leaf disease detection using vision transformers,’’ 2021.

There are many kinds of cassava leaf diseases firmly harm cassava yield, including four main types as followings: Cassava Bacterial Blight (CBB), Cassava Brown Streak Disease (CBSD), Cassava Green Mottle (CGM), and Cassava Mosaic Disease (CMD). In a traditional way, leaf diseases were diagnosed intuitively by farmers. This process is inefficient and unreliable. Several studies have recently relied on deep neural networks for identifying leaf diseases. In this research, we exploit the novel model named Vision Transformer (ViT) in place of a convolution neural network (CNN) for classifying cassava leaf diseases. Experimental results show that this model can obtain competitive accuracy at least 1% higher than popular CNN models (Efficient Net, Resnet50d) on Cassava Leaf Disease Dataset. These results also indicate the potential superiority of the ViT over established methods in analysing leaf diseases. Next, we quantize the original model and successfully deploy it onto the Edge device named Raspberry Pi 4, which can be attached to a drone that allows farmers to automatically and efficiently detect infected leaves. This result has a significant capability for many future applications in smart agriculture.

7. A. Khamparia, G. Saini, D. Gupta, A. Khanna, S. Tiwari, and V. H. C. de Albuquerque, ''Seasonal crops disease prediction and classification using deep convolutional encoder network,'' 2020.

Agriculture plays a significant role in the growth and development of any nation's economy. But, the emergence of several crop-related diseases affects the productivity in the agriculture sector. To cope up this issue and to make aware the farmers to prevent the expansion of diseases in crops and to implement effective management, crop disease diagnosis plays its significant role. Researchers had already used many techniques for this purpose, but some vision-related techniques are yet to be explored. Commonly used techniques are support vector machine, *k*-means clustering, radial basis functions, genetic algorithm, image processing techniques like filtering and segmentation, deep structured learning techniques like convolutional neural network. We have designed a hybrid approach for detection of crop leaf diseases using the combination of convolutional neural networks and autoencoders. This research paper provides a novel technique to detect crop diseases with the help of convolutional encoder networks using crop leaf images. We have obtained our result over a 900-image dataset, out of which 600 constitute the training set and 300 test set. We have considered 3 crops and 5 kinds of crop diseases. The proposed network was trained in such a way that it can distinguish the crop disease using the leaf images.

8. A. J. Rozaqi and A. Sunyoto, ‘‘Identification of disease in potato leaves using convolutional neural network (CNN) algorithm,’’ 2020.

Potato crops have many benefits for human life, one of the most useful benefits of potatoes for humans is the carbohydrate content in them and carbohydrates are the main food for humans. The development of potato crop agriculture is very important for the sustainability of human life. There are several obstacles in developing potato farming, including a disease that attacks potato leaves which if left untreated will result in poor production or even crop failure in the future. One of the obstacles in the development of potato plants is the disease on potato leaves, namely early blight caused by the fungus Alternia solani, then late bligt disease caused by Microbe phytopthora infestans de bary. This disease has its respective symptoms so that farmers can take precautions if they see symptoms on potato leaves, but in this preventive step can only be done by experts who have knowledge in the field of diseases in potato plants while the average farmer does not have sufficient knowledge. So, the identification process becomes less accurate and takes a long time. Technology in the field of informatics in the form of digital image processing can be used to solve problems in disease identification in potato leaves, so this research will propose the right method for detecting disease in potato leaves. The identification process in this study uses three types of data in the form of healthy leaves, early blight, and late blight. The method used to identify is deep learning using the Convolutional Neural Network (CNN) architecture. The result of this research is that the 70:30 data division produces better accuracy than the 80:20 data division. The accuracy obtained is 97% on training data and 92% on validation data using 20 batch sizes at 10 epochs.

9. A. Singh and H. Kaur, ‘‘Potato plant leaves disease detection and classification using machine learning methodologies,’’ 2021.

Agriculture is one of the essential sectors for the survival of humankind. At the same time, digitalization touching across all the fields that became easier to handle various difficult tasks. Adapting technology as well as digitalization is very crucial for the field of agriculture to benefit the farmer as well as the consumer. Due to adopting technology and regular monitoring, one can able to identify the diseases at the very initial stages and those can be eradicated to obtain a better yield of the crop. In this document, a methodology was proposed for the detection as well as the classification of diseases that occur for the potato plants. For this scenario, the openly accessible, standard, and reliable data set was considered which was popularly known as Plant Village Dataset. For the process of image segmentation, the K-means methodology was considered, for the feature extraction purpose, the grey level co-occurrence matrix concept was utilized, and for the classification purpose, the multi-class support vector machine methodology was utilized. The proposed methodology able to attain an accuracy of 95.99%.

10. J. Johnson, G. Sharma, S. Srinivasan, S. K. Masakapalli, S. Sharma, J. Sharma, and V. K. Dua, ‘‘Enhanced field-based detection of potato blight in complex backgrounds using deep learning,’’2021.

Rapid and automated identification of blight disease in potato will help farmers to apply timely remedies to protect their produce. Manual detection of blight disease can be cumbersome and may require trained experts. To overcome these issues, we present an automated system using the Mask Region-based convolutional neural network (Mask R-CNN) architecture, with residual network as the backbone network for detecting blight disease patches on potato leaves in field conditions. The approach uses transfer learning, which can generate good results even with small datasets. The model was trained on a dataset of 1423 images of potato leaves obtained from fields in different geographical locations and at different times of the day. The images were manually annotated to create over 6200 labelled patches covering diseased and healthy portions of the leaf. The Mask R-CNN model was able to correctly differentiate between the diseased patch on the potato leaf and the similar-looking background soil patches, which can confound the outcome of binary classification. To improve the detection performance, the original RGB dataset was then converted to HSL, HSV, LAB, XYZ, and YCrCb colour spaces. A separate model was created for each colour space and tested on 417 field-based test images. This yielded 81.4% mean average precision on the LAB model and 56.9% mean average recall on the HSL model, slightly outperforming the original RGB colour space model. Manual analysis of the detection performance indicates an overall precision of 98% on leaf images in a field environment containing complex backgrounds.

CHAPTER-III

SYSTEM IMPLEMENTATION

3.1 Existing System

- In existing system Using a pre-trained model to reduce the dimensionality of the image, making the next stage less complex.
- Extract the deep feature of the leaf using a combination of the pre-trained model and ViT making the classifier more accurate.
- Comparisons between the TLMViT and five other transfer learning-based models are made.

3.2 Proposed system

The proposed system leaf disease using the DCNN .

- Collect a comprehensive dataset of leaf images, including various types of diseases and healthy leaves.
- Choose a pre-trained DCNN model from deep learning libraries such as TensorFlow (with Keras) or PyTorch. For example, you can use DCNN.
- Train the model on the training dataset. Use an appropriate loss function (e.g., categorical cross-entropy for multi-class classification) and an optimizer (e.g., Adam).
- Monitor and record training metrics like accuracy, loss, and validation accuracy.

3.3 Block Diagram

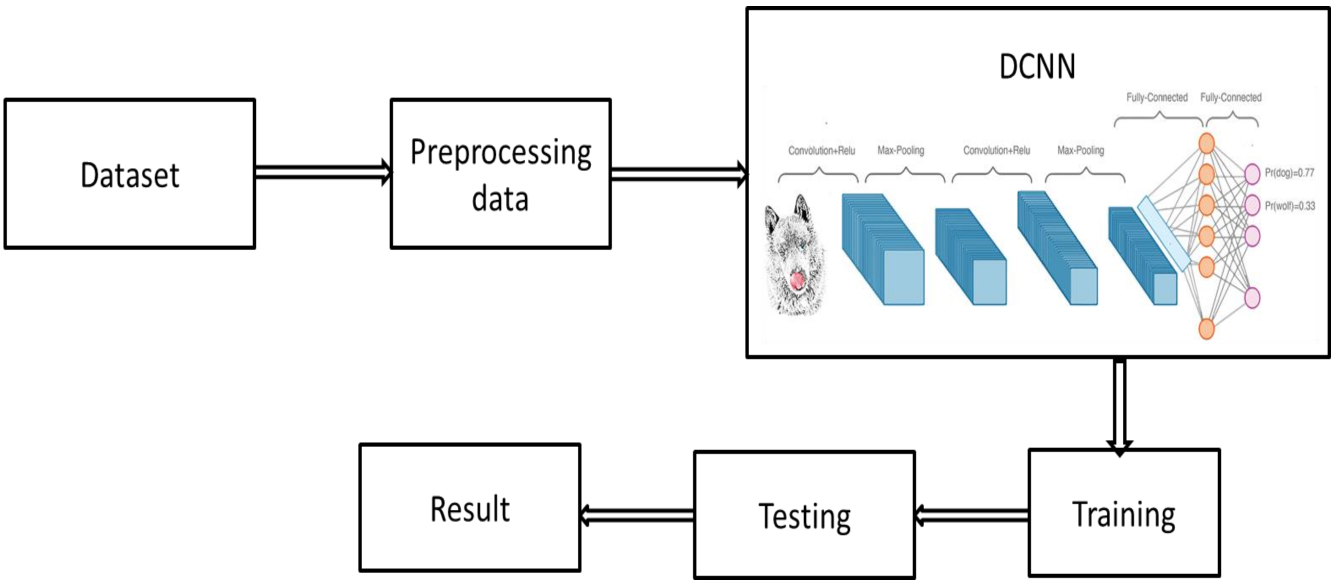


Fig 3.1: Block Diagram for DCNN

Modules

- Data collection
- Preprocessing
- Training
- Testing

Data collection

- The first step is to Collecting data on plant leave diseases is essential for research, disease management, and developing solutions to protect crops and plants. we are collecting the tomato leaves.

Pre-processing

- The collected data must be pre-processed to remove any errors or inconsistencies. This may involve removing missing data.

DEEP LEARNING

Deep learning (also known as deep structured learning) is part of a broader family of machine learning methods based on artificial neural networks with representation learning. Learning can be supervised, semi-supervised or unsupervised.

Deep learning architectures such as deep neural networks, deep belief networks, recurrent neural networks and convolution neural networks have been applied to fields including computer vision, machine vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics, drug design, medical image analysis, material inspection and board game programs, where they have produced results comparable to and in some cases surpassing human expert performance.

The adjective "deep" in deep learning comes from the use of multiple layers in the network. Early work showed that a linear perceptron cannot be a universal classifier, and then that a network with a nonpolynomial activation function with one hidden layer of unbounded width can on the other hand so be. Deep learning is a modern variation which is concerned with an unbounded number of layers of bounded size, which permits practical application and optimized implementation, while retaining theoretical universality under mild conditions. In deep learning the layers are also permitted to be heterogeneous and to deviate widely from biologically informed connectionist models, for the sake of efficiency, trainability and understandability, whence the "structured" part.

CONVOLUTIONAL NEURAL NETWORK

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics. They have applications in image and video recognition, recommender systems, image classification, medical image analysis, natural language processing, and financial time series.

CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "fully-connectedness" of these networks makes them prone to overfitting data. Typical ways of regularization include adding some form of magnitude measurement of weights to the loss function. CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns. Therefore, on the scale of connectedness and complexity, CNNs are on the lower extreme.

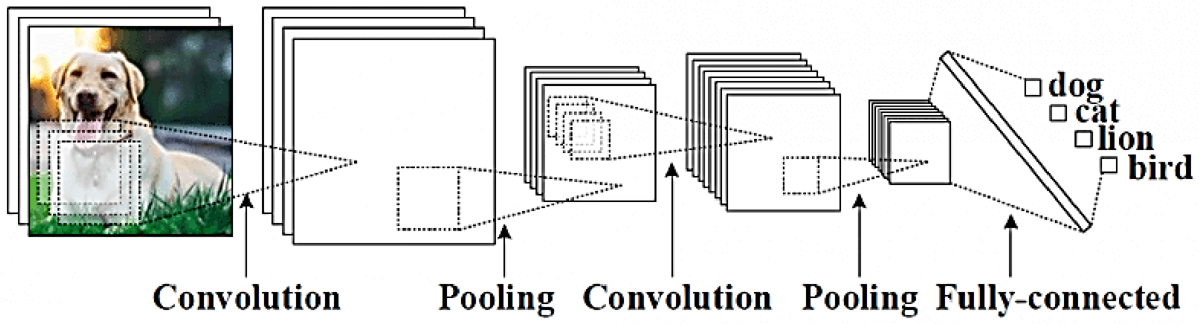


Fig 3.2: Architecture of Convolutional neural network

Steps in CNN

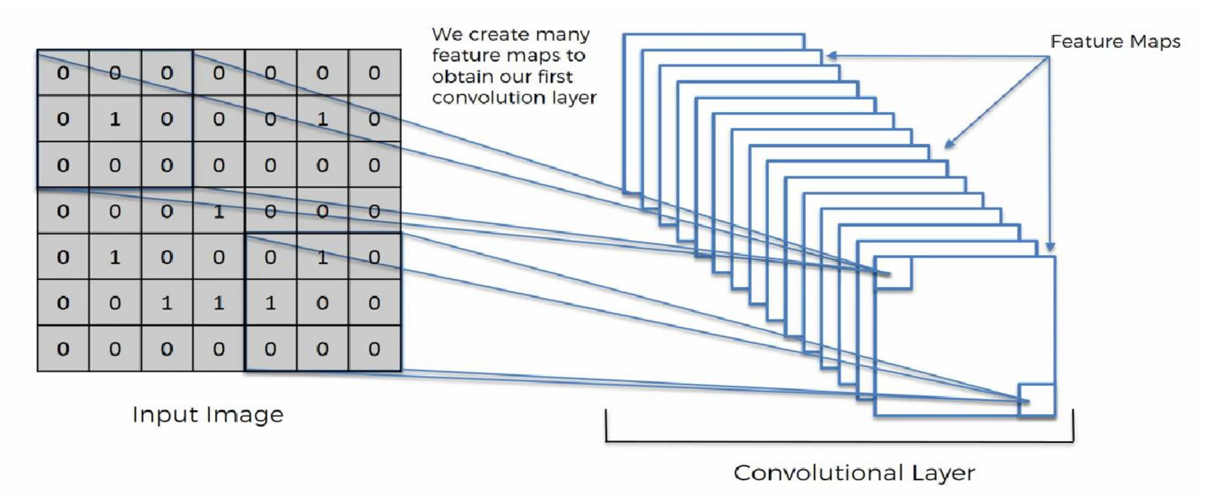
- **Step 1: Convolution**
- **Step 2: Max pooling**
- **Step 3: Flattening**
- **Step 4: Fully connection**

Convolution Layer

When programming a CNN, the input is a tensor with shape (number of images) x (image height) x (image width) x (image depth). Then after passing through a convolutional layer, the image becomes abstracted to a feature map, with shape (number of images) x (feature map height) x (feature map width) x (feature map channels). A convolutional layer within a neural network should have the following attributes:

- Convolutional kernels defined by a width and height (hyper-parameters).
- The number of input channels and output channels (hyper-parameter).
- The depth of the Convolution filter (the input channels) must be equal to the number channels (depth) of the input feature map.

Convolutional layers convolve the input and pass its result to the next layer. This is similar to the response of a neuron in the visual cortex to a specific stimulus. Each convolutional neuron processes data only for its receptive field. Although fully connected feedforward neural networks can be used to learn features as well as classify data, it is not practical to apply this architecture to images. A very high number of neurons would be necessary, even in a shallow (opposite of deep) architecture, due to the very large input sizes associated with images, where each pixel is a relevant variable. For instance, a fully connected layer for a (small) image of size 100 x 100 has 10,000 weights for each neuron in the second layer. The convolution operation brings a solution to this problem as it reduces the number of free parameters, allowing the network to be deeper with fewer parameters. For instance, regardless of image size, tiling regions of size 5 x 5, each with the same shared weights, requires only 25 learnable parameters. By using regularized weights over fewer parameters, the vanishing gradient and exploding gradient problems seen during backpropagation in traditional neural networks are avoided.


We create many feature maps to obtain our first convolution layer
Feature Maps
Input Image
Convolutional Layer

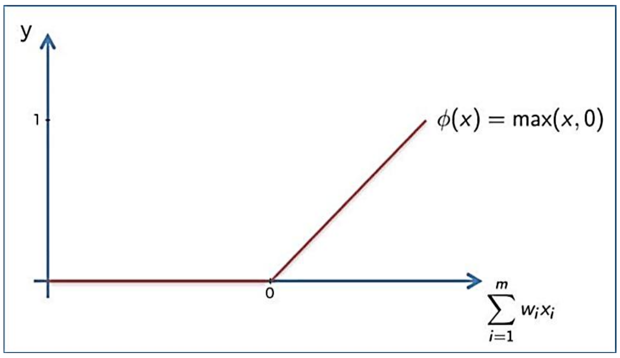


Fig 3.3: Appling ReLu Activation function to decrease the linearity in the image, because the image originally nonlinear

Pooling Layer

A **pooling** layer is another building block of a **CNN**. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network. **Pooling** layer operates on each feature map independently. The most common approach used in **pooling** is max **pooling**.

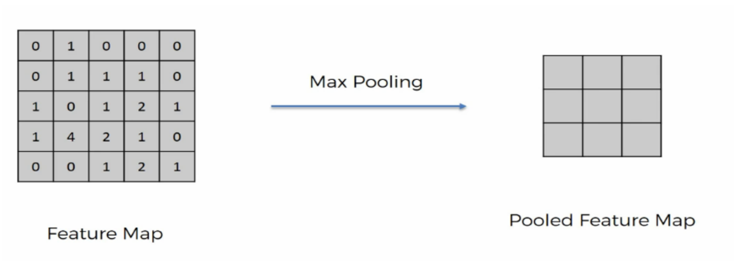


Fig 3.4: Pooling

Max / Avg. Pooling

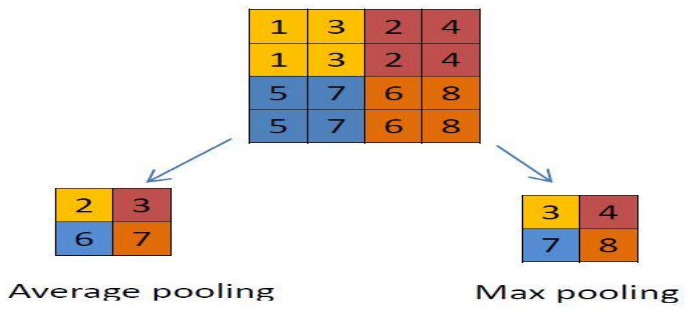
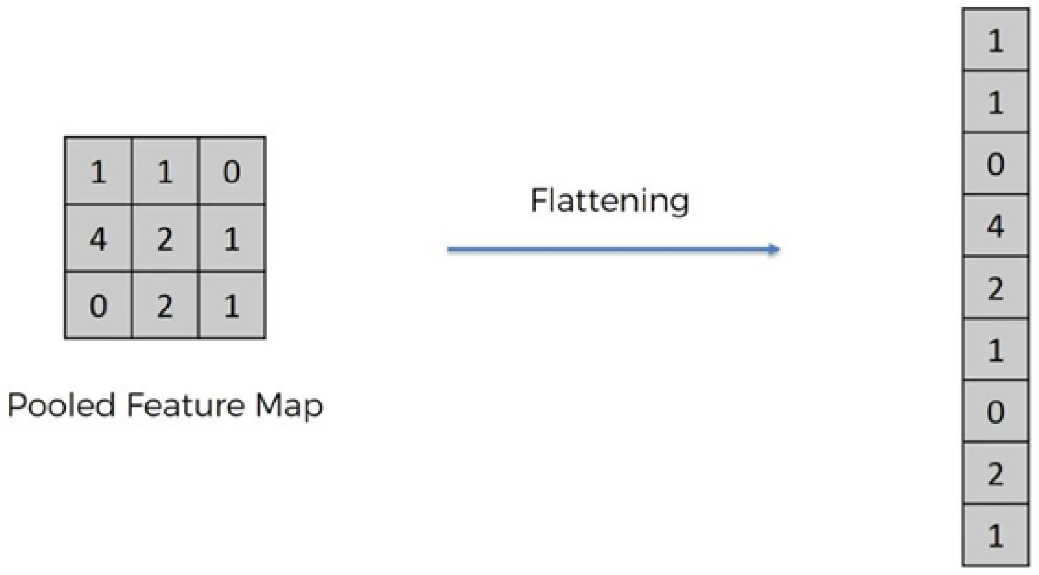


Fig 3.5: Max or Avg Pooling

Flattening

Flattening is converting the data into a 1-dimensional array for inputting it to the next layer. We **flatten** the output of the convolutional layers to create a single long feature vector. And it is connected to the final classification model, which is called a fully-connected layer



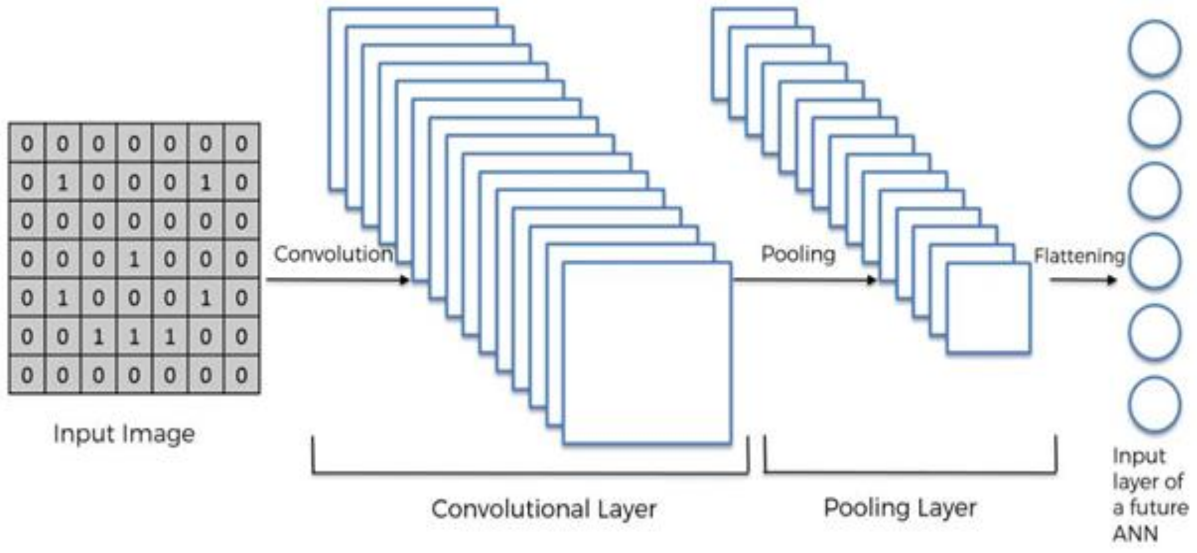


Fig 3.6: Flattening Layers

Fulling Connection

Fully connected layers connect every neuron in one layer to every neuron in another layer. It is in principle the same as the traditional multi-layer perceptronneural network (MLP). The flattened matrix goes through a fully connected layer to classify the images.

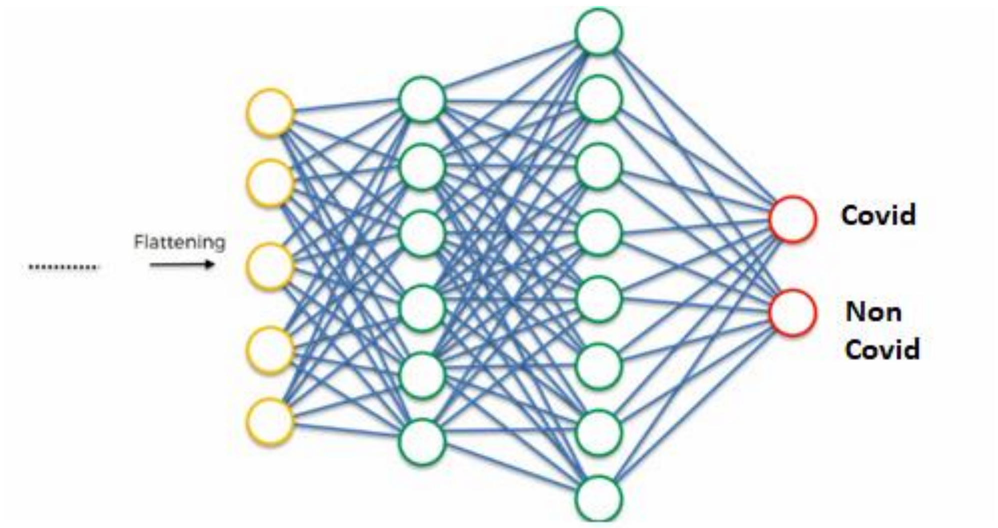


Fig 3.7: Fulling Connection

Machine Learning

Artificial Intelligence and Machine Learning Background

AI is a machine that displays cognitive behaviour similar to that of a human being. AI is an umbrella term that consists of multiple computer programs that display human capabilities. A computer program that has the ability to learn, self-improve, process information, and predict an output fall under the branch of AI .

AI can be divided into three domains namely; Robotics, Cognitive Systems, and Machine Learning.

1) Robotics: It deals with direct interaction with human beings in the physical world. Robotics is found to be useful in improving work in our daily life.

2) Cognitive Systems: It is based in the human world where humans and machines communicate and work together towards a common goal. An example is a communication interface called chatbot.

3) Machine Learning: It is based in the information world. Machines utilize data to learn and derive meaning in order to make predictions. Deep Learning is a subset of machine learning which deals with multi-layer neural networks.

ML is a branch of AI where a system has the capability to learn and improve based on experience without external programming. ML consists of algorithms or methods that are used to create learning models from data. The prime focus of ML is to enable automatic learning for computers without human assistance in order to make accurate decisions in the future.

Machine Learning Types

The three types of ML are supervised, unsupervised, and reinforcement learning.

1) **Supervised Learning**

The most commonly used type of learning is Supervised Learning. In SL, the model is trained on labelled data. The labelled data helps the model to learn the relationship between data points. The system is powerful enough to make predictions after plentiful training. This type of learning can compare the predicted output with the expected output to detect errors. These errors are then rectified by modifying the ML model accordingly.

A majority of ML models use Supervised ML techniques. SL helps to process and classify data with the help of machine language. There are two types of SL techniques namely; Regression and Classification.

Regression helps fit the data and produce a continuous value output based on the labelled input data. And Classification deals with the separation and grouping of data into classes. When the input data is labelled into two distinct classes, it is known as Binary Classification. Grouping data into more than two classes is called Multiple/ Multiclass Classification.

In the thesis, three classification based supervised learning algorithms: K-Nearest Neighbour (KNN), Random Forest (RF), and Logistic Regression (LR) are used to train the ML model.

2) **Unsupervised Learning**

In contrast to SL, Unsupervised Learning (UL) is when the training involved is unlabelled or not classified. This type of learning is advantageous than SL as it can process large datasets without spending time on data pre-processing. Due to the absence of labels in UL, hidden patterns are created from the unlabelled data. The creation of such patterns does not require external input from humans. This makes deployment and development of UL more versatile and functional than SL.

3) **Reinforcement Learning**

Reinforcement Learning (RL) is a type of learning that is inspired by how humans learn. This learning is based on a trial-and-error reward system as it learns continuously by interacting with new environments. The agent is rewarded for correct predictions and penalized for wrong answers. The model trains itself based on the reward points gained and this process is repeated when exposed to a new environment.

Steps in Machine Learning Background

The machine learning process can be explained in seven steps. Figure 1 depicts the seven steps in ML with data being the key factor for each step.

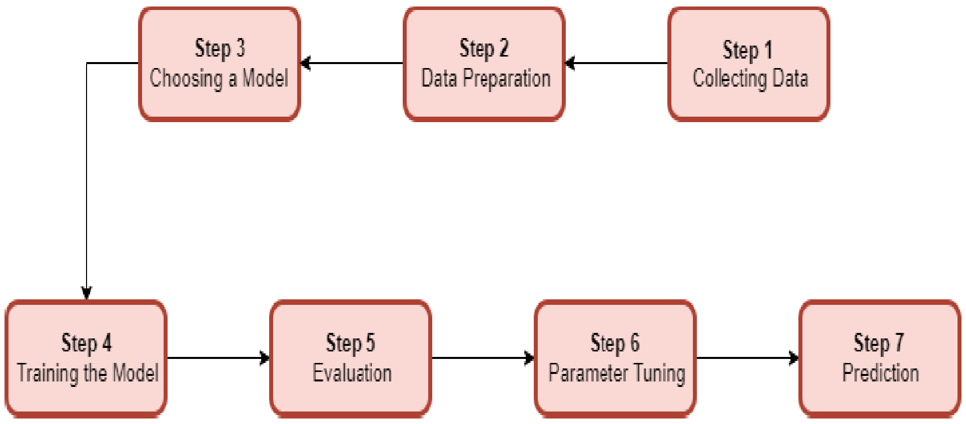


Fig 3.8: Machine Learning Steps

1) Collecting/ Gathering Data: The first step where relevant data is gathered for ML processing. The quality and quantity of data determines the accuracy of the predictive model.

2) Data Preparation: The raw collected from the previous step is not useful by itself and needs to be cleaned. This is done by removing duplicates, converting data types, and dealing with errors and missing values. Data is visualized to detect outliers, patterns, biases, and relationships between variables.

3) Choosing a Model: There are several models available for different purposes in the field of ML and this step deals with selecting the right model for our goal.

4) Training the Model: Training a model is one the most important steps in ML. It is to improve the model predictions by using the training data. The weights and biases are updated during each iteration cycle also known as the training step.

5) Evaluation: In this step, the trained model is tested on unseen data. The performance of the model is evaluated using performance metrics such as Accuracy, Recall, Precision, etc. A good train-test split is around 70/30 depending on the dataset.

6) Parameter Tuning: This step is otherwise known as "hyper parameter tuning". The parameters are tuned to improve the performance of the model. Simple hyper

parameters involve tuning the learning rate, increasing number of training steps, distribution, etc.

7) Prediction: The final step in ML where the model after undergoing the first six steps is ready to make predictions for practical applications.

Pre-processing in Machine Learning Background

Real-world data generally consists of missing values, noises, and redundancies or available in a format that is not compatible with the ML model. As the capability of a model to learn is directly dependent on the quality of data that is fed as input, data pre-processing becomes a crucial step. Data pre-processing is the process of converting raw data into a machine-understandable format.

A dataset consists of a collection of data objects which are also called as samples, vectors, records, events, or cases. These data objects are signified by the number of features. Features describe the characteristics of an object and are also known as attributes, variables, fields, or dimensions. Features of a dataset can be of two types: Categorical or Numerical. Categorical features are those which take on a fixed set of values (eg: Boolean: True or False). Numerical features are values that are continuous or integers.

There are various methods of data pre-processing steps available for ML. The five types of data processing methods are as follows:

● Data Quality Assessment: Data collected from various sources often contains irregularities in terms of format and quality of information. These irregularities can be due to human error, bugs during the data collection process, or measuring device limitations This becomes a mandatory step that is required when working on any kind of ML problem. There are three techniques to deal with such issues:

1. Missing Values: This technique helps estimate missing values and eliminate rows/columns with missing values. In the case of a small percentage of missing values, it is filled using interpolation methods or with the mean, mode, or median values. In the case of a feature consisting of an extremely large number of missing values, that feature itself can be removed.

2. Inconsistent Values: Data assessment is performed to correct value inconsistencies (eg: data type of feature).

3. Duplicate Values: Redundant values are a common occurrence in datasets and duplicates are removed to avoid bias during the processing of ML algorithms.

● Feature Aggregation: It is a method of pooling aggregated values to provide a better perspective for data analysis [30]. This helps provide a stable visual of aggregated values than scattered individual values. Feature Aggregation can help reduce the overall processing time and memory consumption.

● Feature Sampling: Sampling is a technique to select a subset of a dataset for analysis. Large datasets can put a dent in terms of cost, time, and memory constraints, and using a sample size of the dataset can be a better option. Sampling of the dataset is performed by preserving the properties of the original dataset. Feature Sampling can be disadvantageous in the case of an imbalanced dataset where the number of instances of one class is extremely higher than another class.

● Dimensionality Reduction: Datasets consisting of a large number of features can cause issues during the data analysis phase. It becomes extremely difficult for the model to visualize with the increase in the number of dimensions. Dimensionality Reduction helps convert a high dimensional dataset to a low dimensional one. This is performed using two types of techniques namely Principal Component Analysis (PCA) and Singular Value Decomposition.

● Feature Encoding: It is the process of transforming/encoding data to an easy and acceptable input for the ML algorithm without compromising the original meaning of the dataset.

CHAPTER-IV

SIMULATION RESULTS & DISCUSSION

SOFTWARE DESCRIPTION

4.1 PYTHON

The Python language had a humble beginning in the late 1980s when a Dutchman Guido Von Rossum started working on a fun project, which would be a successor to ABC language with better exception handling and capability to interface with OS Amoeba at Centrum Wiskunde and Informatica. It first appeared in 1991. Python 2.0 was released in the year 2000 and Python 3.0 was released in the year 2008. The language was named Python after the famous British television comedy show Monty Python's Flying Circus, which was one of Guido's favourite television programmes. Here we will see why Python has suddenly influenced our lives and the various applications that use Python and its implementations.

In this chapter, you will be learning the basic installation steps that are required to perform on different platforms (that is Windows, Linux, and Mac), about environment variables, setting up of environment variables, file formats, Python interactive shell, basic syntaxes and finally printing out formatted output.

4.1.1 Why Python?

Now you might be suddenly bogged with the question, why Python? According to Institute of Electrical and Electronics Engineers (IEEE) 2016 ranking Python ranked third after C and Java. As per Indeed.com's data of 2016, the Python job market search ranked fifth. Clearly, all the data points to the ever rising demand in the job market for Python. It’s a cool language if you want to learn just for fun or if you want to build your career around Python, you will adore the language. At school level, many schools have started including Python programming for kids. With new technologies taking the market by surprise Python has been playing a dominant role. Whether it is cloud platform, mobile app development, Big Data, IoT with Raspberry Pi, or the new Block chain technology, Python is being seen as a niche language platform to develop and deliver a scalable and robust applications.

Some key features of the language are:

- Python programs can run on any platform, you can carry code created in Windows machine and run it on Mac or Linux
- Python has inbuilt large library with prebuilt and portable functionality, also known as the standard library
- Python is an expressive language
- Python is free and open source
- Python code is about one third of the size of equivalent C++ and Java code
- Python can be both dynamically and strongly typed--dynamically typed means it is a type of variable that is interpreted at runtime, which means, in Python, there is no need to define the type (int or float) of the variable

Python applications

One of the most famous platforms where Python is extensively used is YouTube. The other places where you will find Python being extensively used are the special effects in Hollywood movies, drug evolution and discovery, traffic control systems, ERP systems, cloud hosting, e-commerce platform, CRM systems, and whatever field you can think of.

Versions

At the time of writing this book, two main versions of the Python programming language were available in the market, which are Python 2.x and

Python 3.x. The stable release as of writing the book were Python 2.7.13 and Python 3.6.0.

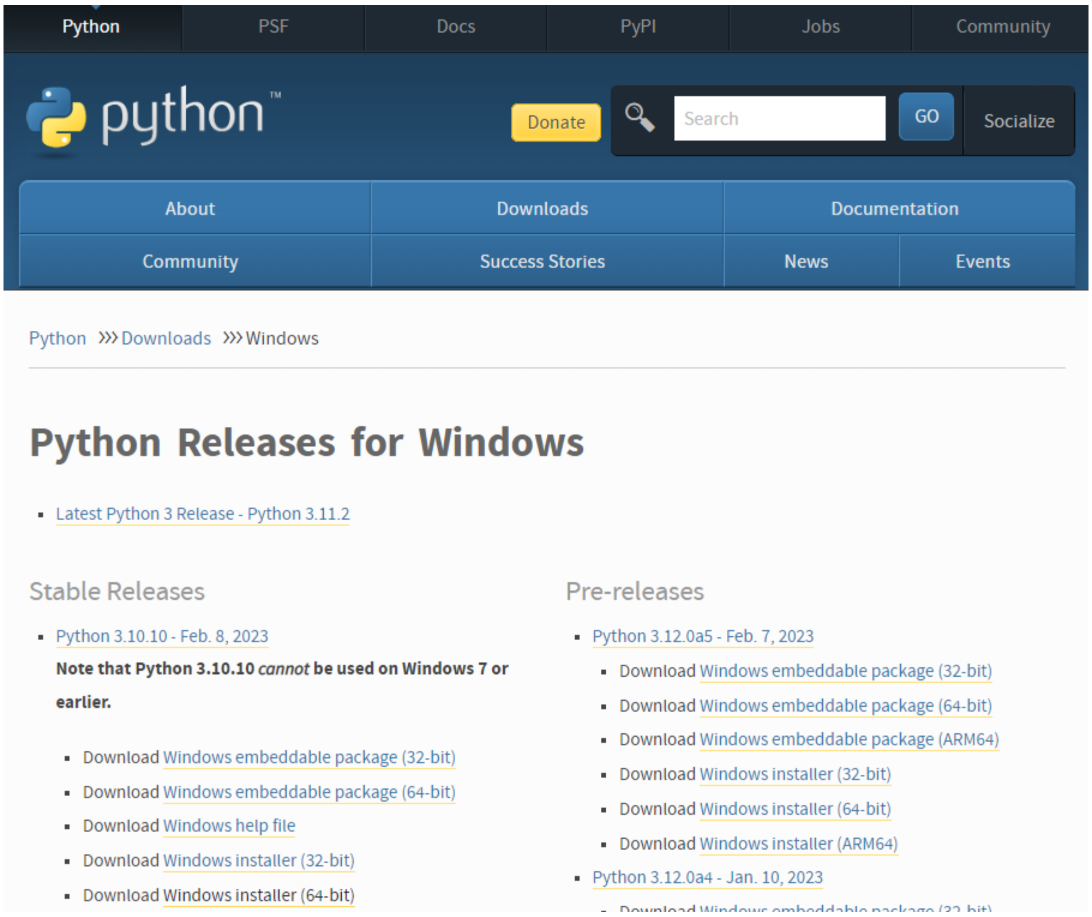
Implementations of Python

Major implementations include C-Python, Jython, IronPython, MicroPython, and PyPy.

4.1.2 Installation

STEP 1 — Downloading The Python Installer

1. Go to the official python download page for windows.
2. Find a stable python 3 release. This tutorial was tested with python version 3.10.10.
3. Click the appropriate link for your system to download the executable file: **windows installer (64-bit)** or **windows installer (32-bit)**.



Python »» Downloads »» Windows

Python Releases for Windows

- Latest Python 3 Release - Python 3.11.2

Stable Releases

- Python 3.10.10 - Feb. 8, 2023

  Note that Python 3.10.10 *cannot* be used on Windows 7 or earlier.

  - Download Windows embeddable package (32-bit)
  - Download Windows embeddable package (64-bit)
  - Download Windows help file
  - Download Windows installer (32-bit)
  - Download Windows installer (64-bit)

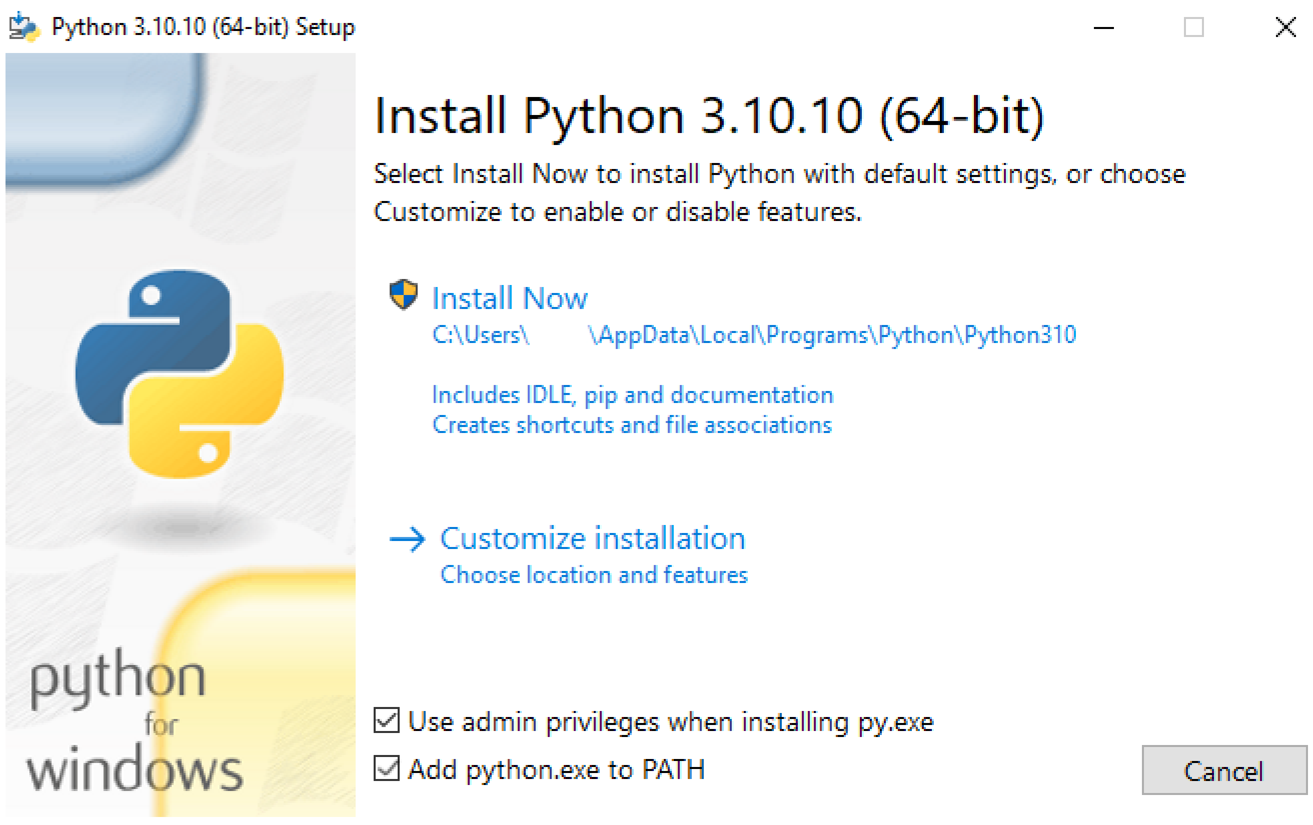
Pre-releases

- Python 3.12.0a5 - Feb. 7, 2023
  - Download Windows embeddable package (32-bit)
  - Download Windows embeddable package (64-bit)
  - Download Windows embeddable package (ARM64)
  - Download Windows installer (32-bit)
  - Download Windows installer (64-bit)
  - Download Windows installer (ARM64)
- Python 3.12.0a4 - Jan. 10, 2023

Fig 4.1: Python Download

STEP 2 — Running The Executable Installer

1. After the installer is downloaded, double-click the .exe file, for example python-3.10.10-amd64.exe, to run the python installer.
2. Select the **install launcher for all users** checkbox, which enables all users of the computer to access the python launcher application.
3. Select the **add python.exe to path** checkbox, which enables users to launch python from the command line.


Python 3.10.10 (64-bit) Setup

Install Python 3.10.10 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now
C:\Users\ \AppData\Local\Programs\Python\Python310

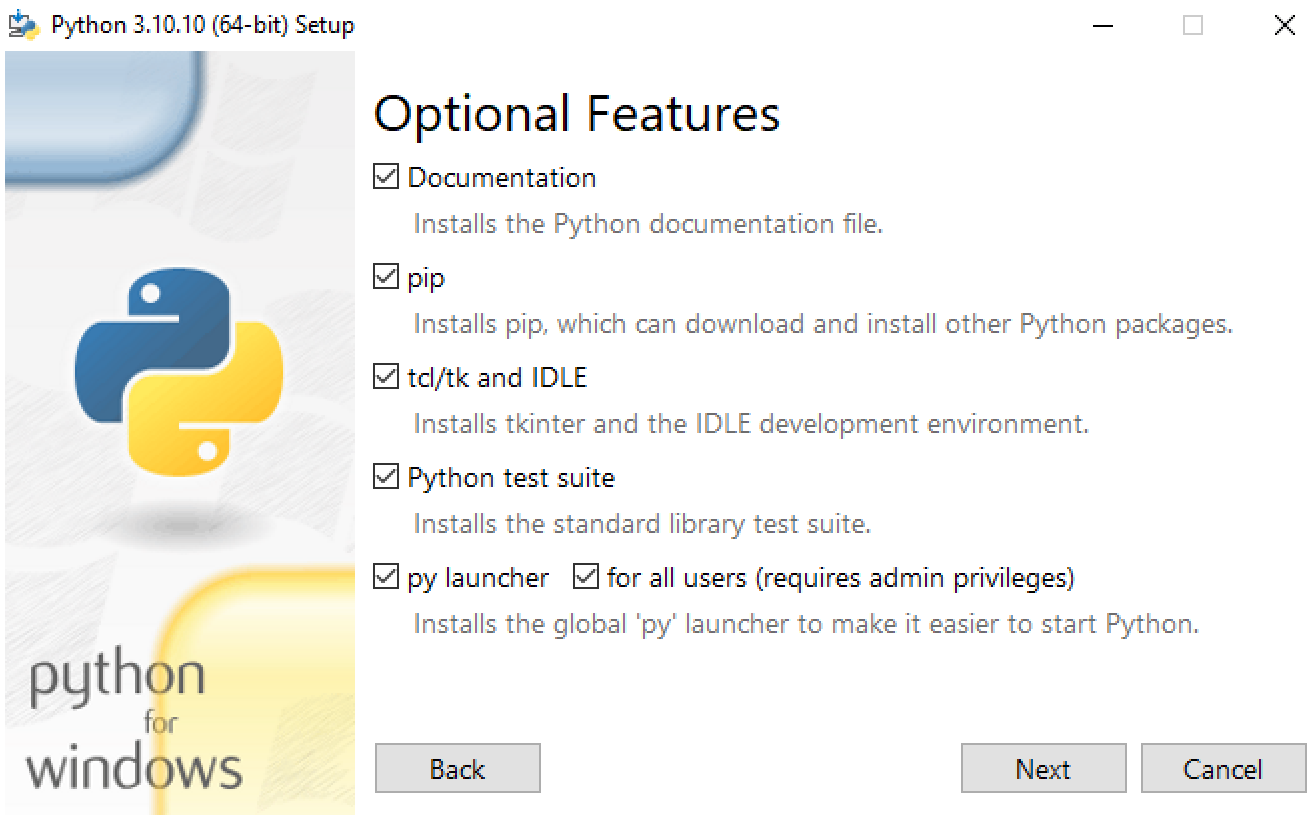
Includes IDLE, pip and documentation
Creates shortcuts and file associations

Customize installation
Choose location and features

☑ Use admin privileges when installing py.exe
☑ Add python.exe to PATH

Fig 4.2: Python Installation Step 1

4. If you're just getting started with python and you want to install it with default features as described in the dialog, then click **install now** and go to step 4 - verify the python installation. To install other optional and advanced features, click **customize installation** and continue.
5. The **optional features** include common tools and resources for python and you can install all of them, even if you don't plan to use them.


Python 3.10.10 (64-bit) Setup

Optional Features

☑ Documentation

Installs the Python documentation file.

☑ pip

Installs pip, which can download and install other Python packages.

☑ tcl/tk and IDLE

Installs tkinter and the IDLE development environment.

☑ Python test suite

Installs the standard library test suite.

☑ py launcher ☑ for all users (requires admin privileges)

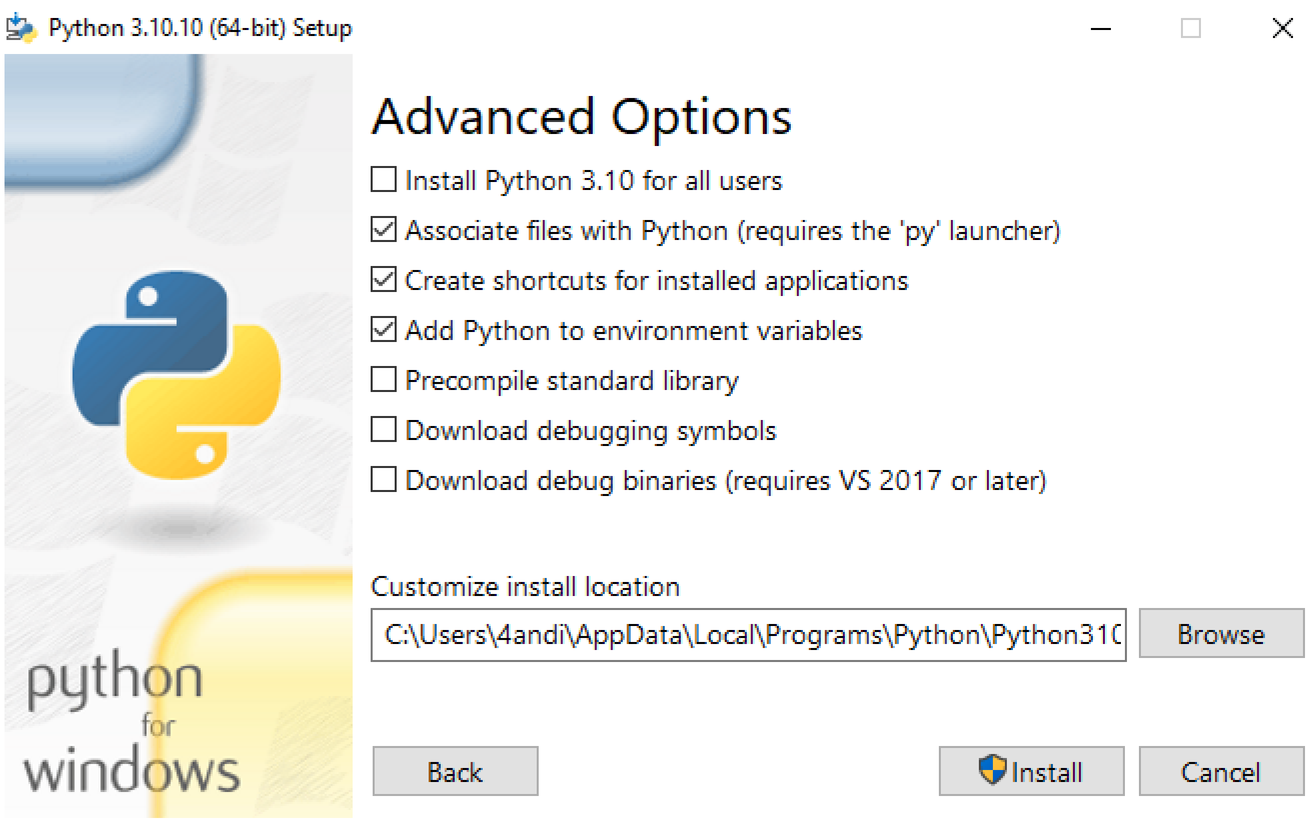
Installs the global 'py' launcher to make it easier to start Python.

Fig 4.3: Python Installation Step 2

Select some or all of the following options:

- **Documentation**: recommended
- **Pip**: recommended if you want to install other python packages, such as Numpy or pandas
- **Tcl/tk and idle**: recommended if you plan to use idle or follow tutorials that use it
- **Python test suite**: recommended for testing and learning
- **Py launcher** and **for all users**: recommended to enable users to launch python from the command line

6. Click **next**.
7. The **advanced options** dialog displays.


Python 3.10.10 (64-bit) Setup

Advanced Options

- ☐ Install Python 3.10 for all users
- ☑ Associate files with Python (requires the 'py' launcher)
- ☑ Create shortcuts for installed applications
- ☑ Add Python to environment variables
- ☐ Precompile standard library
- ☐ Download debugging symbols
- ☐ Download debug binaries (requires VS 2017 or later)

Customize install location

C:\Users\4andi\AppData\Local\Programs\Python\Python310

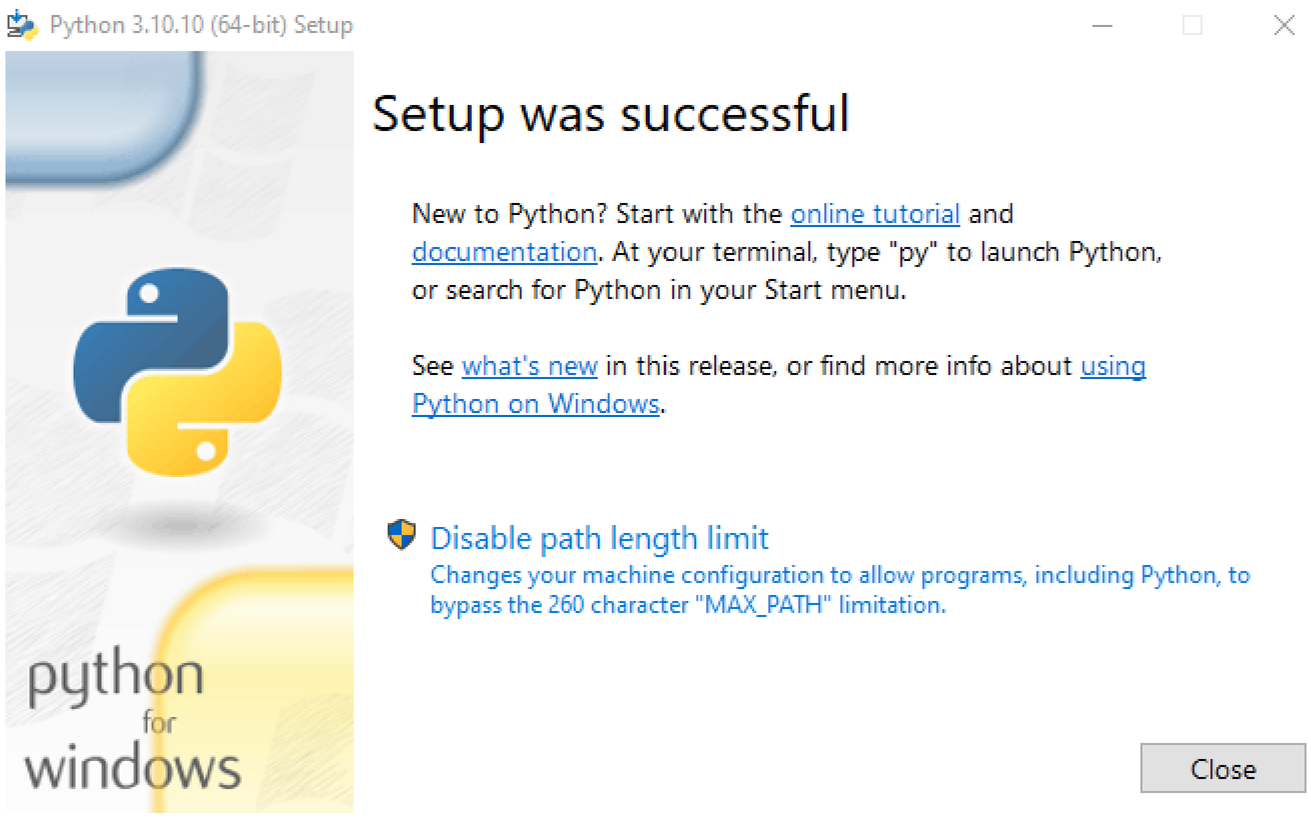
Fig 4.4: Python Installation Step 3

Select the options that suit your requirements:

- **Install for all users**: recommended if you're not the only user on this computer
- **Associate files with python**: recommended, because this option associates all the python file types with the launcher or editor
- **Create shortcuts for installed applications**: recommended to enable shortcuts for python applications
- **Add python to environment variables**: recommended to enable launching python
- **Precompile standard library**: not required, it might down the installation
- **Download debugging symbols** and **download debug binaries**: recommended only if you plan to create C or C++ extensions

Make note of the python installation directory in case you need to reference it later.

8. Click **install** to start the installation.
9. After the installation is complete, a **setup was successful** message displays.



Setup was successful

New to Python? Start with the online tutorial and documentation. At your terminal, type "py" to launch Python, or search for Python in your Start menu.

See what's new in this release, or find more info about using Python on Windows.

Disable path length limit
Changes your machine configuration to allow programs, including Python, to bypass the 260 character "MAX_PATH" limitation.

Fig 4.5: Python Installation Successful

STEP 3 — Adding Python to the Environment Variables (Optional)

Skip this step if you selected **add python to environment variables** during installation.

If you want to access python through the command line but you didn't add python to your environment variables during installation, then you can still do it manually.

Before you start, locate the python installation directory on your system. The following directories are examples of the default directory paths:

- C:\program files\python310: if you selected **install for all users** during installation, then the directory will be system wide
- C:\users\sammy\appdata\local\programs\python\python310: if you didn't select **install for all users** during installation, then the directory will be in the windows user path

  Note that the folder name will be different if you installed a different version, but will still start with python.

1. Go to **start** and enter advanced system settings in the search bar.
2. Click **view advanced system settings**.
3. In the **system properties** dialog, click the **advanced** tab and then click **environment variables**.
4. Depending on your installation:

- If you selected **install for all users** during installation, select **path** from the list of **system variables** and click **edit**.
- If you didn't select **install for all users** during installation, select **path** from the list of **user variables** and click **edit**.

5. Click **new** and enter the python directory path, then click **ok** until all the dialogs are closed.

STEP 4 — Verify the Python Installation

You can verify whether the python installation is successful either through the command line or through the integrated development environment (idle) application, if you chose to install it.

Go to **start** and enter cmd in the search bar. Click **command prompt**.

Enter the following command in the command prompt:

Python --version

An example of the output is:

Output

Python 3.10.10

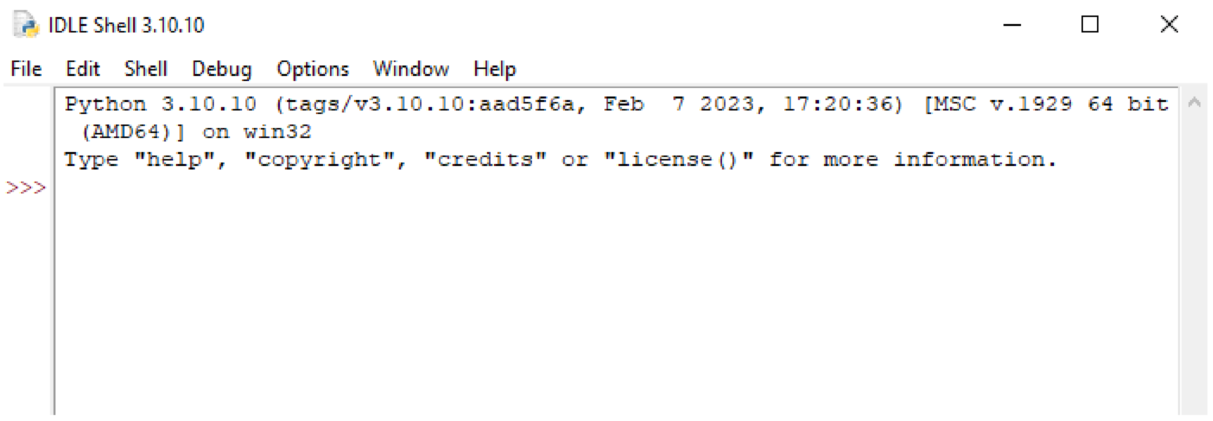
You can also check the version of python by opening the idle application. Go to **start** and enter python in the search bar and then click the idle app, for example **idle (python 3.10 64-bit)**.

4.1.3 Python file formats

Every language understands a file format, for example, like the C language file extension is .c likewise java language has a file extension .java. The Python file extension is .py while bytecode file extension is .pyc.

Python interactive shell

Python interactive shell is also known as **Integrated Development Environment** (**IDLE**). With the Python installer, two interactive shells are provided: one is IDLE (Python GUI) and the other is Python (command line). Both can be used for running simple programs. For complex programs and executing large files, the windows command prompt is used, where after the system variables are set automatically, large files are recognized and executed by the system.

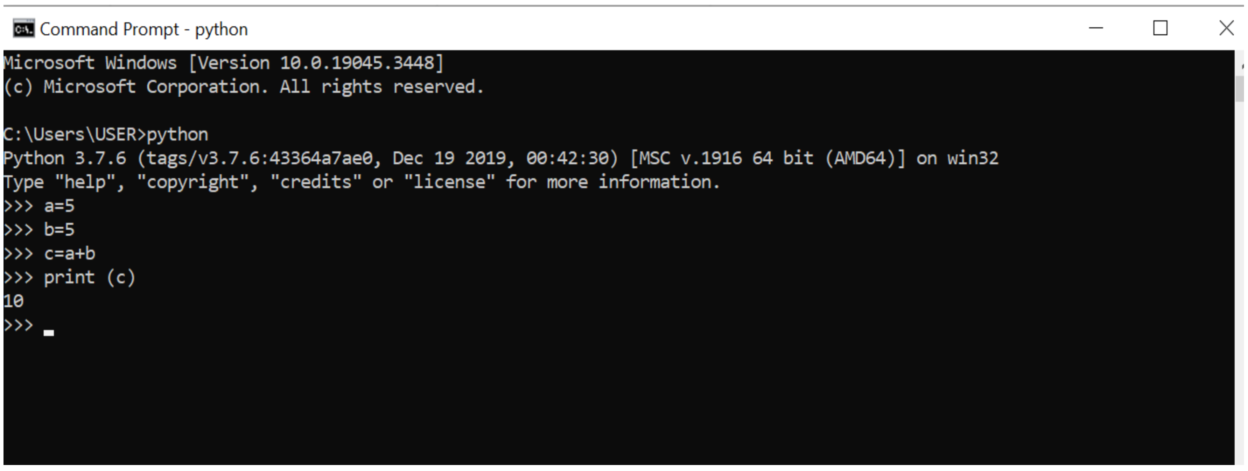

IDLE Shell 3.10.10

```
Python 3.10.10 (tags/v3.10.10:aad5f6a, Feb  7 2023, 17:20:36) [MSC v.1929 64 bit
 (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

Fig 4.6: Python IDLE Shell

You can start coding in python using idle or your preferred code editor.

The preceding screenshot is what we call Python IDLE, which comes bundled with the Python installation. The next screenshot is of the command line that also comes bundled with the Python installation, or we can simply launch the Python command through the windows command line and get Python command line. For most of our programming instructions, we will be using the Python command line:


Command Prompt - python

```
Microsoft Windows [Version 10.0.19045.3448]
(c) Microsoft Corporation. All rights reserved.

C:\Users\USER>python
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a=5
>>> b=5
>>> c=a+b
>>> print (c)
10
>>>
```

Fig 4.7: Python Console

4.1.4 Syntax and semantics

Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use curly brackets to delimit blocks, and semicolons after statements are optional. It has fewer syntactic exceptions and special cases than C or Pascal.

Indentation

Python uses whitespace indentation, rather than curly brackets or keywords, to delimit blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block. Thus, the program's visual structure accurately represents the program's semantic structure. This feature is sometimes termed the off-side rule, which some other languages share, but in most languages indentation doesn't have any semantic meaning.

Statements and control flow

Python's statements include (among others):

- The **assignment** statement (token '=', the equals sign). This operates differently than in traditional imperative programming languages, and this fundamental mechanism (including the nature of Python's version of variables) illuminates many other features of the language. Assignment in C, e.g., x = 2, translates to "typed variable name x receives a copy of numeric value 2". The (right-hand) value is copied into an allocated storage location for which the (left-hand) variable name is the symbolic address. The memory allocated to the variable is large enough (potentially quite large) for the declared type. In the simplest case of Python assignment, using the same example, x = 2, translates to "(generic) name x receives a reference to a separate, dynamically allocated object of numeric (int) type of value 2." This is termed binding the name to the object. Since the name's storage location doesn't contain the indicated value, it is improper to call it a variable. Names may be subsequently rebound at any time to objects of greatly varying types, including strings, procedures, complex objects with data and methods, etc. Successive assignments of a common value to multiple names, e.g., x = 2; y = 2; z = 2 result in allocating storage to (at most) three names and one numeric object, to which all three names are bound. Since a name is a generic reference holder it is unreasonable to associate a fixed data type with it. However at a given time a name will be bound to some object, which will have a type; thus there is dynamic typing.

- The **if** statement, which conditionally executes a block of code, along with else and elif (a contraction of else-if).
- The **for** statement, which iterates over an iterable object, capturing each element to a local variable for use by the attached block.
- The **while** statement, which executes a block of code as long as its condition is true.
- The **try** statement, which allows exceptions raised in its attached code block to be caught and handled by except clauses; it also ensures that clean-up code in a finally block will always be run regardless of how the block exits.
- The **raise** statement, used to raise a specified exception or re-raise a caught exception.
- The **class** statement, which executes a block of code and attaches its local namespace to a class, for use in object-oriented programming.
- The **def** statement, which defines a function or method.

- The **with** statement, from Python 2.5 released in September 2006, which encloses a code block within a context manager (for example, acquiring a lock before the block of code is run and releasing the lock afterwards, or opening a file and then closing it), allowing Resource Acquisition Is Initialization (RAII)-like behaviour and replaces a common try/finally idiom.
- The **break** statement, exits from the loop.
- The **continue** statement, skips this iteration and continues with the next item.
- The **pass** statement, which serves as a NOP. It is syntactically needed to create an empty code block.
- The **assert** statement, used during debugging to check for conditions that ought to apply.
- The **yield** statement, which returns a value from a generator function. From Python 2.5, yield is also an operator. This form is used to implement coroutines.
- The **import** statement, which is used to import modules whose functions or variables can be used in the current program. There are three ways of using import: import <module name> [as <alias>] or from <module name> import * or from <module name> import <definition 1> [as <alias 1>], <definition 2> [as <alias 2>], ...
- The **print** statement was changed to the print() function in Python 3.

4.2 Result

Dataset

Below picture is dataset of leaves this leaf are healthy, bacterial, mold etc..

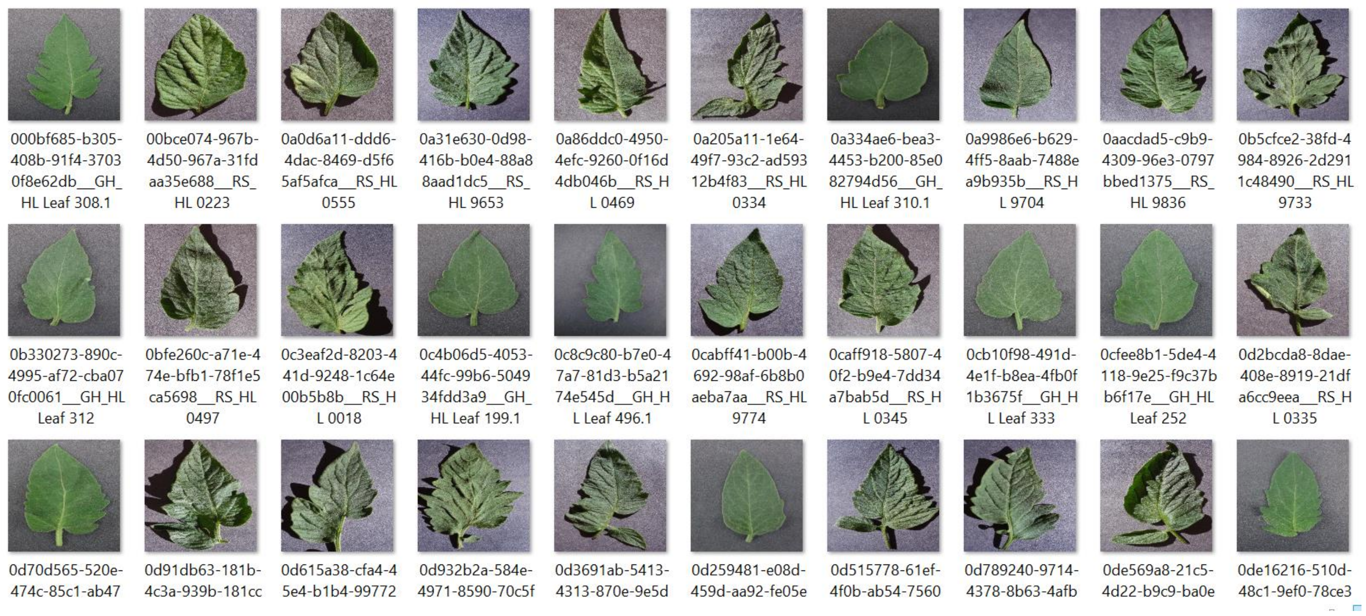
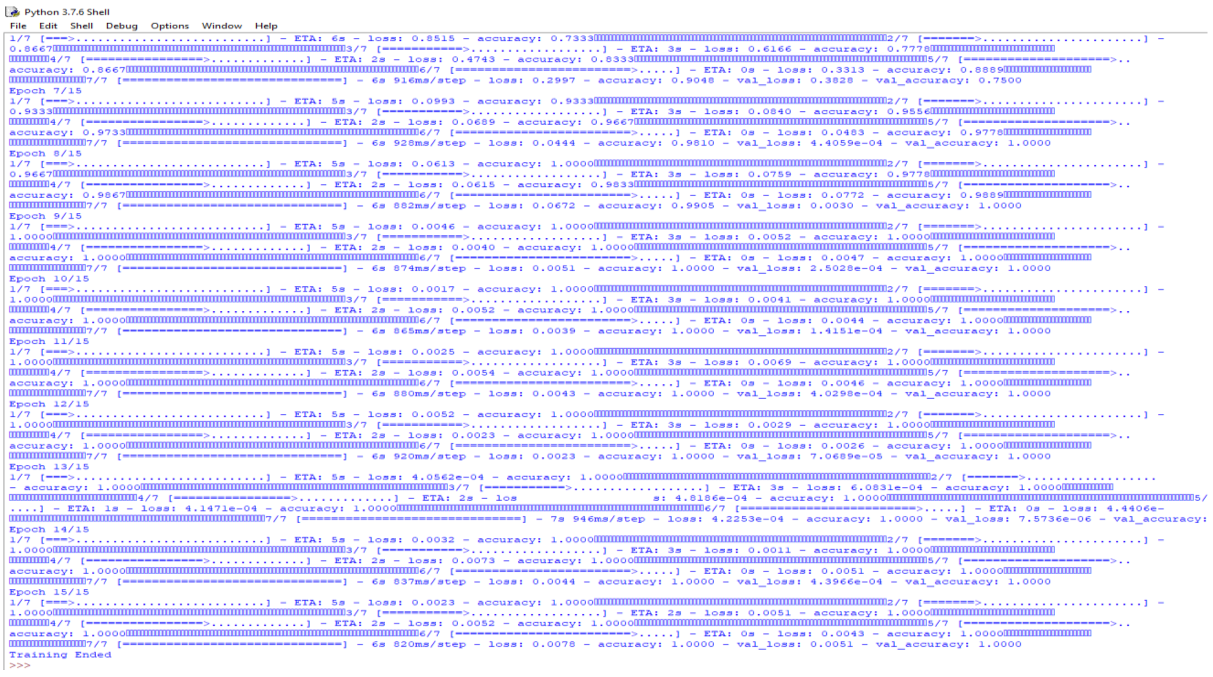

000bf685-b305-408b-91f4-37030f8e62db__GH_HL Leaf 308.1
00bce074-967b-4d50-967a-31fdaa35e688__RS_HL 0223
0a0d6a11-ddd6-4dac-8469-d5f65af5afca__RS_HL 0555
0a31e630-0d98-416b-b0e4-88a88aad1dc5__RS_HL 9653
0a86ddc0-4950-4efc-9260-0f16d4db046b__RS_HL 0469
0a205a11-1e64-49f7-93c2-ad59312b4f83__RS_HL 0334
0a334ae6-bea3-4453-b200-85e082794d56__GH_HL Leaf 310.1
0a9986e6-b629-4ff5-8aab-7488ea9b935b__RS_HL 9704
0aacdad5-c9b9-4309-96e3-0797bbed1375__RS_HL 9836
0b5cfce2-38fd-4984-8926-2d2911c48490__RS_HL 9733
0b330273-890c-4995-af72-cba070fc0061__GH_HL Leaf 312
0bfe260c-a71e-474e-bfb1-78f1e5ca5698__RS_HL 0497
0c3eaf2d-8203-441d-9248-1c64e00b5b8b__RS_HL 0018
0c4b06d5-4053-44fc-99b6-504934fdd3a9__GH_HL Leaf 199.1
0c8c9c80-b7e0-47a7-81d3-b5a2174e545d__GH_HL Leaf 496.1
0cabff41-b00b-4692-98af-6b8b0aeba7aa__RS_HL 9774
0caff918-5807-40f2-b9e4-7dd34a7bab5d__RS_HL 0345
0cb10f98-491d-4e1f-b8ea-4fb0f1b3675f__GH_HL Leaf 333
0cfee8b1-5de4-4118-9e25-f9c37bb6f17e__GH_HL Leaf 252
0d2bcda8-8dae-408e-8919-21dfa6cc9eea__RS_HL 0335
0d70d565-520e-474c-85c1-ab47
0d91db63-181b-4c3a-939b-181cc
0d615a38-cfa4-45e4-b1b4-99772
0d932b2a-584e-4971-8590-70c5f
0d3691ab-5413-4313-870e-9e5d
0d259481-e08d-459d-aa92-fe05e
0d515778-61ef-4f0b-ab54-7560
0d789240-9714-4378-8b63-4afb
0de569a8-21c5-4d22-b9c9-ba0e
0de16216-510d-48c1-9ef0-78ce3

Fig 4.8: Dataset

Training



```
1/7 [===>..........................] - ETA: 6s - loss: 0.8515 - accuracy: 0.73332/7 [=======>......................] -
0.86673/7 [===========>..................] - ETA: 3s - loss: 0.6166 - accuracy: 0.7778
4/7 [=================>............] - ETA: 2s - loss: 0.4743 - accuracy: 0.83335/7 [=====================>..
accuracy: 0.86676/7 [=========================>.....] - ETA: 0s - loss: 0.3313 - accuracy: 0.8889
7/7 [==============================] - 6s 916ms/step - loss: 0.2997 - accuracy: 0.9048 - val_loss: 0.3828 - val_accuracy: 0.7500
Epoch 7/15
1/7 [===>..........................] - ETA: 5s - loss: 0.0993 - accuracy: 0.93332/7 [=======>......................] -
0.93333/7 [===========>..................] - ETA: 3s - loss: 0.0840 - accuracy: 0.9556
4/7 [=================>............] - ETA: 2s - loss: 0.0689 - accuracy: 0.96675/7 [=====================>..
accuracy: 0.97336/7 [=========================>.....] - ETA: 0s - loss: 0.0483 - accuracy: 0.9778
7/7 [==============================] - 6s 928ms/step - loss: 0.0444 - accuracy: 0.9810 - val_loss: 4.4059e-04 - val_accuracy: 1.0000
Epoch 8/15
1/7 [===>..........................] - ETA: 5s - loss: 0.0613 - accuracy: 1.00002/7 [=======>......................] -
0.96673/7 [===========>..................] - ETA: 3s - loss: 0.0759 - accuracy: 0.9778
4/7 [=================>............] - ETA: 2s - loss: 0.0615 - accuracy: 0.98335/7 [=====================>..
accuracy: 0.98676/7 [=========================>.....] - ETA: 0s - loss: 0.0772 - accuracy: 0.9889
7/7 [==============================] - 6s 882ms/step - loss: 0.0672 - accuracy: 0.9905 - val_loss: 0.0030 - val_accuracy: 1.0000
Epoch 9/15
1/7 [===>..........................] - ETA: 5s - loss: 0.0046 - accuracy: 1.00002/7 [=======>......................] -
1.00003/7 [===========>..................] - ETA: 3s - loss: 0.0052 - accuracy: 1.0000
4/7 [=================>............] - ETA: 2s - loss: 0.0040 - accuracy: 1.00005/7 [=====================>..
accuracy: 1.00006/7 [=========================>.....] - ETA: 0s - loss: 0.0047 - accuracy: 1.0000
7/7 [==============================] - 6s 874ms/step - loss: 0.0051 - accuracy: 1.0000 - val_loss: 2.5028e-04 - val_accuracy: 1.0000
Epoch 10/15
1/7 [===>..........................] - ETA: 5s - loss: 0.0017 - accuracy: 1.00002/7 [=======>......................] -
1.00003/7 [===========>..................] - ETA: 3s - loss: 0.0041 - accuracy: 1.0000
4/7 [=================>............] - ETA: 2s - loss: 0.0052 - accuracy: 1.00005/7 [=====================>..
accuracy: 1.00006/7 [=========================>.....] - ETA: 0s - loss: 0.0044 - accuracy: 1.0000
7/7 [==============================] - 6s 865ms/step - loss: 0.0039 - accuracy: 1.0000 - val_loss: 1.4151e-04 - val_accuracy: 1.0000
Epoch 11/15
1/7 [===>..........................] - ETA: 5s - loss: 0.0025 - accuracy: 1.00002/7 [=======>......................] -
1.00003/7 [===========>..................] - ETA: 3s - loss: 0.0069 - accuracy: 1.0000
4/7 [=================>............] - ETA: 2s - loss: 0.0054 - accuracy: 1.00005/7 [=====================>..
accuracy: 1.00006/7 [=========================>.....] - ETA: 0s - loss: 0.0046 - accuracy: 1.0000
7/7 [==============================] - 6s 880ms/step - loss: 0.0043 - accuracy: 1.0000 - val_loss: 4.0298e-04 - val_accuracy: 1.0000
Epoch 12/15
1/7 [===>..........................] - ETA: 5s - loss: 0.0052 - accuracy: 1.00002/7 [=======>......................] -
1.00003/7 [===========>..................] - ETA: 3s - loss: 0.0029 - accuracy: 1.0000
4/7 [=================>............] - ETA: 2s - loss: 0.0023 - accuracy: 1.00005/7 [=====================>..
accuracy: 1.00006/7 [=========================>.....] - ETA: 0s - loss: 0.0026 - accuracy: 1.0000
7/7 [==============================] - 6s 920ms/step - loss: 0.0023 - accuracy: 1.0000 - val_loss: 7.0689e-05 - val_accuracy: 1.0000
Epoch 13/15
1/7 [===>..........................] - ETA: 5s - loss: 4.0562e-04 - accuracy: 1.00002/7 [=======>.................
- accuracy: 1.00003/7 [===========>..................] - ETA: 3s - loss: 6.0831e-04 - accuracy: 1.0000
4/7 [=================>............] - ETA: 2s - los                s: 4.8186e-04 - accuracy: 1.00005/
....] - ETA: 1s - loss: 4.1471e-04 - accuracy: 1.00006/7 [=========================>.....] - ETA: 0s - loss: 4.4406e-
7/7 [==============================] - 7s 946ms/step - loss: 4.2253e-04 - accuracy: 1.0000 - val_loss: 7.5736e-06 - val_accuracy:
Epoch 14/15
1/7 [===>..........................] - ETA: 5s - loss: 0.0032 - accuracy: 1.00002/7 [=======>......................] -
1.00003/7 [===========>..................] - ETA: 3s - loss: 0.0011 - accuracy: 1.0000
4/7 [=================>............] - ETA: 2s - loss: 0.0073 - accuracy: 1.00005/7 [=====================>..
accuracy: 1.00006/7 [=========================>.....] - ETA: 0s - loss: 0.0051 - accuracy: 1.0000
7/7 [==============================] - 6s 837ms/step - loss: 0.0044 - accuracy: 1.0000 - val_loss: 4.3966e-04 - val_accuracy: 1.0000
Epoch 15/15
1/7 [===>..........................] - ETA: 5s - loss: 0.0023 - accuracy: 1.00002/7 [=======>......................] -
1.00003/7 [===========>..................] - ETA: 2s - loss: 0.0051 - accuracy: 1.0000
4/7 [=================>............] - ETA: 2s - loss: 0.0052 - accuracy: 1.00005/7 [=====================>..
accuracy: 1.00006/7 [=========================>.....] - ETA: 0s - loss: 0.0043 - accuracy: 1.0000
7/7 [==============================] - 6s 820ms/step - loss: 0.0078 - accuracy: 1.0000 - val_loss: 0.0051 - val_accuracy: 1.0000
Training Ended
>>>
```

Fig 4.9: Machine Training

Accuracy Graph Loss Graph

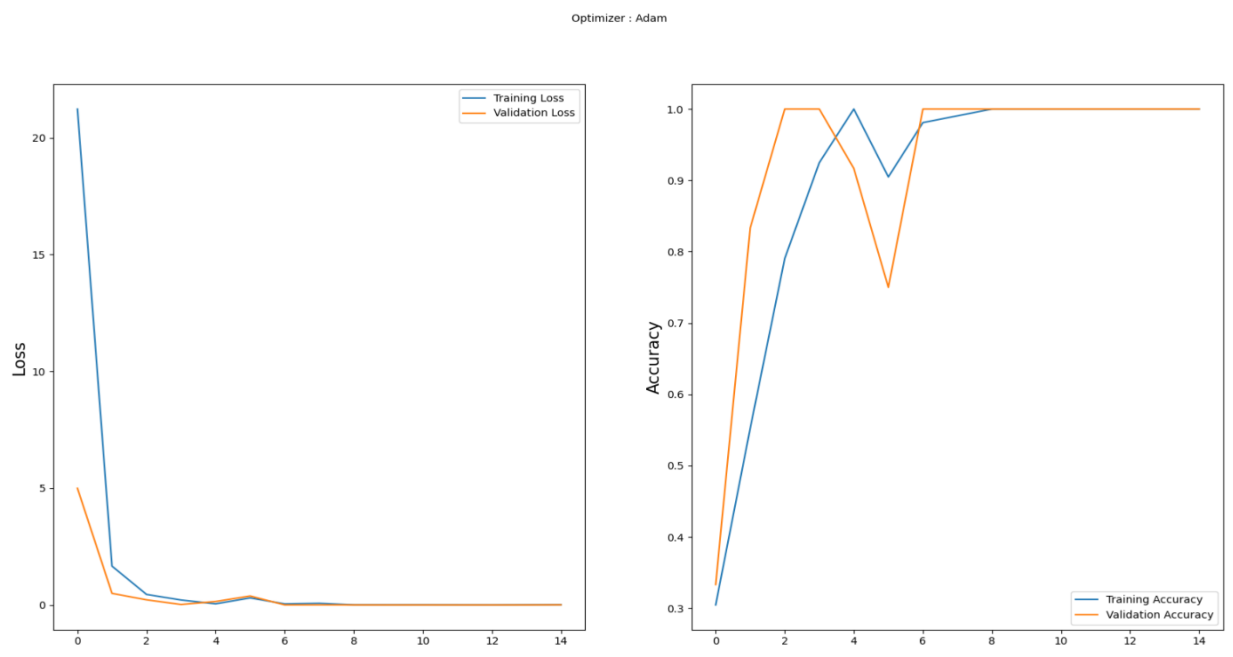


Fig 4.10: Accuracy Graph and Loss Graph

Testing

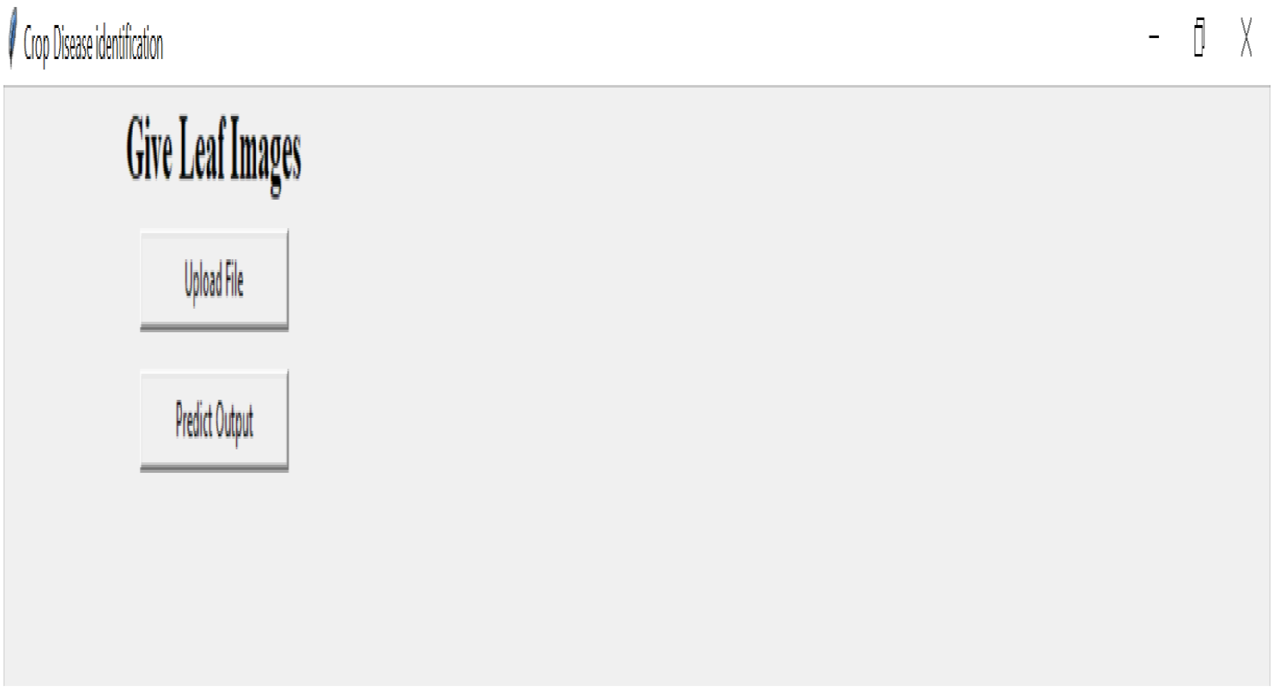
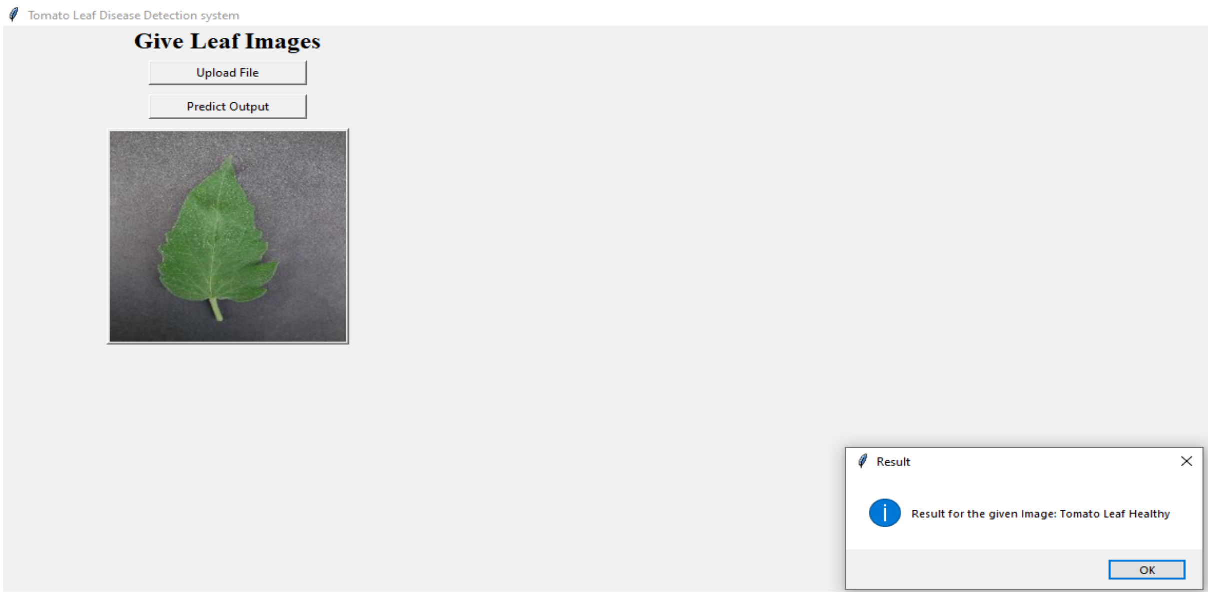


Fig 4.11: Above the picture testing home page,this page is used to upload the leaf image then predicted the output.

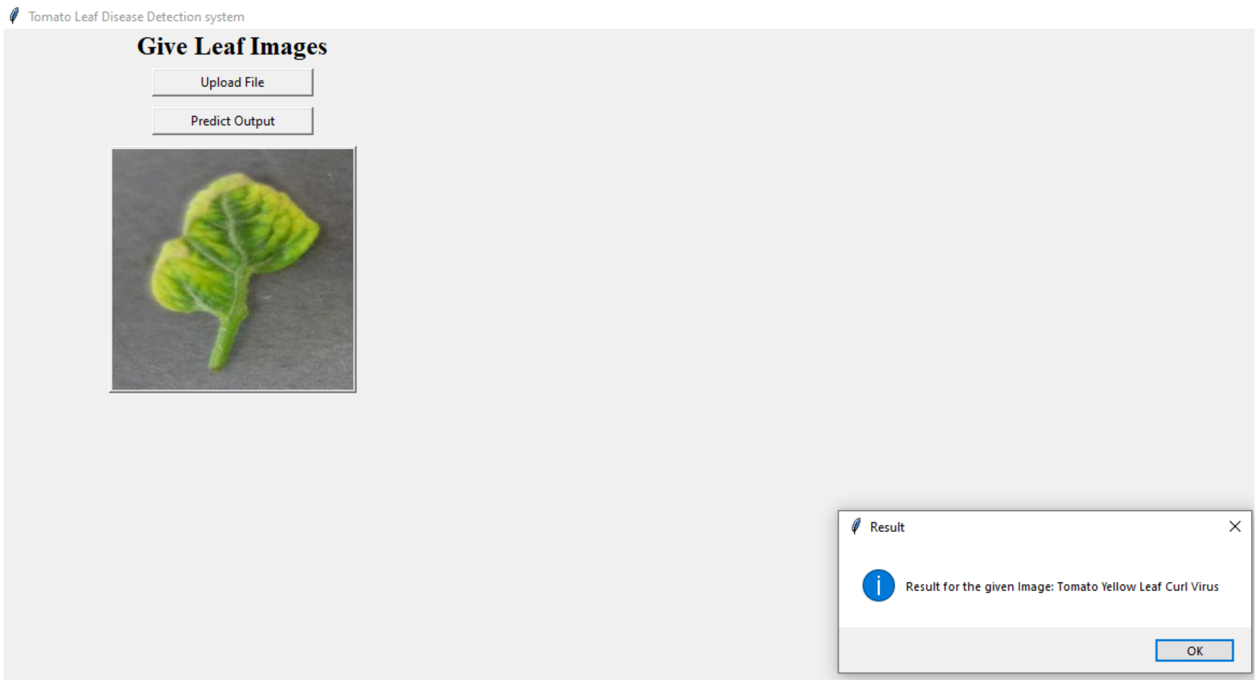
Result-1 health leaf



Result for the given Image: Tomato Leaf Healthy

Fig 4.12: Above the image is result of health leaf

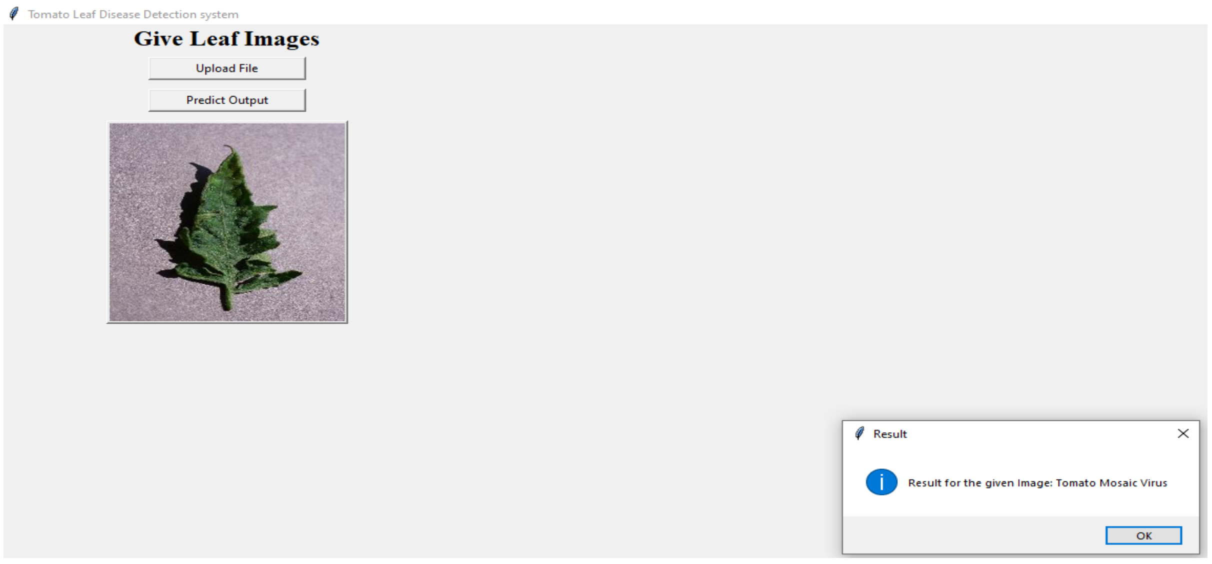
Result-2 Yellow Leaf Curl Detected



Result for the given Image: Tomato Yellow Leaf Curl Virus

Fig 4.13: Above the image is result of Yellow Leaf Curl Detected

Result-3 Mosaic Virus Detected



Result for the given Image: Tomato Mosaic Virus

Fig 4.14: Above the image is result of Mosaic Virus Detected

CHAPTER-V

APPENDIX

5.1 Source Code

DCNN TRAIN.py

```
import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.layers import *

from tensorflow.keras.models import *

from tensorflow.keras.preprocessing import image

#Training model

model = Sequential()   ## creating a blank model

model.add(Conv2D(32,kernel_size=(3,3),activation='relu',input_shape=(224,22
4,3)))

model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64,(3,3),activation='relu'))

model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Dropout(0.25))    ### reduce the overfitting

model.add(Flatten())    ### input layer

model.add(Dense(256,activation='relu'))    ## hidden layer of ann

model.add(Dropout(0.5))

model.add(Dense(3,activation='softmax'))   ## output layer
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accu
racy'])

model.summary()

#Moulding train images

train_datagen = image.ImageDataGenerator(rescale = 1./255, shear_range =
0.2,zoom_range = 0.2, horizontal_flip = True)

test_dataset = image.ImageDataGenerator(rescale=1./255)

#Reshaping test and validation images

train_generator = train_datagen.flow_from_directory(

    'dataset/train',

    target_size = (224,224),

    batch_size = 15,

    class_mode = 'categorical')

validation_generator = test_dataset.flow_from_directory(

    'dataset/val',

    target_size = (224,224),

    batch_size = 15,

    class_mode = 'categorical')

#### Train the model

history = model.fit_generator(
    train_generator,
    steps_per_epoch=7,
    epochs = 15,
    validation_data = validation_generator)
model.save("dcnn.h5");
print("Training Ended")
plt.figure(figsize=(20,10))
plt.subplot(1, 2, 1)
plt.suptitle('Optimizer : Adam', fontsize=10)
plt.ylabel('Loss', fontsize=16)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend(loc='upper right')
plt.subplot(1, 2, 2)
plt.ylabel('Accuracy', fontsize=16)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend(loc='lower right')
plt.show()
```

Application.py

```
import tkinter as tk

from tkinter import filedialog

from PIL import Image, ImageTk

import os

import numpy as np

from tensorflow.keras.models import load_model

from tensorflow.keras.preprocessing import image

from tkinter import messagebox

model1 = load_model("dcnn.h5")

print("success")

my_w = tk.Tk()

my_w.geometry("400x400")

my_w.title('Tomato Leaf Disease Detection system')

my_font1 = ('times', 18, 'bold')

l1 = tk.Label(my_w, text='Give Leaf Images', width=30, font=my_font1)

l1.grid(row=1, column=1)

import os

def upload_file():

    global img
    global filename
    f_types = [('Jpg Files', '*.jpg')]
    filename = filedialog.askopenfilename(filetypes=f_types)
    if filename:
        file_directory = os.path.dirname(filename)
        desired_directory = r"D:\leaf\test"
        desired_directory = os.path.normpath(desired_directory)
        file_directory = os.path.normpath(file_directory)
        if file_directory == desired_directory:
            if filename.lower().endswith('.jpg'):
                image_obj = Image.open(filename)
                imgs = image_obj.resize((224, 224))  # Resize the image
                img = ImageTk.PhotoImage(imgs)
                b2 = tk.Button(my_w, image=img)
                b2.grid(row=9, column=1, padx=5, pady=5)
                print(filename)
        else:
            messagebox.showerror("Error", "Please select a file from the Unwanted
image.")
    else:
        messagebox.showerror("Error", "Please select a file.")
```

```
# Button to upload file
b1 = tk.Button(my_w, text='Upload File', width=20, command=upload_file)
b1.grid(row=2, column=1, padx=5, pady=5)
# Function to predict output
def predict():
    global filename
    if filename:
        # Initialize variables
        ft, st, lt, rt, ut = 0, 0, 0, 0, 0
        h = ""
        out = ""
        outv = 5
        # Load and preprocess the image
        img = image.load_img(filename, target_size=(224, 224))
        img = image.img_to_array(img, dtype='uint8')
        img = np.expand_dims(img, axis=0)
        # Make predictions using the loaded model
        y_pred = np.argmax(model1.predict(img), axis=-1)
        # Determine the result based on the predicted class
        if y_pred[0] == 0:
            out = "Result for the given Image: Tomato Yellow Leaf Curl Virus"
            outv = 0
        elif y_pred[0] == 1:
            out = "Result for the given Image: Tomato Mosaic Virus"
            outv = 1
        elif y_pred[0] == 2:
            out = "Result for the given Image: Tomato Leaf Healthy"
            outv = 1
        print(out)
        # Show result in a messagebox
        messagebox.showinfo("Result", out)
    else:
        # Display error message if no file is uploaded
        messagebox.showerror("Error", "Please upload an image first.")
# Button to predict output
b3 = tk.Button(my_w, text='Predict Output', width=20, command=predict)
b3.grid(row=6, column=1, padx=5, pady=5)
my_w.mainloop()  # Keep the window open
```

CHAPTER-VI

CONCLUSION

In this project, we applied the deep convolutional neural network architecture to the task of plant disease classification. Plant diseases can have a devastating impact on crop yields and food security, making early and accurate detection crucial. By leveraging deep learning techniques like DCNN, we have made significant strides in automating the detection process. Our results demonstrate the effectiveness of DCNN in accurately classifying various plant diseases. The model achieved a high level of accuracy in identifying these diseases, surpassing traditional methods and showcasing the potential for AI in agriculture.

6.1 Future Enhancement

Future work should explore the development of mobile or edge-based applications that can provide real-time plant disease detection and recommendations to farmers. Additionally, improvements in model interpretability and robustness are important for practical implementation. In conclusion, this project illustrates the potential of DCNN-based deep learning models for automating the detection of plant diseases. The accuracy and generalizability of our model show great promise for improving crop yields and food security, and with further research and data collection, this technology can have a significant impact on agriculture.

6.2 REFERENCES

1. A. K. Pradhan, S. Swain, and J. K. Rout, ‘‘Role of machine learning and cloud-driven platform in IoT-based smart farming,’’2022.

2. A. Bhargava and A. Bansal, ‘‘Fruits and vegetables quality evaluation using computer vision: A review,’’2021.

3. A. Ahmad, D. Saraswat, and A. El Gamal, ‘‘A survey on using deep learning techniques for plant disease diagnosis and recommendations for development of appropriate tools,’’ 2023

4. R. Karthik, M. Hariharan, S. Anand, P. Mathikshara, A. Johnson, and R. Menaka, ‘‘Attention embedded residual CNN for disease detection in tomato leaves,’’ 2020.

5. O. O. Abayomi-Alli, R. Damasevicius, S. Misra, and R. Maskeliunas, ‘‘Cassava disease recognition from low-quality images using enhanced data augmentation model and deep learning,’’ 2021.

6. H.-T. Thai, N.-Y. Tran-Van, and K.-H. Le, ‘‘Artificial cognition for early leaf disease detection using vision transformers,’’ 2021.

7. A. Khamparia, G. Saini, D. Gupta, A. Khanna, S. Tiwari, and V. H. C. de Albuquerque, ‘‘Seasonal crops disease prediction and classification using deep convolutional encoder network,’’ 2020.

8. A. J. Rozaqi and A. Sunyoto, ‘‘Identification of disease in potato leaves using convolutional neural network (CNN) algorithm,’’ 2020.

9. A. Singh and H. Kaur, ‘‘Potato plant leaves disease detection and classification using machine learning methodologies,’’ 2021.

10. J. Johnson, G. Sharma, S. Srinivasan, S. K. Masakapalli, S. Sharma, J. Sharma, and V. K. Dua, ‘‘Enhanced field-based detection of potato blight in complex backgrounds using deep learning,’’2021.