

CHAPTER 1: INTRODUCTION

Introduction:-

Indian economy is dependent of agricultural productivity. Over 70% of rural homes depend on agriculture. Agriculture pays about 17% to the total GDP and provides employment to over 60% of the population. Therefore detection of plant rice leaf diseases plays a vital key role in the arena of agriculture. Indian agriculture is composed of many crops like rice, wheat. Indian farmers also grow sugarcane, oilseeds, potatoes and non-food items like coffee, tea, cotton, rubber. All these crops grow based on strength of leaves and roots. There are things that lead to different rice leaf disease for the plant leaves, which spoiled crops and finally it will effect on economy of the country.

CHAPTER 2: LITERATURE SURVEY

FUNDAMENTALS OF DIGITAL IMAGE

2.1 IMAGE:

An image is a two-dimensional picture, which has a similar appearance to some subject usually a physical object or a person.

Image is a two-dimensional, such as a photograph, screen display, and as well as a three-dimensional, such as a statue. They may be captured by optical devices—such as cameras, mirrors, lenses, telescopes, microscopes, etc. and natural objects and phenomena, such as the human eye or water surfaces.

The word image is also used in the broader sense of any two-dimensional figure such as a map, a graph, a pie chart, or an abstract painting. In this wider sense, images can also be rendered manually, such as by drawing, painting, carving, rendered automatically by printing or computer graphics technology, or developed by a combination of methods, especially in a pseudo-photograph.



Fig: Colour image to Gray scale Conversion Process

An image is a rectangular grid of pixels. It has a definite height and a definite width counted in pixels. Each pixel is square and has a fixed size on a given display. However different computer monitors may use different sized pixels. The pixels that constitute an image are ordered as a grid (columns and rows); each pixel consists of numbers representing magnitudes of brightness and color.

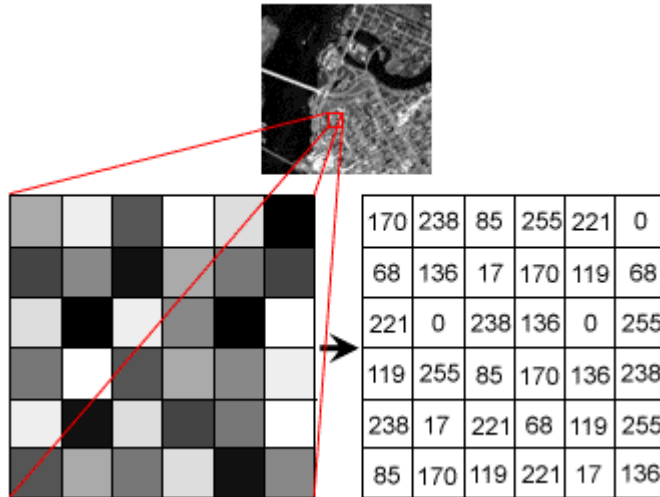


Fig: Gray Scale Image Pixel Value Analysis

Each pixel has a color. The color is a 32-bit integer. The first eight bits determine the redness of the pixel, the next eight bits the greenness, the next eight bits the blueness, and the remaining eight bits the transparency of the pixel.

Fig: BIT Transferred for Red, Green and Blue plane (24bit=8bit red;8-bit green;8bit blue)

2.2 IMAGE FILE SIZES:

Image file size is expressed as the number of bytes that increases with the number of pixels composing an image, and the color depth of the pixels. The greater the number of rows and columns, the greater the image resolution, and the larger the file. Also, each pixel of an image increases in size when its color depth increases,

an 8-bit pixel (1 byte) stores 256 colors, a 24-bit pixel (3 bytes) stores 16 million colors, the latter known as true color. Image compression uses algorithms to decrease the size of a file. High resolution cameras produce large image files, ranging from hundreds of kilobytes to megabytes, per the camera's resolution and the image-storage format capacity. High resolution digital cameras record 12 megapixel (1MP = 1,000,000 pixels / 1 million) images, or more, in true color. For example, an image recorded by a 12 MP camera; since each pixel uses 3 bytes to record true color, the uncompressed image would occupy 36,000,000 bytes of memory, a great amount of digital storage for one image, given that cameras must record and store many images to be practical. Faced with large file sizes, both within the camera and a storage disc, image file formats were developed to store such large images.

2.3 IMAGE FILE FORMATS:

Image file formats are standardized means of organizing and storing images. This entry is about digital image formats used to store photographic and other images. Image files are composed of either pixel or vector (geometric) data that are rasterized to pixels when displayed (with few exceptions) in a vector graphic display. Including proprietary types, there are hundreds of image file types. The PNG, JPEG, and GIF formats are most often used to display images on the Internet.

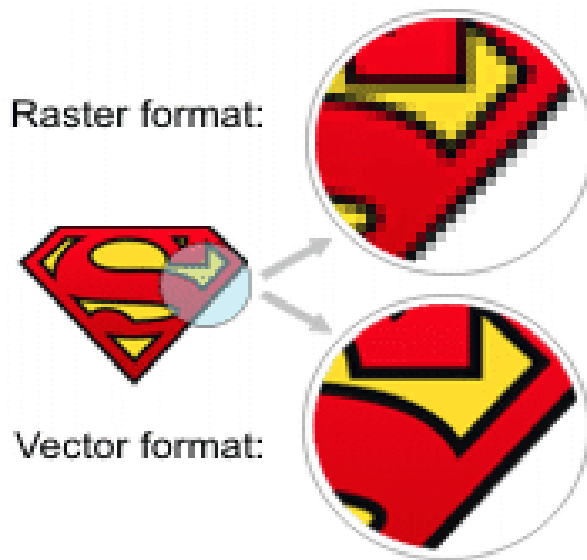


Fig: Horizontal and Vertical Process

In addition to straight image formats, Metafile formats are portable formats which can include both raster and vector information. The metafile format is an intermediate format. Most Windows applications open metafiles and then save them in their own native format.

2.4 IMAGE PROCESSING:

Digital image processing, the manipulation of images by computer, is relatively recent development in terms of man's ancient fascination with visual stimuli. In its short history, it has been applied to practically every type of images with varying degree of success. The inherent subjective appeal of pictorial displays attracts perhaps a disproportionate amount of attention from the scientists and also from the layman. Digital image processing like other glamour fields, suffers from myths, mis-connections, mis-understandings and mis-information. It is vast umbrella under which fall diverse aspect of optics, electronics, mathematics, photography graphics and computer technology. It is truly multidisciplinary endeavor ploughed with imprecise jargon.

Several factors combine to indicate a lively future for digital image processing. A major factor is the declining cost of computer equipment. Several new technological trends promise to further promote digital image processing. These include parallel processing mode practical by low cost microprocessors, and the use of charge coupled devices (CCDs) for digitizing, storage during processing and display and large low cost of image storage arrays.

FUNDAMENTAL STEPS IN DIGITAL IMAGE PROCESSING:

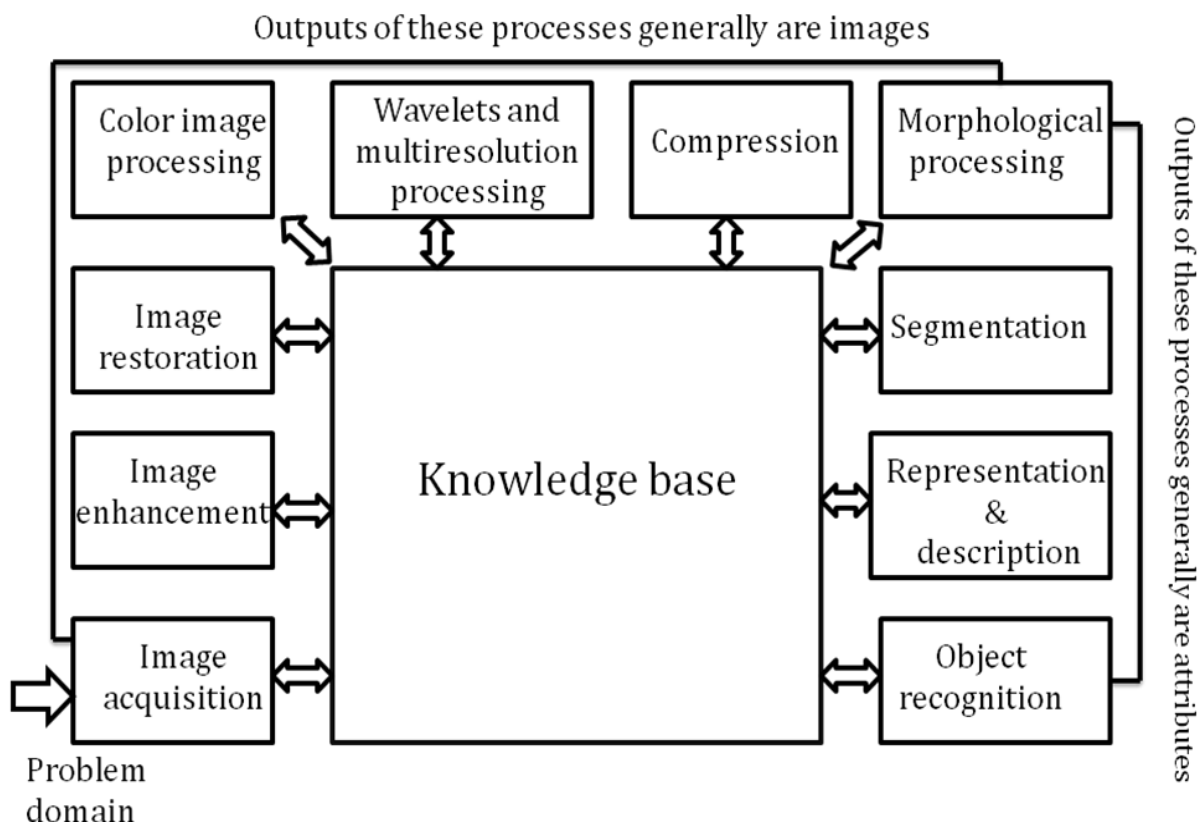


Fig: Basics steps of image Processing

2.4.1 Image Acquisition:

Image Acquisition is to acquire a digital image. To do so requires an image sensor and the capability to digitize the signal produced by the sensor. The sensor could be monochrome or color TV camera that produces an entire image of the

problem domain every $1/30$ sec. the image sensor could also be line scan camera that produces a single image line at a time. In this case, the objects motion past the line.



Fig: Digital camera

Scanner produces a two-dimensional image. If the output of the camera or other imaging sensor is not in digital form, an analog to digital converter digitizes it. The nature of the sensor and the image it produces are determined by the application.



Fig: Mobile based Camera

2.4.2 Image Enhancement:

Image enhancement is among the simplest and most appealing areas of digital image processing. Basically, the idea behind enhancement techniques is to bring out detail that is obscured, or simply to highlight certain features of interesting an image. A familiar example of enhancement is when we increase the contrast of an image because “it looks better.” It is important to keep in mind that enhancement is a very subjective area of image processing.



Fig: Image enhancement process for Gray Scale Image and Colour Image using Histogram Bits

2.4.3 Image restoration:

Image restoration is an area that also deals with improving the appearance of an image. However, unlike enhancement, which is subjective, image restoration is objective, in the sense that restoration techniques tend to be based on mathematical or probabilistic models of image degradation.

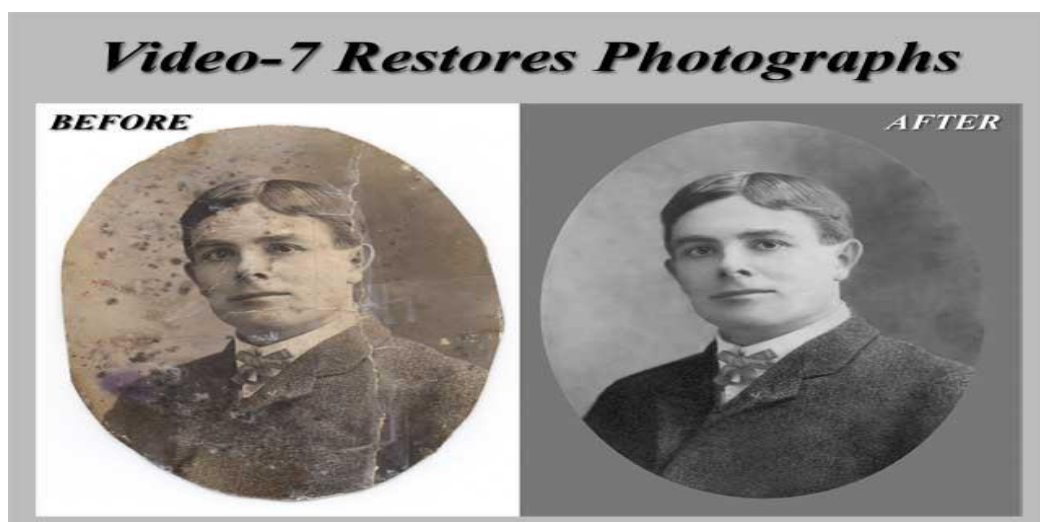


Fig: Noise image □ Image Enhancement

Enhancement, on the other hand, is based on human subjective preferences regarding what constitutes a “good” enhancement result. For example, contrast stretching is considered an enhancement technique because it is based primarily on the pleasing aspects it might present to the viewer, whereas removal of image blur by applying a deblurring function is considered a restoration technique.

2.5 Color image processing:

The use of color in image processing is motivated by two principal factors. First, color is a powerful descriptor that often simplifies object identification and extraction from a scene. Second, humans can discern thousands of color shades and intensities, compared to about only two dozen shades of gray. This second factor is particularly important in manual image analysis.



Fig: gray Scale image □ Colour Image

2.6 Segmentation:

Segmentation procedures partition an image into its constituent parts or objects. In general, autonomous segmentation is one of the most difficult tasks in digital image processing. A rugged segmentation procedure brings the process a long way toward successful solution of imaging problems that require objects to be identified individually.

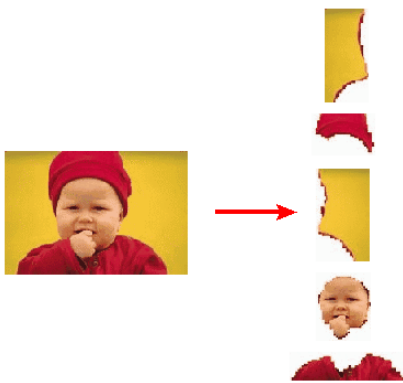


Fig: Image Segment Process

On the other hand, weak or erratic segmentation algorithms almost always guarantee eventual failure. In general, the more accurate the segmentation, the more likely recognition is to succeed.

Digital image is defined as a two dimensional function $f(x, y)$, where x and y are spatial (plane) coordinates, and the amplitude of f at any pair of coordinates (x, y) is called intensity or grey level of the image at that point. The field of digital image processing refers to processing digital images by means of a digital computer. The digital image is composed of a finite number of elements, each of which has a particular location and value. The elements are referred to as picture elements, image elements, pels, and pixels. Pixel is the term most widely used.

2.7 Image Compression

Digital Image compression addresses the problem of reducing the amount of data required to represent a digital image. The underlying basis of the reduction process is removal of redundant data. From the mathematical viewpoint, this amounts to transforming a 2D pixel array into a statically uncorrelated data set. The data redundancy is not an abstract concept but a mathematically quantifiable entity. If n_1 and n_2 denote the number of information-carrying units in two data sets that represent the same information, the relative data redundancy R_D [2] of the first data set (the one characterized by n_1) can be defined as,

$$R_D = 1 - \frac{1}{C_R}$$

Where C_R called as compression ratio [2]. It is defined as

$$C_R = \frac{n_1}{n_2}$$

In image compression, three basic data redundancies can be identified and exploited: Coding redundancy, interpixel redundancy, and psychovisual redundancy. Image compression is achieved when one or more of these redundancies are reduced or eliminated. The image compression is mainly used for image transmission and storage. Image transmission applications are in broadcast television; remote sensing via satellite, air-craft, radar, or sonar; teleconferencing; computer communications; and facsimile transmission. Image storage is required most commonly for educational and business documents, medical images that arise in computer tomography (CT), magnetic resonance imaging (MRI) and digital radiology, motion pictures, satellite images, weather maps, geological surveys, and so on.

Image Compression Model

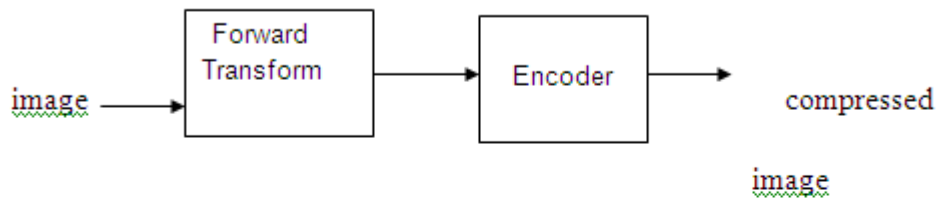
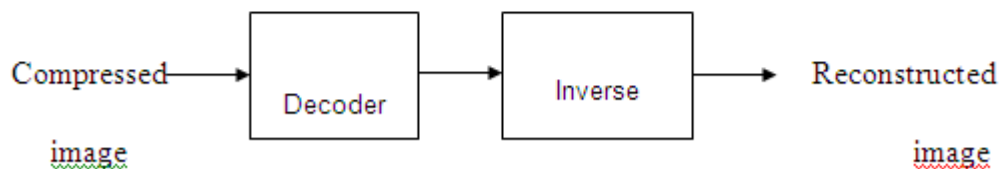


Figure 1.1.a) Block Diagram of Image compression



4

Fig:1.1b) Decompression Process for Image

2.8 CLASSIFICATION OF IMAGES:

There are 3 types of images used in Digital Image Processing. They are

1. Binary Image
2. Gray Scale Image
3. Colour Image

1.BINARY IMAGE:

A binary image is a digital image that has only two possible values for each pixel. Typically the two colors used for a binary image are black and white though any two colors can be used. The color used for the object(s) in the image is the foreground color while the rest of the image is the background color.

Binary images are also called bi-level or two-level. This means that each pixel is stored as a single bit (0 or 1). This name black and white, monochrome or monochromatic are often used for this concept, but may also designate any images that have only one sample per pixel, such as grayscale images

Binary images often arise in digital image processing as masks or as the result of certain operations such as segmentation, thresholding, and dithering. Some input/output devices, such as laser printers, fax machines, and bi-level computer displays, can only handle bi-level images

2.GRAY SCALE IMAGE

A grayscale Image is digital image is an image in which the value of each pixel is a single sample, that is, it carries only intensity information. Images of this sort, also known as black-and-white, are composed exclusively of shades of gray(0-255), varying from black(0) at the weakest intensity to white(255) at the strongest.

Grayscale images are distinct from one-bit black-and-white images, which in the context of computer imaging are images with only the two colors, black, and white (also called bi-level or binary images). Grayscale images have many shades of gray in between. Grayscale images are also called monochromatic, denoting the absence of any chromatic variation.

Grayscale images are often the result of measuring the intensity of light at each pixel in a single band of the electromagnetic spectrum (e.g. infrared, visible light, ultraviolet, etc.), and in such cases they are monochromatic proper when

only a given frequency is captured. But also they can be synthesized from a full color image; see the section about converting to grayscale.

3.COLOUR IMAGE:

A (digital) color image is a digital image that includes color information for each pixel. Each pixel has a particular value which determines its appearing color. This value is qualified by three numbers giving the decomposition of the color in the three primary colors Red, Green and Blue. Any color visible to human eye can be represented this way. The decomposition of a color in the three primary colors is quantified by a number between 0 and 255. For example, white will be coded as $R = 255, G = 255, B = 255$; black will be known as $(R,G,B) = (0,0,0)$; and say, bright pink will be : $(255,0,255)$.

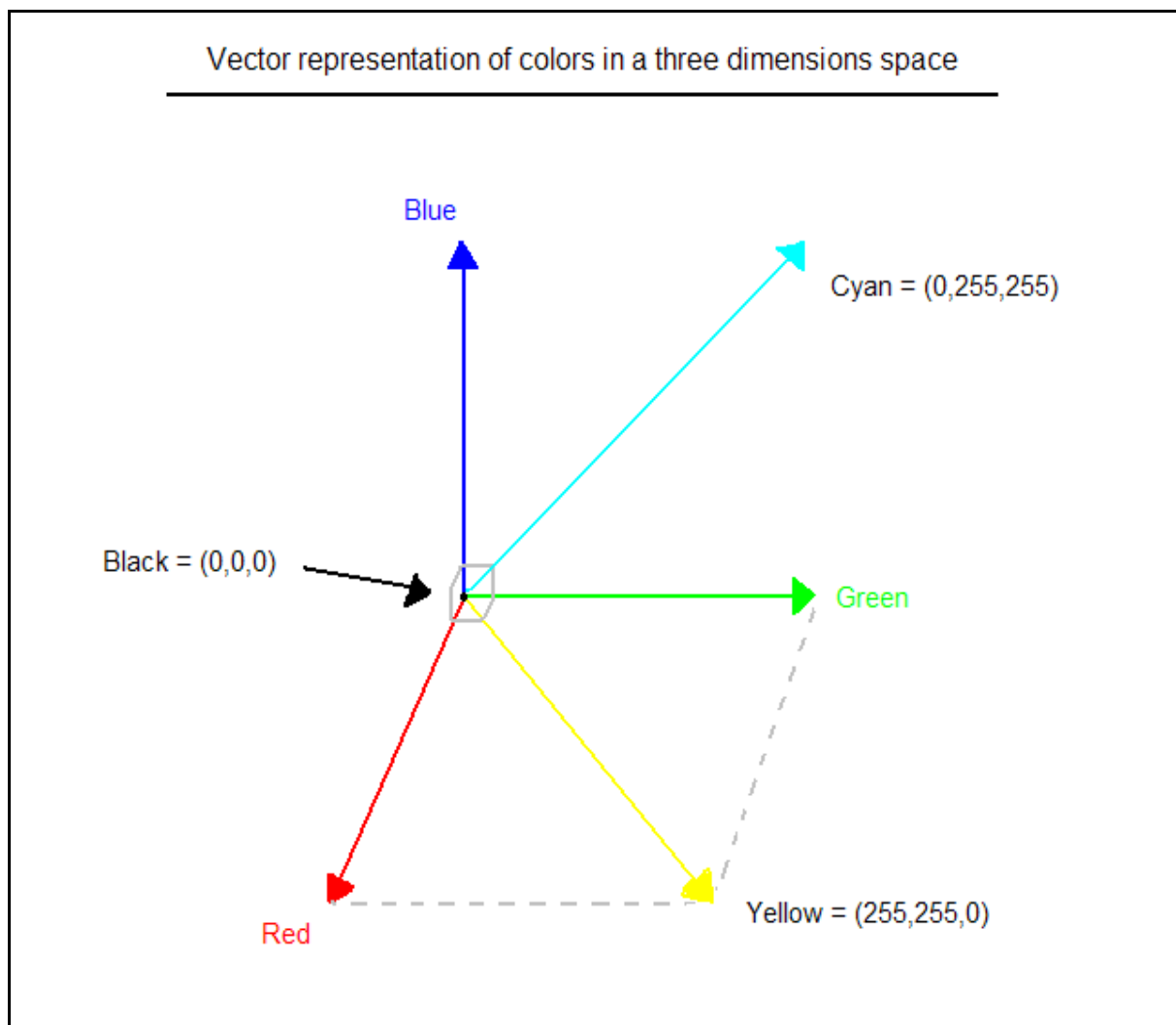


Fig.1 Hue Saturation Process of RGB SCALE Image

From the above figure, colors are coded on three bytes representing their decomposition on the three primary colors. It sounds obvious to a mathematician to immediately interpret colors as vectors in a three dimension space where each axis stands for one of the primary colors. Therefore we will benefit of most of the geometric mathematical concepts to deal with our colors, such as norms, scalar product, projection, rotation or distance.

CHAPTER 3: EXISTING SYSTEMS

3.1 Existing system

- Principal Component Analysis
- Region based segmentation
- KNN classifier

Principal Component Analysis

PCA is a mathematical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called **principal components**. The number of principal components is less than or equal to the number of original variables. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it be orthogonal to (i.e., uncorrelated with) the preceding components. Principal components are guaranteed to be independent only if the data set is jointly normally distributed. PCA is sensitive to the relative scaling of the original variables. Depending on the field of application, it is also named the discrete Karhunen–Loève transform (KLT), the Hotelling transform or proper orthogonal decomposition (POD).

Region growing methods

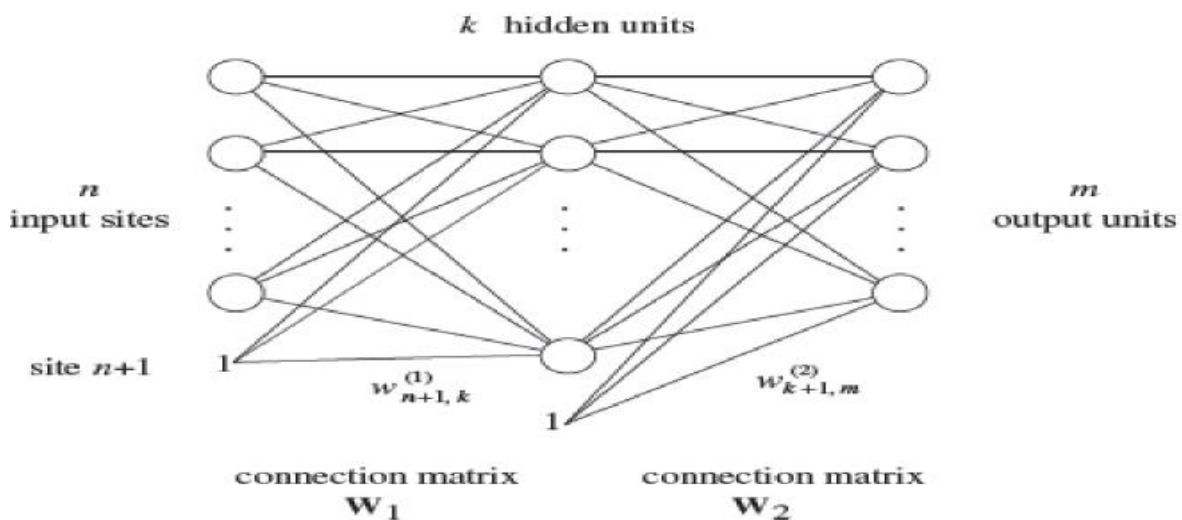
The first region growing method was the seeded region growing method. This method takes a set of seeds as input along with the image. The seeds mark each of the objects to be segmented. The regions are iteratively grown by comparing all unallocated neighbouring pixels to the regions. The difference between a pixel's intensity value and the region's mean, δ , is used as a measure of similarity. The pixel with the smallest difference measured this way is allocated to the respective region. This process continues until all pixels are allocated to a region.

Seeded region growing requires seeds as additional input. The segmentation results are dependent on the choice of seeds. Noise in the image can cause the seeds to be poorly placed. Unseeded region growing is a modified algorithm that doesn't require explicit seeds. It starts

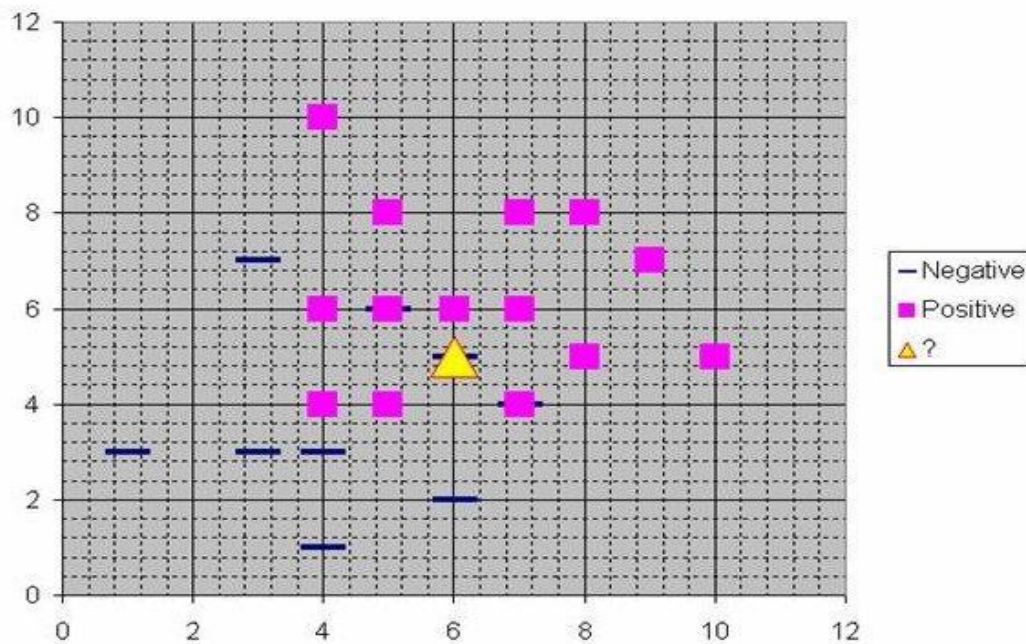
off with a single region A_1 – the pixel chosen here does not significantly influence final segmentation. At each iteration it considers the neighbouring pixels in the same way as seeded region growing. It differs from seeded region growing in that if the minimum δ is less than a predefined threshold T then it is added to the respective region A_j . If not, then the pixel is considered significantly different from all current regions A_i and a new region A_{n+1} is created with this pixel.

One variant of this technique, proposed by Haralick and Shapiro (1985), is based on pixel intensities. The mean and scatter of the region and the intensity of the candidate pixel is used to compute a test statistic. If the test statistic is sufficiently small, the pixel is added to the region, and the region's mean and scatter are recomputed. Otherwise, the pixel is rejected, and is used to form a new region.

KNN classifier



Although the implementation is very different, back propagation networks are conceptually similar to K-Nearest Neighbor (k-NN) models. The basic idea is that a predicted target value of an item is likely to be about the same as other items that have close values of the predictor variables. Consider this figure:

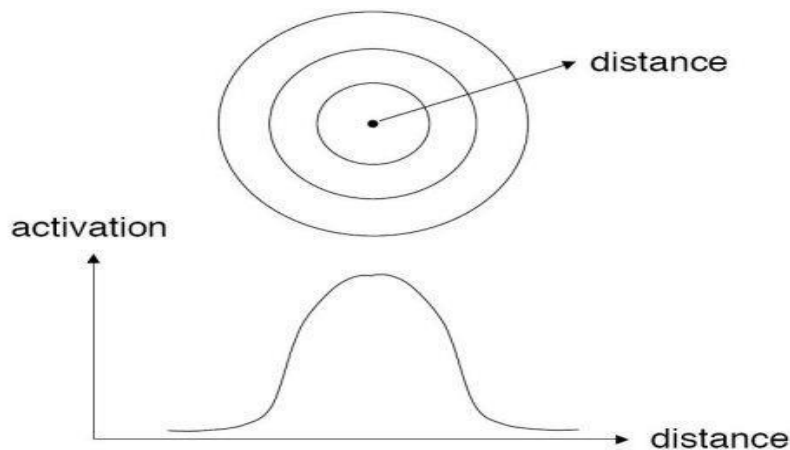


Assume that each case in the training set has two predictor variables, x and y . The cases are plotted using their x, y coordinates as shown in the figure. Also assume that the target variable has two categories, positive which is denoted by a square and negative which is denoted by a dash. Now, suppose we are trying to predict the value of a new case represented by the triangle with predictor values $x=6, y=5.1$. Should we predict the target as positive or negative.

Notice that the triangle is position almost exactly on top of a dash representing a negative value. But that dash is in a fairly unusual position compared to the other dashes which are clustered below the squares and left of center. So it could be that the underlying negative value is an odd case.

The nearest neighbor classification performed for this example depends on how many neighboring points are considered. If 1-NN is used and only the closest point is considered, then clearly the new point should be classified as negative since it is on top of a known negative point. On the other hand, if 9-NN classification is used and the closest 9 points are considered, then the effect of the surrounding 8 positive points may overbalance the close negative point.

The further some other point is from the new point, the less influence it has.



3.2 Drawbacks

- High Computational load
- Poor discriminatory power
- Less accuracy in classification

3.3 Proposed system

- Feature extraction
- convolution neural network
- pre-process

3.4 Advantages

- It is easily identify the rice leaf disease by using convolution neural network.
- more accuracy
- feature recognition is easy

3.5 System Requirements

Hardware:

- system
- 4 GB of RAM
- 500 GB of Hard disk

Software:

- python idle
- anaconda navigator
- Opencv packages
- System design

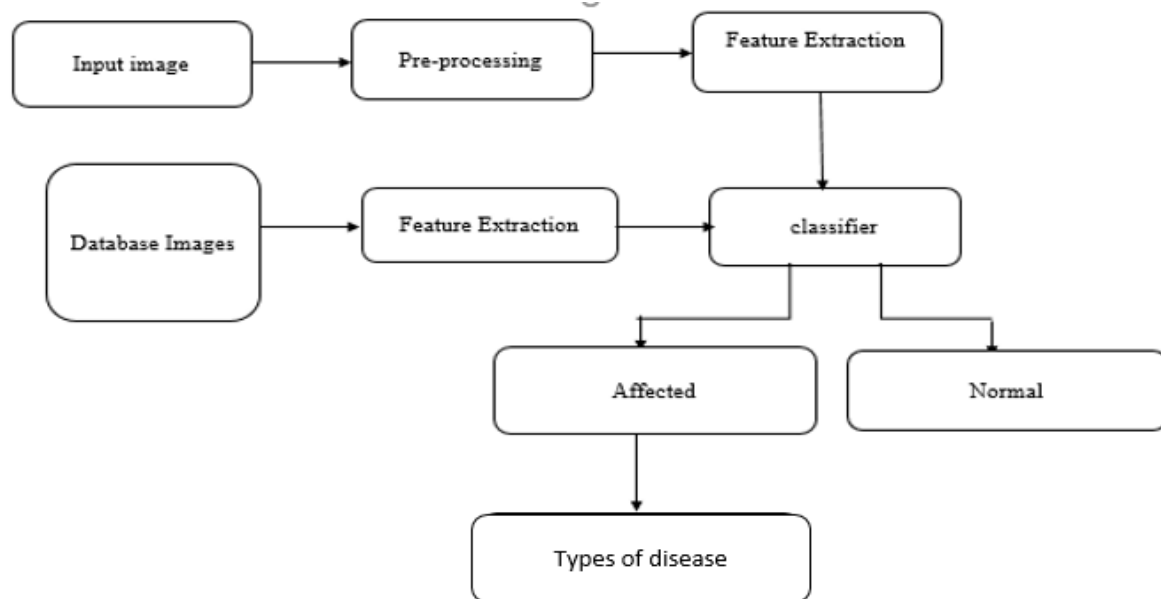
3.6 Feasibility study

Feasibility study in the sense it's a practical approach of implementing the proposed model of system . Here for a machine learning projects .we generally collect the input from online websites and filter the input data and visualize them in graphical format and then the data is divided for training and testing . That training is testing data is given to the algorithms to predict the data .

1. First, we take cervical dataset.
2. Filter dataset according to requirements and create a new dataset which has attribute according to analysis to be done
3. Perform Pre-Processing on the dataset
4. Split the data into training and testing
5. Train the model with training data then analyze testing dataset over classification algorithm
6. Finally you will get results as accuracy metrics.
7. Atlast we will creating web page using flask package with pickle python library file.

CHAPTER 4: SYSTEM DESIGN

4.1 System architecture



4.2 Modules

- IMAGE ACQUISITION
- IMPAGE PRE-PROCESSING
- DISEASE SEGMENTATION
- FEATURE EXTRACTION
- CLASSIFICATIONS

IMAGE ACQUISITION

Image Acquisition is to acquire a digital image. To do so requires an image sensor and the capability to digitize the signal produced by the sensor. The sensor could be monochrome or color TV camera that produces an entire image of the problem domain every 1/30 sec. the image sensor could also be line scan camera that produces a single image line at a time. In this case, the objects motion past the line. Scanner produces a two-dimensional image. If the output of the camera or other imaging sensor is not in digital form, an analog to digital converter digitizes it. The nature of the sensor and the image it produces are determined by the application.

IMPAGE PRE-PROCESSING

Digital image processing, the manipulation of images by computer, is relatively recent development in terms of man's ancient fascination with visual stimuli. In its short history, it has been applied to practically every type of images with varying degree of success. The inherent subjective appeal of pictorial displays attracts perhaps a disproportionate amount of attention from the scientists and also from the layman. Digital image processing like other glamour fields, suffers from myths, mis-connections, mis-understandings and mis-information. It is vast umbrella under which fall diverse aspect of optics, electronics, mathematics, photography graphics and computer technology. It is truly multidisciplinary endeavor ploughed with imprecise jargon.

Several factor combine to indicate a lively future for digital image processing. A major factor is the declining cost of computer equipment. Several new technological trends promise to further promote digital image processing. These include parallel processing mode practical by low cost microprocessors, and the use of charge coupled devices (CCDs) for digitizing, storage during processing and display and large low cost of image storage arrays.

DISEASE SEGMENTATION

Adjacent segment disease (ASD) is a broad term encompassing many complications of spinal fusion , including ,listhesis , instability, herniated nucleus pulposus, stenosis, hypertrophic facet arthritis, scoliosis, and vertebral compression fracture.

FEATURE EXTRATION

Next thing is to do Feature extraction is an attribute reduction process. Unlike feature selection, which ranks the existing attributes according to their predictive significance, feature extraction actually transforms the attributes. The transformed attributes, or features, are linear combinations of the original attributes. Finally, our models are trained using Classifier algorithm. We use classify module on Natural Language Toolkit library on Python. We use the labelled dataset gathered. The rest of our labelled data will be used to evaluate the models. Some machine learning algorithms were used to classify pre-processed data. The chosen classifiers were Random forest. These algorithms are very popular in text classification tasks.

4.3 UML Diagrams

The Unified Modeling Language (UML) is used to specify, visualize, modify, construct and document the artifacts of an object-oriented software intensive system under development. UML offers a standard way to visualize a system's architectural blueprints, including elements such as:

- actors
- business processes
- (logical) components
- activities
- programming language statements
- database schemas, and
- Reusable software components.

UML combines best techniques from data modeling (entity relationship diagrams), business modeling (work flows), object modeling, and component modeling. It can be used with all processes, throughout the software development life cycle, and across different implementation technologies. UML has synthesized the notations of the Booch method, the Object-modeling technique (OMT) and Object-oriented software engineering (OOSE) by fusing them into a single, common and widely usable modeling language. UML aims to be a standard modeling language which can model concurrent and distributed systems.

Sequence Diagram:

Sequence Diagrams Represent the objects participating the interaction horizontally and time vertically. A Use Case is a kind of behavioral classifier that represents a declaration of an offered behavior. Each use case specifies some behavior, possibly including variants that the subject can perform in collaboration with one or more actors. Use cases define the offered behavior of the subject without reference to its internal structure. These behaviors, involving interactions between the actor and the subject, may result in changes to the state of the subject and communications with its environment. A use case can include possible variations of its basic behavior, including exceptional behavior and error handling.

Activity Diagrams-:

- Activity diagrams are graphical representations of Workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

Usecase diagram:

- UML is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems.
- UML was created by Object Management Group (OMG) and UML 1.0 specification draft was proposed to the OMG in January 1997.
- OMG is continuously putting effort to make a truly industry standard.
- UML stands for Unified Modeling Language.
- UML is a pictorial language used to make software blue prints

Collaboration

A collaboration diagram resembles a flowchart that portrays the roles, functionality and behavior of individual objects as well as the overall operation of the system in real time. Objects are shown as rectangles with naming labels inside. These labels are preceded by colons and may be underlined. The relationships between the objects are shown as lines connecting the rectangles. The messages between objects are shown as arrows connecting the relevant rectangles along with labels that define the message sequencing

Class diagram

The class diagram is the main building block of object-oriented modeling. It is used for general conceptual modeling of the systematic of the application, and for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling.[1] The classes in a class diagram represent both the main elements, interactions in the application, and the classes to be programmed.

In the diagram, classes are represented with boxes that contain three compartments:

The top compartment contains the name of the class. It is printed in bold and centered, and the first letter is capitalized.

The middle compartment contains the attributes of the class. They are left-aligned and the first letter is lowercase.

The bottom compartment contains the operations the class can execute. They are also left-aligned and the first letter is lowercase.

Component diagram

Component diagram is a special kind of diagram in UML. The purpose is also different from all other diagrams discussed so far. It does not describe the functionality of the system but it describes the components used to make those functionalities.

Thus from that point of view, component diagrams are used to visualize the physical components in a system. These components are libraries, packages, files, etc.

Component diagrams can also be described as a static implementation view of a system. Static implementation represents the organization of the components at a particular moment.

A single component diagram cannot represent the entire system but a collection of diagrams is used to represent the whole.

The purpose of the component diagram can be summarized as –

- Visualize the components of a system.

- Construct executables by using forward and reverse engineering.
- Describe the organization and relationships of the components.

ER Diagram

An entity relationship diagram (ERD) shows the relationships of entity sets stored in a database. An entity in this context is an object, a component of data. An entity set is a collection of similar entities. These entities can have attributes that define its properties.

By defining the entities, their attributes, and showing the relationships between them, an ER diagram illustrates the logical structure of databases.

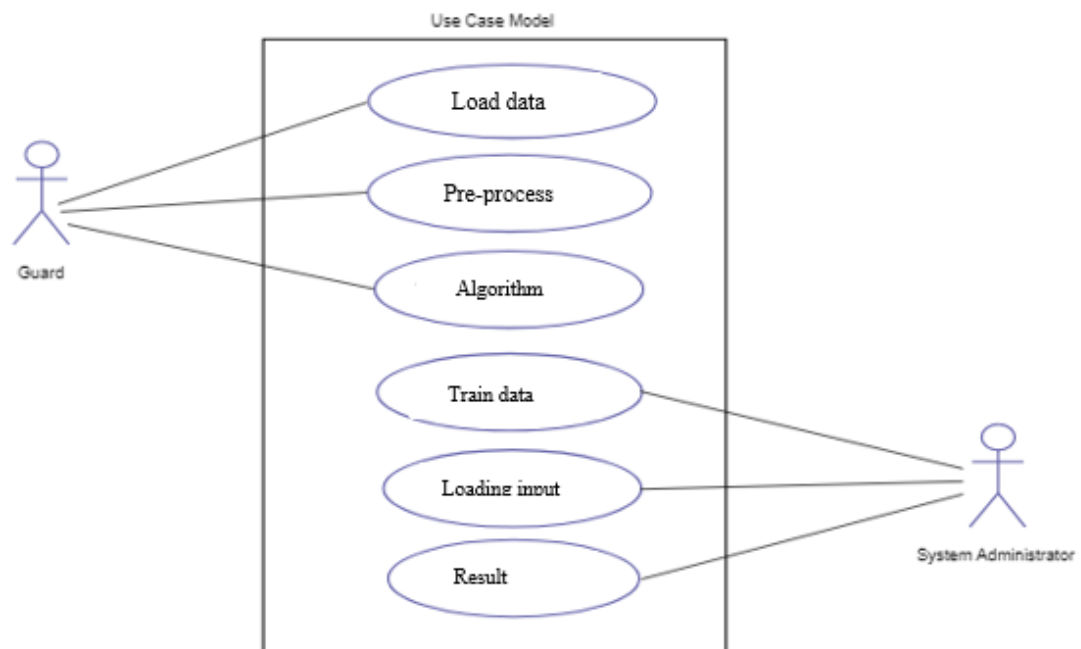
ER diagrams are used to sketch out the design of a database.

Data flow diagram

Also known as DFD, Data flow diagrams are used to graphically represent the flow of data in a business information system. DFD describes the processes that are involved in a system to transfer data from the input to the file storage and reports generation.

Data flow diagrams can be divided into logical and physical. The logical data flow diagram describes flow of data through a system to perform certain functionality of a business. The physical data flow diagram describes the implementation of the logical data flow

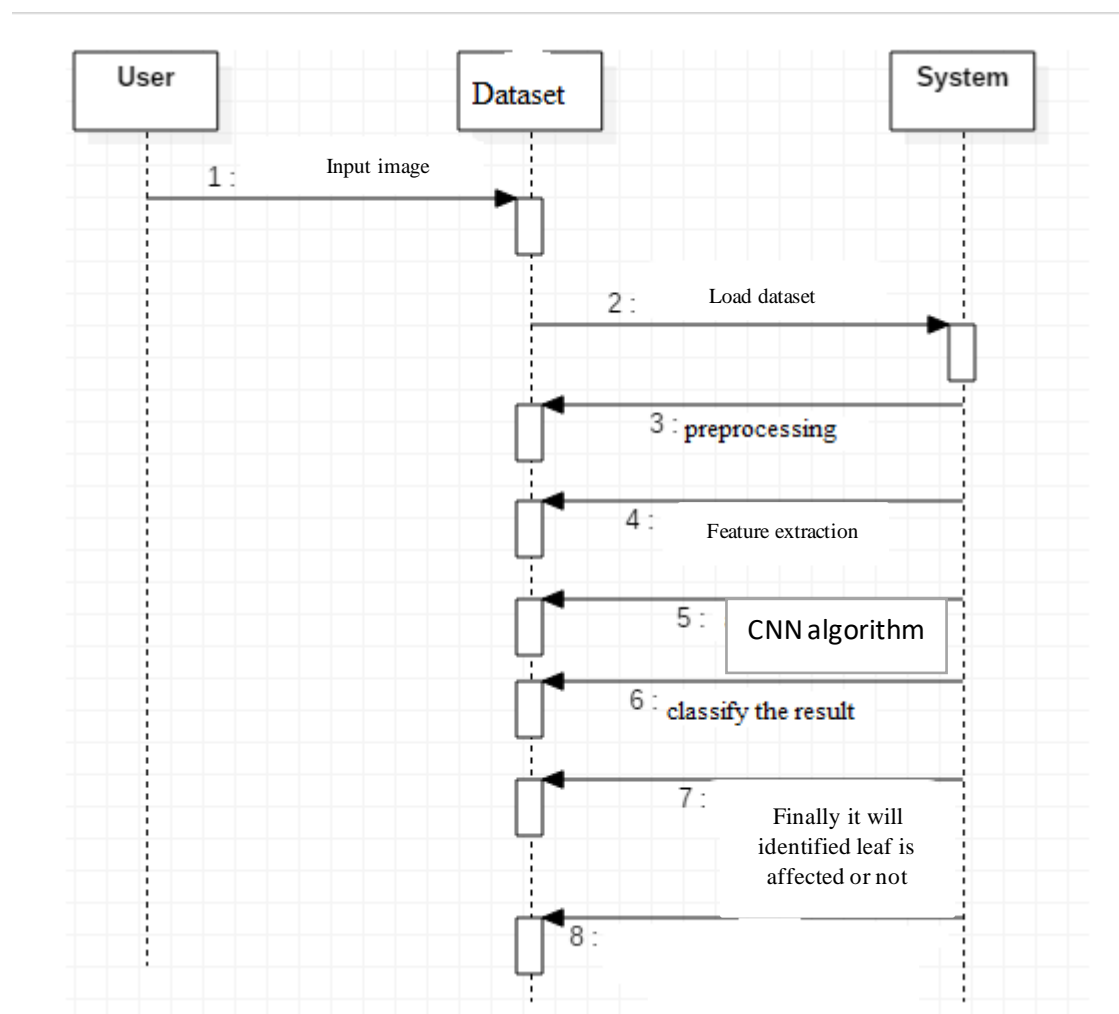
Use case Diagram:



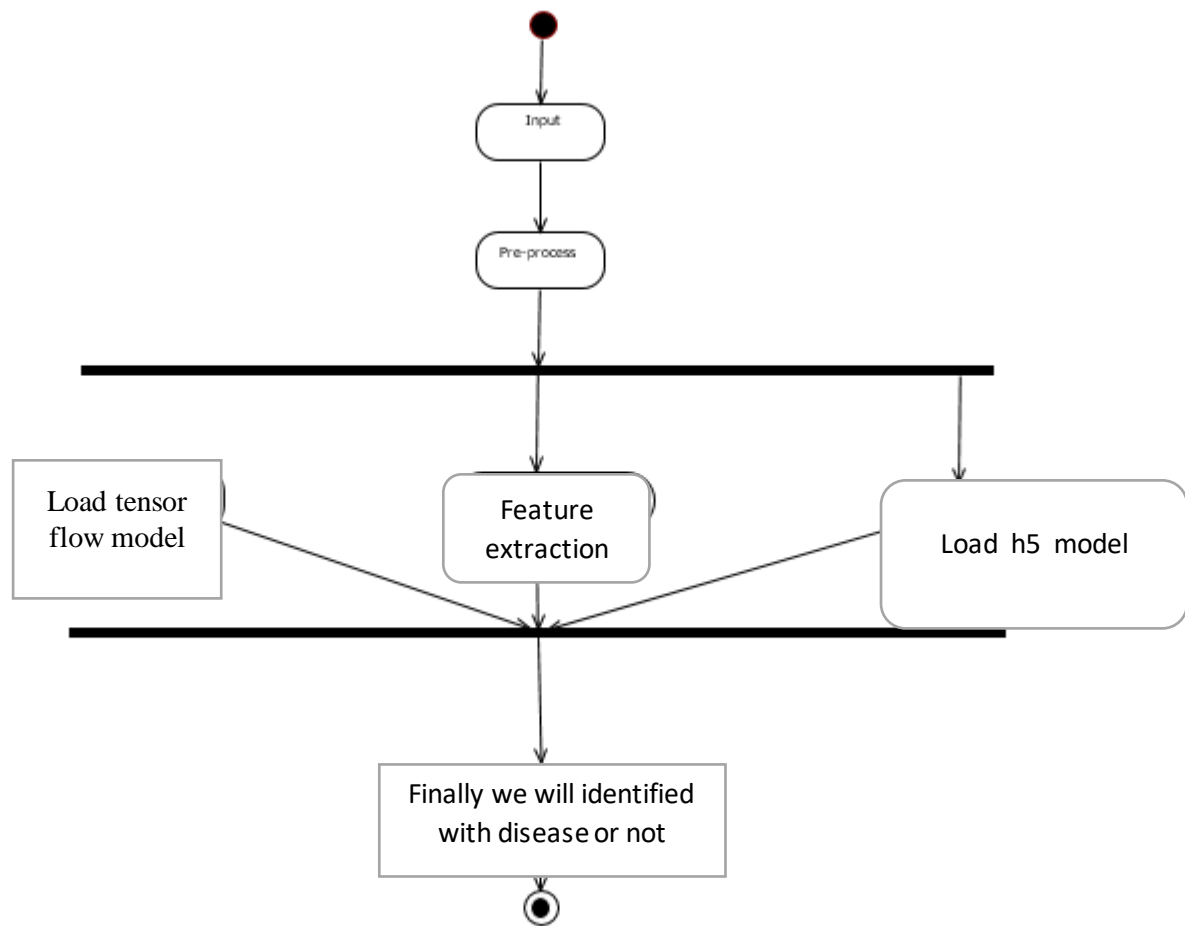
Class Diagram:



Sequence:



Activity:



CHAPTER 5: SYSTEM IMPLEMENTATION

1.1 Source code

CNN PROGRAM : Cnn_train_fin

```
# -*- coding: utf-8 -*-
from keras.models import Sequential
#initialize nn

from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
#convert pooling features space to large feature vector for fully
#connected layer
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dense
#from keras.layers import PIL
from PIL import Image

from keras.layers import BatchNormalization
from keras.layers import Dropout

#basic cnn
```

```
model=Sequential()
model.add(Conv2D(32, kernel_size=(3,3), activation='relu', input_shape=(128,128,3)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(96, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(25, activation='softmax'))

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

train_datagen = ImageDataGenerator(rescale = None,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)

training_set = train_datagen.flow_from_directory(r'C:\Users\Skkha\Desktop\leaf final\dataset\train',
                                                target_size=(128, 128),
                                                batch_size=32,
                                                class_mode='categorical')

#print(test_datagen);
labels=(training_set.class_indices)
print(labels)

test_set = test_datagen.flow_from_directory(r'C:\Users\Skkha\Desktop\leaf final\dataset\test',
                                           target_size=(128, 128),
                                           batch_size=32,
                                           class_mode='categorical')

labels2=(test_set.class_indices)
print(labels2)

model.fit_generator(training_set,
                   steps_per_epoch=375,
                   epochs=10,
                   validation_data=test_set,
                   validation_steps=125)

# Part 3 - Making new predictions

model_json=model.to_json()
```

```
with open("model1.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
    model.save_weights("model1.h5")
print("Saved model to disk")
```

f1 PROGRAM :

```
import tkinter as tk
from tkinter import filedialog
import numpy as np
from tensorflow.keras.preprocessing import image
from keras.models import Sequential
from keras.layers import Dense
from keras.models import model_from_json
import os

import cv2
from tkinter import *
from PIL import ImageTk, Image
import _tkinter # with underscore, and lowercase 't'

win=tk.Tk()

def b1_click():
    global path2
    try:
        json_file = open('model1.json', 'r')
        loaded_model_json = json_file.read()
        json_file.close()
        loaded_model = model_from_json(loaded_model_json)
        # load weights into new model
        loaded_model.load_weights("model1.h5")
        print("Loaded model from disk")

label=["Apple___Apple_scab","Apple___Black_rot","Apple___Cedar_apple_rust","Apple___Healthy"
,
        "Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot","Corn_(maize)___Common_rust",
        "Corn_(maize)___Healthy","Corn_(maize)___Northern_Leaf_Blight","Grape___Black_rot",
"Grape___Esca_(Black_Measles)","Grape___Healthy","Grape___Leaf_blight_(Isariopsis_Leaf_Spot)"
,
"Potato___Early_blight","Potato___Healthy","Potato___Late_blight","Tomato___Bacterial_spot",
"Tomato___Early_blight","Tomato___Healthy","Tomato___Late_blight","Tomato___Leaf_Mold",
        "Tomato___Septoria_leaf_spot","Tomato___Spider_mites Two-
spotted_spider_mite","Tomato___Target_Spot",
        "Tomato___Tomato_Yellow_Leaf_Curl_Virus","Tomato___Tomato_mosaic_virus"]

#lbl2=tk.Label(win,image=img)

#lbl2.pack(side = "bottom", fill = "both", expand = "yes")
#img1=(F:/py/leaf_disease_final( COMPLETE )/1.jpg)

#lbl2=tk.Label(win,image=img1)
```

```
#lb2.pack(side = "bottom", fill = "both", expand = "yes")
#loading image
path2=filedialog.askopenfilename()
print(path2)

#img = ImageTk.PhotoImage(Image.open(path2))

#lb2=tk.Label(win,image=img)
#lb2.pack(side = "bottom", fill = "both", expand = "yes")

#The Label widget is a standard Tkinter widget used to display a text or image on the screen.
#panel=tk.Label(win, image = img)
#panel.pack( fill = "both", expand = "yes")
#imr=cv2.imread(path2)
#a=cv2.imshow(imr)
#print(imr)
test_image = image.load_img(path2, target_size = (128, 128))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = loaded_model.predict(test_image)
#print(result)
#print(result)
fresult=np.max(result)
label2=label[result.argmax()]
print(label2)
#lb2.configure(image=img)
#lb2.image=img
lb1.configure(text=label2)

#lb2(ent.config(state='disabled'))
win.mainloop()

except IOError:
    pass

#button

#labelframe = LabelFrame(win, text="Leaf Disease Detection using OPENCV")
#labelframe.pack(fill="both", expand="yes")
label1 = Label(win, text="GUI For Leaf Disease Detection using OPENCV", fg='blue')
label1.pack()

b1=tk.Button(win, text='browse image',width=25,height=3,fg='red',command=b1_click)
b1.pack()
lb1 = Label(win, text="Result", fg='blue')
lb1.pack()

#image =ImageTk.PhotoImage(file='a.JPG')

#img1='1.JPG'
#lb2 = Label(win,image=image)
#lb2.pack()

#lb1.grid(column=0, row=0)
```

```
win.geometry("550x250")
win.title("Leaf Disease Detection using OPENCV")
win.bind("<Return>", b1_click)
win.mainloop()
```

prototype:

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of artificial neural network (ANN), most commonly applied to analyze visual imagery. CNNs are also known as Shift Invariant or Space Invariant Artificial Neural Networks (SIANN), based on the shared-weight architecture of the convolution kernels or filters that slide along input features and provide translation-equivariant responses known as feature maps. Counter-intuitively, most convolutional neural networks are not invariant to translation, due to the downsampling operation they apply to the input. They have applications in image and video recognition, recommender systems, image classification, image segmentation, medical image analysis, natural language processing, brain–computer interfaces, 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 "full connectivity" of these networks make them prone to overfitting data. Typical ways of regularization, or preventing overfitting, include: penalizing parameters during training (such as weight decay) or trimming connectivity (skipped connections, dropout, etc.) CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble patterns of increasing complexity using smaller and simpler patterns embossed in their filters. Therefore, on a scale of connectivity and complexity, CNNs are on the lower extreme.

Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns to optimize the filters (or kernels) through automated learning, whereas in traditional algorithms these filters are hand-engineered. This independence from prior knowledge and human intervention in feature extraction is a major advantage.

Pooling layers:

Convolutional networks may include local and/or global pooling layers along with traditional convolutional layers. Pooling layers reduce the dimensions of data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. Local pooling combines small clusters, tiling sizes such as 2×2 are commonly used. Global pooling acts on all the neurons of the feature map. There are two common types of pooling in popular use: max and average. *Max pooling* uses the maximum value of each local cluster of neurons in the feature map, while *average pooling* takes the average value.

Fully connected layers

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

Receptive field

In neural networks, each neuron receives input from some number of locations in the previous layer. In a convolutional layer, each neuron receives input from only a restricted area of the previous layer called the neuron's *receptive field*. Typically the area is a square (e.g. 5 by 5 neurons). Whereas, in a fully connected layer, the receptive field is the *entire previous layer*. Thus, in each convolutional layer, each neuron takes input from a larger area in the input than previous layers. This is due to applying the convolution over and over, which takes into account the value of a pixel, as well as its surrounding pixels. When using dilated layers, the number of pixels in the receptive field remains constant, but the field is more sparsely populated as its dimensions grow when combining the effect of several layers.

Weights[edit]

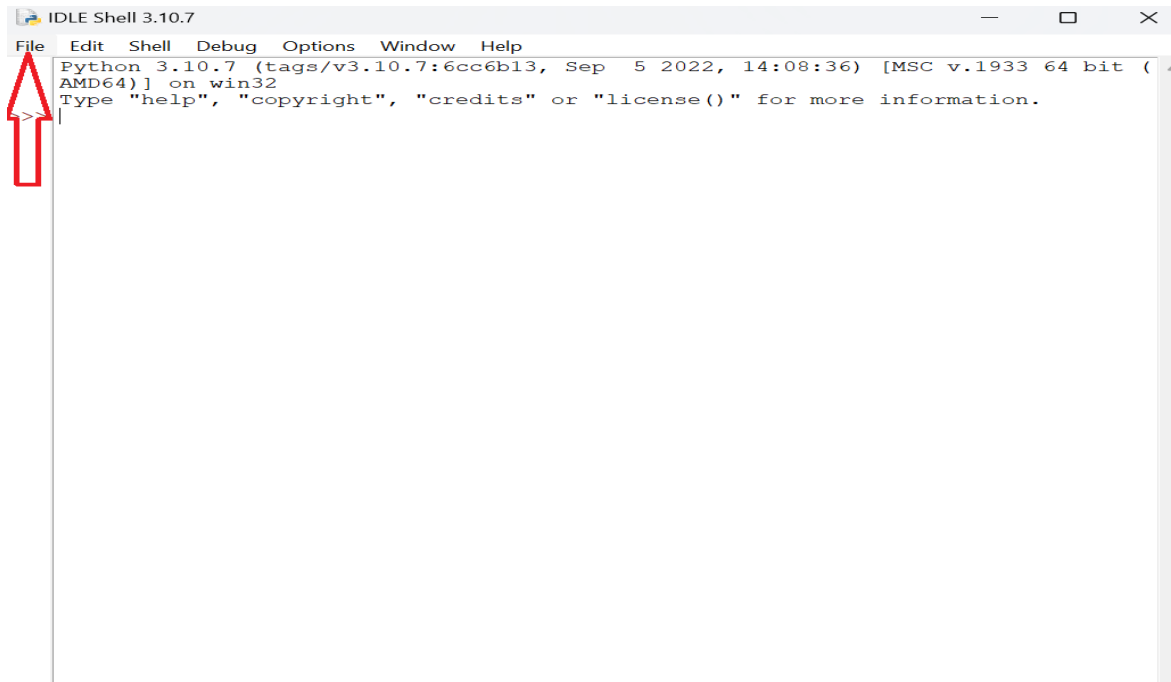
Each neuron in a neural network computes an output value by applying a specific function to the input values received from the receptive field in the previous layer. The function that is applied to the input values is determined by a vector of weights and a bias (typically real numbers). Learning consists of iteratively adjusting these biases and weights.

The vectors of weights and biases are called *filters* and represent particular features of the input (e.g., a particular shape). A distinguishing feature of CNNs is that many neurons can share the same filter. This reduces the memory footprint because a single bias and a single vector of weights are used across all receptive fields that share that filter, as opposed to each receptive field having its own bias and vector weighting.^[22]

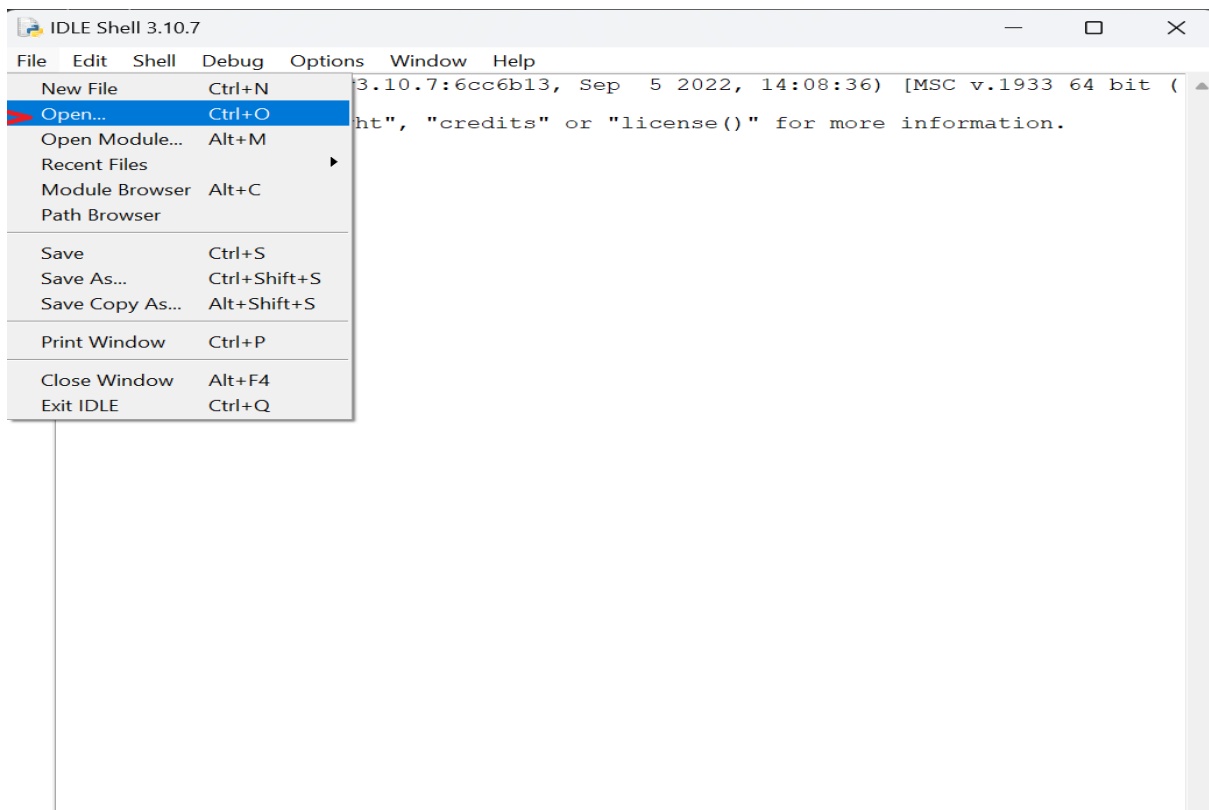
CHAPTER 6: RESULTS

6.1 Screenshots

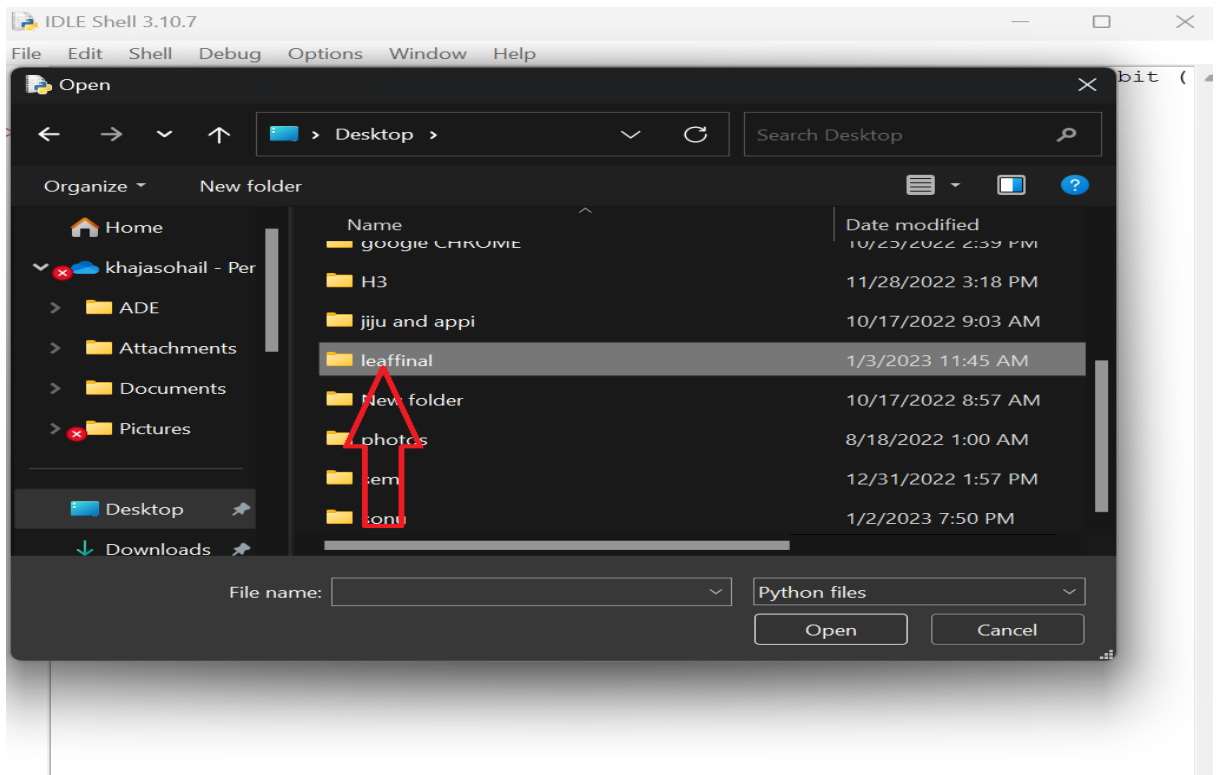
- i. **Open python IDLE shell and click on file**



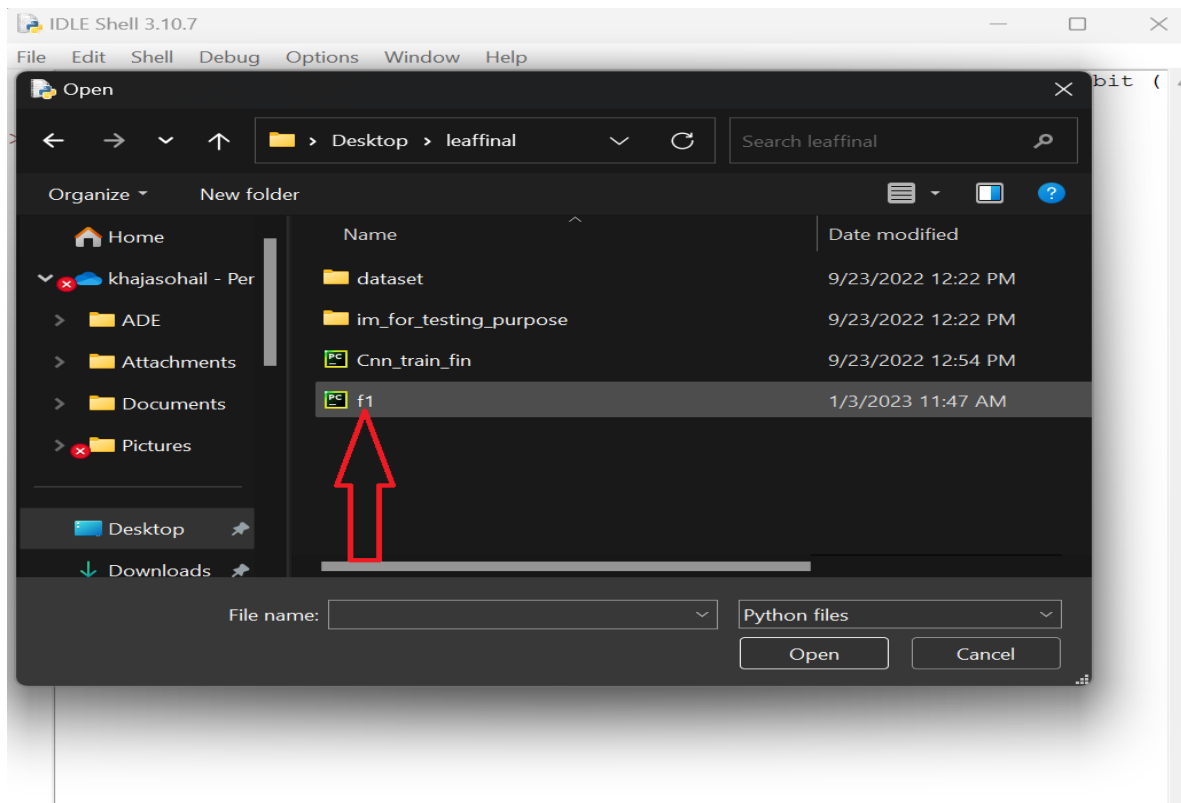
ii. **Open file location**



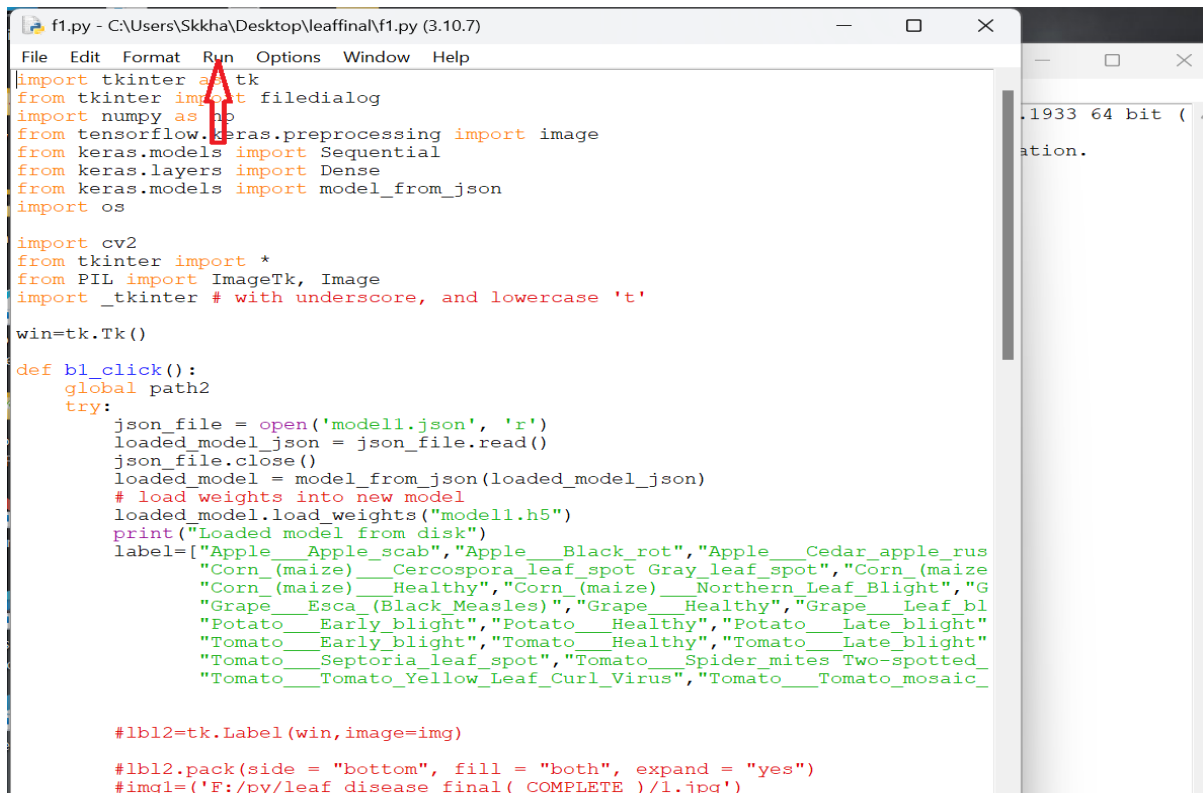
iii. **Choosing the file**



iv. **Open the required code**



v. **Run the code**



```

f1.py - C:\Users\Skkha\Desktop\leaffinal\f1.py (3.10.7)
File Edit Format Run Options Window Help
import tkinter as tk
from tkinter import filedialog
import numpy as np
from tensorflow.keras.preprocessing import image
from keras.models import Sequential
from keras.layers import Dense
from keras.models import model_from_json
import os

import cv2
from tkinter import *
from PIL import ImageTk, Image
import _tkinter # with underscore, and lowercase 't'

win=tk.Tk()

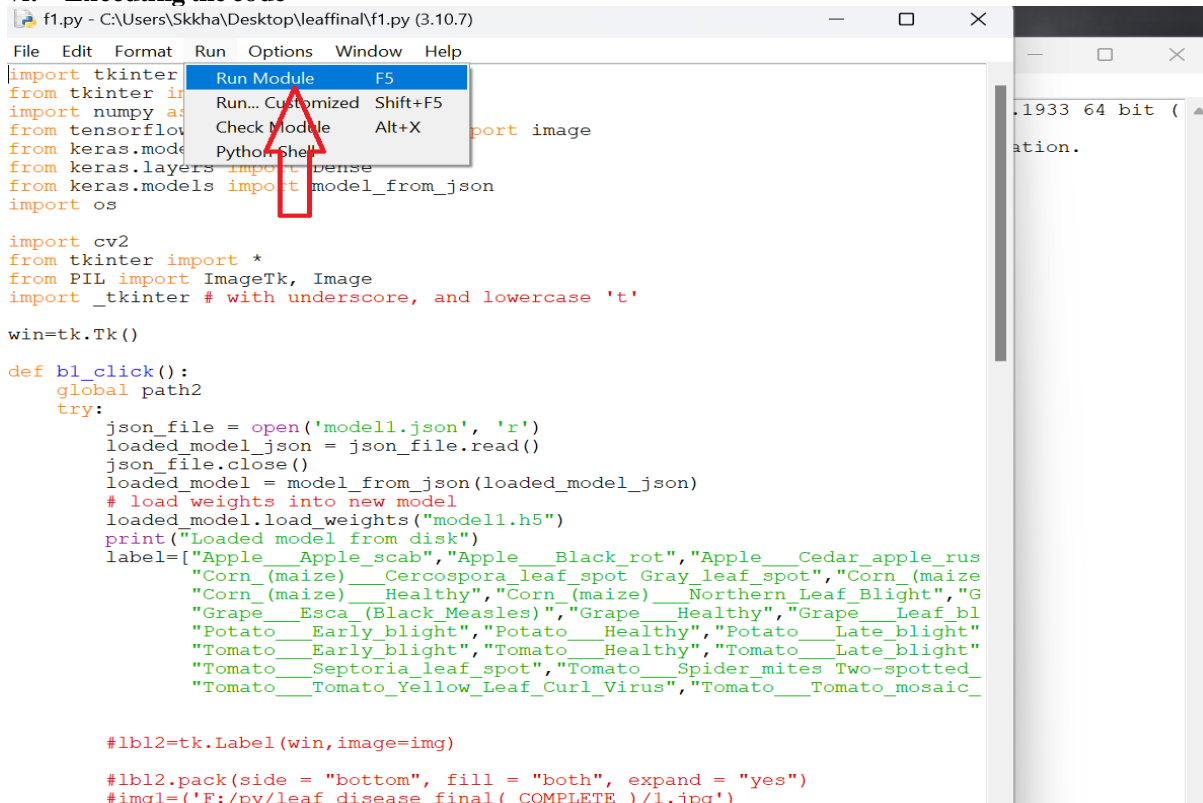
def bl_click():
    global path2
    try:
        json_file = open('modell.json', 'r')
        loaded_model_json = json_file.read()
        json_file.close()
        loaded_model = model_from_json(loaded_model_json)
        # load weights into new model
        loaded_model.load_weights("modell.h5")
        print("Loaded model from disk")
        label=["Apple__Apple_scab","Apple__Black_rot","Apple__Cedar_apple_rus
               "Corn_(maize)__Cercospora_leaf_spot_Gray_leaf_spot","Corn_(maize
               "Corn_(maize)__Healthy","Corn_(maize)__Northern_Leaf_Blight","G
               "Grape__Esca_(Black_Measles)","Grape__Healthy","Grape__Leaf_bl
               "Potato__Early_blight","Potato__Healthy","Potato__Late_blight"
               "Tomato__Early_blight","Tomato__Healthy","Tomato__Late_blight"
               "Tomato__Septoria_leaf_spot","Tomato__Spider_mites_Two-spotted_
               "Tomato__Tomato_Yellow_Leaf_Curl_Virus","Tomato__Tomato_mosaic_

        #lbl2=tk.Label(win,image=img)

        #lbl2.pack(side = "bottom", fill = "both", expand = "yes")
        #img1=('F:/py/leaf_disease_final( COMPLETE )/1.jpg')

```

vi. Executing the code



```

f1.py - C:\Users\Skkha\Desktop\leaffinal\f1.py (3.10.7)
File Edit Format Run Options Window Help
import tkinter as tk
from tkinter import filedialog
import numpy as np
from tensorflow.keras.preprocessing import image
from keras.models import Sequential
from keras.layers import Dense
from keras.models import model_from_json
import os

import cv2
from tkinter import *
from PIL import ImageTk, Image
import _tkinter # with underscore, and lowercase 't'

win=tk.Tk()

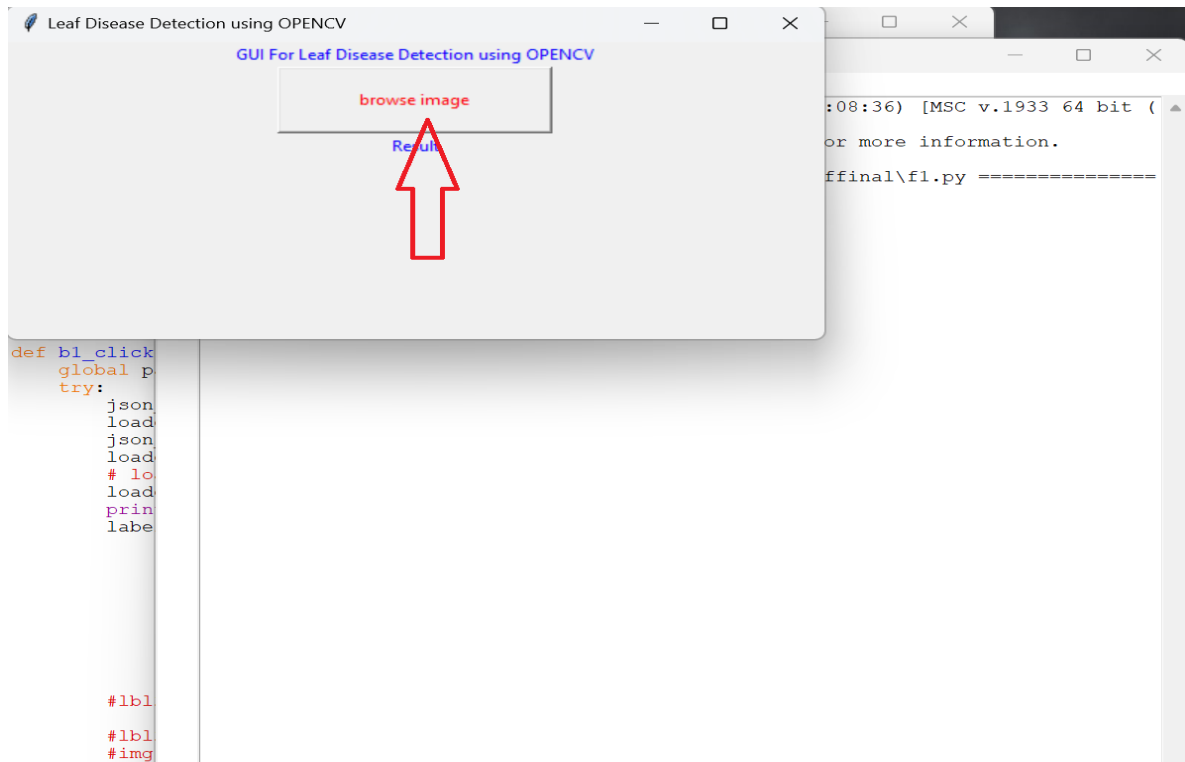
def bl_click():
    global path2
    try:
        json_file = open('modell.json', 'r')
        loaded_model_json = json_file.read()
        json_file.close()
        loaded_model = model_from_json(loaded_model_json)
        # load weights into new model
        loaded_model.load_weights("modell.h5")
        print("Loaded model from disk")
        label=["Apple__Apple_scab","Apple__Black_rot","Apple__Cedar_apple_rus
               "Corn_(maize)__Cercospora_leaf_spot_Gray_leaf_spot","Corn_(maize
               "Corn_(maize)__Healthy","Corn_(maize)__Northern_Leaf_Blight","G
               "Grape__Esca_(Black_Measles)","Grape__Healthy","Grape__Leaf_bl
               "Potato__Early_blight","Potato__Healthy","Potato__Late_blight"
               "Tomato__Early_blight","Tomato__Healthy","Tomato__Late_blight"
               "Tomato__Septoria_leaf_spot","Tomato__Spider_mites_Two-spotted_
               "Tomato__Tomato_Yellow_Leaf_Curl_Virus","Tomato__Tomato_mosaic_

        #lbl2=tk.Label(win,image=img)

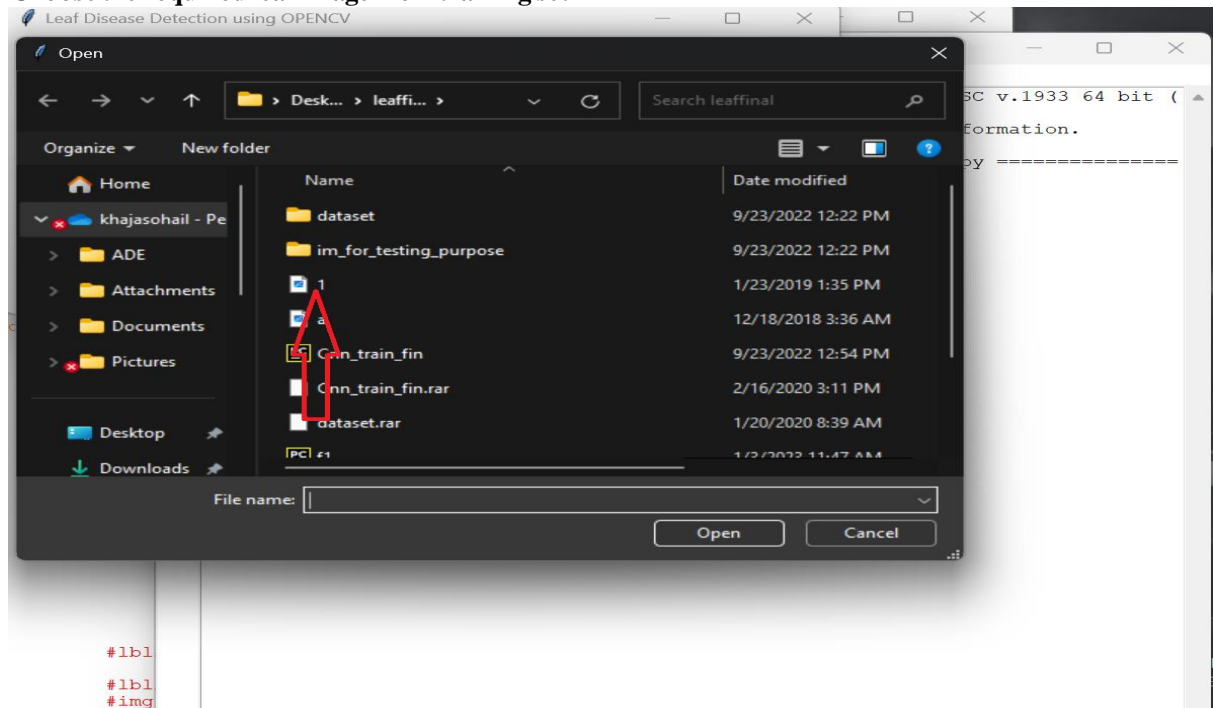
        #lbl2.pack(side = "bottom", fill = "both", expand = "yes")
        #img1=('F:/py/leaf_disease_final( COMPLETE )/1.jpg')

```

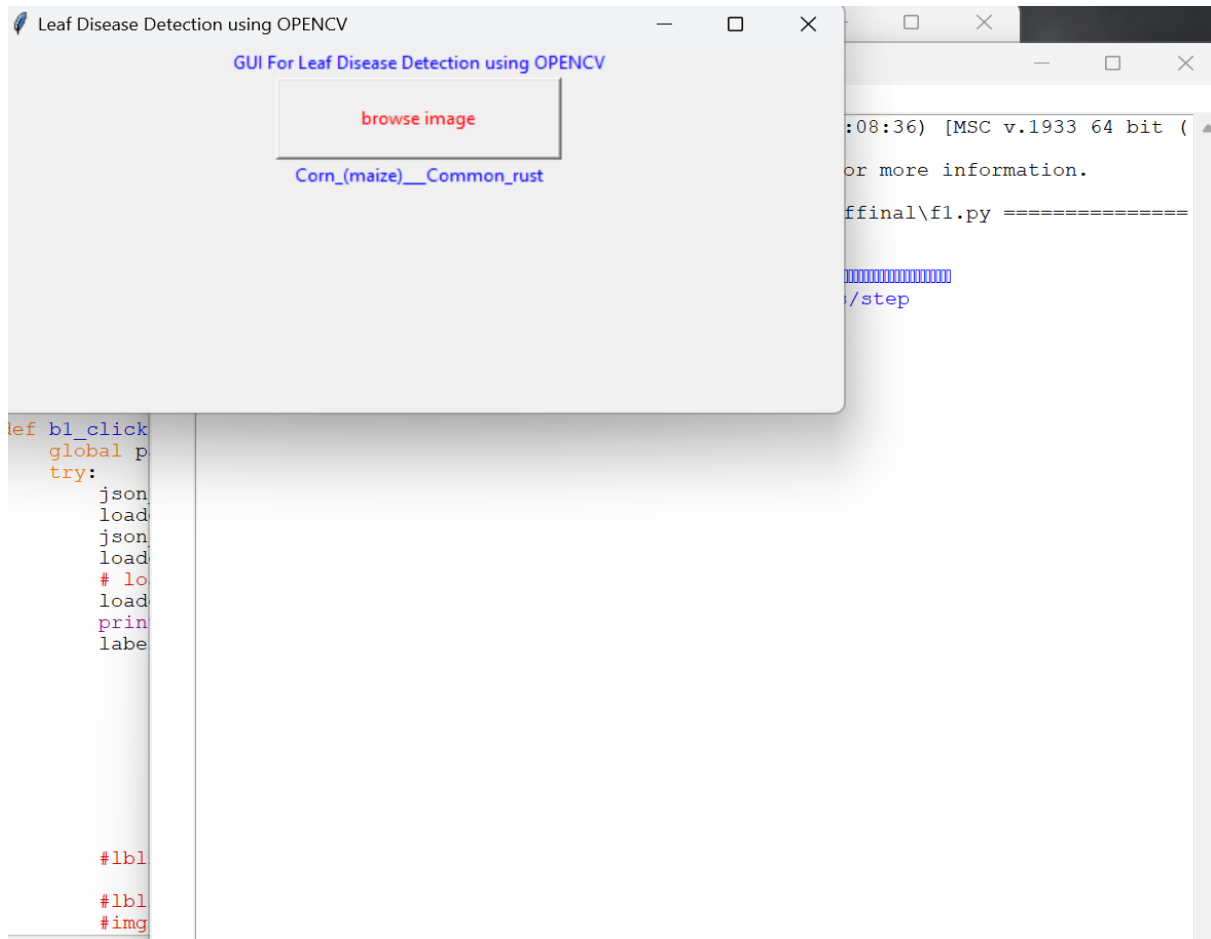
vii. Click on browse image



viii. Choose the required leaf image from training set



ix. Identified as common rust



CHAPTER 7: TESTING

7.1 Testing

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software Testing also provides an objective, independent view of the software to allow the business to appreciate and understand the risks at implementation of the software. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs.

Software Testing can also be stated as the process of validating and verifying that a software program/application/product:

- Meets the business and technical requirements that guided its design and Development.
- Works as expected and can be implemented with the same characteristics.

7.2 Design of test cases and scenarios

TESTING METHODS

- **Functional Testing**

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

- Functions: Identified functions must be exercised.
- Output: Identified classes of software outputs must be exercised.
- Systems/Procedures: system should work properly

Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

7.3 Validation

| SL# | TEST CASE NAME | DESCRIPTION | STEP NO | STEPS | ACTION TO BE TAKEN(DEESIGN STEPS) | EXPECTED(DESIGN STEP) | TEST EXECUTIONRESULT(PAS/FAIL) |
|-----|----------------|----------------------------|---------|--------|--|--|--------------------------------|
| 1. | Load models | Objective: use leaf image. | 1 | Step2: | Load data base models | Successfully loaded data base models | Pass |
| | | | 2 | Step2: | Start load image | Successfully loaded with images | Pass |
| | | | 3 | Step3: | Pre-process and feature extraction | Successfully completed pre-process and feature extraction. | Pass |
| | | | 4 | Step4: | Finally it's detecting with leaf disease | Successfully getting output with accuracy | pass |

RESULTS

The system is developed using Python language with required libraries. Implemented using three machine learning algorithm on the given dataset for mental disorder detection shows that Random forest model outperforms other models. SVM and Random forest algorithms have high accuracy compared to other Decision Tree algorithm. The above table represents the accuracy of machine learning algorithms for mental disorder detection. The below figure shows the experimental study of Decision Tree algorithm.

CHAPTER 8: CONCLUSION

8.1 Conclusion

The above Literature survey has detailed explanation of the importance of disease detection both to plants and to mankind. To have a meaningful impact of plant diseases & techniques in the area of agriculture, deliberation of proper input is necessary. Research issues addressed here are to develop a systematic approach to detect and recognize the plant diseases would assist farmers and pathologist in prospect exploration. The paper depicts the importance of image processing in agriculture field and considering the type of disease for further research work.

CHAPTER 9: FUTURE ENHANCEMENTS

9.1 Future enhancements

In future we increased the performance of this process and able to get more accuracy .

10.REFERENCES

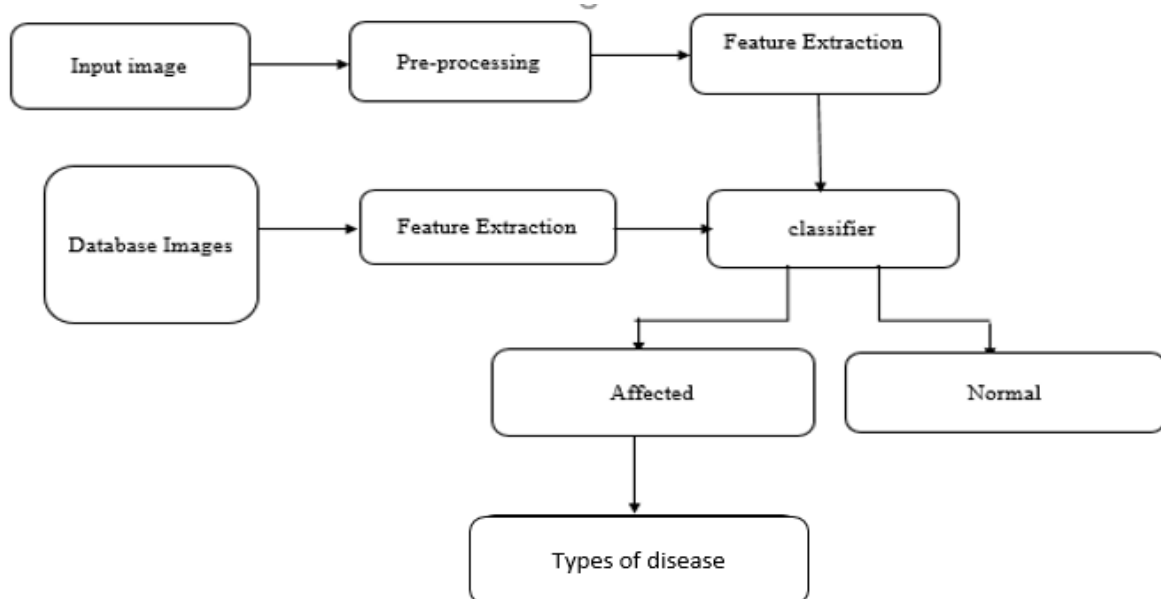
REFERENCES

- [1] “Indian agriculture economy.”. Available: [http:// statistics times.com/economy/sectorwise-gdp-Contribution-ofindia.Php](http://statistics.times.com/economy/sectorwise-gdp-Contribution-ofindia.Php)
- [2] “Common rust in maize”, Available: <https://www.pioneer.com/home/site/us/agronomy/library/common-rustin-corn/>
- [3] Indian Council of Agricultural Research”, Available: [https://www.apsnet.org/publications/imageresource/ Pages/Fi00158.aspx](https://www.apsnet.org/publications/imageresource/Pages/Fi00158.aspx)
- [4] “family of trees”, [https:// plantvillage .psu. edu/ topics/ co conut/infos](https://plantvillage.psu.edu/topics/coconut/infos)
- [5] “Agropedia”, Available:<http://agropedia.iitk.ac.in / content /papaya-rice leaf diseases-its-control>

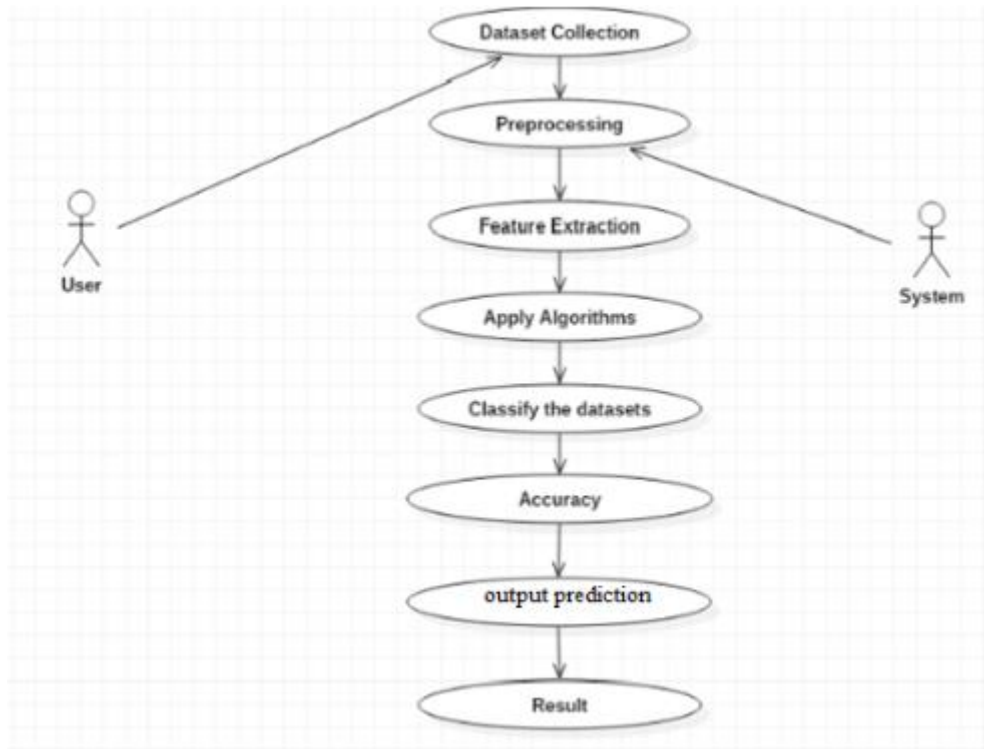
LIST OF FIGURES

Name of the Figure

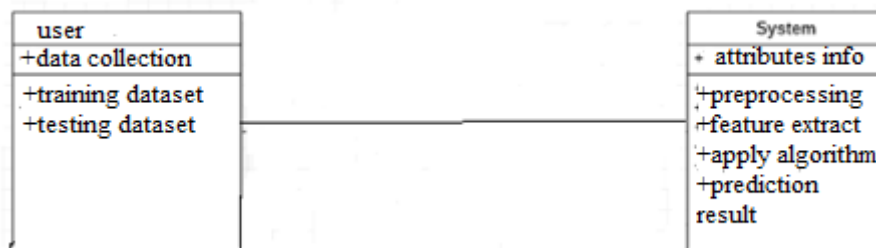
1. System architecture of the model



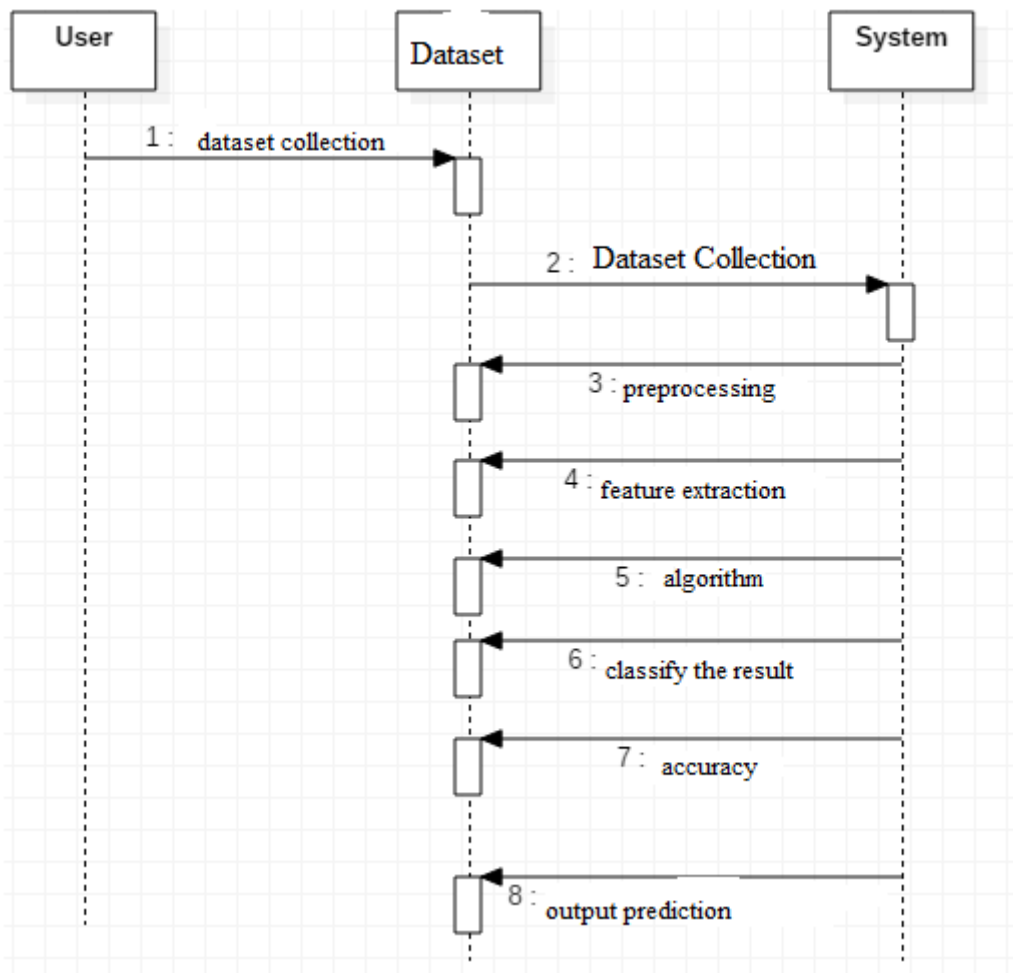
2. Use case Diagram



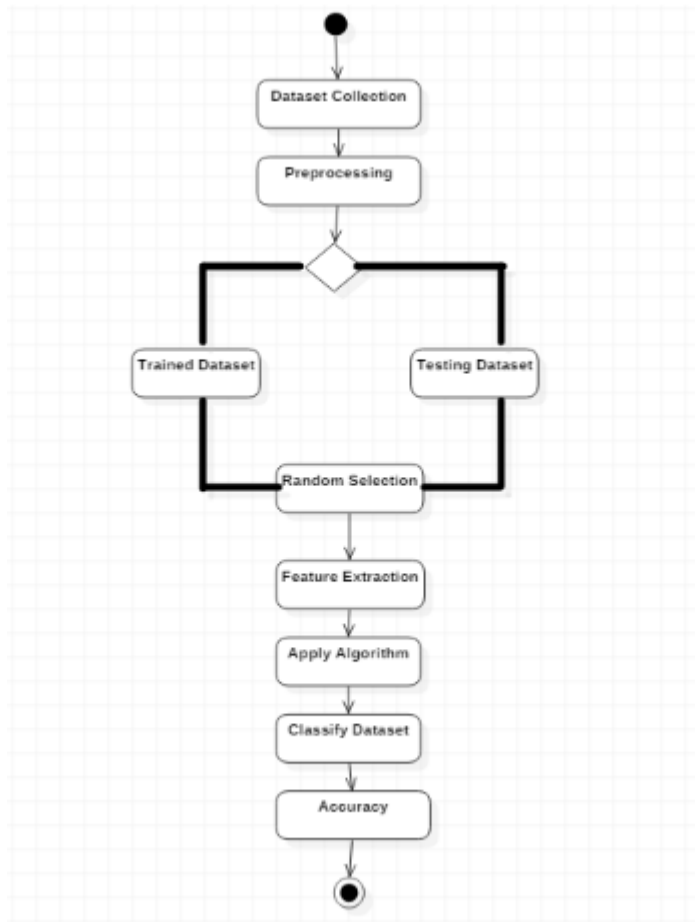
3. Class Diagram



4. Sequence Diagram



5. Activity Diagram



Domain Specification

PYTHON

Python is an object-oriented, high level language, interpreted, dynamic and multipurpose programming language.

Python is easy to learn yet powerful and versatile scripting language which makes it attractive for Application Development.

Python's syntax and dynamic typing with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas.

Python supports multiple programming pattern, including object oriented programming, imperative and functional programming or procedural styles.

Python is not intended to work on special area such as web programming. That is why it is known as multipurpose because it can be used with web, enterprise, 3D CAD etc.

We don't need to use data types to declare variable because it is dynamically typed so we can write `a=10` to declare an integer value in a variable.

Python makes the development and debugging fast because there is no compilation step included in python development and edit-test-debug cycle is very fast.

2. Python Features

1) Easy to Use:

Python is easy to very easy to use and high level language. Thus it is programmer-friendly language.

2) Expressive Language:

Python language is more expressive. The sense of expressive is the code is easily understandable.

3) Interpreted Language:

Python is an interpreted language i.e. interpreter executes the code line by line at a time. This makes debugging easy and thus suitable for beginners.

4) Cross-platform language:

Python can run equally on different platforms such as Windows, Linux, Unix , Macintosh etc. Thus, Python is a portable language.

5) Free and Open Source:

Python language is freely available(www.python.org).The source-code is also available. Therefore it is open source.

6) Object-Oriented language:

Python supports object oriented language. Concept of classes and objects comes into existence.

7) Extensible:

It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in your python code.

8) Large Standard Library:

Python has a large and broad library.

9) GUI Programming:

Graphical user interfaces can be developed using Python.

10) Integrated:

It can be easily integrated with languages like C, C++, JAVA etc.

3. Python History

- Python laid its foundation in the late 1980s.
- The implementation of Python was started in the December 1989 by Guido Van Rossum at CWI in Netherland.
- ABC programming language is said to be the predecessor of Python language which was capable of Exception Handling and interfacing with Amoeba Operating System.
- Python is influenced by programming languages like:
 - ABC language.
 - Modula-3

5. Python Applications

Python as a whole can be used in any sphere of development.

Let us see what are the major regions where Python proves to be handy.

1) Console Based Application

Python can be used to develop console based applications. For example: IPython.

2) Audio or Video based Applications

Python proves handy in multimedia section. Some of real applications are: TimPlayer, cplay etc.

3) 3D CAD Applications

Fandango is a real application which provides full features of CAD.

4) Web Applications

Python can also be used to develop web based application. Some important developments are: PythonWikiEngines, Pocoo, PythonBlogSoftware etc.

5) Enterprise Applications

Python can be used to create applications which can be used within an Enterprise or an Organization. Some real time applications are: OpenErp, Tryton, Picalo etc.

6) Applications for Images

Using Python several application can be developed for image. Applications developed are: VPython, Gogh, imgSeek etc.

There are several such applications which can be developed using Python

6. Python Example

Python code is simple and easy to run. Here is a simple Python code that will print "Welcome to Python".

A simple python example is given below.

1. `>>> a="Welcome To Python"`
2. `>>> print a`
3. `Welcome To Python`
4. `>>>`

Explanation:

- Here we are using IDLE to write the Python code. Detail explanation to run code is given in Execute Python section.

- A variable is defined named "a" which holds "Welcome To Python".
- "print" statement is used to print the content. Therefore "print a" statement will print the content of the variable. Therefore, the output "Welcome To Python" is produced.

Python 3.4 Example

In python 3.4 version, you need to add parenthesis () in a string code to print it.

1. `>>> a=("Welcome To Python Example")`
2. `>>> print a`
3. Welcome To Python Example
4. `>>>`

7. How to execute python

There are three different ways of working in Python:

1) Interactive Mode:

You can enter python in the command prompt and start working with Python.

Press Enter key and the Command Prompt will appear like:

Now you can execute your Python commands.

2) Script Mode:

Using Script Mode , you can write your Python code in a separate file using any editor of your Operating System.

Save it by .py extension.

Now open Command prompt and execute it by :

3) Using IDE: (Integrated Development Environment)

You can execute your Python code using a Graphical User Interface (GUI).

All you need to do is:

Click on Start button -> All Programs -> Python -> IDLE(Python GUI)

You can use both Interactive as well as Script mode in IDE.

1) Using Interactive mode:

Execute your Python code on the Python prompt and it will display result simultaneously.

2) Using Script Mode:

i) Click on Start button -> All Programs -> Python -> IDLE(Python GUI)

ii) Python Shell will be opened. Now click on File -> New Window.

A new Editor will be opened . Write your Python code here.

Run then code by clicking on Run in the Menu bar.

Run -> Run Module

Result will be displayed on a new Python shell as:

- Opencv:

INTRODUCTION TO COMPUTER VISION

Using software to parse the world's visual content is as big of a revolution in computing as mobile was 10 years ago, and will provide a major edge for developers and businesses to build amazing products.

Computer Vision is the process of using machines to understand and analyze imagery (both photos and videos). While these types of algorithms have been around in various forms since the 1960's, recent advances in Machine Learning, as well as leaps forward in data storage, computing capabilities, and cheap high-quality input devices, have driven major improvements in how well our software can explore this this kind of content.

What is Computer Vision?

Computer Vision is the broad parent name for any computations involving visual content – that means images, videos, icons, and anything else with pixels

involved. But within this parent idea, there are a few specific tasks that are core building blocks:

- In object classification, you train a model on a dataset of specific objects, and the model classifies new objects as belonging to one or more of your training categories.
- For object identification, your model will recognize a specific instance of an object – for example, parsing two faces in an image and tagging one as Tom Cruise and one as Katie Holmes.

A classical application of computer vision is handwriting recognition for digitizing handwritten content (we'll explore more use cases below). Outside of just recognition, other methods of analysis include:

- Video motion analysis uses computer vision to estimate the velocity of objects in a video, or the camera itself.
- In image segmentation, algorithms partition images into multiple sets of views.
- Scene reconstruction creates a 3D model of a scene inputted through images or video (check out [Selva](#)).
- In image restoration, noise such as blurring is removed from photos using Machine Learning based filters.

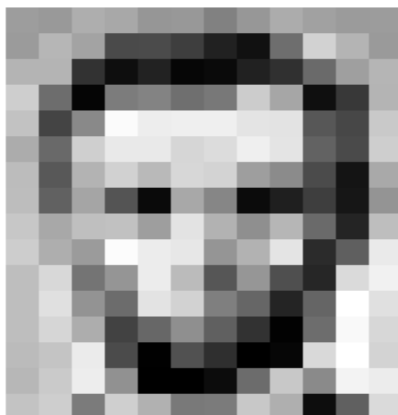
Any other application that involves understanding pixels through software can safely be labeled as computer vision.

How Computer Vision Works

One of the major open questions in both Neuroscience and Machine Learning is: how exactly do our brains work, and how can we approximate that with our own algorithms? The reality is that there are very few working and comprehensive theories of brain computation; so despite the fact that Neural Nets are supposed to “mimic the way the brain works,” nobody is quite sure if that’s actually true. Jeff Hawkins has an entire book on this topic called On Intelligence.

The same paradox holds true for computer vision – since we’re not decided on how the brain and eyes process images, it’s difficult to say how well the algorithms used in production approximate our own internal mental processes. For example, studies have shown that some functions that we thought happen in the brain of frogs actually take place in the eyes. We’re a far cry from amphibians, but similar uncertainty exists in human cognition.

Machines interpret images very simply: as a series of pixels, each with their own set of color values. Consider the simplified image below, and how grayscale values are converted into a simple array of numbers:



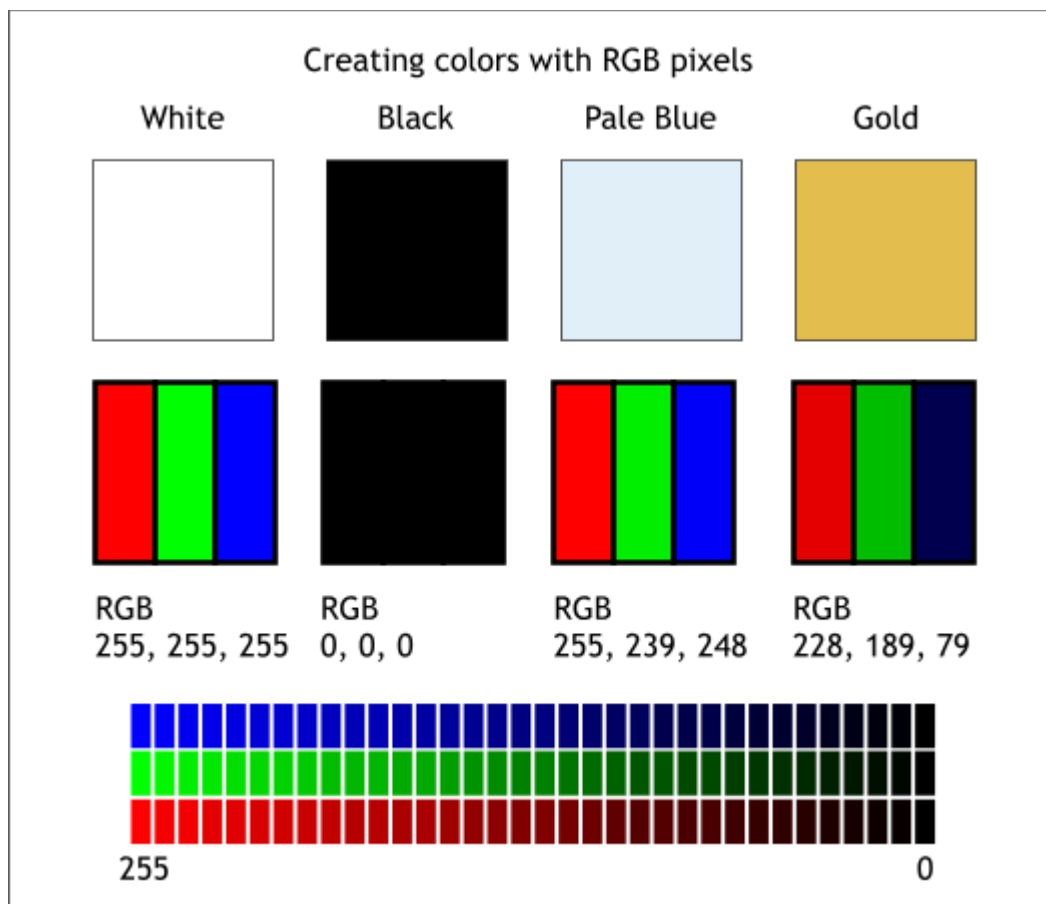
| | | | | | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 157 | 153 | 174 | 168 | 150 | 152 | 129 | 151 | 172 | 161 | 155 | 156 |
| 155 | 182 | 163 | 74 | 75 | 62 | 33 | 17 | 110 | 210 | 180 | 154 |
| 180 | 180 | 50 | 14 | 34 | 6 | 10 | 33 | 48 | 106 | 159 | 181 |
| 206 | 109 | 5 | 124 | 131 | 111 | 120 | 204 | 166 | 15 | 56 | 180 |
| 194 | 68 | 137 | 251 | 237 | 239 | 239 | 228 | 227 | 87 | 71 | 201 |
| 172 | 105 | 207 | 233 | 233 | 214 | 220 | 239 | 228 | 98 | 74 | 206 |
| 188 | 88 | 179 | 209 | 185 | 215 | 211 | 158 | 139 | 75 | 20 | 169 |
| 189 | 97 | 165 | 84 | 10 | 168 | 134 | 11 | 31 | 62 | 22 | 148 |
| 199 | 168 | 191 | 193 | 158 | 227 | 178 | 143 | 182 | 106 | 36 | 190 |
| 205 | 174 | 155 | 252 | 236 | 231 | 149 | 178 | 228 | 43 | 95 | 234 |
| 190 | 216 | 116 | 149 | 236 | 187 | 85 | 150 | 79 | 38 | 218 | 241 |
| 190 | 224 | 147 | 108 | 227 | 210 | 127 | 102 | 36 | 101 | 255 | 224 |
| 190 | 214 | 173 | 66 | 103 | 143 | 95 | 50 | 2 | 109 | 249 | 215 |
| 187 | 196 | 235 | 75 | 1 | 81 | 47 | 0 | 6 | 217 | 255 | 211 |
| 183 | 202 | 237 | 145 | 0 | 0 | 12 | 108 | 200 | 138 | 243 | 236 |
| 195 | 206 | 123 | 207 | 177 | 121 | 123 | 200 | 175 | 13 | 96 | 218 |

| | | | | | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 157 | 153 | 174 | 168 | 150 | 152 | 129 | 151 | 172 | 161 | 155 | 156 |
| 155 | 182 | 163 | 74 | 75 | 62 | 33 | 17 | 110 | 210 | 180 | 154 |
| 180 | 180 | 50 | 14 | 34 | 6 | 10 | 33 | 48 | 106 | 159 | 181 |
| 206 | 109 | 5 | 124 | 131 | 111 | 120 | 204 | 166 | 15 | 56 | 180 |
| 194 | 68 | 137 | 251 | 237 | 239 | 239 | 228 | 227 | 87 | 71 | 201 |
| 172 | 105 | 207 | 233 | 233 | 214 | 220 | 239 | 228 | 98 | 74 | 206 |
| 188 | 88 | 179 | 209 | 185 | 215 | 211 | 158 | 139 | 75 | 20 | 169 |
| 189 | 97 | 165 | 84 | 10 | 168 | 134 | 11 | 31 | 62 | 22 | 148 |
| 199 | 168 | 191 | 193 | 158 | 227 | 178 | 143 | 182 | 106 | 36 | 190 |
| 205 | 174 | 155 | 252 | 236 | 231 | 149 | 178 | 228 | 43 | 95 | 234 |
| 190 | 216 | 116 | 149 | 236 | 187 | 85 | 150 | 79 | 38 | 218 | 241 |
| 190 | 224 | 147 | 108 | 227 | 210 | 127 | 102 | 36 | 101 | 255 | 224 |
| 190 | 214 | 173 | 66 | 103 | 143 | 95 | 50 | 2 | 109 | 249 | 215 |
| 187 | 196 | 235 | 75 | 1 | 81 | 47 | 0 | 6 | 217 | 255 | 211 |
| 183 | 202 | 237 | 145 | 0 | 0 | 12 | 108 | 200 | 138 | 243 | 236 |
| 195 | 206 | 123 | 207 | 177 | 121 | 123 | 200 | 175 | 13 | 96 | 218 |

Source: Openframeworks

Think of an image as a giant grid of different squares, or pixels (this image is a very simplified version of what looks like either Abraham Lincoln or a Dementor). Each pixel in an image can be represented by a number, usually from 0 – 255. The series of numbers on the right is what software sees when you input an image. For our image, there are 12 columns and 16 rows, which means there are 192 input values for this image.

When we start to add in color, things get more complicated. Computers usually read color as a series of 3 values – red, green, and blue (RGB) – on that same 0 – 255 scale. Now, each pixel actually has 3 values for the computer to store in addition to its position. If we were to colorize President Lincoln (or Harry Potter’s worst fear), that would lead to $12 \times 16 \times 3$ values, or 576 numbers.



Source: [Xaraxone](#)

For some perspective on how computationally expensive this is, consider this tree:

- Each color value is stored in 8 bits.
- 8 bits x 3 colors per pixel = 24 bits per pixel.
- A normal sized 1024 x 768 image x 24 bits per pixel = almost 19M bits, or about 2.36 megabytes.

That's a lot of memory to require for one image, and a lot of pixels for an algorithm to iterate over. But to train a model with meaningful accuracy – especially when you're talking about Deep Learning – you'd usually need tens of thousands of images, and the more the merrier. Even if you were to use Transfer Learning to use the insights of an already trained model, you'd still need a few thousand images to train yours on.

With the sheer amount of computing power and storage required just to train deep learning models for computer vision, it's not hard to understand why advances in those two fields have driven Machine Learning forward to such a degree.

Business Use Cases for Computer Vision

Computer vision is one of the areas in Machine Learning where core concepts are already being integrated into major products that we use every day. Google is using maps to leverage their image data and identify street names, businesses, and office buildings. Facebook is using computer vision to identify people in photos, and do a number of things with that information.

But it's not just tech companies that are leverage Machine Learning for image applications. Ford, the American car manufacturer that has been around literally

since the early 1900's, is investing heavily in autonomous vehicles (AVs). Much of the underlying technology in AVs relies on analyzing the multiple video feeds coming into the car and using computer vision to analyze and pick a path of action.

Another major area where computer vision can help is in the medical field. Much of diagnosis is image processing, like reading x-rays, MRI scans, and other types of diagnostics. Google has been working with medical research teams to explore how deep learning can help medical workflows, and have made significant progress in terms of accuracy. To paraphrase from their research page:

“Collaborating closely with doctors and international healthcare systems, we developed a state-of-the-art computer vision system for reading retinal fundus images for diabetic retinopathy and determined our algorithm’s performance is on par with U.S. board-certified ophthalmologists. We’ve recently published some of our research in the Journal of the American Medical Association and summarized the highlights in a blog post.”

But aside from the groundbreaking stuff, it’s getting much easier to integrate computer vision into your own applications. A number of high-quality third party providers like Clarifai offer a simple API for tagging and understanding images, while Kairos provides functionality around facial recognition. We’ll dive into the open-source packages available for use below.

Computer Vision on Algorithmia

Algorithmia makes it easy to deploy computer vision applications as scalable microservices. Our marketplace has a few algorithms to help get the job done:

- SalNet automatically identifies the most important parts of an image
- Nudity Detection detects nudity in pictures

- Emotion Recognition parses emotions exhibited in images
- DeepStyle transfers next-level filters onto your image
- Face Recognition...recognizes faces.
- Image Memorability judges how memorable an image is.

A typical workflow for your product might involve passing images from a security camera into Emotion Recognition and raising a flag if any aggressive emotions are exhibited, or using Nudity Detection to block inappropriate profile pictures on your web application.

For a more detailed exploration of how you can use the Algorithmia platform to implement complex and useful computer vision tasks,

Computer Vision Resources

Packages and Frameworks

OpenCV – “OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Adopted all around the world, OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. Usage ranges from interactive art, to mines inspection, stitching maps on the web or through advanced robotics.”

SimpleCV – “SimpleCV is an open source framework for building computer vision applications. With it, you get access to several high-powered computer vision libraries such as OpenCV – without having to first learn about bit depths, file formats, color spaces, buffer management, eigenvalues, or matrix versus bitmap storage.”

Mahotas – “Mahotas is a computer vision and image processing library for Python. It includes many algorithms implemented in C++ for speed while operating in numpy arrays and with a very clean Python interface. Mahotas

currently has over 100 functions for image processing and computer vision and it keeps growing.

- **NUMPY:**
- NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed. This tutorial explains the basics of NumPy such as its architecture and environment. It also discusses the various array functions, types of indexing, etc. An introduction to Matplotlib is also provided. All this is explained with the help of examples for better understanding.
- **Audience**
- This tutorial has been prepared for those who want to learn about the basics and various functions of NumPy. It is specifically useful for algorithm developers. After completing this tutorial, you will find yourself at a moderate level of expertise from where you can take yourself to higher levels of expertise.
- **Prerequisites**
- You should have a basic understanding of computer programming terminologies. A basic understanding of Python and any of the programming languages is a plus.
- NumPy is a Python package. It stands for 'Numerical Python'. It is a library consisting of multidimensional array objects and a collection of routines for processing of array.

Numeric, the ancestor of NumPy, was developed by Jim Hugunin. Another package Numarray was also developed, having some additional functionalities. In 2005, Travis Oliphant created NumPy package by incorporating the features

of Numarray into Numeric package. There are many contributors to this open source project.

Operations using NumPy

Using NumPy, a developer can perform the following operations –

- Mathematical and logical operations on arrays.
- Fourier transforms and routines for shape manipulation.
- Operations related to linear algebra. NumPy has in-built functions for linear algebra and random number generation.

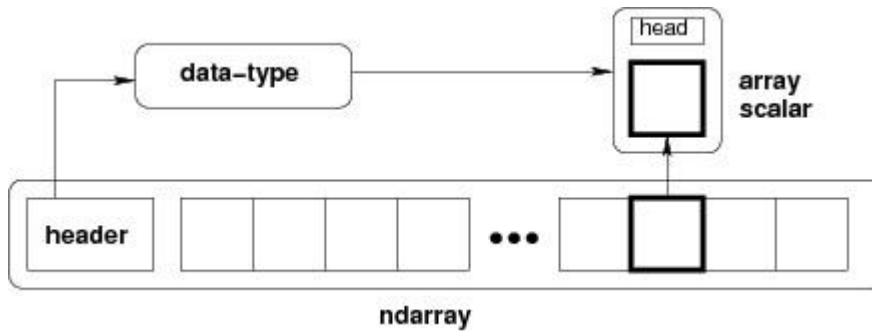
NumPy – A Replacement for MatLab

NumPy is often used along with packages like SciPy (Scientific Python) and Matplotlib (plotting library). This combination is widely used as a replacement for MatLab, a popular platform for technical computing. However, Python alternative to MatLab is now seen as a more modern and complete programming language.

The most important object defined in NumPy is an N-dimensional array type called ndarray. It describes the collection of items of the same type. Items in the collection can be accessed using a zero-based index.

Every item in an ndarray takes the same size of block in the memory. Each element in ndarray is an object of data-type object (called dtype).

Any item extracted from ndarray object (by slicing) is represented by a Python object of one of array scalar types. The following diagram shows a relationship between ndarray, data type object (dtype) and array scalar type –



An instance of ndarray class can be constructed by different array creation routines described later in the tutorial. The basic ndarray is created using an array function in NumPy as follows –

```
numpy.array
```

It creates an ndarray from any object exposing array interface, or from any method that returns an array.

Imutils:

A series of convenience functions to make basic image processing operations such as translation, rotation, resizing, skeletonization, and displaying Matplotlib images easier with OpenCV and Python.

Transalation

Translation is the shifting of an image in either the x or y direction. To translate an image in OpenCV you need to supply the (x, y)-shift, denoted as (t_x , t_y) to construct the translation matrix M:

$$M = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$$

And from there, you would need to apply the cv2.warpAffine function.

Instead of manually constructing the translation matrix M and calling `cv2.warpAffine`, you can simply make a call to the `translate` function of `imutils`

Rotation

Rotating an image in OpenCV is accomplished by making a call to `cv2.getRotationMatrix2D` and `cv2.warpAffine`. Further care has to be taken to supply the (x, y)-coordinate of the point the image is to be rotated about. These calculation calls can quickly add up and make your code bulky and less readable. The `rotate` function in `imutils` helps resolve this problem.

resizing

Resizing an image in OpenCV is accomplished by calling the `cv2.resize` function. However, special care needs to be taken to ensure that the aspect ratio is maintained. This `resize` function of `imutils` maintains the aspect ratio and provides the keyword arguments `width` and `height` so the image can be resized to the intended width/height while (1) maintaining aspect ratio and (2) ensuring the dimensions of the image do not have to be explicitly computed by the developer.

Another optional keyword argument, `inter`, can be used to specify interpolation method as well.

Skeletonization is the process of constructing the “topological skeleton” of an object in an image, where the object is presumed to be white on a black background. OpenCV does not provide a function to explicitly construct the skeleton, but does provide the morphological and binary functions to do so.

For convenience, the `skeletonize` function of `imutils` can be used to construct the topological skeleton of the image.

The first argument, `size` is the size of the structuring element kernel. An optional argument, `structuring`, can be used to control the structuring element — it defaults to `cv2.MORPH_RECT`, but can be any valid structuring element.

DISPLAYING WITH MATPLOTLIB

In the Python bindings of OpenCV, images are represented as NumPy arrays in BGR order. This works fine when using the `cv2.imshow` function. However, if you intend on using Matplotlib, the `plt.imshow` function assumes the image is in RGB order. A simple call to `cv2.cvtColor` will resolve this problem, or you can use the `opencv2matplotlib` convenience function.

History of Python

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, Unix shell, and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

Python Features

Python's features include:

Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.

Easy-to-read: Python code is more clearly defined and visible to the eyes.

Easy-to-maintain: Python's source code is fairly easy-to-maintain.

A broad standard library: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.

Interactive Mode: Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.

Portable: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.

Extendable: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.

Databases: Python provides interfaces to all major commercial databases.

GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries, and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

Scalable: Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below:

- IT supports functional and structured programming methods as well as OOP.

- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- IT supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Python is available on a wide variety of platforms including Linux and Mac OS X. Let's understand how to set up our Python environment.

ANACONDA NAVIGATOR

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows you to launch applications and easily manage conda packages, environments and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository. It is available for Windows, mac OS and Linux.

Why use Navigator?

In order to run, many scientific packages depend on specific versions of other packages. Data scientists often use multiple versions of many packages, and use multiple environments to separate these different versions. The command line program conda is both a package manager and an environment manager, to help data scientists ensure that each version of each package has all the dependencies it requires and works correctly.

Navigator is an easy, point-and-click way to work with packages and environments without needing to type conda commands in a terminal window. You can use it to find the packages you want, install them in an environment, run the packages and update them, all inside Navigator.

WHAT APPLICATIONS CAN I ACCESS USING NAVIGATOR?

The following applications are available by default in Navigator:

- Jupyter Lab
- Jupyter Notebook
- QT Console
- Spyder
- VS Code
- Glue viz
- Orange 3 App
- Rodeo
- RStudio

Advanced conda users can also build your own Navigator applications

How can I run code with Navigator?

The simplest way is with Spyder. From the Navigator Home tab, click Spyder, and write and execute your code.

You can also use Jupyter Notebooks the same way. Jupyter Notebooks are an increasingly popular system that combine your code, descriptive text, output, images and interactive interfaces into a single notebook file that is edited, viewed and used in a web browser.

What's new in 1.9?

- Add support for **Offline Mode** for all environment related actions.
- Add support for custom configuration of main windows links.

Numerous bug fixes and performance enhancements.

