**CHAPTER 1**

**INTRODUCTION**

**1.1 INTRODUCTION**

Lung cancer is one of the leading causes of cancer-related deaths worldwide. Early detection significantly increases the chances of successful treatment and improves survival rates. Among various diagnostic methods, ComputedTomography(CT) scans are widely used due to their high resolution and ability to reveal detailed structures within the lungs.

However, manual analysis of CT images by radiologists can be time-consuming, subjective, and prone to error—especially in early-stage cases where signs may be subtle. To overcome these limitations, computer-aided diagnosis (CAD) systems are increasingly being developed to assist in the automatic detection of lung cancer.

In this project, we propose an efficient method for lung cancer detection using SupportVectorMachine(SVM), a powerful supervised machine learning algorithm. The system uses CT images as input, processes them through several stages including preprocessing, feature extraction, and classification. Techniques like HistogramofOrientedGradients **(HOG)** or **Local Binary Pattern (LBP)** are applied to extract meaningful features from the images, which are then used to train the SVM classifier.

By combining medical imaging with machine learning, this approach aims to provide a fast, accurate, and cost-effective tool for early lung cancer diagnosis, thereby supporting clinical decision-making and improving patient outcomes.

* 1. **PURPOSE**

The project aims to develop an efficient, accurate, and accessible lung cancer detection system using Support Vector Machine (SVM) algorithms on CT scan images. It enhances early diagnosis, supports clinical decision-making, and reduces diagnostic errors. By applying image processing and machine learning techniques like HOG and GLCM, the system automates cancer classification, increases scalability, and ensures cost-effective healthcare solutions. It promotes AI integration in diagnostics, encourages research with open datasets, and contributes to building a patient-centric, AI-assisted healthcare ecosystem.

## MOTIVATION

## Lung cancer is a leading cause of cancer-related deaths, mainly due to late detection. Although CT scans are effective for diagnosis, manual analysis by radiologists can be slow, subjective, and error-prone. This project is motivated

## by the need for an intelligent, automated system that can assist in early and

## accurate detection of lung cancer.

## By using Support Vector Machine (SVM) and image processing techniques, the goal is to develop a tool that improves diagnostic efficiency, reduces human error, and supports medical professionals in delivering faster, more reliable diagnoses. It also aims to make lung cancer detection more accessible, especially in resource-limited settings.

* 1. **PROBLEM DEFINITION**

Lung cancer often goes undetected until advanced stages due to the lack of visible symptoms and the complexity of interpreting CT scan images. Manual diagnosis by radiologists is time-consuming, subjective, and may lead to inconsistent or delayed results, especially in early-stage cases where cancerous patterns are subtle.

The problem this project addresses is the need for an automated, accurate, and efficient system to detect lung cancer from CT scan images. By leveraging machine learning—specifically the Support Vector Machine (SVM) algorithm—and feature extraction techniques, the project aims to classify lung CT images as cancerous or non-cancerous, thereby supporting early diagnosis, reducing human error, and improving healthcare accessibility.

## 1.5 OBJECTIVES OF THE PROJECT

* **Early Detection:** Identify lung cancer in its early stages using CT scan analysis.
* **High Accuracy:** Use SVM for precise classification of cancerous vs non-cancerous tissues.
* **Clinical Support:** Assist radiologists with a reliable diagnostic tool.
* **Feature Extraction:** Extract key image features using methods like HOG or GLCM.
* **Model Optimization:** Improve accuracy through data-driven tuning.
* **Efficiency & Accessibility:** Deliver fast, scalable, and cost-effective diagnosis.
* **Support Research:** Leverage open datasets and contribute to AI in medical imaging.
* **AI Integration:** Show how machine learning enhances healthcare systems.
* **Standardization:** Provide consistent and reproducible diagnostic results.
* **Patient-Centric Care:** Enable timely interventions and better treatment planning.

**CHAPTER 2**

**SYSTEM ANALYSIS**

**2.1 EXIXTING SYSTEM**

The existing system for lung cancer detection relies heavily on manual analysis of CT scans by radiologists. This process is often time-consuming, subjective, and varies based on individual expertise. Early-stage cancer signs can be subtle and may be overlooked, leading to misdiagnosis. Human error is a significant concern, and the lack of automation limits diagnostic consistency. Additionally, the absence of intelligent tools increases the workload on medical professionals and can delay timely treatment, ultimately affecting patient outcomes.

## 2.2 PROPOSED SYSTEM

## The proposed system introduces an automated approach to lung cancer detection using Support Vector Machine (SVM) and image processing techniques. It processes CT scan images through stages like preprocessing, feature extraction, and classification to identify cancerous tissues accurately. Techniques such as HOG and GLCM are used to extract relevant features from the images. The SVM model then classifies the input as cancerous or non-cancerous with high sensitivity and specificity. This reduces the dependency on manual diagnosis and minimizes human error. The system delivers faster results, supporting timely medical intervention. It also eases the diagnostic burden on radiologists and ensures consistent outcomes. Designed for scalability, it can be deployed in both urban hospitals and rural clinics. It is cost-effective and aims to make early cancer detection more accessible. Overall, the system empowers healthcare professionals with intelligent tools for better patient care.

**2.3 FUNCTIONAL REQUIREMENTS**

**HARDWARE REQUIREMNETS:**

|  |  |
| --- | --- |
| **COMPONENTS** | **SPECIFICATION** |
| PROCESSOR | Multi-core processor (Quad-core or higher recommended) |
| RAM | 4 GB (8 GB or more recommended for smoother performance). |
| STORAGE | Minimum 16 GB (32 GB or more recommended) |

**SOFTWARE REQUIREMENTS:**

|  |  |
| --- | --- |
| **COMPONENT** | **SPECIFICATION** |
| OPERATING SYSTEM | WINDOWS 10 |
| LANGUAGE | Python |
| FRAME WORK | Django,Flask |
| USER INTERFACE DESIGN | HTML,JavaScript, CSS |

1. **User Registration and Authentication**
   1. Users (radiologists, medical professionals, administrators) can register and create accounts.
   2. Secure login with multi-factor authentication (MFA).
   3. User role management (radiologist, medical professional, admin).
2. **User Profiles**
   1. Radiologists can create and manage profiles to store personal details and access specific project datasets.
   2. Medical professionals can set up profiles to track patient data and classification results.
   3. Admin users have the ability to oversee system management and user accounts.
3. **CT Image Upload and Management**
   1. Users can upload CT scan images (in DICOM, JPEG, PNG formats) for processing and classification.
   2. Images are organized by patient or session for easy access and tracking.
   3. The system allows users to view uploaded images and related metadata.
4. **Lung Cancer Detection**
   1. The system uses SVM models to process and classify CT images for the presence of lung cancer.
   2. The output of the classification is a **binary result**: cancerous or non-cancerous, with a confidence score.
   3. Provide a visual representation of regions of interest (ROIs) where the model detects potential cancer areas.
5. **Real-Time Model Inference**
   1. The SVM classifier processes CT images in real-time or near real-time.
   2. Display model results immediately after the image is processed, with predictions and confidence levels.
   3. Allow users to save and track previous classifications for comparison.
6. **Results and Reporting**
   1. Provide detailed reports for each CT scan classification, including:
      1. Classification result (cancerous/non-cancerous).
      2. Confidence score and model interpretation.
      3. Visual markers on the CT image where cancerous regions are detected.
   2. Enable export of results to PDF, Excel, or other formats for easy sharing.

**CHAPTER 3**

**SOFTWARE ENVIRONMENT**

The lung cancer detection system is built using a modern tech stack that supports machine learning, web development, and image processing:

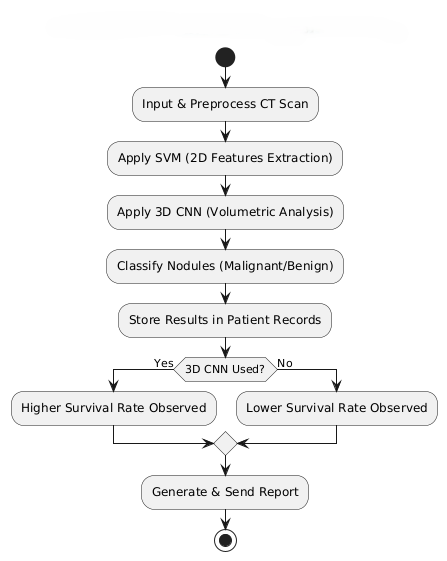
* **Operating System**: Windows 10
* **Programming Language**: Python – used for image processing, model training, and backend logic.
* **Frameworks:**
  + **Django –** For backend web development, handling user management, and integrating the SVM model.
  + **Flask –** Used for modular microservices and testing new features independently.
* **User Interface:** HTML, CSS, JavaScript – For creating responsive, interactive dashboards for image uploads and result display.
* **Libraries & Tools:**
  + **OpenCV** – For CT image preprocessing, enhancement, and segmentation.
  + **Scikit-learn** – For implementing and training the SVM model.
  + **NumPy, joblib, skimage** – For feature extraction (e.g., HOG), data handling, and model loading.

This environment enables efficient development, secure user interactions, and robust SVM-based classification—all optimized for real-time lung cancer detection.

**CHAPTER 4**

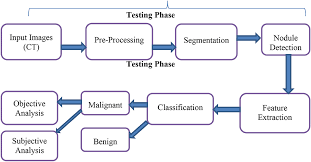
**SYSTEM DESIGN AND UML DIAGRAMS**

**4.1 DATA FLOW DIAGRAM**



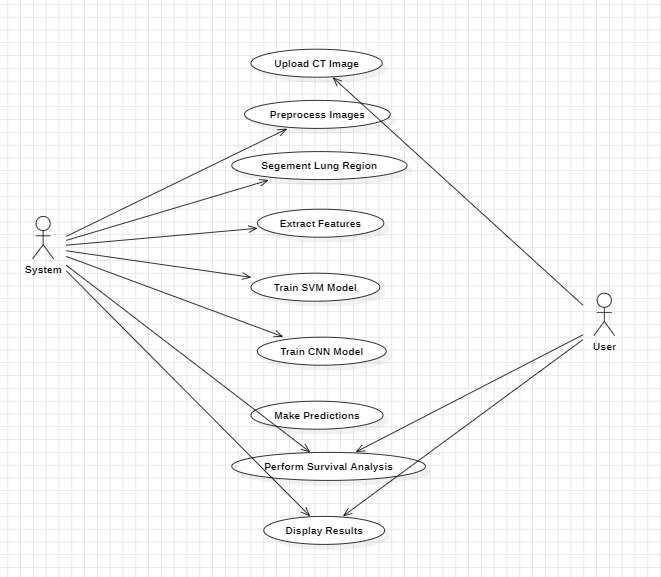
**Fig 4.1: Data Flow Diagram**

**4.2 SYSTEM ARCHITECTURE**



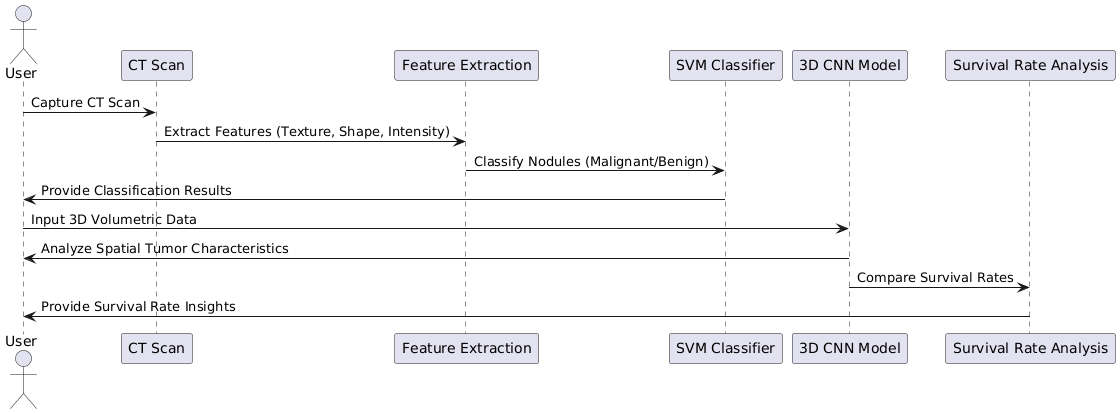
**Fig 4.2: System Architecture**

**4.3 USE CASE DIAGRAM:**



**Fig 4.3: Use Case Diagram**

**4.4 SEQUENCE DIAGRAM:**



**Fig 4.4: Sequence Diagram**

CHAPTER 5

SOFTWARE DEVELOPMENT LIFE CYCLE

## 5.1 SDLC

A software life cycle model (also termed process model) is a pictorial and diagrammatic representation of the software life cycle. A life cycle model represents all the methods required to make a software product transit through its life cycle stages. It also captures the structure in which these methods are to be undertaken.

In other words, a life cycle model maps the various activities performed on a software product from its inception to retirement. Different life cycle models may plan the necessary development activities to phases in different ways. Thus, no element in which life cycle model is followed; the essential activities are contained in all life cycle models though the action may be carried out in distinct orders in different life cycle models. During any life cycle stage, more than one activity may also be carried out.

The development team must determine a suitable life cycle model for a particular plan and then observe to it. Without using an exact life cycle model, the development of a software product would not be in a systematic and disciplined manner. When a team is developing a software product, there must be a clear understanding among team representatives about when and what to do. Otherwise, it would point to chaos and project failure. This problem can be defined by using an example. Suppose a software development issue is divided into various parts and the parts are assigned to the team members. From then on, suppose the team representative is allowed the freedom to develop the roles assigned to them in whatever way they like. One representative might start writing the code for his part, another might choose to prepare the test documents first, and some other engineer might begin with the design phase of the roles assigned to him. This would be one of the perfect methods for project failure.

A software life cycle model describes entry and exit criteria for each phase. A phase can begin only if its stage-entry criteria have been fulfilled. So, without a software life cycle model, the entry and exit criteria for a stage cannot be recognized. Without software life cycle models, it becomes tough for software project managers to monitor the progress of the project.



### **Fig 5.1: Phases of SDLC**

### **Phase 1: Requirement collection and analysis**

The requirement is the first stage in the SDLC process. It is conducted by the senior team

members with inputs from all the stakeholders and domain experts in the industry. Planning for the quality assurance requirements and recognition of the risks involved is also done at this stage.

This stage gives a clearer picture of the scope of the entire project and the anticipated issues, opportunities, and directives which triggered the project.

Requirements Gathering stage need teams to get detailed and precise requirements. This helps companies to finalize the necessary timeline to finish the work of that system.

### **Phase 2: Defining Requirements**

Once the requirement analysis is done, the next stage is to certainly represent and document the software requirements and get them accepted by the project stakeholders.

This is accomplished through the "SRS"- Software Requirement Specification document which contains all the product requirements to be constructed and developed during the project life cycle.

### **Phase 3: Designing the Software**

The next phase is about to bring down all the knowledge of requirements, analysis, and design of the software project. This phase is the product of the last two, like inputs from the customer and requirement gathering.

### **Phase 4: Developing the project**

In this phase of SDLC, the actual development begins, and the programming is built. The implementation of design begins concerning writing code. Developers have to follow the coding guidelines described by their management and programming tools like compilers, interpreters, debuggers, etc. are used to develop and implement the code.

### **Phase 5: Testing**

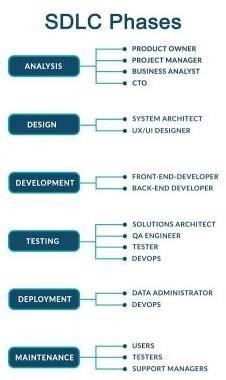
After the code is generated, it is tested against the requirements to make sure that the products are solving the needs addressed and gathered during the requirements stage. During this stage, unit testing, integration testing, system testing, acceptance testing are done.

### **Phase 6: Deployment**

Once the software is certified, and no bugs or errors are stated, then it is deployed. Then based on the assessment, the software may be released as it is or with suggested enhancement in the object segment. After the software is deployed, then its maintenance begins.

### **Phase 7: Maintenance**

Once when the client starts using the developed systems, then the real issues come up and requirements to be solved from time to time. This procedure where the care is taken for the developed product is known as maintenance.



### Fig 5.2: Roles in SDLC

**Spiral Model Application**

The Spiral Model is widely used in the software industry as it is in sync with the natural development process of any product, i.e. learning with maturity which involves minimum risk for the customer as well as the development firms.

The following pointers explain the typical uses of a Spiral Model −

* When there is a budget constraint and risk evaluation is important.
* For medium to high-risk projects.
* Long-term project commitment because of potential changes to economic priorities as the requirements change with time.
* Customer is not sure of their requirements which is usually the case.
* Requirements are complex and need evaluation to get clarity.
* New product line which should be released in phases to get enough customer feedback. Significant changes are expected in the product during the development cycle.

CHAPTER 6

IMPLEMENTATION

**6.1 SAMPLE CODE:**

import Os

import cv2

import numpy as np

import joblib

from skimage.feature import hog

from tkinter import Tk, filedialog

# Get the main project directory

base\_path = os.path.abspath(os.getcwd())

model\_path = os.path.join(base\_path, "model")

# Load trained model

svm = joblib.load(os.path.join(model\_path, "svm\_lung\_cancer.pkl"))

# Open file dialog to upload an image

def upload\_image():

    Tk().withdraw()  # Hide the root Tkinter window

    image\_path = filedialog.askopenfilename(title="Select a Lung CT Scan",

                                            filetypes=[("Image Files", "\*.jpg;\*.png;\*.jpeg")])

    return image\_path

# Function to preprocess and extract features

def predict\_lung\_cancer(image\_path):

    if not image\_path:

        print("❌ No image selected. Please upload an image.")

        return

    if not os.path.exists(image\_path):

        print(f"❌ ERROR: Image '{image\_path}' not found! Please check the file path.")

        return

    img = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)

    # Ensure image is read properly

    if img is None:

        print(f"❌ ERROR: Unable to read image '{image\_path}'. Check if the file is corrupted.")

        return

    img = cv2.resize(img, (128, 128))

    img = cv2.equalizeHist(img)

    # Extract HOG features

    features, \_ = hog(img, orientations=9, pixels\_per\_cell=(8, 8),

                      cells\_per\_block=(2, 2), visualize=True)

    features = np.array(features).reshape(1, -1)  # Reshape for prediction

    # Predict

    prediction = svm.predict(features)

    # Map prediction to class

    class\_map = {0: "Benign (Non-Cancerous)", 1: "Malignant (Cancerous)", 2: "Normal"}

    return class\_map[prediction[0]]

# Upload image and predict

test\_image\_path = upload\_image()

if test\_image\_path:

    result = predict\_lung\_cancer(test\_image\_path)

    if result:

        print(f"\n✅ Prediction for '{test\_image\_path}': {result}")

CHAPTER 7

TESTING

## INTRODUCTION

Testing is a group of techniques to determine the correctness of the application under the predefined script but, testing cannot find all the defects of application. The main intent of testing is to detect failures of the application so that failures can be discovered and corrected. It does not demonstrate that a product functions properly under all conditions but only that it is not working in some specific conditions.

Testing includes an examination of code and also the execution of code in various environments, and conditions as well as all the examining aspects of the code. In the current scenario of software development, a testing team may be separate from the development team so that Information derived from testing can be used to correct the process of software development.

The success of software depends upon the acceptance of its targeted audience, easy graphical user interface, strong functionality load test, etc. For example, the audience of banking is totally different from the audience of a video game. Therefore, when an organization develops a software product, it can assess whether the software product will be beneficial to its purchasers and other audiences.

Types of manual testing:

* White Box Testing
* Black Box Testing
* Gray Box Testing

### **White Box Testing:**

The box testing approach of software testing consists of black box testing and white box testing. White box testing which also known as glass box is testing, structural testing, clear box testing, open box testing, and transparent box testing. It tests the internal coding and infrastructure of a software focused on checking predefined inputs against expected and desired outputs. It is based on the inner workings of an application and revolves around internal structure testing. In this type of testing programming skills are required to design test cases. The primary goal of white box testing is to focus on the flow of inputs and outputs through the software and strengthen the security of the software.

Generic steps of white box testing:

* Design all test scenarios, and test cases and prioritize them according to high-priority number.
* This step involves the study of code at runtime to examine the resource utilization, non- accessured areas of the code, time taken by various methods and operations and so on.
* In this step testing of internal subroutines takes place. Internal subroutines such as nonpublic methods, and interfaces can handle all types of data appropriately or not.
* This step focuses on testing control statements like loops and conditional statements to check the efficiency and accuracy of different data inputs.
* In the last step white box testing includes security testing to check all possible security loopholes by looking at how the code handles security.

**Black box testing:**

Black box testing is a technique of software testing that examines the functionality of the software without peering into its internal structure or coding. The primary source of black box testing is a specification of requirements that are stated by the customer.

In this method, the tester selects a function that gives input value to examine its functionality and checks whether the function is giving the expected output or not. If the function produces the correct output, then it is passed in testing, otherwise failed. The test team reports the result to the development team and then tests the next function. After completing testing of all functions if there are severe problems, then it is given back to the development team for correction.

Generic steps of black box testing:

* The black box test is based on the specification of requirements, so it is examined in the beginning.
* In the second step, the tester creates a positive test scenario and an adverse test scenario by selecting valid and invalid input values to check that the software is processing them correctly or incorrectly.
* In the third step, the tester develops various test cases such as decision table, all pairs test, equivalent division, error estimation, cause-effect graph, etc.
* The fourth phase includes the execution of all test cases.
* In the fifth step, the tester compares the expected output against the actual output.
* In the sixth and final step, if there is any flaw in the software, then it is cured and tested again.

**Unit Testing:**

Unit testing involves the testing of each unit or an individual component of the software application. It is the first level of functional testing. The aim behind unit testing is to validate unit components with their performance.

A unit is a single testable part of a software system and is tested during the development phase of the application software.

The purpose of unit testing is to test the correctness of isolated code. A unit component is an individual function or code of the application. The white box testing approach is used for unit testing and is usually done by the developers.

Whenever the application is ready and given to the Test engineer, he/she will start checking every component of the module or module of the application independently or one by one, and this process is known as Unit testing or components testing.

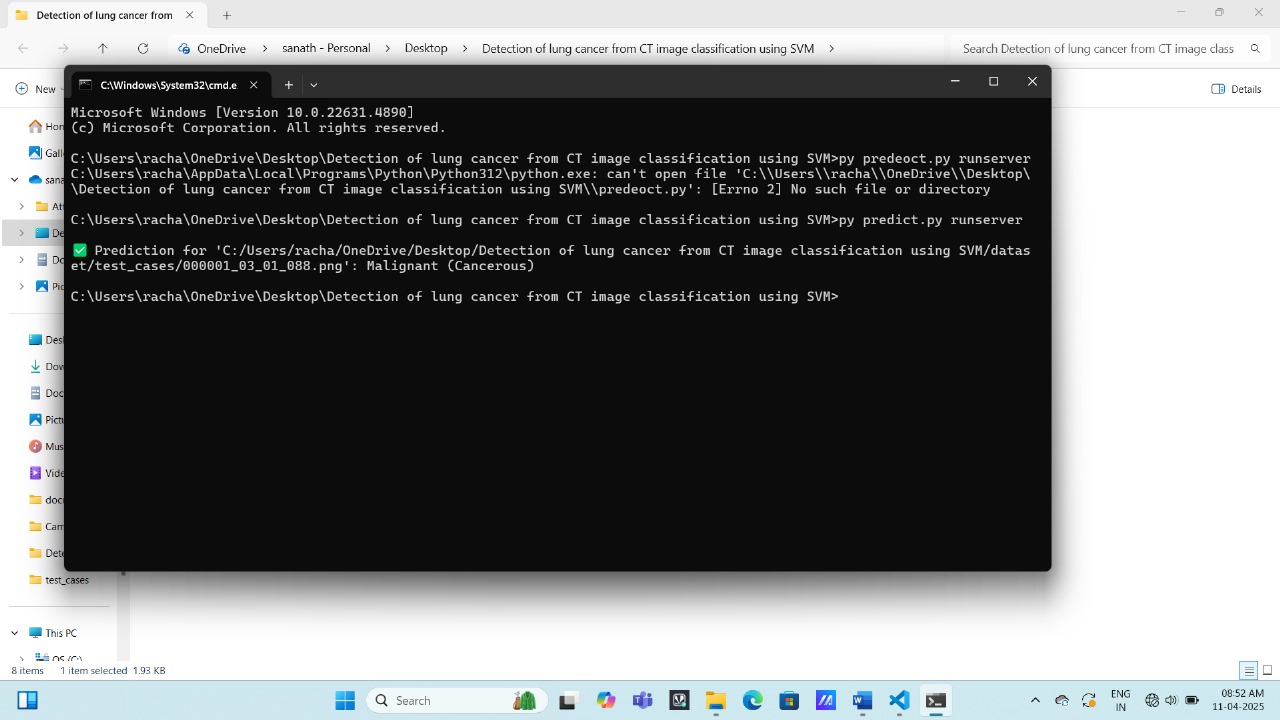
**Integration Testing:**

Integration testing is the second level of the software testing process comes after unit testing. In this testing, units or individual components of the software are tested in a group. The focus of the integration testing level is to expose defects at the time of interaction between integrated components or units.

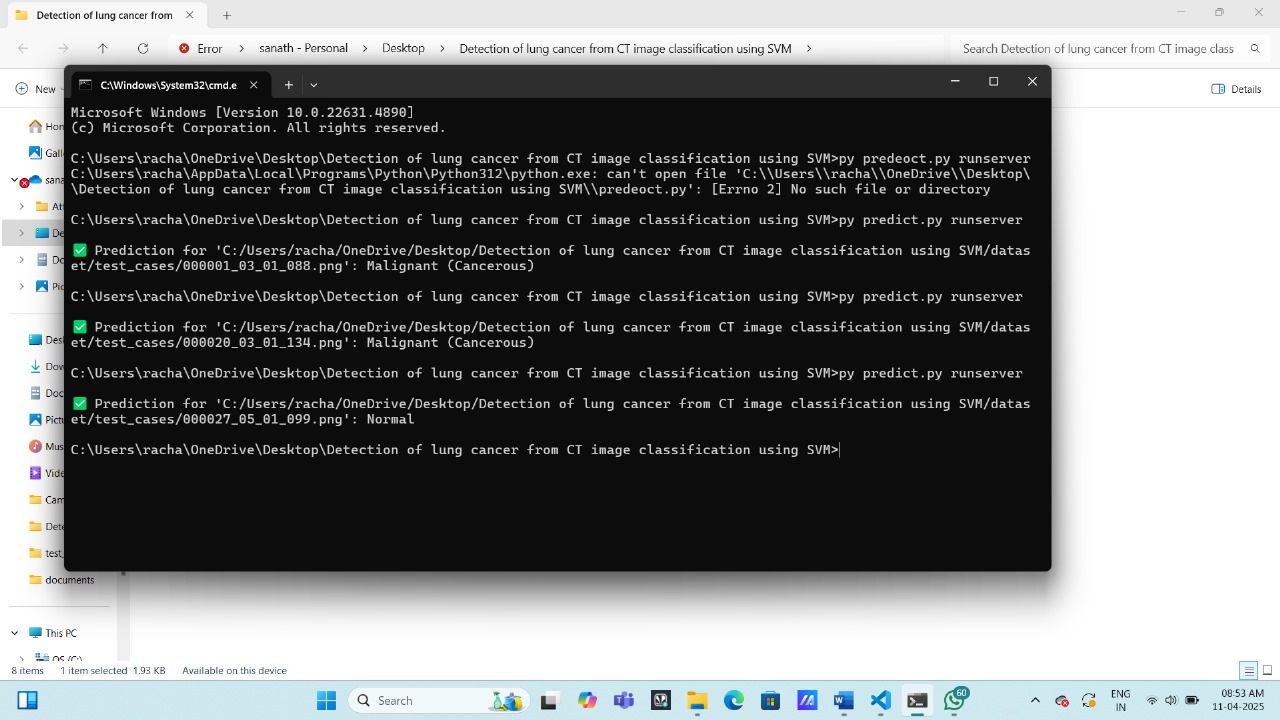
Unit testing uses modules for testing purposes, and these modules are combined and tested in integration testing. The Software is developed with several software modules that are coded by different coders or programmers. The goal of integration testing is to check the correctness of communication among all the modules.

CHAPTER 8

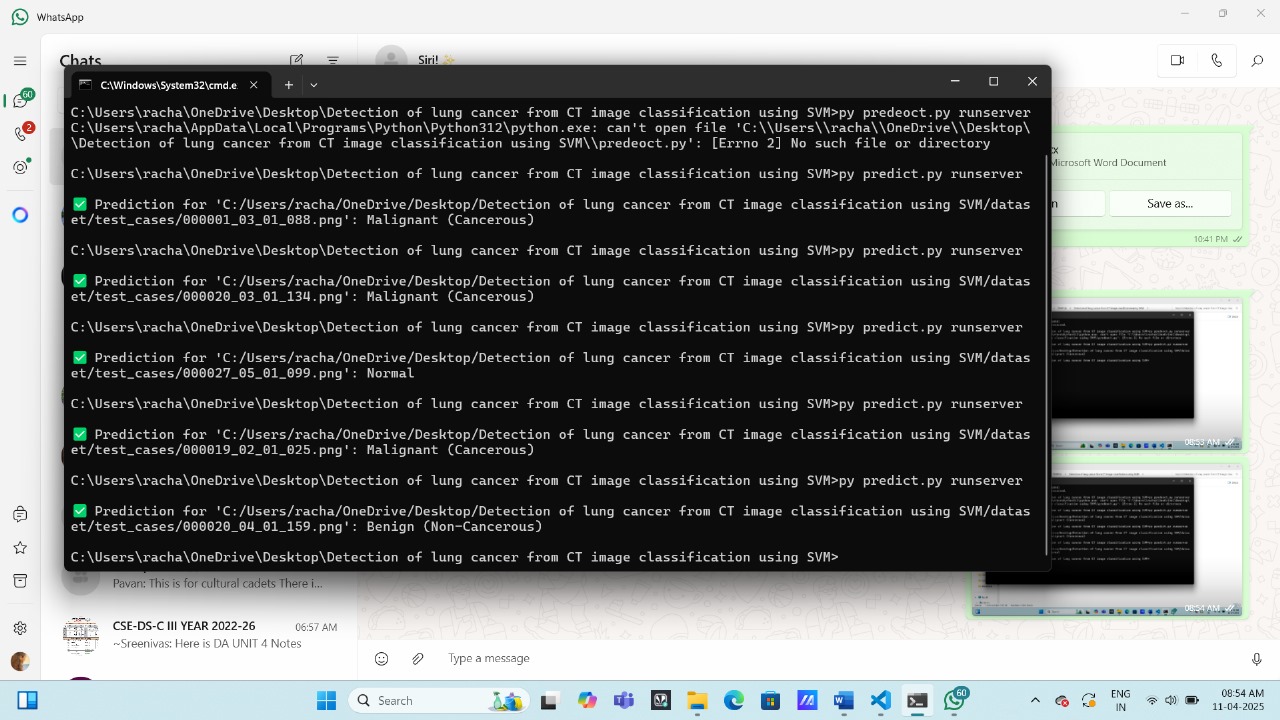
OUTPUT SCREENS



**Fig.8.1 showing classification of disease**



**Fig 8.2 classification of condition**



**Fig 8.3 showing various states of condition**

CHAPTER 9

CONCLUSION AND FUTURE SCOPE

**9.1 CONCLUSION**

This project successfully demonstrates the application of machine learning—specifically Support Vector Machine (SVM)—in the field of medical imaging for the early detection of lung cancer. By preprocessing CT scan images and extracting meaningful features using methods such as Histogram of Oriented Gradients (HOG), the system classifies images into cancerous and non-cancerous categories with notable efficiency. The integration of Python, OpenCV, Django, and Scikit-learn provided a solid backend for image processing, model training, and web deployment. Additionally, the web-based interface allows radiologists and healthcare professionals to interact with the system seamlessly, enabling quick and accessible diagnostic support. Overall, the project represents a valuable step toward enhancing diagnostic capabilities and promoting AI-assisted healthcare

**9.2 FUTURE SCOPE**

**Integration of Deep Learning Models**

Replace or enhance the SVM classifier with deep learning models such as Convolutional Neural Networks (CNNs) or hybrid architectures (e.g., CNN + SVM) to achieve better accuracy, especially on large and complex datasets.

**Larger and More Diverse Datasets**

Incorporate larger, multi-center datasets such as LIDC-IDRI to improve model robustness and generalization across different demographics and imaging settings.

**Multiclass Classification**

Expand the binary classification to include multiple lung disease categories (e.g., benign tumor, malignant tumor, pneumonia) to offer a broader diagnostic tool.

**Real-Time Cloud Deployment**

Host the platform on cloud services (e.g., AWS, Azure) to allow real-time diagnosis and remote accessibility for hospitals, clinics, and rural healthcare centers.

**Mobile Application Integration**

Develop a mobile app version to further improve accessibility for on-the-go healthcare providers.

**Explainable AI (XAI)**

Integrate explainable AI techniques to provide visual or textual justification for predictions, enhancing trust among medical professionals.

**Integration with Electronic Health Records (EHR)**

Enable interoperability with existing hospital systems for seamless data exchange, allowing longitudinal tracking of patient data.

**Continuous Learning**

Implement online or incremental learning capabilities so that the system improves over time as more data becomes available.

**Regulatory Compliance and Certification**

Work toward aligning the system with medical device regulations such as FDA, CE, or HIPAA to prepare for real-world clinical deployment.

CHAPTER – 10

REFERENCES

1. <https://github.com/>
2. <https://chat.openai.com/>
3. <https://github.com/>
4. <https://chat.openai.com/>
5. Ahmad, Y. J, Serafy, S. E and Lutz, E. (eds) 1989, Environmental AccountingforSustainable Development, The World Bank, Washington, D.C.
6. Crosson, P. and Anderson, J. R. 1993, Concerns for Sustainability: Integration of Natural Resource and Environmental Issues for the Research Agendas of NARS, Research Report 4, ISNAR, The Hague.
7. Crosson, P. and Anderson, J. R. 1995a, achieving a Sustainable Agricultural SysteminSub-Saharan Africa, Building Blocks for Africa 2025, Paper No. 2, AFTES, The WorldBank, Washington, D.C.