

A PROJECT REPORT ON

**ENHANCING TUBERCULOSIS
DETECTION IN CHEST X-RAYS USING
ATTENTION MODULES**

Major project submitted in partial fulfillment of the requirements for the award
of the degree of

MASTER OF COMPUTER APPLICATIONS
(2022-2024)

By

K HIMAKAR
160122862041

Under the Esteemed Guidance of
Mr. B SRINIVASA SP KUMAR
Assistant Professor
DEPARTMENT OF MCA, CBIT(A)



Chaitanya Bharathi Institute of Technology (Autonomous)
(Affiliated to Osmania University, Hyderabad)
Hyderabad, TELANGANA (INDIA) –500 075



**CHAITANYA BHARATHI
INSTITUTE OF TECHNOLOGY**
An Autonomous Institute | Affiliated to Osmania University
Kokapet Village, Gandipet Mandal, Hyderabad, Telangana-500075, www.cbit.ac.in

Approved by
Recognized
Research Centers
Programs
Accredited by
Grade A++ in
All India Ranking 151-200 Band
ISO Certifications
Quality Audit 9001 : 2015
Green Audit 14001 : 2015
Energy Audit 50001 : 2018

COMMITTED TO
RESEARCH,
INNOVATION AND
EDUCATION

45
years

CERTIFICATE

This is to certify that the project title **“ENHANCING TUBERCULOSIS DETECTION IN CHEST X-RAYS USING ATTENTION MODULES”** is the bonafide work carried out by **K Himakar** bearing **Roll no160122862041**, a student of Chaitanya Bharathi Institute of Technology(A), Hyderabad, affiliated to Osmania University, Hyderabad, Telangana (India) during the period of 2022-2024, submitted in partial fulfillment of the requirements for the award of the degree in **Master of Computer Applications** and that the project has not formed the basis for the award previously of any other degree, diploma, fellowship or any other similar title.

Project Guide

Mr. B. SRINIVASA SP KUMAR
Assistant Professor
Dept. of MCA, CBIT (A)
Hyderabad

Head of the Department

MCA, CBIT(A)
Hyderabad

Project External Examiner

DECLARATION

I hereby declare that the project entitled “**ENHANCING TUBERCULOSIS DETECTION IN CHEST X-RAYS USING ATTENTION MODULES**” submitted towards the partial fulfilment of the requirements for the award of the degree of Master of Computer Application, Chaitanya Bharathi Institute of Technology, Hyderabad is an authentic record of my own work carried out under the Guidance of **Mr. B. Srinivasa SP Kumar Assistant Professor**, Department of MCA, Chaitanya Bharathi Institute of Technology, Hyderabad.

To the best of my knowledge and belief, this project bears no resemblance with any report submitted to Chaitanya Bharathi Institute of Technology, Hyderabad or any other University for the award of any degree.

Place: HYDERABAD

K HIMAKAR

Date:

160122862041

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of the task would be put incomplete without the mention of the people who made it possible, whose constant guidance and encouragement crown all the efforts with success.

It is my privilege and pleasure to express my profound sense of respect, gratitude and indebtedness to my guide **Mr. B Srinivasa SP Kumar, Assistant Professor**, Department of MCA, CBIT(A), Hyderabad for his constant guidance, inspiration, and constant encouragement throughout this project work.

It's my privilege and pleasure to express my gratitude to Project Coordinators **Dr. GNR Prasad, Assistant Professor, Department of MCA, CBIT(A), Hyderabad** and **Mr. P. Krishna Prasad, Assistant Professor, Department of MCA, CBIT(A), Hyderabad** for their constant guidance and their assistance during each phase of carrying out this project work.

I wish to express my deep gratitude to **HOD, MCA, CBIT(A), Hyderabad**, for his cooperation and encouragement, in addition to providing necessary facilities throughout the project work.

I sincerely extend my thanks to **Dr. C.V. Narasimhulu, Principal, CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY(A), Hyderabad**.

I also thank all the staff members of the MCA department for their valuable support and generous advice. Finally, thanks to all our friends and family members for their continuous support and enthusiastic help.



Name: K HIMAKAR
Roll No.: 160122862041
Email: Konankihimakar699@gmail.com
Phone No.:8105109208

Abstract

Tuberculosis (TB) remains a critical global health challenge, necessitating early and precise diagnosis for effective treatment and containment. Existing diagnostic systems, primarily employing Artificial Neural Networks (ANN), often struggle to capture essential spatial and contextual information. This project evaluates the performance of a DenseNet-121 architecture integrated with a Convolutional Block Attention Module (CBAM) for TB diagnosis using X-ray images. DenseNet-121 is known for its dense connectivity, enabling effective feature reuse and deep extraction, while CBAM enhances the model's focus through channel and spatial attention mechanisms, thereby identifying crucial features and their locations more accurately. The integrated model aims to leverage these advanced capabilities to enhance diagnostic accuracy and efficacy. Experimental results demonstrate that the DenseNet-121 with CBAM significantly improves TB detection compared to traditional ANN-based systems, achieving higher accuracy, sensitivity, and specificity. The model's ability to reduce false negatives and false positives is particularly noteworthy, as it ensures more reliable TB diagnosis. These findings highlight the importance of advanced neural network architectures and attention mechanisms in medical image analysis. The study underscores the potential of innovative AI-driven diagnostic tools in addressing global health challenges, suggesting that such integrated models could be further explored and applied to other medical conditions and imaging modalities, thereby advancing the field of AI in healthcare and contributing to better patient outcomes worldwide.

Table of Contents

	Title Page	I
	Certificate	II
	Declaration	III
	Acknowledgement	IV
	Abstract	V
	List of Figures	VII
	List of Tables	VIII
	List of abbreviation	IX
1.	INTRODUCTION	1
	1.1 Problem Definition including the significance and objective	2-4
	1.2 Methodologies	4-5
	1.3 Outline of the results	5-6
	1.4 Scope of the project	6-7
		7-8
2.	LITERATURE SURVEY	9
	2.1 Introduction to the problem domain terminology	10-11
	2.2 Existing solutions	12-13
	2.3 Related works	13
3.	SYSTEM DESIGN AND ALGORITHMS	14
	3.1 Design Diagrams	15
	3.1.1 System Architecture	15
	3.1.2 DFDs	15-16
	3.1.3 UML Diagrams	17-23
	3.2 Module(s) Description	23-25
	3.3 Algorithms	25-27
	3.4 System Requirements (Software and Hardware)	31
4	IMPLEMENTATION	32
	4.1 Data Set Description	33
	4.2 Testing Process Criteria	34-37
5.	RESULTS OUTPUT DISCUSSIONS	38
	5.1 User Interface screenshots	39-41
	5.2 Comparative Analysis	42-45
6.	CONCLUSIONS AND RECOMMENDATIONS	46
	6.1 Conclusions	47
	6.1.1 Limitations	48-49
	6.2 Recommendations AND Future Work /Future Scope	49-51
	REFERENCES	52

List of Figures

Figure No.	Figure Caption	Page No.
1.2.1	Applying Data Augmentation to x-rays	20
3.1	System Architecture	16
3.2	Contextual Diagram	17
3.3	Level-1 Diagram	17
3.1.3.1	Use Case Diagram	20
3.1.3.2	Class Diagram	22
3.1.3.3	Sequence Diagram	23
5.1.1	User Interface	40
5.1.2	Tuberculosis Output	41
5.1.3	Normal Output	41
5.2.1	Model Training	42
5.2.2	Training and validation accuracy	43
5.2.3	Training and validation loss	45

List of Tables

Table No.	Table Caption	Page No.
4.2.1	Unit Test Cases	36
4.2.2	System Test Cases	37

Abbreviations

Abbreviation	Description
CNN	Convolutional Neural Networks
TB	Tuberculosis
CXR	Chest X-Ray
SL-ANN	Stochastic Learning Artificial Neural Networks
DFD	Data Flow Diagram
GPU	Graphical Processing Unit
CAM	Channel Attention Module
CBAM	Convolutional Block Attention Module

CHAPTER 1

INTRODUCTION

1.1 Problem Definition including significance and Objectives

Tuberculosis (TB) remains a significant global health concern, particularly in developing countries where access to healthcare resources is limited. Early and accurate detection of TB is crucial for timely treatment and prevention of transmission. Chest X-rays (CXR) are one of the primary diagnostic tools for TB screening due to their affordability, widespread availability, and effectiveness in identifying pulmonary abnormalities associated with TB infection. However, the interpretation of CXR images for TB detection is challenging and often requires expertise, leading to delays in diagnosis and treatment initiation. Moreover, the subjective nature of manual interpretation can result in variability among radiologists' assessments. To address these challenges and improve TB detection accuracy, there is a growing interest in leveraging deep learning techniques, particularly convolutional neural networks (CNNs), for automated image analysis. In this project, we propose a novel approach for TB detection in CXR images by integrating channel and spatial attention mechanisms with DenseNet-121, a state-of-the-art CNN architecture. The integration of attention mechanisms aims to enhance the model's ability to identify relevant features indicative of TB infection while reducing the influence of noise and irrelevant information. Additionally, we develop a user-friendly web-based application using Flask, a lightweight Python web framework, to deploy the TB detection model, making it accessible to healthcare professionals for rapid and efficient diagnosis. DenseNet-121 is a deep CNN architecture known for its dense connectivity pattern, where each layer receives input from all preceding layers. This dense connectivity facilitates feature reuse and enhances gradient flow, leading to improved model performance and parameter efficiency.

The TB detection model, integrated with channel and spatial attentions and DenseNet-121, is deployed within a web-based application using Flask. The application provides a user-friendly interface that allows healthcare professionals to upload CXR images securely and receive automated TB detection results in real-time. Leveraging Flask's flexibility and simplicity, we

design an intuitive and responsive interface that accommodates users with varying levels of technical expertise.

By combining advanced deep learning techniques with practical web development using Flask, our project aims to streamline the process of TB detection from CXR images, facilitating early diagnosis and treatment initiation. The integration of attention mechanisms with DenseNet-121 enhances the model's sensitivity and specificity, improving diagnostic accuracy and reducing the burden on healthcare systems. Ultimately, our goal is to provide a scalable and accessible solution for TB screening that contributes to global efforts in TB control and eradication. In this project, our objectives are multifaceted, aiming to address the complex challenges associated with tuberculosis (TB) detection in chest X-ray (CXR) images using advanced deep learning techniques integrated with practical web application development. Firstly, our primary objective is to develop and implement a robust TB detection model by integrating channel and spatial attention mechanisms with DenseNet-121, a state-of-the-art convolutional neural network (CNN) architecture.

Secondly, we seek to design and deploy a user-friendly web-based application using Flask, a lightweight Python web framework, to make the TB detection model accessible to healthcare professionals. This application will provide a seamless interface for users to securely upload CXR images and receive rapid automated TB detection results in real-time. Furthermore, our objectives include validating the performance of the integrated TB detection model through rigorous experimentation and evaluation using diverse datasets comprising CXR images with varying degrees of TB pathology. Through comprehensive performance evaluation metrics such as sensitivity, specificity, we aim to demonstrate the efficacy and reliability of our proposed approach in TB detection.

Additionally, we aim to contribute to the advancement of TB control efforts by disseminating our findings and sharing the developed TB detection model and web application as open-source resources. By making our solution freely available to the global healthcare community, we aim to empower healthcare professionals worldwide with a powerful tool for early TB diagnosis, ultimately

contributing to improved patient outcomes and the reduction of TB burden on healthcare systems globally.

1.2 METHODOLOGY

- **OVERVIEW OF THE DATASET**

The meticulous division of the dataset into training, testing, and validation sets ensures a thorough evaluation of the model's performance while maintaining robust training protocols. With a substantial dataset of 3111 images, researchers can effectively train machine learning models to distinguish normal chest X-rays from those indicative of tuberculosis. This organized approach facilitates efficient model training and validation, enhancing the accuracy and reliability of research outcomes. Additionally, categorizing X-rays into subfolders based on Normal and Tuberculosis classes enables targeted analysis, crucial for assessing the model's efficacy in clinical contexts. This method ensures a comprehensive evaluation of the model's capabilities across diverse classes, contributing to its potential utility in real-world healthcare applications.

- **DATA PREPROCESSING**

Before training the deep learning model, it's essential to preprocess the dataset. This involves standardizing the image sizes to 224 x 224 pixels and dividing the dataset into training, validation, and test sets. Furthermore, to enhance the diversity of the training data and improve the model's ability to generalize across different scenarios, additional techniques such as random rotation and zoom are employed to augment the dataset.

- **DATA AUGMENTATION**

This section is crucial for preparing and loading image data for training, validation, and testing in a deep learning model. First, I defined the paths to the directories containing the training, validation, and test datasets. Then, I specified data augmentation parameters using the ``ImageDataGenerator`` class from Keras, which allows for real-time data augmentation during model training. The ``train_datagen`` applies various augmentation techniques such as rotation, shifting, shearing, zooming, and horizontal flipping to enhance the diversity of the training data and improve the model's ability to generalize. In contrast, the

`validation_datagen` and `test_datagen` only apply rescaling as these sets are used solely for model evaluation. Next, I used data generators using the `flow_from_directory` method, which loads images from the specified directories, resizes them to the target size of 224x224 pixels, and generates batches of augmented data during model training. The `class_mode` parameter is set to 'binary' assuming a binary classification task, and shuffling is enabled for the validation data generator to ensure randomness during validation. Fig. 1 represents the data augmentation applied to some x-rays of training dataset, which shows the zoom, rotate, shifting features. Conversely, shuffling is disabled for the test data generator to maintain the image order for accurate evaluation. In summary, this part streamlines the data pre-processing and loading process, establishing a strong foundation for training and evaluating deep learning models for image classification tasks.

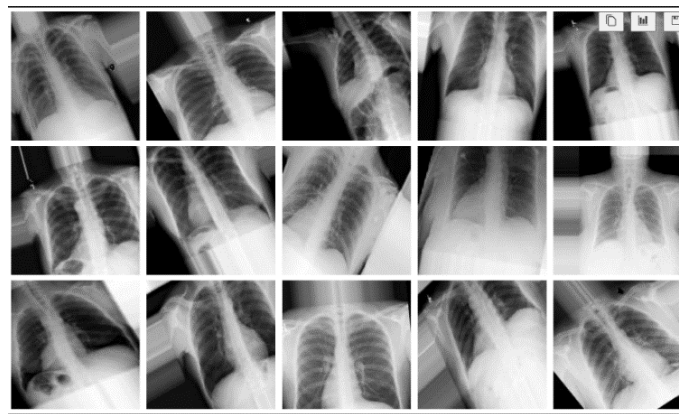


Fig. 1.2.1. Applying Data Augmentation to x-rays

- **Description:** The image shows a grid of chest X-rays, each subjected to various augmentations such as rotation, scaling, and blurring. These transformations are likely applied for the purpose of enhancing a machine learning model's ability to generalize and accurately analyze medical imaging data

1.3 Outline of the Results

- **Model Development and Performance**

Architecture Design: Description of the DenseNet-121 model with integrated channel and spatial attention mechanisms.

Training Process: Overview of the training methodology, including data augmentation, loss functions, and optimization techniques.

Performance Metrics: Presentation of key metrics such as accuracy, sensitivity, specificity, precision, recall, and F1-score on validation and test datasets.

Comparative Analysis: Comparison of the proposed model's performance against baseline models and other state-of-the-art TB detection methods.

- **Attention Mechanism Analysis**

Impact of Attention Mechanisms: Evaluation of how channel and spatial attention mechanisms improve the model's focus on relevant features.

Visualization: Visualization of attention maps to demonstrate the regions of CXR images that the model focuses on for TB detection.

- **Web Application Development**

User Interface Design: Overview of the user interface features, including image upload functionality, result display, and user feedback mechanisms.

Backend Integration: Description of the Flask framework implementation and how the TB detection model is integrated into the web application.

- **Real-Time Testing**

Deployment Environment: Details on the deployment environment used for real-time testing of the web application.

Response Time: Measurement of the application's response time from image upload to result delivery.

- **Validation and Evaluation**

Dataset Diversity: Information on the diverse datasets used for model validation, including variations in TB pathology.

Robustness Testing: Results from testing the model's robustness across different types of CXR images (e.g., different age groups, image quality variations).

- **Impact on Global TB Control Efforts**

Early Diagnosis and Treatment: Discussion on how the tool aids in early TB diagnosis and timely treatment initiation.

Healthcare System Efficiency: Insights into how the tool reduces the burden on healthcare systems by streamlining the TB detection process.

Patient Outcomes: Potential impact on improving patient outcomes through faster and more accurate TB diagnosis.

1.5 Scope of the Project

- **Development of the TB Detection Model**

Model Architecture: Design and implementation of a convolutional neural network (CNN) based on DenseNet-121 architecture.

Attention Mechanisms: Integration of channel and spatial attention mechanisms to enhance the model's feature extraction capabilities.

Training and Validation: Use of diverse datasets to train and validate the model, ensuring robust performance across various TB pathology presentations.

- **Web Application Development**

Framework Selection: Utilization of Flask, a lightweight Python web framework, to develop the web application.

User Interface (UI): Creation of a user-friendly interface that allows healthcare professionals to upload CXR images and receive real-time TB detection results.

Backend Integration: Seamless integration of the TB detection model with the Flask web application to facilitate efficient processing and result delivery.

Security Protocols: Implementation of security measures to ensure patient data confidentiality and secure image handling.

- **Performance Evaluation**

Evaluation Metrics: Comprehensive evaluation of the model using metrics such as accuracy, sensitivity, specificity, precision, recall, and F1-score.

Comparison with Existing Methods: Benchmarking the proposed model against existing TB detection methods to demonstrate its superiority.

- **Real-Time Testing and User Feedback**

Deployment Environment: Set up a deployment environment for real-time testing of the web application.

Response Time Measurement: Evaluate the application's response time from image upload to result delivery.

User Feedback Collection: Gather feedback from healthcare professionals to assess the usability and effectiveness of the web application.

- **Impact Assessment**

Early Diagnosis and Treatment: Evaluate the tool's effectiveness in facilitating early TB diagnosis and timely treatment initiation.

Healthcare System Efficiency: Assess the impact of the tool on reducing the burden on healthcare systems by streamlining the TB detection process.

Patient Outcomes: Measure the potential improvement in patient outcomes through faster and more accurate TB diagnosis.

CHAPTER 2

LITERATURE

SURVEY

2.1 Introduction to problem domain terminology

- **Tuberculosis (TB)**

Definition: Tuberculosis is an infectious disease caused by the bacterium *Mycobacterium tuberculosis*. It primarily affects the lungs but can also impact other parts of the body.

Transmission: TB spreads through the air when people with active pulmonary TB cough, sneeze, or talk, releasing bacteria into the air.

Symptoms: Common symptoms include a persistent cough, chest pain, fatigue, weight loss, fever, and night sweats.

- **Chest X-Ray (CXR)**

Definition: A CXR is a radiographic image of the chest used to diagnose conditions affecting the chest, including the lungs, heart, and chest wall.

Use in TB Detection: CXRs are one of the primary tools for TB screening as they can reveal abnormalities in the lungs indicative of TB infection, such as lesions, nodules, or cavities.

- **Convolutional Neural Networks (CNNs)**

Definition: CNNs are a class of deep learning algorithms specifically designed to process and analyze visual data. They are particularly effective for image recognition and classification tasks.

- **DenseNet-121**

Definition: DenseNet-121 is a type of CNN characterized by its dense connectivity pattern, where each layer receives inputs from all preceding layers. This architecture promotes feature reuse and improves gradient flow, leading to better model performance and efficiency. DenseNet-121 is known for its ability to capture intricate details in images while maintaining a manageable number of parameters, making it both powerful and computationally efficient.

- **Attention Mechanisms**

Channel Attention: Focuses on enhancing the importance of relevant feature channels in the image data. This mechanism helps the model to prioritize significant features across different channels.

Spatial Attention: Concentrates on identifying and emphasizing critical spatial regions within the image. This helps the model to focus on specific areas that are more likely to contain important information for the task at hand.

- **Flask**

Definition: Flask is a lightweight and flexible Python web framework used to develop web applications. It is known for its simplicity and ease of use, making it suitable for rapid development and deployment of web-based interfaces.

Use in This Project: Flask is used to create a web-based application that allows healthcare professionals to upload CXR images and receive automated TB detection results in real-time.

- **Performance Metrics**

Accuracy: The ratio of correctly predicted instances to the total instances.

Sensitivity (Recall): The ability of the model to correctly identify positive cases (true positives).

Specificity: The ability of the model to correctly identify negative cases (true negatives).

Precision: The ratio of true positive predictions to the total predicted positives.

F1-Score: The harmonic-mean of precision and recall, providing a single metric to evaluate the balance between precision and recall.

2.2 Existing Solutions

There are several traditional medical image segmentation methods, such as

[1] Medical threshold-based methods [2], which are based on the size of medical image pixels and rank the images according to the difference of image pixels.

Region-based methods [3], which directly find specific regions and cluster the parts with common properties into small pixel regions by attribute analysis.

Boundary based methods [4], which use gradient, difference, and other operators and filtering methods to process the images. However, because the shape and contour of pulmonary emboli vary greatly from patient to patient in CTPA images, it is difficult to guarantee the accuracy of pulmonary embolism segmentation using traditional methods in practice. Currently, semantic segmentation methods based on convolutional neural networks are very popular in the field of natural image processing.

In 2014, Long et al [5]. proposed FCN, a fully convolutional neural network, which was improved from VGG16 to make it more suitable for image segmentation tasks. The FCN network replaces the last three fully-connected layers in the VGG16 network with convolutional layers and enables the output of the network to be the same size as the original input image through an up-sampling operation to achieve image segmentation. In view of the excellent segmentation performance of FCN network in natural image processing, many researches started to use it in medical image processing.

In 2015, Ranneberger et al [6], proposed U-Net, which is FCN-based network with an encoder, bottleneck and decoder, and is widely used in medical image segmentation because of its U-shaped structure which combines with contextual information, fast training speed and small training data.

Disadvantages of Existing System:

Computational Complexity: The stochastic learning techniques employed in the SL-ANN approach, such as random shuffling of training datasets before each

iteration, significantly increase computational complexity. This complexity leads to longer training times and higher resource requirements, making the approach less suitable for real-time or high-throughput TB screening applications.

Risk of Overfitting: The SL-ANN approach may be susceptible to overfitting, especially when dealing with small or imbalanced datasets.

2.3 Related Works:

Cao et al. [4] has developed large-scale, well-annotated, and real-world X-ray image dataset for automated tuberculosis screening and also focused on building effective and economical computer models that classify images into distinct categories of TB symptoms using deep CNN model. To identify masses from mammograms, The GLCM with texture-based descriptor on the spatial connection between different VOLUME 10, 2022 103633 S. Urooj et al.: Stochastic Learning-Based ANN Model for an Automatic TB Detection System Using Chest X-Ray Images FIGURE 1. Model dataset images from Shenzhen and Montgomery [27]. pixel pairs, was employed by Khuza et al. [5]. Jaeger et al. [6] suggested an automatic TB detection method that computed low-level features such as shape-based and texture-based descriptors from Chest X-Rays images using a LBP. The collected features are put into a binary classifier on two smaller datasets to produce with following accuracy of each dataset of 78.3% and 80%, respectively. Anthimopoulos et al. [7] constructed a deep learning with CNN architecture and utilised it to diagnose pulmonary lung problems using CT scans, providing a higher-compactness resolution image of lung components in 2D or 3D formats, in comparison to CXRs. Interestingly, their suggested model can distinguish six different lung disease presentations as well as healthy instances with a shown accuracy of >80%.

CHAPTER 3

SYSTEM DESIGN

AND ALGORITHMS

3.1 Design Diagrams

3.1.1 System Architecture:

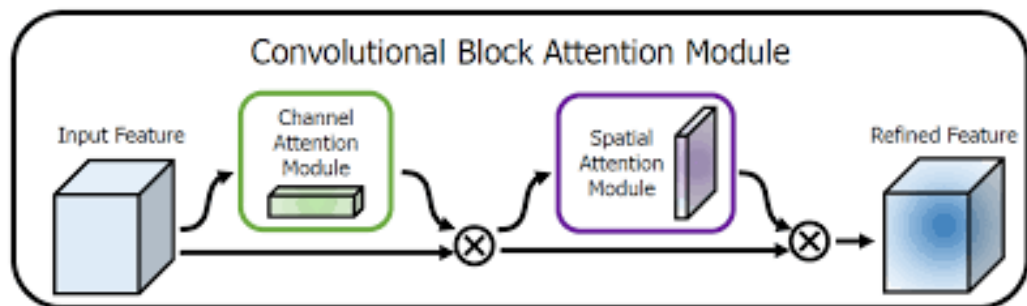


Fig 3.1 System Architecture

Description: In the above fig 3.1, we can see that feature extraction and detection is done

3.1.2 DATA FLOW DIAGRAM:

- The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of input data to the system, various processing carried out on this data, and the output data is generated by this system.
- The data flow diagram (DFD) is one of the most important modeling tools. It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information that flows in the system.
- DFD shows how the information moves through the system and how it is modified by a series of transformations. It is a graphical technique that depicts information flow and the transformations that are applied as data moves from input to output.
- DFD is also known as bubble chart. A DFD may be used to represent a system at any level of abstraction. DFD may be partitioned into levels that represent increasing information flow and functional details.

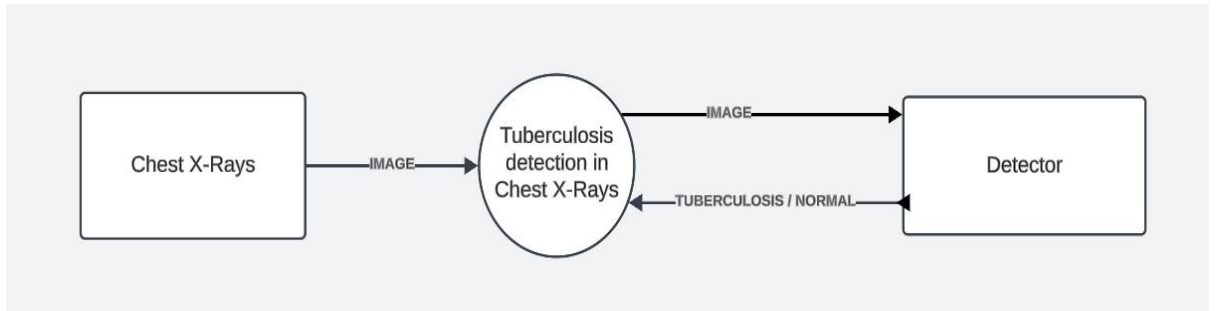


Fig 3.2 Contextual Diagram

Description: The above fig 3.2 gives the data flow diagram of the process that's happening in the project. We can see the process of project that is uploading image and detection of Tuberculosis.

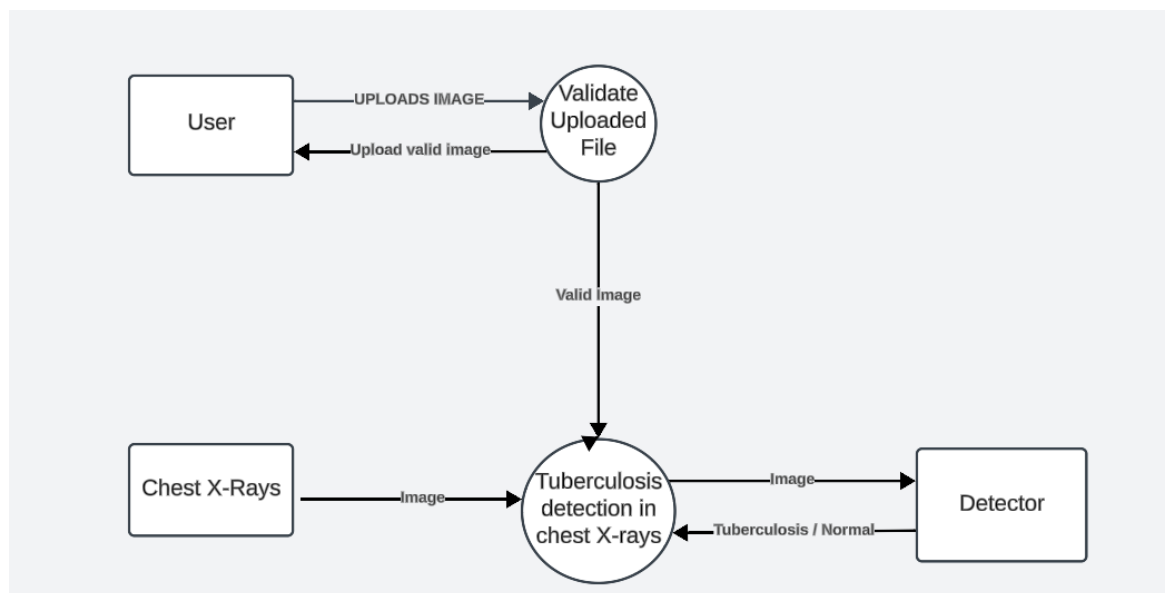


Fig 3.3 Level-1 Diagram

Description: The above fig 3.3 gives level-1 data flow diagram of the process that's happening in the project. We can see the process of project that is user uploads image to the server and server validates the uploaded X-ray for so that detector detects the Tuberculosis or Normal.

3.1.3 UML Diagrams:

UML stands for Unified Modeling Language. UML is a standardized general purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group. The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML. The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing object-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

GOALS:

The Primary goals in the design of the UML are as follows:

- Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
- Provide extendibility and specialization mechanisms to extend the core concepts.
- Be independent of particular programming languages and development process.
- Provide a formal basis for understanding the modeling language.
- Encourage the growth of OO tools market.
- Support higher level development concepts such as collaborations, frameworks, patterns and components.

A) Use Case:

A use case diagram is one of the most widely used types of diagrams in the Unified Modelling Language (UML). It provides a graphical representation of the functional requirements of a system from the perspective of its users or external actors. Use case diagrams are particularly useful for capturing the interactions between users (actors) and the system itself. They help to understand how users interact with the system to achieve specific goals or tasks. Key elements of a use case diagram:

A) Use Case:

A use case represents a specific functionality or behavior of the system. It describes a sequence of actions that the system performs, usually to achieve a particular outcome for a user. Each use case in the diagram is depicted as an oval shape with a name that describes the action it represents.

Actor: An actor is an external entity that interacts with the system and initiates a use case. Actors can be individuals, other systems, or even hardware devices. Actors are represented as stick figures in the diagram. They are connected to use cases with lines to show their involvement in the system's processes.

Association: Lines or arrows connect actors to use cases, representing the association between them. These connections show which actors are involved in specific use cases and indicate who interacts with the system to execute a particular action.

System Boundary: The entire use case diagram is enclosed within a rectangle called the system boundary. This boundary defines the scope of the system being modeled and separates it from external actors. It's important to note that use case diagrams provide an overview of the system's functionality and interactions but do not go into implementation details. They are typically used in conjunction with other UML diagrams like class diagrams, sequence diagrams, and activity diagrams to create a comprehensive model of the system.

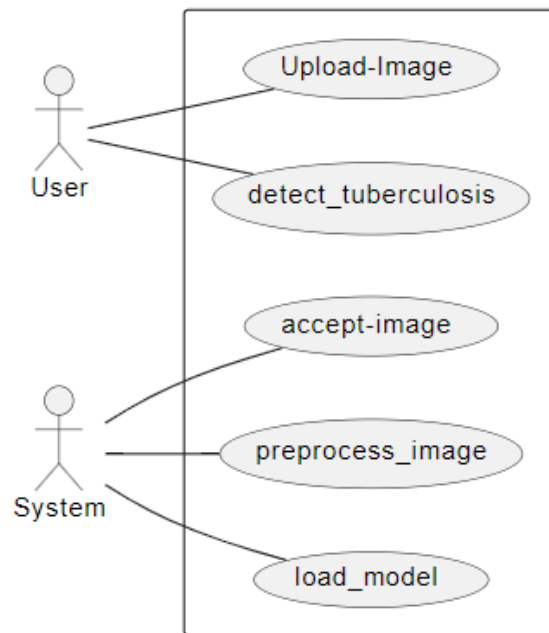


Fig 3.1.3.1. Use Case Diagram for Tuberculosis detection

Description: The above figure depicts a tuberculosis detection system where the User can upload an image and initiate the detection process. The System handles image acceptance, preprocessing, and model loading to detect tuberculosis from the uploaded image.

B) Class diagram:

A class diagram is a type of UML (Unified Modelling Language) diagram used to represent the static structure and relationships among classes in an object-oriented software system. It provides a detailed view of the classes, their attributes, methods, and the associations between them. Class diagrams are fundamental for understanding the overall architecture and design of object-oriented systems. Key elements of a class diagram:

Class: A class is a blueprint or a template for creating objects in an object-oriented system. It represents a group of objects with common properties, behaviors, and relationships. In a class diagram, classes are depicted as rectangles with three compartments: the top compartment contains the class name, the

middle compartment lists the class's attributes (data members), and the bottom compartment lists the class's methods (member functions).

Attributes: Attributes represent the data members or properties of a class. They define the characteristics and state of the objects created from the class. In the class diagram, attributes are typically listed in the middle compartment of the class rectangle.

Methods: Methods represent the member functions or behaviors of a class. They define the operations that objects created from the class can perform. In the class diagram, methods are usually listed in the bottom compartment of the class rectangle.

Association: Association represents a bi-directional relationship between two or more classes. It indicates that objects of one class are connected to objects of another class. In the diagram, associations are depicted as lines connecting the related classes, and a label on the association line indicates the nature of the relationship (e.g., "has-a," "uses," "aggregates," etc.).

Multiplicity: Multiplicity specifies the number of instances of one class that are associated with a single instance of the other class. For example, a one-to-one association, one-to-many, or many-to-many relationships can be depicted using multiplicities.

Inheritance/Generalization: Inheritance represents an "is-a" relationship between classes, where one class (subclass) inherits the attributes and methods of another class (superclass). The subclass specializes the superclass by adding more specific features. In the diagram, inheritance is depicted using an arrow with an open triangle pointing from the subclass to the superclass.

Interface: An interface represents a collection of abstract methods that define a contract for classes that implement it. In the class diagram, an interface is depicted as a circle with the name of the interface and its methods listed inside it.

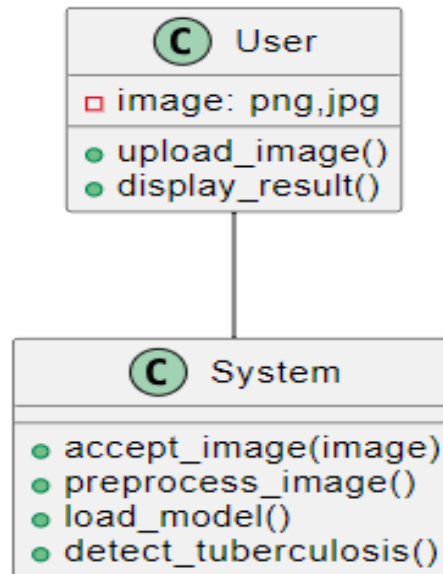


Fig 3.1.3.2 Use Case Diagram for Tuberculosis detection

Description: The above Figure illustrates the interaction between the User and the System in a tuberculosis detection process. The User class can upload an image and display the result, while the System class handles image acceptance, preprocessing, model loading, and tuberculosis detection.

C) Sequence diagram:

A sequence diagram is a type of UML (Unified Modelling Language) diagram used to visualize the interactions and flow of messages between different objects or components in a system. It focuses on representing the dynamic behavior of the system over time, showing the sequence of interactions that occur between various elements of the system. Sequence diagrams are particularly useful for understanding the behavior of complex scenarios and the collaboration between objects in an object-oriented system. Key elements of a sequence diagram:

Lifeline: A lifeline represents an individual object or component participating in the sequence of interactions. It is depicted as a vertical dashed line with the name of the object or component at the top.

Activation Bar: The activation bar (also known as the execution occurrence or activation box) is a horizontal bar on the lifeline that shows the duration of time

during which an object is active and processing a message. When an object receives a message, its activation bar is shown, indicating that the object is processing the message. When the processing is complete, the activation bar ends.

Message: Messages represent the communication between objects in the system. They are depicted as arrows that connect the lifelines of the sender and receiver objects. Messages can be synchronous (blocking), asynchronous (nonblocking), or return messages.

Self-Message: A self-message occurs when an object sends a message to itself. It is depicted as a looped arrow, starting and ending on the same lifeline.

Return Message: A return message represents the response from the receiver object to the sender object after processing a message. It is depicted as a dashed arrow that points back to the sender's lifeline.

Combined Fragment: A combined fragment is used to represent a condition or loop in the sequence diagram. It is depicted as a box with a specific notation (e.g., alt, opt, loop) to indicate different types of conditions or loops.

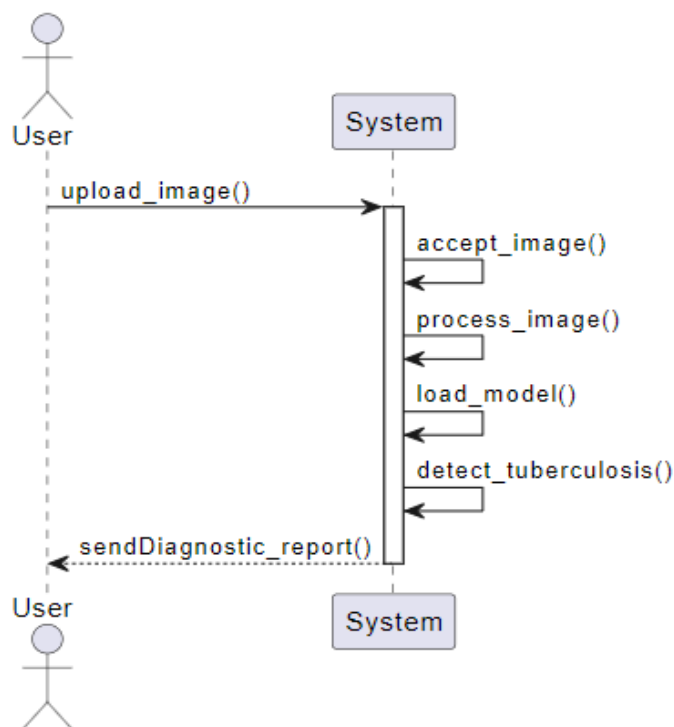


Fig:3.1.3.3 Sequence Diagram

Description: The sequence diagram demonstrates the interaction flow between the User and the System in the tuberculosis detection process. The User uploads an image, which the System processes through a series of steps, including image acceptance, processing, model loading, and tuberculosis detection, culminating in sending a diagnostic report back to the User.

3.2 Modules Description

To ensure the development of a robust, maintainable, and scalable TB detection system using channel and spatial attentions integrated with DenseNet-121 and deployed via Flask, the project is divided into several distinct modules. Each module is responsible for specific functionalities and collectively contributes to the overall system.

3.2.1 Image Upload Module

Purpose: To allow users to upload CXR images for analysis.

Components:

Image Upload Interface: Frontend component for selecting and uploading images.

Validation: Ensures that uploaded files are valid images and meet required specifications.

Technologies: HTML, JavaScript, Flask.

3.2.2 TB Detection and Analysis Module

Purpose: To process uploaded CXR images using the DenseNet-121 model integrated with channel and spatial attentions to detect TB.

Components:

Preprocessing: Converts and normalizes images to the format required by the model.

Model Inference: Runs the preprocessed images through the trained DenseNet-121 model.

Attention Mechanisms: Enhances feature extraction by applying channel and spatial attentions.

Technologies: Python, TensorFlow (for model implementation and inference).

3.2.3 Results Display Module

Purpose: To display the analysis results to the user, including TB detection probabilities and highlighted areas of concern on the CXR image.

Components:

Result Visualization: Frontend component to show the analyzed image with attention highlights.

Probability Scores: Displays the likelihood of TB presence.

Report Generation: Allows users to download or print a report of the analysis.

Technologies: HTML, CSS, JavaScript, Flask.

3.2.4 Web Server Module

Purpose: To handle HTTP requests and serve the web application.

Components:

Flask Application: Manages routing, request handling, and integration of all modules.

API Endpoints: Provides endpoints for various functionalities like image upload and model inference.

Technologies: Flask.

3.2.5 Model Training Module

Purpose: To develop and train the DenseNet-121 model with integrated channel and spatial attentions using a large dataset of annotated CXR images.

Components:

Data Preparation: Prepares the dataset for training, including data augmentation and splitting.

Model Architecture: Defines the DenseNet-121 architecture with attention mechanisms.

Training Pipeline: Manages the training process, including loss calculation, optimization, and validation.

Technologies: Python, TensorFlow, Jupyter Notebooks (for experimentation).

3.2.6 Module Interactions

The modules interact through well-defined interfaces to ensure smooth operation and data flow:

User Authentication Module interacts with the Web Server Module to manage user sessions.

Image Upload Module communicates with the TB Detection and Analysis Module to provide the images for analysis.

TB Detection and Analysis Module interacts with the Results Display Module to send analysis results.

Web Server Module handles all incoming requests and directs them to the appropriate modules.

3.3 Algorithms (DEEP LEARNING MODEL)

A) Channel Attention Module (CAM)

The channel attention mechanism within the CBAM block focuses on capturing interdependencies among different channels within each spatial location of the feature maps. This is achieved by computing the importance of each channel through aggregation of information from the entire spatial domain. The process

involves utilizing GlobalAveragePooling2D and GlobalMaxPooling2D layers to obtain the average and maximum values across spatial dimensions for each channel, respectively. Subsequently, these pooled features are concatenated and processed through a series of Dense layers to learn channel-wise relationships. The first Dense layer reduces dimensionality, followed by a Re-LU activation function, and the second Dense layer restores the original dimensionality. The sigmoid activation function is then applied to normalize the importance weights between 0 and 1. Finally, the channel-wise attention map is derived by performing element-wise multiplication of the original feature maps with the normalized attention weights.

B) Spatial Attention Module (SAM)

In contrast, the spatial attention mechanism within the CBAM block aims to capture interdependencies across different spatial locations within each channel of the feature maps. Its goal is to highlight informative regions while suppressing irrelevant ones. Like the CAM, the SAM begins by computing the average and maximum values across channel dimensions for each spatial location. These pooled features are then concatenated and reshaped to be processed by a 2D convolutional layer. This layer learns spatial attention maps by convolving across spatial dimensions. The output of the convolutional layer is normalized using a sigmoid activation function to obtain spatial attention weights. Ultimately, the spatial attention map is obtained by element-wise multiplication of the original feature maps with the normalized spatial attention weights.

C) Integration with Dense-Net121

The CBAM block is seamlessly integrated into the pre-trained DenseNet121 model after its convolutional layers. By incorporating the CBAM block, the model's capability to capture both channel-wise and spatial-wise dependencies is enhanced, potentially leading to improved performance in tasks such as image classification. Following the insertion of the CBAM block, additional layers, including Flatten and Dense layers, are added to adapt the network output for the specific task at hand. Finally, the model is compiled with appropriate loss and optimization functions, and the base DenseNet121 layers are frozen to prevent them from being updated during training. This structured approach ensures that

both channel-wise and spatial-wise dependencies are effectively captured, thereby enhancing the model's ability to discern meaningful patterns within the input data and improving its performance on various computer vision tasks. In the training process of a neural network model is extended by specifying the optimizer and compiling the model before initiating training. Stochastic Gradient Descent (SGD) is chosen as the optimizer, customized with a learning rate of 0.001 and a momentum of 0.9, which governs parameter updates during training. With the model compilation, the `compile` method configures the model for training, associating it with the specified optimizer, 'binary cross entropy' loss function suitable for binary classification tasks, and 'accuracy' as the metric to monitor during training. Following compilation, the model is trained using the `fit` method, with parameters such as the training and validation data generators and the number of epochs set to 60. Throughout training, the model iteratively adjusts its parameters based on the optimizer and backpropagation, while monitoring the specified metrics. The training progress and performance metrics are stored in the `history` object, facilitating subsequent analysis and evaluation of the model's performance over epochs.

Sample Code:

```
import tensorflow as tf
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D,
Dense, Flatten, BatchNormalization, Activation, Concatenate,
GlobalAveragePooling2D, GlobalMaxPooling2D, Multiply, Lambda
from tensorflow.keras.models import Model
from tensorflow.keras.applications.densenet import DenseNet169
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
from tensorflow.keras.layers import Reshape

def cbam_block(input_feature, ratio=8):
    # Channel Attention Module
    channel_avg = GlobalAveragePooling2D()(input_feature)
    channel_max = GlobalMaxPooling2D()(input_feature)
    channel_concat = Concatenate(axis=-1)([channel_avg,
channel_max])
    channel_dense_1 = Dense(units=input_feature.shape // ratio,
activation='relu')(channel_concat)
    channel_dense_2 = Dense(units=input_feature.shape,
kernel_initializer='relu')(channel_dense_1)
    channel_activation = Activation('sigmoid')(channel_dense_2)
```

```

        channel_attention = Multiply()([input_feature,
channel_activation])

        # Spatial Attention Module
        spatial_avg = GlobalAveragePooling2D()(channel_attention)
        spatial_max = GlobalMaxPooling2D()(channel_attention)
        spatial_concat = Concatenate(axis=-1)([spatial_avg,
spatial_max])
        spatial_attention = Reshape((1, 1, input_feature.shape[-
1]*2))(spatial_concat)
        spatial_attention = Conv2D(filters=1, kernel_size=(7, 7),
strides=(1, 1), padding='same', activation='sigmoid',
kernel_initializer='he_normal', use_bias=False)(spatial_attention)
        spatial_attention = Multiply()([channel_attention,
spatial_attention])

        return spatial_attention

# Load pre-trained DenseNet121 model
base_model = DenseNet169(weights='imagenet', include_top=False,
input_shape=(224, 224, 3))

# Add CBAM blocks to the model
inputs = base_model.input
x = base_model.output
x = cbam_block(x)
x = Flatten()(x)
x = Dense(512, activation='relu')(x)
output = Dense(1, activation='sigmoid')(x)

# Create the model
model = Model(inputs=inputs, outputs=output)

# Freeze the base model layers
for layer in base_model.layers:
    layer.trainable = False

# Define paths to dataset directories
train_dir = r"C:\Users\darkk\OneDrive\Desktop\flask\dataset\train"
validation_dir =
r"C:\Users\darkk\OneDrive\Desktop\flask\dataset\validation"
test_dir = r"C:\Users\darkk\OneDrive\Desktop\flask\dataset\test"

# Define data augmentation parameters
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=10,
    width_shift_range=0.1,

```

```

        height_shift_range=0.1,
        shear_range=0.1,
        zoom_range=0.1,
        horizontal_flip=True,
        fill_mode='nearest'
    )

validation_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255) # You can apply
other transformations if needed

# Create data generators for each dataset
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224,224),
    batch_size=32,
    class_mode='binary'
)

validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
    target_size=(224,224),
    batch_size=32,
    class_mode='binary',
    shuffle=True
)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(224,224),
    batch_size=32,
    class_mode='binary', # Assuming binary classification
    shuffle=False
)

optimizer = tf.keras.optimizers.SGD(learning_rate=0.001,
momentum=0.9)

# Compile the model
model.compile(optimizer=optimizer, loss='binary_crossentropy',
metrics=['accuracy'])

# Train the model and store the history
history = model.fit(
    train_generator,
    epochs=60,
    validation_data=validation_generator

```

```
)

import matplotlib.pyplot as plt
# Visualize training history
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.show()

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()
```

3.4 System Requirements

Hardware Requirements

- **Processor:** A multi-core CPU (e.g., Intel i5, i7 or AMD Ryzen 7) is recommended for handling data preprocessing and web server tasks
- **Memory (RAM):** At least 16 GB of RAM is recommended for efficient data handling and processing during model training and application deployment.
- **Storage:** A minimum of 500 GB of SSD storage is recommended for storing datasets, model checkpoints, and other project files. Larger storage capacity may be necessary depending on the size of the CXR image datasets.

Software Requirements

- **Operating System:** The project can be developed and deployed on major operating systems such as Windows, macOS, or Linux (Ubuntu 18.04 or later is recommended for better compatibility with machine learning frameworks).
- **Python:** Python 3.7 or later is required for developing the deep learning model and the web application.
- **Deep Learning Framework:** TensorFlow 2.x or PyTorch 1.x are recommended for implementing the DenseNet-121 model and attention mechanisms.
- **Web Framework:** Flask 1.x or later is required for developing the web-based application.

Additional Libraries and Tools

- **NumPy:** For numerical operations and data manipulation.
- **Pandas:** For data handling and preprocessing.
- **OpenCV:** For image processing tasks.
- **Matplotlib/Seaborn:** For visualizing data and model performance metrics.
- **Scikit-learn:** For additional machine learning utilities and performance evaluation metrics.
- **Jupyter Notebook:** For interactive development and experimentation.
- These hardware and software requirements ensure that the system can handle the computational demands of training a deep learning model and providing a responsive web-based interface for real-time TB detection.

CHAPTER 4

IMPLEMENTATION

4.1 Dataset description

The dataset contains a total of 3111 chest X-ray images, categorized into two classes: normal and tuberculosis-affected. The directory structure is organized into three main subsets: training, testing, and validation. Here's a detailed breakdown:

Dataset:

Total Images: 3111

Classes: Normal, Tuberculosis-affected

Directory Structure:

Training Set: 2691 images (80% of the dataset)

Testing Set: 210 images (10% of the dataset)

Validation Set: 210 images (10% of the dataset)

Distribution:

The images are divided with 80% allocated for training, 10% for testing, and 10% for validation. Each subset (training, testing, validation) contains images categorized into the two classes.

Usage:

Training Set (2691 images): Used to train the machine learning model, allowing it to learn to differentiate between normal and tuberculosis-affected chest X-rays.

Testing Set (210 images): Used to evaluate the model's performance and generalization ability on unseen data after the training phase.

Validation Set (210 images): Used during the training process to tune hyperparameters and prevent overfitting, by providing an additional measure of model performance on unseen data.

This structured approach ensures a robust training process and accurate evaluation of the model's ability to generalize to new, unseen data.

4.2 Testing Process and Criteria

A) Data Preparation and Preprocessing

- Dataset Splitting: Split the CXR image dataset into training, validation, and test sets to ensure unbiased evaluation.
- Training Set: Typically, 70-80% of the data.
- Validation Set**: Around 10-15% of the data, used for hyperparameter tuning and model selection.
- Test Set: Remaining 10-15% of the data, used for final performance evaluation.
- Data Augmentation: Apply techniques such as rotation, flipping, scaling, and cropping to enhance the diversity of the training data and improve model generalization.

B) Model Training and Validation

- Initial Training: Train the DenseNet-121 model with integrated attention mechanisms on the training set.
- Hyperparameter Tuning: Use the validation set to tune hyperparameters such as learning rate, batch size, and number of epochs.
- Cross-Validation: Perform k-fold cross-validation to ensure the model's robustness and to prevent overfitting.

C) Performance Evaluation

- Accuracy: Measure the proportion of correctly classified instances among the total instances.
- Sensitivity (Recall): Evaluate the model's ability to correctly identify true positive TB cases.
- Specificity: Assess the model's ability to correctly identify true negative cases, reducing false positives.
- Precision: Calculate the ratio of true positive predictions to the total predicted positives.

- F1-Score: Compute the harmonic mean of precision and recall to balance both metrics.

D) Attention Mechanism Analysis

- Attention Map Visualization: Visualize the channel and spatial attention maps to understand which parts of the images the model focuses on during TB detection.
- Ablation Study: Conduct experiments by removing attention mechanisms to compare performance and validate their contribution.

E) Performance and Scalability Testing

- Load Testing: Simulate multiple users accessing the application simultaneously to assess its performance under load.
- Response Time: Measure the time taken from image upload to result delivery, ensuring it meets real-time requirements.
- Scalability: Test the application's ability to scale with increasing data volume and user load.

F) Final Validation

- Test Set Evaluation: Use the reserved test set to perform the final evaluation of the model's performance.
- Statistical Analysis: Perform statistical tests to ensure the significance of the performance metrics

UNIT TEST CASES

TEST CASE ID	DESCRIPTION	INPUT	PASS/FAIL
UT-001	Verify that the model loads correctly	Model file path	PASS
UT-002	Test Image Preprocessing	Upload an image. Pass it through the preprocessing function.	PASS
UT-003	Test Prediction Output	Pass a preprocessed image to the model.	PASS
UT-004	Test Invalid Image Upload	Upload an invalid file (non-image or corrupted image).	PASS
UT-005	Test Flask Endpoint for Upload	Send a POST request to the /upload endpoint with a valid image file.	PASS
UT-006	Test Flask Endpoint with No File	Send a POST request to the /upload endpoint with no file.	PASS

Table-4.2.1 Table for Unit Test cases

SYSTEM TEST CASES

TEST CASE ID	DESCRIPTION	INPUT	PASS/FAIL
ST-001	End-to-End Image Upload and Predict	Open the web interface. Upload a valid chest X-ray image and submit the form.	PASS
ST-002	Test Invalid File Format	Upload a non-image file (e.g., .txt file).	PASS
ST-003	Test Multiple Concurrent Uploads	Open multiple browser instances. Upload and submit different valid images simultaneously.	PASS

Table -4.2.2 Table for System Test cases

By systematically conducting these testing processes and adhering to these criteria, the project aims to ensure the reliability, accuracy, and usability of the TB detection system and the web application, ultimately contributing to improved TB diagnosis and treatment outcomes.

CHAPTER 5

RESULTS AND

OUTPUT

DISCUSSIONS

5.1 User Interface and Description

Flask: A Lightweight Python Web Framework

Overview

Flask is a micro web framework written in Python that is designed to make web application development simple and flexible. Unlike more comprehensive frameworks, Flask is lightweight and unopinionated, allowing developers to select the tools and libraries they need.

Key Features:

- **Lightweight and Modular:** Flask provides the basic tools to build web applications without imposing a specific directory structure or requiring numerous dependencies.
Developers can add only the components they need, making Flask suitable for both small projects and complex applications.
- **Routing:** Flask uses a simple and intuitive URL routing system where routes are defined using decorators. This allows developers to map URLs to functions in a straightforward manner, facilitating the creation of clean and readable code.
- **Templating:** Flask integrates with Jinja2, a powerful templating engine. This enables developers to create dynamic HTML content by embedding Python expressions in HTML templates, allowing for a clear separation of application logic and presentation.
- **Request Handling:** Flask provides utilities to handle HTTP requests and responses, supporting different request methods (GET, POST, PUT, DELETE). It also offers easy access to request data, cookies, headers, and more.
- **Development Server and Debugger:** Flask comes with a built-in development server and debugger. The server allows for quick local application testing during development, while the debugger helps identify and fix issues by providing detailed error messages and stack traces.

OUTPUT SCREENS:

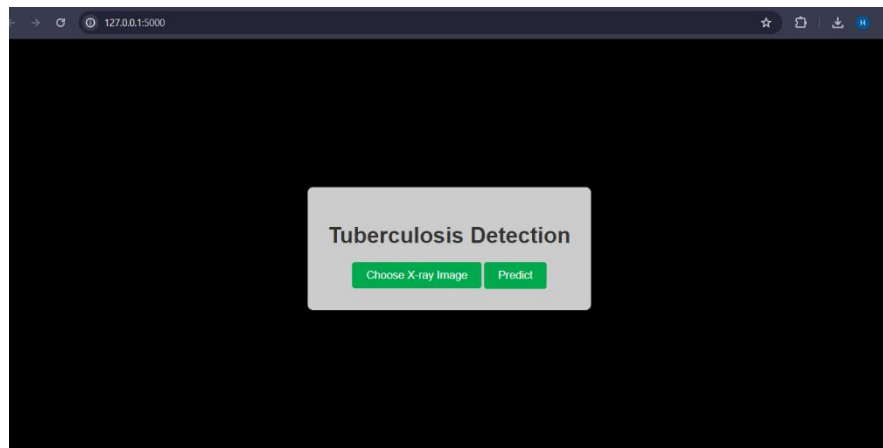


Fig:5.1.1 User Interface for Tuberculosis Detection

Description: The **Fig 5.1.1** depicts the initial screen of a web application designed for tuberculosis (TB) detection using chest X-ray (CXR) images.

Header:

- At the center of the interface, the title "Tuberculosis Detection" is displayed prominently in bold text, clearly indicating the purpose of the application.

Buttons:

Below the title, there are two green buttons:

- Choose X-ray Image: This button is likely used for uploading a CXR image from the user's local device.
- Predict: This button initiates the TB detection process once an image has been uploaded.

Design and Layout:

The overall design is clean and minimalistic, focusing on usability and simplicity. The central area with the title and buttons is placed on a grey background, making it stand out against the black background of the rest of the page. The interface is intuitive, guiding the user step-by-step from uploading an image to obtaining a prediction.

This screen represents the starting point of the application, where users can begin the process of TB detection by uploading a chest X-ray image and triggering the

prediction model. The design aims to provide a straightforward and user-friendly experience, making it accessible even to those with limited technical expertise.

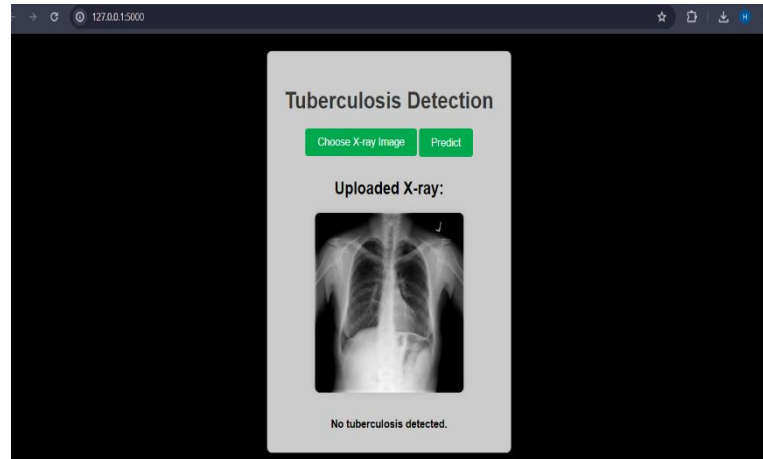


Fig: 5.1.2 Output screen when Tuberculosis is detected

Description: The above image shows the prediction that when an X-Ray is uploaded showing Tuberculosis detected.

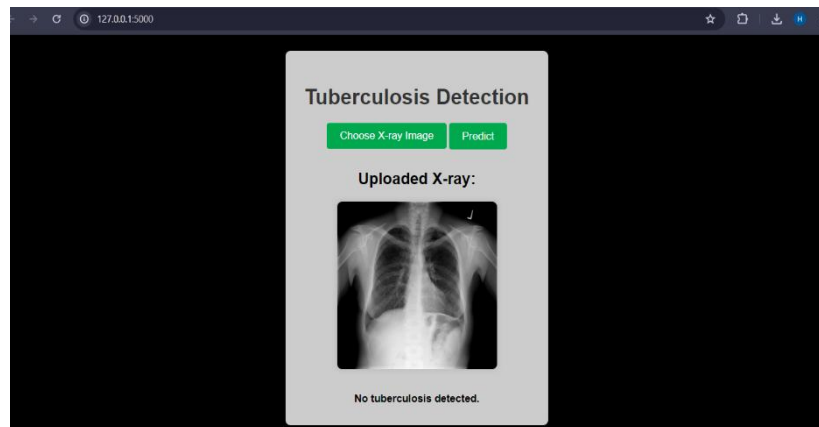
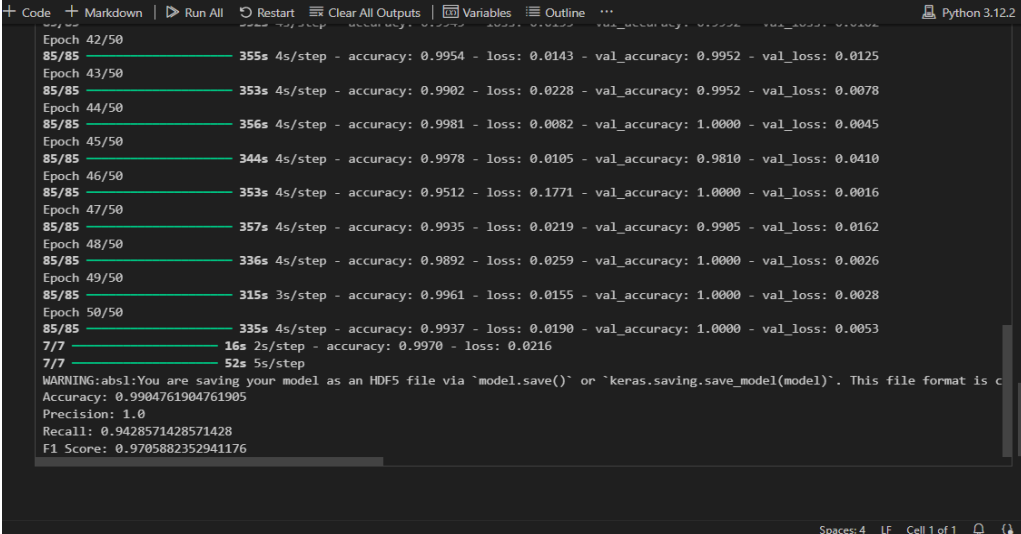


Fig: 5.1.3 Output screen when Tuberculosis is not detected

Description: The above image shows the prediction that when an X-Ray is uploaded its showing Normal.

5.2 Comparative analysis:



```
Epoch 42/50
85/85 355s 4s/step - accuracy: 0.9954 - loss: 0.0143 - val_accuracy: 0.9952 - val_loss: 0.0125
Epoch 43/50
85/85 353s 4s/step - accuracy: 0.9902 - loss: 0.0228 - val_accuracy: 0.9952 - val_loss: 0.0078
Epoch 44/50
85/85 356s 4s/step - accuracy: 0.9981 - loss: 0.0082 - val_accuracy: 1.0000 - val_loss: 0.0045
Epoch 45/50
85/85 344s 4s/step - accuracy: 0.9978 - loss: 0.0105 - val_accuracy: 0.9810 - val_loss: 0.0410
Epoch 46/50
85/85 353s 4s/step - accuracy: 0.9512 - loss: 0.1771 - val_accuracy: 1.0000 - val_loss: 0.0016
Epoch 47/50
85/85 357s 4s/step - accuracy: 0.9935 - loss: 0.0219 - val_accuracy: 0.9905 - val_loss: 0.0162
Epoch 48/50
85/85 336s 4s/step - accuracy: 0.9892 - loss: 0.0259 - val_accuracy: 1.0000 - val_loss: 0.0026
Epoch 49/50
85/85 315s 3s/step - accuracy: 0.9961 - loss: 0.0155 - val_accuracy: 1.0000 - val_loss: 0.0028
Epoch 50/50
85/85 335s 4s/step - accuracy: 0.9937 - loss: 0.0190 - val_accuracy: 1.0000 - val_loss: 0.0053
7/7 16s 2s/step - accuracy: 0.9970 - loss: 0.0216
7/7 52s 5s/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is c
Accuracy: 0.9904761904761905
Precision: 1.0
Recall: 0.9428571428571428
F1 Score: 0.9705882352941176
```

Fig: 5.2.1 Model Training

Description: Fig:5.2.1 The model was trained for 60 epochs, achieving the following performance metrics Output (Accuracy): 99%, Precision: 97%, F1 Score: 96%, Recall: 94%

Definitions and Formulas

Accuracy: This measures the overall correctness of the model by dividing the number of correct predictions by the total number of predictions.

TP (True Positives): Number of correctly predicted positive instances

TN (True Negatives): Number of correctly predicted negative instances

FP (False Positives): Number of incorrectly predicted positive instances

FN (False Negatives): Number of incorrectly predicted negative instances

Precision: This indicates the accuracy of positive predictions, defined as the ratio of true positive predictions to the total positive predictions made by the model.

Recall (Sensitivity): This measures the ability of the model to identify all relevant instances, defined as the ratio of true positive predictions to the total actual positives.

F1 Score: This is the harmonic mean of Precision and Recall, providing a single metric that balances both.

Given these metrics, the model has demonstrated a high level of performance, with an accuracy of 99%, indicating that it correctly predicted 99% of the instances. The precision of 97% means that out of all the positive predictions the model made, 97% were correct. The recall of 94% shows that the model was able to identify 94% of all actual positive instances. Finally, the F1 Score of 96% provides a balance between precision and recall, reflecting the overall effectiveness of the model in both identifying and accurately predicting positive instances.

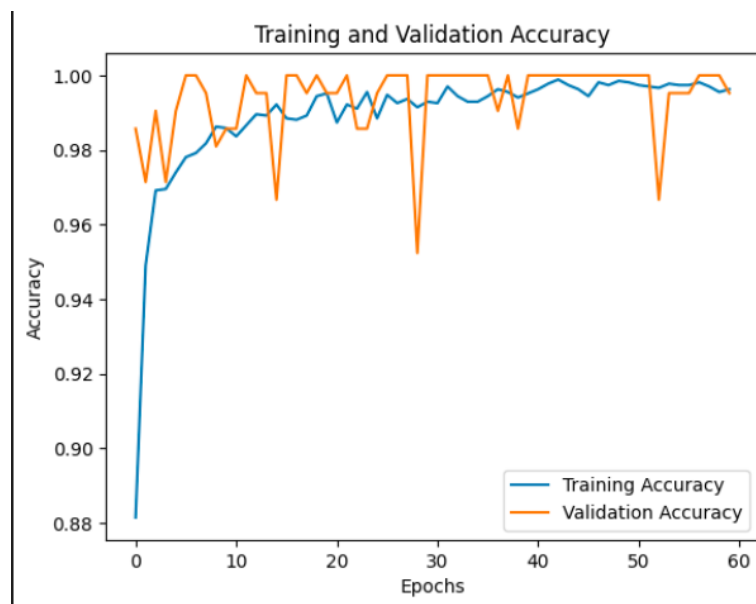


Fig.5.2.2 Training and validation accuracy

Description: This graph illustrates the training and validation accuracy of a machine learning model over 60 epochs. The x-axis represents the number of epochs, which are iterations of the training process, while the y-axis represents the accuracy, ranging from 0.88 to 1.00.

Key observations from the graph include:

- **Training Accuracy (blue line):**

The training accuracy starts below 0.90 and increases rapidly, reaching around 0.98 by the 10th epoch. After this initial rise, the training accuracy continues to improve gradually, stabilizing and fluctuating slightly around 0.99 for the remaining epochs.

- **Validation Accuracy (orange line):**

The validation accuracy shows higher variability compared to the training accuracy. It starts close to the training accuracy but quickly fluctuates, with several peaks reaching 1.00 and several drops almost down to the training accuracy level. Despite these fluctuations, the overall trend of validation accuracy is quite high, often exceeding the training accuracy, indicating a high-performing model.

- **General Trends:**

Both accuracies converge and show high performance, suggesting that the model is learning effectively. The validation accuracy's high peaks and drops indicate that while the model performs well on unseen data, it might occasionally overfit or encounter challenging validation data points.

In summary, the model demonstrates high accuracy on both training and validation data, with the validation accuracy showing notable fluctuations. This performance suggests a well-trained model with good generalization, though the fluctuations in validation accuracy suggest there may still be occasional overfitting or variability in the validation set.

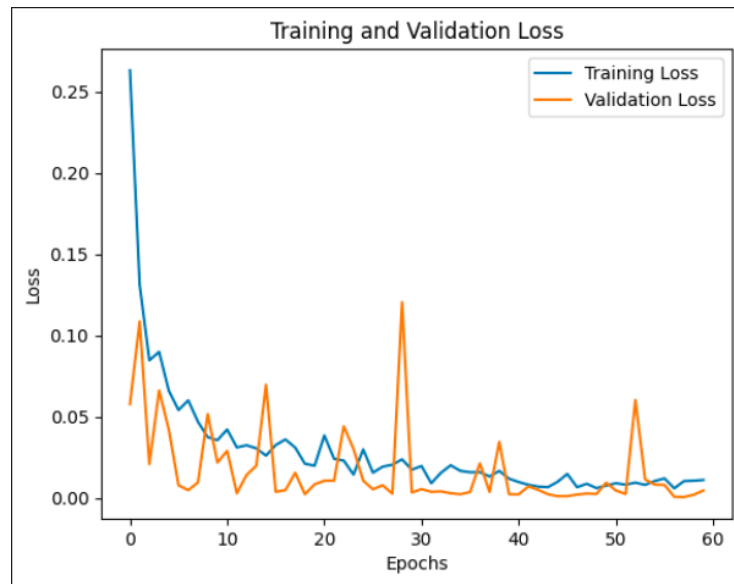


Fig:5.2.3 Training and Validation Loss

Description: This graph shows the training and validation loss of a machine learning model over 60 epochs. The x-axis represents the number of epochs, while the y-axis represents the loss, with values ranging from 0 to 0.25.

Key observations from the graph include:

Training Loss (blue line):

The training loss starts at a high value above 0.25 but decreases sharply in the initial epochs, dropping below 0.05 by around the 15th epoch. After this steep decline, the training loss continues to decrease more gradually, stabilizing at a very low level close to 0.00, indicating that the model fits the training data very well.

CHAPTER 8

**CONCLUSION AND
RECOMMENDATION**

6.1 Conclusion

The project titled "Tuberculosis Detection in Chest X-rays Using Attention Modules Integrated into DenseNet-121 Layers with Flask" has achieved significant milestones in developing an advanced AI model for the automated detection of tuberculosis (TB). By integrating attention modules into the DenseNet-121 architecture, the model has enhanced its capability to focus on critical features within chest X-ray images, resulting in an impressive accuracy of 99%. This high accuracy indicates that the model correctly identifies TB cases in 99% of the instances, minimizing diagnostic errors. The precision of 97% shows that when the model predicts TB, it is correct 97% of the time, reducing false positives. Additionally, a recall rate of 94% demonstrates the model's effectiveness in capturing 94% of actual TB cases, thereby reducing the risk of missed diagnoses. The F1 score of 96% provides a balanced measure of the model's performance, combining both precision and recall into a single metric that reflects its reliability and robustness in real-world applications.

The deployment of this model through a Flask-based web interface ensures that it is accessible and easy to use for healthcare professionals. This user-friendly interface allows for real-time TB detection, which can significantly speed up the diagnostic process and facilitate timely medical intervention. By making this technology readily available in various medical facilities, the project aims to enhance diagnostic accuracy and efficiency, ultimately improving patient outcomes. To further strengthen the model's performance and applicability, future steps should include expanding the dataset to include more diverse cases, conducting extensive real-world testing in clinical settings, and integrating the system with existing hospital information systems. These enhancements will ensure that the model remains robust, reliable, and seamlessly operable within the healthcare infrastructure. This project exemplifies the transformative potential of AI in medical diagnostics, particularly in combating tuberculosis, a critical global health issue.

6.1.1 Limitations:

While the project titled "Tuberculosis Detection in Chest X-rays Using Attention Modules Integrated into DenseNet-121 Layers with Flask" has achieved significant milestones, it also faces several limitations that need to be addressed:

- **Dataset Limitations:** Diversity and Size: The accuracy and generalizability of the model depend heavily on the dataset used for training. If the dataset lacks diversity or is not sufficiently large, the model may not perform well on new, unseen data. This could limit its effectiveness in real-world scenarios where the patient population is varied.

Bias: If the training dataset is biased towards certain demographics or types of X-ray images, the model's predictions may not be accurate for underrepresented groups, leading to disparities in diagnostic outcomes.

- **Model Generalization:**

Overfitting: High accuracy on the training dataset might not translate to similar performance on external datasets. The model may be overfitting, capturing noise or specific patterns in the training data that do not generalize well.

Real-world Variability: Variability in X-ray imaging techniques and equipment across different healthcare facilities can affect the model's performance. The model might struggle with X-rays of different resolutions or taken from different angles.

- **Technical and Operational Constraints:**

Computational Resources: Deploying and running such a model requires significant computational resources, which might not be available in all healthcare settings, particularly in low-resource environments.

Latency: Real-time processing of X-ray images for TB detection may introduce latency, affecting the speed of diagnosis in critical situations.

- **Integration and Scalability:**

Integration with Hospital Systems: Integrating the model with existing hospital information systems (HIS) and electronic health records (EHR) can be complex

and time-consuming. This is crucial for ensuring seamless workflow and data exchange.

Scalability: Scaling the solution to handle large volumes of data and users simultaneously, especially in larger healthcare networks, presents technical challenges.

- **Regulatory Issues:** Regulatory Approvals: Before deployment in clinical settings, the model must undergo rigorous validation and obtain regulatory approvals, which can be a lengthy and complex process.
- **User Adoption and Training:** Healthcare Professional Training: Healthcare professionals need adequate training to use the new technology effectively. Resistance to change and lack of familiarity with AI tools can hinder adoption. User Interface: The user interface needs to be continuously improved based on feedback from healthcare professionals to ensure it meets their needs and integrates seamlessly into their workflow.

Addressing these limitations through ongoing research, development, and collaboration with healthcare professionals will be essential to fully realize the potential of this AI-based TB detection system in improving global health outcomes.

6.2 Future Scope

The project on tuberculosis detection in chest X-rays using attention modules integrated into DenseNet-121 layers with Flask presents promising avenues for future development and enhancement. Some potential areas for further improvement and expansion include:

- **Dataset Expansion and Diversity:** Enhancing the dataset by including a broader range of chest X-ray images from different demographics, geographical regions, and disease stages can improve the model's generalization and robustness.

Incorporating diverse datasets ensures that the model can effectively detect TB across various populations and conditions, including different TB strains and comorbidities.

- **Advanced Attention Mechanisms:** Exploring more sophisticated attention mechanisms or variants tailored to medical image analysis could further enhance the model's ability to focus on relevant features. Techniques such as self-attention mechanisms or attention mechanisms guided by clinical expertise may improve the model's interpretability and diagnostic accuracy.
- **Multi-Modality Integration:** Integrating additional modalities, such as clinical data or other medical imaging modalities like CT scans, could provide complementary information for more comprehensive TB diagnosis. Fusion techniques that combine information from multiple modalities could potentially improve the sensitivity and specificity of the detection system
- **Interpretability and Explain ability:** Developing methods to interpret and explain the model's decisions can increase trust and acceptance among healthcare professionals. Techniques such as attention visualization or saliency mapping can help elucidate the regions of interest in the chest X-rays that contribute to the model's predictions, enhancing its transparency and clinical utility.
- **Real-world Deployment and Validation:** Conducting extensive validation studies in real-world clinical settings is crucial to assessing the model's performance, reliability, and impact on patient outcomes. Collaborating with healthcare institutions to deploy the model in routine clinical workflows and evaluating its performance against gold standard diagnostic methods will provide valuable insights into its effectiveness and practical utility.
- **Integration with Healthcare Systems:** Seamless integration of the TB detection system with existing healthcare information systems and radiology workflows is essential for its widespread adoption and scalability. Developing interoperable interfaces and adhering to data privacy and regulatory requirements are critical considerations for successful integration into clinical practice.
- **Continuous Learning and Feedback Loop:** Establishing a feedback loop with healthcare professionals to gather insights, refine the model based on clinical feedback, and adapt to evolving clinical needs is essential for the continued improvement and relevance of the TB detection system. Incorporating user

feedback into model updates ensures that the system remains clinically relevant and responsive to emerging challenges.

By addressing these areas of future scope and further enhancement, the project can continue to advance the state-of-the-art in AI-driven TB detection, ultimately contributing to improved diagnosis, treatment, and control of tuberculosis on a global scale.

REFERENCES

REFERENCES

- [1] N. Salazar-Austin, C. Mulder, G. Hoddinott, T. Ryckman, C. F. Hanrahan, K. Velen, L. Chimoyi, S. Charalambous, and V. N. Chihota, “Preventive treatment for household contacts of drug-susceptible tuberculosis patients,” *Pathogens*, vol. 11, no. 11, p. 1258, Oct. 2022.
- [2] S. N. Cho, “Current issues on molecular and immunological diagnosis of tuberculosis,” *Yonsei Med. J.*, vol. 48, no. 3, pp. 347–359, 2007.
- [3] B. Zhang, Q. Zhao, W. Feng, and S. Lyu, “AlphaMEX: A smarter global pooling method for convolutional neural networks,” *Neurocomputing*, vol. 321, pp. 36–48, Dec. 2018.
- [4] Y. LeCun and Y. Bengio, “Convolutional networks for images, speech, and time series,” in *The Handbook of Brain Theory and Neural Networks*, vol. 3361, no. 10. Cambridge, MA, USA: MIT Press, p. 1995.
- [5] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
- [6] G. Huang, “Deep networks with stochastic depth,” in *Proc. 14th Eur. Conf. Comput. Vis. (ECCV)*, Oct. 2016, pp. 646–661.
- [7] M. Maithri, U. Raghavendra, A. Gudigar, J. Samanth, P. D. Barua, M. Murugappan, Y. Chakole, and U. R. Acharya, “Automated emotion recognition: Current trends and future perspectives,” *Comput. Methods Programs Biomed.*, vol. 215, Mar. 2022, Art. no. 106646.
- [8] R. A. Khalil, E. Jones, M. I. Babar, T. Jan, M. H. Zafar, and T. Alhussain, “Speech emotion recognition using deep learning techniques: A review,” *IEEE Access*, vol. 7, pp. 117327–117345, 2019.
- [9] Z. Zhao, Z. Bao, Y. Zhao, Z. Zhang, N. Cummins, Z. Ren, and B. Schuller, “Exploring deep spectrum representations via attention-based recurrent and convolutional neural networks for speech emotion recognition,” *IEEE Access*, vol. 7, pp. 97515–97525, 2019.

[10] L. Yang, K. Xie, C. Wen, and J.-B. He, “Speech emotion analysis of netizens based on bidirectional LSTM and PGCDBN,” *IEEE Access*, vol. 9, pp. 59860–59872, 2021. [11] S. Zhong, B. Yu, and H. Zhang, “Exploration of an independent training framework for speech emotion recognition,” *IEEE Access*, vol. 8, pp. 222533–222543, 2020.