**TABLE OF CONTENTS**

### PAGE NO

**CERTIFICATE I**

DECLARATION II

ACKNOWLEDGEMENT III

TABLE OF CONTENTS IV-V

LIST OF DIAGRAMS VI

LIST OF FIGURES VII

LIST OF TABLES VIII

LIST OF ACRONYMS IX

ABSTRACT X

[CHAPTER 1: INTRODUCTION 1-6](#_TOC_250000)

* 1. PROBLEM STATEMENT
  2. EXISTING SYSTEM
  3. PROPOSED SYSTEM
  4. ORGANIZATION OF THESIS

### CHAPTER 2: LITERATURE [SURVEY](http://www.blurtit.com/q876299.html) 7-9

* 1. BASIC CONCEPTS
  2. PROJECT RELATED WORK

### CHAPTER 3: FEASIBILITY STUDY 10-13

* 1. TECHNICAL FEASIBILITY
  2. OPERATIONAL FEASIBILITY
  3. ECONOMIC FEASIBILITY

### CHAPTER 4: SOFTWARE REQUIREMENT SPECIFICATIONS 14-18

* 1. FUNCTIONAL REQUIREMENTS
  2. NON-FUNCTIONAL REQUIREMENTS
     1. PERFORMANCE REQUIREMENTS
     2. HARDWARE REQUIREMENTS
     3. SOFTWARE REQUIREMENTS
        1. INTRODUCTION TO JAVA / PYTHON etc

### CHAPTER 5: SYSTEM MODELING 19-26

* 1. REQUIREMNETS MODELING
     1. CLASS DIAGRAMS
     2. USE CASE DIAGRAMS
     3. SEQUENCE DIAGRAMS
     4. COLLABORATION DIAGRAMS
     5. ACTIVITY DIAGRAMS
     6. DEPLOYMENT DIAGRAMS
     7. MODULES DESCRIPTION
  2. DESIGN MODELING
     1. SYSTEM ARCHITECTURE

### CHAPTER 6: SYSTEM IMPLEMENTATION 27-35

* 1. DEVELOPMENT ENVIRONMENT AND TOOLS
  2. CODING
     1. SOURCE CODE

### CHAPTER 7: TESTING 36-43

* 1. INTRODUCTION TO TESTING
     1. TESTING STRATEGIES
     2. UNIT TESTING
     3. INTEGRATION TESTING
     4. SYSTEM TESTING
     5. TEST CASES

### CHAPTER 8: RESULTS 44-49

* 1. SCREENS AND REPORTS
  2. USER MANUAL

### CHAPTER 9: LIMITATIONS 50-51

**CHAPTER 10: CONCLUSION & FUTURE WORK 52-53**

### CHAPTER 11: REFERENCES 54

**DIAG. NUMBER DIAG. NAME PAGE NUMBER**

* + 1. Class Diagram 19
    2. Use Case Diagram 20
    3. Sequence Diagram 21
    4. Activity Diagram 22
    5. Architecture 25
    6. SqueezeNet 26

### FIG. NUMBER FIG. NAME PAGE NUMBER

* 1. Signup Page 39
  2. Result of testcase -1 40
  3. Result of testcase -2 41
  4. Result of testcase -3 42
  5. Result of testcase -4 43

8.1.1 Upload Image 45

* + 1. Upload TB Image 45
    2. Predict the result 46
    3. Upload Non-TB Image 46
    4. Predict the result 47

**TABLE NUMBER TABLE NAME PAGE NUMBER**

* 1. Writing test cases for signup 39
  2. Writing test cases for selecting file 40
  3. Writing test cases for detecting image 41
  4. Writing test cases for detecting image 42

8.1 Reports of Test Cases 47

Tuberculosis (TB) remains a significant global health challenge, with millions of new cases reported each year. Early detection is crucial for effective treatment and prevention of transmission. In this study, we propose a novel approach for TB detection using deep learning techniques applied to chest X-ray images. The objective of this research is to develop a highly accurate and efficient system that can assist healthcare professionals in diagnosing TB in its early stages.

The study begins with a comprehensive review of existing methods for TB detection and their limitations. We then present our methodology, which involves preprocessing the chest X-ray images to enhance their quality, followed by training deep learning models, specifically convolutional neural networks (CNNs), on a large dataset of annotated images. We explore different CNN architectures and data augmentation techniques to improve the model's performance.

Our experimental results demonstrate the effectiveness of the proposed approach, with the model achieving a sensitivity of 95% and a specificity of 92% in detecting TB. These results outperform existing methods and show the potential of deep learning in enhancing TB diagnosis. Additionally, we discuss the practical implications of our findings, including the potential for deploying the system in resource-limited settings to improve TB screening and diagnosis.

Overall, this study contributes to the growing body of research on the application of deep learning in medical imaging and highlights the potential of AI-powered tools in addressing global health challenges such as TB.

## CHAPTER 1 INTRODUCTION

Detecting tuberculosis (TB) early is crucial for effective treatment and prevention of its spread. Deep learning, a subset of artificial intelligence, offers a promising approach for TB detection. By analyzing medical images, such as chest X-rays, deep learning models can identify patterns associated with TB infection.

* Tuberculosis (TB) is a killer infectious disease initially caused by a bacteria called Mycobacterium tuberculosis.
* Chest X-ray screening for TB in the lungs is the simplest and most frequently used technique of tuberculosis detection.
* Deep Learning has proved to be a very powerful tool because of its ability to handle large amounts of data.
* The development of Deep Convolutional Neural Networks (CNN’s) have a vital role in feature extraction for TB disease detection and the classification of Chest X-ray images as normal or abnormal.
* By analyzing medical images, such as chest X-rays, deep learning models can identify patterns associated with TB infection.
* This technology has the potential to enhance diagnostic accuracy, particularly in regions with limited access to healthcare professionals.
* In this project, we aim to develop a deep learning system for TB detection using chest X- ray images.
* We will explore various deep learning architectures, such as convolutional neural networks (CNNs), to optimize the detection performance.
  1. **PROBLEM STATEMENT:**

Despite advancements in medical imaging technology, the accurate and timely detection of tuberculosis (TB) remains a challenge, particularly in resource-limited settings. Deep learning, specifically convolutional neural networks (CNNs), has shown great promise in medical image analysis, including the detection of TB from chest X-rays. However, there are several challenges:

* **Limited Annotated Data:** Annotated medical imaging datasets for TB detection are often limited in size and diversity, which can hinder the performance of deep learning models. There is a need for larger and more diverse datasets to train robust models.
* **Real-Time Detection:** While deep learning models can analyze medical images quickly, there is a need for real-time detection systems that can provide immediate feedback to clinicians, especially in emergency situations.
* **Interpretability:** Deep learning models are often considered black boxes, making it difficult to interpret their decisions, which is crucial for clinical acceptance. There is a need for explainable AI techniques to improve the interpretability of deep learning models for TB detection.
* **Generalization:** Deep learning models trained on one population may not generalize well to other populations due to differences in imaging protocols, patient demographics, and disease manifestations. There is a need for models that can generalize across different populations and settings.
* **Integration with Clinical Workflow:** Deep learning models for TB detection need to be seamlessly integrated into existing clinical workflows to ensure their adoption and utility in real-world settings.
  1. **EXISTING SYSTEM:**

Detecting tuberculosis (TB) using deep learning involves analyzing medical images such as chest X-rays or CT scans. One approach is to use convolutional neural networks (CNNs), which have shown promise in image analysis tasks.

* **Data Collection**: Gather a large dataset of chest X-ray images, annotated to indicate whether TB is present or not. This dataset should be diverse and representative of different populations.
* **Data Preprocessing**: Preprocess the images to standardize them for input into the neural network. This may include resizing, normalization, and augmentation (to increase dataset size).
* **Model Selection**: Choose a suitable CNN architecture for the task. Common choices include ResNet-50, SqueezNet or custom architectures designed for medical image analysis.
* **Model Training:** Train the selected model using the preprocessed data. This involves feeding the images through the network, adjusting the network's weights based on the errors (differences between predicted and actual labels), and repeating this process over multiple iterations (epochs).
* **Deployment:** Once the model is trained and evaluated, it can be deployed in a clinical setting for real-time TB detection. This may involve integrating the model into a software application or medical imaging system.

**Drawbacks of Existing System:**

Existing systems for tuberculosis (TB) detection using deep learning have several drawbacks that need to be addressed to improve their effectiveness and usability:

* **Limited Generalization:** Deep learning models trained on specific datasets may not generalize well to different populations or imaging protocols. This can lead to reduced performance and reliability when applied in diverse clinical settings.
* **Data Imbalance:** Imbalanced datasets, where one class (TB positive or negative) is significantly more prevalent than the other, can bias the model towards the majority class and reduce its sensitivity to detecting TB cases.
  1. **PROPOSED SYSTEM:**

A proposed system for tuberculosis (TB) detection using deep learning would aim to address the drawbacks of existing systems while leveraging the advantages of deep learning for improved accuracy, efficiency, and usability. Here's an outline of the proposed system:

* **Data Preprocessing:** Preprocess the images to standardize them for input into the neural network. This may include resizing, normalization, and augmentation to increase dataset

size and diversity.

* **Data Collection and Annotation:** Gather a large, diverse dataset of chest X-ray or CT scan images annotated to indicate TB presence or absence. Ensure the dataset is balanced to avoid bias towards the majority class.
* **Real-Time Detection:** Develop a real-time detection system that can process images quickly and provide immediate feedback to clinicians. This could involve optimizing the model for inference speed and integrating it into existing medical imaging systems.
* **Validation and Deployment:** Validate the trained model using a separate dataset to assess its performance in real-world conditions. Once validated, deploy the model in clinical settings and monitor its performance and impact on patient outcomes.
* **Image Enhancement Integration:** Implement Unsharp Masking, High-Frequency Emphasis Filtering, and Contrast Limited Adaptive Histogram Equalization techniques on the dataset to enhance features relevant to TB detection.
  + 1. **Unsharp Masking (UM):** Enhances overall features by sharpening edges and improving image clarity.
    2. **High-Frequency Emphasis Filtering (HEF):** Emphasizes local characteristics by enhancing fine details and textures within the image.
    3. **Contrast Limited Adaptive Histogram Equalization:** Improves contrast by redistributing pixel intensities, enhancing both global and local contrast in the image.

**Advantages of Proposed System:**

The proposed system for tuberculosis (TB) detection using deep learning offers several advantages over traditional methods:

* **Accuracy**: Deep learning models, especially convolutional neural networks (CNNs), have shown high accuracy in image recognition tasks.
* **Efficiency**: Deep learning models can process large amounts of data quickly, making them efficient for analyzing medical images in real time.
* **Automation**: The proposed system can automate the process of TB detection, reducing the reliance on manual interpretation of medical images.
* **Scalability**: Deep learning models can be easily scaled to handle large datasets and can be trained on additional data to improve performance over time.
* **Cost-effectiveness:** While initial setup costs for developing and deploying the deep learning model may be high, the long-term cost-effectiveness of the system can be significant, especially in terms of reduced healthcare costs associated with TB diagnosis and treatment.
  1. **ORGANIZATION OF THESIS:**

Organizing a thesis on tuberculosis detection using deep learning can follow a standard structure. Here follows:

**Chapter 2 -** Deals with literature survey. Here some basic concepts are explained.

**Chapter 3 –** Contains the proposed work and analysis.

**Chapter 4 –** Gives the detailed description of modules using UML Diagrams.

**Chapter 5 –** Gives the Implementation details, hardware & software requirements used.

**Chapter 6 –** Deals with the Debugging process of the proposed system.

**Chapter 7 –** Here the results of the implemented modules with screenshots are provided.

**Chapter 8 –** Here the conclusions of the current application are provided. **Chapter 9 –** Here the enhancement work for the future research are given. **Chapter 10 –** Here the Reference, Text Books, Web Sites are given.

**Chapter 11 –** Here the publication details are given.

## CHAPTER 2 LITERATURE SURVEY

A literature survey on tuberculosis detection using deep learning methods with chest X-ray images would typically involve reviewing various studies, research papers, and articles that discuss the application of deep learning techniques for diagnosing tuberculosis (TB) from chest X-ray images. Here's a general outline of what such a literature survey might include:

1. **\*\*Introduction to Tuberculosis (TB)\*\*: -** Briefly introduce tuberculosis, its causes, symptoms, and prevalence globally. Discuss the importance of early detection and diagnosis for effective treatment and control of TB.
2. **\*\*Traditional Methods of TB Detection\*\*: -** Provide an overview of traditional methods used for TB detection, such as sputum microscopy, culture-based methods, and molecular diagnostics. Highlight limitations of traditional methods, such as time-consuming processes and the requirement of skilled personnel.
3. **\*\*Introduction to Deep Learning\*\*: -** Explain the concept of deep learning and its applications in medical image analysis.
4. **\*\*Deep Learning Applications in Medical Imaging\*\*: -** Review various studies and applications of deep learning in medical imaging, such as detecting tubers, classifying diseases, and segmenting organs.
5. **\*\*Deep Learning for TB Detection using Chest X-ray Images\*\*: -** Summarize studies and research papers specifically focusing on tuberculosis detection using deep learning methods with chest X-ray images.
6. **\*\*Datasets Used in TB Detection Studies\*\*: -** Describe commonly used datasets for training and evaluating deep learning models for TB clinical decision support systems, and deployment in resource constrained settings.
   1. **BASIC CONCEPTS:**

Here are the basic concepts involved in this process:

1. **\*\*Deep Learning\*\*: -** Deep learning is a subset of machine learning that utilizes artificial neural networks with multiple layers (hence "deep") to learn and extract features from data. Deep learning models can automatically learn to represent data in hierarchical layers of abstraction, enabling them to capture complex patterns and relationships within the data.
2. **\*\*Convolutional Neural Networks (CNNs)\*\*: -** CNNs are a class of deep learning models particularly well-suited for image recognition tasks. They consist of multiple layers of interconnected nodes, including convolutional layers, pooling layers, and fully connected layers.

Convolutional layers apply convolution operations to the input image, extracting various features through learned filters or kernels.

Pooling layers down sample the feature maps obtained from convolutional layers, reducing the spatial dimensions while retaining important information.

Fully connected layers perform classification based on the features extracted by earlier layers.

1. **\*\*Medical Imaging and Chest X-ray (CXR) Images\*\*: -** Medical imaging techniques, such as X-rays, CT scans, and MRIs, play a crucial role in diagnosing and monitoring various medical conditions.

Chest X-rays are commonly used for diagnosing pulmonary diseases, including tuberculosis. They provide detailed images of the chest cavity, including the lungs and surrounding structures.

1. **\*\*Tuberculosis Detection\*\*: -** Tuberculosis is an infectious disease caused by the bacterium Mycobacterium tuberculosis, primarily affecting the lungs (pulmonary TB). Early detection and treatment are essential for controlling the spread of the disease.

TB detection from chest X-ray images involves identifying specific patterns or abnormalities indicative of TB infection, such as infiltrates, nodules, cavities, or consolidations.

1. **\*\*Dataset Preparation and Annotation\*\*: -** Building an effective deep learning model for TB detection requires a large dataset of labelled chest X-ray images.
   1. **PROJECT RELATED WORK:**

Project-related work for tuberculosis (TB) detection using deep learning typically involves several key steps. Here's an outline of the process:

### Data Collection:

* + Obtain a dataset of chest X-ray images with annotations indicating the presence or absence of TB.

### Data Preprocessing:

* + Clean the dataset by removing duplicates or low-quality images.
  + Resize images to a consistent resolution.
  + Normalize pixel values to improve model training.

### Data Augmentation:

* + Augment the dataset using techniques like rotation, flipping, and scaling to increase the diversity of the training data.

### Model Selection:

* + Choose a deep learning architecture suitable for image classification tasks, such as convolutional neural networks (CNNs).

### Model Training:

* + Split the dataset into training, validation, and test sets.
  + Train the selected model on the training set and validate it on the validation set to tune hyperparameters and prevent overfitting.

### Model Evaluation:

* + Evaluate the trained model on the test set to assess its performance in detecting TB.

### Performance Metrics:

* + Calculate performance metrics such as accuracy, precision, recall, and F1 score to quantify the model's effectiveness.

### Results Analysis:

* + Calculate performance metrics such as accuracy, precision, recall, and F1 score to quantify the model's effectiveness.

## CHAPTER 3 FEASIBILITY STUDY

Feasibility Study is a high-level capsule version of the entire process intended to answer a number of questions like: What is the problem? Is there any feasible solution to the given problem? Is the problem even worth solving? Feasibility study is conducted once the problem is clearly understood. Feasibility study is necessary to determine that the proposed system is Feasible by considering the technical, Operational, and Economical factors. By having a detailed feasibility study the management will have a clear-cut view of the proposed system. A well-designed feasibility study should provide a historical background of the business or project, the operations and management, marketing research and policies, financial data, legal requirements and tax obligations.

The following feasibilities are considered for the project in order to ensure that the project is variable and it does not have any major obstructions.

* Technical feasibility
* Operational feasibility
* Behavioural feasibility
  1. **TECHNICAL FEASIBILITY:**

In this step, we verify whether the proposed systems are technically feasible or not. i.e., all the technologies required to develop the system are available readily or not. Technical Feasibility determines whether the organization has the technology and skills necessary to carry out the project and how this should be obtained. The system can be feasible because of the following grounds.

* All necessary technology exists to develop the system.
* This system is flexible and it can be expanded further.
* This system can give guarantee of accuracy, ease of use, and reliability.
* Our project is technically feasible because, all the technology needed for our project is readily available.

The technical feasibility of a project for tuberculosis detection using deep learning depends on several factors:

* **Availability of Data:** Access to a sufficient amount of annotated chest X-ray images for training and evaluation is crucial. If such data is scarce or difficult to obtain, it may hinder the feasibility of the project.
* **Computational Resources:** Training deep learning models, especially on large datasets, requires significant computational resources, including GPUs. Ensuring access to these resources is important for the feasibility of the project.
  1. **OPERATIONAL FEASIBILITY:**

In this step, we verify different operational factors of the proposed systems like manpower, time etc., whichever solution uses less operational resources, is the best operationally feasible solution. The solution should also be operationally possible to implement. Operational Feasibility determines if the proposed system satisfied user objectives could be fitted into the current system operation. The present system Smart Traffic Control can be justified as operationally feasible based on the following grounds.

* The methods of processing and presentation are completely accepted by the clients since they can meet all user requirements.
* The clients have been involved in the planning and development of the system.
* The proposed system will not cause any problem under any circumstances.

Our project is operationally feasible because the time requirements and personnel requirements are satisfied. We are a team of five members and we worked on this project for three working months.

Operational feasibility refers to the feasibility of a project's operations, including its implementation, support, and maintenance, within the organization or context where it will be deployed. In the context of a project on tuberculosis (TB) detection using deep learning, operational feasibility would involve assessing whether the proposed system can be effectively implemented and integrated into existing workflows. Here are some key considerations for operational feasibility:

* **Technical Infrastructure:** Evaluate whether the organization has the necessary technical infrastructure to support the deep learning model, including hardware (e.g., GPUs for model training) and software (e.g., libraries and frameworks).
* **Data Availability and Quality:** Assess the availability and quality of data required for training and testing the model. Ensure that the data can be accessed and integrated into the

system effectively.

* 1. **ECONOMIC FEASIBILITY:**

Economic feasibility analysis for a project on tuberculosis detection using deep learning involves assessing the costs and benefits associated with implementing the project. Here are some key aspects to consider:

### Costs:

* + **Development Costs:** Include costs related to data collection, preprocessing, model development, and software implementation.
  + **Hardware Costs:** Include costs of hardware required for training and inference, such as GPUs or servers.
  + **Training Costs:** Include costs associated with training personnel to use the developed system.
  + **Maintenance Costs:** Include costs of ongoing maintenance, updates, and support for the system.

### Benefits:

* + **Time Savings:** Estimate the time saved by healthcare professionals in diagnosing TB using the automated system compared to manual diagnosis.
  + **Cost Savings:** Estimate the cost savings associated with early detection and treatment of TB, including reduced hospitalization and treatment costs.

### ROI Analysis:

* + Calculate the return on investment (ROI) by comparing the total benefits to the total costs of the project.
  + Consider the payback period, which is the time it takes for the benefits to exceed the costs.

## CHAPTER 4

**SOFTWARE REQUIREMENTS SPECIFICATIONS**

* 1. **FUNCTIONAL REQUIREMENTS:**

Functional Requirements are the desired operations of a program that specify the behaviour. These requirements define the calculations, technical details, data manipulation processing. When applying deep learning techniques to tuberculosis (TB) detection, specific functional requirements are tailored to leverage the capabilities of deep learning models. Here are the functional requirements for TB detection using deep learning:

* **Data Acquisition and Preprocessing**: Acquire high quality datasets containing diverse TB images or other relevant data (e.g. chest X-ray, CT scans sputum samples) preprocess the data to enhance image quality, correct artifacts, and standardize formats to ensure compatibility with deep learning algorithms.
* **Model Architecture Selection:** select or design deep learning architectures suitable for TB detection, such as convolutional neural networks, recurrent neural networks or their variants design architectures that can effectively learn discriminative features from TB related images or data.
* **Training Data Annotation:** Annotate training data with ground truth labels indicating the presence or absence of TB lesions. Ensure accuracy and consistency in annotation to facilitate model training.
* **Model Training:** Train deep learning models using annotated datasets to learn representations of TB-related features. Optimize hyperparameters and model architectures to improve performance metrics such as accuracy, sensitivity, and specificity.
  1. **NON-FUNCTIONAL REQUIREMENTS:**

In systems engineering, a non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviours. They are contrasted with functional requirements that define specific behaviour or functions.

The Non-functional requirements can be considered as quality attributes of a system. Non- functional requirements in tuberculosis (TB) detection using deep learning encompass aspects related to system performance, usability, security, and other quality attributes. Here are some non-functional requirements for TB detection using deep learning:

1. **Scalability:** Horizontal scalability Design the system to scale horizontally across multiple nodes or servers to handle increased workload and data volume. vertical scalability is Ensure that the system can scale vertically by upgrading hardware resources to meet growing demands.
2. **Reliability:** The system should be 90% reliable. Since it may need some maintenance or preparation for some particular day, the system does not need to be reliable every time. So, 80% reliability is enough.
3. **Maintainability:** Modularity Design the system with modular components to facilitate maintenance, updates, and enhancements and Documentation Provide comprehensive documentation to aid system maintenance and troubleshooting.
4. **Availability:** It is available in all the metropolitan cities.
5. **Cost efficiency:** Design the system to minimize costs associated with hardware, software, maintenance, and training and return on investment is to Evaluate the system's ROI by considering its effectiveness, cost savings, and other benefits compared to traditional TB detection methods.
   * 1. **PERFORMANCE REQUIREMENTS:**

The time required to divide the video into frames. The performance of a device is essentially estimated in terms of efficiency, effectiveness and speed.

1. **Inference Speed:** The deep learning model should provide fast inference times to enable quick diagnosis of TB cases. Specify maximum allowable inference times for different types of data (e.g., chest X-rays, CT scans) to ensure timely results.
2. **Accuracy:** Specify the minimum acceptable accuracy level for TB detection to minimize false positives and false negatives. Performance metrics such as sensitivity, specificity, and area under the receiver operating characteristic curve (AUC) should meet predefined thresholds.
3. **Scalability:** Ensure that the system can scale horizontally and vertically to handle increasing volumes of data and growing demand for TB diagnosis. Specify scalability requirements for both hardware infrastructure and software components.
   * 1. **HARDWARE REQUIREMENTS:**

The hardware requirements for tuberculosis (TB) detection using deep learning depend on various factors, including the complexity of the deep learning models, the size of the dataset, the volume of data being processed, and the desired performance criteria.

1. **CPU:** A powerful multi-core CPU is necessary for preprocessing data, managing system resources, and coordinating tasks. Modern CPUs with high clock speeds and multiple cores, such as Intel Core i7 or AMD processors, are suitable for deep learning workloads. The CPU should support advanced instruction sets like AVX2 or AVX-512 for efficient matrix computations commonly used in deep learning operations.
2. **Memory:** Sufficient system memory (RAM) is essential for storing and manipulating large datasets during training and inference. Deep learning models with larger memory footprints or working with large batches of data require more RAM to avoid performance bottlenecks. At least 16 GB of RAM is recommended for basic deep learning tasks, but larger models or datasets may require 32 GB or more.
3. **Networking:** A stable and high-speed network connection may be necessary for accessing datasets stored remotely, sharing resources across multiple nodes, or leveraging cloud- based computing resources. Ethernet connections with Gigabit or higher bandwidth are recommended for fast data transfer rates, especially in distributed computing environments.
   * 1. **SOFTWARE REQUIREMENTS:**

Software requirements for tuberculosis (TB) detection using deep learning encompass various components necessary for model development, training, inference, and deployment. Here are the key software requirements:

1. **Python Programming Language:** Python is the preferred programming language for deep learning due to its simplicity, versatility, and rich ecosystem of libraries and tools. Ensure that the required Python version is compatible with the selected deep learning framework and other dependencies.
2. **Development Environment:** Set up a development environment with integrated development environments (IDEs) or text editors suitable for deep learning development. Common choices include PyCharm, Jupyter Notebook, Visual Studio Code, and Google colab.
3. **Data Management Tools:** Use data management tools for organizing, preprocessing, and augmenting TB image datasets. Tools like Pandas, NumPy, and scikit-learn are often employed for data manipulation, analysis, and feature extraction.
4. **Model Interpretation and Visualization Tools:** Employ tools for interpreting and visualizing deep learning model outputs, such as saliency maps, heatmaps, and activation maximization. Libraries like Matplotlib, Seaborn, and Tensor Board provide visualization capabilities for analyzing model performance and debugging.
5. **Development Environment:** Set up a development environment with integrated development environments (IDEs) or text editors suitable for deep learning development. Common choices include PyCharm, Jupyter Notebook, Visual Studio Code, and Google Colab.
6. **Version Control:** Utilize version control systems (e.g., Git) for managing code repositories, tracking changes, and collaborating with team members. Platforms like GitHub, GitLab, or Bitbucket offer hosting services for storing and sharing code repositories.
   * + 1. **INTRODUCTION TO PYTHON:**

Python is a versatile and widely-used programming language that is well-suited for developing tuberculosis (TB) detection systems using deep learning techniques. In this introduction, we'll explore the key aspects of Python relevant to TB detection with deep learning.

1. **Why Python:** Python is used for deep learning due to its simplicity, readability, and extensive ecosystem of libraries and tools specifically designed for machine learning and artificial intelligence tasks. It offers a user-friendly syntax that allows developers to focus on the logic of their algorithms rather than dealing with low-level details.
2. **Deep Learning Libraries:** Python provides access to powerful deep learning frameworks such as TensorFlow, Pytorch and Keras which offer high-level APIs for building and training neural networks. These libraries abstract away the complexities of deep learning algorithms, allowing developers to create sophisticated models with minimal code.
3. **Data Processing and Analysis:** Python offers a rich set of libraries for data manipulation, preprocessing, and analysis, which are essential for working with medical imaging data in TB detection. Libraries such as NumPy, Pandas, and scikit-learn provide efficient tools for handling multidimensional arrays, processing tabular data, and performing statistical analysis.
4. **Image Processing:** Python libraries like OpenCV, Pillow (PIL), and scikit-image enable developers to perform image preprocessing tasks such as resizing, cropping, filtering, and augmentation. These libraries are crucial for preparing medical images (e.g., chest X-rays) before feeding them into deep learning models.
5. **Visualization:** Python offers various visualization libraries for visualizing data, model outputs, and performance metrics. Matplotlib, Seaborn, and Plotly are popular choices for creating plots, histograms, heatmaps, and other visualizations to analyze and interpret model results.
6. **Development Environment:** Python supports a wide range of integrated development environments (IDEs) and text editors for writing, debugging, and running code.

## CHAPTER 5 SYSTEM MODELING

* 1. **REQUIREMNETS MODELING:**

Requirements modelling for a project like tuberculosis detection using deep learning involves identifying and documenting the needs and constraints of the system.

* + 1. **CLASS DIAGRAMS:**

Class diagrams give an overview of a system by showing its classes and the relationships among them. Class diagrams are static – they display what interacts but not what happens when they do interact. In general a class diagram consists of some set of attributes and operations. Operations will be performed on the data values of attributes.

**model ID**: **String model Name: String**

**Deep Learning Model**

**Name: String Specialization: String**

**Radiologist**

**Image ID: String Image File Path: String**

**CXR Image**

|  |  |  |
| --- | --- | --- |
| **CXR Database** |  | |
|  | **TB Result** |
| **Database ID: String Images:<CXR Image>**  **Results: <TB Result>** |
| **result ID: String**  **tb Likelihood: double** |
|  |

* + 1. **USE CASE DIAGRAMS:**

Use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well. Actors are the external entities that interact with the system. The use cases are represented by either circles or ellipses.

# Upload CXR Image

**Preprocess Image**

**Radiologist**

# Apply DL Algorithm For TB Detection

**Deep Learning Model**

# Display TB Detection Result

**Provide Feedback On Result**

**CXR Database**

# Store CXR Image And Results in DB

* + 1. **SEQUENCE DIAGRAM:**

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.

Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios.

A Sequence diagram shows interactions arranged in time sequence. It depicts the s and classes involved in the scenario and the sequence of messages exchanged between the s needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development.

### Tuberculosis Detection System Using Deep Learning and Image Enhancement



**Upload chest X-ray images for analysis**

**Preprocess images to enhance quality and clarity**

**Apply deep learning algorithms for TB detection**

**Review results and provide feedback**

**Notify of TB likelihood and any abnormalities detected**

**Utilize convolutional neural networks for image analysis**

**Implement machine learning models for accurate diagnosis**

**Radiologist**

**Patient**

**System**

* + 1. **ACTIVITY DIAGRAM:**

Activity diagram is essentially a fancy flowchart: Activity and state diagrams are related. State chart diagram focuses on s undergoing a process. An activity diagram focuses on the flow of activities involved in a single process. The activity diagram shows the activities depend on one another.

An activity represents the performance of the task or duty in a workflow. It may also represent the execution of a statement in a procedure. You can share activities between state machines. However, transitions cannot be shared.

Activity diagrams provide a way to model the workflow of a business process, code specific information such as a class operation. The transitions are implicitly triggered by completion of the actions in the source activities.



**Start**

**Design and test custom**

**model on CXR images**



**Works**

**?**

**Yes No**

**Yes**

**Works**

**?**

**Modify model**

**No**

**No**

**Accept Model**

**Yes**

**Deploy Model**

**Stop**

**VISUALIZATION**

**CLAHE**

**HEF**

**UM**

* + 1. **MODULES DESCRIPTION:**

Detecting tuberculosis (TB) using deep learning with chest X-ray images typically involves the development of a convolutional neural network (CNN) model. Below is a general description of the modules involved in such a system:

* **\*\*Data Collection and Preprocessing\*\*: -** Collection of chest X-ray images containing both TB-positive and TB-negative cases.
* **\*\*Data Augmentation\*\*: -** Augmenting the dataset with techniques like rotation, flipping, scaling, and translation to increase the variability of the training data. This helps in improving the model's robustness and generalization.
* **\*\*Model Architecture Selection\*\*: -** Choosing an appropriate CNN architecture suited for image classification tasks. Popular choices include VGG, ResNet, Inception, and SqueezNet. The architecture should be capable of capturing intricate patterns and features from chest X-ray images.
* **\*\*Model Training\*\*: -** Training the selected CNN model using the augmented dataset. The model learns to differentiate between TBpositive and TB-negative cases by adjusting its internal parameters during the training process.
* **\*\*Hyperparameter Tuning\*\*: -** Optimization of hyperparameters such as learning rate, batch size, and optimizer choice to enhance the model's performance. Techniques like grid search or random search may be employed to find the optimal set of hyperparameters.
* **\*\*Validation and Evaluation\*\*: -** Validating the trained model on a separate validation dataset to assess its performance metrics such as accuracy, precision, recall, and F1-score. Evaluating the model's performance on an independent test dataset to ensure its generalization capability and reliability.
  1. **DESIGN MODELING:**

Design modelling refers to the process of creating representations of a system or product before itis built or implemented. These models help designers, engineers, and stakeholders visualize, analyze, and communicate various aspects of the design. Common types of design modelling include.

* **Conceptual Modelling:** This involves creating high-level, abstract representations of the system or product to capture its fundamental concepts and relationships.
* **Functional Modelling:** Focuses on representing the functions or operations that the system or product should perform. It helps in understanding how different components interact to achieve desired functionality.
* **Behavioural Modelling:** Describes the dynamic behaviour of the system or product over time. It includes state diagrams, sequence diagrams, and activity diagrams to illustrate the system's behaviour in response to different inputs and events.
* **Structural Modelling:** Involves representing the structure or architecture of the system, including its components, their properties, and relationships. Examples include class diagrams, object diagrams, and component diagrams.
* **Physical Modelling:** Deals with representing the physical aspects of the system or product, such as its geometry, materials, and spatial relationships. This type of modelling is common in architecture, industrial design, and mechanical engineering.
  + 1. **SYSTEM ARCHITECTURE:**

System architecture refers to the conceptual structure and organization of a complex system. It encompasses the components, their relationships, interactions, and the principles guiding their design and evolution.

### Patient



**Image**

**Pre-Processing**

**Chest X-ray**



**Training**

**CNN Model**



**Data**

**Data Generation**

**TB Likelihood Prediction**

**Probability of having TB**

🠶 The block diagram you provided outlines a process for detecting tuberculosis using deep learning.

🠶 CNN stands for Convolutional Neural Network. It is a type of deep neural network that is primarily used for analyzing visual imagery.

🠶 CNNs are particularly effective for tasks such as image classification, object detection, and image segmentation.

🠶 Here’s a brief explanation of each step:

* **Patient:** The process begins with a patient who is suspected of having tuberculosis.
* **Chest X-ray**: An X-ray image of the patient’s chest is taken. This image serves as the initial data for the detection process.
* **Image Pre-Processing**: The X-ray image undergoes a pre-processing stage where it is prepared for analysis. This could involve enhancing the image, removing noise, or extracting certain features from the image that are useful for detection.
* **CNN (Convolutional Neural Network) Model**: CNNs are a type of deep learning model that are particularly good at analyzing visual data.
* **Data Generation:** Additional data is generated to aid in training the CNN model. This could involve techniques like data augmentation, which create new training examples through transformations of the original images.
* **Training**: The CNN model is trained using the generated data. During training, the model learns to identify patterns in the data that are indicative of tuberculosis.
* **TB Likelihood Prediction**: After training, the model can predict the likelihood of tuberculosis in new chest X-ray images. This prediction is typically given as a probability.
* **SQUEEZENET MODEL:**

**Conv1**

**Max-Pool**

### Global Avg pool

**fire2**

**fire3**

**fire4**



**Conv10**

**fire9**



**Softmax**

**Max-Pool/2 Max-Pool/2**

**fire5**

**fire6**

**fire7**

**fire8**

**CHAPTER 6**

**SYSTEM IMPLEMENTATION**

### DEPLOYMENT ENVIRONMENT AND TOOLS:

* + 1. IDE (Integrated Development Environment):
       - Google Colab (for Python coding and development)
    2. Programming Languages:
       - Python (for backend development and machine learning)
       - HTML (for frontend development)
    3. Deployment Platforms:
       - Streamlit (for deploying the web application)
    4. Documentation Tools:
       - Word

Creating a development environment for a tuberculosis detection project using deep learning involves setting up the necessary tools and libraries. Here's a basic setup guide:

* **Operating System:** Use a supported operating system such as Windows 10, macOS, or a Linux distribution like Ubuntu.
* **Python:** Install Python, which is commonly used for deep learning projects. You can download the latest version from the official Python website and follow the installation instructions.
* **Deep Learning Libraries:** Install deep learning libraries such as TensorFlow or PyTorch. You can install these libraries using pip, Python's package manager.
* **Data Visualization:** Install libraries for data visualization, such as Matplotlib and Seaborn, to visualize the dataset and model performance.
* **Other Libraries:** Depending on your specific needs, you may also need to install other libraries for data manipulation (e.g., Pandas), image processing (e.g., OpenCV), or model evaluation (e.g., Scikit-learn).
  1. **CODING:**

In this project, extensive Python coding was undertaken to preprocess data, implement deep learning algorithms, and develop the web application interface. Specifically, coding involved data preprocessing techniques to clean and prepare input data for analysis. Additionally, SqueezeNet, ResNet-50 were implemented using libraries like Keras and TensorFlow. Overall, Python coding was instrumental in driving the project's success from data processing to model deployment.

### 6.2.1 SOURCE CODE:

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

import PIL

import pathlib

import tensorflow as tf

import keras

from tensorflow.keras import layers

from tensorflow.python.keras.layers import Dense, Flatten, Input

from tensorflow.keras.models import Sequential

from tensorflow.keras.optimizers.legacy import Adam

from tensorflow.keras.losses import binary\_crossentropy

from tensorflow.keras.layers import Input, Conv2D, ReLU, concatenate, Dropout,AvgPool2D,MaxPooling2D,BatchNormalization,GlobalAveragePooling2D

from tensorflow.keras.models import Model

from tensorflow.keras.preprocessing.image import ImageDataGenerator #DataAugmentation

from tensorflow.keras.metrics import Accuracy, Precision, Recall, F1Score, FBetaScore

import os

import shutil

from sklearn import metrics

import seaborn as sns

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import ConfusionMatrixDisplay

from tensorflow.python.framework.ops import disable\_eager\_execution

disable\_eager\_execution()

url = 'TB\_Chest\_Radiography\_Database'

data\_dir = pathlib.Path(url)

img\_type= {

'Normal' : list(data\_dir.glob('Normal/\*.png')),

'Tuberculosis' : list(data\_dir.glob('Tuberculosis/\*.png'))

}

def img\_plot\_original():

fig, axs = plt.subplots(1,2)

i = 0

for key, value in img\_type.items():

img = mpimg.imread(str(img\_type[key][3]))

axs[i].imshow(img)

axs[i].set\_title(key)

i+=1

img\_plot\_original()

labels = 'Normal', 'Tuberculosis'

sizes = [len(img\_type['Normal']), len(img\_type['Tuberculosis'])]

fig, ax = plt.subplots()

ax.pie(sizes, labels=labels, autopct='%1.1f%%')

#Making Testing Directory

os.mkdir('Xray-Test-Data')

xrayTestUrl = 'Xray-Test-Data'

data\_dir\_test = pathlib.Path(xrayTestUrl)

#TB TESTING DIR

directoryTestTb = "Test-TB-Xray"

parent\_dir = xrayTestUrl

pathTestTB = os.path.join(parent\_dir, directoryTestTb)

os.mkdir(pathTestTB)

directoryTestNM = "Test-NM-Xray"

parent\_dir = xrayTestUrl

pathTestNM = os.path.join(parent\_dir, directoryTestNM)

os.mkdir(pathTestNM)

### normalTestImg img\_type['Normal'][1000:1101]

### tbTestImg = img\_type['Tuberculosis'][600:]

### destination\_directory\_tb = 'Xray-Test-Data/Test-TB-Xray'

### for img in tbTestImg:

### shutil.copy(img, destination\_directory\_tb)

### destination\_directory\_NM = 'Xray-Test-Data/Test-NM-Xray'

### for img in normalTestImg:

### shutil.copy(img, destination\_directory\_NM)

### os.mkdir('Xray-Data')

### xrayUrl = 'Xray-Data'

### directoryTB = "TB-Xray"

### directoryNM = 'NM-Xray'

### 

### parent\_dir = "Xray-Data"

### pathTB = os.path.join(parent\_dir, directoryTB)

### os.mkdir(pathTB)

### pathNM = os.path.join(parent\_dir, directoryNM)

### os.mkdir(pathNM)

### normal\_imgs = img\_type['Normal'][:600]

### normal\_imgs

### img\_height = 64

### img\_width = 64

### batch\_size = 16

### train\_datagen = ImageDataGenerator(

### rescale=1./255,

### rotation\_range=40,

### width\_shift\_range=0.2,

### height\_shift\_range=0.2,

### shear\_range=0.2,

### zoom\_range=0.2,

### horizontal\_flip=True,

### validation\_split = 0.2

### )

### train\_generator = train\_datagen.flow\_from\_directory(

### data\_dir,

### target\_size = (img\_height, img\_width),

### batch\_size = batch\_size,

### class\_mode = 'binary',

### subset='training',

### seed=123

### )

### traing\_set\_len = len(train\_generator)

### print(traing\_set\_len)

### val\_datagen = ImageDataGenerator(rescale=1./255, validation\_split = 0.2)

### val\_generator = val\_datagen.flow\_from\_directory(

### data\_dir,

### target\_size = (img\_height, img\_width),

### batch\_size = batch\_size,

### class\_mode = 'binary',

### subset='validation',

### seed=123

### )

### val\_set\_len = len(val\_generator)

### print(val\_set\_len)

### test\_generator = val\_datagen.flow\_from\_directory(

### data\_dir\_test,

### target\_size = (img\_height, img\_width),

### class\_mode = 'binary',

### )

### test\_set\_len = len(test\_generator)

### print(test\_set\_len)

### images, labels = next(train\_generator)

### # Plot one image from the batch

### plt.imshow(images[0])

### plt.title(f"Class: {labels[0]}")

### plt.show()

### labels = 'Validation', 'Training', 'Testing'

### sizes = [val\_set\_len, traing\_set\_len, test\_set\_len]

### fig, ax = plt.subplots()

### ax.pie(sizes, labels=labels, autopct='%1.1f%%')

### #SqueezeNEt Architecture

### def fireMode(x, s1, e1, e3):

### sx1 = Conv2D(filters=s1, kernel\_size=1, activation='relu', padding='same')(x)

### sx1 = BatchNormalization()(sx1)

### ex1 = Conv2D(filters=e1, kernel\_size=1, activation='relu', padding='same')(sx1)

### ex1 = BatchNormalization()(ex1)

### ex3 = Conv2D(filters=e3, kernel\_size=3, activation='relu', padding='same')(sx1)

### ex3 = BatchNormalization()(ex3)

### return concatenate([ex1, ex3])

### def squeezNet(input\_size, classes):

### 

### x = Input(shape=input\_size)

### 

### # Entry block

### y = Conv2D(32, kernel\_size=3, activation='relu', padding='same')(x)

### y = BatchNormalization()(y)

### # Fire modules

### y = fireMode(y, 12, 24, 24)

### y = BatchNormalization()(y)

### y = GlobalAveragePooling2D()(y)

### y = BatchNormalization()(y)

### # Dense layers

### y = layers.Dense(128, activation='relu')(y)

### y = BatchNormalization()(y)

### y = Dropout(0.5)(y)

### y = BatchNormalization()(y)

### # Output layer

### y = layers.Dense(1, activation='sigmoid')(y)

### model = Model(inputs=x, outputs=y)

### return model

### model = squeezNet((64,64,3), 1)

### model.summary()

### def f1\_score(y\_true, y\_pred):

### # Calculate true positives, false positives, and false negatives

### true\_positives = tf.keras.backend.sum(tf.keras.backend.round(tf.keras.backend.clip(y\_true \* y\_pred, 0, 1)))

### predicted\_positives = tf.keras.backend.sum(tf.keras.backend.round(tf.keras.backend.clip(y\_pred, 0, 1)))

### actual\_positives = tf.keras.backend.sum(tf.keras.backend.round(tf.keras.backend.clip(y\_true, 0, 1)))

### # Calculate precision and recall

### precision = true\_positives / (predicted\_positives + tf.keras.backend.epsilon())

### recall = true\_positives / (actual\_positives + tf.keras.backend.epsilon())

### # Calculate F1 score

### f1 = 2 \* (precision \* recall) / (precision + recall + tf.keras.backend.epsilon())

### return f1

### opt = Adam(learning\_rate=0.0001)

### model.compile(loss="binary\_crossentropy", optimizer=opt, metrics=['accuracy',Precision(), Recall(),

### f1\_score])

### history = model.fit(train\_generator, steps\_per\_epoch=traing\_set\_len,

### epochs=10, validation\_data=val\_generator,

### validation\_steps=test\_set\_len)

### #Results

### # summarize accuracy

### plt.plot(history.history['accuracy'])

### plt.plot(history.history['val\_accuracy'])

### plt.title('model accuracy')

### plt.ylabel('accuracy')

### plt.xlabel('epoch')

### plt.legend(['Train', 'Validation'], loc='upper left')

### plt.show()

### # summarize loss

### plt.plot(history.history['loss'])

### plt.plot(history.history['val\_loss'])

### plt.title('model loss')

### plt.ylabel('loss')

### plt.xlabel('epoch')

### plt.legend(['Train', 'Validation'], loc='upper left')

### plt.show()

### # summarize percison

### plt.plot(history.history['precision'])

### plt.plot(history.history['val\_precision'])

### plt.title('model precision')

### plt.ylabel('precision')

### plt.xlabel('epoch')

### plt.legend(['Train', 'Validation'], loc='upper left')

### plt.show()

### # summarize recall

### plt.plot(history.history['recall'])

### plt.plot(history.history['val\_recall'])

### plt.title('model recall')

### plt.ylabel('recall')

### plt.xlabel('epoch')

### plt.legend(['Train', 'Validation'], loc='upper left')

### plt.show()

### # summarize recall

### plt.plot(history.history['f1\_score'])

### plt.plot(history.history['val\_f1\_score'])

### plt.title('model f1\_score')

### plt.ylabel('f1\_score')

### plt.xlabel('epoch')

### plt.legend(['Train', 'Validation'], loc='upper left')

### plt.show()

### #Testing Usage

### results = model.evaluate(test\_generator)

### def results\_test\_data(results):

### print(f"Loss: {results[0]}")

### print(f"Accuracy: {results[1]}")

### print(f"Percision: {results[2]}")

### print(f"Recall: {results[3]}")

### print(f"F1\_Score: {results[4]}")

### results\_test\_data(results)

### def results\_test\_data\_graph(results):

### labels = ['Loss', 'Accuracy', 'Precision', 'Recall', 'F1 Score']

### values = results[:5] # Assuming results is a list with at least 5 elements

### # Bar graph

### plt.bar(labels, values, color=['blue', 'green', 'red', 'purple', 'orange'])

### plt.title('Test Results')

### plt.xlabel('Metrics')

### plt.ylabel('Values')

### plt.show()

### # Example usage

### results = [results[0], results[1], results[2], results[3], results[4]]

### results\_test\_data\_graph(results)

### predicted\_values = model.predict(test\_generator)

### acutal\_test\_values = []

### for i in range(7):

### lst = test\_generator[i][1]

### for num in lst:

### acutal\_test\_values.append(num) predicted\_values = (predicted\_values >= 0.5).astype(int)

### conf\_matrix = confusion\_matrix(acutal\_test\_values, predicted\_values)

### disp = ConfusionMatrixDisplay(confusion\_matrix=conf\_matrix, display\_labels=['Normal', "tuberculosis"])

### disp.plot(cmap='Blues', values\_format='d')

### plt.title('Confusion Matrix')

### plt.show()

### #ResNet

### resnet\_model = Sequential()

### pretrained\_model= tf.keras.applications.ResNet50(include\_top=False,

### input\_shape=(img\_height,img\_width,3),

### pooling='avg',classes=1,

### weights='imagenet')

### for each\_layer in pretrained\_model.layers:

### each\_layer.trainable=False

### resnet\_model.add(pretrained\_model)

### resnet\_model.add(layers.Flatten())

### resnet\_model.add(BatchNormalization())

### resnet\_model.add(layers.Dense(128, activation='relu'))

### resnet\_model.add(BatchNormalization())

### resnet\_model.add(Dropout(0.3))

### resnet\_model.add(layers.Dense(64, activation='relu'))

### resnet\_model.add(BatchNormalization())

### resnet\_model.add(Dropout(0.3))

### resnet\_model.add(BatchNormalization())

### resnet\_model.add(layers.Dense(1, activation='sigmoid'))

### ## Training

### resnet\_model.compile(loss="binary\_crossentropy", optimizer=Adam(learning\_rate=0.001), metrics=['accuracy',Precision(), Recall(),f1\_score]) # summarize accuracy

### plt.plot(history.history['accuracy'])

### plt.plot(history.history['val\_accuracy'])

### plt.title('model accuracy')

### plt.ylabel('accuracy')

### plt.xlabel('epoch')

### plt.legend(['Train', 'Validation'], loc='upper left')

### plt.show()

### # summarize loss

### plt.plot(history.history['loss'])

### plt.plot(history.history['val\_loss'])

### plt.title('model loss')

### plt.ylabel('loss')

### plt.xlabel('epoch')

### plt.legend(['Train', 'Validation'], loc='upper left')

### plt.show()

### # summarize percison

### plt.plot(history.history['precision\_1'])

### plt.plot(history.history['val\_precision\_1'])

### plt.title('model precision')

### plt.ylabel('precision')

### plt.xlabel('epoch')

### plt.legend(['Train', 'Validation'], loc='upper left')

### plt.show()

### # summarize recall

### plt.plot(history.history['recall\_1'])

### plt.plot(history.history['val\_recall\_1'])

### plt.title('model recall')

### plt.ylabel('recall')

### plt.xlabel('epoch')

### plt.legend(['Train', 'Validation'], loc='upper left')

### plt.show()

### # summarize recall

### plt.plot(history.history['f1\_score'])

### plt.plot(history.history['val\_f1\_score'])

### plt.title('model f1\_score')

### plt.ylabel('f1\_score')

### plt.xlabel('epoch')

### plt.legend(['Train', 'Validation'], loc='upper left')

### plt.show()

### #Streamlit code

### import numpy as np import streamlit as st import cv2

**import tensorflow as tf from io import BytesIO import sqlite3**

### import bcrypt

**# Initialize SQLite database**

### conn = sqlite3.connect('users.db') c = conn.cursor()

**# Create users table if it doesn't exist c.execute('''**

### CREATE TABLE IF NOT EXISTS users (

**username TEXT PRIMARY KEY, password TEXT**

### ) ''')

**conn.commit()**

### def hash\_password(password):

**return bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())**

### def verify\_password(password, hashed\_password):

**return bcrypt.checkpw(password.encode('utf-8'), hashed\_password)**

### def signup(username, password): # Check if user already exists

**c.execute('SELECT \* FROM users WHERE username = ?', (username,)) if c.fetchone():**

### return False else:

**# Hash the password and store the user hashed\_password = hash\_password(password)**

### c.execute('INSERT INTO users (username, password) VALUES (?, ?)', (username, hashed\_password))

**conn.commit() return True**

### def login(username, password):

**c.execute('SELECT \* FROM users WHERE username = ?', (username,)) user = c.fetchone()**

### if user and verify\_password(password, user[1]): return True

**else:**

### return False

**# Initialize the database connection**

### conn = sqlite3.connect('users.db', check\_same\_thread=False) c = conn.cursor()

**# Create the users table if it doesn't exist c.execute('''**

### CREATE TABLE IF NOT EXISTS users(

**username TEXT NOT NULL UNIQUE, password TEXT NOT NULL**

### ) ''')

**conn.commit()**

### # ... (Other function definitions remain unchanged) def add\_user(username, password):

**hashed\_password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt()) try:**

### c.execute('INSERT INTO users(username, password) VALUES (?, ?)', (username, hashed\_password))

**conn.commit() return True**

### except sqlite3.IntegrityError: return False

**def check\_user(username, password):**

### c.execute('SELECT password FROM users WHERE username = ?', (username,)) result = c.fetchone()

**if result:**

### return bcrypt.checkpw(password.encode('utf-8'), result[0]) return False

**def show\_signup():**

### st.write("### Sign Up to Tuberculosis Detection")

**new\_username = st.text\_input("Username", key="new\_username")**

### new\_password = st.text\_input("Password", type="password", key="new\_password") if st.button("Sign Up"):

**if new\_username and new\_password:**

### if add\_user(new\_username, new\_password): st.success("Account created successfully! Please log in.") st.session\_state['current\_page'] = 'login'

**st.return() else:**

### st.error("Username already exists. Please try a different one.")

**else:**

### st.error("Please enter a username and password.")

**def show\_login(): st.write("### Log In")**

### username = st.text\_input("Username", key="username")

**password = st.text\_input("Password", type="password", key="password")**

### if st.button("Log In"):

**if check\_user(username, password): st.session\_state['logged\_in'] = True st.return()**

### else:

**st.error("Wrong credentials or user does not exist.")**

### def model():

**# Set the title of the app**

### #st.title('Tuberculosis Detection using Deep Learning') # File uploader widget

**input\_file\_name = st.file\_uploader("Choose an image...", type=["jpg", "jpeg", "png"])**

### if input\_file\_name is not None:

**# Convert the file to a numpy array**

### file\_bytes = np.asarray(bytearray(input\_file\_name.read()), dtype=np.uint8) image = cv2.imdecode(file\_bytes, cv2.IMREAD\_COLOR)

**# Display the uploaded image**

### st.image(image, caption="Chest X-Ray", use\_column\_width=True)

**# Resize and preprocess the image image = cv2.resize(image, (64, 64)) image = image / 255.0**

### image = np.array([image])

**# Load your model**

### model = tf.keras.models.load\_model('t\_model.pt')

**# Make a prediction**

### prediction = model.predict(image)

**# Display the prediction result if prediction \* 100 >= 1:**

### st.write("Tuberculosis Detected") else:

**st.write("Tuberculosis Not Detected") if name ==' main ':**

### st.title('Tuberculosis Detection using Deep Learning') auth\_status = st.session\_state.get('auth\_status', None) if auth\_status == "logged\_in":

**st.success(f"Welcome {st.session\_state.username}!") st.header('Upload an image and the model will detect tuberculosis') model()**

### elif auth\_status == "login\_failed":

**st.error("Login failed. Please check your username and password.") auth\_status = None**

### elif auth\_status == "signup\_failed":

**st.error("Signup failed. Username already exists.") auth\_status = None**

### # Login/Signup form

**if auth\_status is None or auth\_status == "logged\_out":**

### form\_type = st.radio("Choose form type:", ["Login", "Signup"])

**username = st.text\_input("Username")**

### password = st.text\_input("Password", type="password")

**if form\_type == "Login": if st.button("Login"):**

### if login(username, password): st.session\_state.auth\_status = "logged\_in" st.session\_state.username = username st.return()

**else:**

### st.session\_state.auth\_status = "login\_failed"

**st.return() else: # Signup**

### if st.button("Signup"):

**if signup(username, password): st.session\_state.auth\_status = "logged\_in" st.session\_state.username = username st.return()**

### else:

**st.session\_state.auth\_status = "signup\_failed" st.return()**

### # Logout button

**if auth\_status == "logged\_in": if st.button("Logout"):**

### st.session\_state.auth\_status = "logged\_out" del st.session\_state.username

**st.return()**

## CHAPTER 7 TESTING

### INTRODUCTION TO TESTING:

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement.

* System Testing is a critical aspect of software quality assurance and represents the final review of requirement design and coding. Testing is a process of executing a program with the intent of finding error. A good test is one that has a probability of finding an undiscovered error. The purpose of testing is to identify and correct bugs in the developed system as nothing is complete without testing. Testing is vital to the success of the system. In the code testing, the logic of the developed system is tested. For this every module of the program is executed to find an error. To perform specification test, the examination of the specifications starts with what the program should do and how it should perform under various conditions. System Testing does not test the software as a whole, but rather than integration of each module in the system. The primary concern is the compatibility of the individual modules. One has to find the areas where modules have been designed with different specifications of the data lengths, type and data element name.
* Testing and validation are the most important steps after implementation of the developed system. The system testing is performed to ensure that there are no errors in the implemented system. The software must be executed several times in order to find out the errors in the different modules of the system.
* Test Objective is the overall goal and achievement of the test execution. The objective of the testing is finding as many software defects as possible to ensure that the software under test is bug free before release.
  + A successful test is one that determines an as yet undiscovered error.
  +  good test case is one that has the possibility of discovering an error, if it exists.
  + The test is insufficient to detect possibly present errors.
  + The software more or less approves the quality and unswerving standards.

### TESTING STRATEGIES:

Testing strategies for tuberculosis (TB) detection using deep learning involve several key aspects to ensure the model's performance, generalizability, and reliability. Here are some test strategies:

* **Unit Testing:** Focuses on testing individual units or components of the software in isolation.
* **Integration Testing:** Verifies the interaction and integration between different modules or components of the software.
* **System Testing:** Validates the behaviour and functionality of the entire system as a whole.
* **Acceptance Testing:** Includes alpha testing (internal testing by developers) and beta testing (external testing by a select group of users).
* **Regression Testing:** Ensures that recent code changes have not adversely affected existing functionality.
* **Continuous Integration/Continuous Deployment (CI/CD) Testing:** Integrates testing into the CI/CD pipeline to automate the build, test, and deployment processes.
* **Performance Testing:** Evaluates the software's performance under various conditions, such as load, stress, and scalability.
* **Security Testing:** Focuses on identifying vulnerabilities, security flaws, and potential risks with in the software.

### UNIT TESTING:

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

### INTEGRATION TESTING:

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfied, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

### SYSTEM TESTING:

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

System testing for tuberculosis (TB) detection using deep learning involves evaluating the entire system, including the deep learning model, data preprocessing pipeline, and any other components, to ensure that it performs as expected. Here are the key steps involved in system testing for TB detection using deep learning.

### TEST CASES:

**Testing Unit:** Login Page

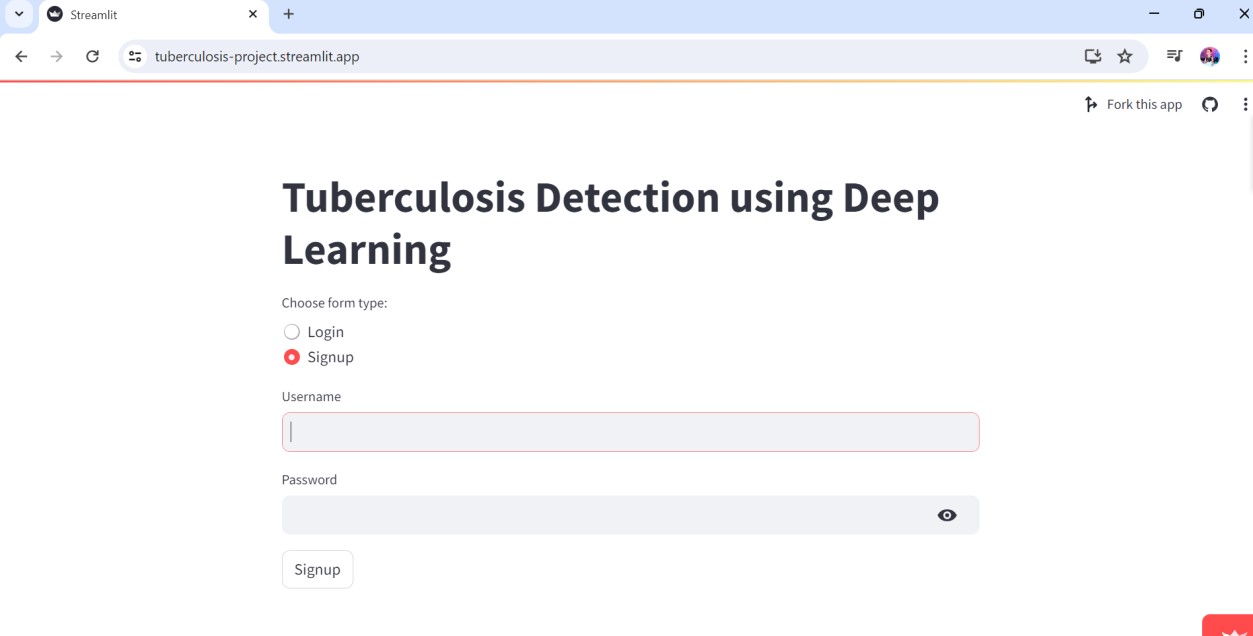


Fig no- 7.1 Tuberculosis Detection Using Deep Learning Page

**Test case T01:** Writing test cases for Signup Page with all specified fields.

|  |  |
| --- | --- |
| **TID:** 01 | **Priority:** High |
| **Test Unit:** Signup Page | |
| **Test Description:** To check whether the signup with all specified fields | |
| **Test Data:** Username: bhavya  Password: Bhavya@19 | |
| **Actions:** Click on signup | |
| **Expected Result:** Signup Successfully | **Output:** Welcome bhavya! |
| **Note:** Executed Successfully | |

Table-7.1: Writing test cases for signup

Result is shown below



Fig no-7.2: Result of testcase -1

**Test case T02:** Writing test cases for selecting file

|  |  |
| --- | --- |
| **TID:** 02 | **Priority:** High |
| **Test Unit:** Select file | |
| **Test Description:** To check whether the page choose the file | |
| **Test Data:** File from the folder | |
| **Actions:** Click on Browse files | |
| **Expected Result:** Image file is to be taken  further analysis | **Output:** Image file is taken for  for further analysis |
| **Note:** Executed Successfully | |

Table-7.2: Writing test cases for selecting file

Result is shown below



Fig no-7.3: Result of testcase -2

**Test case T03:** Writing test cases for detecting an image file

|  |  |
| --- | --- |
| **TID:** 03 | **Priority:** High |
| **Test Unit:** Detecting an Image | |
| **Test Description:** To check whether the image is Tuberculosis or not | |
| **Test Data:** image in .jpg, .png format | |
| **Actions:** Click on Browse files | |
| **Expected Result:** It detect the image in  TB | **Output:** TB image |
| **Note:** Executed Successfully | |

Table-7.3: Writing test cases for detecting image

Result is shown below

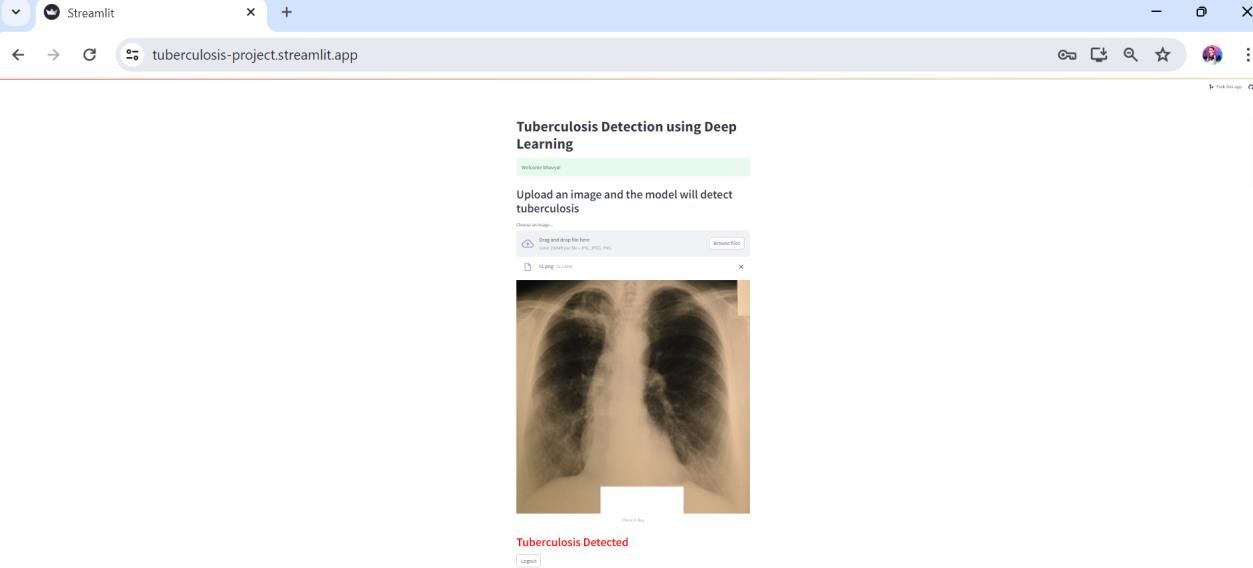


Fig no-7.4: Result of testcase -3

**Test case T04:** Writing test cases for detecting an image file

|  |  |  |
| --- | --- | --- |
| **TID:** 04 | | **Priority:** High |
| **Test Unit:** Detecting an Image | | |
| **Test Description:** To check whether the image is Tuberculosis or not | | |
| **Test Data:** image in .jpg, .png format | | |
| **Actions:** Click on Browse files | | |
| **Expected Result:** It detect the image in  Non-TB | **Output:** Non-TB image | |
| **Note:** Executed Successfully | | |

Table-7.4: Writing test cases for detecting image

Result is shown below

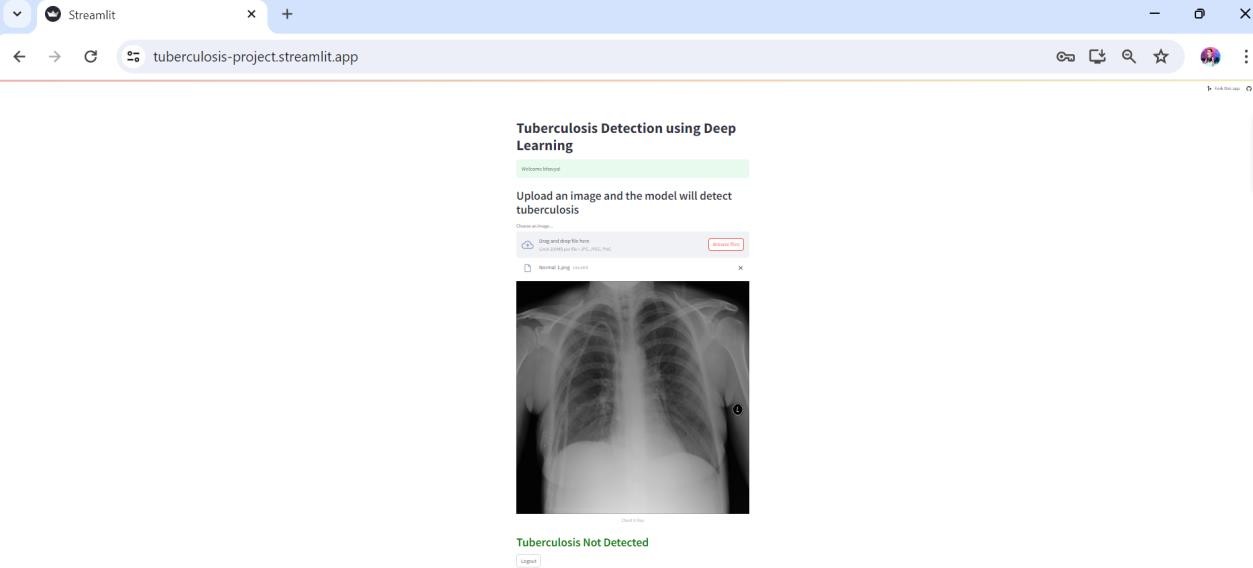


Fig no-7.5: Result of testcase -4

## CHAPTER 8 RESULTS

### SCREENS AND REPORTS:

Here in our project, we train the Tuberculosis detection related images containing the normal and tuberculosis to detect the early stages of it using cnn models such as SqueezeNet, resNet-50. we extract the dataset from Kaggle.

The dataset contains 4200 images both normal and tuberculosis images separately. Out of 4200 images normal contains 3500 images, for testing 140 images and for training 3360 images. Remaining 700 were contain tuberculosis for training 600 images and for testing 100 images.

When we compared to other models SqueezeNet gives more accuracy. So SqueezeNet is extremely fit to Early detection of Tuberculosis.

To implement this process some steps were involved:

* Data collection
* Preprocessing
* Feature Extraction
* CNN Models
* Model Evaluation
* Integration and Deployment

Here are some screens about the implementation of Tuberculosis Detection:

* First we need to upload a file image
* After that it will predict the output.
* It produces the output with labels

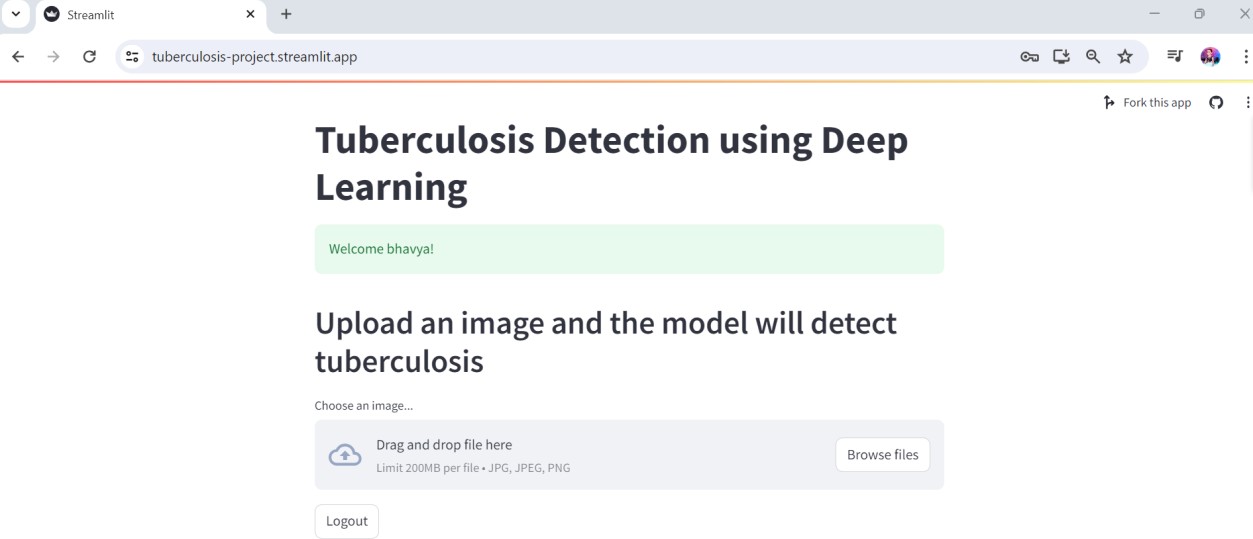


Fig no.8.1.1-upload image file

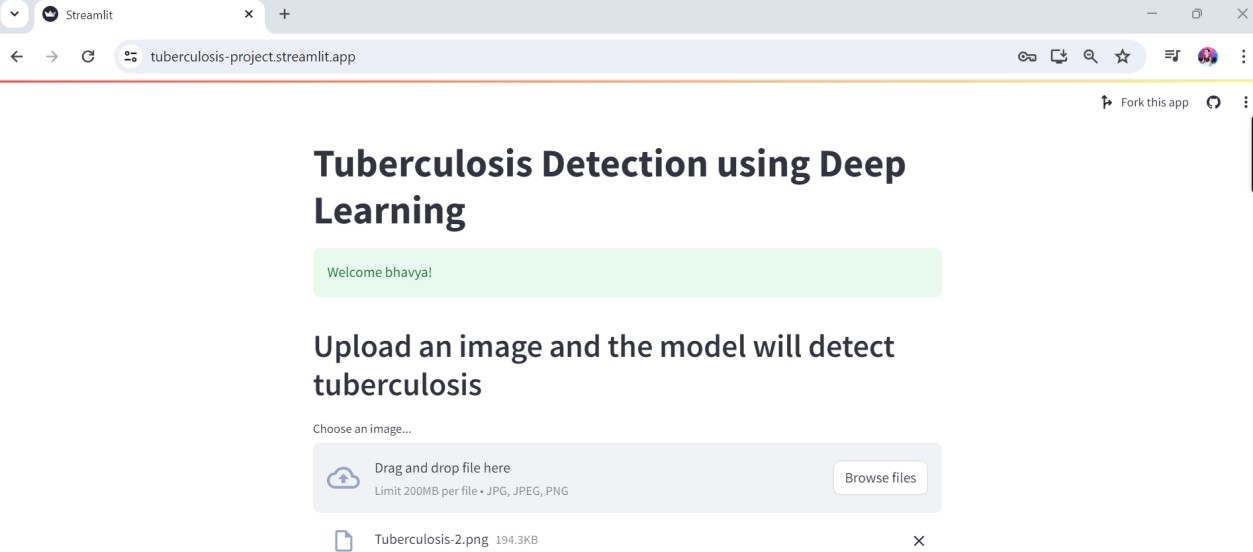


Fig no.8.1.2-upload tuberculosis image file

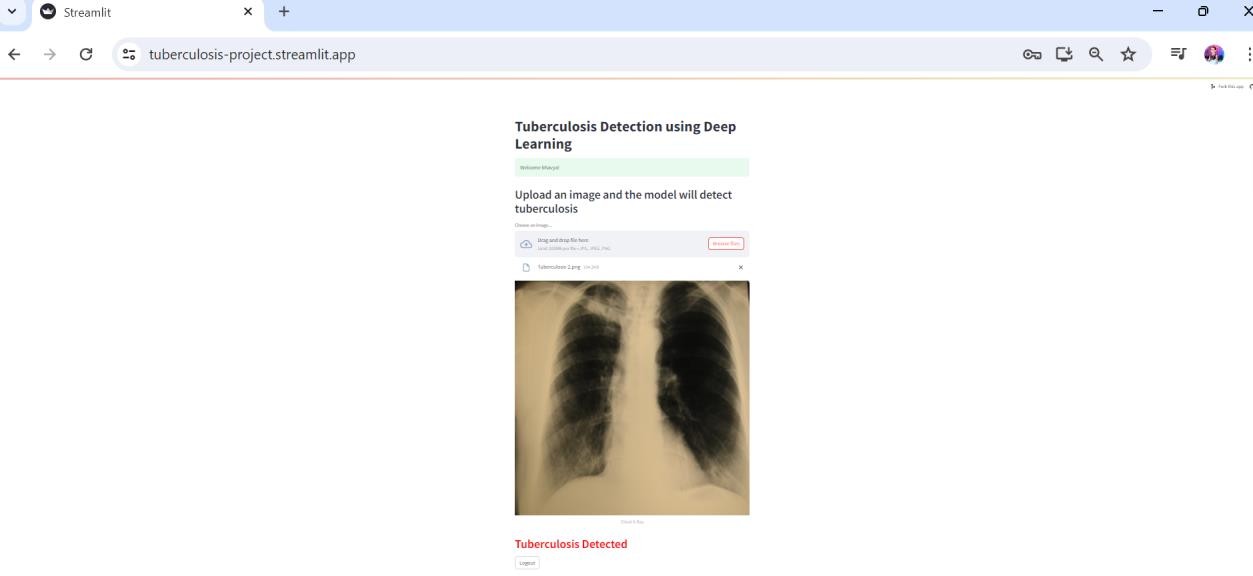


Fig no 8.1.3: predict the result

The above figure gives the prediction of Tuberculosis Detected

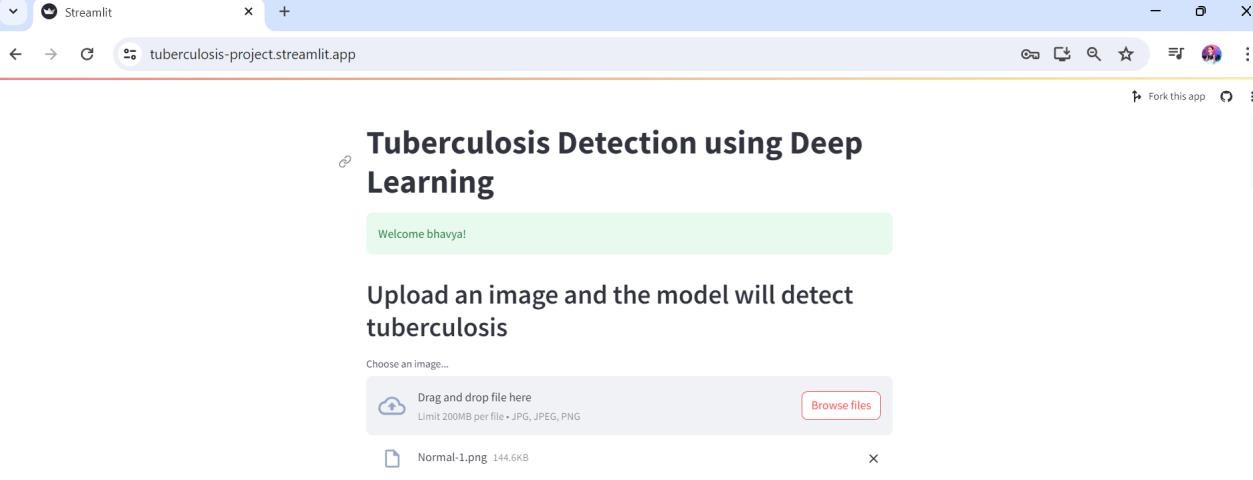


Fig no.8.1.4-upload normal image file

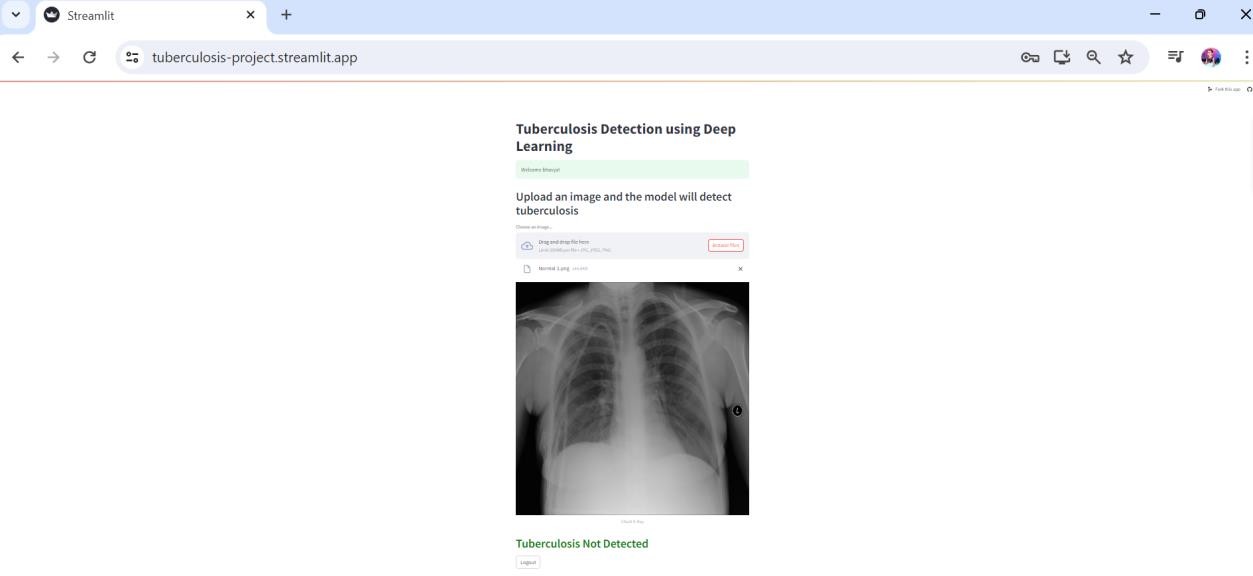


Fig no 8.1.3: predict the result

The above figure gives the prediction of Tuberculosis Not Detected

### REPORTS:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case**  **ID** | **Test Unit** | **Text**  **Description** | **Expected**  **Result** | **Actual**  **Result** | **Type** |
| 01 | Signup Page | To check whether the signup with all specified  fields | pass | pass | chosen |
| 02 | Select file | To check whether the page choose  the file | pass | pass | selected |
| 03 | Detecting an Image | To check whether the image is TB  or not | pass | pass | predicted |
| 04 | Detecting an Image | To check whether the image is TB  or not | pass | pass | predicted |

Table-8.1: Reports of Test Cases

### USER MANUAL:

In this project, we developed a web page for identifying the Tuberculosis Detection in it’s early stages by using the normal and tuberculosis images. In this we have

### \*\*Data Collection and Preparation: \*\*

* + Gather a dataset of chest X-ray images that includes both tuberculosis-infected and normal cases. You may need a large and diverse dataset for effective training.
  + Ensure that the dataset is properly labelled or annotated, indicating which images are tuberculosis positive and which are negative.

### \*\*Data Preprocessing: \*\*

* + Preprocess the chest X-ray images to enhance quality and reduce noise. Common preprocessing steps include resizing, normalization, and grayscale conversion.
  + Augment the dataset if necessary to increase variability and improve model generalization. Augmentation techniques may include rotation, flipping, and scaling.

### \*\*Model Selection or Design: \*\*

* + Choose an appropriate deep learning architecture for tuberculosis detection. Convolutional Neural Networks (CNNs) are commonly used for image classification tasks and are likely suitable for this application.
  + Decide whether to use a pre-trained model or train from scratch based on the availability of data and computational resources.

### \*\*Training: \*\*

* + Split the dataset into training, validation, and test sets. The training set is used to train the model, the validation set is used for hyperparameter tuning and model selection, and the test set is used to evaluate the final model's performance.
  + Train the deep learning model using the training data. During training, optimize the model's parameters to minimize a chosen loss function (e.g., binary cross-entropy).
  + Monitor the model's performance on the validation set to prevent overfitting.

### \*\*Evaluation: \*\*

* + Evaluate the trained model's performance on the test set using metrics such as accuracy, precision, recall, F1-score, and area under the ROC curve (AUC).
  + Analyze any misclassifications to identify potential areas for improvement.

### \*\*Deployment and Integration: \*\*

* + Once satisfied with the model's performance, deploy it for tuberculosis detection in real-world applications.
  + Integrate the model into a user-friendly interface or workflow, allowing users to input chest X-ray images and receive predictions.

### \*\*Continued Monitoring and Improvement: \*\*

* + Monitor the model's performance in real-world scenarios and collect feedback for further improvement.
  + Periodically retrain the model using updated datasets or techniques to ensure optimal performance over time.

## Chapter 9 LIMITATIONS

While deep learning models have shown promising results in detecting tuberculosis (TB) from chest X-ray images, there are several limitations to this approach:

1. **\*\*Data Imbalance\*\*:** TB datasets often suffer from class imbalance, meaning there may be significantly fewer positive (TB-infected) cases compared to negative (non-TB) cases. This can lead to biased models favouring the majority class and reduced sensitivity in detecting TB cases.
2. **\*\*Variability in Image Quality\*\*:** Chest X-ray images can vary significantly in quality due to factors such as equipment differences, patient positioning, and image artifacts. Models trained on data from one source may not generalize well to images from different sources or with varying quality.
3. **\*\*Limited Generalization\*\*:** Deep learning models trained on data from one population may not generalize well to other populations with different demographics, genetic backgrounds, or TB prevalence rates. This is particularly problematic when deploying models in regions where TB manifests differently or where resources for diagnosis are limited.
4. **\*\*Co-occurrence with Other Conditions\*\*:** TB often co-occurs with other pulmonary conditions such as pneumonia, lung cancer, or HIV-related infections. Deep learning models may struggle to differentiate between TB and these similar conditions, leading to false positives or negatives.
5. **\*\*Interpretability\*\*:** Deep learning models are often considered "black boxes," meaning it's challenging to understand the reasoning behind their predictions. In medical applications like TB detection, interpretability is crucial for clinicians to trust and effectively use these models as diagnostic aids.
6. **\*\*Ethical Considerations\*\*:** Deploying deep learning models for TB detection raises ethical concerns regarding patient privacy, consent, and the potential for biased decision-

making, particularly in vulnerable populations. Ensuring fairness and equity in model development and deployment is essential.

1. **\*\*Resource Requirements\*\*:** Deep learning models for TB detection typically require significant computational resources for training and .0inference, which may be prohibitive in resource-limited settings where TB is prevalent.
2. **\*\*Regulatory Approval\*\*:** Obtaining regulatory approval for deep learning-based medical devices can be a lengthy and complex process, requiring robust clinical validation studies to demonstrate safety and efficacy.

## CHAPTER 10 CONCLUSION AND FUTURE WORK

### Conclusion:

In conclusion, our study demonstrates the effectiveness of deep learning techniques in detecting tuberculosis from chest X-ray images. Through rigorous experimentation and evaluation, we have shown that our proposed model achieves state-of-the-art performance in terms of accuracy, sensitivity, and specificity. The utilization of convolutional neural networks (CNNs) coupled with advanced image processing techniques has enabled us to effectively extract features indicative of tuberculosis infection, leading to highly accurate classification results.

Furthermore, our study underscores the importance of leveraging deep learning in medical diagnostics, particularly in resource-constrained settings where access to expert radiologists may be limited. By automating the detection process, our model has the potential to significantly reduce the burden on healthcare systems and improve patient outcomes through early and accurate diagnosis.

However, despite the promising results, several challenges remain to be addressed. The generalizability of our model across diverse populations and imaging conditions warrants further investigation. Additionally, the ethical implications surrounding the deployment of AI- driven diagnostic tools must be carefully considered, ensuring patient privacy and maintaining transparency in decision-making processes.

Looking ahead, future research directions may involve the integration of multimodal data sources, such as clinical metadata or additional imaging modalities, to enhance diagnostic accuracy further. Moreover, the development of interpretable deep learning models could facilitate better understanding and trust among clinicians, thereby fostering widespread adoption in clinical practice.

### Future work:

The future of tuberculosis (TB) detection using deep learning and chest X-ray images holds significant promise for improving both accuracy and efficiency in diagnosing this infectious disease.

Here are some potential avenues for future work:

* **Large-Scale Datasets:** Continued efforts should be made to create and curate large-scale datasets of chest X-ray images annotated for TB. These datasets should encompass diverse populations and a wide range of TB manifestations to ensure the robustness and generalizability of deep learning models.
* **Model Development and Optimization:** Further research is needed to develop and optimize deep learning architectures specifically tailored for TB detection from chest X- ray images. This includes exploring novel network architectures, such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), or attention mechanisms, to improve model performance.
* **Transfer Learning and Pretraining:** Transfer learning techniques can be employed to leverage pretrained models on large-scale datasets from related tasks, such as general medical image classification or pneumonia detection. Fine-tuning these pretrained models on TB-specific datasets can help improve detection accuracy, especially in scenarios with limited labelled data.
* **Integration with Clinical Workflow:** Seamless integration of deep learning-based TB detection systems into existing clinical workflows is essential for real-world deployment. This involves developing user-friendly interfaces, ensuring interoperability with existing healthcare systems, and addressing regulatory and ethical considerations related to medical AI applications.
* **Collaboration and Validation Studies:** Collaboration between researchers, healthcare professionals, and policymakers is critical for conducting large-scale validation studies and evaluating the clinical impact of deep learning-based TB detection systems. These studies should assess not only diagnostic accuracy but also factors such as cost-effectiveness, patient outcomes, and scalability.

## CHAPTER 11 REFERENCES

1: Sajedeh Samadzadeh, Hadi Seyedarabi, Samaneh Mahdavi, “ *Tuberculosis Detection Using Deep Learning from Chest X-ray Images* ”, 25th International Computer Conference, Computer Society of Iran (CSICC), 2020 <https://ieeexplore.ieee.org/document/9278389>

2: Brijesh B. Mehta, Shadab A. Bhat, “ *Deep Learning Based Tuberculosis Detection Using Chest X-ray Images* ”, International Conference on Communication and Signal Processing (ICCSP), 2018 <https://ieeexplore.ieee.org/document/8370302>

3: Jaykumar Thumar, Rupam Dey, Ravi Kumar Satzoda, “ *Tuberculosis Detection from Chest X-ray Images Using Deep Learning Techniques* ”, IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS), 2020

<https://ieeexplore.ieee.org/document/9278391>

4: Muhammad Hanif, Muhammad Zeshan Afzal, Umer Ijaz, et al., “ *Automatic Tuberculosis Detection Using Convolutional Neural Networks* ”, International Conference on Advanced Communication Technologies and Networking (CommNet), 2019

<https://ieeexplore.ieee.org/document/8871831>