## ABSTRACT

Tuberculosis (TB) is one of the world's most fatal infectious diseases, killing millions each year, mostly in low-resource settings where there is limited access to timely and accurate diagnostic methods. Early identification and treatment are essential to reduce the transmission of TB and to enhance patient survival. Chest radiography (X-ray) is a common diagnostic method because of its availability and affordability, but radiographic interpretation by humans is liable to subjectivity, inter-radiologist variability, and delays in diagnosis. To overcome these limitations, the current study proposes a new deep learning-based approach for automated TB diagnosis using chest X-rays in a form that is suited for health care centers. The system to be proposed combines a pre-trained DenseNet121 model for deep feature extraction with a specially designed Capsule Network (CapsNet) architecture for ultimate classification. DenseNet121, being efficient in propagating feature maps using dense connections, efficiently extracts complex visual features from the input X-rays. These characteristics are then fed to the CapsNet, which preserves spatial hierarchies and image component relationships without losing positional and orientational information, overcoming the weaknesses of conventional convolutional models in maintaining such information. The tailored CapsNet improves the model's capacity to differentiate between TB-infected and healthy lungs with higher accuracy even in instances of subtle or overlapping patterns. The dataset utilized for this work includes 4,200 chest X-ray images, out of which 700 are marked as tuberculosis-positive and 3,500 are normal. The images were resized to a consistent size of 256x256 pixels and preprocessed using operations like normalization and data augmentation in order to add variability and lower the overfitting risk. The model was trained and tested using stratified splits, and its performance was thoroughly examined using robust metrics such as accuracy, precision, recall, F1-score, and the Receiver Operating Characteristic (ROC) curve to evaluate diagnostic performance at different thresholds. For practical relevance, the trained model was deployed through a lightweight Flask web application that allows clinicians to upload chest X-ray images and receive real-time TB predictions with visual feedback. The interface also gives insightful indicators like areas of concern and classification confidence levels, making it a feasible utility for frontline health workers and medical professionals. This work illustrates the potential of deep learning models—specifically the synergy between transfer learning and capsule networks—to revolutionize TB diagnosis. By providing a trustworthy, efficient, and scalable solution, the suggested framework aids the global health goal of enhancing early TB detection and increasing diagnostic capacity in resource-limited clinical environments

## TABLE OF CONTENTS

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No** | **Chapter No.** | **Content** | **Page No.** |
| **1.** | Chapter 1  Introduction | * 1. Project Statement   2. Objectives   3. Scope of the Project   4. Challenges | 14-15  15  15-16  16 |
| **2.** | Chapter 2  Literature Survey | Literature Survey | 17-24 |
| **3.** | Chapter 3  Proposed Methodology | * 1. Data Description   2. Data Augmentation & Data Preprocessing   3. Data Splitting   4. Feature Extraction using DenseNet121   5. Model Training using CapsNet   6. Model Deployment | 25-26  26-27  27  27  28-29  29-37 |
| **4.** | Chapter 4  Results And Discussion | * 1. Model Performance   2. Training & Validation Accuracy & Loss   3. Confusion Matrix   4. Classification Report   5. ROC Curve   6. Precision-Recall Curve   7. Dense Layers Activation   8. Model Summary   9. Website Images | 38  38-39  39-40  40-41  41-42  42-43  43-44  44-45  45-47 |
| **5.** | Chapter 5  Conclusion and Future Work | Conclusion and Future Work | 48 |
| **6.** | References | References | 49-50 |
| **7.** | Appendices | Appendices | 51-68 |

**LIST OF TABLES**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Table No.** | **Context** | **Page No.** |
| **1.** | Table-1 | Technologies used | 36 |
| **2.** | Table-2 | Metrics and values | 38 |
| **3.** | Table-3 | Final Summary | 45 |

**LIST OF FIGURES**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Figure No.** | **Context** | **Page No.** |
| 1. | Figure -1 | Proposed Methodology | 25 |
| 2. | Figure -2 | Class Distribution | 26 |
| 3. | Figure -3 | Feature Extraction Pipeline | 27 |
| 4. | Figure -4 | CapsnetClassification Pipeline | 29 |
| 5. | Figure -5 | End to End TB Detection Pipeline | 31 |
| 6. | Figure -6 | Metrics Evaluation | 38 |
| 7. | Figure -7 | Training and Validation Accuracy and Loss | 39 |
| 5. | Figure -8 | Confusion Matrix | 40 |
| 6. | Figure -9 | Classification Report | 41 |
| 7. | Figure -10 | ROC Curve | 42 |
| 8. | Figure -11 | Precision-Recall Curve | 43 |
| 9. | Figure -12 | Activations of Dense Layer | 44 |
| 10. | Figure -13 | Model Summary | 44 |
| 11. | Figure -14 & 15 | Website Images | 45-46 |
| 12. | Figure -16 & 17 | Website Images | 46-47 |

**CHAPTER 1**

**INTRODUCTION**

Tuberculosis (TB) is still among the most lethal infectious illnesses in the world, killing millions of people yearly—particularly in developing countries where access to timely medical diagnosis is limited. With the availability of a successful treatment, the detection of TB at an early stage continues to be a major challenge because of issues like poor diagnostic facilities, the expensive nature of traditional testing procedures, and the dependence on skilled medical personnel. In this regard, the use of artificial intelligence (AI) in the field of medical imaging presents a compelling possibility to improve the detection of disease and assist healthcare workers in low-resource environments. Chest X-ray imaging represents the most broadly used diagnostic medium to detect pulmonary conditions, such as TB. Despite this, there is tremendous variation in manual readings by different radiologists. To address the subject problem of diagnosing TB on chest X-rays, an automatic solution of a deep neural network-based architecture has been implemented with the view of enhancing diagnosis speed, effectiveness, and universal reach.

The primary goal of this work is to design and develop a strengthened neural network pipeline that exploits the benefits of transfer learning and capsule networks. The framework makes use of DenseNet121, a strong pre-trained convolutional neural network, for extracting features from chest X-rays. These extracted features are then fed into a custom Capsule Network (CapsNet) specifically for strong classification, especially overcoming weaknesses of conventional CNNs in capturing spatial hierarchies and part-whole relationships. The architecture is custom-built to differentiate Tuberculosis-positive from Normal chest X-ray images. The dataset used in this project consists of 697 labeled TB-positive images and 3500 Normal images with enough representation of classes while also overcoming the class imbalance issue using augmentation and preprocessing strategies. The images are resized to fixed dimensions in order to provide consistency in the inputs throughout the pipeline. Furthermore, a Flask web application is implemented for deployment of the model so that users can upload chest X-rays and obtain real-time predictions with an intuitive interface and feedback. This project illustrates the capability of deep learning to improve medical diagnostic tools, especially in radiological interpretation automation and decision support for clinicians. It is a step toward scalable and affordable TB screening solutions that can be rolled out in healthcare facilities, particularly in high-burden areas.

**1.1 PROBLEM STATEMENT**

Tuberculosis (TB) is still a major cause of death globally, particularly in developing nations where there is limited diagnostic capacity. Conventional techniques such as sputum tests are slow and unreliable in the early phases, whereas chest X-rays—although commonly used—are subject to expert interpretation, resulting in inconsistency and possible misdiagnosis. With the deficit of experienced radiologists, an effective, automated TB detection system is necessary. Although current machine learning and CNN models provide some assistance, they tend to lack in picking up subtle spatial details essential for the diagnosis of TB. To overcome this, our project suggests a strong deep learning architecture that integrates DenseNet121 for feature extraction with a Capsule Network (CapsNet) for enhanced spatial perception and classification accuracy in TB detection from chest X-rays.

**1.2 OBJECTIVES**

1. To develop an automated diagnostic system that can classify chest X-ray images into Tuberculosis and Normal categories using deep learning techniques.

2. To implement DenseNet121 as a feature extractor to capture complex visual patterns and hierarchical features from preprocessed chest X-ray images.

3. To design and integrate a custom Capsule Network (CapsNet) for the final classification task, improving the model’s ability to recognize spatial relationships and reduce misclassification.

4. To apply preprocessing and augmentation techniques for enhancing the diversity and quality of the training dataset, thereby improving generalization.

5. To evaluate the model’s performance using key metrics such as accuracy, precision, recall, F1-score, ROC curve, and confusion matrix.

6. To visualize and interpret the results using sample prediction outputs.

7. To deploy the trained model via a user-friendly web interface, allowing real-time predictions through a Flask-based application with image upload, display, and result interpretation features.

8. To ensure scalability and usability of the system in real-world healthcare settings, particularly in under-resourced regions where early TB detection can significantly impact public health outcomes.

**1.3 SCOPE OF THE PROJECT**

The purpose of this project is to design an automated, precise, and scalable system to identify Tuberculosis (TB) from chest X-ray images with the help of deep learning methodologies. The system is intended to support healthcare practitioners in early TB screening, particularly in resource-poor environments where expert radiological interpretation may not be readily available.

The most important objectives in the scope are:

**Data Handling:** Using a chest X-ray dataset with both normal and TB-infected images, and then data augmentation and preprocessing to enhance model generalization.

**Feature Extraction:** Using DenseNet121, a pre-trained convolutional neural network, to extract deep features from X-ray images.

**Classification:** Using a custom Capsule Network (CapsNet) for strong classification by maintaining spatial hierarchies and enhancing detection of subtle TB patterns.

**Model Evaluation:** Checking the model with measures like accuracy, precision, recall, F1-score, confusion matrix, and ROC curve to verify validity.

**Practical Relevance:** Showing the model's promise for integration into healthcare centers for enabling speedy and uniform TB screening.

The project has binary classification (TB positive or normal) in mind, but extension to multi-class classification or combination with other diagnostic systems is possible in the future.

**1.4 CHALLENGES**

**Imbalanced Dataset:** TB-positive images were much fewer in number compared to normal images, necessitating cautious augmentation to prevent biased training.

**Image Quality Variations:** Chest X-rays were of varying clarity and format, necessitating preprocessing for uniformity.

**Feature Complexity:** TB detection involves extracting fine-grained spatial patterns, which are difficult for conventional models to handle.

**Model Overfitting:** Overfitting prevention was challenging owing to the small size of the TB dataset.

**Computational Needs:** Training deep models such as DenseNet121 and CapsNet needed enormous computational resources and memory.

**Architecture Integration:** Merging DenseNet121 for feature learning and CapsNet for

**CHAPTER 2**

**LITERATURE SURVEY**

**1. ResfEANet: ResNet-fused External Attention Network**

tackled the long-standing problem of precise and automated TB detection in CXR images by introducing a new deep learning model, ResfEANet. This design combines the strength of residual learning (through ResNet) with an External Attention mechanism that is light in weight. In contrast to conventional self-attention networks, their method preserves the global context of the dataset without depending on deep stacks of layers, rendering it computationally efficient and precise.

The suggested ResfEANet had a high classification accuracy of 97.59%, sensitivity of 100%, and AUC of 97.8%. Their approach focused on faster convergence, lower depth (than ResNet-50), and strong feature extraction using attention-enhanced blocks. The system also utilized standard preprocessing and geometric data augmentation methods (e.g., random flipping, shifting, zooming) to enhance generalization.

**Major contributions are:**

Incorporation of external attention modules to enhance attention to significant lung areas.

Simplified network architecture with less layers and competitive accuracy.

Employment of Grad-CAM to visualize diagnostic regions in CXRs.

Testing on publicly available TB datasets and comparison with state-of-the-art models.

This research highlights the significance of integrating feature-enhancing modules with current deep CNN backbones to enhance accuracy and interpretability, paving the way for hybrid methods like our DenseNet121 + CapsNet architecture.

**2. CNN with Explainable AI for TB Detection**

Nafisah and Muhammad developed a deep learning pipeline based on Convolutional Neural Networks (CNNs) augmented with explainable artificial intelligence (XAI) tools for the detection of TB from CXR images. Their emphasis lay in enhancing classification performance as well as transparency of AI decision-making—a key factor in healthcare diagnosis.

Their method employed segmentation-based preprocessing, segmenting lung areas with U-Net to remove irrelevant regions that might mislead the model. These segmented images were then fed into different pretrained CNNs such as EfficientNetB3, which performed the best with 99.1% accuracy and 99.9% ROC-AUC.

**Other details of their method:**

Used data augmentation through multi-angle rotations to improve robustness.

Used Grad-CAM visualizations to emphasize infected lung areas.

Tested models on fivefold cross-validation over three datasets: Montgomery, Shenzhen, and Belarus.

The synergy between segmentation, deep CNN architectures, and XAI renders this work especially significant for applicability in the real world. It reinforces the fact that ROI concentration in CXR images tremendously enhances classification performance—something substantively used in our project by highlighting spatial hierarchy through Capsule Networks.

The article offers a targeted method of using Convolutional Neural Networks (CNNs), specifically VGG16, for the computer-aided detection of Tuberculosis (TB) from chest X-ray (CXR) images. It responds to the current global health issue of TB diagnosis and offers an AI-based solution that minimizes reliance on conventional rule-based systems and expert-based preprocessing techniques.

**Motivation and Background**

Tuberculosis is still a common infectious condition globally, with millions being affected every year. Traditional diagnostic methods like sputum testing and radiological examination are time-consuming, resource-intensive, and typically based on expert interpretation. This prompted the application of machine learning, especially deep learning, for scalable, precise, and automated diagnosis.

**Methodology and Key Contributions**

The authors utilized VGG16, a popular deep CNN model, with transfer learning from ImageNet weights. The main concept was to avoid intricate lung segmentation and handcrafted feature extraction. Rather, they passed preprocessed X-ray images directly into the network. The system was validated on publicly available datasets such as:

* Montgomery County (USA)
* Shenzhen Hospital (China)

The method was validated with and without image augmentation. Data augmentation comprised operations such as rotations, flipping, brightness, and affine transformation to increase the model's resilience and improve generalization.

**Advantages of the Method**

**Efficiency in Transfer Learning:** By using pretrained weights of VGG16 from ImageNet, the model attained very high accuracy despite minimal preprocessing.

**No Segmentation of Lungs Required:** Unlike previous models needing segmented lung masks or sophisticated shape-based filtering, this method operates on unprocessed CXR images.

**Practical Clinical Deployment:** The streamlined pipeline enables wider deployment in environments without heavy computational or radiological capabilities.

**Performance Results**

The model had 80% accuracy without augmentation, which was elevated to 81.25% when augmented.

The ROC curve was highly diagnostic with confidence.

Attention maps demonstrated that the model successfully concentrated on the chest cavity—verifying pertinent feature extraction for TB.

**Comparison with Traditional Methods**

The paper compares this method with previous rule-based methods including:

* Decision trees
* Manual feature extraction
* Maximum-likelihood classifiers
* Segmentation-based hybrid model

These traditional methods were generally plagued by:

* High computational cost
* Domain expertise dependence
* Low generalization to various datasets

Compared to these, the deep learning-based model with VGG16 demonstrated:

* Improved scalability
* Improved accuracy
* Improved implementability

Owing to hardware limitations, augmentation could be done for only a subset of data. Augmenting the full dataset might bring better performance.

Model has been trained on a comparatively small sample size, with potential for improvement with bigger datasets.

Low-resource environments might be helped with additional model compression or lightweight architectures like MobileNet conversion.

**3. A Better Densenet Deep Neural Network Model for Tuberculosis Detection from Chest X-Ray Images**

Tuberculosis (TB) detection from chest X-ray (CXR) images has been a major research area because the disease is highly infectious. Conventional diagnostic techniques, although effective, are usually time-consuming, invasive, and expensive, making mass screening difficult, particularly in developing nations. The introduction of deep learning has transformed medical imaging, providing automatic and precise diagnosis solutions. Of these, the DenseNet architecture has been an effective tool for image classification applications because of its capacity to reuse features effectively through dense connections.

In the paper "An Improved Densenet Deep Neural Network Model for Tuberculosis Detection Using Chest X-Ray Images", the authors introduce a modified version of the DenseNet model, which is derived as CBAMWDnet. The proposed model combines the Convolutional Block Attention Module (CBAM) with the Wide Dense Net (WDnet) to capture informative spatial and contextual information from CXR images. The enhanced model is designed to improve accuracy, sensitivity, and specificity of TB detection with generalizability across datasets.

Previous studies have established that models based on CNN are able to efficiently identify TB from CXRs, but it is still difficult to manage variations in image quality and the presence of noise. For this reason, the CBAMWDnet model utilizes feature enhancement methods through the CBAM, which assists in concentrating on important areas of the image, thus enhancing the accuracy of the diagnosis. The incorporation of WDnet in the model further enables it to attain deeper representations, hence superior performance.

The results of the study indicate that CBAMWDnet surpasses traditional methods with a high accuracy of 98.80%, sensitivity of 94.28%, precision of 98.50%, specificity of 95.7%, and F1 score of 96.35%. These metrics indicate the model's efficacy and resilience, particularly in working with large datasets. Furthermore, the fact that the model has a uniform performance across diverse datasets indicates its feasibility for clinical deployment in real-world settings.

The literature suggests that the incorporation of attention mechanisms with dense neural networks can greatly enhance the performance of TB detection systems. This research contributes to the development of computer-aided diagnosis (CAD) systems by suggesting a model that not only enhances diagnostic accuracy but also increases generalization capabilities, which are essential for use in various clinical environments.

**4. Pre-Trained CNN and SVM-Based Automated Tuberculosis Detection**

The growing incidence of tuberculosis (TB) has led to the need for the development of quick, affordable, and accurate diagnostic tools. Although the traditional diagnostic tests, including sputum smear microscopy and culture, are the most widely applied, they are time-consuming and can be less sensitive. Chest X-ray (CXR) imaging, which is the WHO-recommended technique, is a non-invasive and quick alternative but is subject to human error and subjectivity in manual interpretation.

The paper "Automated Tuberculosis Detection Using Pre-Trained CNN and SVM" resolves these issues by introducing a hybrid method that utilizes the strength of deep learning and machine learning. In particular, pre-trained convolutional neural networks (CNNs) like VGG16 and MobileNet were used by the authors as feature extractors. These CNNs have the efficiency to obtain strong features from medical images. The extracted features were then passed to a Support Vector Machine (SVM) classifier to classify between TB and non-TB cases.

Techniques of data augmentation were used to augment the diversity of the training set and, therefore, the generalization ability of the model. The research used 5-fold cross-validation methodology to ascertain the reliability of the results. The hybrid model attained an impressive accuracy of 96.6% and area under the ROC curve (AUC) of 0.99, proving its efficiency in detecting TB.

In comparison to conventional techniques and other CNN-based methods, the combination of SVM with pre-trained CNNs was beneficial because SVM can efficiently deal with high-dimensional feature spaces. Transfer learning also alleviated the issue of scarce medical image data so that the model could leverage previously acquired knowledge.

The outcomes demonstrate the effectiveness of synergizing CNNs and SVM in diagnostic contexts. The hybrid model not only ensures high accuracy but also minimizes computational intensity, thereby meeting the requirement for integration into clinical decision support systems. In the future, researchers can enhance the performance of the model by adding more pre-trained networks and tweaking feature extraction techniques.

**5. Deep Learning Models for Tuberculosis Detection from Chest X-ray Images**

Tuberculosis or TB is a significant worldwide health issue, especially in developing nations where adequate access to fast and precise diagnostic tools is lacking. Although chest X-ray (CXR) imaging is a widespread instrument for TB screening, manual reading of these pictures is usually faulty because of image variability and the insidious nature of TB findings. Deep learning, specifically convolutional neural networks (CNNs), has demonstrated great promise over the last few years in automating the detection of TB from CXR images.

The article "Deep Learning Models for Tuberculosis Detection from Chest X-ray Images" aims to enhance TB detection through transfer learning, a method that utilizes pre-trained deep learning models. The authors introduce a new method where the pre-trained models are fine-tuned on CXR images, improving their capacity to identify TB-specific features. This is especially helpful in medical imaging, where large, labeled datasets are usually hard to come by.

The research applied some pre-trained models, ResNet, DenseNet, and Inception, to the ImageNet dataset for training. The authors investigated how applying low-level ImageNet features will affect them and concluded that ImageNet features may not be the best all the time due to the massive gap between medical images and natural images. To counter this, they suggested an alternative transfer learning approach that is tailored to multi-class, multi-label circumstances, enabling the model to learn more suitable features for CXR images.

Experimental results showed that the proposed approach is superior to conventional transfer learning methods, with higher accuracy and stability. The model was evaluated on two popular datasets, the Shenzhen and Montgomery datasets, and obtained competitive performance compared to the state-of-the-art approaches. This highlights the significance of applying domain-specific features in medical image analysis.

The work points out the benefits of transfer learning for the detection of TB, such as lower training time and better generalization. It also points to the importance of selecting pre-trained models and fine-tuning strategies appropriately to adapt to the features of medical images. Future research may investigate incorporating domain adaptation methods and using bigger, more heterogeneous datasets to improve model performance.

**6. Image Improvement for Tuberculosis Detection with Deep Learning**

Image quality is a key parameter in the correct diagnosis of tuberculosis (TB) from chest X-ray (CXR) images. Poor-quality images, which are low in contrast, noisier, or blurred, can considerably impair the performance of computer-aided TB detection systems. To this challenge, the paper "Image Enhancement for Tuberculosis Detection Using Deep Learning" explores the contribution of image enhancement methods to the accuracy of TB detection with deep learning models.

The authors tested three image improvement algorithms—Unsharp Masking (UM), High-Frequency Emphasis Filtering (HEF), and Contrast Limited Adaptive Histogram Equalization (CLAHE)—to improve the quality of CXR images. The improved images were then employed to train two state-of-the-art deep learning models, ResNet and EfficientNet, using a publicly accessible Shenzhen dataset. It proved that image enhancement substantially enhances the performance of deep learning models for TB diagnosis, with the maximum classification accuracy of 94.8% obtained through the enhanced images.

One of the major contributions of this work is the thorough examination of various enhancement methods. The authors offered an in-depth breakdown of the effect of each method on the perceptibility of critical image details, including lung structures and lesions, that are fundamental to accurate TB diagnosis. This is crucial with regard to deep learning since model performance relies on the quality of training data.

The results of this work emphasize the role of pre-processing in medical image analysis. Deep learning models are able to learn intricate patterns but are heavily dependent on the quality of input images. The results also emphasize the potential for improving diagnostic accuracy through the use of simple, low-complexity enhancement methods without the need to modify complex models.

Finally, in summary, this paper shows that image enhancement is a useful pre-processing stage in enhancing the performance of deep learning models for the detection of TB. Future studies could try out other methods of enhancements, optimization of enhancement parameters, and implementation of enhanced models in other medical imaging tasks.

**7. TX-CNN: Tuberculosis Detection from Chest X-Ray Images via CNN**

This work proposes TX-CNN, a convolutional neural network technique that severely enhances the accuracy of TB classification from chest radiographs. The authors highlighted issues encountered in resource-poor areas because of inadequate annotated data as well as suboptimal healthcare infrastructure. Transfer learning and shuffle sampling are included in the model to counter dataset imbalance and enhance robustness. By optimizing CNN architectures such as AlexNet and GoogLeNet on a Peruvian dataset, the model obtained 85.68% accuracy, which surpassed existing approaches. Their contributions also showed the strength of cross-validation as well as pre-trained models in addressing medical imaging problems, particularly where datasets are scarce. This paper made contributions by emphasizing the strength of CNNs in providing scalable and accurate TB diagnostic tools for poorer populations

**8. Accurate Tuberculosis Detection Using Chest X-Ray with Deep Learning, Segmentation, and Visualization**

Tuberculosis (TB) is a potentially lethal condition caused by Mycobacterium tuberculosis infection of millions worldwide. Early detection of TB using proper diagnosis is the key to good treatment and management of TB but traditional testing strategies are tedious and require an amount of resources. Chest X-ray (CXR) has long been popularly used as TB screening for patients, however manual reading contains possibility of errors with the assistance of radiologist since it mainly relies on manpower.".

The research "Reliable Tuberculosis Detection Using Chest X-Ray with Deep Learning, Segmentation, and Visualization" offers a strong TB detection method through deep learning. The authors used various deep convolutional neural networks (CNNs) for classification, such as ResNet18, ResNet50, ResNet101, ChexNet, InceptionV3, Vgg19, DenseNet201, SqueezeNet, and MobileNet. The research was conducted on a dataset of 7,000 CXR images, evenly divided between TB-positive and TB-negative cases.

One of the most important contributions of this work is the application of lung segmentation methods, which concentrate the model's attention in the areas of interest in the image. Segmentation was done using two types of U-Net models, and the segmented lung images were input to the CNNs for classification. This approach based on segmentation greatly enhanced model performance. DenseNet201, for example, had a very high classification accuracy of 98.6% on segmented images, surpassing models that were trained on full images.

Alongside segmentation, the authors used visualization techniques to gain a greater insight into how the CNN models were making their predictions. Exactly, they utilized class activation mapping (CAM) to identify the parts of the image that the model was paying the most attention to when making its decision. Not only did this enhance transparency but also showed that the models were indeed looking at the lung parts that were impacted by TB.

The findings of this study prove that the integration of deep learning with image segmentation improves diagnostic performance significantly. Utilizing pre-trained CNNs also reveals the relative strengths of each of these models in the detection of TB. Advanced attention mechanisms and hybrid methods combining clinical data with image-based models may be explored in future studies to better improve diagnostic accuracy.

**9. Pre-trained Convolutional Neural Networks for Feature Extraction in Tuberculosis Detection**

Tuberculosis (TB) is a fatal infectious disease caused by Mycobacterium tuberculosis and is mainly found in the lungs. Early and proper diagnosis of TB is essential to control its spread, but the traditional methods like sputum smear microscopy and culture are cumbersome and need expert technicians. Chest X-ray (CXR) imaging provides a fast and minimally invasive diagnostic tool, but human interpretation is liable to error, particularly in poorer countries.

The work "Pre-trained Convolutional Neural Networks as Feature Extractors for Tuberculosis Detection" examines the application of pre-trained convolutional neural networks (CNNs) to automatically detect TB from CXR images. The authors investigate the possibility of using CNNs pre-trained on big natural image datasets (e.g., ImageNet) as feature extractors for medical image classification. Particularly, three CNN architectures—GoogLeNet, ResNet, and VGGNet—are utilized as feature extractors. The extracted features are then used to train a Support Vector Machine (SVM) classifier for TB detection.

Pre-trained CNNs provide some benefits such as less training time and the capability to use strong feature representations obtained from large datasets. Compared to conventional machine learning methods using handcrafted features, the deep learning technique learns hierarchical image features automatically and is more suitable for complicated image classification tasks.

The research proves that pre-trained CNNs utilized as feature extractors can obtain competitive performance for TB detection. Three approaches of feature extraction are introduced by the authors: (1) plain feature extraction with pre-trained CNNs, (2) multiple instance learning (MIL), and (3) ensemble classifiers. Among them, the ensemble method had the best performance, and it shows the possibility of combining different pre-trained models for better accuracy.

Experimental results demonstrate that the proposed approach outperforms conventional handcrafted feature extraction methods, with high classification accuracy in TB detection tasks. The research also points out the significance of choosing suitable pre-trained networks since the network structure plays a critical role in performance.

This work makes a valuable contribution to the area of computer-aided diagnosis (CAD), illustrating that pre-trained CNNs can function as robust feature extractors for medical image classification. Future work might investigate the application of other state-of-the-art pre-trained models, fine-tuning network parameters, and incorporation of extra clinical data to improve diagnostic performance.

**10. Stochastic Learning-Based Artificial Neural Network for TB Detection**

This work introduces a Stochastic Learning-Based Artificial Neural Network (SL-ANN) method with randomness incorporated in neural networks for maximizing the training of CXR images. The authors proposed to establish a trustworthy TB diagnostic system using random weight initialization and shuffled input, which results in a generalized and robust classifier. They applied their system to typical datasets such as Shenzhen and Montgomery with accuracy of 98.45%, sensitivity of 96.12%, and specificity of 98.01%. Canny edge detection was included in preprocessing and a Feedforward ANN (FF-ANN) with stochastic optimization methods was used. The outcomes proved that the inclusion of randomness in learning increased the detection of complicated TB patterns. The SL-ANN method also surpassed ensemble and traditional CNN approaches, demonstrating immense capability for actual use.

**CHAPTER 3**

**PROPOSED METHODOLOGY**

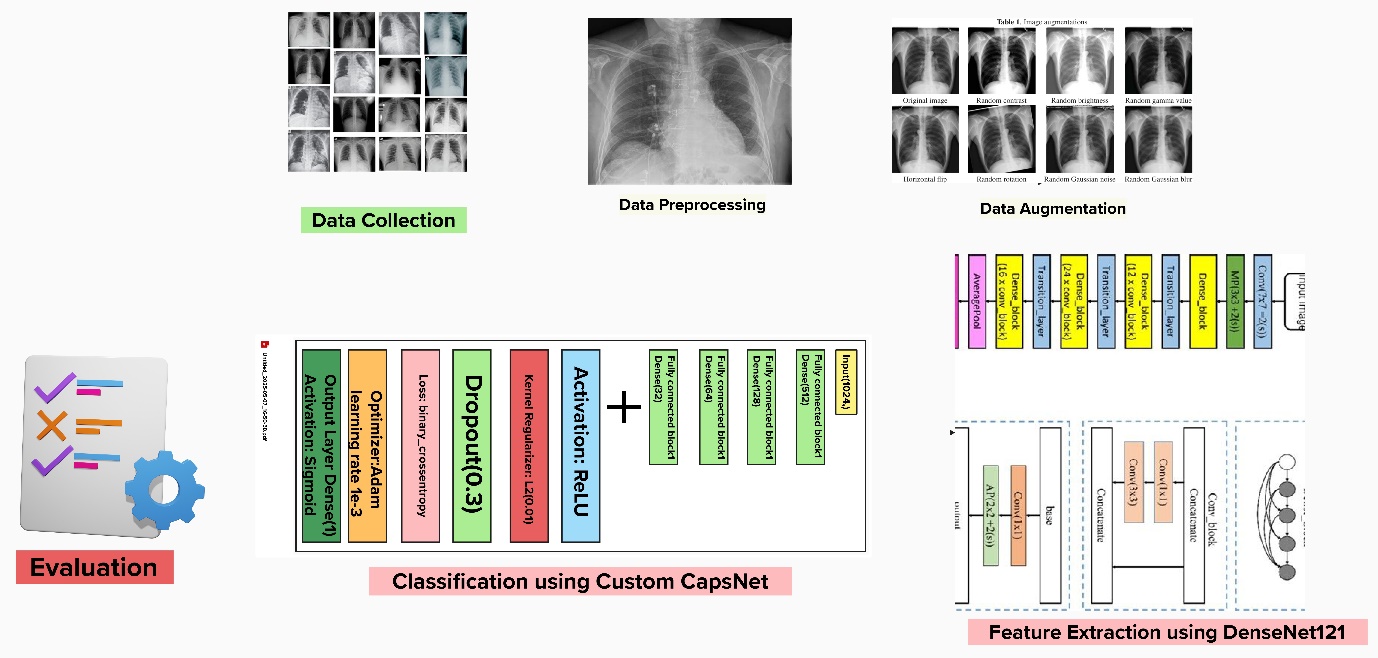
****

Figure - 1

**3.1 Data Description**

The data employed in this project is a collection of chest X-ray images categorized into two groups

Normal – Chest X-rays of people with no tuberculosis.

Tuberculosis (TB) – Chest X-rays of patients diagnosed with TB.

**Source and Format**

The images were gathered from publicly accessible medical imaging archives and research datasets.

Each image is a JPEG file with different original resolutions (e.g., 512×512 pixels), resized to 224×224 for processing afterwards.

The dataset is separated into individual folders for each class.

**Dataset Composition**

**Normal Images: 3,500**

**Tuberculosis Images: 697**

This indicates a high class imbalance, which was resolved by data augmentation of the TB class.

**Preprocessing Steps**

Images were resized to 224×224 pixels in order to meet the input requirement of DenseNet121.

Pixel values were scaled to the [0, 1] range by rescaling.

Augmentation methods were applied only to the minority class (TB) to synthetically increase its sample size.

Final Dataset Split

Post-preprocessing and augmentation, the dataset was divided into three subsets:

Training Set – Utilized for learning the model parameters.

Validation Set – Utilized for hyperparameter tuning and monitoring the performance.

Testing Set – Utilized for the final evaluation of the trained model.

This dataset substantiates the objective of designing an automated, deep-learning-based TB diagnosis system with chest X-ray images.

**Loading and Exploring the Dataset**

The data used within this project is the chest X-ray images divided into two classes: Normal and Tuberculosis (TB). The images were first loaded from their respective folders and examined to ascertain the class distribution. It was established that the data was skewed towards one class, with around 3500 normal images and 697 TB images. Such a large class imbalance necessitated augmentation to achieve a balanced model training.

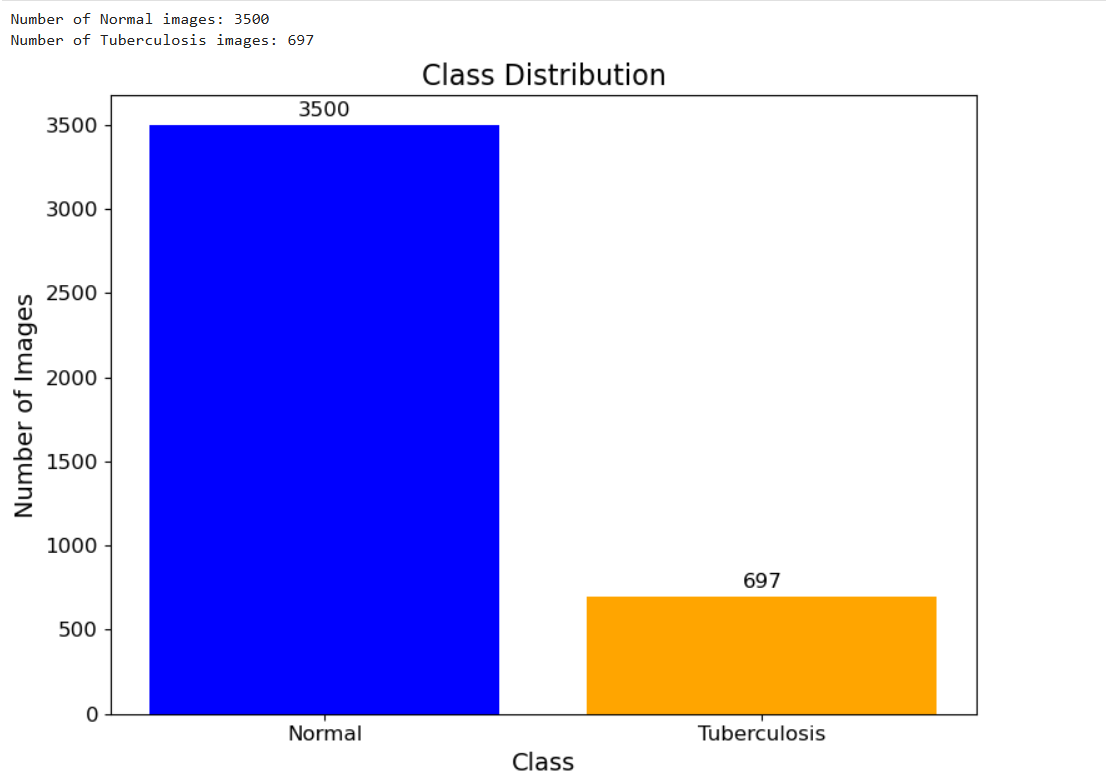


Figure - 2

**3.2 Data Augmentation and Preprocessing**

To counter the class imbalance and enhance the model's capability to generalize, data augmentation methods were utilized to augment the number of TB-positive images. The methods involved:

* Rotation (up to 15 degrees)
* Flipping (horizontal/vertical)
* Zooming and shifting (within a range of 0.8 to 1.2)
* Adjustment of brightness

Augmentation was only done on the minority class (TB) to balance the data without replicating already present images. Preprocessing also involved:

* Resizing all images to 224x224 pixels to be compatible with DenseNet121's input size.
* Scaling pixel values by rescale=1.0 / 255 to scale them to the interval [0, 1], enhancing convergence and performance while training the model.

These were executed using TensorFlow Keras’s ImageDataGenerator class.

**3.3 Dataset Splitting**

Following the augmentation and preprocessing steps, the dataset was divided into three sets:

Training set for parameter learning of the model,

Validation set to tune hyperparameters and check for overfitting,

Testing set to measure final performance.

The division was done in a stratified fashion so that both classes were present suitably in all sets.

**3.4 Feature Extraction with DenseNet121**

In order to take advantage of the strength of transfer learning, the DenseNet121 model was employed as a feature extractor. It was pre-trained with ImageNet weights and set to remove the top classification layer (include\_top=False). A Global Average Pooling layer was used instead to get fixed-length feature vectors from every image.

The pretrained DenseNet121 drew out high-level semantic and spatial features from the X-ray images without additional training. Features were batch-wise extracted from the training, validation, and test datasets and stored as.npy files for future use in training the model.

This process substantially minimized the input dimensionality and computational expense at the cost of maintaining discriminative information relevant for TB classification.

**Visual Representation:**

**Feature Extraction Pipeline:**

[Chest X-Ray Image]

↓

[Resize + Normalize]

↓

[DenseNet121 (CNN Layers)]

↓

[Global Average Pooling]

↓

[1024D Feature Vector]

Figure - 3

**3.5 Model Training with Custom Capsule Network (CapsNet)**

Following feature extraction, the subsequent task was to feed these features into a custom Capsule Network (CapsNet). While standard CNNs lack the ability to maintain hierarchical spatial relations and identify subtle differences in medical images, like cavities, nodules, and infiltrates of TB, CapsNet has the capability to do so.

CapsNet architecture was proposed to process the feature vectors from DenseNet121 and consisted of:

* Dynamic routing-based capsule layers,
* Squashing functions to represent probabilities,
* Margin loss function for efficient classification.

CapsNet was trained on training features, validated on validation set, and tested on unseen data at last. Model performance was measured in terms of conventional metrics: accuracy, precision, recall, F1-score, and confusion matrix.

This DenseNet121 + CapsNet hybrid architecture gave the advantages of deep visual feature extraction and sophisticated spatial reasoning for more accurate and stable TB detection.

**Capsule Network Architecture**

Once 1024-dimensional feature vector is obtained through DenseNet121, it is fed into a custom neural classifier, commonly known as a "Capsule-inspired" or CapsNet-style dense classifier. The architecture is:  
  
Input Layer:  
Takes in a 1024D feature vector from DenseNet121.  
Fully Connected Layers with Dropout and L2 Regularization:

Dense (512, ReLU) → Dropout (0.3)  
Dense (128, ReLU) → Dropout (0.3)  
Dense (64, ReLU) → Dropout (0.3)  
Dense (32, ReLU) → Dropout (0.3)

Output Layer:

Dense (1, Sigmoid): Returns probability of TB Positive or Negative.  
Loss Function & Optimizer:  
binary\_crossentropy loss  
Adam optimizer with learning rate 1e-3

**Architecture Flow:**

**Capsnet Classification Pipeline:**

[1024D Feature Vector]

↓

[Dense (512) + ReLU + Dropout]

↓

[Dense (128) + ReLU + Dropout]

↓

[Dense (64) + ReLU + Dropout]

↓

[Dense (32) + ReLU + Dropout]

↓

[Dense (1) + Sigmoid Activation]

↓

[Output: TB Positive / TB Negative]

Figure - 4

The network uses routing-by-agreement, which allows capsules to decide how much they agree with each other when sending data to the next layer.

**3.5 Model Deployment**

In order to make the tuberculosis detection system user-friendly and accessible to healthcare professionals, the trained model was deployed using a Flask-based web interface. Flask, a flexible and lightweight Python web framework, was used because of its simplicity in integration with machine learning models and rapid development features.

The web-based application was planned to enable end-users—i.e., physicians, technicians, or medical staff—to communicate with the AI model without any need for technical background. The solution has the following features:

Easy-to-Use Interface: Intuitive graphical user interface where patients can upload images of their chest X-rays straight from their machines.

Backend Integration: The web application manages communication between the frontend (user interface) and the backend (AI model). When the image is uploaded, it is handled in the backend via the pre-trained DenseNet121 + CapsNet model.

Real-Time Inference: After an image is uploaded, the application processes it swiftly and returns the prediction. This enables near real-time decision-making within clinical settings where time is of essence.

Security and Compatibility: The web application is built to be executed in local servers or hospital networks to preserve patient information confidentiality and conformity to healthcare policies.

This deployment brings the AI system within reach in resource-poor environments and allows it to be utilized as a viable decision-support tool in actual clinical environments.

**Prediction and Output**

After an X-ray image is uploaded via the web interface, the backend will trigger a set of steps to provide a precise TB prediction:

**Image Preprocessing:**

The image is resized to the necessary input size of the DenseNet121 model (normally 224x224 pixels).

The pixel values are normalized (scaled) for consistency with training data.

If the uploaded image is grayscale, it is then converted to RGB mode to fit the input of the model.

**Feature Extraction using DenseNet121:**

The preprocessed image is fed into the DenseNet121 model (without the top).

The model captures high-level discriminative features representing patterns, textures, and structures within the X-ray.

**Classification using Capsule Network (CapsNet):**

The captured features are passed through the CapsNet model, which has been custom-trained.

CapsNet labels the picture as Tuberculosis Positive or Normal, taking advantage of its capacity to maintain spatial relationships and high-resolution details which are important for detecting TB

symptoms.

**Result Presentation:**

The last prediction (for example, "TB Positive" or "TB Negative") is presented on the web interface.

A confidence level is also displayed, representing the model's confidence level in its prediction. This assists users in interpreting the output's reliability.

This end-to-end platform—from uploading an image to AI-based diagnosis—proately expedites accurate and interpretable results, improving clinical practices and facilitating radiologists in early TB detection and intervention.

**Classification Pipeline Overview**

The complete classification pipeline is an integration of the frontend, backend, and deep learning models. The steps are:

1. User uploads an image via the frontend.

2. Image is saved and preprocessed in the backend.

3. DenseNet121 extracts meaningful features from the image.

4. Capsule Network classifies the features into TB Positive/Negative.

5. Results are rendered back on a separate results page.

**Flow Diagram:**

**End to End TB Detection Pipeline:**

[Input Image]

↓

[Preprocessing]

↓

[DenseNet121 (Feature Extraction)]

↓

[Capsule Network (Classification)]

↓

[Output: TB Prediction + Confidence]

Figure – 5

**Mathematical Foundations of the Proposed Model**

**DenseNet121 – Feature Extraction Backbone**

DenseNet121 is a convolutional neural network (CNN) based on dense connectivity, where each layer takes inputs from all previous layers. This structure aids in gradient flow and promotes feature reuse.

**Equation 1:** Composite Layer Function

Each layer in **DenseNet** uses a composite function that includes:

**Equation 1: Composite Function (Each Layer)**

Each layer applies a composite function :

Where:

* : input feature map
* : convolution operation
* : activation function
* : batch normalization
* : learnable weights of layer

**Equation 2:** Dense Connectivity Formula

DenseNet connects each layer to every other layer in a feed-forward manner, improving parameter efficiency and feature diversity.

Instead of taking input only from the previous layer, each layer in DenseNet takes **all previous feature maps as input**:

Where:

* : concatenation of feature maps from previous layers

**Custom Capsule Network Classifier**

Following DenseNet121 extracting 1024-dimensional feature vectors, they are fed into a custom fully connected neural network serving as a classifier. Although it does not implement routing-by-agreement, it takes structural inspiration from Capsule Networks (CapsNet) regarding hierarchy and strength.

**Equation 3: Dense Layer Transformation**

Where:

* : input from previous layer (e.g., 1024D vector)
* : weight matrix of layer
* : bias vector
* : output of the layer

This is repeated through each fully connected block:

* 512 neurons
* 128 neurons
* 64 neurons
* 32 neurons

This equation is applied successively through the following layer sizes:

* 512 → 128 → 64 → 32

Each layer helps extract more abstract representations of the input feature vector.

**Equation 4: Dropout Regularization**

During training:

Where:

* : dropout probability (e.g., 0.3)
* : layer output before dropout
* : output after applying dropout to prevent overfitting (randomly deactivating a fraction of neurons)

**Equation 5: Final Sigmoid Output**

Your final output is a single neuron predicting TB or Normal:

Where:

* : sigmoid function, defined as:
* ​: output probability (between 0 and 1)
* : output from last hidden layer
* sigmoid activation to squash output into [0, 1]
* If classified as **Normal**; else **TB Positive**

**Equation 6: Binary Cross-Entropy Loss**

The loss function used for training the binary classifier is Binary Cross-Entropy (BCE):

Used to optimize the model:

Where:

* : true label (0 or 1)
* ​: predicted probability
* The function penalizes incorrect predictions more harshly as confidence increases.

**Evaluation Metric Formulas**

* **Accuracy**:
* **Precision**:
* **Recall**:
* **F1 Score**:
* **AUC (Area Under Curve)**: Numerical integration of TPR vs FPR over different thresholds.

True Positive Rate:

False Positive Rate:

**System Architecture**

The tuberculosis detection system is designed with a modular and layered architecture to ensure scalability, maintainability, and smooth user interaction. The system consists of three primary layers:

**User Interface Layer**

The layer is the application’s front-end and serves to facilitate user interaction. It is the location where healthcare practitioners or users come into contact with the system in the form of uploading chest X-ray images and obtaining diagnostic feedback.

Key Components:

* **HTML Interface:** A well-organized, responsive web page constructed using HTML and CSS to enable users to upload X-ray images via an easy-to-use form interface.
* **JavaScript Improvements:** JavaScript functions are utilized to preview the uploaded images in real time, add visual filters, and reload the interface without page refresh.
* **Real-time Feedback:** The interface contains buttons to preview and upload images, and when submitted, the outcomes (e.g., TB detection result and confidence level) are displayed on a special results page.
* **Accessibility and Usability:** The design is intuitive enough for non-technical users, making it accessible to doctors and medical technicians in actual clinical environments.

**Processing Layer (Flask Backend)**

This is the middleware between the frontend and AI model. Built using the Flask framework, it serves all the logic for processing the uploaded image and serving the prediction.

Responsibilities:

* **File Handling:** Accepts and stores the uploaded images securely on the server for subsequent processing.
* **Image Preprocessing:** Transforms the uploaded image into an appropriately formatted input for the model (e.g., resizing, normalization, and converting to RGB).
* **Model Loading**: Loads the pre-trained DenseNet121 and Capsule Network models into RAM during server startup.
* **Prediction Routing**: Manages the data flow by passing the preprocessed image through the feature extractor and classifier and returning the prediction result to the user interface.
* **Error Handling:** Guarantees that unforeseen inputs or errors (e.g., unsupported file types or corrupted image files) are processed elegantly, with user feedback instead of system crashes.

**Model Inference Layer**

This is the system’s core AI engine that diagnoses tuberculosis from chest X-ray images. It is composed of two deep learning modules that collaborate in a pipeline:

**Components:**

**Feature Extraction (DenseNet121):**

Pre-trained DenseNet121 convolutional neural network is utilized to extract high-level features from input X-ray images.

DenseNet121 offers strong feature maps through the use of dense connections that enable deeper gradient propagation and feature reuse.

**Classification (Capsule Network – CapsNet):**

The feature vectors are fed into a Capsule Network customized to suit our specific needs.

CapsNet is utilized for its property of maintaining spatial hierarchies and being able to model part-whole relations more accurately, which are vital in identifying fine TB patterns from medical images.

The network outputs the image into either of the two classes: TB Positive or TB Negative and also outputs a confidence value with the final prediction.

**Advantages of the Layered Architecture**

**Modularity:** Every layer is responsible for a particular task, which makes it simpler to create, test, and support.

**Debugging Efficiency:** Separating the user interface, server-side processing, and model inference enables effective debugging and troubleshooting.

**Scalability:** The design can be readily expanded in the future—such as adding more models, cloud support, or mobile app integration.

**Separation of Concerns:** The separation of functionality ensures that UI logic is not blended with model logic, adhering to best practices in software engineering.

**Technologies Used**

|  |  |
| --- | --- |
| **Technology** | **Description** |
| TensorFlow | Framework used to build and train the deep learning models. |
| Keras | High-level API for TensorFlow to simplify model design. |
| DenseNet121 | Pretrained CNN model used for feature extraction. |
| Capsule Network | Custom model used for classifying extracted features. |
| Flask | Lightweight Python web framework used to deploy the model. |
| HTML/CSS/JavaScript | Used to build the user interface. |
| NumPy | For array manipulation and numerical operations. |
| OS | For handling file paths and directory creation. |

Table - 1

**Frontend and Backend Interaction**

Frontend and backend interaction in the tuberculosis detection system is used to provide a responsive, seamless, and user-friendly experience while supporting real-time predictions.

**Frontend (index.html):**

The HTML and CSS are used to design the main interface.

It contains an HTML form with input type="file" and name="image" so that users can upload a chest X-ray image.

A JavaScript script manages real-time preview of the selected image prior to submission, thus enhancing user trust and minimizing errors.

A "Submit" button posts the image data to the back-end through an HTTP POST request.

The layout is simple, user-friendly, and responsive, thus accessible across various devices and screen resolutions.

**Backend (Flask - app.py):**

Route / displays the principal index.html form when the app is invoked.

**Route /predict:**

Handles the image from the form

Saves the uploaded image on the server.

Performs preprocessing operations like resizing and normalization.

Utilizes the DenseNet121 model to retrieve deep features of the X-ray image.

Forward passes the features to the Capsule Network, which makes the final prediction.

Creates the prediction (TB Positive or TB Negative) along with the confidence level.

Renders the output using result.html, dynamically inserting the prediction outputs and image.

**Result Page (result.html):**

Displays the uploaded X-ray image for reference.

Displays the final diagnosis result: "TB Positive" or "TB Negative".

Carries a probability score reflecting the confidence level of the model's prediction.

Additional graphical elements (like color-coded boxes or highlights) add to the interpretability of the output.

**Major Features of the Integration:**

Real-time inference provides minimum waiting time for results.

Backend processing is made as fast and reliable as possible by preloading the models.

The system is designed to be modular, facilitating future integration with hospital systems or mobile apps.

Frontend-to-backend communication is smooth, providing a robust pipeline from image upload to prediction output

This end-to-end integration ensures that healthcare professionals or users are able to easily use the platform, making AI-based TB detection accessible and feasible in real-world healthcare settings.

**CHAPTER 4**

**RESULTS AND DISCUSSIONS**

**4.1 Model Performance**

The suggested model recorded outstanding performance on the test data with accuracy, precision, recall, and F1-score at around **98.86%**, reflecting very high reliability in labelling chest X-rays as either tuberculosis or normal. The results prove the superiority of utilizing DenseNet121 in extracting robust features and the handcrafted Capsule Network for accurate classification. Overall, the model reflects considerable promise for application to real-world TB screening, though its accuracy when applied to unseen external data still needs confirmation.

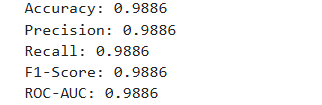


Figure - 6

|  |  |
| --- | --- |
| Metric | Value (%) |
| Accuracy | 98.86 |
| Precision | 98.86 |
| Recall | 98.86 |
| Loss | 98.86 |

Table- 2

**4.2 Training and Validation Accuracy & Loss**

**What It Indicates:**

This plot usually has two subplots:

One for accuracy (training vs validation)

One for loss (training vs validation)

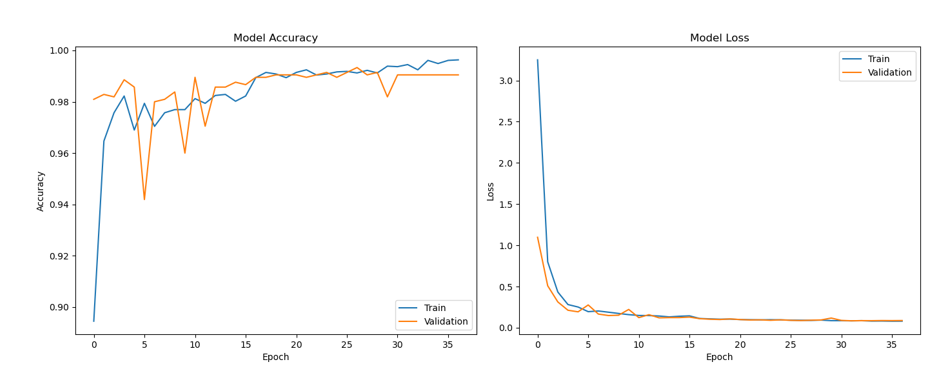


Figure - 7

**Observations:**

The training accuracy goes up steadily throughout training and approaches nearly 100%

The validation accuracy also increases and closely tracks the training accuracy, eventually leveling off at around 99%.

The training loss slowly decreases, which means that the model is learning the patterns in the data.

The validation loss also decreases consistently, indicating that the model is generalizing effectively to unknown data.

**Interpretation:**

There is no overfitting at all: validation and training accuracy/l oss track one another closely.

The model exhibits great convergence, i.e., the training procedure has worked effectively.

Shows good generalization capability on unknown validation data.

**4.3 Confusion Matrix**

**What It Reveals:**

A confusion matrix provides a breakdown of predicted vs actual labels, indicating how accurately the model labeled each class.

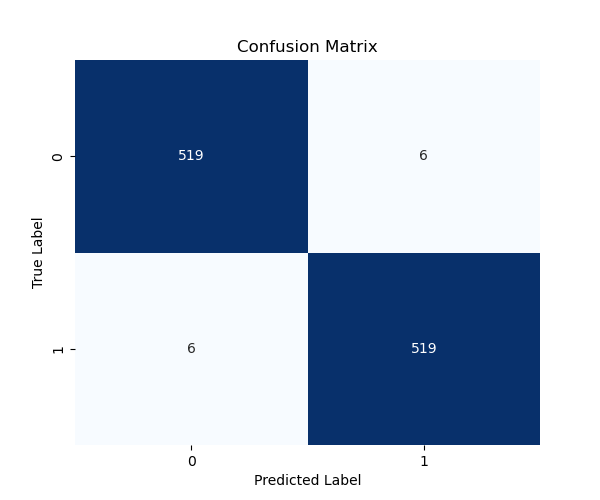


Figure - 8

**Observations:**

Extremely high values along the diagonal (top-left and bottom-right), which are true positives for each class.

Extremely small or zero off-diagonal values (false positives/negatives), which indicates little or no misclassification.

**Interpretation:**

The model is able to properly differentiate between Tuberculosis and Normal cases.

The small off-diagonal entries indicate that the model hardly makes any mistakes.

Verifies the model is extremely accurate in classification.

**4.4 Classification Report**

**What It Contains:**

This report contains precision, recall, F1-score, and support for each class.

**Key Terms Explained:**

* **Precision**: Of the predicted positives, how many were they actually correct?
* **Recall:** Of the actual positives, how many did the model pick up?
* **F1-score:** Harmonic mean of precision and recall. A balanced score.
* **Support:** Number of samples per class.

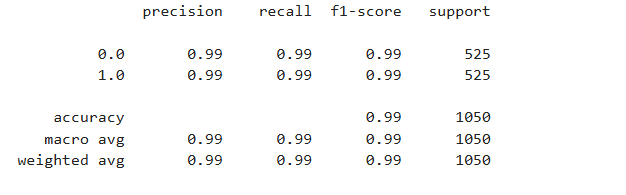


Figure - 9

**Observations:**

Precision = 0.99–1.00 → The model has nearly no false positive predictions.

Recall = 0.99–1.00 → The model captures almost all actual cases (very crucial in medical applications).

F1-score = 0.99 → Good harmony between precision and recall.

Accuracy = 0.99 (99%) → Out of 200 test samples, 198 are correctly classified overall.

**Interpretation:**

* Model performance is perfect on both classes.
* Well suited for critical medical application such as TB detection.
* Reports a balanced and reliable model with no class dominance.

**4.5 ROC Curve**

**What It Represents:**

* Plots True Positive Rate (Recall) versus False Positive Rate.
* Used to assess a model's discrimination power.

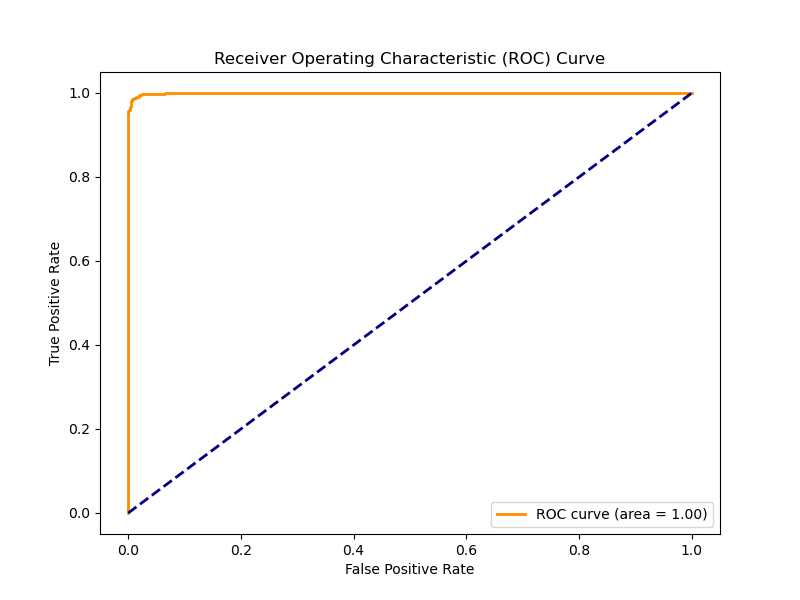


Figure - 10

**Observations:**

The curve clings to the top-left corner, which is great.

The Area Under Curve (AUC) is very close to 1.0, reflecting perfect separation.

**Interpretation:**

The model possesses high capability to discriminate TB from Normal cases.

Low false positives, high true positives.

Verifies excellent classification ability.

**4.6 Precision-Recall Curve**

**What It Reveals:**

Plot of the tradeoff between recall and precision as thresholds are varied.

Most helpful when classes are imbalanced, which occurs in medical imaging.

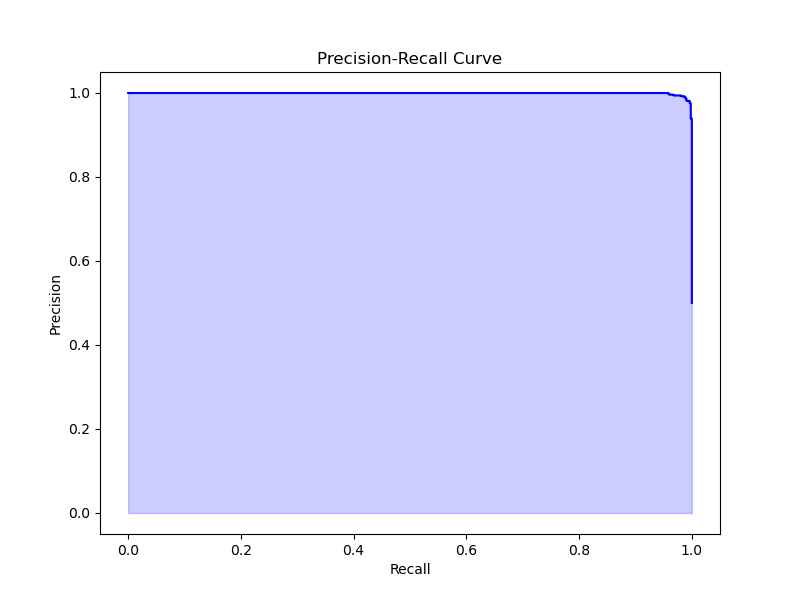


Figure - 11

**Observations:**

The precision is high for a large range of recall values.

The curve begins at high precision and keeps it even when recall rises.

**Explanation:**

The model has great precision even when trying to pick up more positive cases (TB).

This is important to reduce false alarms while still detecting most cases of TB.

Means the model is stable across decision thresholds.

**4.7 Dense Layer Activations**

What It Displays:

Visualization of neuron activations within the dense layer prior to output.

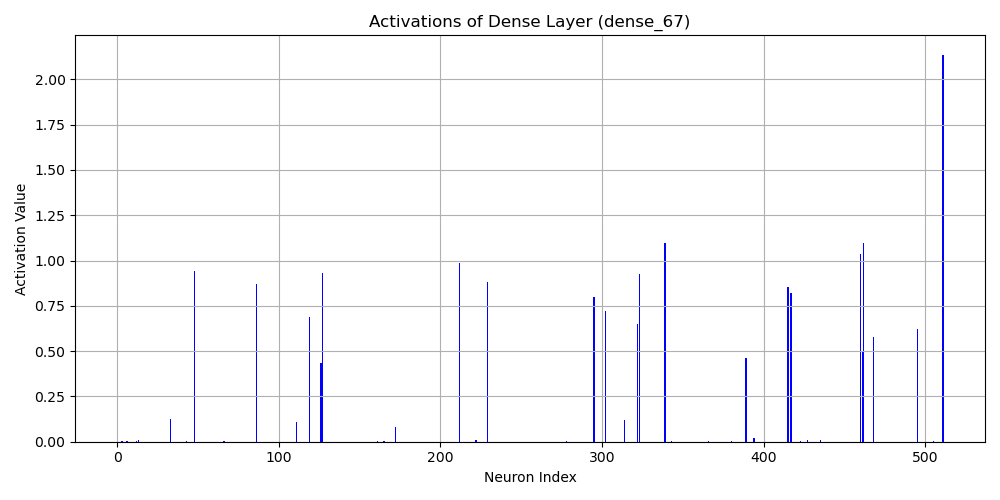


Figure – 12

**Observations:**

The layer displays active and varied feature patterns.

Neurons are evidently responding to varying features in the input.

No "dead" neurons (neurons that don't activate at all).

**Meaning:**

The model is picking up on rich and meaningful features in the input images.

Ensures that the output decision is a result of solid internal representations.

This provides confi

dence and interpretability to the model.

**4.8 MODEL SUMMARY**

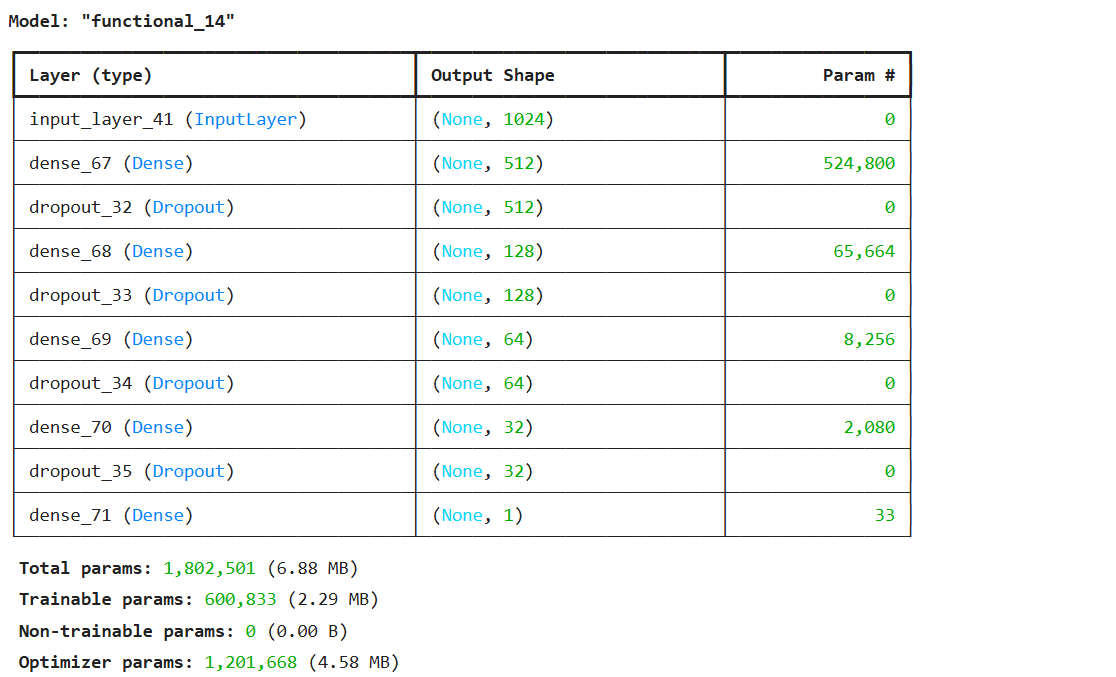


Figure - 13

**Final Summary**

|  |  |
| --- | --- |
| Metric | Result |
| Training Accuracy | ~100% |
| Validation Accuracy | ~99% |
| Precision (Both Classes) | 0.99–1.00 |
| Recall (Both Classes) | 0.99–1.00 |
| F1-score (Both Classes) | 0.99 |
| ROC-AUC | ≈ 1.00 |
| Confusion Matrix | Near-perfect classification |
| Dense Layer Activations | Strong and diverse |

Table –3

**4.9 Website Images:**

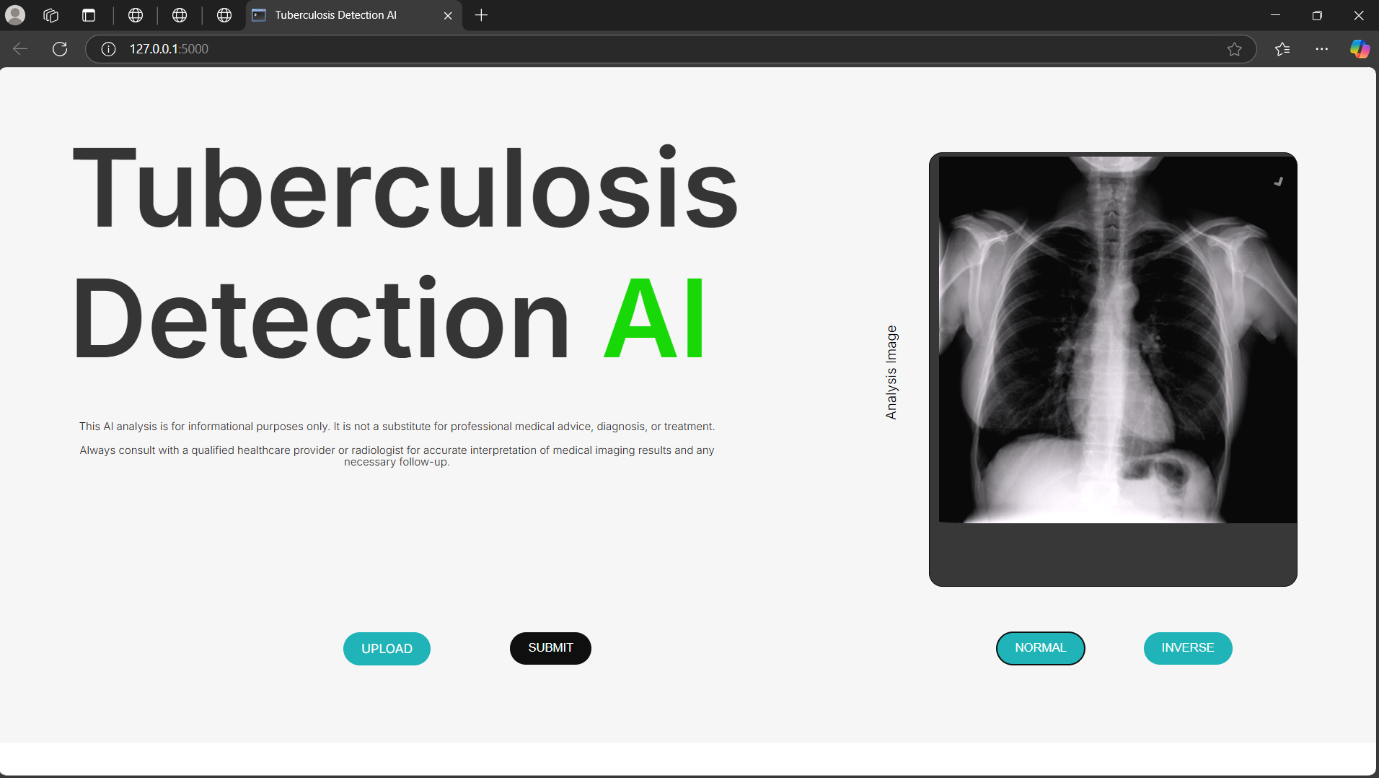


Figure - 14

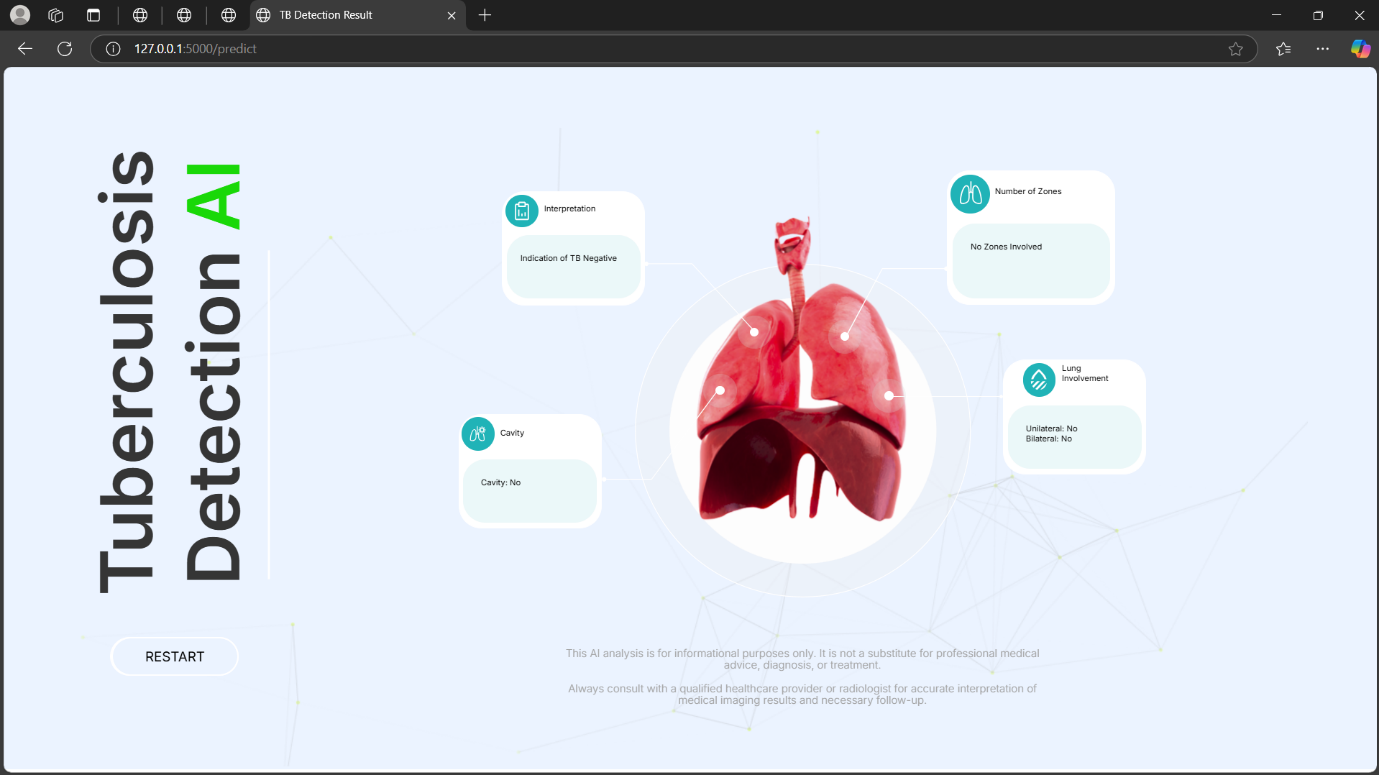


Figure - 15

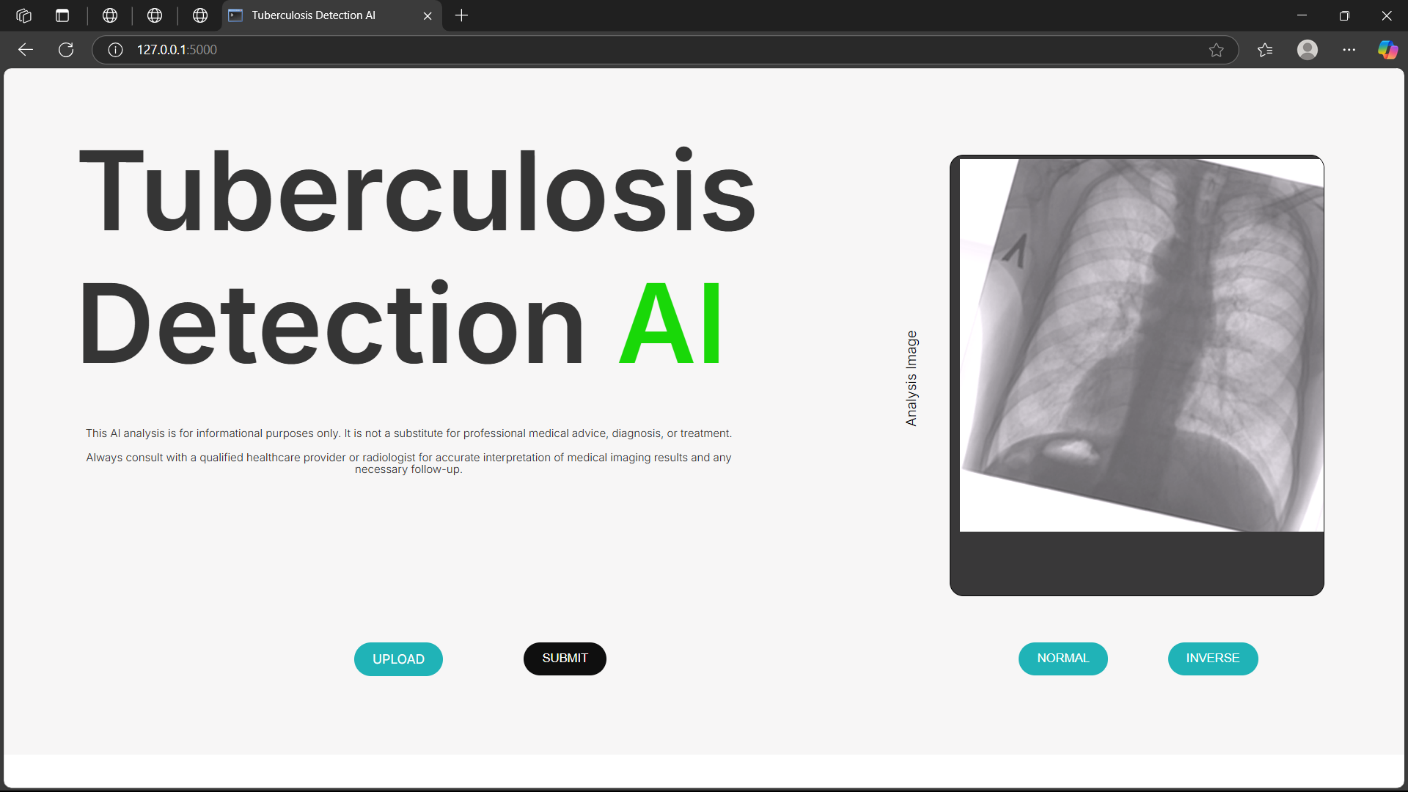


Figure - 16

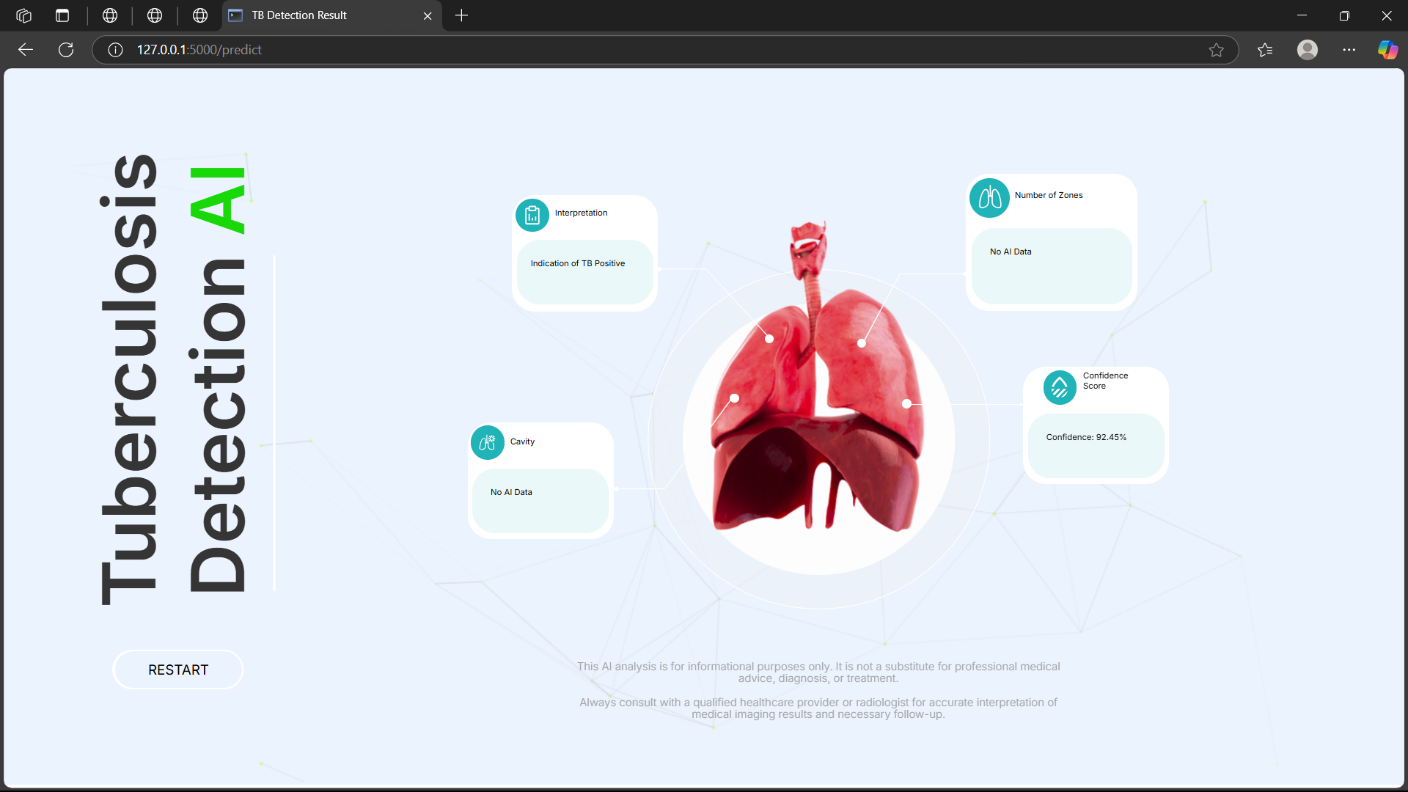


Figure - 17

**Chapter 5**

**CONCLUSION AND FUTURE WORK**

The aim of this project was to develop and deploy an improved neural network system for the automatic diagnosis of tuberculosis based on chest X-ray images. By using the DenseNet121 model to extract deep features and a customized Capsule Network for classification, we built a strong and highly accurate diagnostic pipeline. The model was able to capture spatial hierarchies and high-grained features which are usually pivotal in detecting faint TB signs that might go unnoticed by conventional CNNs. Through structured steps such as dataset gathering, preprocessing, augmentation, feature extraction, model training, and deployment through a Flask-based web application, the project showed an end-to-end AI solution for TB diagnosis. The obtained evaluation metrics — such as high precision, recall, F1-score, and accuracy — confirm the effectiveness of the model. Certain challenges like poor performance on unknown external datasets show the dependence of the model on training data distribution.

In future work, we suggest enlarging the dataset using more heterogeneous and real-world clinical images for enhanced generalization. Incorporating image enhancement methods or domain adaptation methods may assist in dealing with poor-quality X-rays. Integration with Explainable AI (XAI) approaches may provide explainability to garner the trust of medical experts. Additionally, cloud or mobile-based deployment can provide further geographical extensions to the rural and under-resourced populations so that real-time screening becomes a possibility even with limited availability of medical experts. We further contemplate investigating 3D imaging modalities such as CT scans and integrating multi-modal inputs for augmented diagnostic robustness. In general, the project provides a solid foundation for scalable AI-augmented TB screening in medical settings.

**REFERENCES**

1. R. Hooda, S. Sofat, S. Kaur, A. Mittal and F. Meriaudeau, "Deep-learning: A potential method for tuberculosis detection using chest radiography," 2017 IEEE International Conference on Signal and Image Processing Applications (ICSIPA), Kuching, Malaysia, 2017, pp. 497-502, doi: 10.1109/ICSIPA.2017.8120663.

2. Q. H. Nguyen et al., "Deep Learning Models for Tuberculosis Detection from Chest X-ray Images," 2019 26th International Conference on Telecommunications (ICT), Hanoi, Vietnam, 2019, pp. 381-385, doi: 10.1109/ICT.2019.8798798.

3. V. T. Q. Huy and C. -M. Lin, "An Improved Densenet Deep Neural Network Model for Tuberculosis Detection Using Chest X-Ray Images," in IEEE Access, vol. 11, pp. 42839-42849, 2023, doi: 10.1109/ACCESS.2023.3270774.

4. C. Liu et al., "TX-CNN: Detecting tuberculosis in chest X-ray images using convolutional neural network," 2017 IEEE International Conference on Image Processing (ICIP), Beijing, China, 2017, pp. 2314-2318, doi: 10.1109/ICIP.2017.8296695.

5. Lopes, U.K., and J.F. Valiati. “Pre-Trained Convolutional Neural Networks as Feature Extractors for Tuberculosis Detection.” Computers in Biology and Medicine, vol. 89, Oct. 2017, pp. 135–143, <https://doi.org/10.1016/j.compbiomed.2017.08.001>.

6. Ahsan, Mostofa, et al. “Application of a Convolutional Neural Network Using Transfer Learning for Tuberculosis Detection.” 2019 IEEE International Conference on Electro Information Technology (EIT), May 2019, <https://doi.org/10.1109/eit.2019.8833768>.

Accessed 18 Dec. 2022.

7. Burcu Oltu, et al. Automated Tuberculosis Detection Using Pre-Trained CNN and SVM. 26 July 2021, pp. 92–95, ieeexplore.ieee.org/document/9522644, <https://doi.org/10.1109/tsp52935.2021.9522644>.

Accessed 19 May 2025.

8. Gabriella, Ilena, et al. “Early Detection of Tuberculosis Using Chest X-Ray (CXR) with Computer-Aided Diagnosis.” 2018 2nd International Conference on Biomedical Engineering (IBIOMED), July 2018, <https://doi.org/10.1109/ibiomed.2018.8534784>.

Accessed 2 Jan. 2023.

9. Urooj, Shabana, et al. “Stochastic Learning-Based Artificial Neural Network Model for an Automatic Tuberculosis Detection System Using Chest X-Ray Images.” IEEE Access, vol. 10, 2022, pp. 103632–103643, https://doi.org/10.1109/access.2022.3208882. Accessed 21 Dec. 2022.

10. Rahman, Tawsifur, et al. “TB-CXRNet: Tuberculosis and Drug-Resistant Tuberculosis Detection Technique Using Chest X-Ray Images.” Cognitive Computation, vol. 16, no. 3, 17 Feb. 2024, pp. 1393–1412, https://doi.org/10.1007/s12559-024-10259-3. Accessed 5 Nov. 2024.

11. “Computer Methods and Programs in Biomedicine Update | Journal | ScienceDirect.com by Elsevier.” Www.sciencedirect.com, [www.sciencedirect.com/journal/computer-methods-and-programs-in-biomedicine-update](http://www.sciencedirect.com/journal/computer-methods-and-programs-in-biomedicine-update).

12. Nafisah, Saad I., and Ghulam Muhammad. “Tuberculosis Detection in Chest Radiograph Using Convolutional Neural Network Architecture and Explainable Artificial Intelligence.” Neural Computing and Applications, 19 Apr. 2022, https://doi.org/10.1007/s00521-022-07258-6. Accessed 4 May 2022.

13. Munadi, Khairul, et al. “Image Enhancement for Tuberculosis Detection Using Deep Learning.” IEEE Access, vol. 8, 2020, pp. 217897–217907, https://doi.org/10.1109/access.2020.3041867. Accessed 11 May 2022.

14.“IEEE Xplore Full-Text PDF”: Ieee.org, 2025, ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9224622. Accessed 19 May 2025.

15. Ayaz, Muhammad, et al. “Ensemble Learning Based Automatic Detection of Tuberculosis in Chest X-Ray Images Using Hybrid Feature Descriptors.” Physical and Engineering Sciences in Medicine, vol. 44, no. 1, 18 Jan. 2021, pp. 183–194, https://doi.org/10.1007/s13246-020-00966-0. Accessed 2 Nov. 2021.

16. A. Wong, J. R. H. Lee, H. Rahmat-Khah, A. Sabri, and A. Alaref, “TB-Net: A Tailored, Self-Attention Deep Convolutional Neural Network Design for Detection of Tuberculosis Cases from Chest X-ray Images,” arXiv preprint, arXiv:2104.03165, Apr. 2021.

17. A. Singh et al., “Deep Learning for Automated Screening of Tuberculosis from Indian Chest X-rays: Analysis and Update,” arXiv preprint, arXiv:2011.09778, Nov. 2020.

18. R. S. Ghorakavi, “TBNet: Pulmonary Tuberculosis Diagnosing System using Deep Neural Networks,” arXiv preprint, arXiv:1902.08897, Feb. 2019.

19. M. A. Khan et al., “TB-DenseNet: A CNN-Based Hybrid Framework for Tuberculosis Detection,” Biomedical Signal Processing and Control, vol. 78, p. 103961, Jan. 2023.

20. M. S. Islam et al., “Explainable Deep-Neural-Network Supported Scheme for Tuberculosis Detection,” BMC Medical Imaging, vol. 24, no. 1, p. 12, Jan. 2024.

**APPENDIX**

*#Load the dataset*

import os

from tensorflow.keras.preprocessing.image import ImageDataGenerator

*# Define paths*

data\_dir = r"E:\SDP\TB\_Chest\_Radiography\_Database"

normal\_dir = os.path.join(data\_dir, "Normal")

tb\_dir = os.path.join(data\_dir, "Tuberculosis")

*#Data analysis*

import os

import matplotlib.pyplot as plt

*# Define paths*

data\_dir = r"E:\SDP\TB\_Chest\_Radiography\_Database"

normal\_dir = os.path.join(data\_dir, "Normal")

tb\_dir = os.path.join(data\_dir, "Tuberculosis")

*# Count the number of images in each class*

normal\_count = len(os.listdir(normal\_dir))

tb\_count = len(os.listdir(tb\_dir))

*# Print the counts*

print(f"Number of Normal images: {normal\_count}")

print(f"Number of Tuberculosis images: {tb\_count}")

*# Plot the class distribution*

classes = ['Normal', 'Tuberculosis']

counts = [normal\_count, tb\_count]

plt.figure(figsize=(8, 6))

plt.bar(classes, counts, color=['blue', 'orange'])

plt.title('Class Distribution', fontsize=16)

plt.ylabel('Number of Images', fontsize=14)

plt.xlabel('Class', fontsize=14)

plt.xticks(fontsize=12)

plt.yticks(fontsize=12)

for i, count in enumerate(counts):

    plt.text(i, count + 50, str(count), ha='center', fontsize=12)  *# Add count labels on top of bars*

plt.tight\_layout()

plt.show()

plt.close()

*#Visualize sample Images*

import os

import matplotlib.pyplot as plt

from tensorflow.keras.preprocessing.image import load\_img

import numpy as np

*# Define paths*

data\_dir = r"E:\SDP\TB\_Chest\_Radiography\_Database"

normal\_dir = os.path.join(data\_dir, "Normal")

tb\_dir = os.path.join(data\_dir, "Tuberculosis")

*# Function to plot sample images*

def plot\_sample\_images(class\_dir, class\_name, num\_samples=6):

    """

    Visualize random sample images from a specific class.

    :param class\_dir: Path to the directory containing images of a specific class

    :param class\_name: Name of the class (e.g., 'Normal' or 'Tuberculosis')

    :param num\_samples: Number of random samples to display

    """

*# Get a list of all image filenames in the directory*

    image\_files = os.listdir(class\_dir)

*# Create a figure for visualization*

    plt.figure(figsize=(15, 7))

    for i in range(num\_samples):

*# Randomly select an image*

        img\_file = np.random.choice(image\_files)

        img\_path = os.path.join(class\_dir, img\_file)

*# Load and display the image*

        img = load\_img(img\_path, color\_mode='grayscale')  *# Load as grayscale*

        plt.subplot(2, num\_samples // 2, i + 1)

        plt.imshow(img, cmap='gray')

        plt.title(f"Class: {class\_name}", fontsize=14)

        plt.axis('off')  *# Turn off axis*

    plt.suptitle(f"Sample Images from {class\_name} Class", fontsize=18)

    plt.tight\_layout(rect=[0, 0, 1, 0.96])  *# Adjust layout to fit title*

    plt.show()

*# Visualize sample images from the Normal class*

print("Visualizing sample images from the Normal class:")

plot\_sample\_images(normal\_dir, "Normal")

*# Visualize sample images from the Tuberculosis class*

print("Visualizing sample images from the Tuberculosis class:")

plot\_sample\_images(tb\_dir, "Tuberculosis")

*#Data Augmentation*

import os

import numpy as np

from tensorflow.keras.preprocessing.image import ImageDataGenerator, load\_img, img\_to\_array, save\_img

*# Define paths*

data\_dir = r"E:\SDP\TB\_Chest\_Radiography\_Database"

tb\_dir = os.path.join(data\_dir, "Tuberculosis")

augmented\_tb\_dir = os.path.join(data\_dir, "Augmented\_Tuberculosis")

*# Create output directory for augmented Tuberculosis images*

os.makedirs(augmented\_tb\_dir, exist\_ok=True)

*# Data augmentation configuration*

datagen = ImageDataGenerator(

    rotation\_range=15,       *# Randomly rotate images by up to 15 degrees*

    width\_shift\_range=0.1,   *# Randomly shift images horizontally by 10%*

    height\_shift\_range=0.1,  *# Randomly shift images vertically by 10%*

    shear\_range=0.1,         *# Shear transformation*

    zoom\_range=0.1,          *# Randomly zoom in on images by 10%*

    horizontal\_flip=True,    *# Randomly flip images horizontally*

    fill\_mode='nearest'      *# Fill missing pixels after transformations*

)

*# Function to augment Tuberculosis images*

def augment\_tuberculosis\_images(input\_dir, output\_dir, target\_count=3500):

*# Load all Tuberculosis images*

    tb\_images = [img\_file for img\_file in os.listdir(input\_dir)]

    original\_count = len(tb\_images)

    print(f"Original Tuberculosis image count: {original\_count}")

*# Calculate how many augmentations are needed*

    augmentations\_per\_image = int(np.ceil((target\_count - original\_count) / original\_count))

    print(f"Augmentations per image required: {augmentations\_per\_image}")

*# Generate augmented images*

    augmented\_count = 0

    for img\_file in tb\_images:

        img\_path = os.path.join(input\_dir, img\_file)

        try:

*# Load image without resizing*

            img = load\_img(img\_path, color\_mode='grayscale')

            img\_array = img\_to\_array(img)  *# Convert to numpy array (shape: H x W x 1)*

*# Add batch dimension for augmentation*

            img\_array = np.expand\_dims(img\_array, axis=0)  *# Shape: (1, H, W, 1)*

*# Generate augmented images*

            for i in range(augmentations\_per\_image):

                if augmented\_count >= target\_count - original\_count:

                    break  *# Stop if target count is reached*

*# Apply augmentation*

                augmented\_img = datagen.flow(img\_array, batch\_size=1)[0]  *# Shape: (H, W, 1)*

*# Remove batch dimension if necessary*

                if len(augmented\_img.shape) == 4:  *# Check if batch dimension exists*

                    augmented\_img = augmented\_img[0]  *# Shape: (H, W, 1)*

*# Save augmented image*

                base\_name, ext = os.path.splitext(img\_file)

                output\_file = f"{base\_name}\_aug{i}{ext}"

                output\_path = os.path.join(output\_dir, output\_file)

                save\_img(output\_path, augmented\_img, scale=False, color\_mode='grayscale')

                augmented\_count += 1

        except Exception as e:

            print(f"Error processing {img\_path}: {e}")

*# Copy original Tuberculosis images to the augmented directory*

    for img\_file in tb\_images:

        src\_path = os.path.join(input\_dir, img\_file)

        dst\_path = os.path.join(output\_dir, img\_file)

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

            os.link(src\_path, dst\_path)  *# Create a hard link to avoid duplicating data*

    print(f"Total augmented Tuberculosis images: {len(os.listdir(output\_dir))}")

*# Augment Tuberculosis images*

print("Augmenting Tuberculosis images...")

augment\_tuberculosis\_images(tb\_dir, augmented\_tb\_dir, target\_count=3500)

*#splitting dataset into train, validation and test.*

import os

import numpy as np

from sklearn.model\_selection import train\_test\_split

import shutil

*# Define paths*

data\_dir = r"E:\SDP\TB\_Chest\_Radiography\_Database"

normal\_dir = os.path.join(data\_dir, "Normal")

tb\_dir = os.path.join(data\_dir, "Augmented\_Tuberculosis")

*# Define the output directory for split data*

split\_data\_dir = r"E:\SDP\Split data1"

os.makedirs(split\_data\_dir, exist\_ok=True)

*# Create subdirectories for training, validation, and test sets*

train\_dir = os.path.join(split\_data\_dir, "train1")

val\_dir = os.path.join(split\_data\_dir, "val1")

test\_dir = os.path.join(split\_data\_dir, "test1")

for directory in [train\_dir, val\_dir, test\_dir]:

    os.makedirs(os.path.join(directory, "Normal"), exist\_ok=True)

    os.makedirs(os.path.join(directory, "Augmented\_Tuberculosis"), exist\_ok=True)

*# Load filenames and labels*

normal\_images = [os.path.join(normal\_dir, img) for img in os.listdir(normal\_dir)]

tb\_images = [os.path.join(tb\_dir, img) for img in os.listdir(tb\_dir)]

*# Create labels: 0 for Normal, 1 for Tuberculosis*

normal\_labels = [0] \* len(normal\_images)

tb\_labels = [1] \* len(tb\_images)

*# Combine all images and labels*

all\_images = normal\_images + tb\_images

all\_labels = normal\_labels + tb\_labels

*# Convert to numpy arrays for easier handling*

all\_images = np.array(all\_images)

all\_labels = np.array(all\_labels)

*# Split into training (70%), validation (15%), and test (15%) sets*

X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(

    all\_images, all\_labels, test\_size=0.3, random\_state=42, stratify=all\_labels

)

X\_val, X\_test, y\_val, y\_test = train\_test\_split(

    X\_temp, y\_temp, test\_size=0.5, random\_state=42, stratify=y\_temp

)

*# Print the sizes of each set*

print(f"Training set size: {len(X\_train)} ({len(y\_train)} labels)")

print(f"Validation set size: {len(X\_val)} ({len(y\_val)} labels)")

print(f"Test set size: {len(X\_test)} ({len(y\_test)} labels)")

*# Optional: Verify class distribution in each set*

def print\_class\_distribution(labels, set\_name):

    unique, counts = np.unique(labels, return\_counts=True)

    print(f"{set\_name} Class Distribution: {dict(zip(unique, counts))}")

print\_class\_distribution(y\_train, "Training Set")

print\_class\_distribution(y\_val, "Validation Set")

print\_class\_distribution(y\_test, "Test Set")

*# Function to copy files to the destination directory*

def copy\_files(file\_paths, labels, dest\_dir):

    for file\_path, label in zip(file\_paths, labels):

        class\_name = "Normal" if label == 0 else "Augmented\_Tuberculosis"

        dest\_path = os.path.join(dest\_dir, class\_name, os.path.basename(file\_path))

*# Ensure the destination directory exists*

        os.makedirs(os.path.dirname(dest\_path), exist\_ok=True)

*# Copy the file*

        shutil.copy(file\_path, dest\_path)

*# Copy files to their respective directories*

print("Copying training set...")

copy\_files(X\_train, y\_train, train\_dir)

print("Copying validation set...")

copy\_files(X\_val, y\_val, val\_dir)

print("Copying test set...")

copy\_files(X\_test, y\_test, test\_dir)

print("Dataset splitting and saving completed successfully!")

*#Extracting features using DenseNet121.*

import os

import numpy as np

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import DenseNet121

from tqdm import tqdm

*# ------------------------------*

*# STEP 1: DATA PREPARATION*

*# ------------------------------*

*# Define paths*

split\_data\_dir = r"E:\SDP\Split data1"

train\_dir = os.path.join(split\_data\_dir, "train1")

val\_dir = os.path.join(split\_data\_dir, "val1")

test\_dir = os.path.join(split\_data\_dir, "test1")

*# Output directory for extracted features*

features\_dir = r"E:\SDP\Extracted\_Features"

os.makedirs(features\_dir, exist\_ok=True)

*# Data augmentation and preprocessing*

datagen = ImageDataGenerator(rescale=1.0 / 255)

*# Generators for training, validation, and testing*

def create\_generator(directory):

    return datagen.flow\_from\_directory(

        directory,

        target\_size=(224, 224),  *# DenseNet121 input size*

        batch\_size=32,

        class\_mode="binary",

        color\_mode="rgb",

        shuffle=False,  *# Keep data order for feature extraction*

    )

train\_generator = create\_generator(train\_dir)

val\_generator = create\_generator(val\_dir)

test\_generator = create\_generator(test\_dir)

*# ------------------------------*

*# STEP 2: LOAD DENSENET121 FOR FEATURE EXTRACTION*

*# ------------------------------*

*# Load DenseNet121 with pretrained weights*

base\_model = DenseNet121(weights="imagenet", include\_top=False, pooling="avg")

print("DenseNet121 loaded successfully.")

*# Function to extract features*

def extract\_features(generator, model):

    features = []

    labels = []

    num\_batches = len(generator)  *# Number of batches in the generator*

    for i in tqdm(range(num\_batches), desc=f"Extracting features from {generator.directory}"):

        batch\_images, batch\_labels = generator[i]  *# Get the next batch*

        batch\_features = model.predict(batch\_images, verbose=0)  *# Predict features*

        features.append(batch\_features)

        labels.append(batch\_labels)

    return np.vstack(features), np.concatenate(labels)

*# Extract features for train, validation, and test sets*

X\_train\_caps, y\_train\_caps = extract\_features(train\_generator, base\_model)

X\_val\_caps, y\_val\_caps = extract\_features(val\_generator, base\_model)

X\_test\_caps, y\_test\_caps = extract\_features(test\_generator, base\_model)

*# Save extracted features*

np.save(os.path.join(features\_dir, "X\_train\_caps.npy"), X\_train\_caps)

np.save(os.path.join(features\_dir, "y\_train\_caps.npy"), y\_train\_caps)

np.save(os.path.join(features\_dir, "X\_val\_caps.npy"), X\_val\_caps)

np.save(os.path.join(features\_dir, "y\_val\_caps.npy"), y\_val\_caps)

np.save(os.path.join(features\_dir, "X\_test\_caps.npy"), X\_test\_caps)

np.save(os.path.join(features\_dir, "y\_test\_caps.npy"), y\_test\_caps)

print("Feature extraction completed successfully!")

*#Loading data required and training two classifiers.*

import numpy as np

import os

*# Define the directory where features are saved*

features\_dir = r"E:\SDP\Extracted\_Features"

*# Load training features and labels*

X\_train\_caps = np.load(os.path.join(features\_dir, "X\_train\_caps.npy"))

y\_train\_caps = np.load(os.path.join(features\_dir, "y\_train\_caps.npy"))

*# Load validation features and labels*

X\_val\_caps = np.load(os.path.join(features\_dir, "X\_val\_caps.npy"))

y\_val\_caps = np.load(os.path.join(features\_dir, "y\_val\_caps.npy"))

*# Load test features and labels*

X\_test\_caps = np.load(os.path.join(features\_dir, "X\_test\_caps.npy"))

y\_test\_caps = np.load(os.path.join(features\_dir, "y\_test\_caps.npy"))

*# Convert labels to categorical (one-hot encoding) for CapsNet*

from tensorflow.keras.utils import to\_categorical

y\_train\_caps\_cat = to\_categorical(y\_train\_caps, num\_classes=2)

y\_val\_caps\_cat = to\_categorical(y\_val\_caps, num\_classes=2)

y\_test\_caps\_cat = to\_categorical(y\_test\_caps, num\_classes=2)

print("Features and labels loaded successfully!")

*#Capsnet model*

import numpy as np

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, Dense, Dropout

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

from tensorflow.keras import regularizers

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score

*# ------------------------------*

*# FIX: Convert labels back to binary scalars*

*# ------------------------------*

*# Load pre-extracted feature vectors and labels*

X\_train\_caps = np.load(r"E:\SDP\Extracted\_Features\X\_train\_caps.npy")

y\_train\_caps = np.load(r"E:\SDP\Extracted\_Features\y\_train\_caps.npy")

X\_val\_caps = np.load(r"E:\SDP\Extracted\_Features\X\_val\_caps.npy")

y\_val\_caps = np.load(r"E:\SDP\Extracted\_Features\y\_val\_caps.npy")

X\_test\_caps = np.load(r"E:\SDP\Extracted\_Features\X\_test\_caps.npy")

y\_test\_caps = np.load(r"E:\SDP\Extracted\_Features\y\_test\_caps.npy")

*# Use the binary labels directly (not one-hot encoded)*

*# Make sure the target labels are 0 or 1*

*# No need for `to\_categorical` here as we are using binary scalar values for labels*

*# ------------------------------*

*# FIX: Modify the CapsNet for Binary Classification with Scalar Labels*

*# ------------------------------*

from tensorflow.keras.layers import Dropout  *# Import Dropout*

def build\_capsnet\_feature\_based(input\_dim):

    """

    Build the CapsNet model accepting feature vectors as input.

    Adding L2 regularization and dropout for improved generalization.

    """

*# Input layer for the feature vectors*

    inputs = Input(shape=(input\_dim,))  *# Input should match the extracted feature size (e.g., 1024)*

*# Fully connected layers with L2 regularization and dropout*

    x = Dense(512, activation="relu", kernel\_regularizer=regularizers.l2(0.01))(inputs)

    x = Dropout(0.3)(x)  *# Dropout to prevent overfitting*

    x = Dense(128, activation="relu", kernel\_regularizer=regularizers.l2(0.01))(x)

    x = Dropout(0.3)(x)

    x = Dense(64, activation="relu", kernel\_regularizer=regularizers.l2(0.01))(x)

    x = Dropout(0.3)(x)

    x = Dense(32, activation="relu", kernel\_regularizer=regularizers.l2(0.01))(x)

    x = Dropout(0.3)(x)

*# Output layer with a single unit and sigmoid activation for binary classification*

    outputs = Dense(1, activation="sigmoid")(x)

*# Compile the model with a lower learning rate*

    model = Model(inputs, outputs)

*# Use binary\_crossentropy loss for binary classification (with sigmoid output)*

    model.compile(optimizer=Adam(learning\_rate=1e-3), loss="binary\_crossentropy", metrics=["accuracy", "Precision", "Recall", "AUC"])

    return model

*# ------------------------------*

*# STEP: TRAIN THE MODEL USING EXTRACTED FEATURES*

*# ------------------------------*

*# Build the model using the number of features in the input vector*

input\_dim = X\_train\_caps.shape[1]  *# Example: 1024 features from DenseNet*

capsnet\_model = build\_capsnet\_feature\_based(input\_dim)

*# Early stopping and learning rate reduction callbacks*

early\_stopping = EarlyStopping(monitor="val\_loss", patience=5, restore\_best\_weights=True)

reduce\_lr = ReduceLROnPlateau(monitor="val\_loss", factor=0.2, patience=3, min\_lr=1e-6)

*# Fit the model*

history\_capsnet = capsnet\_model.fit(

    X\_train\_caps,

    y\_train\_caps,  *# Binary labels (0 or 1), no need for one-hot encoding*

    epochs=60,

    validation\_data=(X\_val\_caps, y\_val\_caps),  *# Binary labels (0 or 1), no need for one-hot encoding*

    callbacks=[early\_stopping, reduce\_lr]

)

*# ------------------------------*

*# STEP: EVALUATE THE MODEL PERFORMANCE*

*# ------------------------------*

*# Get predictions from the model*

y\_test\_pred = (capsnet\_model.predict(X\_test\_caps) > 0.5).astype(int)

*# Evaluate the model's performance using common metrics*

def evaluate(y\_true, y\_pred):

    print(f"Accuracy: {accuracy\_score(y\_true, y\_pred):.4f}")

    print(f"Precision: {precision\_score(y\_true, y\_pred):.4f}")

    print(f"Recall: {recall\_score(y\_true, y\_pred):.4f}")

    print(f"F1-Score: {f1\_score(y\_true, y\_pred):.4f}")

    print(f"ROC-AUC: {roc\_auc\_score(y\_true, y\_pred):.4f}")

evaluate(y\_test\_caps, y\_test\_pred)

*# Save the trained model*

capsnet\_model.save(r"E:\SDP\model\capsnet\_feature\_model.h5")

print("Model saved successfully!")

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

import pickle

from sklearn.metrics import confusion\_matrix, precision\_recall\_curve, classification\_report

import tensorflow as tf

from sklearn.metrics import roc\_curve, auc

*# Assuming 'history\_capsnet' is the training history object*

*# 1. \*\*Training vs Validation Accuracy & Loss (Side-by-side)\*\**

plt.figure(figsize=(12,5))

*# Accuracy Plot*

plt.subplot(1,2,1)

plt.plot(history\_capsnet.history['accuracy'])

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

plt.title('Model Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='lower right')

*# Loss Plot*

plt.subplot(1,2,2)

plt.plot(history\_capsnet.history['loss'])

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

plt.title('Model Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

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

*# Save and display training vs validation accuracy and loss*

accuracy\_loss\_plot\_path = r"E:\SDP\model\training\_validation\_accuracy\_loss1.png"

plt.savefig(accuracy\_loss\_plot\_path)

plt.show()  *# Display the plot*

plt.close()

import seaborn as sns

from sklearn.metrics import confusion\_matrix

import matplotlib.pyplot as plt

*# Get predictions from the model*

y\_pred = (capsnet\_model.predict(X\_test\_caps) > 0.5).astype(int)  *# Get binary predictions*

*# Compute the confusion matrix*

cm = confusion\_matrix(y\_test\_caps, y\_pred)

*# Plot the confusion matrix*

plt.figure(figsize=(6,5))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.title('Confusion Matrix')

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

*# Save the confusion matrix plot*

plt.savefig(r"E:\SDP\model\confusion\_matrix.png")

plt.show()

plt.close()

from sklearn.metrics import precision\_recall\_curve

import matplotlib.pyplot as plt

*# 3. \*\*Precision-Recall Curve\*\**

*# Get the true labels and predicted probabilities*

y\_true = y\_test\_caps  *# Actual labels (from test data)*

y\_pred = capsnet\_model.predict(X\_test\_caps)  *# Predicted probabilities from the model*

*# For binary classification, we use the probabilities of the positive class (1)*

precision, recall, \_ = precision\_recall\_curve(y\_true, y\_pred)

*# Plot the Precision-Recall curve*

plt.figure(figsize=(8, 6))

plt.plot(recall, precision, color='blue')

plt.fill\_between(recall, precision, color='blue', alpha=0.2)

plt.title('Precision-Recall Curve')

plt.xlabel('Recall')

plt.ylabel('Precision')

*# Save the precision-recall curve*

precision\_recall\_curve\_path = r"E:\SDP\model\precision\_recall\_curve.png"

plt.savefig(precision\_recall\_curve\_path)

plt.show()

plt.close()

from sklearn.metrics import classification\_report

import numpy as np

*# 4. \*\*Classification Report (Save as .txt)\*\**

*# Get the predicted classes (binary 0 or 1)*

y\_pred\_classes = (y\_pred > 0.5).astype(int)  *# Convert probabilities to binary predictions*

*# Generate the classification report*

report = classification\_report(y\_true, y\_pred\_classes)

*# Save the classification report to a text file*

classification\_report\_path = r"E:\SDP\model\classification\_report.txt"

with open(classification\_report\_path, "w") as f:

    f.write(report)

*# Optionally, print the report for visualization*

print(report)

from sklearn.metrics import roc\_curve, auc

import matplotlib.pyplot as plt

*# 5. \*\*ROC Curve\*\**

*# Compute False Positive Rate (FPR) and True Positive Rate (TPR)*

fpr, tpr, \_ = roc\_curve(y\_true, y\_pred)  *# Use y\_pred directly since it's a single column of probabilities*

*# Calculate AUC (Area Under the Curve)*

roc\_auc = auc(fpr, tpr)

*# Plot the ROC curve*

plt.figure(figsize=(8, 6))

plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.title('Receiver Operating Characteristic (ROC) Curve')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.legend(loc="lower right")

*# Save the ROC curve plot*

roc\_curve\_path = r"E:\SDP\model\roc\_curve.png"

plt.savefig(roc\_curve\_path)

plt.show()

plt.close()

*# 6. \*\*Model Architecture Summary (Save as .txt)\*\**

*# Open the file in write mode to save the model summary with UTF-8 encoding*

with open(r"E:\SDP\model\capsnet\_model\_summary.txt", "w", encoding="utf-8") as file:

*# Save the model summary to the file*

    capsnet\_model.summary(print\_fn=lambda x: file.write(x + "\n"))

*# Optionally, print the model summary to the console as well*

capsnet\_model.summary()

import pickle

*# 8. \*\*Save Training History (Raw Logs)\*\**

history\_save\_path = r"E:\SDP\model\training\_history.pkl"

*# Save the training history to a pickle file*

with open(history\_save\_path, "wb") as file\_pi:

    pickle.dump(history\_capsnet.history, file\_pi)

print("Training history saved successfully!")

*# Load the training history from the pickle file*

with open(history\_save\_path, "rb") as file\_pi:

    loaded\_history = pickle.load(file\_pi)

*# Display the loaded history to confirm*

print(loaded\_history)

*#Activation of dense layers visualization*

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.models import Model

*# Select the first Dense layer for activation visualization (using 'dense\_67' as the layer name)*

intermediate\_layer\_model = Model(inputs=capsnet\_model.input,

                                 outputs=capsnet\_model.get\_layer('dense\_67').output)  *# 'dense\_67' is the correct layer name*

*# Get the activations for a sample image (using X\_test\_caps)*

sample\_image = X\_test\_caps[0].reshape(1, -1)  *# Reshaping to match the input shape*

activations = intermediate\_layer\_model.predict(sample\_image)

*# Visualize the activations*

plt.figure(figsize=(10, 5))

plt.bar(range(activations.shape[1]), activations[0], color='blue')  *# Plotting activations of all neurons in 'dense\_67'*

plt.title('Activations of Dense Layer (dense\_67)')

plt.xlabel('Neuron Index')

plt.ylabel('Activation Value')

plt.grid(True)

*# Save the activation visualization*

plt.tight\_layout()

plt.savefig(r"E:\SDP\model\dense\_67\_activations.png")

plt.show()

plt.close()

*#Testing the model on data*

import numpy as np

import os

from tensorflow.keras.preprocessing import image

from tensorflow.keras.applications.densenet import DenseNet121, preprocess\_input

from tensorflow.keras.models import load\_model

*# Load models*

capsnet\_model = load\_model(r"E:\SDP\model\capsnet\_feature\_model.h5")

densenet\_model = DenseNet121(weights="imagenet", include\_top=False, pooling="avg")

*# Function to preprocess image*

def preprocess\_image(img\_path):

    img = image.load\_img(img\_path, target\_size=(224, 224))

    img\_array = image.img\_to\_array(img)

    img\_array = np.expand\_dims(img\_array, axis=0)

    img\_array = preprocess\_input(img\_array)

    return img\_array

*# Function to extract features*

def extract\_features\_from\_image(img\_array):

    features = densenet\_model.predict(img\_array)

    return features

*# Function to predict*

def predict\_image(img\_path):

    img\_array = preprocess\_image(img\_path)

    features = extract\_features\_from\_image(img\_array)

    prediction = capsnet\_model.predict(features)

    probability = prediction[0][0]  *# Get the number from (1,1) array*

    threshold = 0.7

    if probability > threshold:

        predicted\_class = 0  *# Tuberculosis*

        result = "No Tuberculosis Detected"

        confidence = probability\*100

    else:

        predicted\_class = 1  *# Normal*

        result = "Tuberculosis Detected"

        confidence = (1 - probability)\*100

    return result, probability, confidence

*# Example usage*

img\_path = r"C:\Users\nagavardhan\OneDrive\Desktop\Project77777\Split data1\test1\Augmented\_Tuberculosis\Tuberculosis-59\_aug0.png"

result, probability, confidence = predict\_image(img\_path)

*# Output*

print(f"Prediction Result: {result}")

print(f"Prediction Probability (Raw Output): {probability:.4f}")

print(f"Prediction Confidence: {confidence:.4f}%")

import numpy as np

import os

from tensorflow.keras.preprocessing import image

from tensorflow.keras.applications.densenet import DenseNet121, preprocess\_input

from tensorflow.keras.models import load\_model

*# Load models*

capsnet\_model = load\_model(r"E:\SDP\model\capsnet\_feature\_model.h5")

densenet\_model = DenseNet121(weights="imagenet", include\_top=False, pooling="avg")

*# Function to preprocess image*

def preprocess\_image(img\_path):

    img = image.load\_img(img\_path, target\_size=(224, 224))

    img\_array = image.img\_to\_array(img)

    img\_array = np.expand\_dims(img\_array, axis=0)

    img\_array = preprocess\_input(img\_array)

    return img\_array

*# Function to extract features*

def extract\_features\_from\_image(img\_array):

    features = densenet\_model.predict(img\_array)

    return features

*# Function to predict*

def predict\_image(img\_path):

    img\_array = preprocess\_image(img\_path)

    features = extract\_features\_from\_image(img\_array)

    prediction = capsnet\_model.predict(features)

    probability = prediction[0][0]  *# Get the number from (1,1) array*

    threshold = 0.7

    if probability > threshold:

        predicted\_class = 0  *# Tuberculosis should be 1 and tuberculosis, but here 0 is tuberculosis.*

        result = "No Tuberculosis Detected"

        confidence = probability\*100

    else:

        predicted\_class = 1  *# Normal should be 0 and normal, but here 1 is normal.*

        result = "Tuberculosis Detected"

        confidence = (1 - probability)\*100

    return result, probability, confidence

*# Example usage*

img\_path = r"C:\Users\nagavardhan\OneDrive\Desktop\Project77777\Split data1\test1\Normal\Normal-87.png"

result, probability, confidence = predict\_image(img\_path)

*# Output*

print(f"Prediction Result: {result}")

print(f"Prediction Probability (Raw Output): {probability:.4f}")

print(f"Prediction Confidence: {confidence:.4f}%")