## CONTENTS

[CONTENTS iii](#_TOC_250001)

[LIST OF FIGURES v](#_TOC_250000)

LIST OF TABLES vi

LIST OF ACRONYMS vii

CHAPTER 1 INTRODUCTION

* 1. INTRODUCTION 1
  2. OVERVIEW OF THE PROJECT 1
  3. CHALLENGES PRESENT IN PROJECT 2
  4. PROJECT STATEMENT 3
  5. OBJECTIVES 3
  6. SCOPE OF THE PROJECT 4

CHAPTER 2 BACKGROUND

* 1. INTRODUCTION 5
  2. LITERATURE SURVEY 5

CHAPTER 3 METHODOLOGY

* 1. DATASET 8
  2. TECHNIQUES USED 9

*iii*

* + 1. DATA AUGUMENTATION 9
    2. DATA VISUALIZATION 10
    3. HYPER PARAMETER TUNING 12
    4. ACTIVATION FUNCTIONS 13
    5. LOSS FUNCTIONS 13
    6. OPTIMIZERS 15
    7. MODELS 16

CHAPTER 4 RESULTS

* 1. RESULTS AND DISCUSSIONS 18

CHAPTER 5 CONCLUSION

* 1. CONCLUSION 33

CHAPTER 6 FUTURE SCOPE

* 1. FUTURE SCOPE 35

APPENDICES 38

REFERENCES 75

*iv*

### LIST OF FIGURES

* 1. CONFUSION MATRIX OF GOOGLENET 9
  2. CONFUSION MATRIX OF ALEXNET 10
  3. CONFUSION MATRIX OF RESNET 11
  4. CONFUSION MATRIX OF LENET 17
  5. ACCURACY OF GOOGLENET 19
  6. LOSS OF GOOGLENET 21
  7. ACCURACY OF RESNET 23
  8. LOSS OF RESNET 26
  9. ACCURACY OF LENET 27
  10. LOSS OF LENET 28
  11. ACCURACY OF VGG16 28
  12. LOSS OF VGG16 29
  13. ACCURACY OF VGG19 30
  14. LOSS OF VGG19 31
  15. ACCURACY OF HYBRID MODEL 32
  16. LOSS OF HYBRID MODEL 33
  17. GRAD-CAM ON RANDOM IMAGE 1 29
  18. GRAD-CAM ON RANDOM IMAGE 2 30
  19. GRAD-CAM ON RANDOM IMAGE 3 31
  20. GRAD-CAM ON RANDOM IMAGE 4 32
  21. GRAD-CAM ON RANDOM IMAGE 5 33

*v*

**LIST OF TABLES**

1. COMPARISION METRICS 26

*vi*

**LIST OF ACRONYMS**

|  |  |
| --- | --- |
|  |  |
| AI | Artificial Intelligence |
| DL | Deep Learning |
| CNN | Convolutional Neural Network |
| XAI | Explainable Artificial Intelligence |
| TB | Tuberculosis |
| WHO | World Health Organization |
| Grad-CAM | Gradient-weighted Class Activation Mapping |
| X-ray | X-radiation (Radiograph) |
| ReLU | Rectified Linear Unit |
| FP | False Positive |
| FN | False Negative |
| TP | True Positive |
| TN | True Negative |
| ROC | Receiver Operating Characteristic |
| AUC | Area Under the Curve |
|  |  |
| AI | Artificial Intelligence |
| DL | Deep Learning |
| CNN | Convolutional Neural Network |
| XAI | Explainable Artificial Intelligence |
| TB | Tuberculosis |

*vii*

**Chapter 1**

**Introduction**

* 1. INTRODUCTION

Tuberculosis (TB) continues to be a serious global health challenge, particularly affecting developing and under-resourced countries. According to the World Health Organization (WHO), TB remains one of the top ten causes of death worldwide. Early and accurate diagnosis plays a vital role in controlling the spread and mortality of TB. While chest X-ray imaging is commonly used in TB detection due to its speed and cost-effectiveness, interpretation of X-rays requires trained radiologists—a resource not always readily available in remote or high-burden regions.

The emergence of Artificial Intelligence (AI) and Deep Learning (DL) has paved the way for automated diagnostic systems, especially in medical imaging. Convolutional Neural Networks (CNNs) have proven highly effective for image classification tasks, including disease detection in radiographs. However, the major limitation with most deep learning models is their “black box” nature, which makes it difficult for clinicians to trust and understand how decisions are made. This has led to the incorporation of Explainable AI (XAI) techniques, which aim to enhance transparency and reliability in AI-driven healthcare systems.

* 1. OVERVIEW OF PROJECT

This project focuses on the development and implementation of an AI-based system for the detection of tuberculosis using chest X-ray images. A hybrid CNN architecture is designed by integrating the strengths of multiple established models such as ResNet, GoogleNet, AlexNet, VGG19, and LeNet. The system is trained on a labeled dataset consisting of 700 TB-positive and 3500 Normal X-ray images. The images are preprocessed, normalized, and split into training and validation sets.

The hybrid model is designed to optimize feature extraction and classification by stacking and integrating convolutional layers from different architectures. Each sub-network contributes unique capabilities, such as residual learning from ResNet and inception modules from GoogleNet, which collectively enhance the model’s learning capacity.

To address the issue of interpretability, Grad-CAM (Gradient-weighted Class Activation Mapping) is applied to visualize the parts of the X-ray that the model uses to make predictions. Grad-CAM provides heatmaps that help radiologists and researchers understand and validate the model’s focus regions. The goal is not only to achieve high diagnostic accuracy but also to ensure that the model’s decisions are explainable and clinically relevant.

* 1. CHALLENGES PRESENT IN THIS PROJECT

The project involved several technical and practical challenges:

1. **Data Imbalance:** The dataset has significantly more Normal images compared to TB-positive samples, leading to a class imbalance that can bias model performance.
2. **Model Interpretability:** Deep learning models, while accurate, lack intuitive explanation unless supplemented with XAI techniques.
3. **Generalization:** Ensuring the model performs well on unseen data is critical to deploying it in real-world clinical settings.
4. **Computational Requirements:** Training hybrid deep learning models demands considerable computational power and optimization.
5. **Visual Accuracy:** Ensuring that Grad-CAM heatmaps align with actual pathological features is essential for clinical validation.
   1. PROJECT STATEMENT

To develop a hybrid deep learning model capable of accurately detecting tuberculosis in chest X-ray images, while also integrating Explainable AI techniques like Grad-CAM to visualize and validate the decision-making process. The system should be interpretable, reliable, and suitable for potential use in healthcare environments with limited radiological expertise.

* 1. OBJECTIVES

The primary objectives of the project are:

* To design a hybrid CNN model that combines the strengths of multiple architectures.
* To train and evaluate the model on a curated dataset of TB and Normal chest X-rays.
* To implement Grad-CAM for model explainability and visualize feature importance.
* To compare the hybrid model with standard architectures in terms of validation accuracy and interpretability.
* To provide a scalable and interpretable AI-based solution for TB detection in medical imaging.
  1. SCOPE OF THE PROJECT

The scope of this project includes developing a classification system for binary image classification—TB and Normal—using CNN-based models. It explores the impact of combining features from multiple architectures and validates them using performance metrics and visual explanation tools like Grad-CAM. The model is evaluated using metrics such as accuracy, precision, recall, and confusion matrices. The project is limited to 2D chest X-ray images and does not cover other types of medical imaging or TB variants.

Further enhancements may include real-time deployment, integration into clinical decision support systems, and the use of larger, multi-institutional datasets to improve generalization across populations.

**Chapter 2**

**Background**

* 1. INTRODUCTION

Medical imaging has long played a crucial role in the early detection and diagnosis of diseases. Among these, chest X-rays remain one of the most widely used diagnostic tools for pulmonary diseases, including Tuberculosis (TB). However, the accurate interpretation of these images relies heavily on expert radiologists, which poses significant challenges in rural and resource-limited settings. This bottleneck creates a need for automated diagnostic systems that can replicate or supplement expert decision-making.

With the advancement of artificial intelligence (AI), deep learning (DL) has emerged as a powerful technique capable of learning complex patterns from large datasets. In the field of medical image analysis, Convolutional Neural Networks (CNNs) have become the backbone of several successful diagnostic tools. However, while CNNs can produce high classification accuracy, they often lack transparency in their decision-making process. This shortcoming has given rise to Explainable AI (XAI), which focuses on making black-box models more interpretable and trustworthy to end users, particularly in sensitive domains like healthcare.

This chapter explores the background of using AI for TB detection and outlines the state-of-the-art research in this domain. It presents a literature survey covering various models, datasets, and methods that have contributed to the advancement of automated TB diagnosis, as well as recent developments in XAI approaches applied to medical imaging.

* 1. LITERATURE SURVEY

**Deep Learning in TB Detection**

Lakhani and Sundaram (2017) demonstrated that deep CNNs can outperform traditional machine learning approaches in detecting TB from chest radiographs. Using architectures like AlexNet and GoogLeNet, they achieved high sensitivity and specificity, showing the feasibility of AI-driven TB screening in clinical practice.

Pasa et al. (2019) proposed lightweight deep learning architectures optimized for speed and accuracy to enable TB screening in low-resource settings. Their study focused on minimizing computational complexity without significantly compromising accuracy, making their models more deployable in real-world scenarios.

Melendez et al. (2016) developed a computer-aided detection (CAD) system that provided decision support to radiologists. The system used pattern recognition and image processing techniques to highlight abnormal regions, reducing inter-observer variability and increasing diagnostic efficiency.

**Convolutional Architectures and Hybrid Models**

Several studies have explored the use of various CNN architectures such as ResNet, VGGNet, DenseNet, and Inception for TB classification. These models differ in depth, parameter efficiency, and feature extraction capabilities.

Litjens et al. (2017) conducted a broad survey on deep learning applications in medical imaging, emphasizing the success of CNNs in classification, segmentation, and detection tasks. Their work set the foundation for using DL in disease diagnosis, including TB.

Our project builds upon this foundation by constructing a hybrid CNN model combining ResNet, VGG19, LeNet, AlexNet, and GoogleNet. The idea of a hybrid network stems from the goal of extracting the most discriminative features from each model and integrating them into a single architecture for improved performance.

**XAI Techniques in Medical Imaging**

Despite the success of DL in medical diagnostics, the lack of interpretability remains a challenge. Selvaraju et al. (2017) introduced Grad-CAM, an XAI technique that generates heatmaps indicating which regions of an image were most influential in a model’s decision. Grad-CAM has been widely adopted for explaining decisions in medical image classification tasks.

Arun et al. (2020) evaluated the reliability of saliency maps such as Grad-CAM and other XAI tools, showing that while these tools improve transparency, they also need to be used cautiously, especially in clinical settings where incorrect interpretations can have serious implications.

Shen et al. (2017) discussed the ethical concerns and deployment risks of black-box models in clinical diagnosis, advocating for the integration of interpretability frameworks as standard components in medical AI systems.

**Application to Tuberculosis**

The WHO (2023) Global TB Report emphasizes the critical need for early diagnosis and scalable screening systems. AI-based detection systems are identified as potential game-changers, particularly in high-burden regions where radiologist availability is limited.

In our project, we directly address this global need by developing a deep learning-based TB classifier enhanced with Grad-CAM. The use of Grad-CAM allows clinicians to verify the model’s predictions visually, thereby increasing their confidence in the system’s decisions.

The dataset used in this project—consisting of 700 TB and 3500 Normal images—is similar in scope to publicly available datasets used in prior works, such as the Montgomery and Shenzhen datasets. These datasets have been widely used in benchmarking AI models for TB classification.

**Chapter 3**

**Methodology**

* 1. DATA SET

The dataset utilized in this study is the Tuberculosis (TB) Chest X-ray Radiography Database, which provides a rich collection of chest X-ray images specifically categorized into two classes: TB-positive and Normal. It comprises a total of 4,200 images, out of which 700 are labeled as TB and 3,500 are labeled as Normal. The dataset is publicly accessible and has been widely adopted in previous research on pulmonary disease detection using machine learning.

One of the critical characteristics of this dataset is the class imbalance, with significantly fewer TB-positive samples in comparison to Normal images. This imbalance introduces a risk of biased learning during model training, where the algorithm might become skewed toward predicting the majority class. To counter this, appropriate augmentation techniques and class weighting strategies were applied during model development to ensure balanced learning and improved model generalization.

All images in the dataset varied in resolution, brightness, and quality, reflective of real-world clinical variability. To standardize the inputs for deep learning models, each image was resized to a dimension of 64×64 pixels. Normalization was performed to scale pixel intensities between 0 and 1, thereby reducing the impact of lighting variation and expediting the convergence of neural networks. Images were retained in RGB format to preserve channel information compatible with transfer learning models.

The dataset was divided into training and validation subsets using an 80:20 stratified split, ensuring that both classes maintained their original distribution in each subset. This division allowed for unbiased performance evaluation and helped prevent data leakage across the training and validation phases.

.

* 1. TECHNIQUES USED

The development and training of deep learning models for TB detection involved several important preprocessing and optimization techniques. These techniques ensured the reliability, scalability, and interpretability of the models. This section provides a comprehensive account of the methods used for enhancing the dataset, visualizing its structure, optimizing model parameters, and configuring the training workflow.

The overarching pipeline followed for all models included loading and labeling the data, applying preprocessing and augmentation, performing model training and validation, and finally, evaluating the outcomes using performance metrics and visual interpretation techniques. For transparency, each sub-method used during this process is explained in the following sections.

* + 1. DATA AUGMENTATION

To address the dataset imbalance and enhance the model’s ability to generalize, various data augmentation strategies were employed. Data augmentation is particularly important in medical imaging applications, where datasets are typically limited due to privacy concerns and acquisition challenges. Augmentation artificially expands the dataset by generating modified versions of existing images, thereby exposing the model to a wider range of possible input variations.

In this study, a series of transformations were applied to the training images during runtime using Keras, ImageDataGenerator. These transformations included:

* Horizontal flipping: to simulate symmetry and directional invariance in chest images.
* Random rotation: up to ±20 degrees to account for patient posture variations.
* Zoom transformations: up to 10% to enhance scale invariance.
* Width and height shifts: up to 10% to accommodate off-center images.
* Brightness adjustments: to replicate X-ray exposures under different lighting conditions.
* Shear transformations and slight blurring: to provide spatial and texture variation.
* Random noise injection: selectively applied to increase robustness against artifacts.

These augmentations were applied only to the training data, not to the validation set, to ensure a fair evaluation. This technique proved effective in preventing overfitting, especially for deeper networks like VGG19 and GoogleNet, which are prone to memorizing small datasets.

* + 1. DATA VISUALIZATION

Data visualization played a central role in both the exploration and validation phases of this project. Visual tools provided critical insights into the distribution, balance, and structure of the dataset, as well as the behavior and performance of models during and after training. Visualization also facilitated an understanding of how well the models were learning and which areas of the input images influenced predictions.

To begin with, class distribution histograms were plotted to reveal the significant imbalance between the Normal and TB classes. These plots justified the need for augmentation strategies to prevent the model from being biased toward the majority class.

Sample image grids were generated from both classes to visually inspect the quality of X-ray images and confirm successful preprocessing steps such as resizing, normalization, and augmentation. These visual checks helped to verify that no valuable diagnostic features were lost during the transformation processes.

During the model training phase, training and validation accuracy and loss curves were monitored across epochs. These plots allowed for early detection of overfitting or underfitting. If the training accuracy increased while the validation accuracy plateaued or decreased, the model was likely overfitting. This information guided the application of techniques such as dropout and early stopping.

Post-training evaluations included the use of confusion matrices for each model, providing a breakdown of true positives, true negatives, false positives, and false negatives. These matrices highlighted the sensitivity and specificity of each model and identified classes where misclassifications were most frequent.

In addition, ROC (Receiver Operating Characteristic) curves and AUC (Area Under Curve) scores were calculated for the most promising models. ROC curves illustrate the trade-off between sensitivity and specificity, while AUC scores offer a consolidated measure of model performance. These metrics were particularly valuable in evaluating models like ResNet and VGG19, which showed high overall accuracy.

Lastly, Grad-CAM heatmaps were produced to interpret model predictions. These visual explanations helped confirm that the model’s decisions were based on clinically relevant areas in the chest X-rays, such as lesions, cavities, or consolidations typically associated with TB.

* + 1. HYPER PARAMETER TUNING

Hyperparameter tuning is a fundamental step in training deep learning models. The goal is to identify a set of optimal parameters that result in the best performance in terms of accuracy, convergence speed, and generalization. In this project, several key hyperparameters were explored and fine-tuned through iterative testing and empirical observations.

Batch size was tested with values of 32 and 64. Smaller batch sizes allowed the model to generalize better and required less memory, while larger sizes offered faster training but risked poor convergence. A batch size of 32 was found to be the most effective across all models, especially with complex architectures like VGG19 and GoogleNet.

Epochs were initially set between 10 and 15, with the use of an early stopping mechanism to halt training when validation performance stopped improving. This approach avoided overfitting while ensuring that sufficient learning occurred.

Learning rate was one of the most critical parameters. A starting rate of 0.001 was used, followed by adaptive reductions using ReduceLROnPlateau. This scheduler decreased the learning rate if validation loss stagnated for three consecutive epochs, allowing finer adjustments during the later stages of training.

Dropout rates were applied in the range of 0.3 to 0.5 in the fully connected layers to prevent overfitting, particularly for deep networks. Dropout randomly deactivates neurons during training, forcing the model to develop redundant representations and improving robustness.

Additional techniques such as kernel initializers (He Normal and Xavier) were used to optimize weight initialization, further accelerating convergence and improving overall accuracy. These configurations were kept consistent across experiments to ensure a fair comparison between different models.

* + 1. ACTIVATION FUNCTION

Activation functions introduce non-linearity into neural networks, enabling them to learn complex patterns and representations. In this study, two types of activation functions were used based on the position within the architecture:

* ReLU (Rectified Linear Unit) was employed in all convolutional and hidden layers. ReLU is defined as f(x)=max(0,x), allowing only positive activations to pass through. This function is computationally efficient and reduces the likelihood of vanishing gradient problems, especially in deep networks. It helps speed up the learning process and supports the development of sparse activations.
* Softmax was used in the final layer of each model. This function transforms the output scores into probabilities that sum up to one. For binary classification, although a single sigmoid output could suffice, softmax with two neurons was used along with categorical cross-entropy loss for consistency with multi-class capabilities and visualization.

By combining ReLU in the hidden layers and Softmax at the output, the models were able to capture rich, hierarchical features while ensuring probabilistic interpretability during prediction.

* + 1. LOSS FUNCTION

Loss functions are at the core of every learning algorithm, providing a quantitative measurement of how far the model's predictions deviate from the actual labels. During the training process, the goal is to minimize this loss so that the model’s predictions become as accurate as possible.

For this project, the Categorical Cross-Entropy Loss function was utilized. It is particularly effective in classification problems involving one-hot encoded outputs. Given the nature of the classification task—distinguishing between TB and Normal cases—each image is associated with a categorical label (e.g., [1,0] for TB and [0,1] for Normal). Categorical cross-entropy computes the dissimilarity between the predicted probability distribution and the actual label distribution.

The loss function is mathematically represented as:

Where:

* is the true label (0 or 1),
* is the predicted probability for the corresponding class,
* n is the total number of classes (in this case, 2).

The function penalizes incorrect predictions more when the model is highly confident but wrong. This encourages the model to adjust its weights to make better predictions in future iterations. For imbalanced datasets, weights can be applied to the classes in the loss function to give more importance to the minority class. This strategy was explored during experimentation but was ultimately not necessary, as data augmentation sufficiently addressed class imbalance.

Using categorical cross-entropy enabled the models to learn from the probabilistic differences in class predictions effectively and aligned well with the use of the softmax activation function in the output layer.

* + 1. OPTIMIZERS

Optimizers are a fundamental component of training deep learning models, as they guide the updating of model weights to minimize the loss function. The selection and configuration of the optimizer significantly influence the model's learning speed, convergence behavior, and final accuracy.

For this project, the Adam (Adaptive Moment Estimation) optimizer was consistently employed across all models due to its proven efficiency, adaptive learning rate mechanism, and ability to handle sparse gradients. Adam combines the principles of two other optimization algorithms—Momentum and RMSProp to update weights using estimates of first and second moments of gradients. It is especially suitable for tasks involving complex, high-dimensional data like medical images.

The optimizer was configured with the following hyperparameters:

* Learning rate (𝛼): 0.001 – A widely accepted default value for Adam, offering a balance between convergence speed and stability.
* Beta 1: 0.9 – Controls the exponential decay rate for the moving average of the first moment (mean of gradients).
* Beta 2: 0.999 – Controls the exponential decay rate for the moving average of the second moment (variance of gradients).
* Epsilon: 1e-08 – A small constant used to prevent division by zero in the weight update process.

To improve learning efficiency and prevent plateauing during training, a learning rate scheduler was used in combination with Adam. Specifically, the ReduceLROnPlateau scheduler was applied to monitor the validation loss. When no improvement in validation loss was observed for 3 consecutive epochs, the learning rate was reduced by a factor of 0.1. This adaptive scheduling allowed for larger initial learning steps and finer adjustments later in training, ensuring smoother convergence and reduced oscillation near minima.

Additionally, the training process incorporated the EarlyStopping mechanism. This technique was set to monitor the validation loss with a patience value of 5 epochs. If the validation loss did not improve during this period, training was terminated early to prevent overfitting and unnecessary computation. EarlyStopping was particularly effective in models like VGG19 and ResNet, where prolonged training without gain could otherwise result in increased validation loss and generalization error.

The combined use of Adam optimizer, learning rate scheduling, and early stopping led to improved training stability, better generalization, and reduced overfitting, especially in deep architectures. These optimization strategies also helped conserve computational resources and training time while maintaining high performance across all models.

* + 1. MODELS

A wide range of deep learning models were designed, trained, and evaluated in this project to identify the most effective approach for TB detection from chest X-ray images. Each model was chosen for its unique architecture, computational characteristics, and potential contribution to the hybrid approach.

The following six models were implemented:

* LeNet – A simple and efficient baseline convolutional neural network with limited depth.
* AlexNet – A deeper architecture featuring overlapping pooling, ReLU activations, and dropout regularization.
* VGG19 – Known for its uniform layer structure and significant depth, which enables deep hierarchical feature extraction.
* GoogleNet (Inception) – Uses inception modules that combine multiple filter sizes to process information at multiple scales simultaneously.
* ResNet – Employs residual connections to combat the vanishing gradient problem and allow the training of very deep networks.
* Hybrid CNN – A custom architecture combining blocks from multiple pre-existing models to leverage their individual strengths while introducing dropout and interpretability enhancements.

Each of these models underwent rigorous training and evaluation using the same dataset and metrics to ensure a fair comparison. Performance was judged based on validation accuracy, validation loss, training behavior, and explainability (via Grad-CAM).

Chapter 4

**Results**

* 1. RESULTS AND DISCUSSIONS

This chapter presents a comprehensive analysis of the outcomes obtained from training and evaluating multiple deep learning models for tuberculosis (TB) detection using chest X-ray images. The results are evaluated across several performance metrics including validation accuracy, validation loss, confusion matrix, and Grad-CAM visualizations. Additionally, comparative insights are drawn among the models to determine the most effective solution in terms of accuracy, generalization, and interpretability.

**Evaluation Metrics**

To ensure a holistic evaluation of model performance, the following metrics were used:

* **Validation Accuracy**: Percentage of correctly predicted labels in the validation dataset.
* **Validation Loss**: Measure of prediction error on validation data using categorical cross-entropy.
* **Confusion Matrix**: Visual breakdown of true positive, true negative, false positive, and false negative predictions.
* **Precision**: Ratio of true positives to total predicted positives.
* **Recall (Sensitivity)**: Ratio of true positives to actual positives.
* **F1 Score**: Harmonic mean of precision and recall.
* **Grad-CAM**: Visual heatmaps for explainability to understand which regions influenced predictions.

These metrics provide not only a numerical evaluation but also a clinical perspective by interpreting how well the models can identify TB-infected cases from healthy lungs.

**Model Comparison Based on Performance**

All six models were evaluated on the same dataset with identical preprocessing and training configurations. The table below summarizes their performance on the validation set:

| **Model** | **Validation Accuracy** | **Validation Loss** |
| --- | --- | --- |
| LeNet | 97.5% | 0.0924 |
| AlexNet | 96.8% | 0.0907 |
| VGG19 | 98.4% | 0.0408 |
| GoogleNet | 96.9% | 0.0915 |
| ResNet | 97.7% | 0.1062 |
| Hybrid CNN | 95.3% | 0.1786 |

Table 1: comparison of the models performance

VGG19 achieved the highest accuracy and lowest validation loss, making it the most reliable model in terms of raw performance. However, the hybrid model, despite a slightly lower accuracy, showed better interpretability using Grad-CAM, making it suitable for clinical deployment.

**Confusion Matrix Analysis**

Each model's confusion matrix was evaluated to analyze its ability to correctly classify both TB-positive and Normal cases. VGG19 and ResNet demonstrated high sensitivity and specificity, while LeNet and AlexNet occasionally misclassified a few TB-positive images as Normal due to limited capacity to capture complex lung features.

* **LeNet** and **AlexNet**: Mostly accurate but showed minor drops in precision.
* **VGG19** and **ResNet**: Showed strong true positive rates and minimal false negatives.
* **Hybrid Model**: Balanced results but slightly more false positives than expected, possibly due to its broad feature sensitivity.

These insights are vital because in medical diagnosis, false negatives are critical as they represent missed disease detections.

A diagram of a diagram

AI-generated content may be incorrect.

A blue squares with white text

AI-generated content may be incorrect.

Figure 1: Confusion Matrix of GoogleNet

Figure 2: Confusion Matrix of AlexNet

A diagram of a diagram

AI-generated content may be incorrect.A diagram of a diagram

AI-generated content may be incorrect.A diagram of a diagram

AI-generated content may be incorrect.

Figure 4: Confusion Matrix of Lenet

Figure 3: Confusion Matrix of ResNet

**Accuracy and Loss Curves**

Training history plots of accuracy and loss for each model demonstrated convergence trends:

* **LeNet** and **AlexNet**: Converged quickly due to their shallow architectures.
* **VGG19**, **GoogleNet**, and **ResNet**: Showed gradual but stable convergence, with minimal gaps between training and validation accuracy, indicating low overfitting.
* **Hybrid CNN**: Required slightly more epochs to stabilize, possibly due to its combined architecture, but maintained generalization.

These graphs also confirmed that early stopping and learning rate reduction were effective, with most models halting training before 15 epochs while still achieving optimal performance.

A graph with red and blue lines

AI-generated content may be incorrect.A graph with red and blue lines

AI-generated content may be incorrect.

Figure 5: Accuracy of GoogleNet

Figure 6: Loss of GoogleNet

A graph with a line graph

AI-generated content may be incorrect.A graph with a line

AI-generated content may be incorrect.Figure 7: Accuracy of ResNet

Figure 8: Loss of ResNet

A graph with red and blue lines

AI-generated content may be incorrect.

A graph with red and blue lines

AI-generated content may be incorrect.Figure 9: Accuracy of LeNet

Figure 10: Loss of LeNet

A graph with a line graph

AI-generated content may be incorrect.

A graph with a line graph

AI-generated content may be incorrect.Figure 11: Accuracy of VGG16

Figure 12: Loss of VGG16

A graph with a line and a red line

AI-generated content may be incorrect.

A graph with lines and numbers

AI-generated content may be incorrect.Figure 13: Accuracy of VGG19

Figure 14: Loss of VGG19

A graph with a line graph

AI-generated content may be incorrect.

Figure 15: Accuracy of Hybrid model

A graph with red and blue lines

AI-generated content may be incorrect.

Figure 16: Loss of Hybrid model

**Explainability Using Grad-CAM**

To ensure transparency in model predictions, Grad-CAM visualizations were generated, particularly for the hybrid model. The heatmaps highlighted the regions of the lungs the model focused on when identifying TB infections.

Key findings:

* The model consistently highlighted upper lobe opacities, cavitation zones, and other lesion-like structures in TB-positive images.
* Normal images showed diffuse attention, suggesting the model was correctly identifying the absence of pathology.
* The hybrid model's Grad-CAM outputs were particularly sharp and interpretable, showing its strength in visual feature localization.

These findings support the clinical interpretability of the system, allowing radiologists to understand and verify the reasoning behind AI-based predictions.

A screenshot of a medical scan

AI-generated content may be incorrect.

Figure 17: Grad-CAM on random image 1

A close-up of a person's chest

AI-generated content may be incorrect.A close up of a person's chest

AI-generated content may be incorrect.

Figure 19: Grad-CAM on random image 3

Figure 18: Grad-CAM on random image 2

A close-up of a radiography

AI-generated content may be incorrect.A close up of a person's face

AI-generated content may be incorrect.

Figure 20: Grad-CAM on random image 4

Figure 21: Grad-CAM on random image 5

**Discussion on Generalization and Clinical Readiness**

While VGG19 provided the highest accuracy, it required significant computational resources and longer training durations. Conversely, the hybrid model, which was designed by combining the strengths of multiple networks, achieved a balanced trade-off between accuracy and interpretability. The inclusion of dropout layers and simplified convolutional blocks ensured its generalization on unseen data.

The results from confusion matrices and Grad-CAM visualizations validate the hybrid model's potential to assist radiologists in TB screening, particularly in resource-limited settings where computational power may be restricted.

In conclusion, each model exhibited specific strengths:

* **LeNet and AlexNet**: Useful for real-time applications due to speed and simplicity.
* **VGG19 and ResNet**: Best suited for high-performance medical analysis systems.
* **Hybrid Model**: Optimal for interpretable and deployable diagnostic tools.

Chapter 5

**Conclusion**

* 1. CONCLUSION

The increasing global burden of tuberculosis (TB), particularly in low-resource settings, highlights the urgent need for rapid, accurate, and scalable diagnostic tools. This project focused on addressing this challenge by leveraging the power of deep learning and explainable artificial intelligence (XAI) to detect TB from chest X-ray images. The methodology centered on developing and comparing multiple convolutional neural network (CNN) architectures, including LeNet, AlexNet, VGG19, GoogleNet, ResNet, and a custom hybrid model. Each model was rigorously trained, evaluated, and analyzed using standard performance metrics and visualization tools.

The dataset used—comprising 4,200 images (700 TB-positive and 3,500 Normal)—was preprocessed and augmented to ensure high-quality, balanced training data. Through consistent application of data augmentation, careful hyperparameter tuning, and regularization techniques such as dropout and early stopping, all models achieved high validation accuracies, with VGG19 emerging as the top performer at 98.4%.

The custom hybrid model, though slightly behind in accuracy, offered enhanced explainability through Grad-CAM visualizations. This transparency is crucial in medical diagnostics, where trust in AI decisions must be built through interpretable outputs. The hybrid model successfully highlighted the pathological regions of interest in TB-infected images, aligning with clinical understanding and validating its decision-making process.

Comprehensive performance evaluation using accuracy, loss, confusion matrices, and Grad-CAM heatmaps affirmed that the developed models are not only accurate but also clinically relevant. The confusion matrix analysis revealed high sensitivity and specificity, indicating the models’ competence in identifying both TB and Normal cases with minimal misclassification.

In summary, this project demonstrates that deep learning models—especially when enhanced with explainable AI techniques—can serve as powerful diagnostic aids for TB detection. Among all architectures explored, the hybrid model proved to be the most balanced solution, combining robust performance with clinical interpretability. Such models have the potential to support radiologists, reduce diagnostic delays, and improve early detection outcomes in real-world healthcare environments.

This study sets a strong foundation for future research in automated TB diagnosis and paves the way for deploying AI-driven tools in clinical settings, particularly where expert radiologists may not be readily available.

Chapter 6

**Future Scope**

* 1. FUTURE SCOPE

The promising results obtained through this study demonstrate the potential of deep learning and explainable AI (XAI) in facilitating automated diagnosis of tuberculosis (TB) from chest X-ray images. However, the field of AI in medical diagnostics is continuously evolving, and there remain several opportunities to enhance the methodology, broaden applicability, and improve real-world clinical integration.

One of the most significant areas for future work lies in dataset expansion and diversification. The current dataset, though effective for experimentation, is limited in size and demographic variety. Larger datasets containing samples from various age groups, genders, ethnic backgrounds, and different geographical regions could improve the model's ability to generalize across populations. Incorporating images with multiple disease labels, such as pneumonia, COVID-19, or lung cancer, could also enable multi-disease classification systems that are more clinically relevant.

Another valuable enhancement is the integration of multi-modal medical data. Combining chest X-rays with additional patient information such as symptoms, medical history, laboratory results (e.g., sputum test, CBC), and clinical notes could enrich the feature space and enable more accurate and context-aware predictions. Such multimodal models are gaining importance in AI research and have shown great potential in decision support systems.

From an algorithmic perspective, incorporating advanced deep learning architectures such as Vision Transformers (ViTs), Capsule Networks, or attention-based CNNs could lead to improved accuracy and spatial understanding of complex radiological patterns. These models can learn global dependencies and may be more robust to image distortions or unseen variations. Additionally, using ensemble methods, which combine the strengths of multiple models (e.g., VGG19, ResNet, and the Hybrid model), could offer more stable and accurate predictions.

A major direction for future scope involves improving explainability and trustworthiness. While Grad-CAM provided meaningful visualizations in this study, more advanced XAI techniques such as Integrated Gradients, SHAP, or LIME can be explored for deeper insight into model reasoning. Trust in AI systems is critical for clinical adoption, and transparent decision-making must be prioritized in deployment scenarios.

Another important area is the deployment and usability of the developed model in real-world environments. Building a lightweight, optimized version of the model for integration into mobile health applications or edge devices (e.g., Raspberry Pi, Jetson Nano) would make TB screening accessible in remote or under-resourced areas. Cloud-based diagnostic services can also be developed to support large-scale screening through web platforms and telemedicine portals.

In addition, the development of a user-friendly interface for radiologists and healthcare professionals can bridge the gap between technical systems and medical users. Such interfaces could allow uploading of X-rays, viewing of classification results, heatmaps for explainability, and suggestions for further clinical action. This would transform the model from a research prototype to a practical clinical decision support tool.

In terms of research contribution, future work can explore longitudinal tracking of TB progression through X-ray time-series data. AI models trained to identify not only presence but also the stage of TB infection can assist in treatment monitoring and therapy adjustment. Moreover, integration with electronic health record (EHR) systems and hospital information systems (HIS) can enable seamless AI-assisted diagnostics in hospital workflows.

Lastly, comprehensive clinical trials and regulatory approvals will be essential before any AI-based TB diagnostic system is deployed in real clinical settings. Collaboration with healthcare institutions for data validation, ethical testing, and real-world pilot deployment will ensure the safety, fairness, and reliability of the system.

**Appendices**

## ****Appendix 1****

**Code and output:**

!pip install kaggle

Requirement already satisfied: kaggle in /usr/local/lib/python3.11/dist-packages (1.7.4.2)

Requirement already satisfied: bleach in /usr/local/lib/python3.11/dist-packages (from kaggle) (6.2.0)

Requirement already satisfied: certifi>=14.05.14 in /usr/local/lib/python3.11/dist-packages (from kaggle) (2025.4.26)

Requirement already satisfied: charset-normalizer in /usr/local/lib/python3.11/dist-packages (from kaggle) (3.4.1)

Requirement already satisfied: idna in /usr/local/lib/python3.11/dist-packages (from kaggle) (3.10)

Requirement already satisfied: protobuf in /usr/local/lib/python3.11/dist-packages (from kaggle) (5.29.4)

Requirement already satisfied: python-dateutil>=2.5.3 in /usr/local/lib/python3.11/dist-packages (from kaggle) (2.9.0.post0)

Requirement already satisfied: python-slugify in /usr/local/lib/python3.11/dist-packages (from kaggle) (8.0.4)

Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from kaggle) (2.32.3)

Requirement already satisfied: setuptools>=21.0.0 in /usr/local/lib/python3.11/dist-packages (from kaggle) (75.2.0)

Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.11/dist-packages (from kaggle) (1.17.0)

Requirement already satisfied: text-unidecode in /usr/local/lib/python3.11/dist-packages (from kaggle) (1.3)

Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from kaggle) (4.67.1)

Requirement already satisfied: urllib3>=1.15.1 in /usr/local/lib/python3.11/dist-packages (from kaggle) (2.4.0)

Requirement already satisfied: webencodings in /usr/local/lib/python3.11/dist-packages (from kaggle) (0.5.1)

import os os.makedirs('/root/.kaggle', exist\_ok=True)

!mv kaggle.json /root/.kaggle/

!chmod 600 /root/.kaggle/kaggle.json

!kaggle datasets download tawsifurrahman/tuberculosis-tb-chest-xray-dataset

Dataset URL: <https://www.kaggle.com/datasets/tawsifurrahman/tuberculosis-tb-chest-xray-dataset>License(s): copyright-authors

!unzip /content/tuberculosis-tb-chest-xray-dataset.zip

Archive: /content/tuberculosis-tb-chest-xray-dataset.zip inflating: TB\_Chest\_Radiography\_Database/Normal.metadata.xlsx inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-10.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-100.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1000.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1001.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1002.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1003.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1004.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1005.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1006.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1007.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1008.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1009.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-101.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1010.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1011.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1012.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1013.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1014.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1015.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1016.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1017.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1018.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1019.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-102.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1020.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1021.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1022.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1023.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1024.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1025.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1026.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1027.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1028.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1029.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-103.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1030.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1031.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1032.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1033.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1034.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1035.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1036.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1037.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1038.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1039.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-104.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1040.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1041.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1042.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1043.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1044.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1045.png

**

**

**

inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1046.png

inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1047.png

inflating:

TB

Chest

Radiography

Database/Normal/Normal-1048

png

import

numpy

as

np

import

pandas

as

ps

import

matplotlib.pyplot

as

plt

import

os

import

glob

import

seaborn

as

sns

import

cv2

import

io

import

tensorflow

as

tf

from

sklearn.model\_selection

import

train\_test\_split

from

tqdm

import

tqdm

import

keras

from

keras.preprocessing

import

image

from

tensorflow.keras.preprocessing.image

import

ImageDataGenerator

from

PIL

import

Image

from

sklearn.utils

import

shuffle

from

keras

import

layers

,

models

,

optimizers

import

ipywidgets

as

widgets

from

keras.utils

import

to\_categorical

from

keras.models

import

Sequential

from

keras.layers

import

Dense

,

Activation

,

Dropout

,

Flatten

,

Conv2D

,

MaxPooling2D

,

Dropout

,

BatchNormalization

,

Concatenate

,

AveragePooling

from

sklearn.metrics

import

classification\_report

,

confusion\_matrix

from

keras.models

import

Sequential

from

keras.regularizers

import

l2

from

keras.callbacks

import

EarlyStopping

from

google.colab.patches

import

cv2\_imshow

labels=

[

"Tuberculosis"

,

"Normal"

]

X=

[]

Y=

[]

image\_size=

(

227

,

227

)

for

i

in

labels

:

path=

"/content/TB\_Chest\_Radiography\_Database/"

+

i

+

"/"

print

(

path

)

fileRead=glob.glob

(

path+

"\*"

)

print

(

len

(

fileRead

))

for

j

in

fileRead

:

image=cv2.imread

(

j

)

image=cv2.resize

(

image

,

image\_size

)

X.append

(

image

)

Y.append

(

i

)

/content/TB\_Chest\_Radiography\_Database/Tuberculosis/

700

/content/TB\_Chest\_Radiography\_Database/Normal/

3500

[1. 0.]

optimizer = optimizers.Adam(learning\_rate=0.0001) from keras.models import Model

from keras.layers import Input, Conv2D, MaxPooling2D, AveragePooling2D, Dropout, Flatten, Dense, Concatenate Google\_X\_train,Google\_X\_test,Google\_Y\_train,Google\_Y\_test=train\_test\_split(X,Y,test\_size=0.2,random\_state=42)

from keras.models import Model from keras.layers import Input, Conv2D, MaxPooling2D, AveragePooling2D, Dropout, Flatten, Dense, Concatenate

def inception\_module(prev\_layer, filters): tower\_1x1 = Conv2D(filters[0], (1, 1), padding='same', activation='relu')(prev\_layer)

tower\_3x3\_reduce = Conv2D(filters[1], (1, 1), padding='same', activation='relu')(prev\_layer) tower\_3x3 = Conv2D(filters[2], (3, 3), padding='same', activation='relu')(tower\_3x3\_reduce)

tower\_5x5\_reduce = Conv2D(filters[3], (1, 1), padding='same', activation='relu')(prev\_layer) tower\_5x5 = Conv2D(filters[4], (5, 5), padding='same', activation='relu')(tower\_5x5\_reduce)

tower\_pool = MaxPooling2D((3, 3), strides=(1, 1), padding='same')(prev\_layer) tower\_pool\_1x1 = Conv2D(filters[5], (1, 1), padding='same', activation='relu')(tower\_pool) return Concatenate(axis=-1)([tower\_1x1, tower\_3x3, tower\_5x5, tower\_pool\_1x1])

# Input layer input\_layer = Input(shape=(227, 227, 3))

# Convolution and Pooling layers x = Conv2D(64, (7, 7), strides=(2, 2), padding='same', activation='relu')(input\_layer) x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)

# Inception modules x = inception\_module(x, [64, 128, 128, 32, 32, 32]) x = inception\_module(x, [128, 192, 96, 64, 64, 64]) x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x) x = inception\_module(x, [192, 208, 48, 64, 64, 64]) x = inception\_module(x, [160, 224, 64, 64, 64, 128]) x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x) x = inception\_module(x, [128, 256, 64, 64, 64, 128]) x = inception\_module(x, [112, 288, 64, 64, 64, 128]) x = inception\_module(x, [256, 320, 128, 128, 128, 128]) x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)

# Fully connected layers x = AveragePooling2D(pool\_size=(4, 4))(x) x = Flatten()(x) x = Dense(1024, activation='relu')(x) x = Dropout(0.4)(x) output\_layer = Dense(2, activation='softmax')(x) # Assuming 4 classes for the classification task

# Create the model

GoogleNet = Model(inputs=input\_layer, outputs=output\_layer)

# Compile the model

GoogleNet.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy']) early\_stopping = EarlyStopping(monitor='val\_loss', patience=3)

# Print the model summary

GoogleNet.summary()

# Model: "functional\_1"

┏━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━┓ ┃ **Layer (type)** ┃ **Output Shape** ┃ **Param #** ┃ **Connected to** ┃ ┡━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━┩ │ input\_layer\_1 │ (None, 227, 227, │ 0 │ - │ │ (InputLayer) │ 3) │ │ │ ├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ conv2d\_43 (Conv2D) │ (None, 114, 114, │ 9,472 │ input\_layer\_1[0]… │ │ │ 64) │ │ │ ├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ max\_pooling2d\_11 │ (None, 57, 57, │ 0 │ conv2d\_43[0][0] │ │ (MaxPooling2D) │ 64) │ │ │ ├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ conv2d\_45 (Conv2D) │ (None, 57, 57, │ 8,320 │ max\_pooling2d\_11… │ │ │ 128) │ │ │

├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ conv2d\_47 (Conv2D) │ (None, 57, 57, │ 2,080 │ max\_pooling2d\_11… │ │ │ 32) │ │ │ ├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ max\_pooling2d\_12 │ (None, 57, 57, │ 0 │ max\_pooling2d\_11… │ │ (MaxPooling2D) │ 64) │ │ │ ├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ conv2d\_44 (Conv2D) │ (None, 57, 57, │ 4,160 │ max\_pooling2d\_11… │ │ │ 64) │ │ │ ├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ conv2d\_46 (Conv2D) │ (None, 57, 57, │ 147,584 │ conv2d\_45[0][0] │ │ │ 128) │ │ │

├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ conv2d\_48 (Conv2D) │ (None, 57, 57, │ 25,632 │ conv2d\_47[0][0] │ │ │ 32) │ │ │ ├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ conv2d\_49 (Conv2D) │ (None, 57, 57, │ 2,080 │ max\_pooling2d\_12… │ │ │ 32) │ │ │ ├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ concatenate\_7 │ (None, 57, 57, │ 0 │ conv2d\_44[0][0], │ │ (Concatenate) │ 256) │ │ conv2d\_46[0][0], │ │ │ │ │ conv2d\_48[0][0], │

│ │ │ │ conv2d\_49[0][0] │ ├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ conv2d\_51 (Conv2D) │ (None, 57, 57, │ 49,344 │ concatenate\_7[0]… │ │ │ 192) │ │ │

├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ conv2d\_53 (Conv2D) │ (None, 57, 57, │ 16,448 │ concatenate\_7[0]… │ │ │ 64) │ │ │ ├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ max\_pooling2d\_13 │ (None, 57, 57, │ 0 │ concatenate\_7[0]… │ │ (MaxPooling2D) │ 256) │ │ │ ├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ conv2d\_50 (Conv2D) │ (None, 57, 57, │ 32,896 │ concatenate\_7[0]… │ │ │ 128) │ │ │

├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ conv2d\_52 (Conv2D) │ (None, 57, 57, │ 165,984 │ conv2d\_51[0][0] │ │ │ 96) │ │ │ ├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ conv2d\_54 (Conv2D) │ (None, 57, 57, │ 102,464 │ conv2d\_53[0][0] │ │ │ 64) │ │ │ ├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ conv2d\_55 (Conv2D) │ (None, 57, 57, │ 16,448 │ max\_pooling2d\_13… │

│ │ 64) │ │ │

Start coding or ├───────────generate─────── with AI.───┼───────────────────┼────────────┼───────────────────┤

│ concatenate\_8 │ (None, 57, 57, │ 0 │ conv2d\_50[0][0], │

│ (Concatenate) │ 352) │ │ conv2d\_52[0][0], │

Start coding or │ g enerate with AI. │ │ │ conv2d\_54[0][0], │ │ │ │ │ conv2d\_55[0][0] │

├─────────────────────┼───────────────────┼────────────┼───────────────────┤

Google\_history=GoogleNet.fit│ max\_pooling2d\_14 │ ((Google\_X\_trainN ne, 29, 29, │ ,Google\_Y\_train,0 validation\_data=│ concate ate\_8[(0Google\_X\_test]… │ ,Google\_Y\_test),epochs=10,batch\_size=32,verbos

│ (MaxPooling2D) │ 352) │ │ │ ├─────────────────────┼───────────────────┼────────────┼───────────────────┤Epoch 1/10

│ conv2d\_57 (**105/105** ━━━━━━━━━━━━━━━━━━━━Conv2D) │ (None , **80s**29 388ms/step - accuracy: 0.8219 - loss: 2.5248 - val\_accuracy: 0.9476 - val\_loss: 0.1177, 29, │ 73,424 │ max\_pooling2d\_14… │

│ │ Epoch 2/10 208) │ │ │

├─────────────────────┼───────────────────┼────────────┼───────────────────┤**105/105** ━━━━━━━━━━━━━━━━━━━━ **35s** 197ms/step - accuracy: 0.9385 - loss: 0.1541 - val\_accuracy: 0.9702 - val\_loss: 0.0665

│ conv2d\_59 (Epoch 3/10 Conv2D) │ (None, 29, 29, │ 22,592 │ max\_pooling2d\_14… │

│ │ **105/105** ━━━━━━━━━━━━━━━━━━━━64) │ │ │ **42s** 206ms/step - accuracy: 0.9708 - loss: 0.0792 - val\_accuracy: 0.9845 - val\_loss: 0.0505

├─────────────────────┼───────────────────┼────────────┼───────────────────┤Epoch 4/10

│ max\_pooling2d\_15 │ (**105/105** ━━━━━━━━━━━━━━━━━━━━None , **40**29**s** 197ms/step - accuracy: 0.9818 - loss: 0.0456 - val\_accuracy: 0.9560 - val\_loss: 0.1072, 29, │ 0 │ max\_pooling2d\_14… │

│ (Epoch 5/10MaxPooling2D) │ 352) │ │ │

├─────────────────────┼───────────────────┼────────────┼───────────────────┤**105/105** ━━━━━━━━━━━━━━━━━━━━ **41s** 196ms/step - accuracy: 0.9788 - loss: 0.0569 - val\_accuracy: 0.9845 - val\_loss: 0.0478

│ conv2d\_56 (Conv2D) │ (None, 29, 29, │ 67,776 │ max\_pooling2d\_14… │

Epoch 6/10

│ │ 192) │ │ │

**105/105** ━━━━━━━━━━━━━━━━━━━━ **21s** 198ms/step - accuracy: 0.9860 - loss: 0.0343 - val\_accuracy: 0.9821 - val\_loss: 0.0559

├─────────────────────┼───────────────────┼────────────┼───────────────────┤

Epoch 7/10

│ conv2d\_58 (Conv2D) │ (None, 29, 29, │ 89,904 │ conv2d\_57[0][0] │

**105/105** ━━━━━━━━━━━━━━━━━━━━ **21s** 195ms/step - accuracy: 0.9872 - loss: 0.0274 - val\_accuracy: 0.9881 - val\_loss: 0.0393

│ │ 48) │ │ │

Epoch 8/10

├─────────────────────┼───────────────────┼────────────┼───────────────────┤

**105/105** ━━━━━━━━━━━━━━━━━━━━ **42s** 204ms/step - accuracy: 0.9950 - loss: 0.0162 - val\_accuracy: 0.9845 - val\_loss: 0.0431

│ conv2d\_60 (Conv2D) │ (None, 29, 29, │ 102,464 │ conv2d\_59[0][0] │ Epoch 9/10

│ │ 64) │ │ │

**105/105** ━━━━━━━━━━━━━━━━━━━━ **40s** 198ms/step - accuracy: 0.9892 - loss: 0.0233 - val\_accuracy: 0.9869 - val\_loss: 0.0373

├─────────────────────┼───────────────────┼────────────┼───────────────────┤ Epoch 10/10

│ conv2d\_61 (Conv2D) │ (None, 29, 29, │ 22,592 │ max\_pooling2d\_15… │

**105/105** ━━━━━━━━━━━━━━━━━━━━ **41s** 195ms/step - accuracy: 0.9866 - loss: 0.0361 - val\_accuracy: 0.9845 - val\_loss: 0.0408

│ │ 64) │ │ │ ├─────────────────────┼───────────────────┼────────────┼───────────────────┤

│ concatenate\_9 │ (None, 29, 29, │ 0 │ conv2d\_56[0][0], │

train\_loss = Google\_history.history['loss']

│ (Concatenate) │ 368) │ │ conv2d\_58[0][0], │ train\_accuracy = Google\_history.history│ │ │ │ conv2d\_60[['accuracy'] 0][0], │ test\_loss = Google\_history.history│ │ │ │ conv2d\_61[['val\_loss'] 0][0] │ test\_accuracy = Google\_history.history├─────────────────────┼───────────────────┼────────────┼───────────────────┤['val\_accuracy'] epochs = │ conv2d\_63 (range(1, lenConv2D(train\_loss) │ (None)+1, )29, 29, │ 82,656 │ concatenate\_9[0]… │

│ │ 224) │ │ │

├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ conv2d\_65 (Conv2D) │ (None, 29, 29, │ 23,616 │ concatenate\_9[0]… │

│ │ 64) │ │ │

plt.figure(figsize=(8, 4))

├─────────────────────┼───────────────────┼────────────┼───────────────────┤

plt.plot(epochs, train\_loss, 'b', label='Training Loss')

│ max\_pooling2d\_16 │ (None, 29, 29, │ 0 │ concatenate\_9[0]… │ plt.plot│ ((MaxPooling2Depochs, test\_loss) │ , 'r'368, label=) │ │ │'Testing Loss') plt.title├─────────────────────┼───────────────────┼────────────┼───────────────────┤('Training and Testing Loss') plt.xlabel│ conv2d\_62 (('Epochs'Conv2D) ) │ (None, 29, 29, │ 59,040 │ concatenate\_9[0]… │ plt.ylabel│ │ ('Loss') 160) │ │ │ plt.ylim├─────────────────────┼───────────────────┼────────────┼───────────────────┤(0,10) plt.legend│ conv2d\_64 (() Conv2D) │ (None, 29, 29, │ 129,088 │ conv2d\_63[0][0] │ plt.grid│ │ (True) 64) │ │ │

├─────────────────────┼───────────────────┼────────────┼───────────────────┤

plt.show()

│ conv2d\_66 (Conv2D) │ (None, 29, 29, │ 102,464 │ conv2d\_65[0][0] │ │ │ 64) │ │ │ ├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ conv2d\_67 (Conv2D) │ (None, 29, 29, │ 47,232 │ max\_pooling2d\_16… │ │ │ 128) │ │ │

├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ concatenate\_10 │ (None, 29, 29, │ 0 │ conv2d\_62[0][0], │ │ (Concatenate) │ 416) │ │ conv2d\_64[0][0], │ │ │ │ │ conv2d\_66[0][0], │

│ │ │ │ conv2d\_67[0][0] │ ├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ max\_pooling2d\_17 │ (None, 15, 15, │ 0 │ concatenate\_10[0… │ │ (MaxPooling2D) │ 416) │ │ │ ├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ conv2d\_69 (Conv2D) │ (None, 15, 15, │ 106,752 │ max\_pooling2d\_17… │ │ │ 256) │ │ │

├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ conv2d\_71 (Conv2D) │ (None, 15, 15, │ 26,688 │ max\_pooling2d\_17… │ │ │ 64) │ │ │ ├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ max\_pooling2d\_18 │ (None, 15, 15, │ 0 │ max\_pooling2d\_17… │ │ (MaxPooling2D) │ 416) │ │ │ ├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ conv2d\_68 (Conv2D) │ (None, 15, 15, │ 53,376 │ max\_pooling2d\_17… │ │ │ 128) │ │ │

├─────────────────────┼───────────────────┼────────────┼───────────────────┤

│ conv2d\_70 (Conv2D) │ (None, 15, 15, │ 147,520 │ conv2d\_69[0][0] │ plt.plot│ │ (epochs, train\_accuracy64,) │ │ │'b', label='Training Accuracy') plt.plot├─────────────────────┼───────────────────┼────────────┼───────────────────┤(epochs, test\_accuracy, 'r', label='Testing Accuracy') plt.title│ conv2d\_72 (('Training and Testing Accuracy'Conv2D) │ (None, 15, 15), │ 102,464 │ conv2d\_71[0][0] │ plt.xlabel│ │ ('Epochs') 64) │ │ │ plt.ylabel├─────────────────────┼───────────────────┼────────────┼───────────────────┤('Accuracy') plt.ylim│ conv2d\_73 ((0,1) Conv2D) │ (None, 15, 15, │ 53,376 │ max\_pooling2d\_18… │ plt.legend│ │ () 128) │ │ │

├─────────────────────┼───────────────────┼────────────┼───────────────────┤

plt.grid(True)

│ concatenate\_11 │ (None, 15, 15, │ 0 │ conv2d\_68[0][0], │

plt.show()

│ (Concatenate) │ 384) │ │ conv2d\_70[0][0], │ │ │ │ │ conv2d\_72[0][0], │

│ │ │ │ conv2d\_73[0][0] │ ├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ conv2d\_75 (Conv2D) │ (None, 15, 15, │ 110,880 │ concatenate\_11[0… │ │ │ 288) │ │ │

├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ conv2d\_77 (Conv2D) │ (None, 15, 15, │ 24,640 │ concatenate\_11[0… │ │ │ 64) │ │ │ ├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ max\_pooling2d\_19 │ (None, 15, 15, │ 0 │ concatenate\_11[0… │ │ (MaxPooling2D) │ 384) │ │ │ ├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ conv2d\_74 (Conv2D) │ (None, 15, 15, │ 43,120 │ concatenate\_11[0… │ │ │ 112) │ │ │

├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ conv2d\_76 (Conv2D) │ (None, 15, 15, │ 165,952 │ conv2d\_75[0][0] │ │ │ 64) │ │ │ ├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ conv2d\_78 (Conv2D) │ (None, 15, 15, │ 102,464 │ conv2d\_77[0][0] │ │ │ 64) │ │ │ ├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ conv2d\_79 (Conv2D) │ (None, 15, 15, │ 49,280 │ max\_pooling2d\_19… │ │ │ 128) │ │ │

├─────────────────────┼───────────────────┼────────────┼───────────────────┤ │ concatenate\_12 │ (None, 15, 15, │ 0 │ conv2d\_74[0][0], │ │ (Concatenate) │ 368) │ │ conv2d\_76[0][0], │ │ │ │ │ conv2d\_78[0][0], │

│ │ │ │ conv2d\_79[0][0] │ ├─────────────────────┼───────────────────┼────────────┼───────────────────┤

│ conv2d\_81 (Conv2D) │ (None, 15, 15, │ 118,080 │ concatenate\_12[0… │

y\_pred = GoogleNet.predict(Google\_X\_test)

│ │ 320) │ │ │ ├─────────────────────┼───────────────────┼────────────┼───────────────────┤ y\_pred\_labels = np.argmax│ conv2d\_83 (Conv2D)( │ y\_pred(None, axis=, 15, 1)15, │ 47,232 │ concatenate\_12[0… │ y\_true\_labels = np.argmax│ │ (Google\_Y\_test128) │ │ │, axis=1)

├─────────────────────┼───────────────────┼────────────┼───────────────────┤ cm = confusion\_matrix│ max\_p oling2d\_20 │ ((y\_true\_labelsNone, , y\_pred\_labels15, 15, │ ) 0 │ concatenate\_12[0… │

│ (MaxPooling2D) │ 368) │ │ │ plt.figure├─────────────────────┼───────────────────┼────────────┼───────────────────┤(figsize=(10, 8))

│ conv2d 80 (Conv2D) │ (None 15 15 │ 94 464 │ concatenate 12[0 │

│ conv2d\_80 (Conv2D) │ (None, 15, 15, │ 94,464 │ concatenate\_12[0… │

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

│ │ 256) │ │ │

plt.title('Confusion Matrix')

├─────────────────────┼───────────────────┼────────────┼───────────────────┤ plt.colorbar│ conv2d\_82 (() Conv2D) │ (None, 15, 15, │ 368,768 │ conv2d\_81[0][0] │ tick\_marks = np.arange│ │ (len(labels128))) │ │ │ plt.xticks├─────────────────────┼───────────────────┼────────────┼───────────────────┤(tick\_marks, labels, rotation=45) plt.yticks│ onv2d\_84 ((tick\_marksConv2D, labels) │ )(None, 15, 15, │ 409,728 │ conv2d\_83[0][0] │ plt.xlabel│ │ ('Predicted Label')128) │ │ │ plt.ylabel├─────────────────────┼───────────────────┼────────────┼───────────────────┤('True Label') │ conv2d\_85 (Conv2D) │ (None, 15, 15, │ 47,232 │ max\_pooling2d\_20… │

│ │ 128) │ │ │

thresh = cm.max() / 2.

├─────────────────────┼───────────────────┼────────────┼───────────────────┤ for i │ concatenate\_13 │ (in range(cm.shape[0]): None, 15, 15, │ 0 │ conv2d\_80[0][0], │  for│ ( j Concatenatein range(cm.shape) │ [1]):640) │ │ conv2d\_82[0][0], │

plt.text│ │ │ │ conv2d\_84[(j, i, format(cm[i, j], 'd'), ha="center", va="center", color=0][0], │"white" if cm[i, j] > thresh else "black")

│ │ │ │ conv2d\_85[0][0] │ plt.tight\_layout├─────────────────────┼───────────────────┼────────────┼───────────────────┤()

│ max\_pooling2d\_21 │ (

None

,

8

,

8

,

640

│

)

0

│ concatenate\_13[

0

… │

│ (

MaxPooling2D

)

│ │ │ │

├─────────────────────┼───────────────────┼────────────┼───────────────────┤

│ average\_pooling2d\_1 │ (

None

,

2

,

2

,

640

)

│

0

│ max\_pooling2d\_21… │

│ (

AveragePooling2D

)

│ │ │ │

**27**

**/**

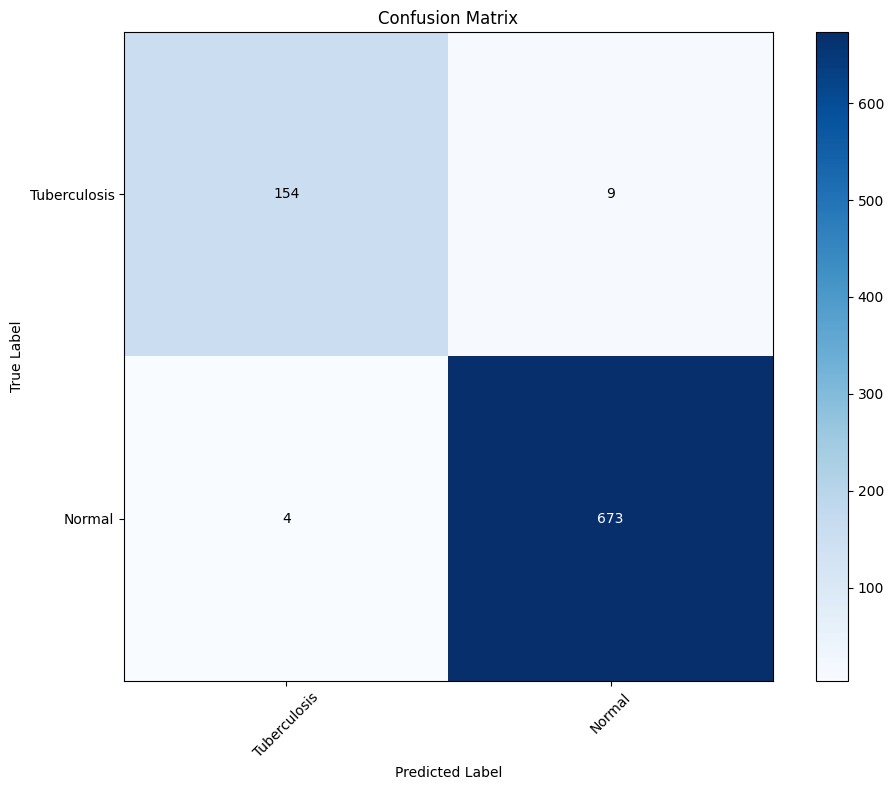
**27**

━━━━━━━━━━━━━━━━━━━━

**8**

**s**

211ms/step



plt.show

()

y\_pred = GoogleNet.predict(Google\_X\_train)

y\_pred\_labels = np.argmax(y\_pred, axis=1) y\_true\_labels = np.argmax(Google\_Y\_train, axis=1) cm = confusion\_matrix(y\_true\_labels, y\_pred\_labels)

plt.figure(figsize=(10, 8)) plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues) plt.title('Confusion Matrix') plt.colorbar() tick\_marks = np.arange(len(labels)) plt.xticks(tick\_marks, labels, rotation=45) plt.yticks(tick\_marks, labels) plt.xlabel('Predicted Label') plt.ylabel('True Label')

thresh = cm.max() / 2. for i in range(cm.shape[0]): for j in range(cm.shape[1]):

plt.text(j, i, format(cm[i, j], 'd'), ha="center", va="center", color="white" if cm[i, j] > thresh else "black") plt.tight\_layout()

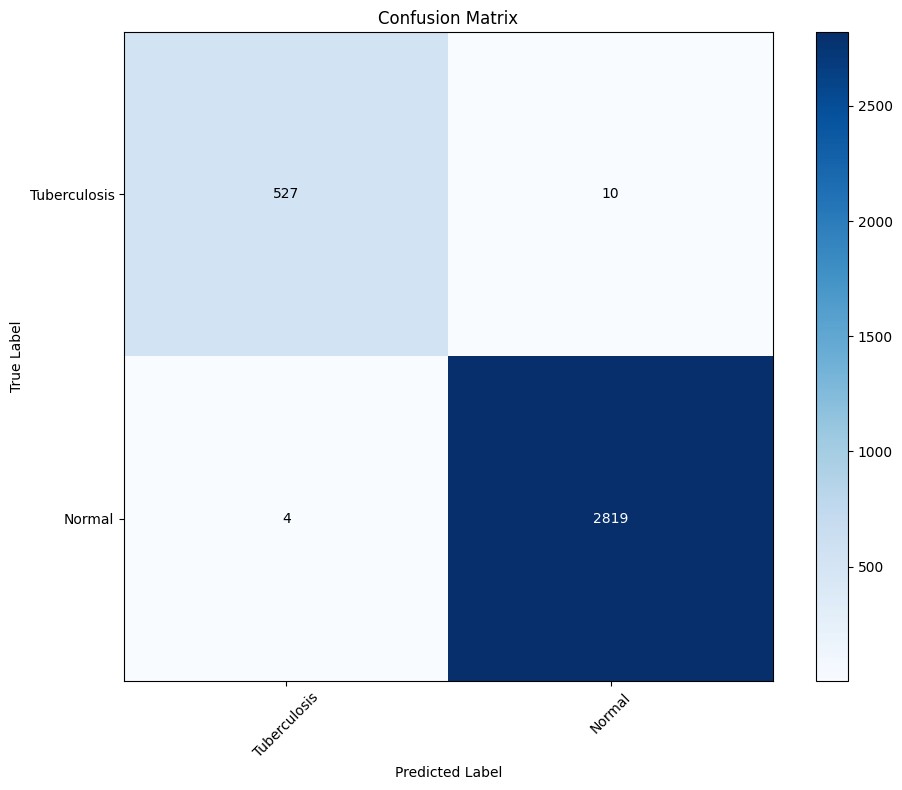
**105/105**

━━━━━━━━━━━━━━━━━━━━

**6**

**s**

57ms/step



plt.show

()

from tensorflow.keras.preprocessing import image def Google\_pred(path): img\_path = path img = image.load\_img(img\_path, target\_size=(227, 227)) img\_array = image.img\_to\_array(img) img\_tensor = np.expand\_dims(img\_array, axis=0) img\_tensor = img\_tensor / 255.0 prediction = GoogleNet.predict(img\_tensor) predicted\_class\_index = np.argmax(prediction) predicted\_class\_name = labels[predicted\_class\_index] imge=cv2.imread(path) cv2\_imshow(imge)

print('Predicted class:', predicted\_class\_name)

Google\_pred('/content/TB\_Chest\_Radiography\_Database/Normal/Normal-1064.png')

**1**

**/**

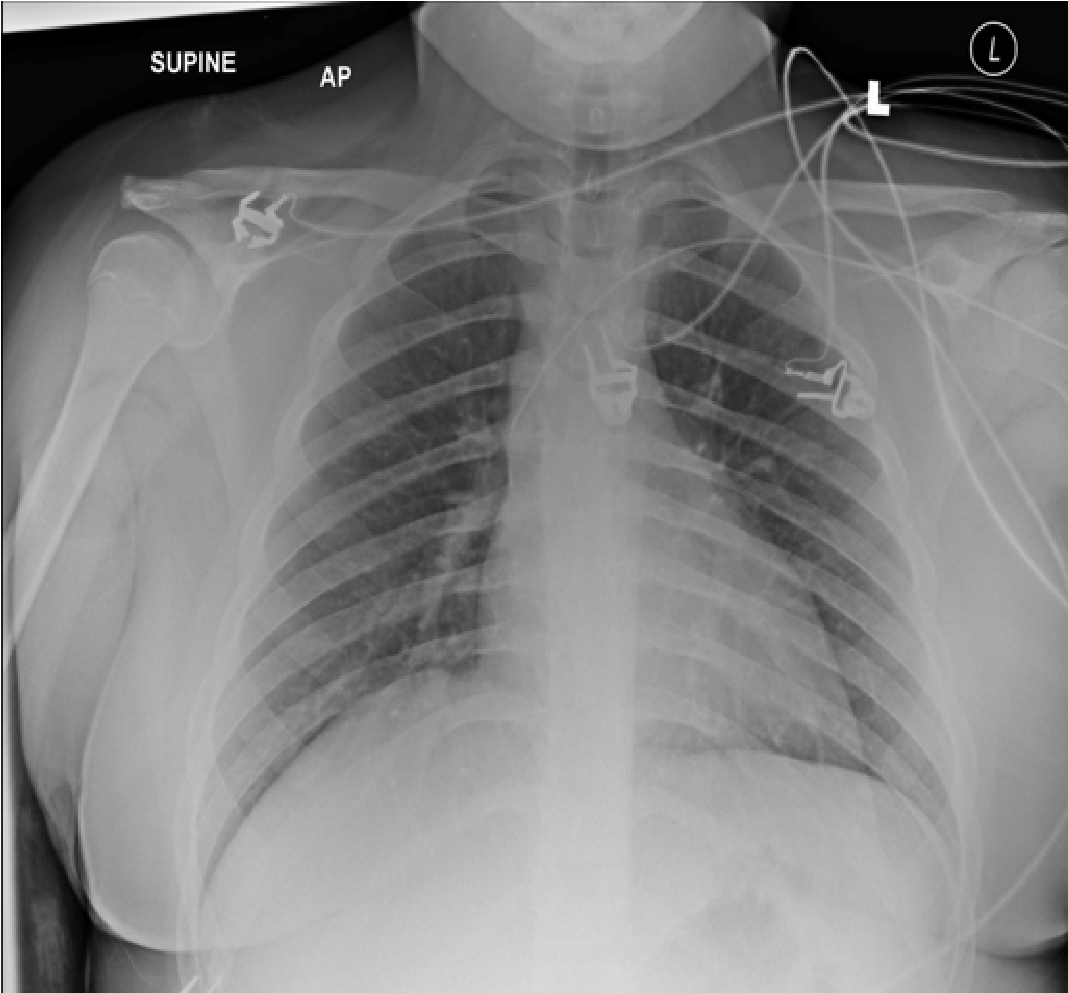
**1**

━━━━━━━━━━━━━━━━━━━━

**0**

**s**

83ms/step



Requirement already satisfied: kaggle in /usr/local/lib/python3.11/dist-packages (1.7.4.2)

!

pip install kaggle

Requirement already satisfied: bleach in /usr/local/lib/python3.11/dist-packages (from kaggle) (6.2.0)

Requirement already satisfied: certifi>=14.05.14 in /usr/local/lib/python3.11/dist-packages (from kaggle) (2025.4.26)

Requirement already satisfied: charset-normalizer in /usr/local/lib/python3.11/dist-packages (from kaggle) (3.4.1)

Requirement already satisfied: idna in /usr/local/lib/python3.11/dist-packages (from kaggle) (3.10)

Requirement already satisfied: protobuf in /usr/local/lib/python3.11/dist-packages (from kaggle) (5.29.4)

Requirement already satisfied: python-dateutil>=2.5.3 in /usr/local/lib/python3.11/dist-packages (from kaggle) (2.9.0.post0)

Requirement already satisfied: python-slugify in /usr/local/lib/python3.11/dist-packages (from kaggle) (8.0.4)

Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from kaggle) (2.32.3)

Requirement already satisfied: setuptools>=21.0.0 in /usr/local/lib/python3.11/dist-packages (from kaggle) (75.2.0)

Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.11/dist-packages (from kaggle) (1.17.0)

Requirement already satisfied: text-unidecode in /usr/local/lib/python3.11/dist-packages (from kaggle) (1.3)

Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from kaggle) (4.67.1)

Requirement already satisfied: urllib3>=1.15.1 in /usr/local/lib/python3.11/dist-packages (from kaggle) (2.4.0)

Requirement already satisfied: webencodings in /usr/local/lib/python3.11/dist-packages (from kaggle) (0.5.1)

import os os.makedirs('/root/.kaggle', exist\_ok=True)

!mv kaggle.json /root/.kaggle/

!chmod 600 /root/.kaggle/kaggle.json

!kaggle datasets download tawsifurrahman/tuberculosis-tb-chest-xray-dataset

Dataset URL: <https://www.kaggle.com/datasets/tawsifurrahman/tuberculosis-tb-chest-xray-dataset>License(s): copyright-authors

!unzip /content/tuberculosis-tb-chest-xray-dataset.zip

Archive: /content/tuberculosis-tb-chest-xray-dataset.zip inflating: TB\_Chest\_Radiography\_Database/Normal.metadata.xlsx inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-10.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-100.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1000.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1001.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1002.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1003.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1004.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1005.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1006.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1007.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1008.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1009.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-101.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1010.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1011.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1012.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1013.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1014.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1015.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1016.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1017.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1018.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1019.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-102.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1020.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1021.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1022.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1023.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1024.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1025.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1026.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1027.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1028.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1029.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-103.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1030.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1031.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1032.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1033.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1034.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1035.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1036.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1037.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1038.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1039.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-104.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1040.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1041.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1042.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1043.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1044.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1045.png

**

**

**

inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1046.png

inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1047.png

inflating:

TB

Chest

Radiography

Database/Normal/Normal-1048

png

import

numpy

as

np

import

pandas

as

ps

import

matplotlib.pyplot

as

plt

import

os

import

glob

import

seaborn

as

sns

import

cv2

import

io

import

tensorflow

as

tf

from

sklearn.model\_selection

import

train\_test\_split

from

tqdm

import

tqdm

import

keras

from

keras.preprocessing

import

image

from

tensorflow.keras.preprocessing.image

import

ImageDataGenerator

from

PIL

import

Image

from

sklearn.utils

import

shuffle

from

keras

import

layers

,

models

,

optimizers

import

ipywidgets

as

widgets

from

keras.utils

import

to\_categorical

from

keras.models

import

Sequential

from

keras.layers

import

Dense

,

Activation

,

Dropout

,

Flatten

,

Conv2D

,

MaxPooling2D

,

Dropout

,

BatchNormalization

,

Concatenate

,

AveragePooling

from

sklearn.metrics

import

classification\_report

,

confusion\_matrix

from

keras.models

import

Sequential

from

keras.regularizers

import

l2

from

keras.callbacks

import

EarlyStopping

from

google.colab.patches

import

cv2\_imshow

labels=

[

"Tuberculosis"

,

"Normal"

]

X=

[]

Y=

[]

image\_size=

(

227

,

227

)

for

i

in

labels

:

path=

"/content/TB\_Chest\_Radiography\_Database/"

+

i

+

"/"

print

(

path

)

fileRead=glob.glob

(

path+

"\*"

)

print

(

len

(

fileRead

))

for

j

in

fileRead

:

image=cv2.imread

(

j

)

image=cv2.resize

(

image

,

image\_size

)

X.append

(

image

)

Y.append

(

i

)

/content/TB\_Chest\_Radiography\_Database/Tuberculosis/

700

/content/TB\_Chest\_Radiography\_Database/Normal/

for i in labels: path="/content/TB\_Chest\_Radiography\_Database/"+i+"/" print(path) fileRead=glob.glob(path+"\*") print(fileRead[5])

/content/TB\_Chest\_Radiography\_Database/Tuberculosis/

/content/TB\_Chest\_Radiography\_Database/Tuberculosis/Tuberculosis-76.png

/content/TB\_Chest\_Radiography\_Database/Normal/

/content/TB\_Chest\_Radiography\_Database/Normal/Normal-2679.png

[1. 0.]

optimizer = optimizers.Adam(learning\_rate=0.0001) from keras.models import Model from keras.layers import Input, Conv2D, MaxPooling2D, AveragePooling2D, Dropout, Flatten, Dense, Concatenate

resnet\_X\_train,resnet\_X\_test,resnet\_Y\_train,resnet\_Y\_test=train\_test\_split(X,Y,test\_size=0.2,random\_state=42)

from keras.models import Sequential from keras.layers import Conv2D, MaxPooling2D, AveragePooling2D, Flatten, Dense, BatchNormalization, Activation

def resnet\_block(model, filters, strides=(1, 1)):

model.add(Conv2D(filters, (3, 3), strides=strides, padding='same')) model.add(BatchNormalization()) model.add(Activation('relu')) model.add(Conv2D(filters, (3, 3), padding='same')) model.add(BatchNormalization()) model.add(Activation('relu')) return model

def resnet18\_model(input\_shape=(227, 227, 3)): model = Sequential()

# Initial Convolution model.add(Conv2D(64, (7, 7), strides=(2, 2), padding='same', input\_shape=input\_shape)) model.add(BatchNormalization()) model.add(Activation('relu')) model.add(MaxPooling2D((3, 3), strides=(2, 2), padding='same'))

# ResNet Blocks model = resnet\_block(model, filters=64) model.add(MaxPooling2D((2, 2), strides=(2, 2), padding='same'))

model = resnet\_block(model, filters=128, strides=(2, 2)) model.add(MaxPooling2D((2, 2), strides=(2, 2), padding='same'))

model = resnet\_block(model, filters=256, strides=(2, 2)) model.add(MaxPooling2D((2, 2), strides=(2, 2), padding='same'))

model = resnet\_block(model, filters=512, strides=(2, 2)) model.add(AveragePooling2D((1, 1))) # Average pooling instead of max pooling model.add(Flatten())

# Classification Head model.add(Dense(len(labels), activation='softmax')) return model

# Create the ResNet-18 model resnet = resnet18\_model()

/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base\_conv.py:107: UserWarning: Do not pass an `input\_shape`/` super().\_\_init\_\_(activity\_regularizer=activity\_regularizer, \*\*kwargs)

* *

resnet.compile(loss='categorical\_crossentropy',optimizer=optimizer,metrics=['accuracy']) early\_stopping=EarlyStopping(monitor='val\_loss',patience=3)

resnet.summary()

**Model: "sequential"**

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓ ┃ **Layer (type)** ┃ **Output Shape** ┃ **Param #** ┃ ┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩ │ conv2d (Conv2D) │ (None, 114, 114, 64) │ 9,472 │ ├─────────────────────────────────┼────────────────────────┼───────────────┤ │ batch\_normalization │ (None, 114, 114, 64) │ 256 │ │ (BatchNormalization) │ │ │ ├─────────────────────────────────┼────────────────────────┼───────────────┤ │ activation (Activation) │ (None, 114, 114, 64) │ 0 │ ├─────────────────────────────────┼────────────────────────┼───────────────┤ │ max\_pooling2d (MaxPooling2D) │ (None, 57, 57, 64) │ 0 │ ├─────────────────────────────────┼────────────────────────┼───────────────┤ │ conv2d\_1 (Conv2D) │ (None, 57, 57, 64) │ 36,928 │ ├─────────────────────────────────┼────────────────────────┼───────────────┤ │ batch\_normalization\_1 │ (None, 57, 57, 64) │ 256 │ │ (BatchNormalization) │ │ │ ├─────────────────────────────────┼────────────────────────┼───────────────┤ │ activation\_1 (Activation) │ (None, 57, 57, 64) │ 0 │ ├─────────────────────────────────┼────────────────────────┼───────────────┤ │ conv2d\_2 (Conv2D) │ (None, 57, 57, 64) │ 36,928 │ ├─────────────────────────────────┼────────────────────────┼───────────────┤ │ batch\_normalization\_2 │ (None, 57, 57, 64) │ 256 │ │ (BatchNormalization) │ │ │ ├─────────────────────────────────┼────────────────────────┼───────────────┤ │ activation\_2 (Activation) │ (None, 57, 57, 64) │ 0 │ ├─────────────────────────────────┼────────────────────────┼───────────────┤ │ max\_pooling2d\_1 (MaxPooling2D) │ (None, 29, 29, 64) │ 0 │ ├─────────────────────────────────┼────────────────────────┼───────────────┤ │ conv2d\_3 (Conv2D) │ (None, 15, 15, 128) │ 73,856 │ ├─────────────────────────────────┼────────────────────────┼───────────────┤ │ batch\_normalization\_3 │ (None, 15, 15, 128) │ 512 │ │ (BatchNormalization) │ │ │ ├─────────────────────────────────┼────────────────────────┼───────────────┤ │ activation\_3 (Activation) │ (None, 15, 15, 128) │ 0 │ ├─────────────────────────────────┼────────────────────────┼───────────────┤ │ conv2d\_4 (Conv2D) │ (None, 15, 15, 128) │ 147,584 │ ├─────────────────────────────────┼────────────────────────┼───────────────┤ │ batch\_normalization\_4 │ (None, 15, 15, 128) │ 512 │ │ (BatchNormalization) │ │ │ ├─────────────────────────────────┼────────────────────────┼───────────────┤ │ activation\_4 (Activation) │ (None, 15, 15, 128) │ 0 │ ├─────────────────────────────────┼────────────────────────┼───────────────┤ │ max\_pooling2d\_2 (MaxPooling2D) │ (None, 8, 8, 128) │ 0 │ ├─────────────────────────────────┼────────────────────────┼───────────────┤ │ conv2d\_5 (Conv2D) │ (None, 4, 4, 256) │ 295,168 │ ├─────────────────────────────────┼────────────────────────┼───────────────┤ │ batch\_normalization\_5 │ (None, 4, 4, 256) │ 1,024 │ │ (BatchNormalization) │ │ │ ├─────────────────────────────────┼────────────────────────┼───────────────┤ │ activation\_5 (Activation) │ (None, 4, 4, 256) │ 0 │ ├─────────────────────────────────┼────────────────────────┼───────────────┤ │ conv2d\_6 (Conv2D) │ (None, 4, 4, 256) │ 590,080 │ ├─────────────────────────────────┼────────────────────────┼───────────────┤ │ batch\_normalization\_6 │ (None, 4, 4, 256) │ 1,024 │ │ (BatchNormalization) │ │ │ ├─────────────────────────────────┼────────────────────────┼───────────────┤ │ activation\_6 (Activation) │ (None, 4, 4, 256) │ 0 │ ├─────────────────────────────────┼────────────────────────┼───────────────┤ │ max\_pooling2d\_3 (MaxPooling2D) │ (None, 2, 2, 256) │ 0 │ ├─────────────────────────────────┼────────────────────────┼───────────────┤ │ conv2d\_7 (Conv2D) │ (None, 1, 1, 512) │ 1,180,160 │ ├─────────────────────────────────┼────────────────────────┼───────────────┤ │ batch\_normalization\_7 │ (None, 1, 1, 512) │ 2,048 │ │ (BatchNormalization) │ │ │ ├─────────────────────────────────┼────────────────────────┼───────────────┤ │ activation\_7 (Activation) │ (None, 1, 1, 512) │ 0 │ ├─────────────────────────────────┼────────────────────────┼───────────────┤ │ conv2d\_8 (Conv2D) │ (None, 1, 1, 512) │ 2,359,808 │ ├─────────────────────────────────┼────────────────────────┼───────────────┤ │ batch\_normalization\_8 │ (None, 1, 1, 512) │ 2,048 │ │ (BatchNormalization) │ │ │ ├─────────────────────────────────┼────────────────────────┼───────────────┤ │ activation\_8 (Activation) │ (None, 1, 1, 512) │ 0 │ ├─────────────────────────────────┼────────────────────────┼───────────────┤ │ average\_pooling2d │ (None, 1, 1, 512) │ 0 │ │ (AveragePooling2D) │ │ │

├─────────────────────────────────┼────────────────────────┼───────────────┤ │ flatten (Flatten) │ (None, 512) │ 0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

**

**

resnet\_history=resnet.fit(resnet\_X\_train,resnet\_Y\_train, validation\_data=(resnet\_X\_test,resnet\_Y\_test),epochs=10,batch\_size=32,verbose=1

Epoch 1/10

**105/105** ━━━━━━━━━━━━━━━━━━━━ **21s** 63ms/step - accuracy: 0.8921 - loss: 0.2674 - val\_accuracy: 0.9036 - val\_loss: 0.6169

Epoch 2/10

**105/105** ━━━━━━━━━━━━━━━━━━━━ **4s** 42ms/step - accuracy: 0.9907 - loss: 0.0384 - val\_accuracy: 0.2857 - val\_loss: 0.9307

Epoch 3/10

**105/105** ━━━━━━━━━━━━━━━━━━━━ **4s** 41ms/step - accuracy: 0.9936 - loss: 0.0233 - val\_accuracy: 0.8393 - val\_loss: 0.3646

Epoch 4/10

**105/105** ━━━━━━━━━━━━━━━━━━━━ **5s** 44ms/step - accuracy: 0.9954 - loss: 0.0186 - val\_accuracy: 0.9679 - val\_loss: 0.0992

Epoch 5/10

**105/105** ━━━━━━━━━━━━━━━━━━━━ **4s** 41ms/step - accuracy: 0.9903 - loss: 0.0258 - val\_accuracy: 0.9333 - val\_loss: 0.2317

Epoch 6/10

**105/105** ━━━━━━━━━━━━━━━━━━━━ **4s** 41ms/step - accuracy: 0.9985 - loss: 0.0078 - val\_accuracy: 0.9798 - val\_loss: 0.0634

Epoch 7/10

**105/105** ━━━━━━━━━━━━━━━━━━━━ **5s** 45ms/step - accuracy: 0.9980 - loss: 0.0085 - val\_accuracy: 0.9881 - val\_loss: 0.0448

Epoch 8/10

**105/105** ━━━━━━━━━━━━━━━━━━━━ **4s** 41ms/step - accuracy: 0.9957 - loss: 0.0132 - val\_accuracy: 0.9905 - val\_loss: 0.0402

Epoch 9/10

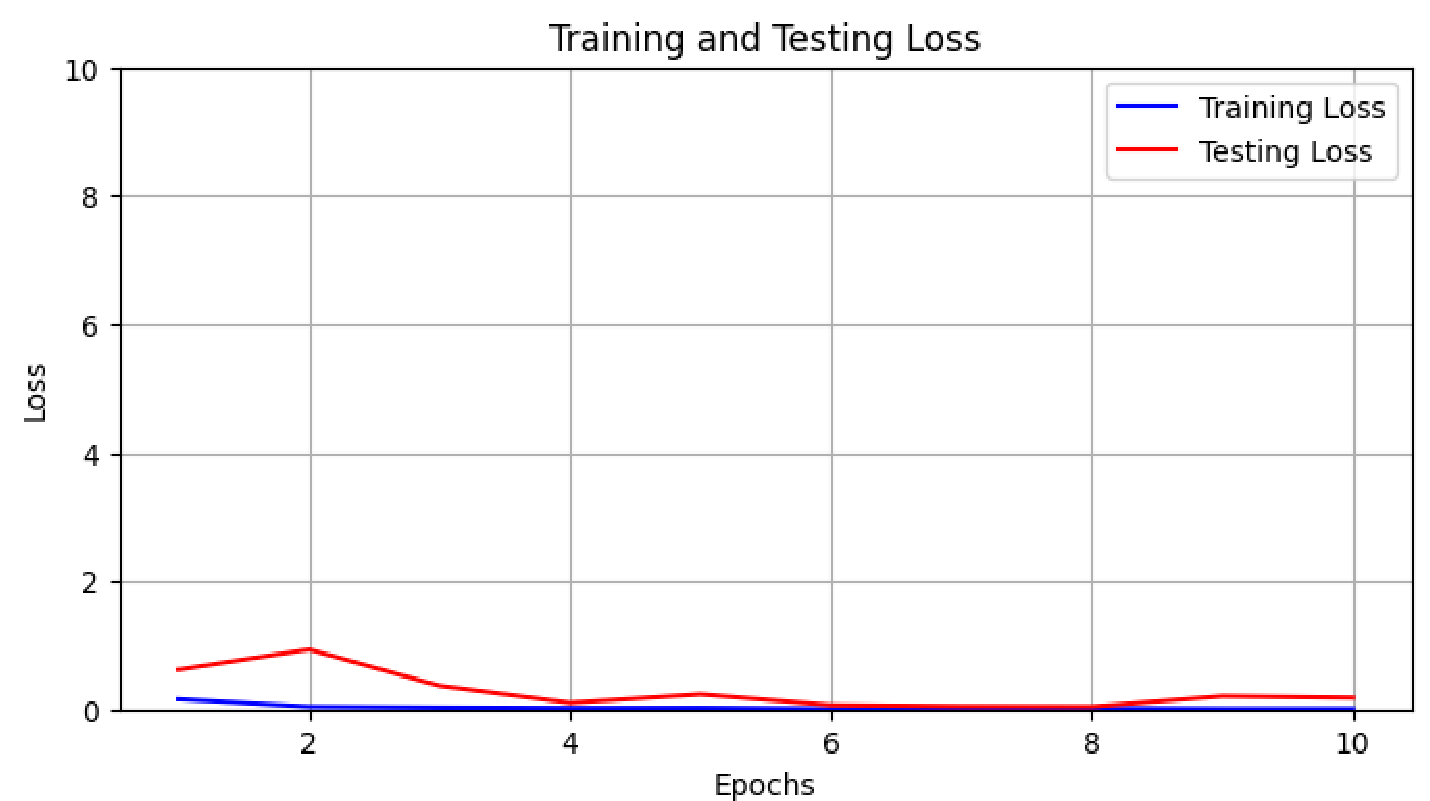
**105/105** ━━━━━━━━━━━━━━━━━━━━ **5s** 41ms/step - accuracy: 0.9991 - loss: 0.0048 - val\_accuracy: 0.9405 - val\_loss: 0.2026

Epoch 10/10

**105/105** ━━━━━━━━━━━━━━━━━━━━ **5s** 42ms/step - accuracy: 0.9978 - loss: 0.0093 - val\_accuracy: 0.9536 - val\_loss: 0.1786

train\_loss = resnet\_history.history['loss'] train\_accuracy = resnet\_history.history['accuracy'] test\_loss = resnet\_history.history['val\_loss'] test\_accuracy = resnet\_history.history['val\_accuracy'] epochs = range(1, len(train\_loss)+1)

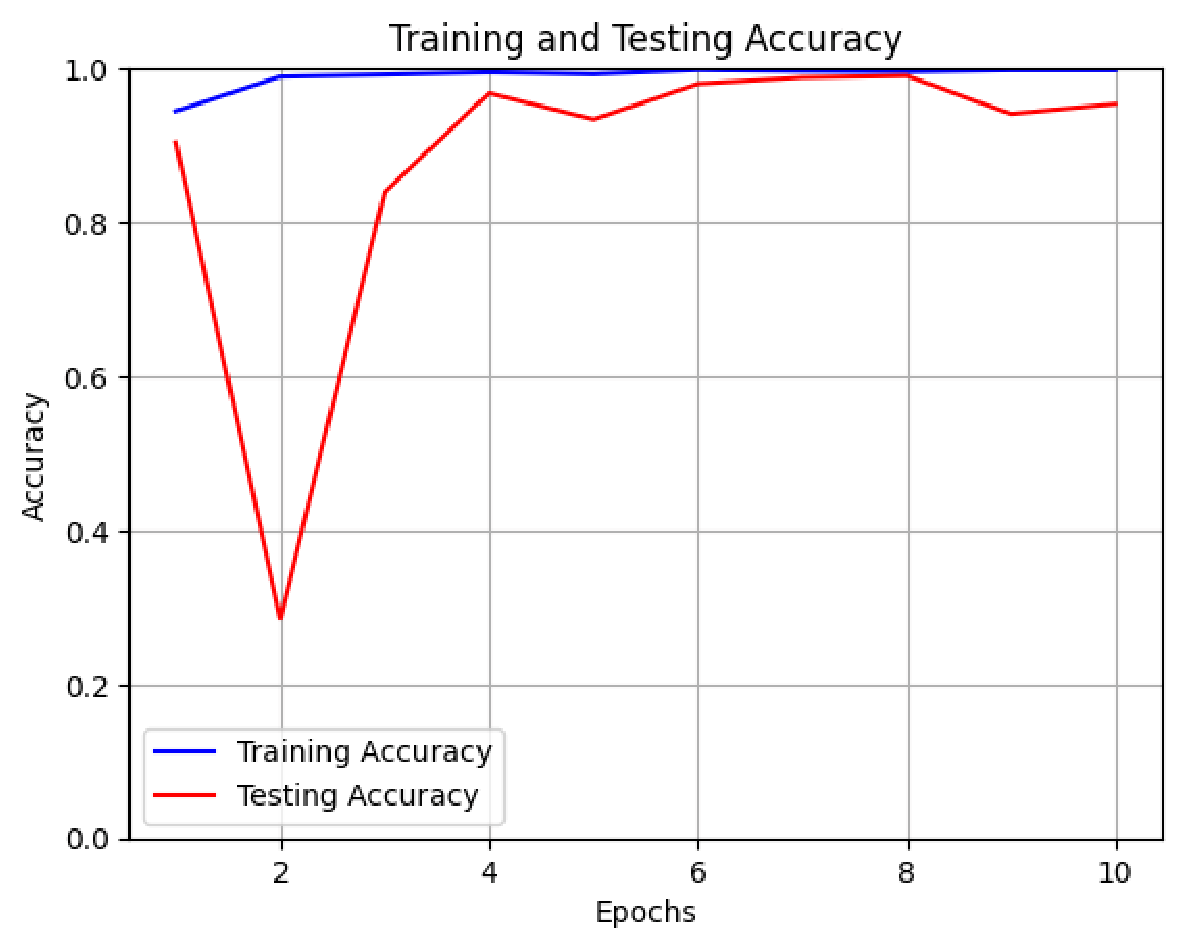
plt.figure(figsize=(8, 4)) plt.plot(epochs, train\_loss, 'b', label='Training Loss') plt.plot(epochs, test\_loss, 'r', label='Testing Loss') plt.title('Training and Testing Loss') plt.xlabel('Epochs') plt.ylabel('Loss') plt.ylim(0,10) plt.legend() plt.grid(True) plt.show()



**

**

plt.plot(epochs, train\_accuracy, 'b', label='Training Accuracy') plt.plot(epochs, test\_accuracy, 'r', label='Testing Accuracy') plt.title('Training and Testing Accuracy') plt.xlabel('Epochs') plt.ylabel('Accuracy') plt.ylim(0,1) plt.legend() plt.grid(True) plt.show()



**

**

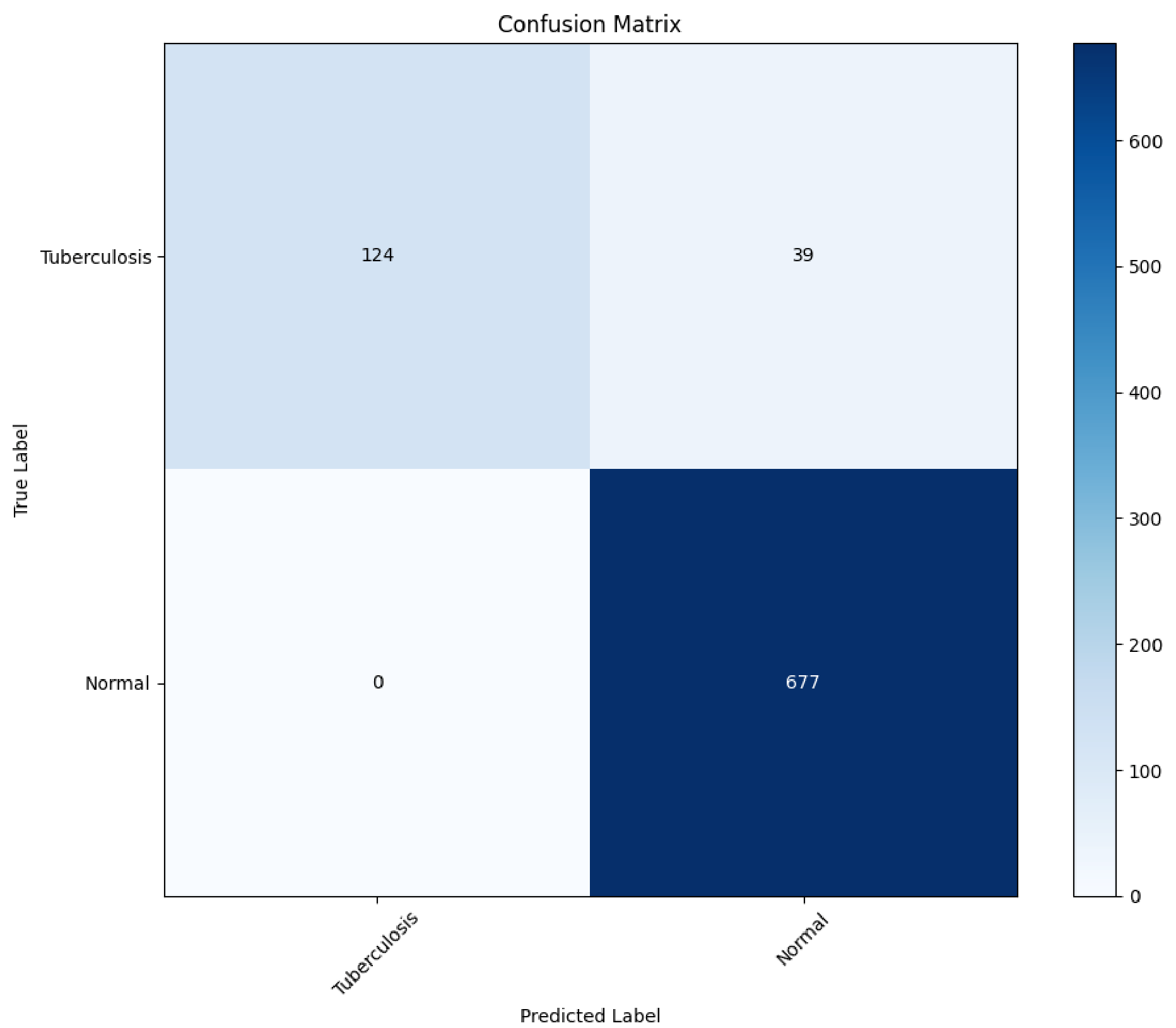
y\_pred = resnet.predict(resnet\_X\_test)

y\_pred\_labels = np.argmax(y\_pred, axis=1) y\_true\_labels = np.argmax(resnet\_Y\_test, axis=1) cm = confusion\_matrix(y\_true\_labels, y\_pred\_labels)

plt.figure(figsize=(10, 8)) plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues) plt.title('Confusion Matrix') plt.colorbar() tick\_marks = np.arange(len(labels)) plt.xticks(tick\_marks, labels, rotation=45) plt.yticks(tick\_marks, labels) plt.xlabel('Predicted Label') plt.ylabel('True Label')

thresh = cm.max() / 2. for i in range(cm.shape[0]): for j in range(cm.shape[1]): plt.text(j, i, format(cm[i, j], 'd'), ha="center", va="center", color="white" if cm[i, j] > thresh else "black")

**27/27** ━━━━━━━━━━━━━━━━━━━━ **2s** 62ms/step



**

**

y\_pred = resnet.predict(resnet\_X\_train)

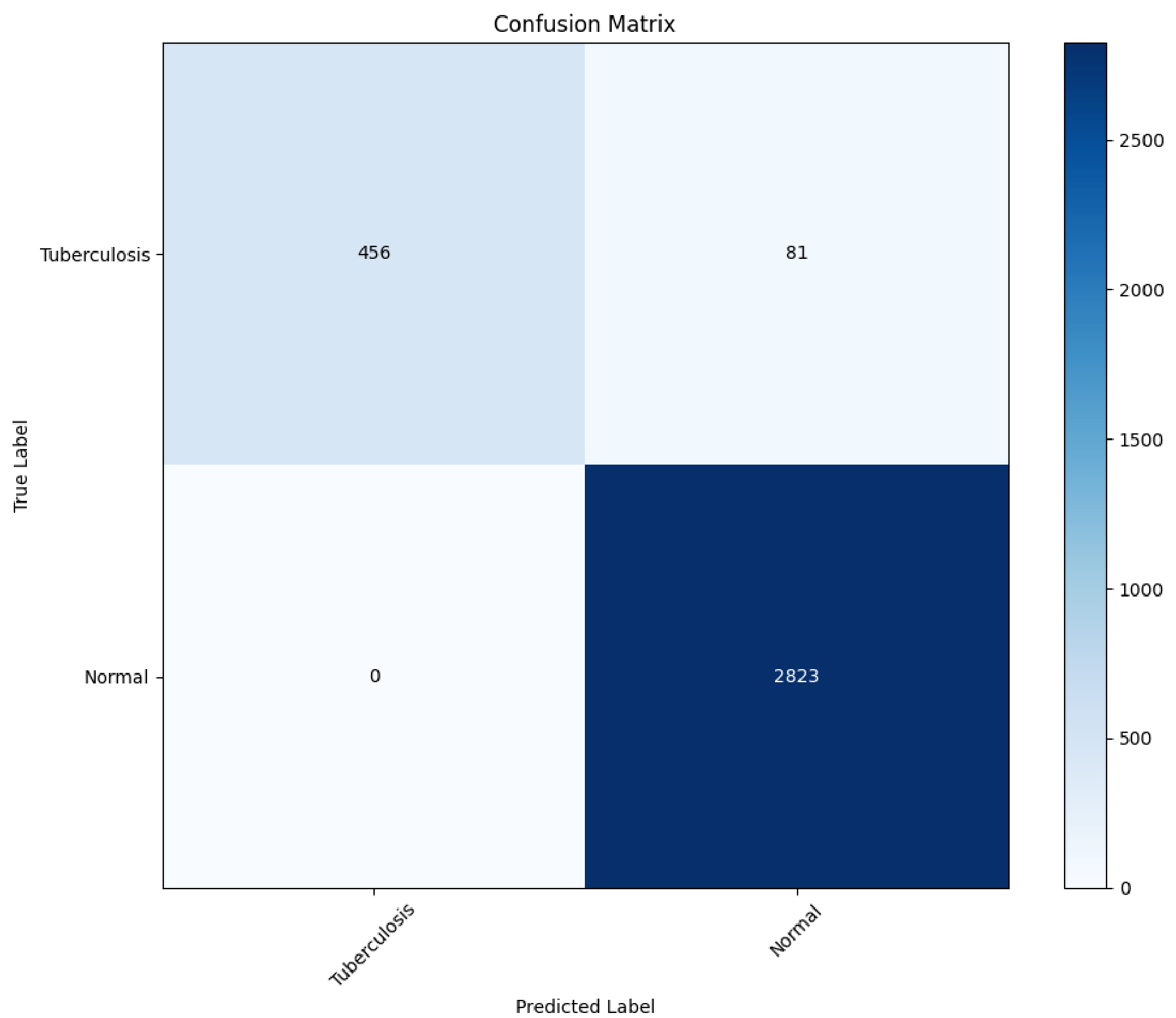
y\_pred\_labels = np.argmax(y\_pred, axis=1) y\_true\_labels = np.argmax(resnet\_Y\_train, axis=1) cm = confusion\_matrix(y\_true\_labels, y\_pred\_labels)

plt.figure(figsize=(10, 8)) plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues) plt.title('Confusion Matrix') plt.colorbar() tick\_marks = np.arange(len(labels)) plt.xticks(tick\_marks, labels, rotation=45) plt.yticks(tick\_marks, labels) plt.xlabel('Predicted Label') plt.ylabel('True Label')

thresh = cm.max() / 2. for i in range(cm.shape[0]): for j in range(cm.shape[1]): plt.text(j, i, format(cm[i, j], 'd'), ha="center", va="center", color="white" if cm[i, j] > thresh else "black")

plt.tight\_layout() plt.show()

**105/105** ━━━━━━━━━━━━━━━━━━━━ **1s** 11ms/step



**

**

from tensorflow.keras.preprocessing import image def resnet\_pred(path): img\_path = path img = image.load\_img(img\_path, target\_size=(227, 227)) img\_array = image.img\_to\_array(img) img\_tensor = np.expand\_dims(img\_array, axis=0) img\_tensor = img\_tensor / 255.0 prediction = resnet.predict(img\_tensor) predicted\_class\_index = np.argmax(prediction) predicted\_class\_name = labels[predicted\_class\_index] imge=cv2.imread(path) cv2\_imshow(imge) print('Predicted class:', predicted\_class\_name)

resnet\_pred('/content/TB\_Chest\_Radiography\_Database/Tuberculosis/Tuberculosis-102.png')

1. activation\_4
2. max\_pooling2d\_2
3. conv2d\_5
4. batch\_normalization\_5
5. activation\_5
6. conv2d\_6
7. batch\_normalization\_6
8. activation\_6
9. max\_pooling2d\_3
10. conv2d\_7
11. batch\_normalization\_7
12. activation\_7
13. conv2d\_8
14. batch\_normalization\_8
15. activation\_8
16. average\_pooling2d
17. flatten
18. dense

for layer in resnet.layers: if 'conv' in layer.name:

print(layer.name)

1

!pipinstallkaggle

Requirement already satisfied: kaggle in /usr/local/lib/python3.11/dist-packages (1.6.17)

Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.11/dist-packages (from kaggle) (1.17.0)

Requirement already satisfied: certifi>=2023.7.22 in /usr/local/lib/python3.11/dist-packages (from kaggle) (2025.1.31)

Requirement already satisfied: python-dateutil in /usr/local/lib/python3.11/dist-packages (from kaggle) (2.8.2)

Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from kaggle) (2.32.3)

Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from kaggle) (4.67.1)

Requirement already satisfied: python-slugify in /usr/local/lib/python3.11/dist-packages (from kaggle) (8.0.4)

Requirement already satisfied: urllib3 in /usr/local/lib/python3.11/dist-packages (from kaggle) (2.3.0)

Requirement already satisfied: bleach in /usr/local/lib/python3.11/dist-packages (from kaggle) (6.2.0)

Requirement already satisfied: webencodings in /usr/local/lib/python3.11/dist-packages (from bleach->kaggle) (0.5.1)

Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.11/dist-packages (from python-slugify->kaggle) (1.3)

Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->kaggle) (3.4.1)

Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->kaggle) (3.10)

**

**













format\_quote







ψ



import

os

os.makedirs

(

'

/root/.kaggle

'

,

exist\_ok=

True

)

!

mv kaggle

.

json

/root/

.

kaggle

/

!

chmod

600

/root/

.

kaggle/kaggle

.

json

!kaggle datasets download tawsifurrahman/tuberculosis-tb-chest-xray-dataset

Warning: Looks like you're using an outdated API Version, please consider updating (server 1.7.4.2 / client 1.6.17) Dataset URL: <https://www.kaggle.com/datasets/tawsifurrahman/tuberculosis-tb-chest-xray-dataset>License(s): copyright-authors

Downloading tuberculosis-tb-chest-xray-dataset.zip to /content

97% 641M/663M [00:10<00:00, 162MB/s]

100% 663M/663M [00:10<00:00, 68.8MB/s]

!unzip /content/tuberculosis-tb-chest-xray-dataset.zip

Archive: /content/tuberculosis-tb-chest-xray-dataset.zip inflating: TB\_Chest\_Radiography\_Database/Normal.metadata.xlsx inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-10.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-100.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1000.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1001.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1002.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1003.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1004.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1005.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1006.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1007.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1008.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1009.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-101.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1010.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1011.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1012.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1013.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1014.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1015.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1016.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1017.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1018.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1019.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-102.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1020.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1021.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1022.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1023.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1024.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1025.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1026.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1027.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1028.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1029.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-103.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1030.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1031.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1032.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1033.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1034.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1035.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1036.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1037.png inflating: TB\_Chest\_Radiography\_Database/Normal/Normal-1038.png

**

A screenshot of a computer

AI-generated content may be incorrect.

img\_tensor = np.expand\_dims(image, axis=0) last\_conv\_layer\_name = 'conv2d\_1' # second conv layer

grad\_model = Model( inputs=model.input, outputs=[model.get\_layer(last\_conv\_layer\_name).output, model.output]

)

with tf.GradientTape() as tape:

conv\_output, predictions = grad\_model(img\_tensor) class\_idx = tf.argmax(predictions[0]) loss = predictions[:, class\_idx] grads = tape.gradient(loss, conv\_output)[0] pooled\_grads = tf.reduce\_mean(grads, axis=(0, 1)) conv\_output = conv\_output[0]

heatmap = tf.reduce\_sum(tf.multiply(pooled\_grads, conv\_output), axis=-1) heatmap = tf.maximum(heatmap, 0) / tf.reduce\_max(heatmap) heatmap = cv2.resize(heatmap.numpy(), (image.shape[1], image.shape[0])) heatmap = np.uint8(255 \* heatmap) heatmap\_color = cv2.applyColorMap(heatmap, cv2.COLORMAP\_JET) superimposed = cv2.addWeighted((image \* 255).astype(np.uint8), 0.6, heatmap\_color, 0.4, 0)

plt.imshow(superimposed) plt.title(f"Predicted: {label\_names[class\_idx]}") plt.axis('off') plt.show()

def create\_simple\_cnn\_functional(input\_shape=(64, 64, 3), num\_classes=2): inputs = Input(shape=input\_shape)

x = Conv2D(16, (3, 3), activation='relu', name="conv1")(inputs) x = MaxPooling2D(2, 2)(x) x = Conv2D(32, (3, 3), activation='relu', name="conv2")(x) x = MaxPooling2D(2, 2)(x) x = Flatten()(x) x = Dense(64, activation='relu')(x) outputs = Dense(num\_classes, activation='softmax')(x)

model = Model(inputs, outputs) model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy']) return model

# Rebuild and retrain the model model = create\_simple\_cnn\_functional()

history = model.fit(X\_train, Y\_train, epochs=5, validation\_data=(X\_test, Y\_test))

Epoch 1/5

**105/105** ━━━━━━━━━━━━━━━━━━━━ **4s** 15ms/step - accuracy: 0.8691 - loss: 0.3486 - val\_accuracy: 0.9429 - val\_loss: 0.1587 Epoch 2/5

**105/105** ━━━━━━━━━━━━━━━━━━━━ **3s** 5ms/step - accuracy: 0.9476 - loss: 0.1526 - val\_accuracy: 0.9571 - val\_loss: 0.1339

Epoch 3/5

**105/105** ━━━━━━━━━━━━━━━━━━━━ **1s** 5ms/step - accuracy: 0.9636 - loss: 0.1104 - val\_accuracy: 0.9690 - val\_loss: 0.0830

Epoch 4/5

**105/105** ━━━━━━━━━━━━━━━━━━━━ **1s** 5ms/step - accuracy: 0.9766 - loss: 0.0729 - val\_accuracy: 0.9738 - val\_loss: 0.0745

Epoch 5/5

**105/105** ━━━━━━━━━━━━━━━━━━━━ **1s** 6ms/step - accuracy: 0.9824 - loss: 0.0541 - val\_accuracy: 0.9750 - val\_loss: 0.0688

def gradcam\_simple(model, image, label\_names): img\_tensor = np.expand\_dims(image, axis=0) last\_conv\_layer\_name = 'conv2' # Use the correct name from above

grad\_model = Model( inputs=model.input, outputs=[model.get\_layer(last\_conv\_layer\_name).output, model.output]

)

with tf.GradientTape() as tape:

conv\_outputs, predictions = grad\_model(img\_tensor) class\_idx = tf.argmax(predictions[0]) loss = predictions[:, class\_idx]

grads = tape.gradient(loss, conv\_outputs)[0] pooled\_grads = tf.reduce\_mean(grads, axis=(0, 1)) conv\_outputs = conv\_outputs[0] heatmap = tf.reduce\_sum(pooled\_grads \* conv\_outputs, axis=-1) heatmap = tf.maximum(heatmap, 0) / tf.reduce\_max(heatmap) heatmap = cv2.resize(heatmap.numpy(), (image.shape[1], image.shape[0])) heatmap = np.uint8(255 \* heatmap) heatmap\_color = cv2.applyColorMap(heatmap, cv2.COLORMAP\_JET) superimposed = cv2.addWeighted((image \* 255).astype(np.uint8), 0.6, heatmap\_color, 0.4, 0)

plt.imshow(superimposed) plt.title(f"Predicted: {label\_names[class\_idx]}") plt.axis('off') plt.show()

Start coding or generate with AI.

Start coding or generate with AI.

from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense, Dropout, Add, BatchNormalization from tensorflow.keras.models import Model

def conv\_block(x, filters, kernel\_size=(3, 3), padding='same', activation='relu'):

"""

A simple block consisting of two convolutional layers followed by batch normalization.

""" x = Conv2D(filters, kernel\_size, padding=padding, activation=activation)(x) x = BatchNormalization()(x) x = Conv2D(filters, kernel\_size, padding=padding, activation=activation)(x) x = BatchNormalization()(x) return x

def build\_hybrid\_model(input\_shape=(64, 64, 3), num\_classes=2): inputs = Input(shape=input\_shape)

# First Block - Convolutional Layer + MaxPooling (VGG-like) x = conv\_block(inputs, 64) residual\_1 = x x = MaxPooling2D((2, 2))(x)

# Add residual connections after the blocks x = Add()([x, residual\_4]) # Skip connection after 4th block x = Add()([x, residual\_3]) # Skip connection after 3rd block x = Add()([x, residual\_2]) # Skip connection after 2nd block x = Add()([x, residual\_1]) # Skip connection after 1st block

# Flatten and Fully Connected Layers x = Flatten()(x) x = Dense(128, activation='relu')(x) x = Dropout(0.3)(x) # Dropout for regularization outputs = Dense(num\_classes, activation='softmax')(x)

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

# Compile the model model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy']) return model

model = build\_deeper\_cnn()

history = model.fit(X\_train, Y\_train, epochs=10, validation\_data=(X\_test, Y\_test))

Epoch 1/10

**105/105** ━━━━━━━━━━━━━━━━━━━━ **9s** 37ms/step - accuracy: 0.8469 - loss: 0.3856 - val\_accuracy: 0.9381 - val\_loss: 0.1559

Epoch 2/10

**105/105** ━━━━━━━━━━━━━━━━━━━━ **5s** 13ms/step - accuracy: 0.9413 - loss: 0.1526 - val\_accuracy: 0.9536 - val\_loss: 0.1079

Epoch 3/10

**105/105** ━━━━━━━━━━━━━━━━━━━━ **3s** 14ms/step - accuracy: 0.9679 - loss: 0.0871 - val\_accuracy: 0.9595 - val\_loss: 0.0897

Epoch 4/10

**105/105** ━━━━━━━━━━━━━━━━━━━━ **1s** 10ms/step - accuracy: 0.9822 - loss: 0.0535 - val\_accuracy: 0.9750 - val\_loss: 0.0653

Epoch 5/10

**105/105** ━━━━━━━━━━━━━━━━━━━━ **2s** 14ms/step - accuracy: 0.9833 - loss: 0.0448 - val\_accuracy: 0.9762 - val\_loss: 0.0639

Epoch 6/10

**105/105** ━━━━━━━━━━━━━━━━━━━━ **2s** 13ms/step - accuracy: 0.9843 - loss: 0.0455 - val\_accuracy: 0.9726 - val\_loss: 0.1215

Epoch 7/10

**105/105** ━━━━━━━━━━━━━━━━━━━━ **1s** 12ms/step - accuracy: 0.9888 - loss: 0.0385 - val\_accuracy: 0.9798 - val\_loss: 0.0657

Epoch 8/10

**105/105** ━━━━━━━━━━━━━━━━━━━━ **2s** 9ms/step - accuracy: 0.9934 - loss: 0.0203 - val\_accuracy: 0.9833 - val\_loss: 0.0604

Epoch 9/10

**105/105** ━━━━━━━━━━━━━━━━━━━━ **1s** 7ms/step - accuracy: 0.9916 - loss: 0.0286 - val\_accuracy: 0.9833 - val\_loss: 0.0496

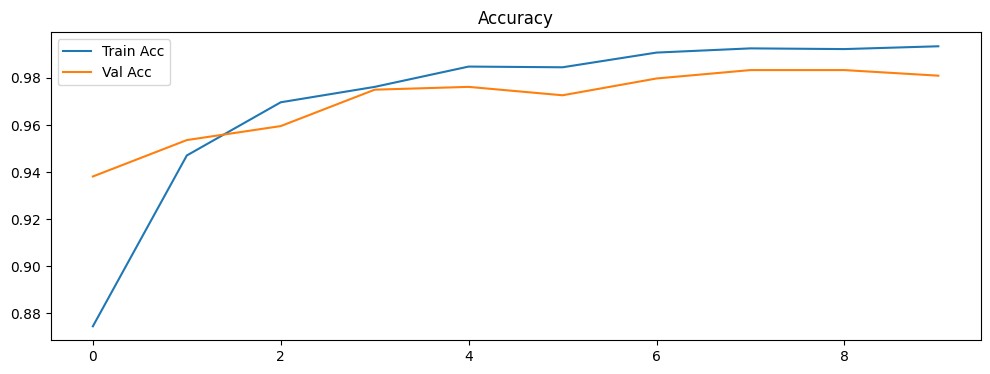
Epoch 10/10

**105/105** ━━━━━━━━━━━━━━━━━━━━ **1s** 8ms/step - accuracy: 0.9939 - loss: 0.0181 - val\_accuracy: 0.9810 - val\_loss: 0.0684

def plot\_metrics(history):

plt.figure(figsize=(12, 4)) plt.plot(history.history['accuracy'], label='Train Acc') plt.plot(history.history['val\_accuracy'], label='Val Acc') plt.legend() plt.title("Accuracy") plt.show()

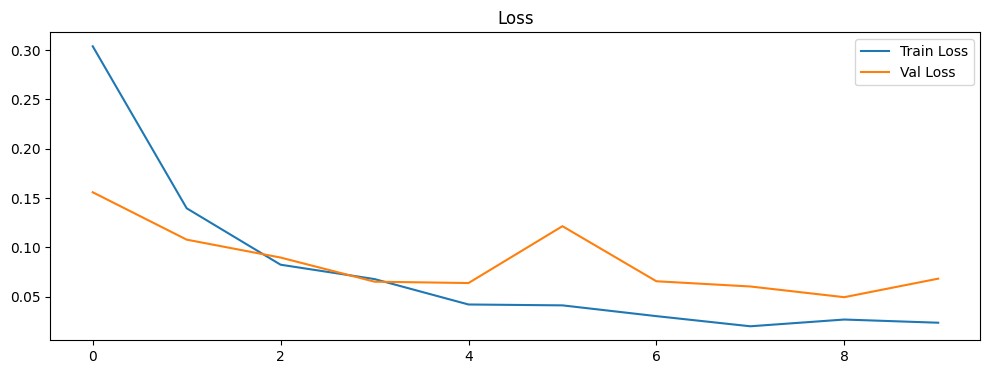
plt.figure(figsize=(12, 4)) plt.show() plt.figure(figsize=(12, 4)) plt.plot(history.history['loss'], label='Train Loss') plt.plot(history.history['val\_loss'], label='Val Loss') plt.legend() plt.title("Loss") plt.show() plot\_metrics(history)



<

Figure size 1200x400 with 0 Axes

>



y\_pred = model.predict

(

X\_test

)

y\_true = np.argmax

(

Y\_test

,

axis=

1

)

y\_pred\_classes = np.argmax

(

y\_pred

,

axis=

1

)

cm = confusion\_matrix

(

y\_true

,

y\_pred\_classes

)

disp = ConfusionMatrixDisplay(cm, display\_labels=labels) disp.plot(cmap='Blues') plt.show()

**27/27** ━━━━━━━━━━━━━━━━━━━━ **1s** 16ms/step

A blue and white squares with white text

AI-generated content may be incorrect.

Start coding or generate with AI.

def gradcam\_simple(model, image, label\_names): img\_tensor = np.expand\_dims(image, axis=0) last\_conv\_layer\_name = 'conv4'

grad\_model = Model( inputs=model.input, outputs=[model.get\_layer(last\_conv\_layer\_name).output, model.output]

)

with tf.GradientTape() as tape:

conv\_outputs, predictions = grad\_model(img\_tensor) class\_idx = tf.argmax(predictions[0]) loss = predictions[:, class\_idx]

grads = tape.gradient(loss, conv\_outputs)[0] pooled\_grads = tf.reduce\_mean(grads, axis=(0, 1)) conv\_outputs = conv\_outputs[0] heatmap = tf.reduce\_sum(pooled\_grads \* conv\_outputs, axis=-1) heatmap = tf.maximum(heatmap, 0) / tf.reduce\_max(heatmap) heatmap = cv2.resize(heatmap.numpy(), (image.shape[1], image.shape[0])) heatmap = np.uint8(255 \* heatmap) heatmap\_color = cv2.applyColorMap(heatmap, cv2.COLORMAP\_JET) superimposed = cv2.addWeighted((image \* 255).astype(np.uint8), 0.6, heatmap\_color, 0.4, 0)

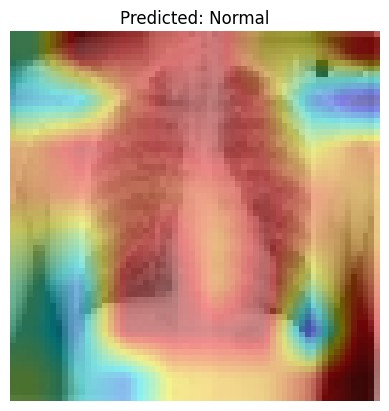
plt.imshow(superimposed) plt.title(f"Predicted: {label\_names[class\_idx]}") plt.axis('off') plt.show()

sample\_image = X\_test[500] gradcam\_simple(model, sample\_image, labels)

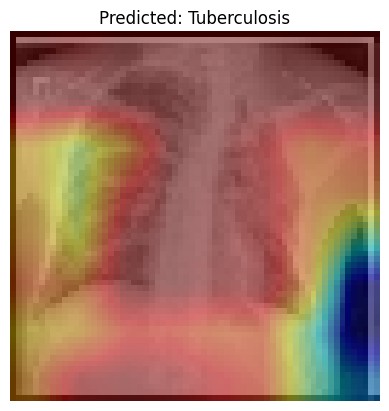
A screenshot of a medical scan

AI-generated content may be incorrect.

sample\_image = X\_test[600] gradcam\_simple(model, sample\_image, labels)



sample\_image = X\_test[102] gradcam\_simple(model, sample\_image, labels)

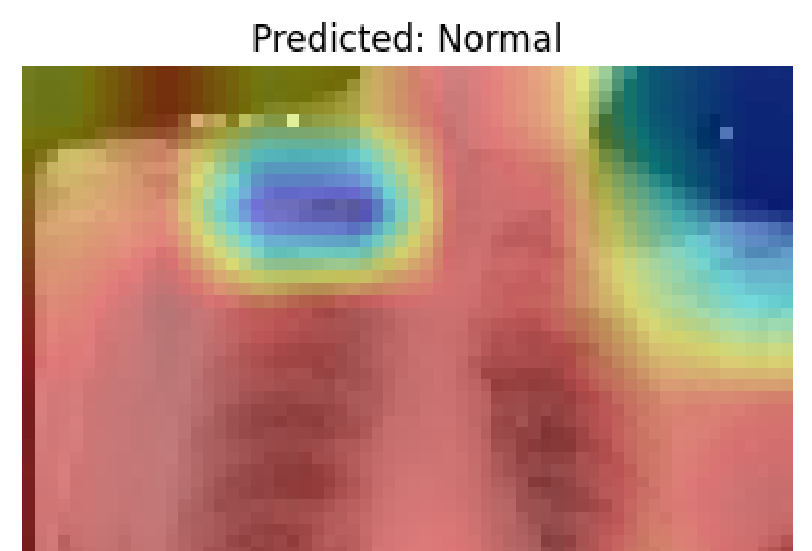


sample\_image = X\_train[349] gradcam\_simple(model, sample\_image, labels)

A close-up of a human chest

AI-generated content may be incorrect.

sample\_image = X\_train[] gradcam\_simple(model, sample\_image, labels)



**REFERENCES**

1. W. H. Organization, “Global tuberculosis report 2023,” World

HealthOrganization, 2023.

1. K. R. Steingart et al., “Sputum processing methods to improve the

sensitivity of smear microscopy for tuberculosis: a systematic review,”

The Lancet Infectious Diseases, vol. 6, no. 10, pp. 664–674, 2006.

1. J. Melendez et al., “A computer-aided diagnosis system for tuberculosis

on chest radiographs: a decision support system for radiologists,” Inter-

national Journal of Computer Assisted Radiology and Surgery, vol. 11,

pp. 163–173, 2016.

1. G. Litjens et al., “A survey on deep learning in medical image analysis,”

Medical Image Analysis, vol. 42, pp. 60–88, 2017.

1. D. Shen et al., “Deep learning in medical image analysis,” Annual Review

of Biomedical Engineering, vol. 19, pp. 221–248, 2017.

1. P. Lakhani and B. Sundaram, “Deep learning at chest radiography:

automated classification of pulmonary tuberculosis by using convolutional

neural networks,” Radiology, vol. 284, no. 2, pp. 574–582, 2017.

1. F. Pasa et al., “Efficient deep network architectures for fast chest x-ray

tuberculosis screening,” Journal of Digital Imaging, vol. 32, no. 1, pp.

38–53, 2019.

1. R. R. Selvaraju et al., “Grad-cam: Visual explanations from deep networks

via gradient-based localization,” in Proceedings of the IEEE International

Conference on Computer Vision, 2017, pp. 618–626.

1. N. Arun et al., “Assessing the trustworthiness of saliency maps for localiz-

ing abnormalities in medical imaging,” Radiology: Artificial Intelligence,

vol. 2, no. 6, p. e190192, 2020.