Enhanced Tuberculosis Detection Using Deep Learning and GAN Data Augmentation Techniques for Chest X-ray Analysis

A PROJECT REPORT

Submitted by

C. VARUN REDDY[RA2111047010200]
K. AVINASH REDDY [RA2111047010201]

Under the Guidance of

DR. KARTHICK S

Associate Professor,

Department of Computational Intelligence

in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

ARTIFICIAL INTELLIGENCE

DEPARTMENT OF COMPUTATIONAL INTELLIGENCE

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR- 603 203

NOVEMBER 2024

Department of Computational Intelligence
SRM Institute of Science & Technology
Own Work* Declaration Form

This sheet must be filled in (each box ticked to show that the condition has been met). It must be signed and dated along with your student registration number and included with all assignments you submit – work will not be marked unless this is done.

To be completed by the student for all assessments

Degree/ Course	**: B.Tech – Artificial Intelligence**
Student Name	**: C. Varun Reddy, K. Avinash Reddy**
Registration Number	**: RA2111047010200, RA2111047010201**
Title of Work	**: Enhanced Tuberculosis Detection Using Deep Learning and GAN Data Augmentation Techniques for Chest X-ray Analysis**

I / We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism**, as listed in the University Website, Regulations, and the Education Committee guidelines.

I / We confirm that all the work contained in this assessment is my / our own except where indicated, and that I / We have met the following conditions:

- Clearly referenced / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc)
- Given the sources of all pictures, data etc. that are not my own
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook / University website

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION:
I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above.
If you are working in a group, please write your registration numbers and sign with the date for every student in your group.

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that 18AIP107L- Minor Project report titled “**Enhanced Tuberculosis Detection Using Deep Learning and GAN Data Augmentation Techniques for Chest X-ray Analysis**” is the bonafide work of “**C. VARUN REDDY[RA2111047010200], K. AVINASH REDDY [RA2111047010201]**” who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE	**SIGNATURE**
DR. KARTHICK S	**DR. R. ANNIE UTHRA**
SUPERVISOR	
ASSOCIATE PROFESSOR	**PROFESSOR &HEAD**
DEPARTMENT OF	DEPARTMENT OF
COMPUTATIONAL INTELLIGENCE	COMPUTATIONAL INTELLIGENCE

ACKNOWLEDGEMENTS

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to **Dr. T. V. Gopal,** Dean-CET, SRM Institute of Science and Technology, for his invaluable support.

We wish to thank **Dr. Revathi Venkataraman**, Professor and Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We encompass our sincere thanks to, **Dr. M. Pushpalatha**, Professor and Associate Chairperson, School of Computing and **Dr. C.Lakshmi,** Professor and Associate Chairperson, School of Computing, SRM Institute of Science and Technology, for their invaluable support.

We are incredibly grateful to our Head of the Department, **Dr. R. Annie Uthra**, Professor, Department of Computational Intelligence, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We want to convey our thanks to our Project Coordinators, **Dr. Karthick S**, Panel Head, and **Dr.Athilakshmi, Dr.M.Kanipriya, Dr. Anand.R**, **Dr. Athira M. Nambiar** Panel Members, Department of Computational Intelligence, SRM Institute of Science and Technology, for their inputs during the project reviews and support.

We register our immeasurable thanks to our Faculty Advisor, **Dr. R.Siva**, Department of Computational Intelligence, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to our guide, **Dr. Karthick S**, Department of Computational Intelligence, SRM Institute of Science and Technology, for providing us with an opportunity to pursue our project under his / her mentorship. He / She provided us with the freedom and support to explore the research topics of our interest. His / Her passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank all the staff and students of Computational Intelligence, School of Computing, S.R.M Institute of Science and Technology, for their help during our project. Finally, we would like to thank our parents, family members, and friends for their unconditional love, constant support and encouragement.

Authors

ABSTRACT

Tuberculosis (TB), which kills about 1.6 million people every year, is still a major public health problem. It is especially bad in places with few resources where early detection and treatment choices are limited. To stop the spread of tuberculosis, it is important to get a correct diagnosis as soon as possible. However, chest X-rays are hard to read by hand, which often causes delays and mistakes. This project uses advanced machine learning by combining deep learning and transfer learning to make TB diagnosis more accurate and reliable.

The suggested system combines several models, including a new hybrid method that combines Histogram of Oriented Gradients (HOG) with Convolutional Neural Networks (CNN), which greatly improves the accuracy of diagnostics. Also, deep learning models that have already been trained, like Xception and MobileNetV2, are fine-tuned for TB diagnosis to get the most out of the small amount of data that is available. To fix the problem of an unbalanced dataset, which happens a lot in medical imaging, we used Deep Convolutional Generative Adversarial Networks (DCGANs) to make fake TB-positive pictures. This balanced the dataset and made the model more reliable.

Tools for explainability, such as Local Interpretable Model-Agnostic Explanations (LIME) and Gradient-weighted Class Activation Mapping (Grad-CAM), were used to make sure the system is correct and easy to understand. These methods show important parts of X-rays that affect model predictions. This helps doctors learn more and builds trust in findings made by AI. Many tests on different datasets showed that the models were very accurate. The mixed HOG + CNN model got a test accuracy of 99.5%, and the Xception and MobileNetV2 models did even better, with accuracies of 94.67% and 96.87%, respectively.

This all-around method provides a scalable and effective way to find TB. It's meant to help healthcare systems in places that aren't well-served and improve patient outcomes. The project shows that combining machine learning and AI that can be explained can help doctors figure out what's wrong with people. This opens the door for using this technology in other important areas of medical imaging.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

CLAHE	Contrast Limited Adaptive Histogram Equalization
DCGAN	Deep Convolutional Generative Adversarial Network
HOG	Histogram of Oriented Gradients
SDG	Sustainable Development Goal
LIME	Local Interpretable Model-agnostic Explanations
Grad-CAM	Gradient-weighted Class Activation Mapping
ML	Machine Learning
DB	Database

CHAPTER 1
INTRODUCTION

1.1 General (Introduction to Project)

Tuberculosis (TB) continues to be a significant global health concern, resulting in an estimated 1.6 million fatalities annually, with a particularly severe impact on low-resource settings. Effective treatment and control necessitate early and precise TB detection, given the substantial risks. Manual interpretation is labor-intensive and susceptible to errors, despite the fact that chest X-ray analysis is a prevalent diagnostic method. In order to automate and improve the accuracy of TB detection from X-ray images, our initiative employs sophisticated machine learning (ML) and deep learning (DL) techniques.

Deep Convolutional Generative Adversarial Networks (DCGAN) were employed to generate synthetic TB images, thereby addressing class imbalances and enhancing the robustness of the dataset, in order to address dataset limitations. Our machine learning infrastructure comprises both conventional and cutting-edge methodologies, including transfer learning models like MobileNet and XceptionNet, custom Convolutional Neural Networks (CNNs), and Histogram of Oriented Gradients (HOG) with Random Forest. Our hybrid HOG-CNN model achieves a remarkable 99.5% accuracy, with each model contributing to enhanced classification accuracy.

In order to guarantee that the solutions are transparent and reliable for clinical application, explainability tools, such as Grad-CAM and Local Interpretable Model-Agnostic Explanations (LIME), were incorporated to visualize and interpret model decisions. The ultimate objective of this project is to substantially influence global TB management efforts by providing a cost-effective, scalable diagnostic tool for TB in healthcare systems with limited resources.

1.2 Motivation

- **Global Health Burden of Tuberculosis (TB)**
  TB is a leading cause of death worldwide, especially prevalent in low-income regions. With around 1.6 million TB-related deaths each year, the disease continues to place a substantial burden on public health systems. There is a pressing need for innovative solutions to aid in its early detection and treatment, particularly in resource-limited areas.
- **Challenges in Manual Diagnosis**
  Radiologists rely heavily on chest X-rays for TB diagnosis, but manual image interpretation is labor-intensive, time-consuming, and prone to human error. This dependence on subjective analysis can lead to delays and inaccuracies, hindering timely intervention. Automation through artificial intelligence can assist healthcare professionals, streamlining diagnostic workflows and increasing reliability.
- **Need for Accessible Diagnostic Tools in Resource-Limited Settings**
  In many low-resource settings, hospitals and clinics lack the specialized equipment and trained personnel required for advanced TB diagnosis. Developing an AI-driven diagnostic tool that can operate on minimal infrastructure provides a sustainable way to improve TB detection, especially in under-resourced healthcare environments.
- **Advances in Machine Learning and Deep Learning for Medical Imaging**
  Recent advancements in ML and DL have demonstrated significant improvements in the accuracy and efficiency of medical imaging diagnostics. Leveraging these innovations for TB detection can enhance model accuracy and support healthcare providers with improved, automated diagnostic capabilities.
- **Importance of Explainable AI in Healthcare**
  For clinical AI applications to be trusted and widely adopted, they must provide interpretable and transparent results. Explainability techniques like LIME and Grad-CAM allow clinicians to understand model decisions by highlighting relevant regions in X-ray images, enhancing the model's reliability and trustworthiness in a healthcare setting. This project's commitment to explainable AI strengthens the potential for safe, real-world application in TB diagnosis.

1.3 Sustainable Development Goal of the Project

Our project is in accordance with the Sustainable Development Goals (SDGs), specifically:

SDG 3: Good Health and Well-Being

This project directly contributes to SDG 3, which aims to ensure healthy lives and promote well-being for all ages. Tuberculosis (TB) disproportionately affects vulnerable populations, including the elderly and those in low-resource settings, where access to timely and accurate diagnostic services may be limited. By creating an AI-powered, cost-effective tool for TB detection using chest X-ray images, our project addresses these disparities and promotes better health outcomes for at-risk communities.

Our TB detection system specifically intends to:

1. **Enhance Access to Quality Healthcare**
   By offering an automated diagnostic solution that requires minimal infrastructure, this project improves access to quality healthcare, especially in remote and underserved regions. The AI model can aid healthcare providers in accurately detecting TB, enabling faster diagnoses, earlier treatment, and better health outcomes for elderly patients and other vulnerable populations.
2. **Support Health Systems in Resource-Constrained Settings**
   By leveraging affordable and scalable AI technology, this project can alleviate pressure on overburdened healthcare systems in low-resource settings. Automating parts of the diagnostic process helps reduce costs and dependency on specialized personnel, making TB detection more sustainable and feasible in healthcare facilities with limited resources.
3. **Contribute to a Global Effort to Reduce TB Mortality**
   TB is a leading cause of mortality from infectious diseases worldwide, and our project contributes to global health efforts to reduce TB-related deaths. Through innovative use of deep learning and data augmentation techniques, the project provides a practical, scalable solution that can be adopted by healthcare providers worldwide, moving closer to SDG 3 targets for reducing infectious disease-related mortality.

CHAPTER 2
LITERATURE SURVEY

2.1 Existing Tuberculosis Detection using AI Works

The integration of machine learning and deep learning techniques has brought significant advancements in medical imaging, particularly for tuberculosis (TB) detection, where accuracy and efficiency are paramount. Various studies have demonstrated the effectiveness of these technologies in TB diagnostics by employing innovative data processing and augmentation techniques, explainable AI, and ensemble learning methods.

1. **Deep Learning for Automated TB Detection**
   Automated TB detection has been a focus of research, with deep learning models achieving impressive accuracy levels. Kant and Srivastava [1] initiated studies on automated TB detection, leveraging deep learning to overcome the limitations of manual diagnostics and presenting early successes in this field. Further developments by Singh et al. [2] highlighted the evolution of deep learning in TB diagnosis, noting improvements in accuracy and efficiency through diverse model architectures and training approaches, underscoring the potential of deep learning to transform TB diagnostics.
2. **Feature Extraction Techniques**
   Feature extraction methods such as Histogram of Oriented Gradients (HOG) have been instrumental in improving TB classification accuracy. Geethamani and Ranichitra [3] used HOG with Random Forest classifiers, achieving reliable results in distinguishing TB from non-TB cases through an edge-focused analysis of X-ray images. This approach allowed for a combination of traditional machine learning with image-specific techniques, enabling high interpretability and robustness in TB classification, particularly when combined with other models.
3. **Explainable Artificial Intelligence (XAI) in Medical Imaging**
   Interpretability in AI models is crucial in healthcare applications, and explainable AI techniques, such as Local Interpretable Model-Agnostic Explanations (LIME) and Grad-CAM, have been effectively applied to TB detection. Maheswari et al. [4] and Özkurt [5] emphasized the role of XAI in making model decisions more transparent,

which is particularly valuable in medical imaging, where understanding model reasoning is critical for clinician trust and patient safety. Ifty et al. [6] further explored XAI techniques for lung disease classification, demonstrating how visual insights into model predictions aid in clinical validation and adoption.

4. **Data Augmentation and Class Imbalance Mitigation**
   Generative Adversarial Networks (GANs) have proven useful in generating synthetic TB images, augmenting limited datasets and balancing classes effectively. Meor Yahaya and Teo [8] demonstrated the application of GANs for data augmentation, noting significant improvements in model generalization. Similarly, Basori et al. [9] combined Deep Convolutional GAN (DCGAN) with extreme gradient boosting to expand TB datasets, which enhanced model robustness and mitigated the limitations of small datasets.
5. **Transfer Learning and Ensemble Models for Enhanced Accuracy**
   Transfer learning techniques have gained prominence for TB detection, especially when utilizing pre-trained models on large datasets, which reduces the training time and enhances accuracy. Rajaraman and Antani [10] explored ensemble learning with transfer learning, achieving notable performance gains in TB detection by combining multiple CNN models. Priya and Vimina [13] employed various CNN architectures for TB classification, showing that fine-tuning pre-trained models like VGG-19 and ResNet can be highly effective in improving diagnostic accuracy.

6. **Hybrid Approaches for Improved Diagnostic Performance**
   Hybrid models that combine multiple machine learning and deep learning techniques offer robust solutions for TB detection. Thomas and Rajiv [11] explored optimization algorithms with transfer learning, which helped improve feature extraction and overall accuracy. Mehrrotraa et al. [14] and Mehta and Mehendale [15] further advanced this approach by integrating convolutional networks and gradient boosting.

Together, these studies highlight the importance of combining advanced machine learning techniques with data augmentation, explainability, and transfer learning to create reliable TB diagnostic tools. These methodologies provide a strong foundation for our project, emphasizing scalability, robustness, and clinical applicability, particularly for resource-constrained settings.

2.2 Limitations Identified from Literature Survey (Research Gaps)

The literature highlights several limitations and research gaps in the existing approaches to TB detection using deep learning and machine learning. These gaps point to areas where improvements are needed to make automated TB detection more accurate, accessible, and clinically effective:

1. **Data Scarcity and Class Imbalance**
   Many studies [1][8][9] identify the issue of limited and imbalanced datasets, especially regarding the availability of TB-positive images. Although Generative Adversarial Networks (GANs) have been employed to generate synthetic TB images, the quality and diversity of generated images are not always sufficient to completely close the gap between TB-positive and normal samples. This gap in data diversity affects model generalizability, particularly in real-world settings with varying patient demographics and disease presentations.

2. **Explainability in AI Models**
   Although explainable AI techniques like LIME and Grad-CAM have been applied to TB detection models [4][5][6], these techniques remain limited in terms of the depth of interpretability they provide. Often, these methods only offer visual cues or highlight regions of interest without clearly showing the relationship between model features and clinical indicators. This limits clinicians' ability to fully understand and trust model predictions, particularly in complex cases.

3. **Generalizability Across Diverse Clinical Settings**
   Studies have largely focused on training and testing models on curated datasets, such as those sourced from online repositories, which may not represent the variability seen in real clinical settings. This lack of generalizability limits the applicability of current models in diverse healthcare settings, particularly in resource-constrained

areas where images might be of lower quality and vary in format or diagnostic clarity [2][3].

4. **Dependence on High-Quality Images for Accurate Results**
   Many existing methods rely on high-resolution X-ray images for accurate predictions [1][11]. However, in low-resource settings, the quality of X-ray imaging may be compromised due to equipment limitations. Models need to be robust enough to handle lower-quality images without significant accuracy loss, which is an area that has received little focus in current literature.

5. **Lack of Validation on Real-World Clinical Data**
   Much of the research on TB detection relies on benchmark datasets, which may not reflect real-world challenges, such as varied patient demographics, different stages of disease progression, and imaging artifacts. Validation of models on real-world clinical data remains limited, as noted in studies like [5][10]. For models to be clinically effective, further validation and fine-tuning on diverse, real-world datasets are essential.

2.3 Research Objectives

Based on the identified limitations and gaps in the literature, this study aims to develop a robust, interpretable, and scalable AI-driven system for TB detection using chest X-rays. The specific research objectives are as follows:

- **Enhance Data Quality and Balance through GAN-based Augmentation**
  Develop and implement a Deep Convolutional Generative Adversarial Network (DCGAN) to generate high-quality synthetic TB images, addressing the class imbalance in TB datasets. This objective aims to create a balanced dataset that will improve model performance and generalizability across diverse cases.

- **Design a Hybrid Model Combining Handcrafted and Deep Learning Features**
  Construct a hybrid model that leverages both traditional feature extraction methods, such as Histogram of Oriented Gradients (HOG), and deep learning architectures. By combining handcrafted and learned features, the model seeks to improve classification accuracy and resilience across different image qualities and clinical presentations.

- **Integrate Explainable AI Techniques for Greater Model Transparency**
  Apply Local Interpretable Model-Agnostic Explanations (LIME) and Gradient-weighted Class Activation Mapping (Grad-CAM) to the model to ensure transparency in decision-making. This objective focuses on enhancing the interpretability of the model's predictions, providing clinicians with insights into the diagnostic reasoning behind each classification.

- **Optimize Performance Across Different CNN Architectures for Transfer Learning**
  Explore the performance of various transfer learning models, such as XceptionNet and MobileNet, on the augmented dataset to identify the optimal architecture for TB classification. This objective focuses on improving the diagnostic accuracy and training efficiency of the model, particularly in cases where dataset sizes are limited.

2.4 Product Backlog (Key user stories with Desired outcomes)

- **Epic 1: Enhance Data Quality and Balance through GAN-based Augmentation**
    - **User Story:**
      As a data scientist, I want to generate high-quality synthetic TB images using a DCGAN to address class imbalance in the dataset so that my model can achieve better performance.
    - **Acceptance Criteria:**
      The DCGAN should successfully generate synthetic TB images, resulting in a dataset that is balanced within a predefined ratio (e.g., 1:1 ratio of TB to normal images) for model training.

- **Epic 2: Design a Hybrid Model Combining Handcrafted and Deep Learning Features**
    - **User Story:**
      As a machine learning engineer, I want to develop a hybrid model that integrates HOG features with deep learning architectures to enhance classification accuracy across various image qualities.
    - **Acceptance Criteria:**
      The hybrid model must demonstrate improved classification accuracy compared to baseline models using only deep learning features.

- **Epic 3: Integrate Explainable AI Techniques for Greater Model Transparency**
    - **User Story:**
      As a clinician, I want to understand the reasoning behind the model’s predictions using LIME and Grad-CAM so that I can trust and effectively utilize the AI's diagnostic insights.
    - **Acceptance Criteria:**
      The implementation of LIME and Grad-CAM should provide clear visualizations and explanations for at least 90% of predictions made by the model, enabling users to interpret the decision-making process easily.

- **Epic 4: Optimize Performance Across Different CNN Architectures for Transfer Learning**
    - **User Story:**
      As a researcher, I want to evaluate various transfer learning models like XceptionNet and MobileNet on the augmented TB dataset to identify the best-performing architecture for classification tasks.
    - **Acceptance Criteria:**
      The evaluation must identify the optimal architecture based on performance metrics (e.g., accuracy, training time) and provide a comprehensive report comparing the results of at least three different models on the augmented dataset.

2.5 Plan of Action (Project Road Map)

The project road map outlines the stages and key milestones for the development and implementation of the Tuberculosis Detection using Explainable AI:

Phase 1: Initial Research and Planning

- Conduct a comprehensive literature review to identify existing research gaps in TB detection using imaging techniques.
- Define clear research objectives and finalize the project plan, including timelines, resources, and milestones.

Phase 2: Identify and Collect Relevant High-Quality Data

- Gather a diverse set of X-ray images, ensuring the dataset includes sufficient representation of both TB and normal cases.
- Perform thorough data preprocessing steps, including image normalization, resizing, and augmentation, to enhance data quality and usability.

Phase 3: Model Generalization through GANs

- Develop a Generative Adversarial Network (GAN) to synthesize high-quality TB X-ray images, addressing class imbalance and enriching the dataset for better model training.

Phase 4: Feature Extraction

- Implement Histogram of Oriented Gradients (HOG) to extract relevant features from the X-ray images, capturing essential gradients and enhancing the dataset for further analysis.

Phase 5: Model Training and Testing

- Experiment with various deep learning architectures, incorporating both HOG features and learned features, to identify the model that achieves the best performance.
- Evaluate and select the optimal model based on classification accuracy, training efficiency, and robustness across different image qualities.

Phase 6: Model Interpretability

- Employ Explainable AI (XAI) techniques such as LIME and Grad-CAM to provide insights into the model's decision-making process, fostering trust and transparency in the AI's predictions.
- Generate visual explanations for a significant percentage of predictions to enhance clinicians' understanding of the diagnostic reasoning behind the model's classifications.

CHAPTER 3
SPRINT PLANNING AND EXECUTION METHODOLOGY

3.1 Sprint I

In Sprint 1, the team's primary goal was to collect and prepare a high-quality dataset for training and testing the TB detection model. This step was crucial as the quality of input data significantly influences the model's performance and reliability, especially in medical imaging, where any inconsistency or error could impact diagnostic accuracy.

The dataset was sourced from publicly available medical image repositories, including 3,500 normal lung X-rays and 700 TB-positive images. This initial dataset was somewhat unbalanced, with a higher number of normal cases compared to TB cases. While this was noted for future steps, the first priority in Sprint 1 was to prepare the data for processing by cleaning and refining it to ensure consistency and clarity.

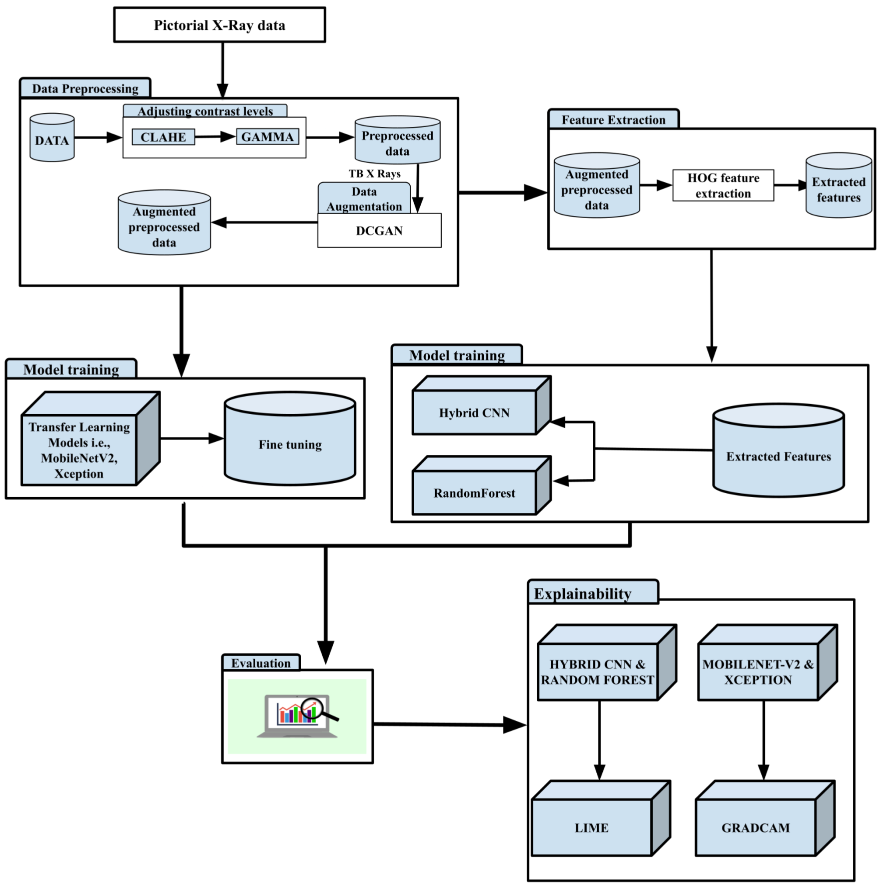


Fig. 3.1 Architecture Diagram for TB Detection using Explainable AI

A custom preprocessing pipeline was developed to handle various image inconsistencies, such as artifacts (e.g., large black or white regions) that could obscure lung structures and impact the model's ability to learn essential features. The preprocessing pipeline included several steps:

1. **Image Denoising**: Removed noise that could affect model performance.
2. **Contrast Adjustment**: Enhanced lung region visibility, especially in underexposed or overexposed images.
3. **Artifact Removal**: Using a custom algorithm, the team identified and removed large, unwanted elements from images, ensuring that only the lung area was visible.

Once the preprocessing phase was complete, the dataset consisted of refined and labeled X-ray images, which provided a stable foundation for future training and testing phases. This sprint highlighted the importance of preprocessing as a critical first step, ensuring that the data quality met the standards necessary for effective model training.

3.1.1 Objectives with User Stories of Sprint I

The primary objective of Sprint I was to establish a high-quality dataset for training and testing the tuberculosis detection model. This sprint was critical to ensuring that only clean, accurate, and representative data would be used in subsequent phases.

- **Data Acquisition:**
  - Objective: Acquire a large, diverse dataset with both TB-positive and normal X-ray images to support robust training and testing.
  - User Story: As a data scientist, I want to gather a diverse dataset of chest X-rays with both TB-positive and normal cases to create a comprehensive basis for model training.

- **Data Cleaning and Quality Control:**
  - Objective: Implement a preprocessing pipeline to filter out poor-quality images and artifacts, ensuring the integrity of data used for training.

    - User Story: As a developer, I want to preprocess the images by removing artifacts, adjusting contrast, and balancing brightness to maximize feature clarity for model training.

- **Data Labeling and Class Balancing:**
    - Objective: Correctly label and balance the dataset to reduce bias and improve model accuracy.
    - User Story: As a developer, I want to label and balance the dataset to ensure that both TB and normal images are equally represented, improving the model's ability to generalize.

3.1.2 Functional Document

The functional document for Sprint I outlines the specific functionalities that need to be implemented to achieve the objectives of the sprint. This document serves as a blueprint for the development team, ensuring that all necessary features are covered and integrated properly.

Functional Requirements:

- **Data Ingestion and Preprocessing:** Develop a pipeline to load, preprocess, and store high-quality X-ray images. The pipeline should be capable of identifying and removing artifacts using tools like OpenCV.
- **Quality Control Mechanisms:** Implement controls to check for quality and consistency in images, specifically targeting brightness, contrast, and artifacts such as large black or white areas that obscure the lung regions.
- **Dataset Storage:** Store preprocessed images in a structured format accessible to the subsequent stages, enabling smooth transitions across workflows.

Non-Functional Requirements:

- **Scalability:** The preprocessing pipeline should be capable of handling large batches of X-ray images without significant performance drops.

- **Reliability:** Ensure high data accuracy by verifying that all images are correctly labeled and processed to reduce noise, ensuring high model performance in future stages.
- **Cost-Effectiveness:** Utilize a serverless architecture for on-demand processing to optimize resource allocation and reduce costs, especially for preprocessing large datasets.

3.1.3 Architecture Document

Microservices: The *Architecture Document* outlined the microservices needed for this phase, promoting modularity and efficiency. Key microservices included:

- **Data Ingestion Service:** Collected, stored, and managed new chest X-ray images.
- **Preprocessing Service:** Handled resizing, denoising, and enhancement of images before they were passed on for feature extraction. It utilized OpenCV for artifact removal and contrast adjustments.
- **Quality Control Service:** Performed quality checks, identifying and flagging images with excessive artifacts, such as large black or white rectangles.

Event-Driven Architecture: The event-driven model facilitated real-time communication between services, allowing for efficient data processing. Events triggered actions based on certain criteria (e.g., a low-quality image could trigger an alert), ensuring that data flows smoothly between processes without delay.

Serverless Architecture: The serverless setup enabled on-demand processing, dynamically allocating resources only when needed. This approach minimized latency and improved efficiency, particularly in handling large volumes of images. Serverless functions were also used to handle contrast adjustment and artifact detection in high batches without the need for dedicated infrastructure

3.1.4 Outcome of Objectives/ Result Analysis

The preprocessing process resulted in a cleaned dataset of 439 TB-infected images and 3,500 normal lung images. This dataset was essential in minimizing noise and improving data quality for model training. The enhancements in contrast and brightness made the lung structures clearer and more distinct, which is crucial for accurate TB classification. The successful removal of low-quality images also mitigated the risk of the model misinterpreting artifacts as features.

3.1.5 Sprint Retrospective

Successes:

- The preprocessing service effectively identified and removed artifacts from images, resulting in a high-quality dataset that met the objectives of Sprint I.
- Event-driven architecture allowed for seamless, real-time communication between services, making the workflow efficient and reducing processing time.

Challenges:

- Detecting and removing artifacts, especially in TB images, proved time-intensive and complex, as it required customized code and quality checks for accurate preprocessing.
- The serverless architecture posed challenges in managing peak loads, requiring optimization to handle large image batches without delays.

Areas of Improvement:

- Future sprints could automate artifact detection using advanced techniques like image segmentation, reducing the manual intervention required.
- Improvements in scalability of the serverless functions could enhance efficiency, allowing faster batch processing of images in high-load situations.

3.2 Sprint II

In Sprint 2, the team aimed to address the class imbalance observed in Sprint 1 by using data augmentation techniques to increase the number of TB images. This was essential to ensure that the model could learn effectively from both classes, avoiding any inherent bias due to class imbalance. The team decided to use a Deep Convolutional Generative Adversarial Network (DCGAN) to generate synthetic TB images that closely resembled real ones, as traditional data augmentation methods like rotation or flipping were insufficient for this complex medical imaging problem.

The DCGAN consisted of two main components: a generator and a discriminator. The generator created synthetic TB images, while the discriminator evaluated whether the generated images resembled the real ones. This adversarial setup allowed the model to progressively improve the quality of the synthetic images, as the generator's goal was to "fool" the discriminator into classifying the synthetic images as real.

3.2.1 Objectives with User Stories of Sprint 2

The second sprint focused on augmenting the dataset to address the class imbalance between TB and normal cases. The objective was to generate synthetic TB images using Deep Convolutional Generative Adversarial Networks (DCGAN), increasing dataset diversity and improving the model's ability to generalize.

- **Dataset Balancing through Augmentation:**
  - ***Objective*****:** Generate synthetic TB images to balance the dataset, ensuring equal representation of TB and normal cases.
  - ***User Story*****:** As a data scientist, I want to use DCGAN to create synthetic images, helping to equalize the dataset distribution and improve model performance.
- **High-Quality Synthetic Image Generation:**
  - ***Objective*****:** Fine-tune the GAN model to ensure that generated images are realistic and reflective of true TB characteristics.

- ○ ***User Story*:** As a developer, I need high-quality synthetic images so the model can learn TB-specific features without misclassification due to unrealistic images.

3.2.2 Functional Document

Functional Requirements:

- **Synthetic Image Generation with DCGAN:** Implement a GAN-based data augmentation method, where the GAN generates images closely resembling real TB images to balance the dataset.
- **Quality Control of Synthetic Images:** Integrate quality assessment tools to evaluate the authenticity of GAN-generated images, ensuring that synthetic images closely resemble true TB cases.

Non-Functional Requirements:

- **Computational Efficiency:** DCGAN training and image generation should be optimized to minimize computation time and resource usage.
- **Reliability:** Ensure the consistency of generated images by testing the GAN model thoroughly, producing realistic and diverse images for the dataset.

3.2.3 Architecture Document

Microservices: Key microservices in this sprint included:

- **Augmentation Service**: Handled DCGAN processing to generate synthetic images, monitored by quality control measures to verify the accuracy of the generated images.
- **Quality Assessment Service**: Performed post-generation checks on synthetic images, flagging any low-quality or unrealistic images for review before they were added to the dataset【12†source】.

Event-Driven Architecture: The event-driven model allowed tasks to be processed asynchronously, making the augmentation workflow efficient.

Serverless Architecture: Serverless functions were utilized for on-demand GAN training and image generation, which optimized costs and prevented resource wastage.

3.2.4 Outcome of Objectives/ Result Analysis

The DCGAN successfully generated 3,000 synthetic TB images, bringing the dataset closer to balance. This improved the model's ability to generalize, especially when handling underrepresented TB cases. Quality control measures ensured that the generated images met specific criteria, closely resembling actual TB images, thus reducing the risk of misclassification in subsequent model training.

3.2.5 Sprint Retrospective

Successes:

- DCGAN effectively addressed the class imbalance by generating high-quality synthetic images, meeting the augmentation needs.
- Event-driven processing made augmentation seamless, allowing each stage to trigger the next, thus minimizing manual oversight.

Challenges:

- Training the GAN required extensive computational resources and careful monitoring to ensure convergence and prevent overfitting.
- Quality control of synthetic images was labor-intensive, as the GAN occasionally produced images that lacked key TB characteristics, requiring adjustments to the model.

Areas of Improvement:

- Future iterations could incorporate automated hyperparameter tuning to streamline GAN training, minimizing resource usage.
- Implementing more advanced quality assessment techniques, like deep learning-based quality control, could improve the reliability of synthetic images without excessive manual intervention.

3.3 Sprint III

In Sprint 3, the primary objective was to develop and test multiple model architectures, incorporating explainability features to ensure that the model's predictions were interpretable and could be trusted by medical professionals. The team explored a hybrid approach using both traditional feature extraction and deep learning, integrating Histogram of Oriented Gradients (HOG) for feature extraction with Convolutional Neural Networks (CNN) for TB classification. They also evaluated transfer learning models, such as XceptionNet and MobileNetV2, to leverage pre-trained knowledge for better accuracy.

The hybrid HOG+CNN model combined traditional image processing with deep learning, providing a unique perspective on feature extraction. HOG emphasized edges and gradients, capturing structural information critical for TB detection. These HOG features were then fed into a CNN, which further learned complex patterns, combining both handcrafted and learned features to improve classification.

To enhance the model's clinical applicability, explainability tools like Grad-CAM were integrated, which allowed the model to produce heat maps highlighting important areas in the image that influenced the classification decision. This feature helped clinicians understand the model's decision-making process by indicating which lung areas were most relevant in predicting TB. Grad-CAM visualizations proved particularly useful for transparency, as they provided an intuitive, visual representation of the model's focus areas, boosting trust among healthcare providers.

3.3.1 Objectives with User Stories of Sprint III

Sprint III focused on model development, with a dual objective of achieving high classification accuracy and enhancing explainability for clinical use. This sprint involved testing hybrid and transfer learning models for TB classification and integrating explainability tools to interpret model predictions.

- **Hybrid Model for TB Classification:**
    - ***Objective*: ** Combine HOG-based feature extraction and CNN architectures to leverage both traditional and deep learning features for accurate TB classification.
    - ***User Story*: ** As a machine learning engineer, I want to use hybrid models combining HOG and CNN features to achieve higher classification accuracy for TB detection.

- **Explainability for Clinical Interpretability:**
    - ***Objective*: I**ntegrate tools like Grad-CAM to visualize critical areas of X-ray images, making model predictions understandable for clinicians.
    - ***User Story*: ** As a healthcare provider, I want to visualize the important regions in X-ray images, helping to validate and understand the model's predictions for better diagnostic trust.

3.3.2 Functional Document

Functional Requirements:

- **HOG and CNN-Based Hybrid Model:** Develop a hybrid model that combines HOG-based feature extraction with CNN layers for more comprehensive feature learning.
- **Explainability with Grad-CAM:** Integrate Grad-CAM to provide visual explanations of model predictions, highlighting critical regions on X-rays that influenced the model's decision.

Non-Functional Requirements:

- **Accuracy:** The hybrid model should achieve high accuracy in classifying TB and normal cases to support reliable diagnostics.
- **Interpretability:** Ensure that the model's predictions are interpretable, providing clinicians with sufficient transparency to validate and trust the model's decisions

3.3.3 Architecture Document

Microservices:

- **Model Training Service**: Facilitated model training and evaluation, with a modular design allowing multiple models to be tested and compared.
- **Explainability Service**: Provided visual explanations of predictions using Grad-CAM, making the model's decision process interpretable 【12†source】.

Event-Driven Architecture: This architecture allowed seamless integration between model training and explainability. After the model generated predictions, an event triggered the Grad-CAM service, automatically creating interpretative heatmaps for clinicians to review.

Serverless Architecture: Serverless functions handled prediction requests and generated Grad-CAM visualizations on demand. This setup minimized resource usage and provided rapid responses for explainability inquiries, making it ideal for clinical applications requiring real-time diagnostics.

3.3.4 Outcome of Objectives/Result Analysis

The hybrid HOG+CNN model achieved superior accuracy, with Grad-CAM visualizations effectively highlighting lung regions associated with TB. This explainability feature was crucial for clinical validation, allowing healthcare providers to understand and trust the model's decision-making process. The hybrid model's success demonstrated the effectiveness of combining traditional and deep learning features for complex classification tasks.

3.3.5 Sprint Retrospective

Successes:

- The hybrid model achieved high accuracy, with Grad-CAM providing insightful visual explanations that facilitated clinical validation.

- Event-driven architecture made model training and explainability workflows efficient, with each service triggering the next step, minimizing latency.

Challenges:

- Grad-CAM occasionally highlighted non-diagnostic features, indicating potential model dependency on irrelevant areas within X-rays.
- Managing interpretability across different models proved complex, especially when reconciling results from Grad-CAM and LIME.

Areas of Improvement:

- Refining Grad-CAM to focus solely on diagnostic regions could improve explainability, reducing potential model bias toward irrelevant features.
- Implementing more advanced interpretability techniques, like SHAP, may offer greater flexibility and reliability for clinical applications.

CHAPTER 4
RESULTS AND DISCUSSIONS

4.1 Project Outcomes (Performance Evaluation, Comparisons, Testing Results)

Dataset:

We utilized a dataset developed by a collaborative team from Qatar University, the University of Dhaka, and researchers from Malaysia, in partnership with medical professionals from Hamad Medical Corporation and Bangladesh. This dataset comprises chest X-ray images classified into two categories: Tuberculosis (TB) positive cases and normal cases. For our analysis, we have 700 TB-positive X-ray images and 3,500 normal X-ray images, creating a balanced dataset for our study. This dataset is crucial for the development and testing of models aimed at detecting TB in chest X-rays.
During our analysis, we encountered images with random large white and black rectangles that could interfere with the extraction of HOG features. To address this issue, we removed these images, resulting in the following composition.

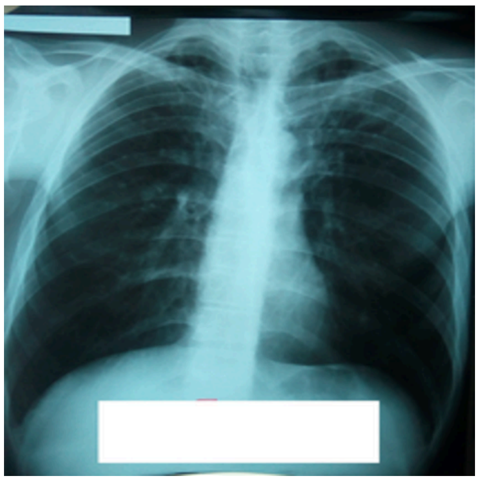


Fig. 4.1 TB X-ray Image having Unwanted White Rectangle

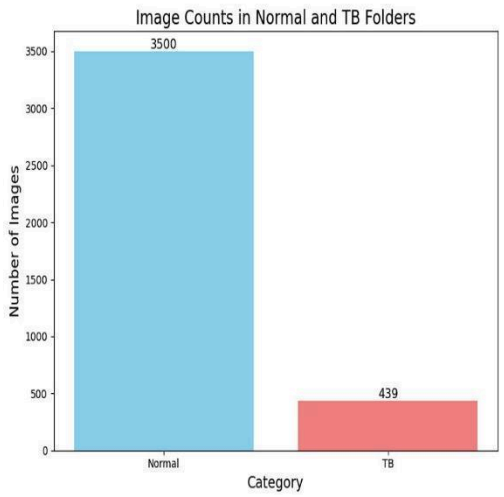
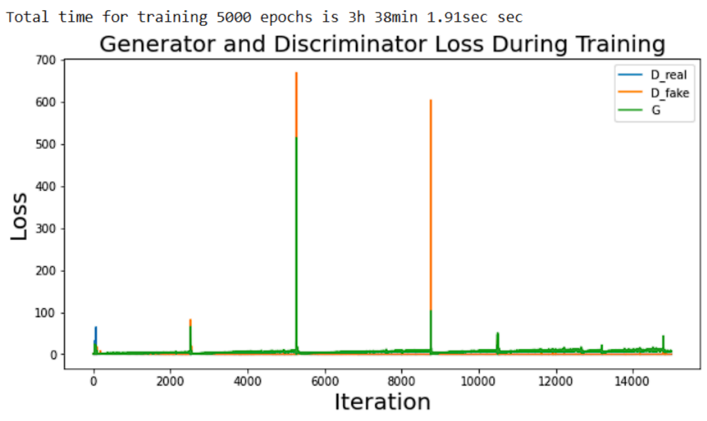


Fig. 4.2 Image count per class after preprocessing

Class Imbalance and the Need for Generalization:

In our dataset, we have a significant class imbalance, with 439 TB-positive X-ray images compared to 3,500 normal X-ray images. This disparity can lead to a model that is biased towards the majority class (normal cases), potentially resulting in poor detection performance for the minority class (TB-positive cases).

To mitigate this issue, generalization is essential. Generalization allows the model to learn features that are representative of the data distribution as a whole, rather than memorizing the training data. By generating high-quality synthetic images of TB-positive cases through the GAN, we can create a more balanced dataset. This approach not only helps in enhancing the model's ability to recognize TB-positive cases but also improves overall robustness and performance, leading to more accurate diagnostic capabilities in real-world scenarios.



```
Total time for training 5000 epochs is 3h 38min 1.91sec sec
```

Fig. 4.3 The Loss of Generator and Discriminator after training for 5000 epochs

The quality of images produced:

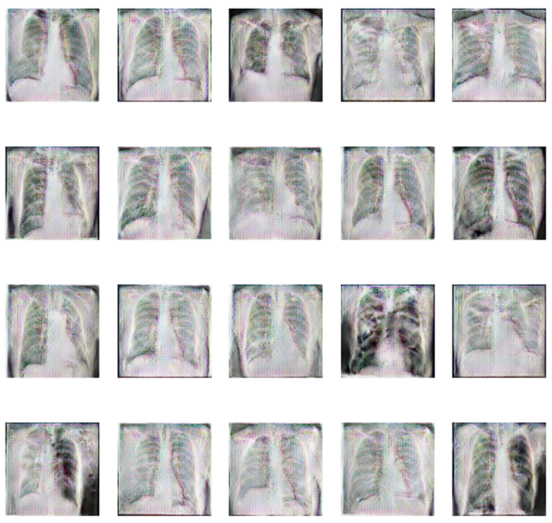
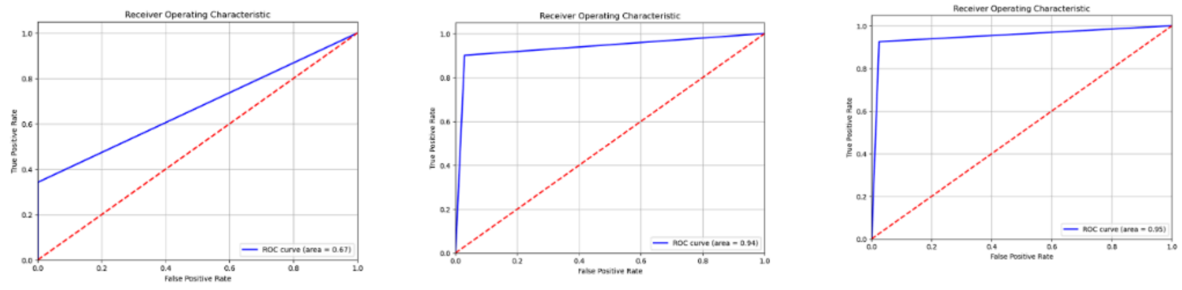


Fig. 4.4 TB X-ray Images generated by GAN

Using the generated images to assess generalization, we conducted model testing with a combination of HOG feature extraction and a Random Forest classifier. The results of this evaluation are as follows:



Without Data Augmentation   With Traditional Data Augmentation   With GAN Data Augmentation

Fig. 4.5 ROC Curve of HOG+Random Forest Model under Various Data Augmentation Techniques

These results indicate the model's effectiveness in distinguishing between TB-positive and normal X-ray images. The performance metrics obtained will help us understand how well the model generalizes to unseen data, particularly in addressing the class imbalance present in the dataset. Overall, the results contribute valuable insights into the potential of our approach for improving tuberculosis detection in chest X-rays.The classification performance of various models on normal and TB lung X-ray images is summarized in Tables 1.0 and 1.1. For the detection of TB in X-rays, both handcrafted features and deep learning models were explored. The HOG + CNN model achieved the highest overall performance with an accuracy, precision, recall, and F1-score of 1.00 for both normal and TB images. The HOG + RandomForest model also demonstrated strong performance, with an accuracy of 0.95 across both normal and TB X-rays, but slightly lower recall values compared to HOG + CNN. Among deep learning architectures, XceptionNet achieved high precision and recall scores, with an accuracy of 0.99 for both normal and TB cases, while MobileNetV2 performed comparably with an accuracy of 0.96

Model	Accuracy	Test Accuracy	Precision	Recall	F1-Score	Support
HOG + RandomForest	0.95	92.97	0.94	0.98	0.96	732
HOG + CNN	1.00	99.5	1.00	1.00	1.00	732
XceptionNet	0.99	94.67	0.98	0.99	0.98	338
MobileNetV2	0.96	96.87	0.94	1.00	0.97	339

Table 4.1 Classification report of normal X rays

Model	Accuracy	Test Accuracy	Precision	Recall	F1-Score	Support
HOG + RandomForest	0.95	92.97	0.97	0.93	0.95	656
HOG + CNN	1.00	99.5	1.00	1.00	1.00	656
XceptionNet	0.99	94.67	0.99	0.98	0.98	339
MobileNetV2	0.96	96.87	1.00	0.94	0.97	354

Table 4.2 Classification report of TB X rays

The application of Explainable AI (XAI) techniques, including LIME and Grad-CAM, provided essential insights into model behavior and interpretability. LIME was applied to the HOG-based models (RandomForest and CNN) to understand feature importance and model decision-making. Although HOG + CNN showed promising results, LIME revealed it occasionally recognized irrelevant or incorrect features in the image, indicating potential overfitting to non-TB related patterns.

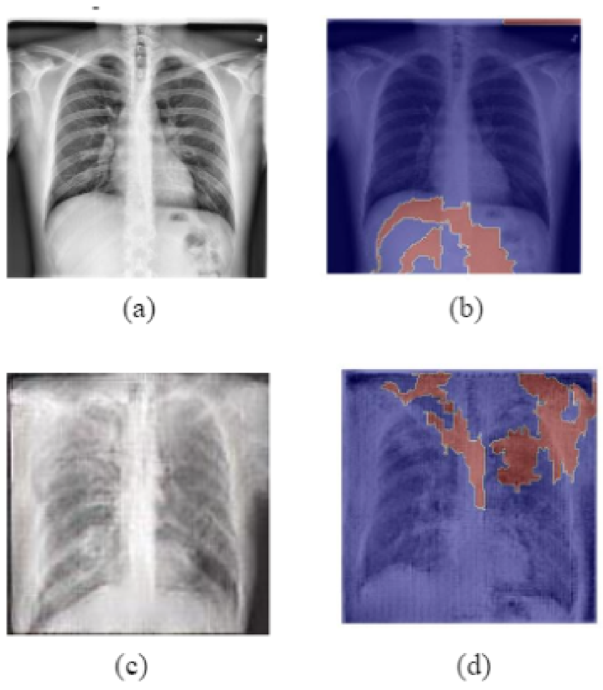


Fig. 4.1(a) Normal Lung X-ray
Fig. 4.1(b) LIME Explanation on Normal Lung X-ray
Fig. 4.1(c) Tuberculosis affected Lung X-ray
Fig. 4.1(d) LIME Explanation on TB affected Lung X-ray

To address this limitation, Grad-CAM was applied to the deep learning models, XceptionNet and MobileNetV2, to visualize model focus areas on the X-rays. The Grad-CAM heatmaps demonstrated that both models effectively highlighted clinically relevant areas of the lung, such as regions with abnormal textures or shadows associated with TB.

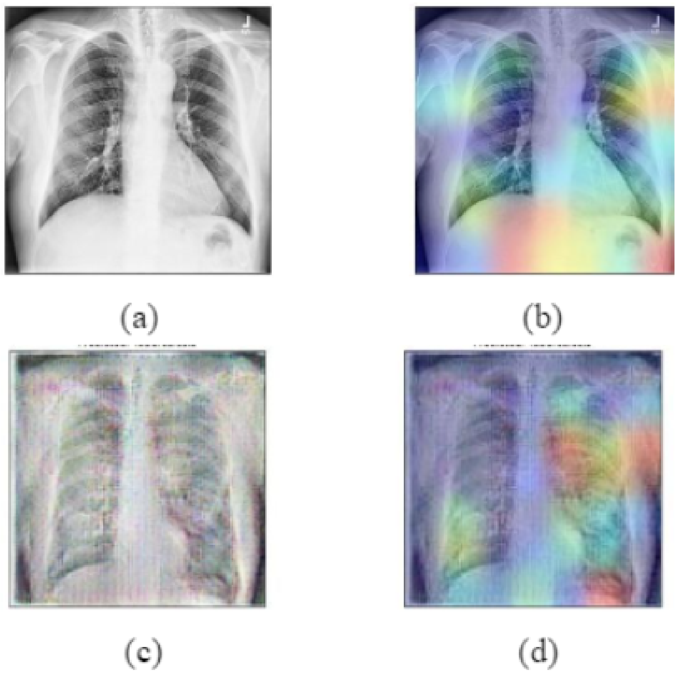


Fig. 4.2(a) Normal Lung X-ray
Fig. 4.2(b) GradCAM Explanation on Normal Lung X-ray
Fig. 4.2(c) Tuberculosis affected Lung X-ray
Fig. 4.2(d) GradCAM Explanation on TB affected Lung X-ray

This added layer of interpretability suggests that these deep learning models are likely recognizing features more aligned with the pathology of TB, enhancing their suitability for real-world diagnostic applications.

CHAPTER 5
CONCLUSION AND FUTURE ENHANCEMENT

Conclusion:

In conclusion, our research successfully demonstrates the power of advanced machine learning techniques, particularly deep learning models, in enhancing the accuracy and efficiency of tuberculosis (TB) detection from chest X-ray images. By leveraging hybrid models, such as the combination of HOG and CNN, and employing transfer learning with architectures like XceptionNet and MobileNetV2, we achieved high levels of diagnostic accuracy. The use of Deep Convolutional Generative Adversarial Networks (DCGANs) for data augmentation played a crucial role in addressing the class imbalance issue, further improving model performance.

These methodologies not only resulted in superior classification accuracy but also provided robust and scalable solutions that can be applied in resource-constrained healthcare environments where manual diagnosis is challenging. Our findings highlight the potential of integrating AI-driven tools into clinical workflows, offering faster, more reliable TB diagnosis, ultimately helping to reduce the burden of the disease globally. As technology continues to evolve, the adoption of such models can significantly impact early disease detection and treatment, improving healthcare outcomes for millions of people.

Future Enhancement:

- **Developing a More Robust GAN Model**

  To improve the quality of the generated images, we aim to enhance our current GAN model. A more robust GAN can produce high-resolution and more realistic synthetic images, which will be vital for augmenting our training dataset. This enhancement will not only help in addressing the class imbalance but also contribute to the overall accuracy and reliability of our models for tuberculosis detection.

- **Incorporating Segmentation Techniques**

  In addition to improving the GAN model, we plan to implement segmentation techniques to refine our analysis. By segmenting the chest X-ray images, we can focus on specific regions of interest, such as the lungs, which will facilitate better interpretation of the data. This approach will enable us to extract more relevant features that are critical for accurately diagnosing TB.

- **Utilizing Grad-CAM for Enhanced Interpretability**

  To further enhance model interpretability, we intend to use Gradient-weighted Class Activation Mapping (Grad-CAM). This technique will allow us to visualize which parts of the X-ray images contribute most to the model's predictions. By applying Grad-CAM in conjunction with the segmented images, we can provide clearer insights into the model's decision-making process, helping both clinicians and researchers understand how the model identifies TB in chest X-rays. This interpretability is crucial for building trust in automated diagnostic systems in clinical settings.

REFERENCES

[1]. Kant, S. and Srivastava, M.M., 2018, November. Towards automated tuberculosis detection using deep learning. In 2018 IEEE Symposium Series on Computational Intelligence (SSCI) (pp. 1250-1253). IEEE.

[2]. Singh, M., Pujar, G.V., Kumar, S.A., Bhagyalalitha, M., Akshatha, H.S., Abuhaija, B., Alsoud, A.R., Abualigah, L., Beeraka, N.M. and Gandomi, A.H., 2022. Evolution of machine learning in tuberculosis diagnosis: a review of deep learning-based medical applications. Electronics, 11(17), p.2634.

[3]. Geethamani, R. and Ranichitra, A., 2023. Enhancing Tuberculosis Detection: Leveraging RF-HOG Model for Automated Diagnosis from Chest X-ray Images. Procedia Computer Science, 230, pp.21-32.

[4]. Maheswari, B.U., Sam, D., Mittal, N., Sharma, A., Kaur, S., Askar, S.S. and Abouhawwash, M., 2024. Explainable deep-neural-network supported scheme for tuberculosis detection from chest radiographs. BMC Medical Imaging, 24(1), p.32.

[5]. Özkurt, C., Improving Tuberculosis Diagnosis using Explainable Artificial Intelligence in Medical Imaging. Journal of Mathematical Sciences and Modelling, 7(1), pp.33-44.

[6]. Ifty, T.T., Shafin, S.A., Shahriar, S.M. and Towhid, T., 2024. Explainable Lung Disease Classification from Chest X-Ray Images Utilizing Deep Learning and XAI. arXiv preprint arXiv:2404.11428.

[7]. Lubis, A.R., Prayudani, S., Fatmi, Y. and Lase, Y.Y., 2021, September. Extraction in Detecting Tuberculosis X-Ray Results using Histogram of Oriented Gradients. In 2021 4th International Conference of Computer and Informatics Engineering (IC2IE) (pp. 5-8). IEEE.

[8]. Meor Yahaya, M.S. and Teo, J., 2023. Data augmentation using generative adversarial networks for images and biomarkers in medicine and neuroscience. Frontiers in Applied Mathematics and Statistics, 9, p.1162760.

[9]. Basori, A.H., Malebary, S.J. and Alesawi, S., 2023. Hybrid Deep Convolutional Generative Adversarial Network (DCGAN) and Xtreme Gradient Boost for X-ray Image Augmentation and Detection. Applied Sciences, 13(23), p.12725.

[10]. Rajaraman, S. and Antani, S.K., 2020. Modality-specific deep learning model ensembles toward improving TB detection in chest radiographs. IEEE Access, 8, pp.27318-27326.

[11]. Thomas, R. and Rajiv, M., 2021. Impact of Optimization Algorithms in Detection of Tuberculosis using Transfer Learning. In National Conference on Emerging Computer Applications (Vol. 3, No. 1).

[12]. PV, G.P., Dinesh, R., Paduri, A.R. and Darapaneni, N., 2024. Detection, Localization of Cardiomegaly and TB Disease of CXR Images using Deep Learning. EAI Endorsed Transactions on Intelligent Systems and Machine Learning Applications, 1.

[13]. Anu Priya, P. and Vimina, E.R., 2021. Tuberculosis detection from CXR: an approach using transfer learning with various CNN architectures. In International Conference on Communication, Computing and Electronics Systems: Proceedings of ICCCES 2020 (pp. 407-418). Springer Singapore.

[14]. Mehrrotraa, R., Ansari, M.A., Agrawal, R., Tripathi, P., Heyat, M.B.B., Al-Sarem, M., Muaad, A.Y.M., Nagmeldin, W.A.E., Abdelmaboud, A. and Saeed, F., 2022. Ensembling of efficient deep convolutional networks and machine learning algorithms for resource effective detection of tuberculosis using thoracic (chest) radiography. IEEE Access, 10, pp.85442-85458.

[15]. Mehta, T. and Mehendale, N., 2021. Classification of X-ray images into COVID-19, pneumonia, and TB using cGAN and fine-tuned deep transfer learning models. Research on Biomedical Engineering, 37, pp.803-813.

APPENDIX A
CODING

- **Data Preprocessing and Cleaning Coding Files:**
- **Eliminating Images with White & Black Rectangles**

```
import cv2
import os
import numpy as np
import shutil

# Function to check if an image contains large white rectangles
def has_white_rectangle(image_path, white_threshold=2000):
    img = cv2.imread(image_path)  # Read the image
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # Convert to grayscale
    _, thresh = cv2.threshold(gray, 240, 255, cv2.THRESH_BINARY)  # Detect white areas

    # Count the number of white pixels
    white_pixels = np.sum(thresh == 255)

    # If the white pixels exceed the threshold, return True
    return white_pixels > white_threshold

# Function to filter images with white rectangles and copy them to a new folder
def filter_and_copy_images_with_white_rectangles(df, removed_folder='removed',
white_threshold=2000):
    # Create the 'removed' folder if it doesn't exist
    if not os.path.exists(removed_folder):
        os.makedirs(removed_folder)

    keep_list = []
    removed_list = []
    for index, row in df.iterrows():
        if (row['label'] == 'tb') and has_white_rectangle(row['filepath'], white_threshold):
            # Copy the image to the 'removed' folder
            destination_path = os.path.join(removed_folder, os.path.basename(row['filepath']))
            shutil.copy2(row['filepath'], destination_path)
            removed_list.append(row)
        else:
            keep_list.append(row)

        # Create a new DataFrame with images that do not contain large white rectangles
    filtered_df = pd.DataFrame(keep_list)
    removed_df = pd.DataFrame(removed_list)

    return filtered_df, removed_df

# Apply the filter to your dataframe
filtered_df, removed_df = filter_and_copy_images_with_white_rectangles(df,
removed_folder='removed', white_threshold=2000)

print(f"Number of images after filtering: {len(filtered_df)}")
print(f"Number of images removed: {len(removed_df)}")
```

- **Calculating and Adjusting Contrast Levels:**

```
# Function to calculate the contrast of an image
def calculate_contrast(image_path):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)  # Read the image in
grayscale
    if img is None:
        return 0  # Return 0 if the image cannot be read
    contrast = np.std(img)  # Calculate the standard deviation of pixel intensities
    return contrast
```

```
# Calculate contrast for all images in the filtered DataFrame
filtered_df['contrast'] = filtered_df['filepath'].apply(calculate_contrast)

# Group by class and calculate average contrast
contrast_means = filtered_df.groupby('label')['contrast'].mean()

# Bar plot showing average contrast levels per class
plt.figure(figsize=(8, 6))
contrast_means.plot(kind='bar', color=['blue', 'orange'])
plt.title('Average Contrast Levels by Class')
plt.xlabel('Class')
plt.ylabel('Average Contrast')
plt.xticks(rotation=0)

# Add the average contrast values on top of the bars
for p in plt.gca().patches:
    plt.annotate(f'{p.get_height():.2f}', (p.get_x() + p.get_width() / 2., p.get_height()),
                 ha='center', va='center', xytext=(0, 5), textcoords='offset points')

plt.show()

# Apply CLAHE for contrast normalization
def clahe(image, clip_limit=2.0, grid_size=(8, 8)):
    if len(image.shape) == 3:  # Convert to grayscale if the image is in color
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    clahe_obj = cv2.createCLAHE(clipLimit=clip_limit, tileGridSize=grid_size)
    clahe_image = clahe_obj.apply(image)

    return clahe_image

# Adjust contrast and save images to new folders
def adjust_contrast_and_save(df, normal_dir, tb_dir):
    adjusted_data = []

    for index, row in df.iterrows():
        img_path = row['filepath']
        label = row['label']

        img = cv2.imread(img_path)
        if img is None:
            continue  # Skip if the image cannot be read

        # Adjust contrast using CLAHE
        adjusted_img = clahe(img)

        # Save the adjusted image in the respective folder
        if label == 'normal':
            output_path = os.path.join(normal_dir, os.path.basename(img_path))
        else:  # label == 'tb'
            output_path = os.path.join(tb_dir, os.path.basename(img_path))

        cv2.imwrite(output_path, adjusted_img)

        # Add the new file path and label to the adjusted data list
        adjusted_data.append({'filepath': output_path, 'label': label})

    # Create a new DataFrame for the adjusted images
    adjusted_df = pd.DataFrame(adjusted_data)

    return adjusted_df

# Main process
def main(df, output_dir='adjusted_images'):
    # Create folders for adjusted images
    normal_dir, tb_dir = create_folders(output_dir)
    # Adjust contrast and save images, and create a new dataframe
    adjusted_df = adjust_contrast_and_save(df, normal_dir, tb_dir)

    return adjusted_df

adjusted_df = main(filtered_df)

print(adjusted_df)

def gamma_correction(image, gamma=1.5):
    inv_gamma = 1.0 / gamma
    table = np.array([((i / 255.0) ** inv_gamma) * 255 for i in np.arange(0,
256)]).astype("uint8")
    return cv2.LUT(image, table)

# Apply gamma correction to all images in adjusted_df and save them to new folders
def apply_gamma_and_save(adjusted_df, gamma_value=1.5,
output_dir='gamma_adjusted_images'):
    # Create new folders for gamma-adjusted images
    normal_dir = os.path.join(output_dir, 'normal_gamma_adjusted')
    tb_dir = os.path.join(output_dir, 'tb_gamma_adjusted')
    os.makedirs(normal_dir, exist_ok=True)
    os.makedirs(tb_dir, exist_ok=True)

    gamma_adjusted_data = []

    for index, row in adjusted_df.iterrows():
        img_path = row['filepath']
        label = row['label']

        img = cv2.imread(img_path)
        if img is None:
            continue  # Skip if the image cannot be read
        # Apply gamma correction
        gamma_img = gamma_correction(img, gamma=gamma_value)

        # Save the gamma-adjusted image in the respective folder
        if label == 'normal':
            output_path = os.path.join(normal_dir, os.path.basename(img_path))
        else:  # label == 'tb'
            output_path = os.path.join(tb_dir, os.path.basename(img_path))

        cv2.imwrite(output_path, gamma_img)

        # Add the new file path and label to the gamma-adjusted data list
        gamma_adjusted_data.append({'filepath': output_path, 'label': label})

    # Create a new DataFrame for the gamma-adjusted images
    gamma_adjusted_df = pd.DataFrame(gamma_adjusted_data)

    return gamma_adjusted_df

# Apply gamma correction to adjusted_df
gamma_adjusted_df = apply_gamma_and_save(adjusted_df, gamma_value=1.5)

# Display the gamma-adjusted DataFrame
print(gamma_adjusted_df)
```

- **Implementing DCGAN:**

```
def define_grid(data_images, nrows=4, ncols=5, plot_grid=True):

    start = time.time()
    # Number of GPUs available. Use 0 for CPU mode.
    ngpu = 1
    # Decide which device we want to run on
    device = torch.device("cuda:0" if (torch.cuda.is_available() and ngpu > 0) else "cpu")
    # Rearange the shaphe of the data
    data_transp = [np.transpose(data_images[i,:,:]) for i in
range(data_images[:nrows*ncols].shape[0])]
    # From to torch type for the grid
    data_transp = torch.Tensor(data_transp)
    print(f'The shape is reordered from {data_images.shape[1:]} to {data_transp.shape[1:]} in
{_time(start, time.time())}')

    # Make the grid
    grid_images = np.transpose(
        vutils.make_grid(
            data_transp.to(device)[:nrows*ncols],
            nrow=nrows,
            padding=2,
            normalize=True,
            scale_each=True,
            pad_value=1,
        ).cpu(), axes=(2,1,0))

    # Show the output grid
    if plot_grid:
        plt.figure(figsize=(12,12))
        plt.axis("off")
        plt.title(f'Grid of {nrows*ncols} real images', fontsize=27)
        plt.imshow(grid_images)

    return grid_images

grid_X_pneumonial = define_grid(X_tb, plot_grid=False)

fig, (ax1)= plt.subplots(nrows=1, ncols=1, figsize=(19, 8))

ax1.imshow(grid_X_pneumonial); ax1.axis('off')
```

```
ax1.set_title(label = 'Grid of X-Ray tb images', fontsize = 27)

plt.tight_layout(pad=1.08, h_pad=None, w_pad=None, rect=[0, 0.03, 1, 0.95])

"""## Set the parameters  """

# Number of training epochs
n_epoch = 5000

# Batch size during training
batch_size = 128

# Size of z latent vector (i.e. size of generator input)
latent_dim = 100

# Spatial size of training images. All images will be resized to this size
cols, rows = 128, 128

# Number of channels in the training images. For RGB color images this is 3
channels = 3
dim = cols, rows # height, width
in_shape = (cols, rows, channels) # height, width, color

# Learning rate for optimizers
lr = 0.0002

# Beta1 hyperparam for Adam optimizers
beta1 = 0.5

# Number of GPUs available. Use 0 for CPU mode.
ngpu = 1

# plot ncols images in row and nrows images in colomn
nrows, ncols = 3, 4
```

```
def define_discriminator(in_shape=(128,128,3)):
    model = models.Sequential()
    # normal
    model.add(layers.Conv2D(128, (5,5), padding='same', input_shape=in_shape))
    model.add(layers.LeakyReLU(alpha=0.2))
    # downsample to 64x64
    model.add(layers.Conv2D(128, (5,5), strides=(2,2), padding='same'))
    model.add(layers.LeakyReLU(alpha=0.2))
    # downsample to 32x32
    model.add(layers.Conv2D(128, (5,5), strides=(2,2), padding='same'))
    model.add(layers.LeakyReLU(alpha=0.2))
    # downsample to 16x16
    model.add(layers.Conv2D(128, (5,5), strides=(2,2), padding='same'))
    model.add(layers.LeakyReLU(alpha=0.2))
    # downsample to 8x8
    model.add(layers.Conv2D(128, (5,5), strides=(2,2), padding='same'))
    model.add(layers.LeakyReLU(alpha=0.2))
    # classifier
    model.add(layers.Flatten())
    model.add(layers.Dropout(0.4))
    model.add(layers.Dense(1, activation='sigmoid'))
    # compile model
    opt = optimizers.Adam(lr=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
    return model

"""## Generator"""

def define_generator(latent_dim):
    model = models.Sequential()
    # foundation for 8x8 feature maps
    n_nodes = 128*8*8
    model.add(layers.Dense(n_nodes, input_dim=latent_dim))
    model.add(layers.LeakyReLU(alpha=0.2))
    model.add(layers.Reshape((8, 8, 128)))
    # upsample to 16x16
    model.add(layers.Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))
    model.add(layers.LeakyReLU(alpha=0.2))
    # upsample to 32x32
    model.add(layers.Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))
    model.add(layers.LeakyReLU(alpha=0.2))
    # upsample to 64x64
    model.add(layers.Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))
    model.add(layers.LeakyReLU(alpha=0.2))
    # upsample to 128x128
    model.add(layers.Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))
    model.add(layers.LeakyReLU(alpha=0.2))
    # output layer 128x128x3
    model.add(layers.Conv2D(3, (5,5), activation='tanh', padding='same'))
    return model

#input of G
def generate_latent_points(latent_dim, n_samples):
    # generate points in the latent space
    x_input = np.random.randn(latent_dim*n_samples)
    # reshape into a batch of inputs for the network
    x_input = x_input.reshape(n_samples, latent_dim)
    return x_input

# use the generator to generate n fake examples, with class labels
def generate_fake_samples(g_model, latent_dim, n_samples):
    # generate points in latent space
    x_input = generate_latent_points(latent_dim, n_samples)
    # predict outputs
    X = g_model.predict(x_input)
    # create 'fake' class labels (0)
    y = np.zeros((n_samples, 1))
    return X, y

"""## Define GAN model"""

def define_gan(g_model, d_model):
    # make weights in the discriminator not trainable
    d_model.trainable = False
    # connect them
    model = models.Sequential()
    # add generator
    model.add(g_model)
    # add the discriminator
    model.add(d_model)
    # compile model
    opt = optimizers.Adam(lr=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer=opt)
    return model

# retrive real samples
def get_real_samples(dataset, n_samples):
    # choose random instances
    ix = np.random.randint(0, dataset.shape[0], n_samples)
    # retrieve selected images
    X = dataset[ix]
    # set 'real' class labels (1)
    y = np.ones((n_samples, 1))
    return X, y

# create and save a plot of generated images
def show_generated(generated, epoch, nrows=4, ncols=5):
    #[-1,1] -> [0,1]
    #generated = (generated+1)/2
    #generated = (generated[:ncols*nrows]*127.5)+127.5
    #generated = generated*255
    plt.figure(figsize=(10,10))
    for idx in range(nrows*ncols):
        plt.subplot(nrows, ncols, idx+1)
        plt.imshow(generated[idx])
        plt.axis('off')
    plt.savefig('image_at_epoch_{:04d}.png'.format(epoch+1))
    plt.show()

# evaluate the discriminator and plot generated images
def summarize_performance(epoch, g_model, d_model, dataset, latent_dim,
n_samples=100):
    # prepare real samples
    X_real, y_real = get_real_samples(dataset, n_samples)
    # evaluate discriminator on real examples
    _, acc_real = d_model.evaluate(X_real, y_real, verbose=0)
    # prepare fake examples
    x_fake, y_fake = generate_fake_samples(g_model, latent_dim, n_samples)
    # evaluate discriminator on fake examples
    _, acc_fake = d_model.evaluate(x_fake, y_fake, verbose=0)
    # summarize discriminator performance
    print('> Accuracy at epoch %d [real: %.0f%%, fake: %.0f%%]'%(epoch+1, acc_real*100,
acc_fake*100))
    # show plot
    show_generated(x_fake, epoch)

def plot_loss(loss):
    plt.figure(figsize=(10,5))
    plt.title("Generator and Discriminator Loss During Training", fontsize=20)
    plt.plot(loss[0], label="D_real")
    plt.plot(loss[1], label="D_fake")
    plt.plot(loss[2], label="G")
    plt.xlabel("Iteration", fontsize=20); plt.ylabel("Loss", fontsize=20)
    plt.legend(); plt.show()
```

```
def save_models(epoch, g_model, d_model, gan_model):
    g_model.save_weights(f'generator_{epoch}_epochs.h5')
    d_model.save_weights(f'discriminator_{epoch}_epochs.h5')
    gan_model.save_weights(f'gan_{epoch}_epochs.h5')

"""# Train the models"""

def train(g_model, d_model, gan_model, dataset, latent_dim=100, n_epochs=100,
n_batch=128, start_epoch=0):

    start = time.time()
    bat_per_epo = int(dataset.shape[0]/n_batch)
    half_batch = int(n_batch/2)
    loss1, loss2, loss3 = [], [], []
    fake_liste = []

    # manually enumerate epochs
    print('Training Start...')
    for i in range(start_epoch, start_epoch+n_epochs):
        start1 = time.time()
        # enumerate batches over the training set
        for j in range(bat_per_epo):
            # get randomly selected 'real' samples
            X_real, y_real = get_real_samples(dataset, half_batch)
            # update discriminator model weights
            d_loss1, _ = d_model.train_on_batch(X_real, y_real)
            # generate 'fake' examples
            X_fake, y_fake = generate_fake_samples(g_model, latent_dim, half_batch)
            # update discriminator model weights
            d_loss2, _ = d_model.train_on_batch(X_fake, y_fake)
            # prepare points in latent space as input for the generator
            X_gan = generate_latent_points(latent_dim, n_batch)
            # create inverted labels for the fake samples
            y_gan = np.ones((n_batch, 1))
            # update the generator via the discriminator's error
            g_loss = gan_model.train_on_batch(X_gan, y_gan)
            # summarize loss on this batch
            loss1.append(d_loss1); loss2.append(d_loss2); loss3.append(g_loss)

        print('Epoch: {:03d}/{:03d}, Loss: [D_real = {:2.3f}, D_fake = {:2.3f}, G = {:2.3f}],
time: {:s}'\
              .format(i+1,start_epoch+n_epochs,d_loss1,d_loss2,g_loss,
_time(start1,time.time())))
        # evaluate the model performance
        if (epoch + 1) % 100 == 0:
            # Save and show generated images
            summarize_performance(i, g_model, d_model, dataset, latent_dim)

        if (epoch + 1) % 1000 == 0:
            save_models(epoch, g_model, d_model, gan_model)

    print('Total time for training {} epochs is {} sec'.format(n_epochs, _time(start,
time.time())))

    # Show loss curves
    loss = (loss1, loss2, loss3)
    plot_loss(loss)

discriminator = define_discriminator()
generator = define_generator(latent_dim)

# create the gan
gan = define_gan(generator, discriminator)

def load_models(g_model, d_model, gan_model, epoch):
    g_model.load_weights(f'/kaggle/input/gan/tensorflow2/default/1/generator_1000_epochs
(1).h5')
```

```
d_model.load_weights(f'/kaggle/input/gan/tensorflow2/default/1/discriminator_1000_epochs
(1).h5')
    gan_model.load_weights(f'/kaggle/input/gan/tensorflow2/default/1/gan_1000_epochs.h5')

load_models(generator, discriminator, gan, epoch=1000)

# train model
train(generator, discriminator, gan, X_tb, latent_dim, n_epochs=n_epoch,
n_batch=batch_size, start_epoch=0)
```

- **Using GAN to generate images:**

```
def XRayFakeGenerator(g_model=generator, latent_dim =100, n_samples=100,
show_gen=False):
    # generate points in latent space
    x_input = generate_latent_points(latent_dim, n_samples)
    # predict outputs
    X = g_model.predict(x_input)

    # Show the generated images
    if show_gen and n_samples<=30:
        ncols = 5
        nrows = int(n_samples/ncols)
        plt.figure(figsize=(12,10))
        for idx in range(nrows*ncols):
            plt.subplot(nrows, ncols, idx+1)
            plt.imshow(X[idx,:,:]); plt.axis('off')
        plt.show();
    return X

XRay_fake = XRayFakeGenerator(generator, n_samples=20)
```

```
# SAVE TO ZIP FILE
import zipfile
output_path = zipfile.PyZipFile('../working/XRayNormalFake3.zip', mode='w')

XRay_generated = XRayFakeGenerator(n_samples=3000)
for idx in range(XRay_generated.shape[0]):
    img_XRayFake  = XRay_generated[idx,:,:]
    name_XRayFake = 'XRay_generated {:04d}.png'.format(idx)
    imageio.imwrite(name_XRayFake, img_XRayFake)

    output_path.write(name_XRayFake)
    os.remove(name_XRayFake)
output_path.close()
```

- **Calculating HOG features:**

```
from skimage.feature import hog
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score

# Function to extract HOG features from an image
def extract_hog_features(image, resize_dim=(128, 128)):
    # Convert image to grayscale
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Resize image to a standard size
    resized_image = cv2.resize(gray_image, resize_dim)
    # Extract HOG features
    hog_features, hog_image = hog(resized_image, orientations=9, pixels_per_cell=(8, 8),
                                  cells_per_block=(2, 2), block_norm='L2-Hys', visualize=True)
    return hog_features
```

```
# Function to load images and extract HOG features
def load_images_and_extract_hog(folder, label):
    features = []
    labels = []
    for filename in os.listdir(folder):
        img_path = os.path.join(folder, filename)
        image = cv2.imread(img_path)
        if image is not None:
            hog_features = extract_hog_features(image)
            features.append(hog_features)
            labels.append(label)
    return features, labels


# In[4]:



# Load and process Normal images
normal_features, normal_labels = load_images_and_extract_hog(normal_dir, label=0)  # 
Label '0' for Normal

# Load and process TB images
tb_features, tb_labels = load_images_and_extract_hog(tb_dir, label=1)  # Label '1' for 
TBtrad_tb, trad_labels =

trad_tb, trad_labels = load_images_and_extract_hog(traditional_tb_dir, label=1)
gan_tb,gan_labels = load_images_and_extract_hog(gan_tb_dir, label=1)
```

- **Using HOG+RANDOM FOREST Model for Generalization Capability:**

```
rf1_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf1_classifier.fit(X1_train, y1_train)
```

```
# Make predictions on the test set
y1_pred = rf1_classifier.predict(X1_test)

# Evaluate the classifier
accuracy = accuracy_score(y1_test, y1_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')

# Print classification report
print("\nClassification Report:")
print(classification_report(y1_test, y1_pred, target_names=['Normal', 'TB']))


# In[8]:


rf2_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf2_classifier.fit(X2_train, y2_train)

# Make predictions on the test set
y2_pred = rf2_classifier.predict(X2_test)

# Evaluate the classifier
accuracy = accuracy_score(y2_test, y2_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')

# Print classification report
print("\nClassification Report:")
print(classification_report(y2_test, y2_pred, target_names=['Normal', 'TB']))


# In[9]:
```

```
rf3_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf3_classifier.fit(X3_train, y3_train)

# Make predictions on the test set
y3_pred = rf3_classifier.predict(X3_test)

# Evaluate the classifier
accuracy = accuracy_score(y3_test, y3_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')

# Print classification report
print("\nClassification Report:")
print(classification_report(y3_test, y3_pred, target_names=['Normal', 'TB']))



# In[12]:



from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y1_test, y1_pred)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.grid()
plt.show()
```

```
# In[13]:


fpr, tpr, thresholds = roc_curve(y2_test, y2_pred)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.grid()
plt.show()


# In[10]:


from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y3_test, y3_pred)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.grid()
plt.show()
```

- **Training HOG+CNN model:**

```
def create_cnn_model(input_shape):
    return keras.Sequential([
        keras.layers.Conv2D(16, (3, 3), activation='relu', input_shape=input_shape,
name='conv2d_1'),
        keras.layers.MaxPooling2D((2, 2)),
        keras.layers.Conv2D(32, (3, 3), activation='relu', name='conv2d_2'),
        keras.layers.MaxPooling2D((2, 2)),
        keras.layers.Flatten(),
        keras.layers.Dense(32, activation='relu'),
        keras.layers.Dropout(0.5),
    ])

def create_hybrid_model(input_shape, hog_input_shape):
    # CNN branch
    cnn_input = keras.layers.Input(shape=input_shape)
    cnn_model = create_cnn_model(input_shape)
    cnn_output = cnn_model(cnn_input)

    # HOG branch
    hog_input = keras.layers.Input(shape=(hog_input_shape,))
    hog_dense = keras.layers.Dense(32, activation='relu')(hog_input)

    # Combine CNN and HOG features
    combined = keras.layers.concatenate([cnn_output, hog_dense])

    # Output layer
    output = keras.layers.Dense(1, activation='sigmoid')(combined)

    # Create the hybrid model
    hybrid_model = keras.Model(inputs=[cnn_input, hog_input], outputs=output)

    return hybrid_model

input_shape = (128, 128, 1)
hog_input_shape = X_train_hog.shape[1]
hybrid_model = create_hybrid_model(input_shape, hog_input_shape)

hybrid_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.001),
                     loss='binary_crossentropy',
                     metrics=['accuracy'])

history = hybrid_model.fit(
    [X_train.reshape(-1, 128, 128, 1), X_train_hog],
    y_train,
    epochs=10,
    batch_size=32,
    validation_split=0.2,
    verbose=1
)

test_loss, test_accuracy = hybrid_model.evaluate([X_test.reshape(-1, 128, 128, 1),
X_test_hog], y_test, verbose=0)
print(f"Test accuracy: {test_accuracy:.4f}")

predictions = hybrid_model.predict([X_test.reshape(-1, 128, 128, 1), X_test_hog])
predicted_labels = (predictions > 0.5).astype(int).flatten()
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test, predicted_labels))
```
Using Transfer Learning:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPool2D, Flatten, Dense, Dropout,
BatchNormalization
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.preprocessing import image
from time import perf_counter
import seaborn as sns

def printmd(string):
    display(Markdown(string))

tb_path = '/kaggle/input/gan-data/tbgenerated (2)/tb+generated/'
tb_path_list = os.listdir(tb_path)
tb_path_list = [tb_path + f for f in tb_path_list]

normal_path = '/kaggle/input/gan-data/normal_gamma_adjusted/normal_gamma_adjusted/'
normal_path_list = os.listdir(normal_path)
normal_path_list = [normal_path + f for f in normal_path_list]

df = pd.DataFrame({'Path': tb_path_list + normal_path_list})
df['Label'] = df['Path'].apply(lambda x: 0 if 'normal' in x else 1)
df['Label_String'] = df['Label'].apply(lambda x: 'normal' if x == 0 else 'Tuberculosis')

# Shuffle
df = df.sample(frac = 1.0).reset_index(drop = True)

# Display the first lines
pd.options.display.max_colwidth = 200
df.head()

df['Label_String'].value_counts().plot.bar(color = ['red','gray'])
```

```
plt.title('Number of pictures', fontsize = 18)
plt.xticks(rotation = 0, fontsize = 15)
plt.show()

# Display some pictures of the dataset
fig, axes = plt.subplots(nrows=4, ncols=6, figsize=(15, 8),
                        subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
    img = image.load_img(df['Path'].iloc[i])
    ax.imshow(img, cmap = 'gray')
    title = df['Label_String'].iloc[i]
    ax.set_title(title, fontsize = 15, color='white')
plt.tight_layout(pad=0.5)
plt.show()

df_original = df.copy()

# Split into training, test and validation sets
val_index = int(df_original.shape[0]*0.1)

train_df = df_original.iloc[val_index:]
test_df = df_original.iloc[:val_index]

# Display the shapes of the sets
train_df.shape, test_df.shape

train_generator = tf.keras.preprocessing.image.ImageDataGenerator(
    preprocessing_function=tf.keras.applications.mobilenet_v2.preprocess_input,
    validation_split=0.1
)

test_generator = tf.keras.preprocessing.image.ImageDataGenerator(
    preprocessing_function=tf.keras.applications.mobilenet_v2.preprocess_input
```

```
)

train_images = train_generator.flow_from_dataframe(
    dataframe=train_df,
    x_col='Path',
    y_col='Label_String',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=32,
    shuffle=True,
    seed=0,
    subset='training',
    rotation_range=30,
    zoom_range=0.15,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.15,
    horizontal_flip=True,
    fill_mode="nearest"
)

val_images = train_generator.flow_from_dataframe(
    dataframe=train_df,
    x_col='Path',
    y_col='Label_String',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=32,
    shuffle=True,
    seed=0,
    subset='validation',
    rotation_range=30,
    zoom_range=0.15,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.15,
    horizontal_flip=True,
    fill_mode="nearest"
)

test_images = test_generator.flow_from_dataframe(
    dataframe=test_df,
    x_col='Path',
    y_col='Label_String',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=32,
    shuffle=False
)

'''#Load the pretained model
pretrained_model = tf.keras.applications.MobileNetV2(
    input_shape=(224, 224, 3),
    include_top=False,
    weights='imagenet',
    pooling='avg'
)
pretrained_model.trainable = False'''

from tensorflow.keras.applications import Xception

# Load the pretrained Xception model
pretrained_model = Xception(
    input_shape=(224, 224, 3),
    include_top=False,
    weights='imagenet',
    pooling='avg'
)
pretrained_model.trainable = False

inputs = pretrained_model.input

x = tf.keras.layers.Dense(128, activation='relu')(pretrained_model.output)
x = tf.keras.layers.Dense(128, activation='relu')(x)

outputs = tf.keras.layers.Dense(2, activation='softmax')(x)

model = tf.keras.Model(inputs=inputs, outputs=outputs)

model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy', patience = 2,
verbose=1,factor=0.5, min_lr=0.00001)

history = model.fit(
    train_images,
    validation_data=val_images,
    batch_size = 32,
    epochs=5,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(
            monitor='val_loss',
            patience=2,
            restore_best_weights=True),
            learning_rate_reduction
```

```
  ]
```

- **Explainability:**
- **LIME:**

```
def get_cnn_submodel(hybrid_model):
    cnn_model = hybrid_model.get_layer('sequential')  # The CNN model is the Sequential
part
    cnn_input = cnn_model.input
    cnn_output = cnn_model.output  # Output from the CNN part
    cnn_submodel = keras.Model(inputs=cnn_input, outputs=cnn_output)
    return cnn_submodel

# LIME function that explains grayscale images (single-channel)
def explain_image_with_lime(model, img):
    explainer = lime_image.LimeImageExplainer()

    # Explain the predictions for the input grayscale image, using felzenszwalb segmentation
    explanation = explainer.explain_instance(
        img[0],  # The input image (single-channel grayscale)
        model.predict,  # Function that predicts using the CNN submodel
        top_labels=2,  # We want the explanation for the top 2 predicted labels
        hide_color=0,  # Superpixels will be zeroed out when perturbed
        num_samples=1000,  # Number of perturbations
        segmentation_fn=lambda x: felzenszwalb(x, scale=100, sigma=0.5, min_size=50)  # Use
felzenszwalb instead of quickshift
    )

    return explanation

# Load an example grayscale image and preprocess it
```

```
img_path = '/kaggle/input/tb-gan-generated-5000-epochs/tb+generated/XRay_generated
0013.png'  # Replace with your own image path
img = tf.keras.preprocessing.image.load_img(img_path, target_size=(128, 128),
color_mode='grayscale')
img_array = tf.keras.preprocessing.image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension

# Get the CNN submodel from the hybrid model
cnn_submodel = get_cnn_submodel(hybrid_model)

# Explain the image using LIME on the CNN submodel
explanation = explain_image_with_lime(cnn_submodel, img_array)

# Get the explanation for the top class
temp, mask = explanation.get_image_and_mask(explanation.top_labels[0],
positive_only=True, num_features=5, hide_rest=False)

# Reshape the mask to match the original grayscale image
mask = np.reshape(mask, (128, 128))  # Adjust according to your image size

# Display the image with the LIME explanation overlaid with a 'jet' heatmap
plt.imshow(img_array[0, :, :, 0], cmap='gray')  # Original grayscale image
plt.imshow(mask, cmap='jet', alpha=0.5)  # Heatmap overlay with 'jet' colormap
plt.axis('off')
```

- **GRADCAM:**

```
def get_img_array(img_path, size):
   img = tf.keras.preprocessing.image.load_img(img_path, target_size=size)
   array = tf.keras.preprocessing.image.img_to_array(img)
   # We add a dimension to transform our array into a "batch"
   # of size "size"
   array = np.expand_dims(array, axis=0)
```

```
    return array

def make_gradcam_heatmap(img_array, model, last_conv_layer_name, pred_index=None):
    # First, we create a model that maps the input image to the activations
    # of the last conv layer as well as the output predictions
    grad_model = tf.keras.models.Model(
        [model.inputs], [model.get_layer(last_conv_layer_name).output, model.output]
    )

    # Then, we compute the gradient of the top predicted class for our input image
    # with respect to the activations of the last conv layer
    with tf.GradientTape() as tape:
        last_conv_layer_output, preds = grad_model(img_array)
        if pred_index is None:
            pred_index = tf.argmax(preds[0])
        class_channel = preds[:, pred_index]

    # This is the gradient of the output neuron (top predicted or chosen)
    # with regard to the output feature map of the last conv layer
    grads = tape.gradient(class_channel, last_conv_layer_output)

    # This is a vector where each entry is the mean intensity of the gradient
    # over a specific feature map channel
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

    # We multiply each channel in the feature map array
    # by "how important this channel is" with regard to the top predicted class
    # then sum all the channels to obtain the heatmap class activation
    last_conv_layer_output = last_conv_layer_output[0]
    heatmap = last_conv_layer_output @ pooled_grads[..., tf.newaxis]
    heatmap = tf.squeeze(heatmap)

    # For visualization purpose, we will also normalize the heatmap between 0 & 1
    heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)
    return heatmap.numpy()

def save_and_display_gradcam(img_path, heatmap, cam_path="cam.jpg", alpha=0.4):
    # Load the original image
    img = tf.keras.preprocessing.image.load_img(img_path)
    img = tf.keras.preprocessing.image.img_to_array(img)

    # Rescale heatmap to a range 0-255
    heatmap = np.uint8(255 * heatmap)

    # Use jet colormap to colorize heatmap
    jet = cm.get_cmap("jet")

    # Use RGB values of the colormap
    jet_colors = jet(np.arange(256))[:, :3]
    jet_heatmap = jet_colors[heatmap]

    # Create an image with RGB colorized heatmap
    jet_heatmap = tf.keras.preprocessing.image.array_to_img(jet_heatmap)
    jet_heatmap = jet_heatmap.resize((img.shape[1], img.shape[0]))
    jet_heatmap = tf.keras.preprocessing.image.img_to_array(jet_heatmap)

    # Superimpose the heatmap on original image
    superimposed_img = jet_heatmap * alpha + img
    superimposed_img = tf.keras.preprocessing.image.array_to_img(superimposed_img)

    # Save the superimposed image
    superimposed_img.save(cam_path)

    # Display Grad CAM
#     display(Image(cam_path))

    return cam_path
```

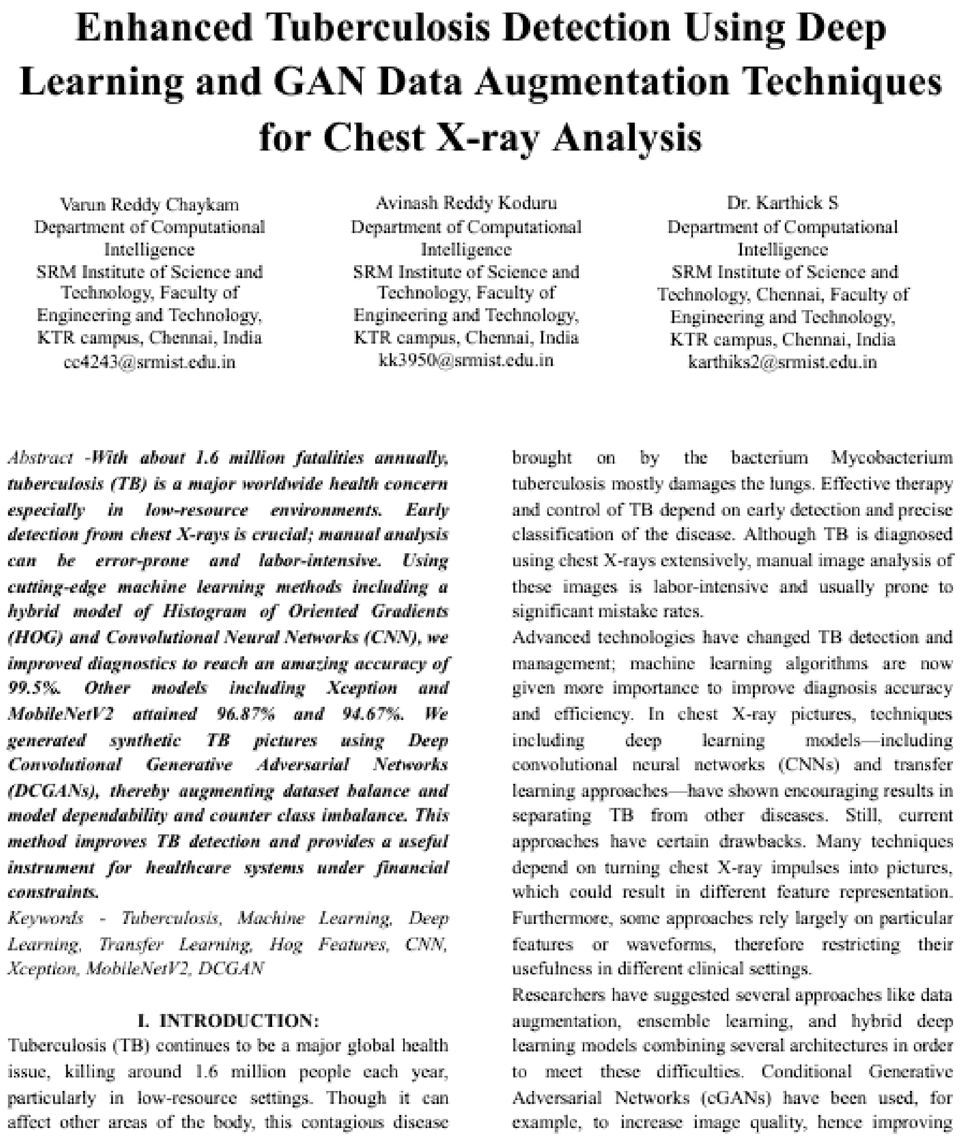
```
preprocess_input = tf.keras.applications.mobilenet_v2.preprocess_input
decode_predictions = tf.keras.applications.mobilenet_v2.decode_predictions

last_conv_layer_name = "conv2d_3"
img_size = (224,224)

# Remove last layer's softmax
model.layers[-1].activation = None

# Display the part of the pictures used by the neural network to classify the pictures
nrows = 10
fig, axes = plt.subplots(nrows=nrows, ncols=2, figsize=(15, 5 * nrows),
                    subplot_kw={'xticks': [], 'yticks': []})

i = 0
for i, nrow in enumerate(range(nrows)):
    img_path = test_df.Path.iloc[i]
    title = f"True: {test_df.Label_String.iloc[i]}\nPredicted: {pred[i]}"

    # Original Picture
    img = image.load_img(img_path)
    axes[nrow,0].imshow(img)
    axes[nrow,0].set_title('ORIGINAL PICTURE\n' + title)
    # Calculate Grad-CAM class activation
    img_array = preprocess_input(get_img_array(img_path, size=img_size))
    heatmap = make_gradcam_heatmap(img_array, model, last_conv_layer_name)
    cam_path = save_and_display_gradcam(img_path, heatmap)
    img = plt.imread(cam_path)
    axes[nrow,1].imshow(img)
    axes[nrow,1].set_title('GRAD-CAM CLASS ACTIVATION\n' + title)
plt.tight_layout()
plt.show()
```

APPENDIX B
CONFERENCE PRESENTATION

- Conference Presentation is under process in upcoming days.

APPENDIX C
PUBLICATION DETAILS

- Our paper titled "**Enhanced Tuberculosis Detection Using Deep Learning and GAN Data Augmentation Techniques for Chest X-ray Analysis**" is under the Submission Process.



Enhanced Tuberculosis Detection Using Deep Learning and GAN Data Augmentation Techniques for Chest X-ray Analysis

Varun Reddy Chaykam
Department of Computational Intelligence
SRM Institute of Science and Technology, Faculty of Engineering and Technology, KTR campus, Chennai, India
cc4243@srmist.edu.in

Avinash Reddy Koduru
Department of Computational Intelligence
SRM Institute of Science and Technology, Faculty of Engineering and Technology, KTR campus, Chennai, India
kk3950@srmist.edu.in

Dr. Karthick S
Department of Computational Intelligence
SRM Institute of Science and Technology, Chennai, Faculty of Engineering and Technology, KTR campus, Chennai, India
karthiks2@srmist.edu.in

Abstract -***With about 1.6 million fatalities annually, tuberculosis (TB) is a major worldwide health concern especially in low-resource environments. Early detection from chest X-rays is crucial; manual analysis can be error-prone and labor-intensive. Using cutting-edge machine learning methods including a hybrid model of Histogram of Oriented Gradients (HOG) and Convolutional Neural Networks (CNN), we improved diagnostics to reach an amazing accuracy of 99.5%. Other models including Xception and MobileNetV2 attained 96.87% and 94.67%. We generated synthetic TB pictures using Deep Convolutional Generative Adversarial Networks (DCGANs), thereby augmenting dataset balance and model dependability and counter class imbalance. This method improves TB detection and provides a useful instrument for healthcare systems under financial constraints.***

Keywords - Tuberculosis, Machine Learning, Deep Learning, Transfer Learning, Hog Features, CNN, Xception, MobileNetV2, DCGAN

I. INTRODUCTION:

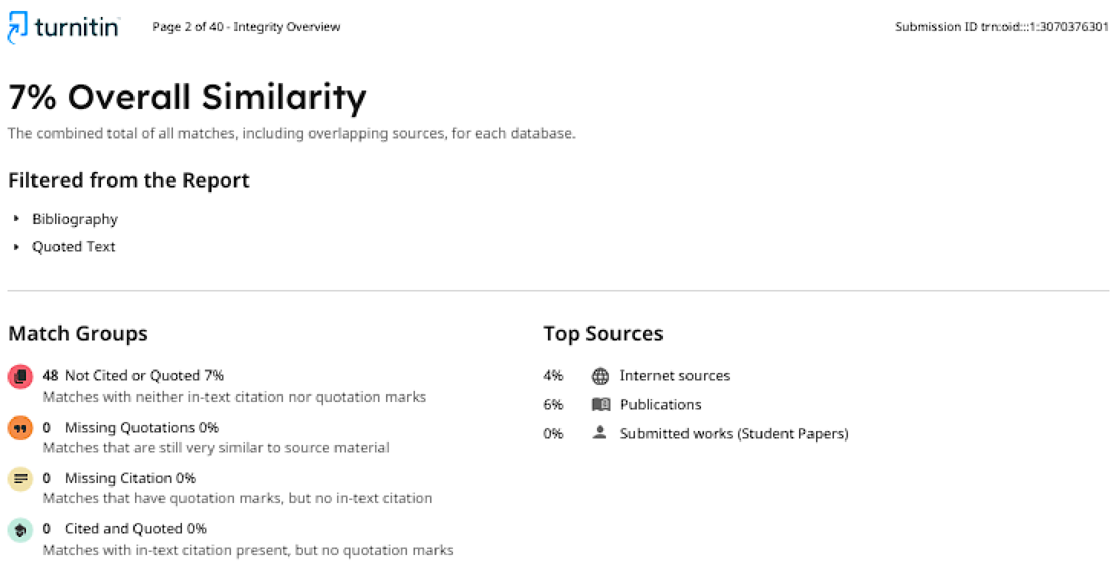
Tuberculosis (TB) continues to be a major global health issue, killing around 1.6 million people each year, particularly in low-resource settings. Though it can affect other areas of the body, this contagious disease brought on by the bacterium Mycobacterium tuberculosis mostly damages the lungs. Effective therapy and control of TB depend on early detection and precise classification of the disease. Although TB is diagnosed using chest X-rays extensively, manual image analysis of these images is labor-intensive and usually prone to significant mistake rates.

Advanced technologies have changed TB detection and management; machine learning algorithms are now given more importance to improve diagnosis accuracy and efficiency. In chest X-ray pictures, techniques including deep learning models—including convolutional neural networks (CNNs) and transfer learning approaches—have shown encouraging results in separating TB from other diseases. Still, current approaches have certain drawbacks. Many techniques depend on turning chest X-ray impulses into pictures, which could result in different feature representation. Furthermore, some approaches rely largely on particular features or waveforms, therefore restricting their usefulness in different clinical settings.

Researchers have suggested several approaches like data augmentation, ensemble learning, and hybrid deep learning models combining several architectures in order to meet these difficulties. Conditional Generative Adversarial Networks (cGANs) have been used, for example, to increase image quality, hence improving

APPENDIX D
PLAGIARISM REPORT

Below is our Plagiarism Report from Turnitin and we got 7% Similarity.



turnitin Page 2 of 40 - Integrity Overview Submission ID trn:oid:::1:3070376301

7% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Filtered from the Report

- Bibliography
- Quoted Text

Match Groups

- 48 Not Cited or Quoted 7%
  Matches with neither in-text citation nor quotation marks
- 0 Missing Quotations 0%
  Matches that are still very similar to source material
- 0 Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- 0 Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

4% Internet sources
6% Publications
0% Submitted works (Student Papers)

Fig. D.1 Plagiarism Report