Tuberculosis Detection Using Deep Learning

A project report

submitted in partial fulfillment of the requirements

for the award of the degree of

Bachelor of Technology

in

Computer Science & Engineering

Submitted by

Aman Yadav (Roll No: 2002840100010)

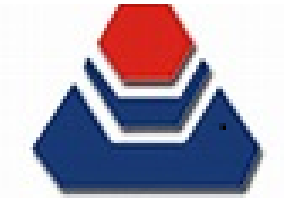
Anchal Pandey (Roll No: 2002840100014)

Harshit Mishra (Roll No: 2002840100059)

Vinayika Krishna (Roll No: 2002840100176)

Under the Guidance of

Mr. Anil Singh



Department of Computer Science & Engineering

UNITED INSTITUTE OF TECHNOLOGY , PRAYAGRAJ

Uttar Pradesh 211010, INDIA.

(Affiliated to Dr. A.P.J. Abdul Kalam Technical University, Lucknow)

2023-2024

Declaration of Academic Ethics

We declare that this written submission represents our ideas in our own words. Where others' ideas or words have been included, we have adequately cited and referenced the original sources. We declare that we have properly and accurately acknowledged all sources used in the production of this project report.

We also declare that we have adhered to all principles of academic honesty and integrity. We have not misrepresented, fabricated, or falsified any idea, data, fact, or source in our submission. We understand that any violation of these principles is unacceptable.

Above actions will be cause for disciplinary action by the Institute and can also result in penal action from the sources that have not been properly cited or from whom proper permission has not been obtained when necessary.

Date: May 21, 2024

Aman Yadav

Anchal Pandey

Harshit Mishra

Vinayika Krishna

Certificate from the Project Guide

This is to certify that the work incorporated in the project report entitled **"Tuberculosis Detection using Deep Learning"** is a record of work carried out by Aman Yadav (2002840100010), Anchal Pandey (2002840100014), Harshit Mishra (2002840100059) and Vinayika Krishna (2002840100176), under the guidance and supervision for the award of Degree of Bachelor of Technology in Computer Science & Engineering.

To the best of my knowledge and belief, the project report Embodies the work of the candidates themselves has duly been completed fulfills the requirements of the ordinance relating to the Bachelor of Technology Degree of the University is up to the desired standards both in respect of contents and language for being referred to the examiners.

This work has not been submitted in part or in full for a degree, a diploma, or a fellowship to any other university or institute. Whenever contributions of others are involved, every effort is made to indicate this clearly, with due acknowledgment of collaborative research and discussions. This thesis is a bonafide record of original work done by me and all sources listed within have been detailed in the bibliography.

Date: May 21, 2024

Mr. Anil Singh

This project work as mentioned above is hereby being recorded and forwarded for examination and evaluation.

Date: May 21, 2024

Head of the Department
Computer Science and Engineering

Certificate from External Examiner

This is to certify that the project report entitled **"Tuberculosis Detection using Deep Learning"** which is submitted by **Aman Yadav (2002840100010)**,**Anchal Pandey (2002840100014)**, **Harshit Mishra (2002840100059)** and **Vinayika Krishna (2002840100176)** has been examined by the undersigned as a part of the examination for the award of Degree of Bachelor of Technology in Computer Science and Engineering.

Internal Examiner **External Examiner**

Date: **Date:**

Acknowledgement

We would like to express our gratitude to our guide **Mr. Anil Singh** and **Head of Department of Computer Science and Engineering** for their unwavering guidance and support throughout the research process, encouragement, insightful suggestions, and mentorship throughout the entirety of our research endeavor and for granting us this wonderful opportunity to be part of this technical project. Their expertise and dedication were instrumental in shaping this project.

We also wish to acknowledge and commend the efforts of our dear friends, whose constructive feedback and unwavering encouragement served as constant pillars of strength throughout the research process. Their support played an integral role in our journey towards achieving our goals.

Additionally, we extend our appreciation to the library staff for their assistance in accessing research materials which contributed to our project work and maintained a friendly atmosphere in the lab.

Also, we would like to express our appreciation to all those who have supported and contributed to the completion of this project. We are also grateful to each of our research team members, for their hard work and dedication. We worked together to collect data, analyze results, and write the research paper.

Finally, this project truly reflects the collective effort of many, and we recognize those who supported and contributed to the completion of this endeavor. It has been possible through their collective contributions that this project has come to fruition.

Abstract

We aim to develop a deep learning model for classifying chest X-ray images to diagnose tuberculosis (TB) using labeled data for TB-positive cases, other respiratory diseases, and normal cases, the model predicts the presence of TB or other conditions. Utilizing a convolutional neural network (CNN) trained on diverse chest X-ray data, rigorous preprocessing, and data augmentation techniques enhance model generalization. Our objectives are to develop a deep learning model to classify chest X-ray images for TB and Non-TB cases and to train the model to predict TB or other conditions using labeled data for that we utilized CNN algorithm.

This study delves into the implementation of tuberculosis detection using Convolutional Neural Networks (CNN), highlighting its efficacy in medical imaging tasks. We begin by discussing the rationale behind selecting CNN architecture, emphasizing its ability to capture spatial hierarchies and patterns in medical images. The dataset preparation section provides insights into the dataset used for training and evaluation, detailing preprocessing steps such as resizing, normalization, and augmentation. We then elaborate on the CNN model architecture tailored for tuberculosis detection, including convolutional layers, pooling layers, activation functions (e.g., ReLU), dropout layers, and the final output layer. The training procedure is detailed, specifying hyperparameters, the training data feeding process, loss function computation (e.g., binary cross-entropy), and model parameter updates via backpropagation. Evaluation metrics are discussed, including accuracy, precision, recall, F1-score, and area under the ROC curve (AUC), highlighting their significance in assessing model performance. Finally, we present the results of the tuberculosis detection experiments, including quantitative metrics and qualitative analysis, addressing challenges encountered, proposing areas for improvement, and outlining potential future research directions. This study underscores the potential of deep learning for accurate and efficient tuberculosis diagnosis, laying the groundwork for future advancements in the field.

This research investigates the implementation of a deep learning approach and the study begins by examining the reasons for selecting CNN architecture, noting its ability to effectively capture complex spatial hierarchies and patterns in medical images, which are crucial for accurate TB diagnosis.

We utilize a comprehensive dataset comprising chest X-ray images labeled with TB-positive, other respiratory conditions, and normal cases. Detailed preprocessing steps, including image resizing, normalization, and augmentation, are applied to enhance data quality and model performance. These steps ensure that the model is trained on high-quality data.

The CNN model architecture is meticulously designed, featuring multiple convolutional layers, pooling layers, and activation functions such as ReLU and sigmoid, along with dropout layers for regularization. These components work together to extract and learn intricate features from the input images.

The training procedure is comprehensively outlined, with a focus on key hyperparameters like learning rate, batch size, number of epochs, and the Adam optimizer. The training process involves feeding the preprocessed data into the model, computing the binary cross-entropy loss, and updating model parameters through backpropagation. These steps are crucial for optimizing the model and ensuring its robustness in detecting TB.

To evaluate the model's performance, we employ a range of metrics including accuracy, precision, recall, F1-score, and area under the ROC and AUC curve. These metrics provide a comprehensive assessment of the model's effectiveness in distinguishing between TB and non-TB cases. The evaluation results are presented with both quantitative metrics and qualitative analysis, demonstrating the model's high performance compared to traditional diagnostic methods.

In the results and analysis section, we detail the outcomes of the experiments conducted, highlighting the model's superior performance in detecting TB. We discuss any challenges encountered during the implementation, propose potential improvements, and suggest directions for future research to further enhance TB detection accuracy and efficiency.

In conclusion, this study demonstrates the significant potential of CNN-based deep learning models for the timely and accurate diagnosis of tuberculosis using chest X-ray images. The findings lay a solid foundation for future advancements in the application of deep learning techniques in medical diagnostics, aiming to improve patient outcomes and streamline TB diagnosis processes on a global scale.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Tuberculosis (TB) remains a significant global health challenge, with millions of new cases reported each year 1. Early and accurate detection of TB is crucial for effective disease management and prevention of its spread. Traditional methods of TB diagnosis often involve time-consuming and labor-intensive processes, such as sputum smear microscopy or culture-based techniques, which may delay timely intervention. In recent years, the integration of deep learning techniques into medical diagnostics has shown great promise, offering a faster and more efficient approach to TB detection. Deep learning is a subset of artificial intelligence that leverages neural networks to automatically learn patterns and features from large datasets. In the context of TB detection, deep learning models can be trained on diverse medical imaging data, such as chest X-rays or computed tomography (CT) scans, to identify characteristic patterns associated with TB infection.

The advantages of using deep learning for TB detection include its ability to handle complex and high-dimensional medical data, adapt to diverse imaging modalities, and continuously improve its performance with additional training data. By automating the analysis of medical images, deep learning models have the potential to enhance the speed and accuracy of TB diagnosis, particularly in resource-limited settings.

This introduction sets the stage for exploring the role of deep learning in revolutionizing TB detection. By harnessing the power of advanced computational methods, we can strive towards faster, more reliable, and cost-effective solutions that contribute to the global efforts to combat tuberculosis. This paper will delve into the current state of research, methodologies, and challenges in the application of deep learning for TB detection, with the ultimate goal of advancing our understanding and capabilities in the fight against this infectious disease.

Problem Statement

Create a deep learning model for automated tuberculosis detection in chest X-ray images, focusing on effective feature extraction and selection techniques for image analysis.

Proposed System

Tuberculosis Detection using deep learning employs a convolutional neural network trained on diverse chest X-ray and CT scan data for tuberculosis detection. Rigorous preprocessing and data augmentation enhance model generalization, with evaluation metrics confirming its superior performance. Comparative analyses showcase advantages over traditional methods, emphasizing Tuberculosis Detection potential for accurate and scalable tuberculosis diagnosis.

Chapter 2

Conceptual Background

2.1 Algorithm

This project's conceptual foundation is centered on Convolutional Neural Networks (CNNs), specialized deep learning algorithms designed for feature extraction from image data. CNNs, with their hierarchical structure, capture complex patterns and relationships within images, making them ideal for tasks like image classification, object detection, and data preprocessing. By leveraging CNNs, the project aims to enhance the accuracy and efficiency of image-based analysis. The multiple layers of convolutions and pooling in CNNs enable the model to progressively learn higher-level features from raw pixel data, which is crucial for recognizing intricate details and variations in images.

In addition to CNNs, the project follows a structured Software Development Life Cycle (SDLC) to ensure systematic development and implementation. The SDLC framework provides a comprehensive approach to planning, designing, building, testing, and deploying the deep learning model. The methodology includes rigorous data preprocessing techniques, model training, and validation strategies. Preprocessing steps involve resizing, normalizing, and augmenting the dataset to improve model performance and generalization. Training the model involves advanced optimization algorithms and hyperparameter tuning, while validation evaluates the model's performance on unseen data. This combination of CNNs, SDLC, and a well-defined methodology lays a strong foundation for achieving the project's objectives, aiming to develop a highly effective and reliable deep learning model for image analysis and classification tasks.

2.1.1 Algorithm Used

Algorithm 1 Tuberculosis Detection Algorithm using CNN

1: **Input:** Chest X-ray images dataset D
2: **Output:** Trained TB detection model M
3: Load the chest X-ray images dataset D
4: Preprocess the images in D (e.g., resize, normalize)
5: Split D into training, validation, and test sets
6: Define the architecture of the convolutional neural network (CNN)
7: Train the CNN on the training set using backpropagation
8: Evaluate the trained CNN on the validation set
9: Tune hyperparameters based on validation performance
10: Test the final model on the test set
11: **Return** Trained TB detection model M

2.1.2 CNN

Convolutional Neural Network (CNN) is a type of deep learning algorithm that is especially useful for image recognition and processing tasks. It is made up of multiple layers, including Convolutional, pooling, and fully connected layers. The architecture of CNNs is inspired by the visual processing in the human brain, and they are useful for capturing hierarchical patterns and spatial dependencies within images.

CNNs are trained using a large dataset of labeled images, where the network learns to recognize patterns and features that are associated with specific objects or classes. They are widely used in areas such as image classification, object detection, facial recognition, and medical image analysis.

Layers of a Convolutional Neural Network include:

I. **Convolutional Layers:** These layers apply convolutional operations to input images, using filters to detect features such as edges, textures, and more complex patterns. Convolutional operations help preserve the spatial relationships between pixels.

II. **Pooling Layers:** Pooling layers downsample the spatial dimensions of the input, reduc-

ing the computational complexity and the number of parameters in the network. Max pooling is a common pooling operation, selecting the maximum value from a group of neighboring pixels.

III. **Activation Functions:** Non-linear activation functions, such as Rectified Linear Unit (ReLU), introduce non-linearity to the model, allowing it to learn more complex relationships in the data.

IV. **Fully Connected Layers:** These layers are responsible for making predictions based on the high-level features learned by the previous layers. They connect every neuron in one layer to every neuron in the next layer.

2.1.3 Activation Function

An activation function, in the context of neural networks and deep learning, is a mathematical operation applied to the output of each neuron in a neural network layer. It determines whether the neuron should be activated or not based on whether the input received from the previous layer is relevant to the current task.

Activation functions play a crucial role in determining the output of a neural network and can greatly influence the network's performance and training dynamics
Following are the activation functions used in this project:

I. **ReLu:**

a) ReLU, which stands for Rectified Linear Unit. It's a simple but powerful mathematical operation applied to the output of each neuron in a neural network layer. ReLU function is defined as f(x) = max(0,x). It only returns a positive output or zero.

b) It is majorly used in the hidden layers of the neural network since it brings sparsity to the data.

c) This leads to more efficient representation of high-dimensional data.

d) ReLU helps in reducing the gradient during backpropagation and hence leads to better and faster learning.

II. **Sigmoid:**

a) Sigmoid function used to introduce non-linearity into the output of a neural network layer. It squashes the input values to the range between 0 and 1, making it suitable for binary classification tasks where the goal is to predict probabilities.

b) The smoothness of the sigmoid function allows for stable and efficient optimization during training.

c) By constraining the output within a bounded range, the sigmoid function prevents the activation values from becoming too large or too small, which can lead to numerical instability.

2.1.4 Optimization Function

The optimization function helps improve the performance of the model. By calculating the difference between the expected and actual outputs of a model. The optimizer's job is to determine which combination of the neural network's weights and biases will give it the best chance to generate accurate predictions. These optimization techniques play a critical role in the training of neural networks, as they help improve the model by adjusting its parameters to minimize the loss of function value.

In this project, ADAM optimization algorithm has been implemented.

ADAM: It is short for Adaptive Moment Estimation, is an advanced optimization technique that is commonly used in deep learning to train neural network models. It is an extension to stochastic gradient descent. As Stochastic gradient descent maintains a single learning rate (termed alpha) for all weight updates and the learning rate does not change during training. Whereas in ADAM the learning rate is maintained for each network weight (parameter) and separately adapted as learning unfolds. It combines the benefits of AdaGrad (Adaptive Gradient Algorithm), Momentum and RMSProp (Root Mean Square Propagation) optimization techniques.

Adam is relatively easy to configure where the default configuration parameters do well on most problems.

The Adam optimization algorithm computes adaptive learning rates and updates the model parameters using the following steps:

a) Initialize the first and second moment variables to zero.

b) Compute the gradients of the loss function with respect to the model parameters.

c) Update the first moment estimates (mean) using exponential decay.

d) Update the second moment estimates (uncentered variance) using exponential decay.

e) Compute bias-corrected estimates of the moments to mitigate the initialization bias.

f) Update the parameters using the corrected moment estimates and the adaptive learning rates.

2.2 SDLC

The software development lifecycle is a conceptual framework utilized in project management to outline the sequential phases encompassing an information systems development project, commencing with preliminary feasibility studies and culminating in the ongoing maintenance of a completed application. Its purpose is to establish and devise the diverse models governing the software development process.

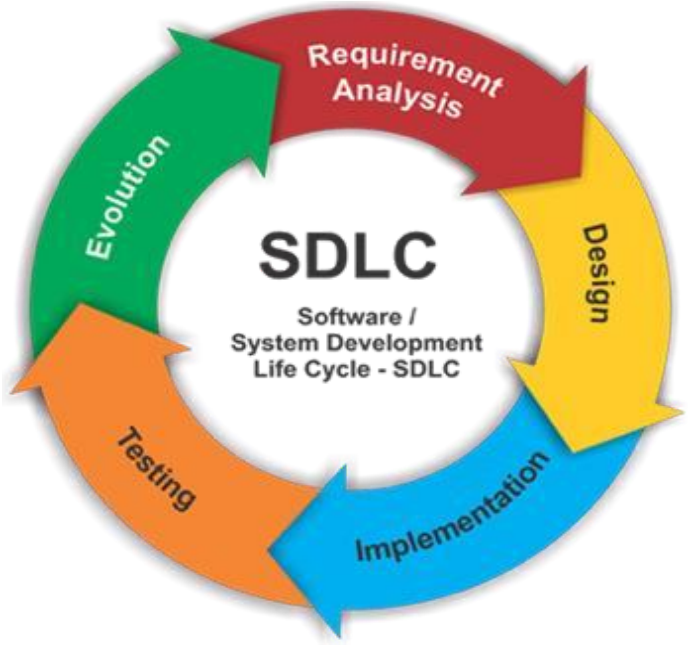


Figure 2.1: SDLC Life Cycle

2.3 Types of SDLC

There are different software development lifecycle model which are used for software development phase.

a) Waterfall Model

b) RAD Model

c) Spiral Model

d) V-Model

e) Incremental Model

f) Agile Model

g) Iterative Model

2.4 Selection of Methodology

For this project, Crisp-DM Methodology has been selected. The reason for opting for the Crisp-DM (Cross-Industry Standard Process for Data Mining) provides a structured framework for the end-to-end development of data-driven solutions, aligning well with the complex nature of chest x-ray images of tuberculosis-affected individuals. Its iterative approach allows for seamless integration of deep learning models into the workflow. Crisp-DM's flexibility accommodates the evolving nature of medical data, fostering adaptability and robustness in the development process.

2.5 Implemented Methodology

The reason for the utilization of this methodology is that it is simple and easy to implement in tuberculosis detection with deep learning due to its structured approach, facilitating systematic development, seamless integration of deep learning models, and adaptability to evolving medical data. The methodology encourages thorough documentation at each phase, which improves transparency and makes it easier to track progress, CRISP-DM's structured approach helps in identifying and mitigating risks early in the project lifecycle. By breaking the project into manageable phases, potential challenges can be addressed proactively.

2.6 Crisp-DM

The CRISP-DM [**?**] (Cross-Industry Standard Process for Data Mining) methodology is a structured framework used for guiding data mining and machine learning projects. It consists of six distinct phases: Business Understanding, Data Understanding, Data Preparation, Modeling,

Figure 2.2: Crisp-DM Life cycle

Evaluation, and Deployment. CRISP-DM [**?**] emphasizes iterative development and collaboration between business stakeholders and data scientists to ensure that data mining projects are aligned with business goals and requirements.

2.6.1 Phase 1: Business Understanding

The primary objective of the Business Understanding phase is to understand the project objectives and requirements from a business perspective. This phase involves understanding the business problem, defining the project goals, and developing a clear understanding of how data mining can provide solutions to meet those goals.

Key Activities:

a) **Identify Business Objectives:** Start by understanding the business problem that needs to be addressed. What are the business objectives, goals, and constraints? It's crucial to

have a clear understanding of what the stakeholders aim to achieve.

b) **Assess Situation:** Gain insights into the current situation by conducting interviews, discussions, and reviewing relevant documentation. Understand the context in which the project operates, including the business environment, industry trends, and challenges.

c) **Determine Data Mining Goals:** Define the specific data mining objectives that align with the business objectives. These goals should be actionable and measurable. For example, if the business objective is to reduce customer churn, the data mining goal might be to develop a predictive model to identify customers at risk of churning.

d) **Produce Project Plan:** Develop a high-level project plan outlining the tasks, resources, timelines, and deliverables for the entire data mining project. Identify potential risks and constraints that could impact the project's success.

2.6.2 Phase 2: Data Understanding

The primary objective of the Data Understanding phase is to collect, explore, and understand the data that will be used for analysis. This phase involves gathering information about the data sources, assessing data quality, and identifying initial insights that may inform subsequent phases of the project.

Key Activities:

a) **Collect Initial Data:** Gather the relevant data from various sources identified in the Business Understanding phase. This may include databases, files, APIs, or other data repositories. Document the data sources and their characteristics.

b) **Describe Data:** Examine the structure, format, and content of the data. Document the attributes, data types, and relationships between different variables. Identify any missing values, outliers, or anomalies that may affect the quality of the data.

c) **Explore Data:** Perform exploratory data analysis (EDA) to gain insights into the data. This involves summarizing the distribution of variables, visualizing patterns and trends,

and identifying potential relationships between variables. Explore data using statistical methods, visualization techniques, and domain knowledge.

d) **Verify Data Quality:** Assess the quality of the data by checking for completeness, consistency, accuracy, and timeliness. Address any data quality issues through data cleaning, preprocessing, or imputation techniques. Document the findings and decisions made regarding data quality.

e) **Identify Data Issues:** Identify any potential challenges or limitations associated with the data. This may include data privacy concerns, data governance issues, or constraints related to data access and availability. Document these issues for consideration in subsequent phases.

2.6.3 Phase 3: Data Preparation

The primary objective of the Data Preparation phase is to transform the raw data collected and understood in the previous phases into a clean, integrated, and suitable format for analysis. This involves preprocessing, cleaning, and transforming the data to address quality issues, handle missing values, and prepare features for modeling.

Key Activities:

a) **Select Data:** Select the relevant subset of data required for analysis based on the project objectives and data understanding gained in previous phases. This may involve filtering, sampling, or aggregating the data to focus on specific subsets or time periods.

b) **Preprocess Data:** Clean and preprocess the data to address quality issues identified during the Data Understanding phase. This may include handling missing values, outliers, duplicates, and inconsistencies in the data. Common preprocessing techniques include imputation, outlier detection, and data normalization or standardization.

c) **Construct Features:** Create new features or variables from the existing data that may enhance the predictive power of the models. Feature engineering techniques may involve

transforming, combining, or encoding the existing variables to extract meaningful information or patterns.

d) **Transform Data:** Transform the data into a format suitable for modeling. This may involve encoding categorical variables, scaling numerical variables, and splitting the data into training, validation, and test sets. Ensure that the data is formatted according to the requirements of the modeling techniques to be applied.

e) **Integrate Data:** Integrate data from multiple sources or sources with different formats to create a unified dataset for analysis. This may involve merging datasets, joining tables, or aggregating data from different sources to create a comprehensive dataset.

2.6.4 Phase 4: Modeling

The primary objective of the Modeling phase is to develop predictive or descriptive models that address the business objectives identified in the earlier phases. This involves selecting appropriate modeling techniques, building and calibrating models, and assessing their performance.

Key Activities:

a) **Select Modeling Techniques:** Choose the appropriate modeling techniques based on the nature of the problem, the available data, and the business objectives. Common modeling techniques include regression, classification, clustering, association rule mining, time series analysis, etc.

b) **Design Experiments:** Define the experimental setup for building and evaluating the models. This includes splitting the data into training and validation sets, defining performance metrics, and selecting evaluation procedures such as cross-validation or holdout validation.

c) **Build Models:** Build predictive or descriptive models using the selected techniques. This may involve training machine learning algorithms, fitting statistical models, or applying other analytical methods to the prepared data.

d) **Calibrate Models:** Fine-tune the parameters of the models and optimize their performance. This may involve hyperparameter tuning, feature selection, model ensemble techniques, or other methods to improve model accuracy and generalization ability.

e) **Assess Model Performance:** Evaluate the performance of the models using appropriate evaluation metrics. This may include measures such as accuracy, precision, recall, F1-score, ROC curve, AUC, etc., depending on the type of model and the nature of the problem.

2.6.5 Phase 5: Evaluation

The primary objective of the Evaluation phase is to assess the performance of the models developed in the previous phase and determine their suitability for addressing the business objectives. This involves evaluating model performance, validating results, and identifying areas for improvement.

Key Activities:

a) **Evaluate Model Performance:** Assess the performance of the predictive models using appropriate evaluation metrics. This may include measures such as accuracy, precision, recall, F1-score, ROC curve, AUC, etc., depending on the nature of the problem and the modeling techniques used.

b) **Validate Results:** Validate the results of the models against the business objectives and requirements. Determine whether the models meet the specified criteria for success and whether they provide actionable insights that can be used to make informed decisions.

c) **Assess Generalization:** Evaluate the generalization ability of the models by testing them on unseen or holdout data. This helps to assess whether the models can generalize well to new, unseen instances and perform effectively in real-world scenarios.

d) **Compare Models:** Compare the performance of different models developed during the Modeling phase. Identify the strengths and weaknesses of each model and select the best-performing model(s) for further analysis or deployment.

e) **Iterate and Refine:** Iterate on the modeling process as needed based on the evaluation results. This may involve refining the models, adjusting parameters, or incorporating feedback from stakeholders to improve model performance.

2.6.6 Phase 6: Deployment

The primary objective of the Deployment phase is to implement the predictive or descriptive models in a production environment where they can be used to support decision-making processes, automate tasks, or enhance business operations. This involves integrating the models into operational systems, monitoring their performance, and ensuring their ongoing maintenance and support.

Key Activities:

a) **Plan Deployment Strategy:** Create a deployment plan detailing steps, resources, and timeline for model deployment. Address infrastructure, integration, data security, and user training.

b) **Implement Models:** Integrate models into operational systems for decision-making or automation. Develop APIs, embed models into applications, or deploy to cloud platforms.

c) **Test Deployment:** Conduct thorough testing of the deployed models to ensure that they perform as expected in the production environment. Validate model predictions against real-world data and assess their impact on business outcomes. Address any issues or discrepancies identified during testing.

d) **Monitor Performance:** Implement real-time monitoring for deployed models, tracking key indicators like accuracy, reliability, latency, and resource use. Use alert systems to notify stakeholders of any performance issues.

e) **Provide User Training:** Provide training and support to users who will interact with or benefit from the deployed models. Ensure that users understand how to interpret model predictions, use model outputs to inform decision-making, and provide feedback for model improvement.

f) **Document Deployment Process:** Document the deployment process, including the steps taken, configurations made, and any challenges encountered. Create user documentation and operational manuals to guide stakeholders in using and maintaining the deployed models.

Chapter 3

Business Understanding

I. **Objective:** Develop a deep learning model to classify chest X-ray images for tuberculosis (TB) diagnosis, aiding healthcare professionals in accurate TB detection.

II. **Context:** Understand TB prevalence, availability of chest X-ray datasets, and existing TB detection methods to guide model development effectively.

III. **Goal:** Achieve accurate classification of TB-positive and TB-negative cases while minimizing false positives and false negatives, improving TB diagnosis efficiency.

IV. **Project Plan:** Develop a comprehensive plan outlining tasks such as data collection, labeling, preprocessing, model development using TensorFlow or PyTorch, performance evaluation, and potential clinical deployment.

V. **Risks:** Identify and address risks related to data privacy, data quality, interpretability of model decisions, and ethical considerations to ensure smooth project execution.

VI. **Success Criteria:** Define clear metrics such as accuracy, sensitivity, specificity, and AUC to evaluate the model's performance, ensuring it meets the desired standards for TB diagnosis.

Chapter 4

Data Understanding

I. **Data Collection:** The dataset comprises 7000 chest X-ray images categorized into Train, Validation, and Test sets, each containing images representing both Normal and TB classes.The division into Train, Validation, and Test sets ensures proper model training, validation, and evaluation 10.1.

II. **Exploratory Data Analysis (EDA):** Conduct a thorough exploration of the dataset to understand its structure, class distribution, and potential biases. This step involves visualizing sample images, plotting class distribution histograms, and identifying any patterns or anomalies in the data.

III. **Data Quality Assessment:** Evaluate the quality of the dataset by assessing factors such as image clarity, labeling consistency, and the presence of metadata. It's essential to ensure that the images are of sufficient quality for accurate model training and that the labels are consistently assigned to the correct class.

IV. **Class Balance Confirmation:** Verify the balance and distribution of classes within each dataset split (Train, Validation, and Test sets). Class imbalance can affect the model's performance, so it's crucial to confirm that each class is adequately represented in all datasets.

Chapter 5

Data Preparation

I. **Data Augmentation:** To increase the diversity of the training dataset and prevent overfitting, data augmentation techniques are applied. These techniques involve generating new synthetic data samples by applying transformations such as rotation, flipping, zooming, and shifting to the original images.

II. **Normalization:** Normalize the pixel values of the preprocessed images to ensure uniformity and facilitate convergence during model training. Normalization typically involves scaling the pixel values to a standard range (e.g., [0, 1]) or using z-score normalization to center the pixel values around zero with unit variance.

III. **Image Resizing:** Resize the preprocessed images to a uniform dimension suitable for model input. This step ensures consistency in image size across the dataset and allows the deep learning model to process images efficiently. Common resizing dimensions for chest X-ray images include 224x224 or 256x256 pixels.

IV. **Dataset Splitting:** Split the preprocessed dataset into Train, Validation, and Test sets according to predefined ratios (e.g., 70%, 15%, and 15%, respectively). The Train set is used to train the model, the Validation set is used to tune hyperparameters and monitor model performance, and the Test set is used to evaluate the final model's performance.

Chapter 6

Modeling

I. **Selection of Deep Learning Architectures:** The architecture chosen is a Convolutional Neural Network (CNN) tailored for image classification tasks. It consists of multiple layers, including convolutional and pooling layers, designed to extract hierarchical features from input images. CNNs are well-suited for analyzing spatial patterns in images and have been widely used in medical image analysis.

II. **Exploration of Architectural Variants:** The CNN architecture consists of an input layer followed by convolutional layers with increasing filters (32, 64, 128) and (3, 3) kernel size. Max-pooling layers (2, 2) reduce dimensions after each convolution. A Flatten layer reshapes feature maps into a 1D vector for fully connected layers with 512 units and ReLU activation. The final layer, a single neuron with sigmoid activation, yields probabilities for binary classification.

III. **Model Training:** Train the selected CNN model for a sufficient number of epochs to allow for convergence and learning of discriminative features from the input data. Monitor the model's performance on the Validation set during training to track metrics such as accuracy, loss, and validation accuracy.

IV. **Performance Benchmark:** Set a performance benchmark for the model based on the desired accuracy threshold. Aim to achieve a high level of accuracy, such as 98%, on the Validation set, indicating the model's ability to generalize well to unseen data and make accurate predictions.

Table 6.1: **Model Summary**

Layer (type)	**Output Shape**	**Param #**
conv2d	(None, 148, 148, 32)	896
max_pooling2d	(None, 74, 74, 32)	0
conv2d_1	(None, 72, 72, 64)	18496
max_pooling2d_1	(None, 36, 36, 64)	0
conv2d_2	(None, 34, 34, 128)	73856
max_pooling2d_2	(None, 17, 17, 128)	0
conv2d_3	(None, 15, 15, 128)	147584
max_pooling2d_3	(None, 7, 7, 128)	0
flatten	(None, 6272)	0
dense	(None, 512)	3211776
dense_1	(None, 1)	513

Chapter 7

Evaluation

The model achieves a test accuracy of 98.48%. This result is obtained by evaluating the model on the test set, yielding a test loss of 0.0398.

Confusion Matrix and Classification Report:

The evaluation process involves collecting true labels and predicted probabilities. From these, class predictions are derived, resulting in a confusion matrix with the following values:

$$\begin{bmatrix} 517 & 8 \\ 8 & 517 \end{bmatrix}$$

This confusion matrix indicates the model's performance in correctly classifying instances across two classes. Additionally, the classification report provides further insights into the model's performance, with precision, recall, and f1-score metrics reported for each class.

ROC Curve and AUC:

True labels and predicted probabilities are used to compute the Receiver Operating Characteristic (ROC) curve and the Area Under the Curve (AUC). The AUC value, representing the model's ability to distinguish between classes, is found to be 0.99. The ROC curve visually demonstrates the trade-off between true positive rate and false positive rate across different thresholds.

Table 7.1: **Classification Report**

	Precision	Recall	F1-score	Support
0.0	0.98	0.98	0.98	525
1.0	0.98	0.98	0.98	525
accuracy	0.98	0.98	0.98	1050
macro avg	0.98	0.98	0.98	1050
weighted avg	0.98	0.98	0.98	1050

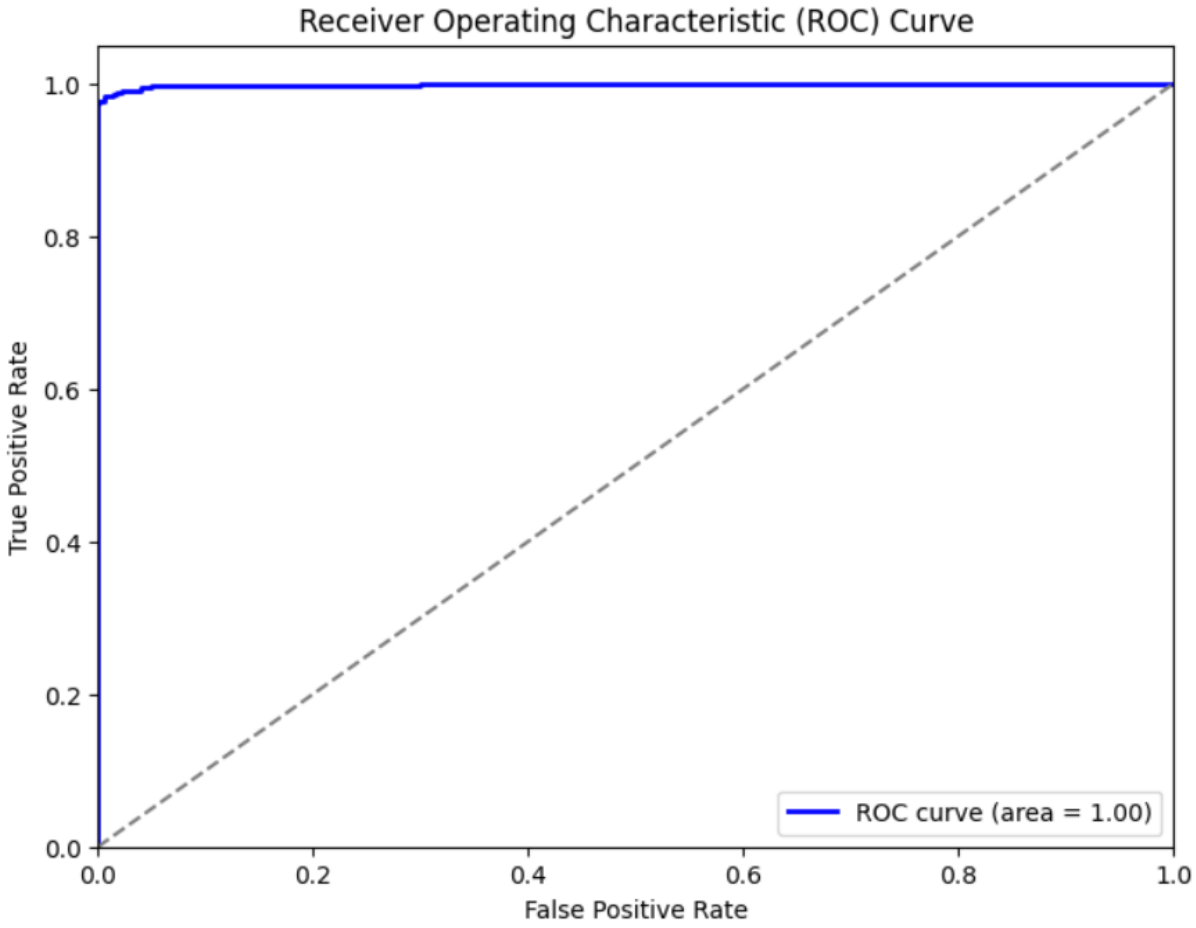


Figure 7.1: ROC graph

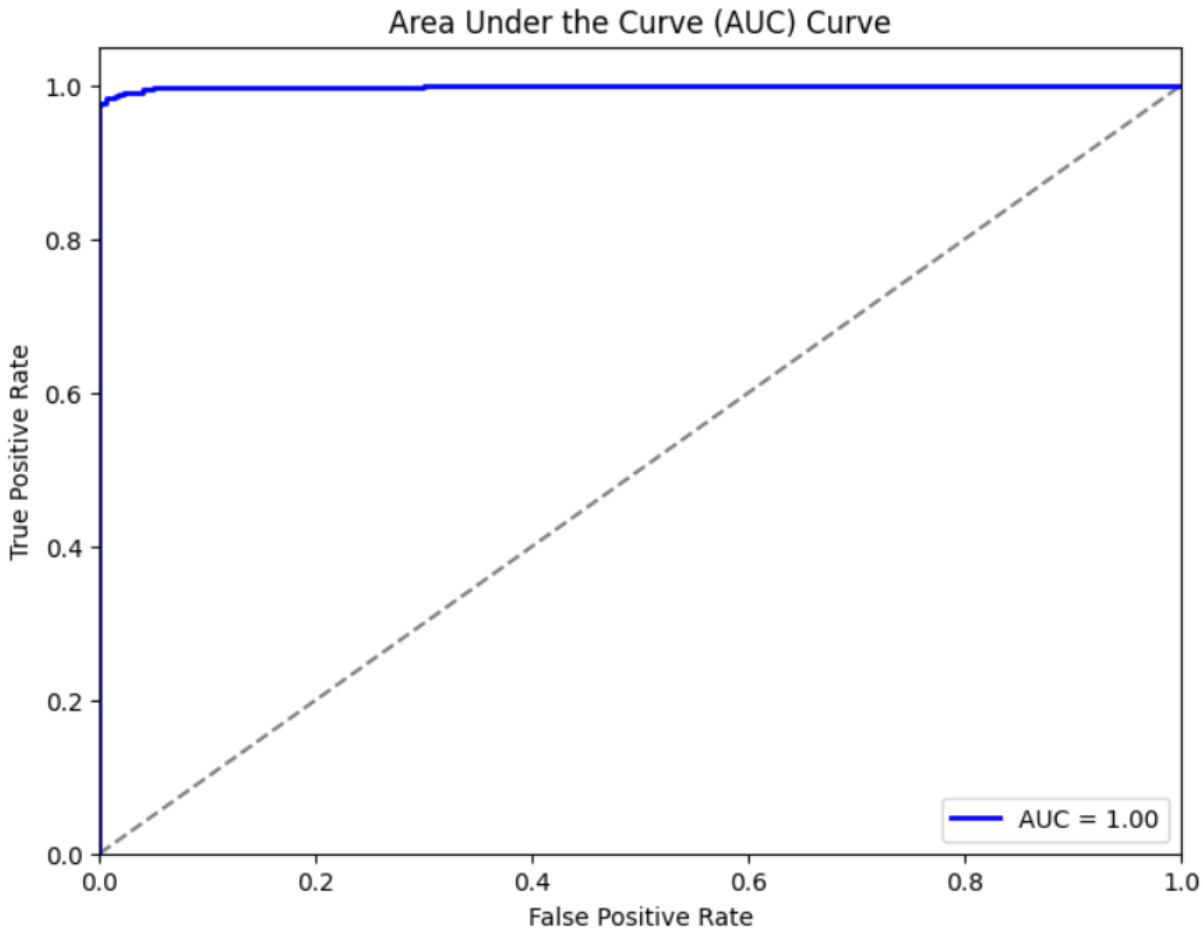


Figure 7.2: AUC graph

Chapter 8

Deployment

8.0.1 Code

In this section, we provide details of the implementation code for tuberculosis detection using deep learning in which Convolutional Neural Networks (CNN) algorithm has been implemented. The CNN architecture has been chosen due to its effectiveness in capturing spatial hierarchies and patterns in medical images, making it well-suited for medical imaging tasks like tuberculosis detection.

We start by describing the dataset used for training and evaluation, including its source, size, and characteristics. The dataset likely consists of chest X-ray images annotated with labels indicating the presence or absence of tuberculosis. We discuss any preprocessing steps applied to the dataset, such as resizing, normalization, or augmentation, to prepare it for training.

Next, we outline the architecture of the CNN model designed for tuberculosis detection. This includes details such as the number and type of convolutional layers, pooling layers, activation functions such as ReLU and Sigmoid.

We explain the training procedure used to optimize the CNN model for tuberculosis detection. This involves specifying hyperparameters such as learning rate, batch size, number of epochs, and optimizer (e.g., Adam). We describe the process of feeding the training data into the model, computing the loss function (e.g., binary cross-entropy), and updating the model parameters.

We detail the metrics used to evaluate the performance of the trained CNN model. Common metrics for binary classification tasks like tuberculosis detection include accuracy, precision, recall, F1-score and area under the ROC and AUC curve. We discuss the rationale behind

choosing these metrics and how they provide insights into the model's effectiveness.

Finally, we present the results of the tuberculosis detection experiments conducted using the CNN model. This includes quantitative metrics obtained during training and evaluation, as well as qualitative analysis of model predictions on test data.

Following is the implemented and tested code:

```
from sklearn.metrics import confusion_matrix,
    classification_report, roc_curve, auc
from tensorflow.keras.preprocessing.image import
    ImageDataGenerator
from tensorflow.keras.utils import plot_model
from tensorflow.keras import layers, models
from keras.preprocessing import image
from keras.models import Sequential
import matplotlib.pyplot as plt
import tensorflow as tf
import seaborn as sns
import pandas as pd
import numpy as np
import random
import os


from google.colab import drive

drive.mount('/content/drive')

dataset_path = '/content/drive/My Drive/demo_tb_dataset'
print(dataset_path)

# Example: Load an image from Google Drive
```

```
image_path = '/content/drive/My Drive/demo_tb_dataset/Test/tb/
    TB (526).png'
image = Image.open(image_path)
plt.imshow(image)
plt.axis('off')  # Turn off axis
plt.show()

Training_Dataset_path = '/content/drive/My Drive/
    demo_tb_dataset/Train'
print(Training_Dataset_path)

Validation_Dataset_path = '/content/drive/My Drive/
    demo_tb_dataset/Validation'
print(Validation_Dataset_path)

Test_Dataset_path = '/content/drive/My Drive/demo_tb_dataset/
    Test'
print(Test_Dataset_path)

# Define constants
image_height = 150
image_width = 150
batch_size = 32
num_classes = 2  # Tuberculosis_positive and Normal


train_datagen = tf.keras.preprocessing.image.ImageDataGenerator
    (
      rescale=1./255,
      rotation_range=20,
      width_shift_range=0.2,
      height_shift_range=0.2,
```

```
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Define data generators for training, validation, and test
    sets
train_generator = train_datagen.flow_from_directory(
    Training_Dataset_path,
    target_size=(image_height, image_width),
    batch_size=batch_size,
    class_mode='binary',  # since there are only two classes:
        Tuberculosis_positive and Tuberculosis_negative
)

validation_datagen = tf.keras.preprocessing.image.
    ImageDataGenerator(rescale=1./255)
validation_generator = validation_datagen.flow_from_directory(
    Validation_Dataset_path,
    target_size=(image_height, image_width),
    batch_size=batch_size,
    class_mode='binary', # specify the class names
)

# Evaluate the model on test data if available
test_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255)
test_generator = test_datagen.flow_from_directory(
    Test_Dataset_path,
    target_size=(image_height, image_width),
    batch_size=batch_size,
```

```
    class_mode='binary'
)


model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(
        image_height, image_width, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

model.summary() # to view the model summary

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

# Train the model
history = model.fit(
    train_generator,
    epochs=100,
    validation_data=validation_generator,
)
```

```
model.save('/content/drive/My Drive/50_epoch_Tuberculosis_Model
    .h5')

test_loss, test_acc = model.evaluate(test_generator)
print('Test accuracy:', test_acc*100)



# Evaluate the model on test data if available
test_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255)
test_generator = test_datagen.flow_from_directory(
     Test_Dataset_path,
     target_size=(image_height, image_width),
     batch_size=batch_size,
     class_mode='binary'
)

loaded_model = tf.keras.models.load_model('/content/drive/My
    Drive/50_epoch_Tuberculosis_Model.h5')

# Show the model architecture
loaded_model.summary()

# Visualize the architecture of the saved model
plot_model(loaded_model, to_file='model_architecture.png',
    show_shapes=True)

test_loss, test_acc = loaded_model.evaluate(test_generator)
print('Test accuracy:', test_acc*100)

# Path to the test folder containing the subdirectories
```

```
test_folder = '/content/drive/My Drive/demo_tb_dataset/Test'

# Get a list of all subdirectories (class folders) in the test
    folder
class_folders = os.listdir(test_folder)

# Counter variable for serial number
serial_number = 1

# Iterate over each class folder
for class_folder in class_folders:
    # Construct the full path to the class folder
    class_folder_path = os.path.join(test_folder, class_folder)

    # Get a list of all image file names in the class folder
    image_files = os.listdir(class_folder_path)

    # Iterate over each image file
    for image_file in image_files:
        # Construct the full path to the image
        img_path = os.path.join(class_folder_path, image_file)

        # Load and preprocess the image
        img = image.load_img(img_path, target_size=(150, 150))
        img_array = image.img_to_array(img)
        img_array = np.expand_dims(img_array, axis=0)
        img_array = img_array / 255.0  # Normalize pixel values

        # Make predictions
        predictions = loaded_model.predict(img_array)

        # Interpret predictions
```

```
        if predictions[0][0] < 0.5:
            print("Test Image Number:", serial_number, "- X-Ray
                :", image_file, "- Normal")
        else:
            print("Test Image Number:", serial_number, "- X-Ray
                :", image_file, "- Tuberculosis Positive")

        # Increment serial number
        serial_number += 1


# List to store true labels and predicted probabilities
true_labels = []
predicted_probabilities = []

# Iterate through the test generator to collect true labels and
    predicted probabilities
for i in range(len(test_generator)):
    batch = test_generator[i]
    true_labels.extend(batch[1])
    predictions = loaded_model.predict(batch[0])
    predicted_probabilities.extend(predictions.flatten())  #
       Flatten predictions to match true labels

# Convert true labels and predicted probabilities to numpy
   arrays
true_labels = np.array(true_labels)
predicted_probabilities = np.array(predicted_probabilities)

# Convert probabilities to class predictions
predicted_classes = (predicted_probabilities > 0.5).astype(int)
```

```
# Compute confusion matrix
conf_matrix = confusion_matrix(true_labels, predicted_classes)
print("Confusion Matrix:")
print(conf_matrix)

# Compute classification report
class_report = classification_report(true_labels,
    predicted_classes)
print("\nClassification Report:")
print(class_report)

# Visualize Confusion Matrix

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='g',
    xticklabels=['Normal', 'TB'], yticklabels=['Normal', 'TB'])
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()

# Visualize Classification Report

# Generate classification report
class_report = classification_report(true_labels,
    predicted_classes, target_names=['Normal', 'TB'],
    output_dict=True)

# Convert classification report to DataFrame for visualization
class_report_df = pd.DataFrame(class_report).transpose()
```

```
# Plot heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(class_report_df.iloc[:-1, :-1], annot=True, cmap='
    Blues', fmt=".2f")
plt.title('Classification Report')
plt.xlabel('Metrics')
plt.ylabel('Classes')
plt.show()

# Visualize 10 Random sample predictions

# Get a list of indices to sample from
indices = random.sample(range(len(test_generator.filepaths)),
    10)

# Visualize sample predictions for the selected indices
for index in indices:
    img_path = test_generator.filepaths[index]
    true_label = true_labels[index]
    pred_label = predicted_classes[index]

    img = image.load_img(img_path, target_size=(150, 150))
    plt.imshow(img)
    plt.axis('off')
    true_class = 'TB' if true_label == 1 else 'Normal'
    pred_class = 'TB' if pred_label == 1 else 'Normal'
    plt.title(f'True: {true_class}, Predicted: {pred_class}')
    plt.show()

# List to store true labels and predicted probabilities
true_labels = []
predicted_probabilities = []
```

```
# Iterate through the test generator to collect true labels and
    predicted probabilities
for i in range(len(test_generator)):
    batch = test_generator[i]
    true_labels.extend(batch[1])
    predicted_probabilities.extend(loaded_model.predict(batch
       [0]))

# Convert true labels and predicted probabilities to numpy
   arrays
true_labels = np.array(true_labels)
predicted_probabilities = np.array(predicted_probabilities)

# Compute ROC curve and AUC
fpr, tpr, thresholds = roc_curve(true_labels,
   predicted_probabilities)
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 =
    %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='gray', 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 (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

```
# Plot AUC curve
plt.figure(figsize=(8, 6))
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.plot(fpr, tpr, color='blue', lw=2, label='AUC = %0.2f' %
    roc_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Area Under the Curve (AUC) Curve')
plt.legend(loc="lower right")
plt.show()
```

8.0.2 Equation

Following are the formulae which have been used in this project to calculate the accuracy, precision, recall, f1 score, and AUC.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{8.1}$$

$$\text{Precision} = \frac{\text{True positives}}{\text{True positives + False positives}} \tag{8.2}$$

$$\text{Recall} = \frac{\text{True positives}}{\text{True positives + False negatives}} \tag{8.3}$$

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision + Recall}} \tag{8.4}$$

$$\text{AUC} = \int_0^1 \text{TPR}(\text{FPR}^{-1}(t))dt \tag{8.5}$$

8.0.3 Output Snapshots

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 32)      896

 max_pooling2d (MaxPooling2  (None, 74, 74, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 72, 72, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 36, 36, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 34, 34, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 17, 17, 128)       0
 g2D)

 conv2d_3 (Conv2D)           (None, 15, 15, 128)       147584

 max_pooling2d_3 (MaxPoolin  (None, 7, 7, 128)         0
 g2D)

 flatten (Flatten)           (None, 6272)              0

 dense (Dense)               (None, 512)               3211776

 dense_1 (Dense)             (None, 1)                 513

=================================================================
Total params: 3453121 (13.17 MB)
Trainable params: 3453121 (13.17 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 8.1: CNN layers description

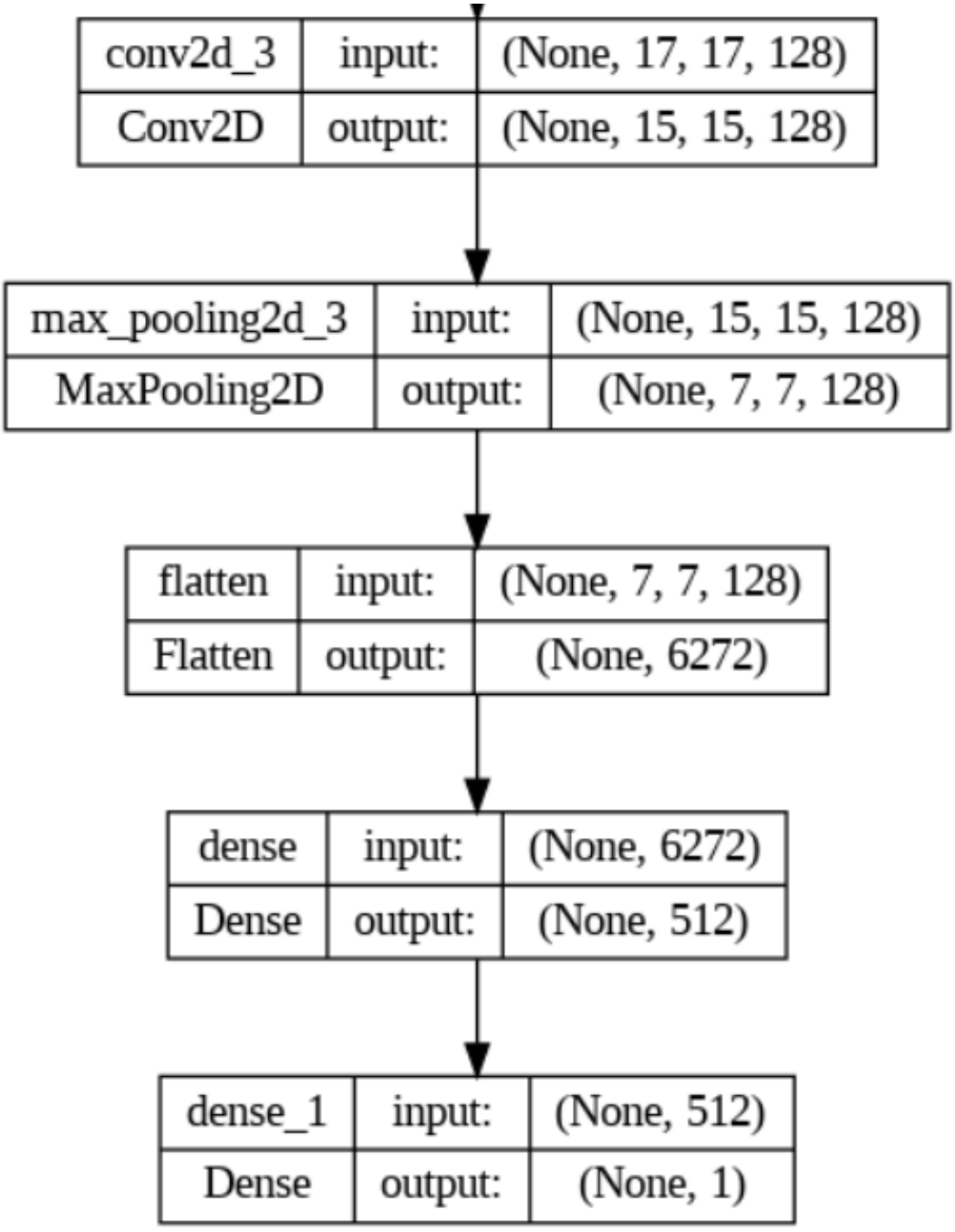


Figure 8.2: Extended description of layer's architecture

```
Test Image Number: 121 - X-Ray: Normal (587).jpg - Tuberculosis Positive
1/1 [==============================] - 0s 42ms/step
Test Image Number: 122 - X-Ray: Normal (593).jpg - Normal
1/1 [==============================] - 0s 42ms/step
Test Image Number: 123 - X-Ray: Normal (598).jpg - Normal
1/1 [==============================] - 0s 41ms/step
Test Image Number: 124 - X-Ray: Normal (606).jpg - Normal
1/1 [==============================] - 0s 39ms/step
Test Image Number: 125 - X-Ray: Normal (591).jpg - Normal
1/1 [==============================] - 0s 37ms/step
Test Image Number: 126 - X-Ray: Normal (599).jpg - Normal
1/1 [==============================] - 0s 40ms/step
Test Image Number: 127 - X-Ray: Normal (604).jpg - Normal
1/1 [==============================] - 0s 41ms/step
Test Image Number: 128 - X-Ray: Normal (597).jpg - Normal
1/1 [==============================] - 0s 39ms/step
Test Image Number: 129 - X-Ray: Normal (608).jpg - Normal
1/1 [==============================] - 0s 43ms/step
Test Image Number: 130 - X-Ray: Normal (607).jpg - Normal
1/1 [==============================] - 0s 43ms/step
Test Image Number: 131 - X-Ray: Normal (592).jpg - Normal
1/1 [==============================] - 0s 41ms/step
Test Image Number: 132 - X-Ray: Normal (595).jpg - Normal
1/1 [==============================] - 0s 45ms/step
Test Image Number: 133 - X-Ray: Normal (601).jpg - Normal
1/1 [==============================] - 0s 37ms/step
Test Image Number: 134 - X-Ray: Normal (610).jpg - Normal
1/1 [==============================] - 0s 37ms/step
Test Image Number: 135 - X-Ray: Normal (588).jpg - Normal
1/1 [==============================] - 0s 39ms/step
Test Image Number: 136 - X-Ray: Normal (594).jpg - Tuberculosis Positive
1/1 [==============================] - 0s 41ms/step
Test Image Number: 137 - X-Ray: Normal (605).jpg - Normal
1/1 [==============================] - 0s 41ms/step
Test Image Number: 138 - X-Ray: Normal (602).jpg - Normal
1/1 [==============================] - 0s 40ms/step
```

Figure 8.3: Predicted output

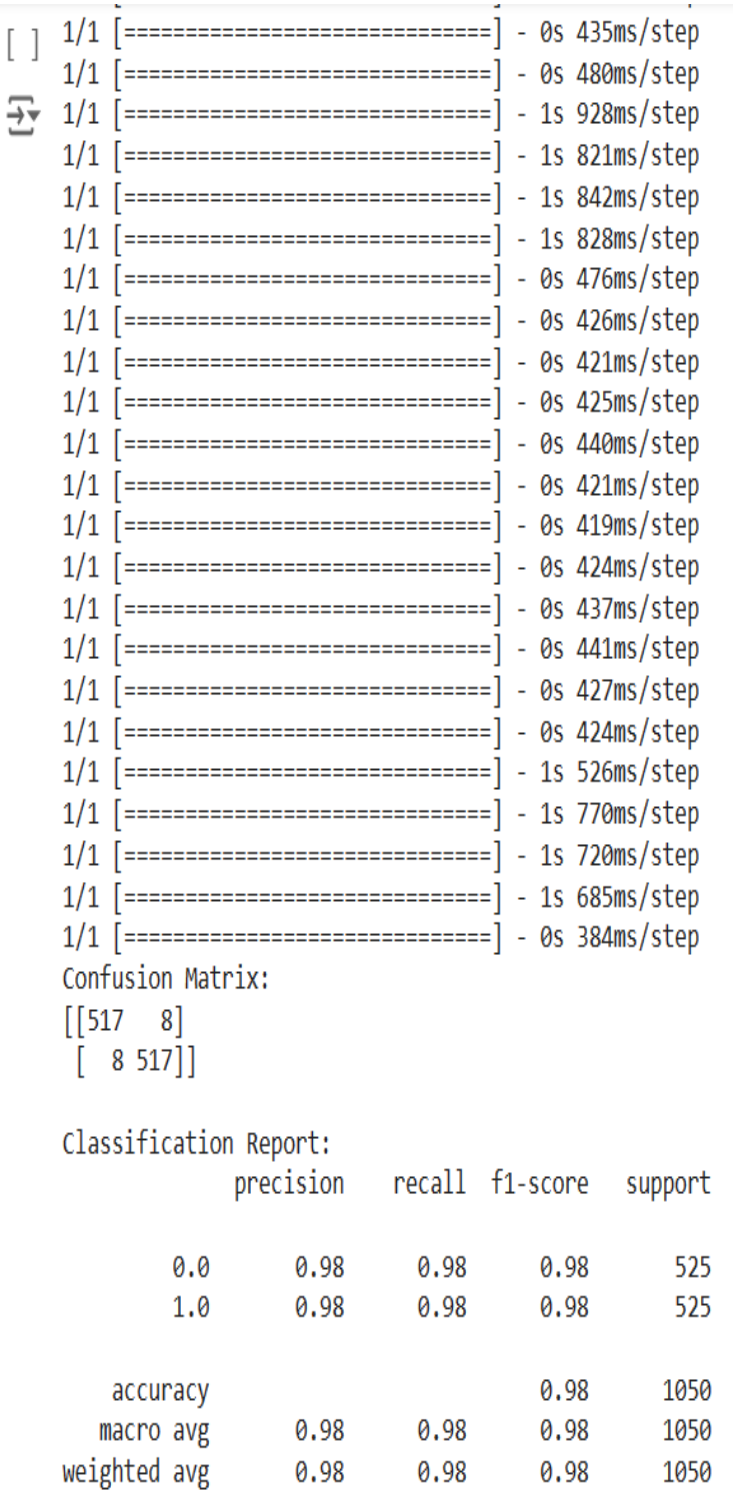


Figure 8.4: Output of confusion matrix and classification report

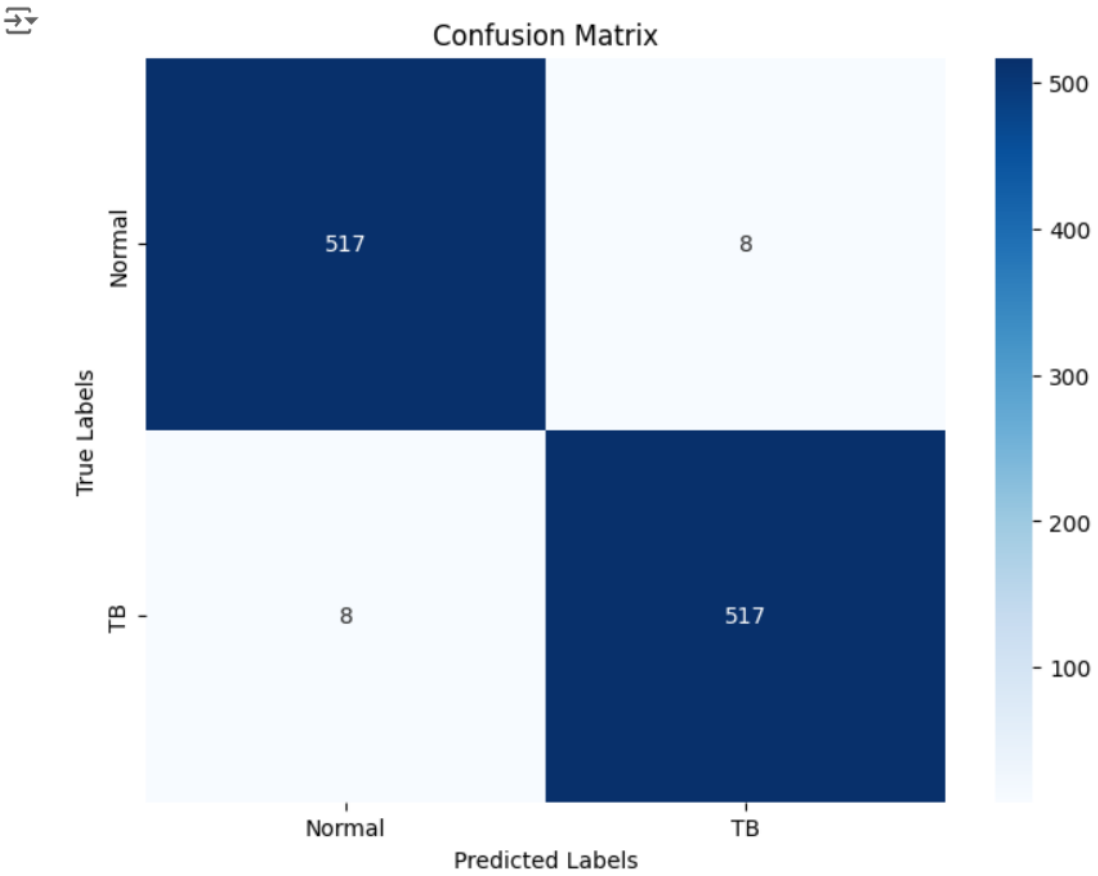


Figure 8.5: Heatmap of confusion matrix

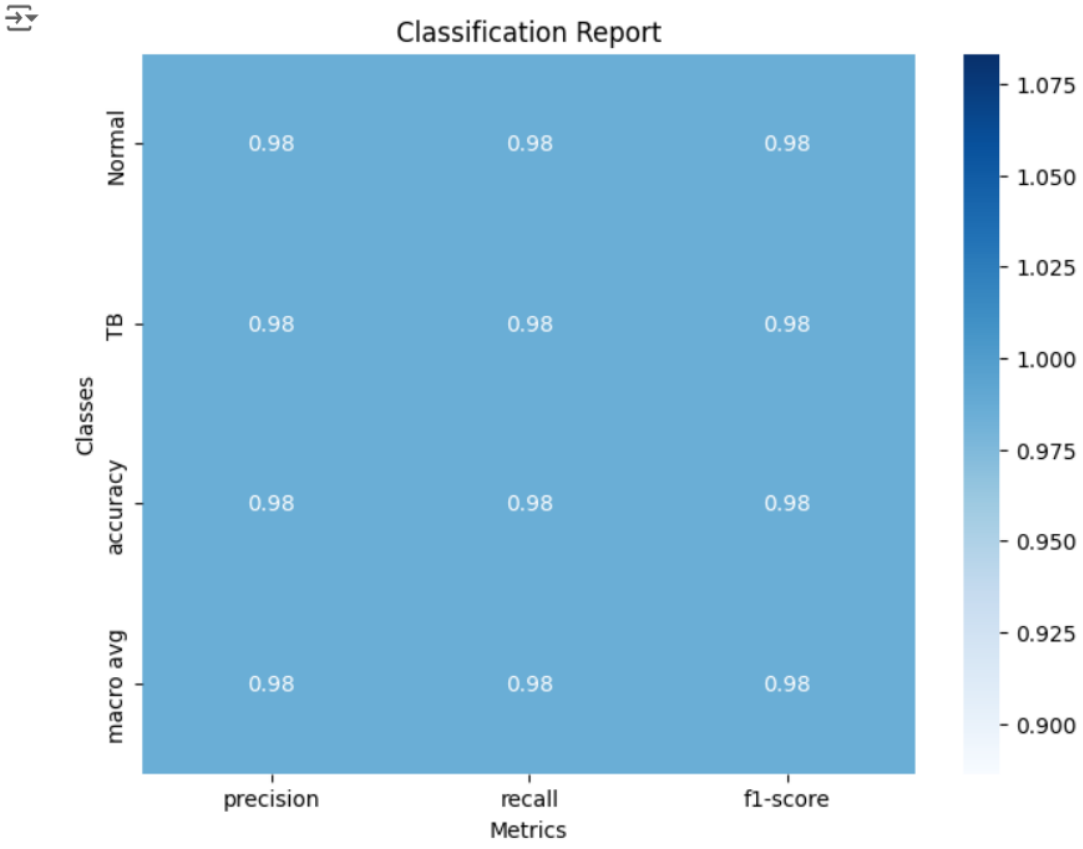


Figure 8.6: Heatmap of classification report

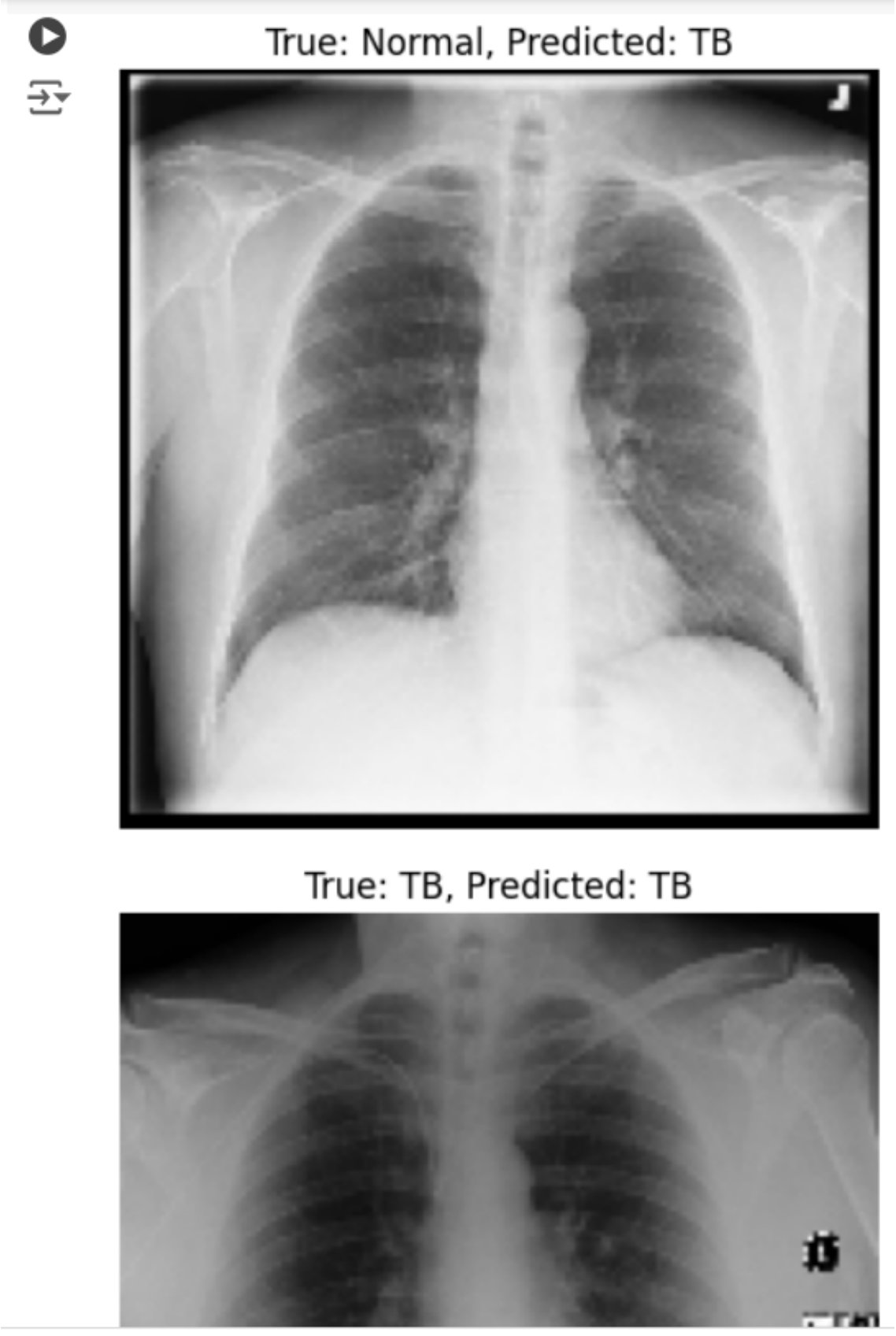


Figure 8.7: Output of random chest x-ray - 1

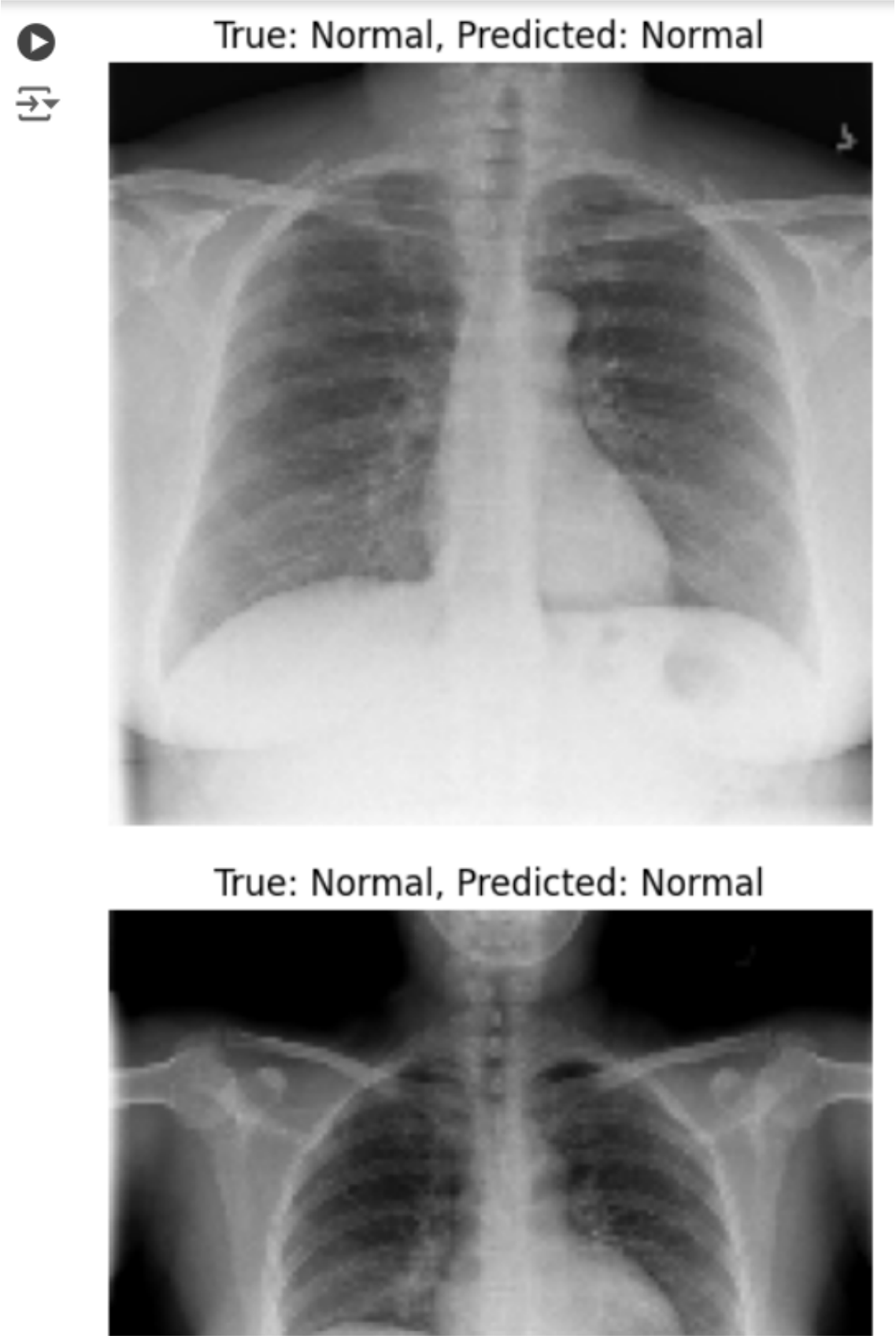


Figure 8.8: Output of random chest x-ray - 2

Chapter 9

Conclusion and Future Work

Conclusion

In conclusion, the evaluation of this model on the test dataset yielded promising results. With a test accuracy of 98.48%, our model demonstrates strong performance in classifying instances across two classes. The confusion matrix and classification report provide detailed insights into the model's precision, recall, and f1-score metrics, indicating balanced performance across both classes. Additionally, the Receiver Operating Characteristic (ROC) curve showcases the model's ability to discriminate between classes, with an Area Under the Curve (AUC) value of 0.99, further validating its efficacy.

Overall, these results suggest that our model has been effectively trained and can generalize well to unseen data, making it a valuable asset in various real-world applications.

Future Work

While our model has shown promising performance, there are several avenues for future work to explore and enhance its capabilities. Some potential directions include:

Fine-tuning **hyperparameters** like **learning rate**, **batch size**, and **model architecture** enhances **performance**. **Data augmentation** diversifies **training data** for better **generalization**. **Ensemble methods** combine models for improved **accuracy**. **Transfer learning** with pre-trained models boosts **performance**. Incorporating **interpretability** techniques reveals model **decision-making**, enhancing **trustworthiness**.

Chapter 10

Appendix

In the dataset utilized for tuberculosis detection using deep learning project, the images are organized into three main folders: train, validation, and test. Each of these folders contains two subfolders, one for tuberculosis (tb) images and another for normal images.

10.0.1 Dataset Classification

Table 10.1: **Chest X-Ray Dataset Classification Table**

	Training	Validation	Test
Normal	2450	525	525
Tuberculosis	2450	525	525
Total	5900	1050	1050

10.0.2 ROC and AUC Graph

The Receiver Operating Characteristic (ROC) curve is a critical tool in assessing the performance of binary classification models. It provides a comprehensive view of how well a model can discriminate between the positive and negative classes by illustrating the trade-off between the true positive rate (TPR) and the false positive rate (FPR) 7.1 across various threshold settings.

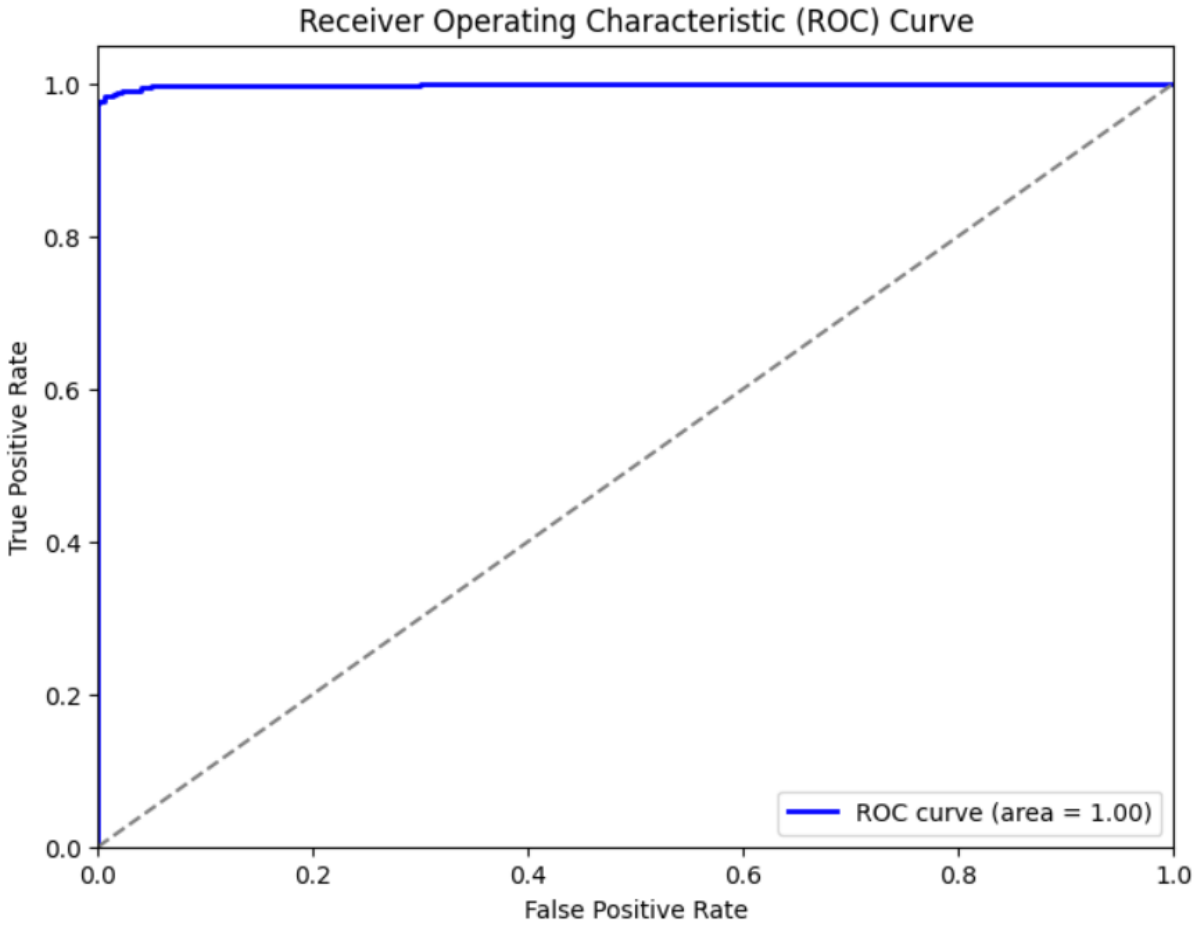


Figure 10.1: ROC Curve

The ROC curve is constructed by plotting the TPR against the FPR at different threshold values. The TPR, also known as sensitivity or recall, measures the proportion of actual positive cases correctly identified by the model. Conversely, the FPR represents the proportion of actual negative cases incorrectly classified as positive.

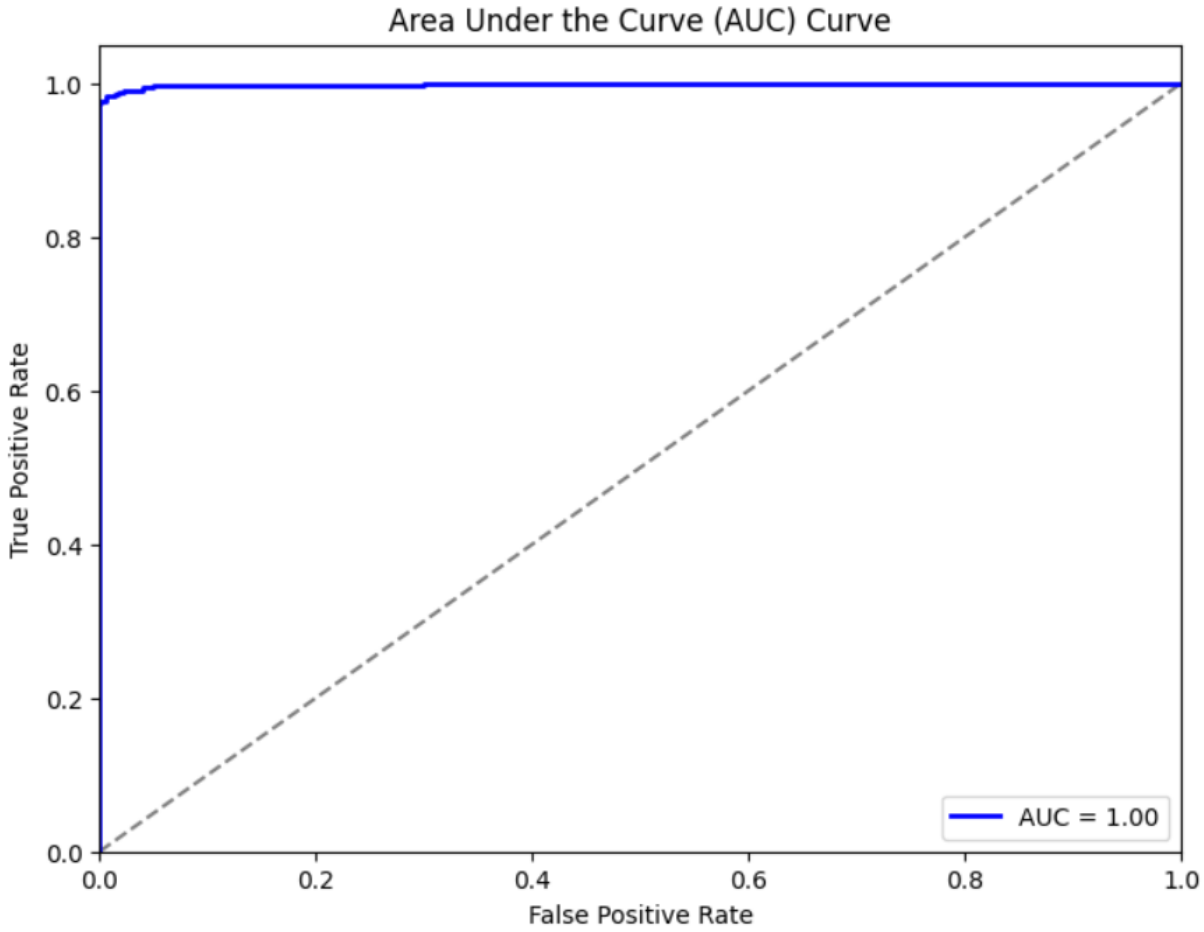


Figure 10.2: AUC Curve

Overall, the ROC curve and AUC provide valuable insights into the performance of binary classification models, empowering practitioners to make informed decisions about model selection, threshold tuning, and deployment in real-world applications.

10.0.3 References Used in Project

Several studies have addressed the classification of chest X-ray (CXR) images, particularly focusing on differentiating COVID-19 from other infectious diseases [1–3]. We drew insights from the work of Huang et al. (2016) [4] to enhance our understanding. Additionally, techniques such as deep learning and convolutional neural networks (CNNs) have been employed for CXR image view classification [5]. Methodologically, we adopted the CRISP-DM framework outlined by Wirth and Chapman (2000) [6, 7]. Data augmentation has been explored to enhance the performance of classifiers for medical imaging tasks, including cardiovascular abnormality classification [8–10].

Hyperparameter optimization plays a crucial role in fine-tuning deep learning models. Research has extensively discussed hyperparameter optimization techniques, challenges, and best practices [11]. Activation functions such as rectified linear unit (ReLU) and sigmoid have been studied for their impact on deep neural network performance [12,13]. The improved Adam optimizer for deep neural networks was informed by Zhang (2018) [14]. Lastly, we studied the advantages of flattened convolutional neural networks from Jin et al. (2015) [15].

Evaluation of classifier performance often involves analyzing receiver operating characteristic (ROC) curves and false positive rates (FPR) [16–18]. Understanding and mitigating false-positive detections, particularly in tuberculosis diagnosis, are critical for improving diagnostic accuracy [19].

Bibliography

[1] N. Salazar-Austin, C. Mulder, G. Hoddinott, T. Ryckman, C. F. Hanrahan, K. Velen, L. Chimoyi, S. Charalambous, and V. N. Chihota, "Preventive treatment for household contacts of drug-susceptible tuberculosis patients," *Pathogens*, vol. 11, no. 11, p. 1258, 2022.

[2] A. Sharma, S. Rani, and D. Gupta, "Artificial intelligence-based classification of chest x-ray images into covid-19 and other infectious diseases," *International journal of biomedical imaging*, vol. 2020, pp. 1–10, 2020.

[3] Z. Xue, D. You, S. Candemir, S. Jaeger, S. Antani, L. R. Long, and G. R. Thoma, "Chest x-ray image view classification," in *2015 IEEE 28th International Symposium on Computer-Based Medical Systems*. IEEE, 2015, pp. 66–71.

[4] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*. Springer, 2016, pp. 646–661.

[5] A. W. Salehi, S. Khan, G. Gupta, B. I. Alabduallah, A. Almjally, H. Alsolai, T. Siddiqui, and A. Mellit, "A study of cnn and transfer learning in medical imaging: Advantages, challenges, future scope," *Sustainability*, vol. 15, no. 7, p. 5930, 2023.

[6] R. Wirth and J. Hipp, "Crisp-dm: Towards a standard process model for data mining," in *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*, vol. 1. Manchester, 2000, pp. 29–39.

[7] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, R. Wirth *et al.*, "Crisp-dm 1.0: Step-by-step data mining guide," *SPSS inc*, vol. 9, no. 13, pp. 1–73, 2000.

[8] A. Madani, M. Moradi, A. Karargyris, and T. Syeda-Mahmood, "Chest x-ray generation and data augmentation for cardiovascular abnormality classification," in *Medical imaging 2018: Image processing*, vol. 10574. SPIE, 2018, pp. 415–420.

[9] L. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning," *arXiv preprint arXiv:1712.04621*, 2017.

[10] X. Jiang and Z. Ge, "Data augmentation classifier for imbalanced fault classification," *IEEE Transactions on Automation Science and Engineering*, vol. 18, no. 3, pp. 1206–1217, 2020.

[11] B. Bischl, M. Binder, M. Lang, T. Pielok, J. Richter, S. Coors, J. Thomas, T. Ullmann, M. Becker, A.-L. Boulesteix *et al.*, "Hyperparameter optimization: Foundations, algorithms, best practices and open challenges. arxiv," *arXiv preprint arXiv:2107.05847*, 2021.

[12] C. Banerjee, T. Mukherjee, and E. Pasiliao Jr, "An empirical study on generalizations of the relu activation function," in *Proceedings of the 2019 ACM Southeast Conference*, 2019, pp. 164–167.

[13] A. A. Waoo and B. K. Soni, "Performance analysis of sigmoid and relu activation functions in deep neural network," in *Intelligent Systems: Proceedings of SCIS 2021*. Springer, 2021, pp. 39–52.

[14] Z. Zhang, "Improved adam optimizer for deep neural networks," in *2018 IEEE/ACM 26th international symposium on quality of service (IWQoS)*. Ieee, 2018, pp. 1–2.

[15] J. Jin, A. Dundar, and E. Culurciello, "Flattened convolutional neural networks for feedforward acceleration.(2014)," *arXiv preprint arXiv:1412.5474*, 2015.

[16] J. A. Hanley *et al.*, "Receiver operating characteristic (roc) methodology: the state of the art," *Crit Rev Diagn Imaging*, vol. 29, no. 3, pp. 307–335, 1989.

[17] Z. H. Hoo, J. Candlish, and D. Teare, "What is an roc curve?" pp. 357–359, 2017.

[18] ——, "What is an roc curve?" pp. 357–359, 2017.

[19] F. P. R. FPR, "False positive rate (fpr)."