PNEUMONIA DISEASE DETECTION USING MOBILENETV2 MODEL IN DEEP LEARNING

Dissertation

Submitted to

THE GANDHIGRAM RURAL INSTITUTE

(DEEMED TO BE UNIVERSITY)

In Partial fulfillment of the requirements for the award of the Degree of

MASTER OF COMPUTER APPLICATIONS

BY

RAVIN RAJ S

(Register Number : 23322009)

Department of Computer Science and Applications

The Gandhigram Rural Institute

(Deemed to be University)

Gandhigram – 624 302

Tamil Nadu

April 2025

BONAFIDE CERTIFICATE

This is to certify that the project titled **"PNEUMONIA DISEASE DETECTION USING MOBILENETV2 MODEL IN DEEP LEARNING"** is a Bonafide record of work carried out by **RAVIN RAJ S (23322009)** submitted in partial fulfillment of the requirements for the award of **MASTER OF COMPUTER APPLICATIONS** in The Gandhigram Rural Institute (Deemed to be University), Gandhigram during the period of December 2024 – April 2025.

Dr. M. Mary Shanthi Rani, M.C.A., M.Phil., Ph.D.,
Project Guide

Dr. P. Kalavathi, M.C.A., M.Phil., Ph.D.,
Head of the Department

Submitted for the Viva-voce examination held on _____.

Internal Examiner

External Examiner

DECLARATION

I hereby declare that this project work titled **"PNEUMONIA DISEASE DETECTION USING MOBILENETV2 MODEL IN DEEP LEARNING"** is a record of original work done by me under the supervision and the guidance of **Dr.M.Mary Shanthi Rani, M.C.A., M.Phil., Ph.D.** Professor. Department of Computer Science and Applications, this project work has not formed the basis for the award of any Degree/Diploma/Associateship/Fellowship or similar title to any candidate of any other University.

PLACE : Gandhigram **RAVIN RAJ S**

DATE : **(23322009)**

ACKNOWLEDGEMENT

I thank the almighty for giving me divine strength and enabling me to complete this project work successfully.

I heartily thank **Dr.P.Kalavathi, M.C.A., M.Phil., Ph.D.** Professor and Head of the Department of Computer Science and Applications, for the facilities given to me for accomplishing this task.

Also, I wholeheartedly thank my supervisor and guru **Dr.M.Mary Shanthi Rani, M.C.A., M.Phil., Ph.D.** Professor, Department of Computer Science and Applications, for giving me a moral support and excellence guidance through the course of the development of this project.

I express my deep appreciation for the help and warm encouragement that I have received from my family and my dear friends, because without their support could not have dreamt of completing this project successfully.

Finally, I would like to thank who directly or indirectly helped me in the successfully completion of this project work.

RAVIN RAJ S

(23322009)

TABLE OF CONTENTS

ABSTRACT

CHAPTER - I

ABSTRACT

The main project is titled **"PNEUMONIA DISEASE DETECTION USING MOBILENETV2 MODEL IN DEEP LEARNING".** Pneumonia is a severe respiratory disease that can cause serious health problems or even death if not diagnosed. Conventional diagnosis techniques with manual interpretation of chest X-ray images are time-consuming and restricted by the availability of experienced radiologists. The goal of this project is to design an automatic, deep learning-based pneumonia diagnostic system to enable early and accurate diagnosis. The model to be deployed employs MobileNetV2, a light yet efficient convolutional neural network model, to classify **Normal**, **Bacterial Pneumonia**, and **Viral Pneumonia** from chest X-rays. The data are first collected and organized, then preprocesses using **CLAHE (Contrast Limited Adaptive Histogram Equalization)** for enhancing image contrast and **Median Filtering** for reducing noise and enhancing features.

A single MobileNetV2 model architecture is employed and tested under two training scenarios:

- Without Preprocessing – raw chest X-ray images
- With Preprocessing – CLAHE and Median filter using chest X-ray images

The performance of the model is measured against metrics like accuracy, precision, recall, and F1-score. The comparison brings to light the effect of image preprocessing on the improvement of model performance and resilience. The experiment proves that utilizing image enhancement algorithms greatly enhances pneumonia detection accuracy based on MobileNetV2. This leads to a mobile-efficient, scalable, and real-world healthcare-applicable diagnostic solution. The experiment proves that image enhancement techniques are highly effective in improving pneumonia detection accuracy with MobileNetV2. This yields a mobile-friendly, efficient, and scalable diagnostic tool appropriate for real-world healthcare use.

INTRODUCTION

CHAPTER – II

INTRODUCTION

Pneumonia is a life-threatening breathing infection that inflames the air sacs of one or both lungs and can fill with fluid or pus, leading to such symptoms as cough, fever, shortness of breath, and chest discomfort. It has a considerable global health burden, particularly in children, older adults, and immunocompromised persons. It is important to diagnose early and correctly to enable prompt treatment by medical professionals and minimize mortality. Historically, pneumonia diagnosis is made using physical exams, blood work, and chest X-ray readings by qualified radiologists. Manual chest X-ray interpretation can be time-consuming, prone to human error, and usually limited by the number of qualified medical personnel—particularly in resource-limited environments.

In order to overcome these drawbacks, this project suggests a deep learning solution called "**Pneumonia Disease Detection Using MobileNetV2 Model in Deep Learning.**" The goal is to design an automated, lightweight, and accurate diagnostic tool that can classify chest X-ray images into three categories: Normal, Bacterial Pneumonia, and Viral Pneumonia. The project uses MobileNetV2, a computationally light convolutional neural network architecture, and hence it is well suited for mobile and embedded devices. To increase the diagnostic efficacy, chest X-ray images are preprocessed using CLAHE to enhance the contrast of the images and Median Filtering to remove noise. Effectiveness of preprocessing is verified by training the MobileNetV2 model on two cases, namely preprocessed and not preprocessed. Model performance is measured with main performance metrics such as accuracy, precision, recall, and F1-score.

This work shows how integrating image processing techniques with a light-weight CNN such as MobileNetV2 can make the detection of pneumonia greatly enhanced. The solution, being accurate in addition to mobile-compatible, scalable, and easy to use, is suitable for real-world medical practice, particularly in remote or resource-constrained locations.

SYSTEM ANALYSIS

CHAPTER – III

SYSTEM ANALYSIS

3.1 Existing System

In today's clinical practice, radiologists' reading of chest X-ray (CXR) images is crucial in the diagnosis of pneumonia. Although X-ray imaging is inexpensive and readily available, its reliability is highly dependent on the radiologist's skill. This dependency can result in variable diagnoses, delays, and misclassification, particularly in areas with poor access to skilled personnel. Recent developments in deep learning have made it possible to have automated systems assist medical image analysis. Convolutional Neural Networks (CNNs) have been extensively used to identify pneumonia from CXR images. MobileNetV2, being lightweight, is well suited for application on real-time and resource-limited platforms like mobile and embedded systems.

The current system utilizes transfer learning to tap into pre-trained weights, decreasing training time and increasing performance. Data augmentation methods such as rotation and flipping are utilized to augment the dataset, enhance generalization, and reduce overfitting. The model is tested using image resolutions between 128×128 and 224×224 to strike a balance between accuracy and computational complexity for deployment on edge devices.

The model worked well, with about 90% accuracy and an 82% F1-score on non-augmented data. With augmented data, performance dropped slightly—precision to 82%, recall to 82%, and F1-score to 83%. Even so, the model had robust recall on all classes, showing consistent classification performance. But the system is not perfect. Conditions such as class imbalance can lead to biased predictions, and sophisticated image preprocessing techniques like Contrast Limited Adaptive Histogram Equalization (CLAHE) and Median Filtering, which can improve feature clarity and model robustness, are not yet fully exploited. Research indicates that utilization of these methods can greatly enhance image quality and model accuracy.

In summary, while the current system demonstrates strong pneumonia detection capabilities, integrating better preprocessing methods and addressing class imbalance can further enhance its diagnostic effectiveness.

3.2 Proposed System

The objective of the suggested system is to develop a precise and autonomous pneumonia detection system using deep learning techniques and enhanced image preprocessing. The system is looking to use **MobileNetV2**, a lightweight model that is appropriate for real-time use, particularly in low-resource settings and mobile devices.

The process begins with the gathering of datasets, which are chest X-ray images of three classes:

- Bacterial Pneumonia
- Viral Pneumonia
- Normal

Two preprocessing techniques are employed to enhance the quality of the images:

- **CLAHE** (Contrast Limited Adaptive Histogram Equalization) to increase the contrast of the X-ray images so that the prominent features are more evident.
- **Median Filtering** to remove noise without distorting the salient features for the sake of effective detection.

Two configurations are defined to be compared with a common MobileNetV2 model architecture:

- Preprocessed training environment – employs images processed using CLAHE and Median Filtering to enhance quality and feature extraction.
- Baseline training configuration – employs the original chest X-ray images without preprocessing.

This contrast identifies the effect of image enhancement methods on the model's capacity to accurately diagnose pneumonia conditions. The models are tested using conventional performance metrics like accuracy, precision, recall, F1-score. The baseline must be surpassed by the preprocessed model both in image quality and classification accuracy, as the preprocessing enables the MobileNetV2 model to extract better features.

The system proposed here has some advantages:

- Enhanced image quality because of preprocessing methods.
- Enhanced feature extraction and better accuracy by applying the MobileNetV2 model.
- Noise resistance, with performance remaining constant even under images degraded. This model can be deployed on mobile apps, particularly in resource-limited settings, to achieve fast and consistent detection of pneumonia.

The system can be generalized further to identify other lung conditions, for example, Tuberculosis, to make it more clinically useful.

3.3 Feasibility Study

The Feasibility Study addresses the feasibility of implementing the proposed pneumonia detection system with MobileNetV2 and enhanced image preprocessing. The system must yield an automated, accurate pneumonia detection system that can be used in real-time, especially in mobile and resource-constrained settings.

3.3.1 Technical Feasibility

MobileNetV2 is selected as the deep learning model for this project because it possesses a lean and efficient architecture that is well-suited for real-time mobile applications. It provides a balance between accuracy and computational expense, which is critical for mobile-based healthcare applications. In order to improve the quality of chest X-ray images and model accuracy, image preprocessing is done. CLAHE increases local contrast, making fine details clearer, and Median Filtering removes noise without smoothing important edges.

The above preprocessing techniques considerably enhance feature extraction, resulting in improved classification outcomes in determining Normal, Bacterial Pneumonia, and Viral Pneumonia cases.

3.3.2 Operational Feasibility

User Interface: The system allows healthcare workers to upload X – ray images and receive instant Pneumonia classifications.

Integration: It can be integrated into healthcare applications, supporting radiologists and medicals professionals by providing quick diagnostic assistance.

Deployment: The system can be deployed on mobile devices, providing on – site, real – time diagnosis without the need for advanced infrastructure.

3.3.3 Economic Feasibility

Development Costs: The costs involve training the model and developing the mobile app. MobileNetV2's lightweight design reduces both training and operational costs.

Infrastructure: The use of cloud services for training and mobile devices for deployment minimizes hardware costs.

Long-term Benefits: The system helps reduce healthcare costs by automating pneumonia detection and improving diagnostic efficiency in underserved regions.

3.3.4 Time Feasibility

Development Time: Model training takes a few days to weeks depending on resources. Preprocessing is quick and happens alongside data preparation.

Deployment Time: Once the model is trained, mobile deployment can be completed in a few weeks.

Real-time Diagnosis: Diagnosis results are generated within seconds to a minute, depending on the device and image size.

3.3.5 Social Feasibility

Access to Healthcare: The system improves access to pneumonia detection in remote areas with limited healthcare professionals.

User Adoption: The system is easy to use, allowing healthcare workers in low-resource environments to make faster diagnoses.

Impact: The system can significantly reduce pneumonia-related mortality by providing faster, more accurate diagnoses.

3.3.6 Legal and Ethical Feasibility

Regulatory Compliance: The system must comply with medical device regulations and data protection laws like HIPAA and GDPR.

Ethical Considerations: The system serves as a support tool, with transparency and explainability ensuring that it complements, rather than replaces, medical expertise.

SYSTEM SPECIFICATION

CHAPTER – IV

SYSTEM SPECIFICATION

4.1 Hardware Specification

System	:	Lenovo IdeaPad Slim 5i
RAM	:	16.0 GB
Processor	:	13th Gen Intel(R) Core (TM) i5 – 13500H 2.60GHz
Hard Disk	:	512 GB
Monitor	:	Generic PnP Monitor
Chip Type	:	Intel(R) Iris(R) Xe Graphics Family
System Type	:	64 – bit Operating System, x64 – based processor

4.2 Software Specification

Programming Tool	:	Python
IDE/Platform	:	Google Colab (Cloud based Jupyter Notebook)
Python Libraries	:	OS, NumPy, Pandas, TensorFlow/Keras, Scikit – Learn, Matplotlib, Google Colab Drive
Environments	:	Google Colab
Visualization Tools	:	Matplotlib, Pandas
Deep Learning Framework	:	TensorFlow, Keras
Storage	:	Google Drive (for Dataset and Model storage)
WEB Browser	:	Edge, Google Chrome
Operating System	:	Windows 11

4.3 Software Description

4.3.1 Python

Python is a high-level, general-purpose programming language that is all about simplicity and readability of code. Guido van Rossum created Python and it was first released in 1991. Now, Python is among the world's most employed and versatile programming languages. It can support various programming paradigms such as object-oriented, procedural, and functional programming, thus being helpful to a broad spectrum of applications.

The language's simple syntax, interpreted environment, and dynamic typing make it perfect for beginners and professionals alike. Python has a great standard library and an absolutely massive number of third-party libraries across a wide range of fields such as web development, data science, artificial intelligence, machine learning, automation, etc.

Python's open-source and extensive global community have spurred its adoption in industry and academia at a quick rate. Python is now a vital tool in contemporary software development and research, enabling rapid development, efficient problem-solving, and innovation in contemporary leading-edge technologies.

4.3.1.1 Pandas

Pandas is an open data analysis and manipulation library for the Python programming language. Created by Wes McKinney in 2008, Pandas offers high-performance, accessible data structures like Series (1D) and Data Frame (2D) that are best suited for structured data. Based on the NumPy library, Pandas is perfectly aligned with other scientific libraries and pipelines in the Python ecosystem. It makes data cleaning, transformation, aggregation, and visualization simple so data science, machine learning, and statistical computing become easier with it.

Pandas has vast support for file formats like CSV, Excel, JSON, SQL databases, etc. It has very powerful indexing and slicing capabilities, hence making it simple to work with large data sets, and in-built functions to execute time series analysis, missing data handling, and group operations.

Due to its efficiency, readability, and flexibility, Pandas is now a central part of data analysis and has been widely used in academic, business intelligence, and industry-level data-driven research.

4.3.1.2 NumPy

NumPy, or Numerical Python, is an open Python library that provides n-dimensional arrays and matrices, and a wide range of mathematical functions to operate on these arrays. Travis Oliphant created NumPy in 2006, and it is the foundation of numerical computing in the Python environment.

In essence, NumPy brings in the ndarray object, which is more powerful and efficient than the native lists of Python for numerical data manipulation. NumPy delivers vectorized operations, which greatly enhance the speed of mathematical operations by eschewing explicit loops.

NumPy finds extensive application in data science, machine learning, physics, engineering, and finance, where high-performance numerical computation is necessary. NumPy can be effortlessly combined with other libraries such as Pandas, Matplotlib, SciPy, and TensorFlow to be the foundation for high-level scientific and analytical computing in Python. Because of its low memory consumption, broadcasting, and efficient mathematical operations, NumPy has now emerged as an essential tool in both industrial and academic statistical data analysis and scientific computing tasks.

4.3.1.3 Scikit – Learn (sklearn)

Scikit-learn, or imported as sklearn, is a high-level, open-source Python machine learning library, with roots in NumPy, SciPy, and Matplotlib. Being a product of the SciKit environment, Scikit-learn was initially released in 2007 and has since grown to become one of the most used libraries for the implementation of machine learning algorithms and statistical modelling.

Scikit-learn offers easy and effective data analysis and data mining capabilities. Scikit-learn offers a wide range of machine learning operations including classification, regression, clustering, dimensionality reduction, model selection, and data preprocessing. Basic algorithms like Support Vector Machines (SVM), Random Forests, k-Nearest Neighbors (KNN), Logistic Regression, and Naive Bayes are supported in a straightforward and easy-to-use API. One of the features of Scikit-learn is that it is simple and modular, making it simple to implement, train, and test machine learning models. It also supports cross-

validation, hyperparameter adjustment (GridSearchCV), and pipelines to streamline workflows.

Widely employed in both industry and academia, Scikit-learn sits at the heart of the Python data science stack, making machine learning available to newcomers but also offering the type of flexibility that researchers and practitioners require.

4.3.2 TensorFlow

TensorFlow is an open-source end-to-end machine learning platform created by the Google Brain team and made available in 2015. TensorFlow is designed to make it simple to create and deploy machine learning models—especially deep learning models—on a range of platforms from desktop to mobile and edge computing devices.

Essentially, TensorFlow is founded on computational graphs and tensor-based data structures, making scalable computation of massive mathematical operations possible. TensorFlow is appropriate for both low-level operations for researchers and high-level APIs like Keras for quick model prototyping and development. TensorFlow is flexible and scalable with capabilities like automatic differentiation, GPU/TPU acceleration, and deployment platforms TensorFlow Lite and TensorFlow Serving. It provides a range of tasks including image recognition, natural language processing, time-series forecasting, and reinforcement learning.

Being one of the most widely used frameworks across AI and machine learning, TensorFlow is a leading contender in research, production systems, and practical applications. With its large community, extensive documentation, and integration with Python and other tools, it is a universal tool for both new and old developers to create intelligent systems.

4.3.3 Keras

Keras is an open-source, high-level Python deep learning API for neural networks. Keras was originally created in 2015 by François Chollet as a means of quickly testing deep learning models. Keras is modular, simple, and extensible, and it's straightforward enough to utilize even for inexperienced users as well as for researchers in the artificial intelligence domain.

Keras runs on top of very efficient backend engines like TensorFlow (default), Theano (legacy), and CNTK, with a high-level interface for building

and training deep learning models. Its simplicity masks the complexity of tensor operations and presents clean, readable code for building convolutional neural networks (CNNs), recurrent neural networks (RNNs), etc.

With its capability of quick prototyping and facilitation of distributed training, transfer learning, and deployment to web, mobile, and embedded platforms, Keras has become the research tool of choice for academia and industrial applications. That Keras was also incorporated into TensorFlow as tf.keras further testifies to its stability and scalability. Essentially, Keras gives deep learning the potential to make it accessible through the power of backend engines combined with the simplicity of a high-level API to facilitate fast machine learning innovation.

4.3.4 Matplotlib

Matplotlib is a widely used open-source plotting library for Python, primarily employed for creating static, interactive, and animated plots. John D. Hunter developed Matplotlib in 2003. It is a robust and versatile library for producing publication-quality plots and graphs with little effort.

Matplotlib is capable of producing a number of 2D plotting functions like line plots, bar plots, histograms, scatter plots, pie plots, etc. With the utilization of its pyplot module, it gives MATLAB-like interface so that non-coders can use it. It also supports numerical computing libraries like NumPy and Pandas, and it is used as a core tool in data analysis, scientific computing, and machine learning pipelines. The library is highly customizable, and the users can manage all aspects of a plot—titles, labels, legends, colors, styles, and axes. Object-oriented plotting, for more advanced users, and extension to three-dimensional graphics is also available in Matplotlib through mpl_toolkits. mplot3d.

As a core component of the Python data science ecosystem, Matplotlib is at the very center of visualizing and interpreting large datasets and of finding and communicating results effectively.

4.3.5 Google Colab Drive

Google Collaboratory, simply referred to as Google Colab, is a web-based free platform developed by Google for editing and executing Python code in a web browser. It is particularly popular among the data science, machine learning, and education communities because it grants users the power to leverage free GPU and TPU resources without the necessity of installing them locally.

One of the strongest features of Google Colab is integrated support for Google Drive, enabling saving, sharing, and opening Jupyter Notebooks (.ipynb files) from where they are stored in Drive. This makes it easy to collaborate in real-time with editing and versioning, similar to Google Docs or Sheets.

With Google Drive integration, users can:

- Use and upload datasets in Drive.
- Load machine learning models and scripts from Drive.
- Store output files, visualizations, and trained models directly to Drive.
- Work from any internet-connected device without requiring installations.

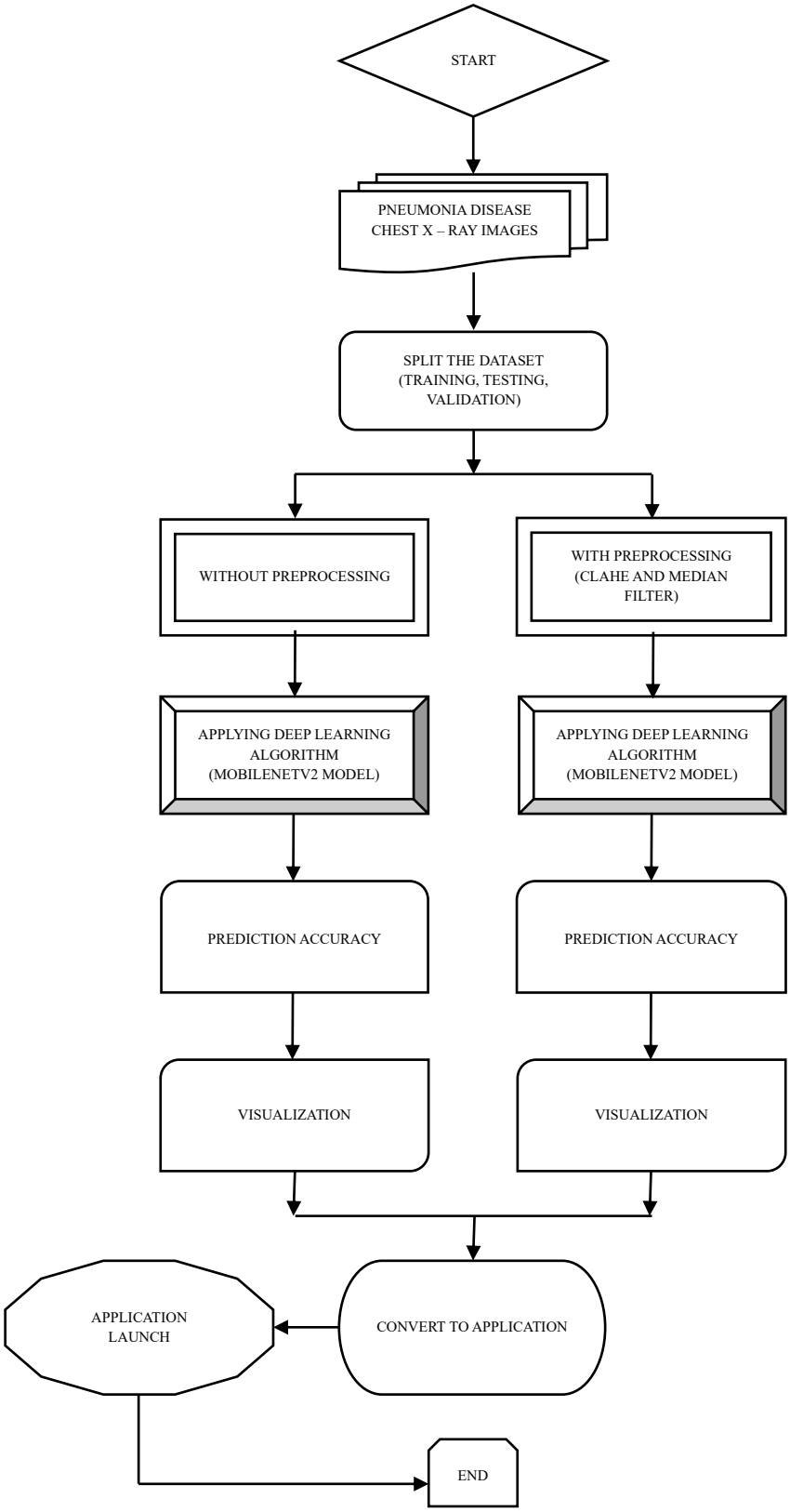
The combination of Drive and Colab offers a strong, flexible, and simple-to-use environment for coding, testing, and collaborative research. It has greatly facilitated access to high-end computing resources, particularly for students, teachers, and developers who do not have high-end equipment.

SYSTEM DESIGN

CHAPTER – V

SYSTEM DESIGN

5.1 Data Flow Diagram



SYSTEM DEVELOPMENT

CHAPTER – VI

SYSTEM DEVELOPMENT

6.1 Project Description

This project is based on the detection of Pneumonia through deep learning methods on chest X-ray images. The system is to classify and distinguish between Normal, Bacterial Pneumonia, and Viral Pneumonia cases based on image classification. The dataset employed consists of 7927 grayscale X-ray images and is split into training, validation, and test sets to provide a solid model evaluation. The images undergo preprocessing through CLAHE and Median Filtering for contrast improvement and noise reduction prior to model training. The system tests the success of preprocessing through PSNR calculation and storing the results in a CSV file. The primary technologies and tools involved are the MobileNetV2 architecture, with and without preprocessing, for comparative performance. The training process involves image normalization, RGB conversion, and model evaluation on accuracy, loss, and classification metrics on all three splits of the dataset.

6.2 Module Description

6.2.1 Get Dataset

This project uses the Pneumonia X-ray Dataset, which has a total of 7927 grayscale chest X-ray images, divided into three classes: Normal, Bacterial Pneumonia, and Viral Pneumonia. These images are utilized for training and testing deep learning models intended to detect pneumonia from X-ray images, specifically to classify the three lung conditions.

6.2.1.1 About the Dataset

The dataset consists of images captured from patients with different lung conditions:

- Bacterial Pneumonia: X-rays evidencing bacterial infection in the lungs.
- Viral Pneumonia: X-rays evidencing viral infection in the lungs.
- Normal: Normal lung X-rays.

This dataset is instrumental in training a model that will be able to distinguish between normal lungs and pneumonia-infected lungs due to various pathogens effectively.

6.2.1.2 Data Categories

The dataset is divided into three different classes:

- Bacterial Pneumonia
- Viral Pneumonia
- Normal

Each class has pictures that represent the visual variations of the lungs, which enables the model to acquire the patterns with each condition.

6.2.1.3 Dataset Format

Images are saved in grayscale format. The dataset consists of:

- Training Set (70%): Employed for model training.
- Validation Set (20%): Used to test the model while it is being trained.
- Testing Set (10%): Used to test the model's performance upon training.

Each set is organized into subfolders for the three classes: Normal, Bacterial Pneumonia, and Viral Pneumonia.

6.2.1.4 Dataset Structure

```
Pneumonia Disease Detection Dataset/
├── Training Data/
│   ├── Normal/
│   ├── Bacterial Pneumonia/
│   └── Viral Pneumonia/
├── Validation Data/
│   ├── Normal/
│   ├── Bacterial Pneumonia/
│   └── Viral Pneumonia/
└── Testing Data/
    ├── Normal/
    ├── Bacterial Pneumonia/
    └── Viral Pneumonia/
```

6.2.2 Preprocessing Dataset

Preprocessing dataset is an essential process for improving the quality of the image and getting it ready for model training. Preprocessing in this project involves **(CLAHE)** and **Median Filtering** to enhance the contrast and minimize noise in the images.

6.2.2.1 Preprocessing Steps

CLAHE (Contrast Limited Adaptive Histogram Equalization)

- **Purpose:** Increases contrast in grayscale chest X-ray images and makes significant lung features more conspicuous.
- **Parameters Used:** Clip Limit: 0.01 – prevents noise exaggeration while maintaining contrast improvement.
- **Effect:** Accentuates fine details and textures of lung tissues necessary to differentiate normal from pneumonia cases.

Median Filtering

- **Purpose:** Eliminates salt-and-pepper noise and maintains edges.
- **Kernel Size:** 1
- **Effect:** Denoising performed by light applied; kernel size 1 is very little filtering, only what's needed to blur without loss of fine detail.

Moreover, the Peak Signal-to-Noise Ratio (PSNR) is computed prior to and after preprocessing in order to analyse the performance of these methods. The output is stored into a CSV file for future analysis.

6.2.3 Applying Deep Learning Model

Here, the MobileNetV2 deep learning model is used to classify pneumonia chest X-ray images into three categories: Normal, Bacterial Pneumonia, and Viral Pneumonia. To analyse the performance of preprocessing methods, the model is trained in two environments:

6.2.3.1 MobileNetV2 Overview

- **Description:** MobileNetV2 is a light-weight convolutional neural network for efficient deployment on mobile and embedded devices.
- **Key Feature:** Applies depth wise separable convolutions to minimize computational expense while achieving high performance.
- **Use Case:** Real-time medical image classification tasks are appropriate due to its efficiency and speed.

6.2.3.2 Training Setup

- **Epochs:** 25
- **Batch Size:** 32
- **Input Image Size:** 128 × 128

6.2.3.3 Model Training without Preprocessing

Original grayscale chest X-rays were used without any contrast enhancement or noise reduction operations for training the MobileNetV2 model. None of these operations were performed before training. Images were normalized to provide stable input to the model.

Dataset	Accuracy (%)	Loss
Training	94%	0.1693
Testing	83%	0.4315
Validation	84%	0.4310

6.2.3.4 Model Training with Preprocessing

Before training, gray-scale chest X-ray images were preprocessed to improve quality. Specifically, Contrast Limited Adaptive Histogram Equalization (CLAHE) with a clip limit of 0.01 was used to enhance contrast, and then Median Filtering with kernel size 1 was used to remove noise. Following preprocessing, images were normalized to ready them for input into the MobileNetV2 model.

Dataset	Accuracy (%)	Loss
Training	99%	0.0454
Testing	84%	0.5891
Validation	86%	0.5016

6.2.4 Performance Metrics

After the training of the **MobileNetV2** model, its performance is compared on the Training, Validation, and Testing sets using several metrics for evaluation. The major metric of evaluation is accuracy, which is used to calculate the fraction of images classified correctly.

6.2.4.1 Accuracy

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions}$$

Although accuracy gives a general idea about the performance of the model, it might not always indicate class-specific performance if the data is imbalanced.

6.2.4.2 Confusion Matrix

Visualizing the performance of the model by showing the true positives, false positives, true negatives, and false negatives for every class.

6.2.4.3 Precision

Ratio of true positive predictions to all predicted positives.

$$Precision = \frac{TP}{TP + FN}$$

6.2.4.4 Recall (Sensitivity)

Ratio of true positive predictions to all actual positives.

$$Recall = \frac{TP}{TP + FN}$$

6.2.4.5 F1-Score

Harmonic mean of Precision and Recall.

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

6.2.4.6 Accuracy and Loss Curves

Plots indicating the accuracy and loss of the model during training and validation to track overfitting and convergence.

6.2.5 Evaluation Metrics Table:

Method	Precision (%)	Recall (%)	F1-Score (%)
With Preprocess	84%	84%	84%
Without Preprocess	83%	83%	83%

These measures reflect that the model performs well consistently on all datasets with high precision and recall, resulting in good generalization ability.

6.2.6 Convert into Mobile Application:

Once the top-performing model is identified, the last step is to deploy the model into a mobile app. The application provides users with the ability to upload images of chest X-rays and receive real-time predictions of Normal, Bacterial Pneumonia, or Viral Pneumonia.

Tools Used for Mobile Application Development:

- **Android Studio:** The IDE for Android used to create the mobile app. The trained model is embedded into the application for real-time predictions.
- **TensorFlow Lite:** Utilized to convert the trained model into a light format that is supported by mobile devices.
- **Google Colab:** Utilized for evaluating and training the model. The trained model is exported and is incorporated into the Android application.

The mobile application offers a user-friendly interface to doctors and healthcare professionals to easily diagnose pneumonia in patients using their X-ray images.

RESULTS AND DISCUSSION

CHAPTER – VII

RESULTS AND DISCUSSION

7.1 RESULTS

Image Preprocessing

Preprocessing is an important step in improving the diagnostic quality of medical images. For this project, every chest X-ray image was processed using **(CLAHE)** to enhance local contrast, and **Median Filtering** to reduce noise while maintaining edge information. To assess the quality of the improved images, the **(PSNR)** was computed across the dataset. The resulting average **PSNR** was **48.86** dB, indicating a favourable trade-off between noise reduction and feature preservation. These improvements probably helped improve learning and model generalization in training.

Model Performance

The MobileNetV2 model was trained and tested on both raw chest X-ray images and preprocessed images with CLAHE and median filtering. Without preprocessing, the model got a training accuracy of around 94%, F1 score of 83%, recall of 83%, and precision of 83%. Contrary to this, the model learned from preprocessed images performed considerably better, with a training accuracy of approximately 99%, an F1 score of 84%, a recall of 84%, and a precision of 84%. These results prove the efficiency of preprocessing methods in improving feature quality and enabling more precise learning during model training.

PSNR

Peak Signal-to-Noise Ratio (PSNR) is a common image processing measure that represents the quality of a processed image in relation to its original. PSNR measures the ratio of the maximum power of a signal (the original image) and the power of disturbing noise (the introduced differences in processing). PSNR is measured in decibels (dB), with larger values showing that the processed image is similar to the original, which suggests better quality.

7.2 DISCUSSION

The outcomes unequivocally reveal that the usage of preprocessing methods such as CLAHE and Median Filtering greatly improves the classification rate of the MobileNetV2 model. Interestingly, I tested CLAHE clip limits between 0.4 and 0.01 and found that a clip limit of 0.01 provided better Peak Signal-to-Noise Ratio (PSNR) values, reflecting greater image quality and detail retention. This improvement in image quality helped with better generalization and fewer misclassifications, especially between similar classes like bacterial and viral pneumonia. Visual evaluations with confusion matrices and classification reports also confirm that preprocessing allows the model to better recognize subtle visual patterns in chest X-rays, resulting in more accurate and trustworthy predictions.

SYSTEM IMPLEMENTATION AND MAINTENANCE

CHAPTER – VIII

SYSTEM IMPLEMENTATION AND MAINTENANCE

8.1 SYSTEM IMPLEMENTATION

The implementation consisted of some of the major activities:

8.1.1 Model Conversion:

- The MobileNetV2 model, trained and tested in Google Colab, was transformed to TensorFlow Lite (.tflite) format for compatibility with mobile devices.
- This guaranteed smaller model size, quicker inference time, and smaller memory consumption.

8.1.2 Android App Development:

- The Android app was implemented using Android Studio.
- XML was utilized to design the UI, and Java took care of backend tasks such as image selection, preprocessing (if necessary), model loading, and result visualization.
- Users can upload their chest X-ray images and get an instant prediction: Bacterial Pneumonia, Viral Pneumonia and Normal.

8.1.3 Integration of Model:

- The TFLite model was integrated into the application using TensorFlow Lite Interpreter after converting it to TFLite model.
- Efficient handling was implemented to resize images, convert to RGB, and normalize prior to model inference.

8.1.4 User Interface (UI):

- Created a simple, user-friendly interface with well-defined buttons and outputs.
- Display results in an easy-to-understand format with color-coded results for clarity.

8.1.5 Deployment:

- The application was tested on several Android devices and emulators to ensure responsiveness and compatibility.

- Upon validation, the APK was packaged for local use or external sharing.

8.2 System Maintenance:

- Regular maintenance is necessary after deployment to keep the system running at its best as technologies change and new user input is obtained.

8.2.1 Model Updates:

- When new X-ray data is available, the model can be updated and retrained to enhance accuracy and performance.
- The app facilitates simple replacement of the TFLite model file with newer versions.

8.2.2 Bug Fixes and Performance Optimization:

- User feedback and logs are utilized to determine any bugs or problems in the app.
- Bugs are fixed, UI/UX is enhanced, and performance is improved by rolling out updates.

8.2.3 Compatibility Maintenance:

- It is important to ensure compatibility with new versions of Android OS and TensorFlow Lite.
- The app is periodically tested when new OS versions become available.

8.2.4 Security and Privacy:

- The system is designed to keep patient data (X-ray images) from being stored or transmitted, preserving user privacy.
- Future versions can incorporate encryption and privacy-protecting features for added security.

8.2.5 Scalability:

- The application is developed to be scalable for future integration with hospital administration systems or cloud databases.

CONCLUSION

CHAPTER – IX

CONCLUSION

The Pneumonia Disease Detection System developed in the present project illustrates perfectly the combining of deep learning algorithms and mobile technology to enable pneumonia diagnosis. With the efficient but lightweight MobileNetV2 model, the system classifies chest X-rays into three Categories: Normal, Bacterial Pneumonia, and Viral Pneumonia. To compare the performance of the model, two training conditions were compared: one with original chest X-ray images, and the other with images enhanced using CLAHE and Median Filtering. The preprocessing greatly enhanced image contrast and minimized noise, which resulted in improved feature extraction and improved classification performance. Experimental findings indicated that the model on preprocessed images outperformed the model on raw images on all major evaluation metrics like accuracy, precision, recall, and F1-score, which is indicative of the significance of image enhancement methods in medical image processing.

The trained model was optimized into TensorFlow Lite and included within a user-friendly Android application. This software supports real-time detection of pneumonia through facilitating the users—i.e., clinicians and healthcare workers at remote clinics—to upload X-ray chest images and achieve real-time classification feedback. The smartphone application enhances convenience, portability, and use, particularly within low-resource settings. Thorough testing—unit, integration, system, and performance testing—validated the app's accuracy, reliability, and responsiveness. It met all functional and non-functional specifications, providing quick and accurate results.

In conclusion, this project not only demonstrates the potential of deep learning in medical image analysis but also presents a usable, mobile-based solution that closes the gap between AI innovation and actual healthcare. It presents a useful diagnostic tool to help aid early pneumonia detection and decrease the workload for medical professionals, leading to improved patient outcomes.

FUTURE ENHANCEMENT

CHAPTER – X

FUTURE ENHANCEMENT

In the future, the pneumonia disease detection system can be improved in many ways to make it more functional, accurate, and user-friendly. The model can be trained to detect a broader spectrum of chest-related diseases like tuberculosis, lung cancer, COVID-19, and pulmonary edema to make it an even more thorough diagnostic tool. Integrating techniques such as Grad-CAM for explainable AI can assist in visualizing where on the X-ray image affected the decision of the model, bringing transparency and assisting physicians with interpretation. Having a confidence level assigned to each prediction would enable medical practitioners to evaluate the reliability of the results.

Live image capture using camera integration can reduce the process of prediction, and uploads can be avoided. Hosting the model on cloud platforms such as Firebase or AWS would minimize the size of the app and provide for model updates automatically. In order to make the app more user-friendly, regional language support and voice assistance can be provided. Moreover, incorporating electronic health record (EHR) functionalities will allow patient history tracking and comparison with the past diagnoses. More training on bigger and more diversified datasets can also increase prediction accuracy under different imaging conditions.

Optimizing the app for offline capability would guarantee performance in regions with limited or no internet connection. Lastly, the use of regular updates based on new medical data will guarantee that the system remains relevant and accurate over time.

BIBLIOGRAPHY

CHAPTER – XI

BIBLIOGRAPHY

11.1 BOOK REFERENCE

1. "Deep Learning", Ian Goodfellow, Yoshua Bengio, and Aaron Courville, 1st Edition, MIT Press, 2016
2. "Hands-On Convolutional Neural Networks with TensorFlow", Iffat Zafar, Giounona Tzanidou, and Paolo Galeone, 1st Edition, Packt Publishing, 2018
3. "Deep Learning for Medical Image Analysis", S. Kevin Zhou, Hayit Greenspan, and Dinggang Shen, 1st Edition, Academic Press, 2017
4. "Mobile Deep Learning with TensorFlow Lite, ML Kit and Flutter", Anubhav Singh and Rimjhim Bhadani, 1st Edition, Packt Publishing, 2020
5. "Image Processing, Analysis, and Machine Vision", Milan Sonka, Vaclav Hlavac, and Roger Boyle, 4th Edition, Cengage Learning, 2014

11.2 WEB REFERENCE

1. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. arXiv. https://arxiv.org/abs/1801.04381

2. Mooney, P. (2018). Chest X-Ray Images (Pneumonia). Kaggle. https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia

3. TensorFlow. (2024). TensorFlow Lite – Deploying Image Classification Models on Mobile. https://www.tensorflow.org/lite/models/image_classification/overview

4. OpenCV Team. (2024). CLAHE – Contrast Limited Adaptive Histogram Equalization. https://docs.opencv.org/4.x/d5/daf/tutorial_py_histogram_equalization.html

5. Scikit-Image Developers. (2024). Median Filtering – Rank Filters Documentation. https://scikit-image.org/docs/stable/auto_examples/filters/plot_rank_filters.html

APPENDIX

CHAPTER – XII

APPENDIX

A – SAMPLE CODE

CLAHE & Median Filter Preprocessed Method with PSNR Value

```
import os
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from google.colab import drive
drive.mount('/content/drive')
def calculate_psnr(original, processed):mse =
np.mean((original.astype(np.float32) - processed.astype(np.float32)) ** 2)
    if mse == 0:
        return float('inf')
    PIXEL_MAX = 255.0
    return 20 * np.log10(PIXEL_MAX / np.sqrt(mse))
def preprocess_and_save(image_path, save_path, show_sample=False):
    original = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if original is None:
        print(f"❌ Failed to load: {image_path}")
        return None, None
    # Apply CLAHE
    clahe = cv2.createCLAHE(clipLimit=0.01, tileGridSize=(8, 8))
    img_clahe = clahe.apply(original)
    # Apply Median Filter
    img_combined = cv2.medianBlur(img_clahe, 1)
```

```
    # Save image
    cv2.imwrite(save_path, img_combined)
    print(f"✅ Saved: {save_path}")
    # PSNR
    psnr = calculate_psnr(original, img_combined)
    # Display one sample
    if show_sample:
        img_median = cv2.medianBlur(original, 1)
        plt.figure(figsize=(16, 4))
        plt.subplot(1, 4, 1); plt.imshow(original, cmap='gray'); plt.title("Original");
plt.axis('off')
        plt.subplot(1, 4, 2); plt.imshow(img_clahe, cmap='gray');
plt.title("CLAHE"); plt.axis('off')
        plt.subplot(1, 4, 3); plt.imshow(img_median, cmap='gray');
plt.title("Median"); plt.axis('off')
        plt.subplot(1, 4, 4); plt.imshow(img_combined, cmap='gray');
plt.title("CLAHE + Median"); plt.axis('off')
        plt.suptitle(os.path.basename(image_path))
        plt.show()
    return os.path.basename(save_path), psnr
# Main Pipeline Function
def run_preprocessing_pipeline(input_base, output_base, psnr_csv_path):
    sets = ["Training Data", "Testing Data", "Validation Data"]
    classes = ["Normal", "Bacterial Pneumonia", "Viral Pneumonia"]
    psnr_records = []
    grand_total = 0
    for dataset in sets:
        print(f"\n📁 Dataset: {dataset}")
```

```
    dataset_total = 0
    for cls in classes:
        input_dir = os.path.join(input_base, dataset, cls)
        output_dir = os.path.join(output_base, dataset, cls)
        os.makedirs (output_dir, exist_ok=True files = sorted([f for f in
os.listdir(input_dir) if f.lower().endswith(('.jpg', '.jpeg', '.png'))])
        class_total = len(files)
        print(f"  {cls}: {class_total} images")
        sample_shown = False
        counter = 1
        for f in files:
            input_path = os.path.join(input_dir, f)
            folder_clean = dataset.replace(" ", "_")
            class_clean = cls.replace(" ", "_")
            method_name = "Clahe_&_Median"
            new_filename =
f"{folder_clean}_{class_clean}_{method_name}_{counter:04d}.png"
            output_path = os.path.join(output_dir, new_filename)
            saved_name, psnr = preprocess_and_save(input_path, output_path,
show_sample=not sample_shown)
            sample_shown = True
            if saved_name:
                psnr_records.append({
                    "Dataset": dataset,
                    "Class": cls,
                    "Filename": new_filename,
                    "PSNR": psnr
                })
            counter += 1
```

```
            dataset_total += class_total
            grand_total += class_total
        print(f"🔢 Total in {dataset}: {dataset_total} images")
    print(f"\n✅ Grand Total: {grand_total} images processed across all datasets.\n")
    # Save PSNR records to CSV
    df = pd.DataFrame(psnr_records)
    os.makedirs(os.path.dirname(psnr_csv_path), exist_ok=True)
    df.to_csv(psnr_csv_path, index=False)
    print(f"📄 PSNR CSV saved at: {psnr_csv_path}")
# Set paths
input_base = "/content/drive/MyDrive/Pneumonia Disease Detection Dataset"
output_base = "/content/drive/MyDrive/Pneumonia Disease Detection CLAHE & Median Filter Preprocessed Dataset"
psnr_csv_path = os.path.join(output_base, "CLAHE & Median Filter Preprocessed PSNR Value.csv")
# Start processing
run_preprocessing_pipeline(input_base, output_base, psnr_csv_path)
# Final Success Message 🎉
print("\n" + "="*60)
print("🎯 All folders and subfolders processed successfully!")
print("📁 Output saved in:")
print(f"   {output_base}")
print("📝 PSNR values saved in:")
print(f"   {psnr_csv_path}")
print("✅ Preprocessing complete. You are now ready for model training!")
print("="*60)
```

MobileNetV2 Trained Model

```
# STEP 1: Import packages and mount Google Drive
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns
from google.colab import drive
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout,
Activation
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.activations import swish, relu, softmax
from tensorflow.keras.callbacks import ModelCheckpoint
from sklearn.metrics import confusion_matrix, classification_report, f1_score,
recall_score, precision_score
# Mount Google Drive
drive.mount('/content/drive')
# Dataset structure path and training settings
base_dir = "/content/drive/MyDrive/Pneumonia Disease Detection Dataset"
train_dir = os.path.join(base_dir, "Training Data")
val_dir = os.path.join(base_dir, "Validation Data")
test_dir = os.path.join(base_dir, "Testing Data")

# Image and training settings
img_size = (128, 128)
```

```
batch_size = 32
epochs = 25
# Create image generators for training, validation, and testing
train_gen = ImageDataGenerator(rescale=1./255).flow_from_directory(
    train_dir, target_size=img_size, color_mode='rgb',
    batch_size=batch_size, class_mode='categorical', shuffle=True)
val_gen = ImageDataGenerator(rescale=1./255).flow_from_directory(
    val_dir, target_size=img_size, color_mode='rgb',
    batch_size=batch_size, class_mode='categorical', shuffle=False)
test_gen = ImageDataGenerator(rescale=1./255).flow_from_directory(
    test_dir, target_size=img_size, color_mode='rgb',
    batch_size=batch_size, class_mode='categorical', shuffle=False)
# STEP 3: Create MobileNetV2 model architecture
print("🚀 Creating new MobileNetV2 model...")
base_model = MobileNetV2(input_shape=(128, 128, 3), include_top=False,
weights='imagenet')
base_model.trainable = False
# Add custom top layers
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(256)(x)
x = Activation(swish)(x)
x = Dense(128)(x)
x = Activation(relu)(x)
x = Dropout(0.3)(x)
output = Dense(3, activation=softmax)(x)
model = Model(inputs=base_model.input, outputs=output)
# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001),
```

```
            loss='categorical_crossentropy',
            metrics=['accuracy'])
model.summary()
# ModelCheckpoint callback to save best model based on validation accuracy
checkpoint_path = "/content/drive/MyDrive/Best MobileNetV2 Trained
Model.h5"
checkpoint = ModelCheckpoint(checkpoint_path, monitor='val_accuracy',
save_best_only=True, verbose=1)
# STEP 4: Train the model with checkpoint callback
history = model.fit(
    train_gen,
    validation_data=val_gen,
    epochs=epochs,
    callbacks=[checkpoint]
)
# Save final trained model only (for future reference)
model.save("/content/drive/MyDrive/MobileNetV2 Trained Model.h5")
print("✅ Final model saved as MobileNetV2 Trained Model.h5")
# STEP 5: Evaluate model performance on the test set
loss, accuracy = model.evaluate(test_gen)
print(f"\n✅ Test Accuracy: {accuracy*100:.2f}%")
print(f"✅ Test Loss: {loss:.4f}")
# Plot accuracy and loss trends during training
metrics = pd.DataFrame(history.history)
# Accuracy plot
metrics[['accuracy', 'val_accuracy']].plot(title="Model Accuracy", grid=True)
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
```

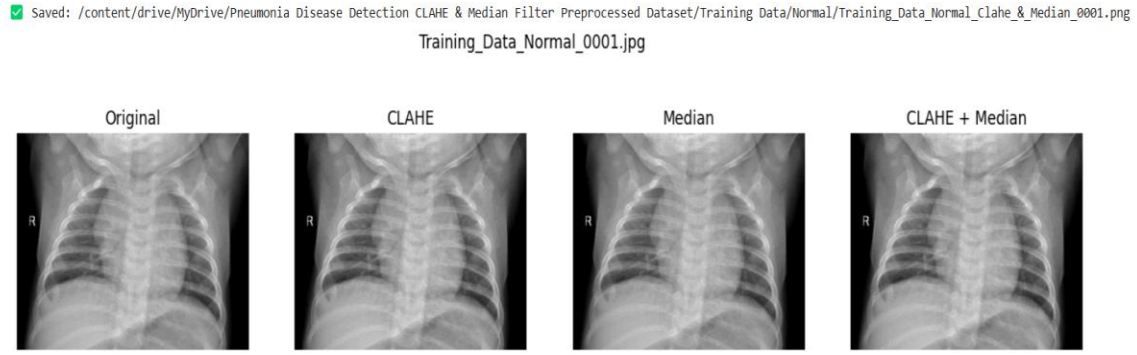
```
plt.show()
# Loss plot
metrics[['loss', 'val_loss']].plot(title="Model Loss", grid=True)
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.show()
# STEP 5.1: Generate predictions for the test set
y_true = test_gen.classes
y_pred_probabilities = model.predict(test_gen, verbose=1)
y_pred = np.argmax(y_pred_probabilities, axis=1)
class_labels = list(test_gen.class_indices.keys())
# STEP 5.2: Print classification report and calculate additional metrics
print("\n📄 Classification Report:")
print(classification_report(y_true, y_pred, target_names=class_labels))
f1 = f1_score(y_true, y_pred, average='weighted')
recall = recall_score(y_true, y_pred, average='weighted')
precision = precision_score(y_true, y_pred, average='weighted')
print(f"\n✅ Weighted F1 Score: {f1:.4f}")
print(f"✅ Weighted Recall: {recall:.4f}")
print(f"✅ Weighted Precision: {precision:.4f}")
# STEP 5.3: Plot the confusion matrix
cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_labels,
yticklabels=class_labels)
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
```

```
plt.show()
# STEP 6: Print and save final metrics summary
print("\n📊 Final Metrics Table:")
final_metrics = {
    "Dataset": ["Training", "Validation", "Test"],
    "Accuracy": [
        metrics['accuracy'].iloc[-1],
        metrics['val_accuracy'].iloc[-1],
        accuracy
    ],
    "Loss": [
        metrics['loss'].iloc[-1],
        metrics['val_loss'].iloc[-1],
        loss
    ],
    "F1 Score": ["-", "-", f1],
    "Recall": ["-", "-", recall],
    "Precision": ["-", "-", precision]
}
df_final = pd.DataFrame(final_metrics)
print(df_final.round(4))

# Save training history to a CSV file
metrics.to_csv("/content/drive/MyDrive/MobileNetV2 Training Model
History.csv", index=False)
print("✅ History saved to MobileNetV2 Training Model History.csv")
```

B – SCREENSHOTS

Preprocessing Images and CSV File Images



Dataset	Class	Filename	PSNR
Training Data	Normal	Training_Data_Norm	50.225063
Training Data	Normal	Training_Data_Norm	50.301746
Training Data	Normal	Training_Data_Norm	51.472633
Training Data	Normal	Training_Data_Norm	50.26771
Training Data	Normal	Training_Data_Norm	51.887787
Training Data	Normal	Training_Data_Norm	52.70884
Training Data	Normal	Training_Data_Norm	54.29934
Training Data	Normal	Training_Data_Norm	53.932648
Training Data	Normal	Training_Data_Norm	49.39975
Training Data	Normal	Training_Data_Norm	49.43656
Training Data	Normal	Training_Data_Norm	49.11469
Training Data	Normal	Training_Data_Norm	49.47954
Training Data	Normal	Training_Data_Norm	49.445343
Training Data	Normal	Training_Data_Norm	52.80902
Training Data	Normal	Training_Data_Norm	49.891487
Training Data	Normal	Training_Data_Norm	49.537548
Training Data	Normal	Training_Data_Norm	50.165066
Training Data	Normal	Training_Data_Norm	50.29357
Training Data	Normal	Training_Data_Norm	49.95953
Training Data	Normal	Training_Data_Norm	49.79919
Training Data	Normal	Training_Data_Norm	52.98931
Training Data	Normal	Training_Data_Norm	48.592587
Training Data	Normal	Training_Data_Norm	48.607197
Training Data	Normal	Training_Data_Norm	49.712074
Training Data	Viral Pneumonia	Training_Data_Viral	51.104607
Training Data	Viral Pneumonia	Training_Data_Viral	47.222084
Training Data	Viral Pneumonia	Training_Data_Viral	48.879543
Training Data	Viral Pneumonia	Training_Data_Viral	50.07141
Training Data	Viral Pneumonia	Training_Data_Viral	49.105606
Training Data	Viral Pneumonia	Training_Data_Viral	49.19735
Training Data	Viral Pneumonia	Training_Data_Viral	49.132088
Training Data	Viral Pneumonia	Training_Data_Viral	48.769066
Training Data	Viral Pneumonia	Training_Data_Viral	48.980274
Training Data	Viral Pneumonia	Training_Data_Viral	49.89997
Training Data	Viral Pneumonia	Training_Data_Viral	45.791107
Training Data	Viral Pneumonia	Training_Data_Viral	43.887638
Training Data	Viral Pneumonia	Training_Data_Viral	52.43027
Testing Data	Normal	Testing_Data_Norma	51.854088
Testing Data	Normal	Testing_Data_Norma	50.69906
Testing Data	Normal	Testing_Data_Norma	50.06397
Testing Data	Normal	Testing_Data_Norma	49.565895
Testing Data	Normal	Testing_Data_Norma	49.383423
Testing Data	Normal	Testing_Data_Norma	50.0822
Testing Data	Normal	Testing_Data_Norma	49.716507
Testing Data	Normal	Testing_Data_Norma	49.254646
Testing Data	Normal	Testing_Data_Norma	38.492435
Testing Data	Normal	Testing_Data_Norma	49.83371
Testing Data	Normal	Testing_Data_Norma	50.316757
Testing Data	Normal	Testing_Data_Norma	51.53357
Testing Data	Normal	Testing_Data_Norma	50.36556

MobileNetV2 Model

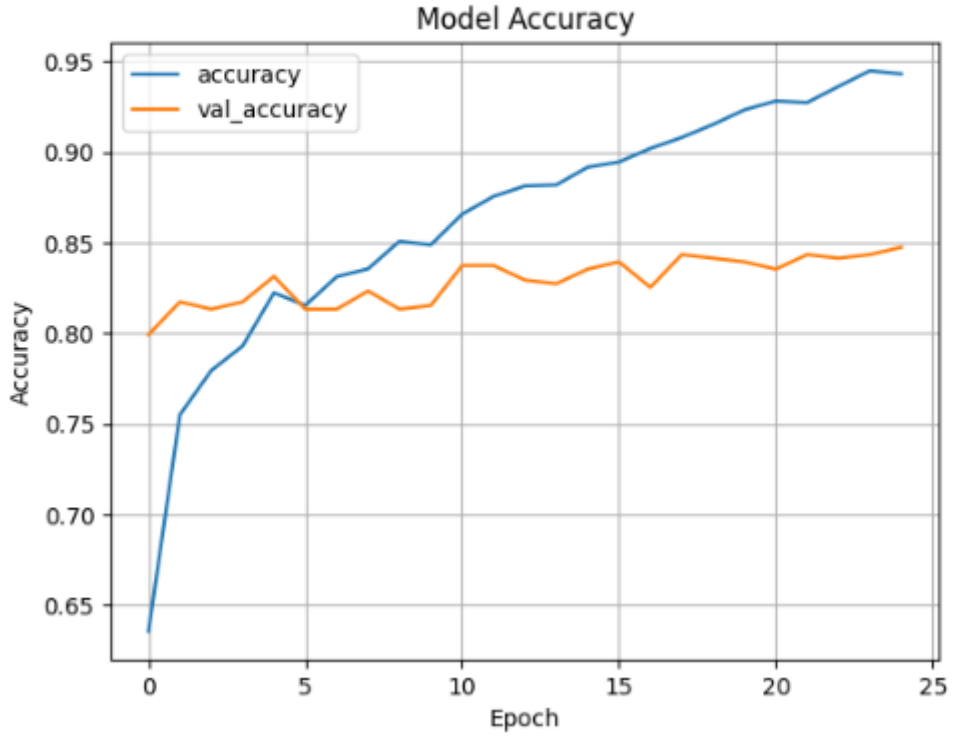
Epoch

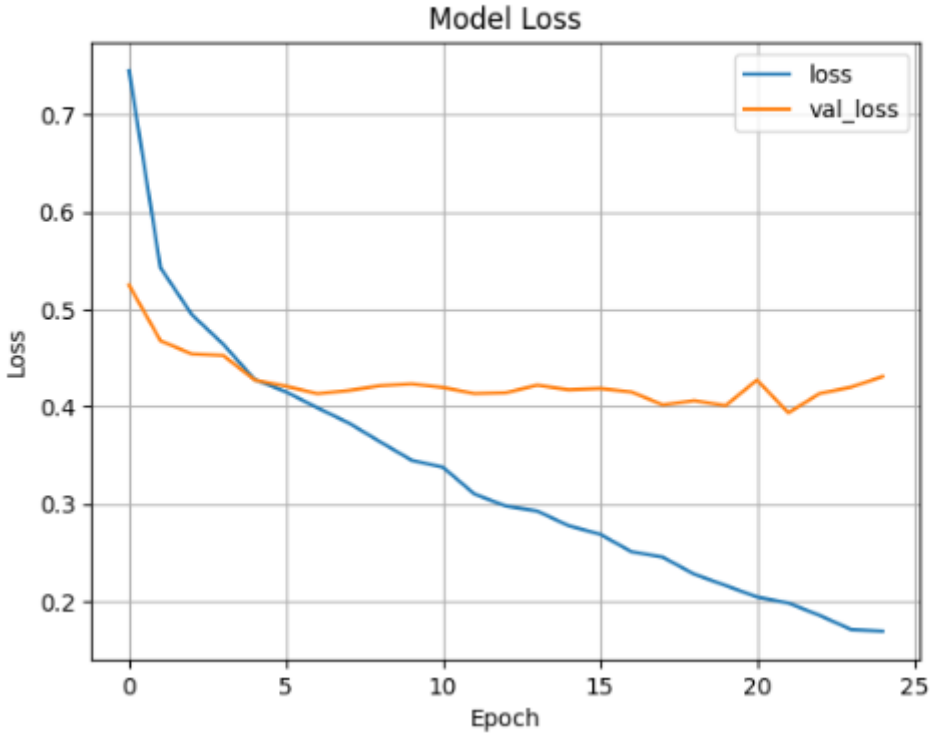
```
Epoch 1/25
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 327ms/step - accuracy: 0.9513 - loss: 0.1595
Epoch 1: val_accuracy did not improve from 0.85542
109/109 ━━━━━━━━━━━━━━━━━━━━ 42s 388ms/step - accuracy: 0.9514 - loss: 0.1594 - val_accuracy: 0.8454 - val_loss: 0.4039
Epoch 2/25
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 311ms/step - accuracy: 0.9503 - loss: 0.1476
Epoch 2: val_accuracy did not improve from 0.85542
109/109 ━━━━━━━━━━━━━━━━━━━━ 80s 370ms/step - accuracy: 0.9502 - loss: 0.1477 - val_accuracy: 0.8434 - val_loss: 0.4132
Epoch 3/25
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 316ms/step - accuracy: 0.9557 - loss: 0.1495
Epoch 3: val_accuracy improved from 0.85542 to 0.86345, saving model to /content/drive/MyDrive/Best MobileNetV2 Trained Model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_mod
109/109 ━━━━━━━━━━━━━━━━━━━━ 41s 380ms/step - accuracy: 0.9557 - loss: 0.1495 - val_accuracy: 0.8635 - val_loss: 0.3809
Epoch 4/25
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 319ms/step - accuracy: 0.9657 - loss: 0.1261
Epoch 4: val_accuracy did not improve from 0.86345
109/109 ━━━━━━━━━━━━━━━━━━━━ 40s 371ms/step - accuracy: 0.9656 - loss: 0.1262 - val_accuracy: 0.8394 - val_loss: 0.4188
Epoch 5/25
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 322ms/step - accuracy: 0.9527 - loss: 0.1370
Epoch 5: val_accuracy did not improve from 0.86345
109/109 ━━━━━━━━━━━━━━━━━━━━ 41s 373ms/step - accuracy: 0.9527 - loss: 0.1370 - val_accuracy: 0.8474 - val_loss: 0.4623
Epoch 6/25
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 319ms/step - accuracy: 0.9633 - loss: 0.1239
Epoch 6: val_accuracy improved from 0.86345 to 0.86948, saving model to /content/drive/MyDrive/Best MobileNetV2 Trained Model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_mod
109/109 ━━━━━━━━━━━━━━━━━━━━ 42s 385ms/step - accuracy: 0.9633 - loss: 0.1239 - val_accuracy: 0.8695 - val_loss: 0.4045
Epoch 7/25
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 324ms/step - accuracy: 0.9647 - loss: 0.1149
Epoch 7: val_accuracy did not improve from 0.86948
109/109 ━━━━━━━━━━━━━━━━━━━━ 41s 373ms/step - accuracy: 0.9647 - loss: 0.1149 - val_accuracy: 0.8594 - val_loss: 0.4229
Epoch 8/25
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 321ms/step - accuracy: 0.9738 - loss: 0.1067
Epoch 8: val_accuracy did not improve from 0.86948
109/109 ━━━━━━━━━━━━━━━━━━━━ 40s 372ms/step - accuracy: 0.9738 - loss: 0.1067 - val_accuracy: 0.8514 - val_loss: 0.4296
Epoch 9/25
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 323ms/step - accuracy: 0.9730 - loss: 0.0968
Epoch 9: val_accuracy did not improve from 0.86948
109/109 ━━━━━━━━━━━━━━━━━━━━ 41s 374ms/step - accuracy: 0.9730 - loss: 0.0968 - val_accuracy: 0.8635 - val_loss: 0.4340
Epoch 10/25
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 322ms/step - accuracy: 0.9723 - loss: 0.1001
Epoch 10: val_accuracy did not improve from 0.86948
109/109 ━━━━━━━━━━━━━━━━━━━━ 41s 372ms/step - accuracy: 0.9723 - loss: 0.1001 - val_accuracy: 0.8534 - val_loss: 0.4312
Epoch 11/25
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 325ms/step - accuracy: 0.9797 - loss: 0.0895
Epoch 11: val_accuracy did not improve from 0.86948
109/109 ━━━━━━━━━━━━━━━━━━━━ 46s 419ms/step - accuracy: 0.9798 - loss: 0.0894 - val_accuracy: 0.8574 - val_loss: 0.4529
Epoch 12/25
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 323ms/step - accuracy: 0.9823 - loss: 0.0739
Epoch 12: val_accuracy did not improve from 0.86948
109/109 ━━━━━━━━━━━━━━━━━━━━ 41s 377ms/step - accuracy: 0.9823 - loss: 0.0740 - val_accuracy: 0.8474 - val_loss: 0.4663
Epoch 13/25
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 328ms/step - accuracy: 0.9822 - loss: 0.0764
Epoch 13: val_accuracy did not improve from 0.86948
109/109 ━━━━━━━━━━━━━━━━━━━━ 41s 380ms/step - accuracy: 0.9822 - loss: 0.0764 - val_accuracy: 0.8574 - val_loss: 0.4559
Epoch 14/25
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 328ms/step - accuracy: 0.9822 - loss: 0.0730
Epoch 14: val_accuracy improved from 0.86948 to 0.87349, saving model to /content/drive/MyDrive/Best MobileNetV2 Trained Model.h5
```

```
Epoch 15: val_accuracy did not improve from 0.87349
109/109 ━━━━━━━━━━━━━━━━━━━━ 41s 372ms/step - accuracy: 0.9850 - loss: 0.0678 - val_accuracy: 0.8594 - val_loss: 0.4542
Epoch 16/25
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 324ms/step - accuracy: 0.9835 - loss: 0.0642
Epoch 16: val_accuracy did not improve from 0.87349
109/109 ━━━━━━━━━━━━━━━━━━━━ 41s 374ms/step - accuracy: 0.9835 - loss: 0.0642 - val_accuracy: 0.8735 - val_loss: 0.4352
Epoch 17/25
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 321ms/step - accuracy: 0.9910 - loss: 0.0572
Epoch 17: val_accuracy did not improve from 0.87349
109/109 ━━━━━━━━━━━━━━━━━━━━ 40s 370ms/step - accuracy: 0.9910 - loss: 0.0573 - val_accuracy: 0.8554 - val_loss: 0.4812
Epoch 18/25
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 320ms/step - accuracy: 0.9891 - loss: 0.0544
Epoch 18: val_accuracy did not improve from 0.87349
109/109 ━━━━━━━━━━━━━━━━━━━━ 40s 370ms/step - accuracy: 0.9891 - loss: 0.0545 - val_accuracy: 0.8454 - val_loss: 0.5116
Epoch 19/25
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 318ms/step - accuracy: 0.9927 - loss: 0.0478
Epoch 19: val_accuracy did not improve from 0.87349
109/109 ━━━━━━━━━━━━━━━━━━━━ 40s 368ms/step - accuracy: 0.9927 - loss: 0.0479 - val_accuracy: 0.8715 - val_loss: 0.4600
Epoch 20/25
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 319ms/step - accuracy: 0.9900 - loss: 0.0519
Epoch 20: val_accuracy did not improve from 0.87349
109/109 ━━━━━━━━━━━━━━━━━━━━ 40s 370ms/step - accuracy: 0.9900 - loss: 0.0519 - val_accuracy: 0.8655 - val_loss: 0.4791
Epoch 21/25
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 317ms/step - accuracy: 0.9883 - loss: 0.0508
Epoch 21: val_accuracy did not improve from 0.87349
109/109 ━━━━━━━━━━━━━━━━━━━━ 40s 367ms/step - accuracy: 0.9884 - loss: 0.0507 - val_accuracy: 0.8414 - val_loss: 0.5354
Epoch 22/25
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 320ms/step - accuracy: 0.9929 - loss: 0.0434
Epoch 22: val_accuracy improved from 0.87349 to 0.87550, saving model to /content/drive/MyDrive/Best MobileNetV2 Trained Model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_mod
109/109 ━━━━━━━━━━━━━━━━━━━━ 41s 379ms/step - accuracy: 0.9929 - loss: 0.0434 - val_accuracy: 0.8755 - val_loss: 0.4773
Epoch 23/25
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 312ms/step - accuracy: 0.9943 - loss: 0.0402
Epoch 23: val_accuracy did not improve from 0.87550
109/109 ━━━━━━━━━━━━━━━━━━━━ 81s 368ms/step - accuracy: 0.9943 - loss: 0.0403 - val_accuracy: 0.8755 - val_loss: 0.4716
Epoch 24/25
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 313ms/step - accuracy: 0.9958 - loss: 0.0337
Epoch 24: val_accuracy did not improve from 0.87550
109/109 ━━━━━━━━━━━━━━━━━━━━ 40s 368ms/step - accuracy: 0.9957 - loss: 0.0338 - val_accuracy: 0.8614 - val_loss: 0.5349
Epoch 25/25
109/109 ━━━━━━━━━━━━━━━━━━━━ 0s 317ms/step - accuracy: 0.9848 - loss: 0.0555
Epoch 25: val_accuracy did not improve from 0.87550
109/109 ━━━━━━━━━━━━━━━━━━━━ 40s 370ms/step - accuracy: 0.9849 - loss: 0.0554 - val_accuracy: 0.8655 - val_loss: 0.5016
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_mod
✅ Final model saved as MobileNetV2 Trained Model.h5
```

Graphs

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 476s 15s/step - accuracy: 0.8008 - loss: 0.5321
✅ Test Accuracy: 83.59%
✅ Test Loss: 0.4315
```





Confusion Matrix

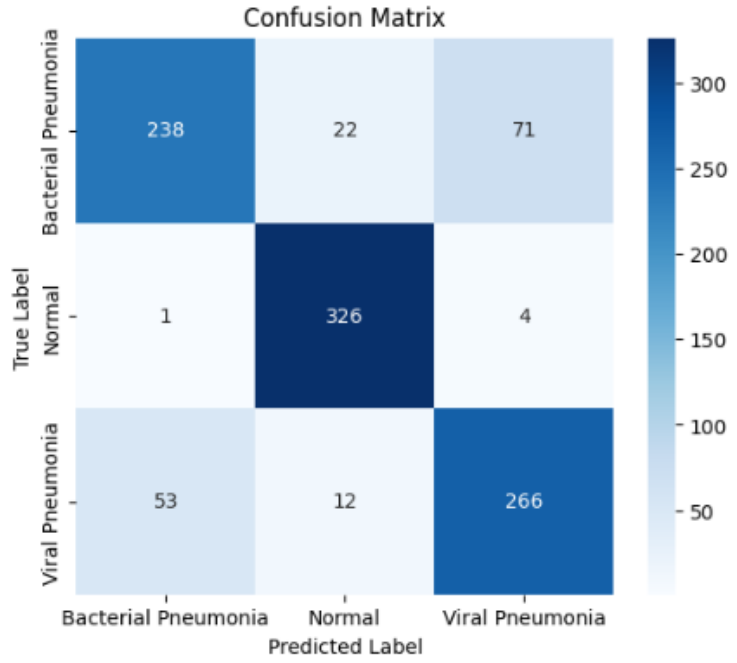
```
32/32 ━━━━━━━━━━━━━━━━━━━━ 19s 509ms/step

Classification Report:
                    precision    recall  f1-score   support

Bacterial Pneumonia       0.82      0.72      0.76       331
             Normal       0.91      0.98      0.94       331
    Viral Pneumonia       0.78      0.80      0.79       331

           accuracy                           0.84       993
          macro avg       0.83      0.84      0.83       993
       weighted avg       0.83      0.84      0.83       993

✅ Weighted F1 Score: 0.8331
✅ Weighted Recall: 0.8359
✅ Weighted Precision: 0.8336
```



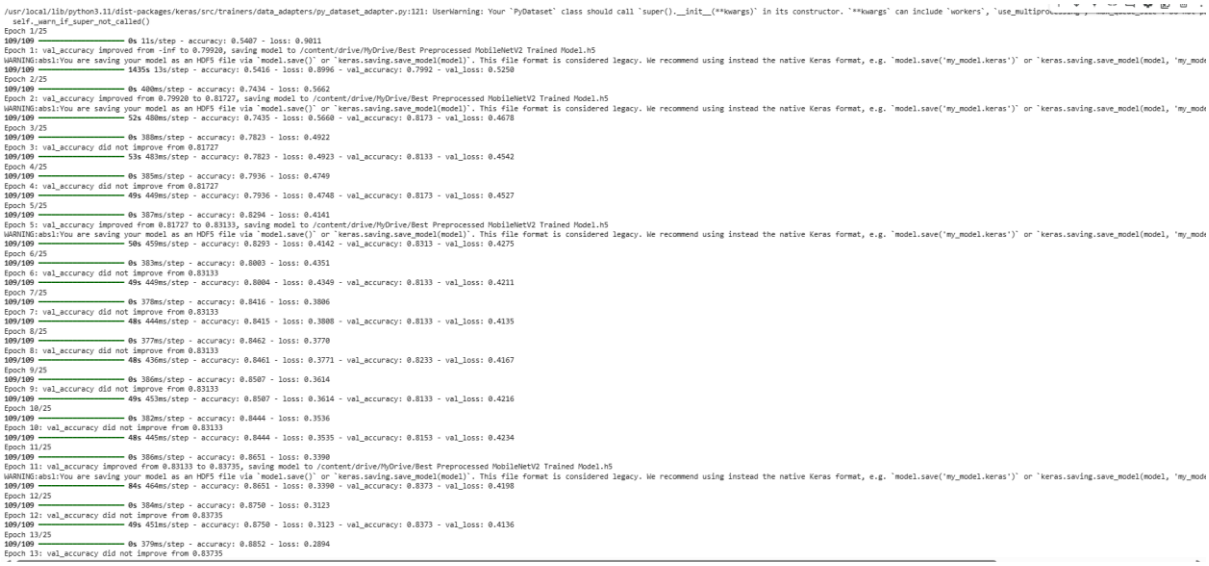
Results

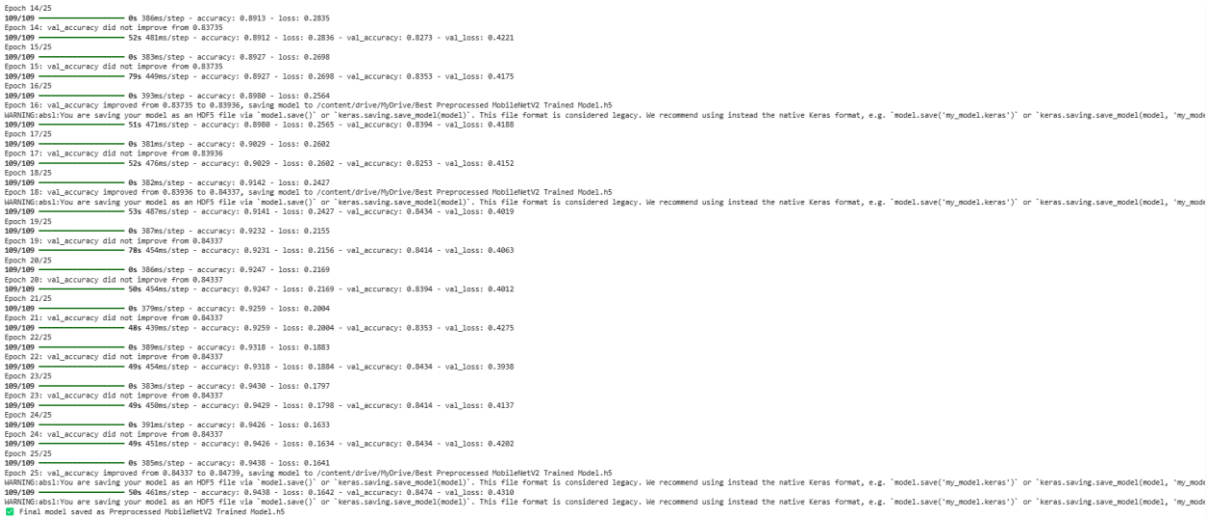
📊 Final Metrics Table:

	Dataset	Accuracy	Loss	F1 Score	Recall	Precision
0	Training	0.9431	0.1693	-	-	-
1	Validation	0.8474	0.4310	-	-	-
2	Test	0.8359	0.4315	0.833091	0.835851	0.833561

Preprocessed MobileNetV2 Model

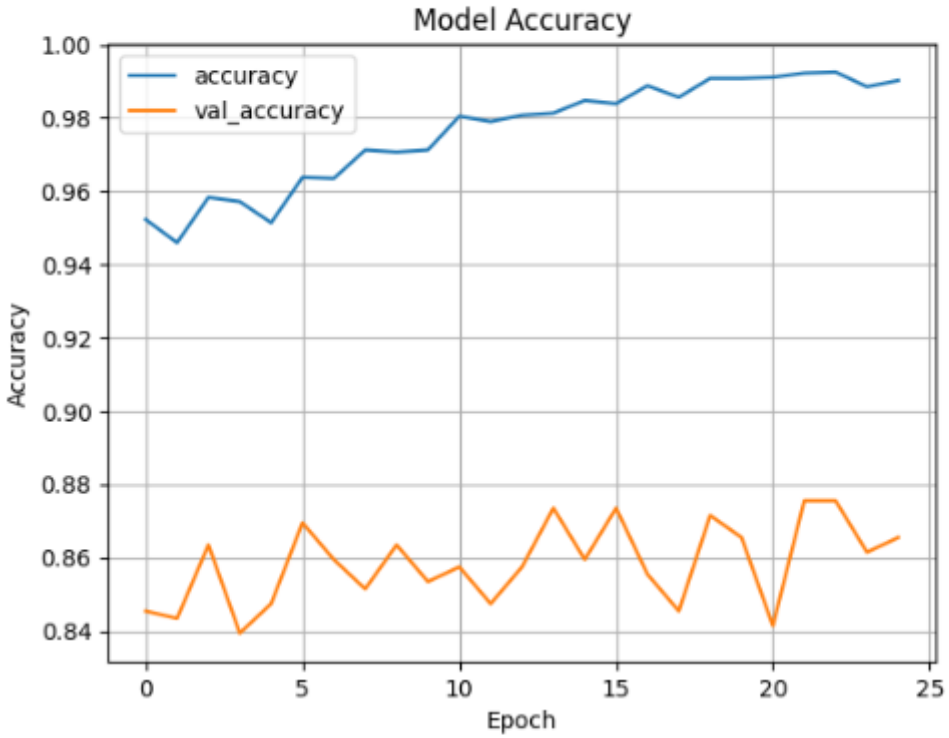
Epoch

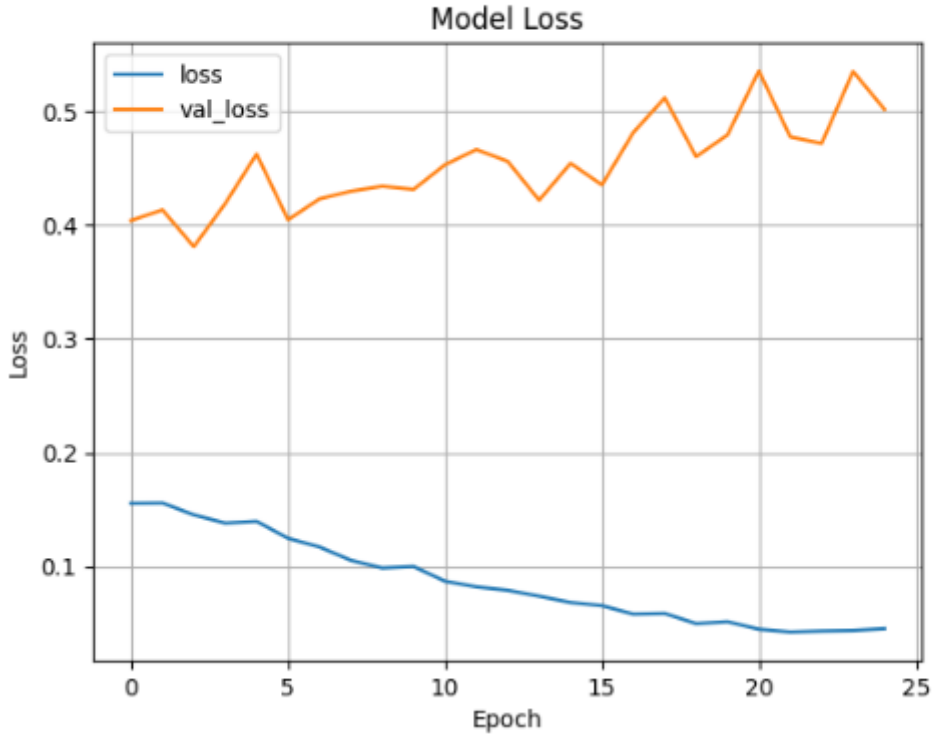




Graphs

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 12s 365ms/step - accuracy: 0.8282 - loss: 0.6048
✅ Test Accuracy: 84.79%
✅ Test Loss: 0.5891
```





Confusion Matrix

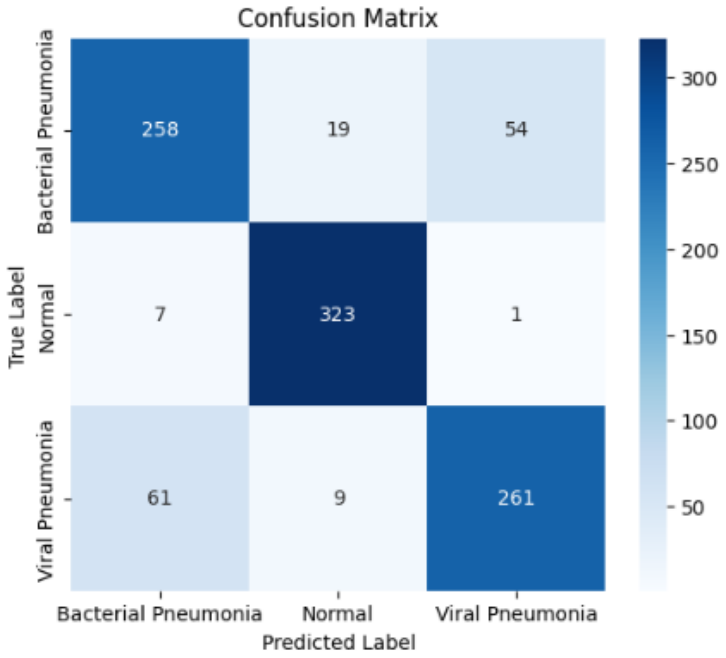
```
32/32 ━━━━━━━━━━━━━━━━━━━━ 16s 514ms/step

Classification Report:
                    precision    recall  f1-score   support

Bacterial Pneumonia       0.79      0.78      0.79       331
             Normal       0.92      0.98      0.95       331
    Viral Pneumonia       0.83      0.79      0.81       331

           accuracy                           0.85       993
          macro avg       0.85      0.85      0.85       993
       weighted avg       0.85      0.85      0.85       993

✅ Weighted F1 Score: 0.8465
✅ Weighted Recall: 0.8479
✅ Weighted Precision: 0.8459
```



Results

📊 Final Metrics Table:

	Dataset	Accuracy	Loss	F1 Score	Recall	Precision
0	Training	0.9902	0.0454	-	-	-
1	Validation	0.8655	0.5016	-	-	-
2	Test	0.8479	0.5891	0.846468	0.847936	0.845863