PNEUMONIA DETECTION AND CLASSIFICATION

A PROJECT REPORT

Submitted by

P A MOHAMMED ARSHAD
N SAI BALAJI

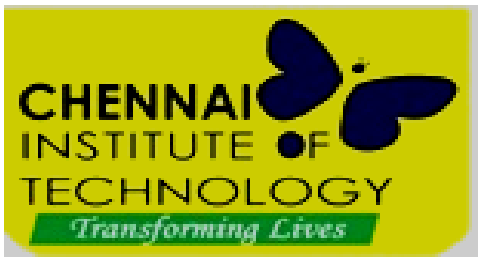
in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING



CHENNAI INSTITUTE OF TECHNOLOGY (AUTONOMOUS)
(Affiliated to Anna University, Chennai)
CHENNAI-600069

ANNA UNIVERSITY: CHENNAI-600 025

April -2023

ABSTRACT

Pneumonia is a life-threatening respiratory disease, which if not detected and treated timely, can lead to severe consequences. With the advancements in technology, computer-aided diagnosis (CAD) systems have become an important tool for diagnosing pneumonia. In recent years, deep learning models have shown promising results in the detection and classification of pneumonia from chest radiographs. In this study, we propose a deep learning-based CAD system for the detection and classification of pneumonia. Our system utilizes a convolutional neural network (CNN) architecture of MobileNet to automatically extract features from the chest radiographs and classify them into normal, bacterial, or viral pneumonia. The proposed system achieved an accuracy of 85% for detecting normal and pneumonia cases and classifying pneumonia cases into bacterial or viral types. The results of this study demonstrate the potential of deep learning-based CAD systems for accurate and efficient detection and classification of pneumonia.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Pneumonia Detection and Classification

Pneumonia is a lung infection that can be caused by bacteria, viruses, or fungi. It is a serious and potentially life-threatening illness, especially for people with weak immune systems or underlying medical conditions. Pneumonia can be difficult to diagnose accurately, and the treatment depends on the cause of the infection. Therefore, it is important to develop an effective pneumonia detection and classification system that can distinguish between different types of pneumonia accurately. The use of medical imaging, such as chest X-rays and CT scans, has been proven to be an effective method for diagnosing pneumonia. In recent years, there has been an increasing interest in developing machine learning algorithms that can assist radiologists in analyzing medical images and providing an accurate diagnosis.

In this project, we aim to develop a pneumonia detection and classification system using chest X-ray images. The system will be designed to distinguish between three classes: normal, viral pneumonia, and bacterial pneumonia. The proposed algorithm will extract features from the chest X-ray images and use a deep learning model to classify the images into the appropriate class.

The primary objective of this project is to improve the accuracy and efficiency of pneumonia diagnosis. The proposed system could potentially reduce the workload of radiologists and enable faster and more accurate diagnosis, leading to better patient outcomes. Additionally, the system could be used in areas where access to expert medical professionals is limited, providing a low-cost and effective alternative for pneumonia diagnosis.

1.2 Machine Learning

Machine learning (ML) is a subfield of artificial intelligence (AI) that focuses on the development of algorithms and models that enable computers to learn from data and make predictions or decisions without being explicitly programmed. In other words, ML is a way to teach computers to learn and improve their performance over time by analyzing patterns and relationships in large data sets. ML has a wide range of applications, including image and speech recognition, natural language processing, predictive analytics, and recommendation systems, among others.

1.3 Deep Learning

Deep learning (DL) is a subset of machine learning (ML) that uses artificial neural networks with multiple layers to learn and extract complex patterns and features from large data sets. DL algorithms are designed to mimic the structure and function of the human brain and are capable of performing tasks such as image and speech recognition, natural language processing, and autonomous driving. Unlike traditional machine learning algorithms, which require hand-engineered features, DL algorithms can automatically learn hierarchical representations of data by processing it through multiple layers of neural networks. This makes DL particularly well-suited for handling big data and solving complex problems that were previously impossible or impractical to solve with traditional machine learning techniques

1.4 Image processing

Image processing is a field of study that focuses on the analysis and manipulation of digital images to improve their quality or extract useful information. Machine learning and deep learning techniques have revolutionized the field of image processing, enabling automatic classification, object detection, and segmentation. Deep learning algorithms are particularly well-suited for image processing because they can recognize complex patterns and features in images without the need for explicit feature engineering. Image processing is a powerful tool used in various applications such as medical imaging, surveillance, autonomous vehicles, and virtual reality.

1.5 Pneumonia Detection and Classification using CNN

Pneumonia detection and classification using Convolutional Neural Networks (CNN) is a deep learning approach to automatically classify chest X-ray images into three categories: normal, viral pneumonia, and bacterial pneumonia. CNNs are a type of neural network that is specifically designed for image processing tasks.

To implement the pneumonia detection and classification system, the CNN is trained on a large dataset of labeled chest X-ray images. The CNN learns to automatically extract features from the images and use them to make accurate predictions about the presence of pneumonia and its type.

The trained model can then be used to classify new, unseen chest X-ray images into one of the three categories. The system can be used in a clinical setting to assist radiologists and clinicians in the diagnosis and treatment of pneumonia.

CNN-based pneumonia detection and classification systems have shown promising results in various studies, demonstrating high accuracy and reliability. Such systems have the potential to improve the efficiency and accuracy of pneumonia diagnosis and reduce the workload of healthcare professionals.

1.6 Existing system

The current system for pneumonia detection utilizes several CNN models that have shown excellent performance in detecting the presence of pneumonia in chest X-ray images. These models employ various convolutional layers to learn features from the input images, enabling them to classify X-ray images as either pneumonia-positive or pneumonia-negative with high accuracy. However, while the detection models work well for identifying the presence of pneumonia, they are not equipped with efficient classification models to accurately categorize the type of pneumonia as bacterial, viral, or normal. The lack of effective classification models can result in a suboptimal diagnostic process, potentially leading to misdiagnosis and inappropriate treatment. Therefore, developing a CNN-based classification model is crucial for improving the overall performance of the system and providing healthcare professionals with accurate and reliable diagnostic tools.

1.7 Proposed system

The proposed system for pneumonia detection and classification uses the MobileNet model, a lightweight CNN architecture designed for mobile devices with limited computational resources. To improve the system's accuracy and efficiency, the MobileNet model is modified by removing and adding layers to better fit the specific characteristics of chest X-ray images. The modified model

is then trained on a large dataset of labeled chest X-ray images, enabling it to learn to automatically extract relevant features and accurately classify pneumonia images as bacterial, viral, or normal. The proposed system's performance is evaluated based on its accuracy and efficiency. The proposed system demonstrates high performance in detecting and good accuracy in classifying pneumonia, it has the potential to improve the diagnostic process and assist healthcare professionals in making accurate and timely diagnoses.

1.8 Advantages of proposed systems

- Higher accuracy of Pneumonia detection and classification.
- Classify the image into Normal, Bacterial pneumonia, and viral pneumonia.
- Improved efficiency.
- Can be deployed in the web app and mobile app with fast access.

1.9 Objectives

- To enhance the given input image by Image acquisition and Image Preprocessing.
- Classify the image into Normal, Bacterial pneumonia, and viral pneumonia.
- Improved efficiency.
- deployed in the web app and mobile app with fast access.

1.10 Purpose

Pneumonia is a serious respiratory infection that can affect individuals of all ages, particularly the elderly, young children, and those with weakened immune systems. According to the World Health Organization (WHO), pneumonia is one of the leading causes of death among children under the age of 5, accounting for approximately 15% of all deaths in this age group. In addition, pneumonia is estimated to be responsible for 10% of all adult deaths worldwide, with the majority of these deaths occurring in developing countries. The exact number of people affected by pneumonia is difficult to determine due to variations in diagnostic and reporting methods, but it is believed to be a significant global health problem.

There are several reasons why classifying pneumonia using deep learning methods is crucial compared to manual methods:

- Accuracy: Deep learning methods have been shown to be highly accurate in identifying patterns and features in medical images, including chest X-ray images. This is particularly important in cases where the disease manifestation can be subtle, making it difficult for human experts to accurately classify.
- Efficiency: Deep learning models can process large amounts of data quickly and efficiently, allowing for a faster and more streamlined diagnostic
- Scalability: Deep learning methods can be easily scaled to handle large volumes of data, making it possible to analyze thousands of images in a short amount of time. This can be particularly useful in situations where there is a high demand for diagnostic services.

- Consistency: Deep learning models produce consistent and reproducible results, reducing the risk of errors and variability associated with manual classification methods. This can be particularly important in situations where there is a need for standardization and consistency in diagnoses.

Overall, deep learning methods offer several advantages over manual methods in classifying pneumonia, including higher accuracy, efficiency, scalability, and consistency. This makes them a valuable tool for improving the accuracy and effectiveness of the diagnostic process, ultimately leading to better patient outcomes.

CHAPTER 2
LITERATURE SURVEY

2.1 Introduction

A literature survey is the most essential phase in the software improvement process. Before building up the software it is important to decide the time factor, economy and complexity. When these things are fulfilled, at that point following stage is to figure out which working framework and operating system can be utilized for building up the software. When the programmer begins fabricating the device the developers require a parcel of outer help. This help can be acquired from books or from sites. Before building the software, the above thought is considered for building up the proposed software. The developer has to go through the existing software and should realize the advantages and disadvantages of the software and should apply this knowledge in implementing the proposed system.

2.2 A methodology for the determination of Pneumonia from Chest X-ray pictures utilizing Deep Learning

In 2019, Enes Ayan et al. proposed a methodology for the analysis of pneumonia from chest X-ray pictures utilizing profound learning procedures. In this methodology, they have utilized two notable convolution neural organization models Xception and Vgg16 for diagnosing pneumonia. They have utilized exchange learning and calibrating for their preparation stage. For the exchange learning strategy they have used the DenseNet-121 layer convolution neural organization, it is otherwise called CheXNet. They prepared their organization with 10.000 frontal view chest X-ray pictures with 14 unique infections and

accomplished 90% of precision. Further, the cycle has been partitioned into two organizations, their first organization depends on the Xception model and the subsequent one is Vgg16 based model. At that point, the two models are thought to be dependent on their exhibition and the outcome shows the Xception model outflanks the Vgg16 model in diagnosing pneumonia. The test aftereffects of their methodology in Vgg16 organization and Xception network are at exactness with 82%, and 87% separately.

2.3 An approach to detect Pneumonia clouds by using image processing methods.

In 2017, Sharma et al.[2] proposed an approach to detect Pneumonia clouds by using image processing methods. In which they proposed a method to diagnose the disease from medical images. They worked on 40 analog Chest X-ray which is partitioned into normal and pneumonia. They developed indigenous algorithms to crop and identify the boundary region of the lung from the chest x-rays. Initially, all 40 images used by them were resized to the optimal size for computational purposes. Further, histogram equalization was performed on the same to enhance the contrast of images. Later, indigenous algorithms were used to crop the abdomen area. Then, the Otsu image thresholding has been performed to detect the region of the lung which is not cloudy and the area is computed. Next, the ratio of this non-cloudy region to that of the extracted total area region of the lung would give information about the cloud formation part in the lung. Therefore, they reached an accuracy of around 76% to diagnose pneumonia.

2.4 A programmed identification of pneumonia investigating ultrasound computerized pictures

In 2016, Barrientos et al. proposed a programmed recognition of pneumonia dissecting ultrasound computerized pictures. This methodology depends on the investigation of examples present in rectangular fragments from ultrasound advanced pictures. They gathered an aggregate of 23 lung ultrasound pictures in which 15 found pneumonia and 8 sounds. Further, the trademark vectors were separated from outlines and arranged comparably for both positive and negative. Next, required areas were distinguished inside pictures. At long last, from the areas of interest highlights were removed and taken care of into the neural organization calculation utilized. This neural organization calculation utilizes sigmoid enactment work which has three layers i.e., input, covered up, and yield. Finally, to accomplish Pneumonia recognition, preparation and testing measures were performed. Along these lines, this paper reasons that the affectability and explicitness of the venture arrived at 91.52% and 100%.

2.5 Re-evaluating the commencement design for the PC vision

In 2015, Christain Szegedy et al[4] with reference to the paper Re-evaluating the commencement design for the PC vision. We find the authors focusing on the higher performance of convolution neural networks such as VGGNet and GoogleNet. It is seen that, though the Inception architecture of GoogleNet processes performs well even under strict memory constraints and computational budget when compared with VGGNet and AlexNet, which requires high computational costs, have concerns regarding its architectural complexity. This paper provides a few general principles and ideas of optimization, which help scale up convolution neural networks in efficient ways and proposes an InceptionV3 model, which relatively has a modest computation cost compared to

simpler and more monolithic architecture. A methodology for item recognition that will limit the calculation time was accomplished with high exactness. The methodology was familiar with developing a face identification framework which is around quicker than any past methodology. This paper unites new calculations, portrayals, and experiences that are very nonexclusive and have more extensive applications in PC vision and picture handling. At long last, this paper presents a lot of investigations on a troublesome face identification dataset that has been generally considered.

2.6 Evaluation of Lung Size in Patients with Pneumonia and Healthy Individuals

Mesut Togacar et al. utilized lung x-ray images that are accessible for the recognition system developed for the diagnosis of pneumonia. Evolutionary neural network (ENN), one of the deep learning models, was used for feature extraction in the resulting image set. The attributes obtained for the diagnosis of the disease were performed using performance comparisons using different classifiers. As a result of the comparison, a high success rate of 95.8% was obtained with the support vector machines (SVM) used in the classification process. It has been observed in the early diagnosis of deadly diseases such as pneumonia, in studies that deep learning models give faster and more accurate results. Chest x-ray (CXR) is important for the diagnosis of pneumonia. It is cheaper and easier to interpret than other imaging systems. In this study, it is aimed to remove the lung areas semi-automatically to diagnose pneumonia or any other lung disease by chest x-ray. In line with this goal, active contour method was applied to the lung x-ray images with bacterial and virus pneumonia in the Matlab environment. The area of healthy and diseased lung areas has been

calculated and the model is intended to operate fully automatically in future studies.

2.7 Proficient Pneumonia Detection in Chest X-ray Images Using Deep Transfer Learning

In 2020, Mohammad Farukh Hashmi, Satyarth Katiyar, A. Keskar, N.Bokde, Z. W. Geem utilized a model that is prepared for a specific tasis utilized as the beginning stage for addressing another assignment. In exchange learning, pre-prepared models are utilized as the starting point for some particular assignments, rather than experiencing the long cycle of preparing with haphazardly introduced loads. Henceforth, it assists with saving the substantial computer assets expected to create neural organization models to take care of these issues. Dish and Yang utilized area, task, and minor probabilities to propose a structure for better understanding the exchange learning.

The area D was characterized as a two-component tuple consisting of the element space, X with a minor likelihood, P(X), where X is an example information point.

Subsequently, numerically, area D can be characterized as, • Here, x is the space of all term vectors, and xi is the I th term vector. D = (x, P(X))

relating to certain archives, and X is a specific learning test (X = x1, xn, E x). Because of the absence of an adequate dataset, preparing a profound learning-based model for clinical finding-related issues is computationally costly, and the outcomes accomplished are additionally not sufficient.

CHAPTER 3

SYSTEM REQUIREMENTS SPECIFICATION

3.1 Hardware Requirements

- Processor: 2.5 gigahertz (GHz) frequency or above.
- RAM: A minimum of 4 GB of RAM.
- Hard disk: A minimum of 20 GB of available space.
- Input Device: High-resolution camera
- Monitor: Minimum Resolution 1024 X 768.

3.2 Software Requirements

- Operating System: Windows 7 and above.
- Programing language: Python 2.7 and above.
- Platform Supporting libraries: JetBrains PyCharm Community Edition 2018.3.5x64
- Supporting libraries: Tensorflow, OpenCV, PIL, tkinter, os, SKlearn, etc.

3.3 Functional requirements

Functional requirements describe the system functionality, while the non-functional requirements describe system properties and constraints. Functional requirements capture the intended behaviour of the system. This behaviour may be expressed as services, tasks, or the functions the system is required to perform. This lays out important concepts and discusses capturing functional requirements

in such a way they can drive architectural decisions and be used to validate the architecture.

Features may be additional functionality, or differ from basic functionality along some quality attribute. In the proposed system, concert assesses the compliance of a workflow by analysing the five established elements required to check for the rule adherence in workflows: activities, data, location, resources, and time limits.A rule describes which activities may, must or must not be performed on what objects by which roles. In addition, a rule can further prescribe the order of activities i.e. which activities have to happen before or after other activities.

3.4 Non-functional requirements:

3.4.1 Security

- System needs to control the user access and session
- It needs to store the data in a secure location and stored in a secure format
- It requires a secure communication channel for the data.

3.4.2 Concurrency and Capacity

- System should be able to handle multiple computations executing simultaneously, and potentially interacting with each other

3.4.3 Performance

- Performance is generally perceived as a time expectation. This is one of the most important considerations especially when the project is in the architecture phase

3.4.4 Reliability

- It is necessary to ensure and notify about the system transactions and processing as simple as keep a system log will increase the time and effort to get it done from the very beginning. Data should be transferred in a reliable way and using trustful protocols.

3.4.5 Maintainability

- Well-done system is meant to be up and running for a long time. Therefore, it will regularly need preventive and corrective maintenance. Maintenance might signify scalability to grow and improve the system features and functionalities.

3.4.6 Usability

- End user satisfaction and acceptance is one of the key pillars that support a project success. Considering the user experience requirements from the project conception is a win bet, and it will especially save a lot of time at the project release, as the user will not ask for changes or even worst misunderstandings.

3.4.7 Documentation

- All projects require a minimum of documentation at different levels. In many cases the users might even need training on it, so keeping good documentation practices and standards will do this task spread along the project development; but as well this must be establish since the project planning to include this task in the list.

CHAPTER 4

SYSTEM DESIGN

4.1 Data Flow Diagram

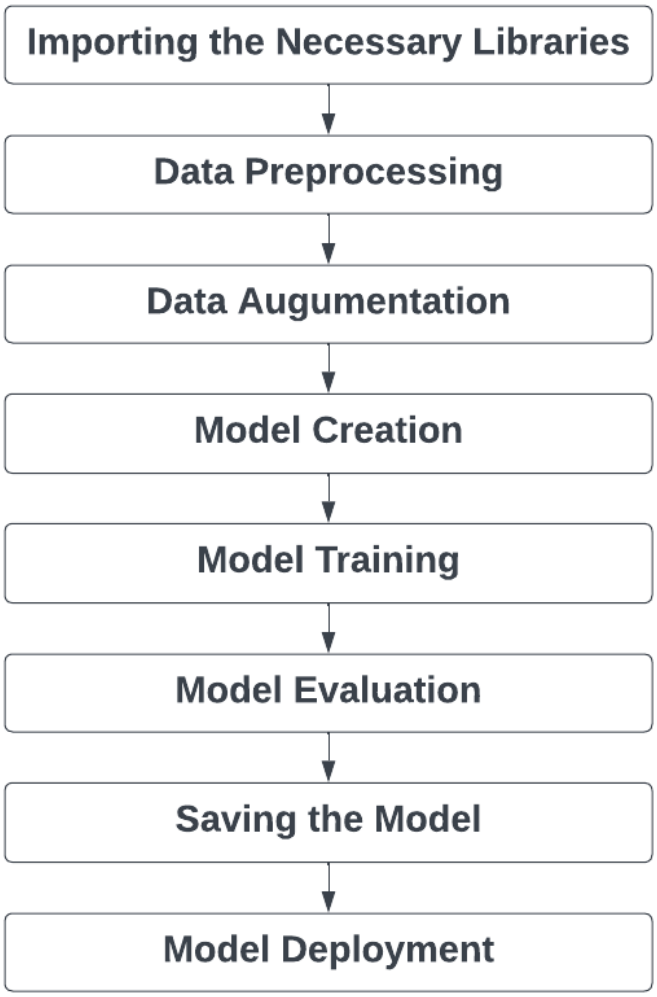


Fig 4.1 Data Flow Diagram

4.2 Importing the Necessary Libraries

- Importing the necessary libraries is the first step in setting up the project environment and preparing it for data exploration, model building, and

evaluation.In this specific project, the libraries that are imported are all essential tools for machine learning and deep learning tasks.

The major libraries used in the project are :

Tensorflow

- TensorFlow library is an open-source, web-based machine learning framework that is widely used for building and deploying machine learning models in web browsers and other JavaScript environments. It provides a set of APIs for creating, training, and running machine learning models using JavaScript.

Keras

- Keras is a high-level neural networks API that simplifies the process of building and training deep learning models. It allows developers to easily build neural networks by providing a simple and user-friendly interface for building complex models.

NumPy

- NumPy is a powerful library for scientific computing in Python. It provides support for array operations, mathematical functions, linear algebra, and many other useful tools for data manipulation and analysis.

Matplotlib

- Matplotlib and Seaborn are data visualization libraries used to create visual representations of data. They provide a wide range of functions and tools

for creating various types of charts, graphs, and plots, which are useful for data exploration and analysis.

Pandas

- Pandas is a Python library used for data manipulation and analysis. It provides a set of tools for reading, cleaning, and manipulating data, which are essential for preparing data for machine learning models.

sklearn

- Scikit-learn, also known as sklearn, is a popular open-source machine learning library for Python. It provides a wide range of tools for data preprocessing, feature extraction, model selection, and evaluation. Scikit-learn is built on top of other scientific computing libraries like NumPy, SciPy, and Matplotlib, which makes it a powerful and flexible library for machine learning tasks.

4.3 Data Preprocessing

- Data preprocessing is an important step in machine learning projects, as it involves preparing the data before feeding it into the machine learning model. In the project, the data preprocessing step involves loading the data for training and validation from the specified directories, and then preprocessing it by resizing the images to a fixed size, normalizing pixel values, and splitting it into training and validation set.

- The data used in this project consists of chest X-ray images that are labeled as either "normal" or "pneumonia". The images are loaded using the Python

Image Library (Pillow) and then resized to a fixed size of 224x224 pixels using the scikit-image library. This step is necessary to ensure that all the images have the same size and format, which is required for feeding them into a machine learning model.

- After resizing the images, the next step is to normalize the pixel values. This is done using the scikit-learn StandardScaler function, which standardizes the pixel values by subtracting the mean and dividing by the standard deviation. This step is important to ensure that the machine learning model is not biased towards any particular pixel values and can learn from the data in a more accurate and consistent manner.

4.4 Data Augmentation

- Data augmentation is a common technique used in machine learning to artificially increase the size of the training dataset and improve the performance of the machine learning model. In this project data augmentation is performed on the training data to avoid overfitting and improve the generalization performance of the machine learning model.

- The data augmentation step involves applying various transformations to the images in the training set. These transformations include random rotation, horizontal and vertical flipping, and zooming. By applying these transformations to the images, we can create new, slightly different versions of the original images, which can help the machine learning model to learn more robust and generalizable features.

4.5 Model Creation

- In this step, the code creates a Convolutional Neural Network (CNN) model using TensorFlow with MobileNet architecture, which includes several layers, such as convolutional layers, pooling layers, and fully connected layers. The MobileNet architecture is optimized for mobile and embedded devices and is known for its speed and efficiency while still maintaining high accuracy in image classification tasks.

- To improve the accuracy of the MobileNet model, the project makes several modifications to the original architecture. One modification involves freezing the first few layers of the MobileNet model and training only the remaining layers, which are fine-tuned to the specific task of diagnosing pneumonia. This technique, called transfer learning, allows the model to leverage the knowledge gained from training on large datasets and apply it to a smaller dataset.

4.6 Model Training

- The Model Training step involves training a convolutional neural network (CNN) model using the MobileNet architecture on preprocessed and augmented data. Stochastic gradient descent (SGD) is utilized as the optimization technique, with categorical cross-entropy serving as the loss function to measure the difference between the predicted and actual labels. Performance metrics such as accuracy, precision, recall, and F1 score are monitored during training to evaluate model performance and prevent overfitting. The training process is carried out in epochs, with the number of epochs set to 20 in this project. The trained model's weights are saved to

a file for later use in making predictions on new data, such as images from a web application.

4.7 Model Evaluation

- Model evaluation involves assessing the performance of the trained convolutional neural network (CNN) model by measuring its accuracy and loss on the validation data. Accuracy measures the proportion of correctly classified images to the total number of images, while loss measures the model's error in prediction. The evaluation is performed using the validation set, which contains images not seen during training to assess the model's ability to generalize to new data. The evaluation metric results provide insight into the model's performance and help identify potential issues such as underfitting or overfitting. The goal is to achieve high accuracy and low loss values to ensure the model's reliability in making predictions on new data.

4.8 Saving the Model

- After the training process is complete, the code saves the trained convolutional neural network (CNN) model to disk using the HDF5 (.h5) format. This format is used to store Keras models in HDF5 format, which can be loaded in Python using Keras or TensorFlow libraries. The trained model is saved as an .h5 file, which can be loaded and used in a web application using TensorFlow.js, a JavaScript library for machine learning. By saving the trained model, it can be used to classify new images and make predictions without having to retrain the model from scratch. This approach is especially useful when working with large datasets and

computationally intensive models. The saved model can also be shared and used by others for similar tasks, thereby contributing to the advancement of machine learning applications.

4.9 Deploying the model using Streamlit

- After the model has been trained and saved, it needs to be deployed in a web application for use. In this project, the model is deployed in a Streamlit web application, which is a Python library used to build interactive web applications. The application takes an input image and classifies it as normal, viral, or bacterial based on the prediction of the trained model.

- To achieve this, the input image is first preprocessed by resizing it to the same dimensions as the images used during training, and then normalized. Next, the saved model is loaded using the TensorFlow.js library, which allows the model to be loaded and used in the web application. The input image is then passed through the loaded model, and the model makes a prediction on the image, classifying it as normal, viral, or bacterial.

- The predicted class is then displayed on the web application along with the input image. The user can input new images and obtain the classification in real-time. The deployment of the model in a web application allows for easy access and use by healthcare professionals, contributing to the early detection and treatment of pneumonia.

CHAPTER 5

SYSTEM IMPLEMENTATION

5.1 System Implementation Diagram

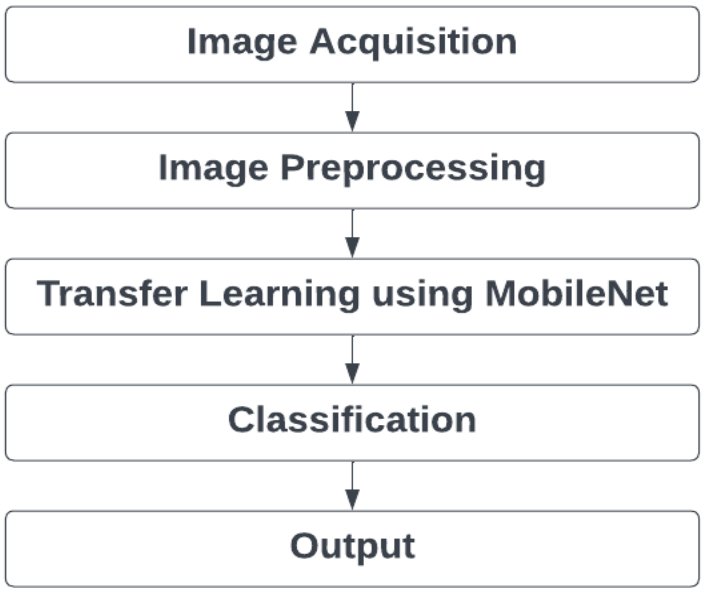


Fig 5.1 System Implementation

5.2 Image Acquisition

- In the context of computer vision, image acquisition refers to the process of obtaining digital images from a physical source, such as a camera or scanner. In the project, the images are obtained from a directory on the local system.
- During image acquisition, it is important to consider factors such as image resolution, color space, and image format. In this case, the images are in JPEG format and are of varying sizes. It is also important to ensure that the images are representative of the real-world data that the model is expected to classify.

- Once the images have been acquired, they can be passed through a series of preprocessing steps, such as resizing, normalization, and augmentation, before being fed into the model for training or inference.

5.3 Image Preprocessing

- In the project, before feeding the input images into the classification model, the code preprocesses them using image resizing and normalization. The images are resized to a fixed size to ensure that they have the same dimensions as the input layer of the model. Additionally, pixel values are normalized to ensure that they fall within a certain range. This helps to reduce the impact of variations in lighting and color on the classification performance of the model. The preprocessing step is critical for ensuring that the input data is in the appropriate format for the model and can yield better classification results.

5.4 Transfer learning using MobileNet

- Transfer learning using MobileNet is a technique where a pre-trained deep learning model, in this case, MobileNet, is used as a starting point to solve a new problem. It involves using a pre-trained model that has already learned a large number of features from a vast dataset, and then fine-tuning the model's parameters for a new dataset to achieve better accuracy. By using transfer learning, we can reduce the time, effort, and resources required to train a new model from scratch.

- In this project, the MobileNet model is used to extract features from the preprocessed input image. MobileNet is a popular deep learning model architecture that is specifically designed for mobile and embedded devices, and has achieved state-of-the-art results in many computer vision tasks. By using MobileNet, we can leverage the pre-trained weights of the model to extract important features from the input image, which can then be used to classify the image as normal, viral, or bacterial pneumonia.

- After extracting the features from the input image using MobileNet, a custom classifier is added on top of the model to perform the final classification. This classifier takes the extracted features as input and uses them to classify the image as one of the three categories. By using MobileNet for feature extraction and a custom classifier for final classification, we can achieve high accuracy with minimal training time and computational resources.

5.5 Classification

- In the project we are using a convolutional neural network (CNN) to classify chest X-ray images as either normal or showing signs of pneumonia. CNNs are chosen as a classification tool due to their well-known success in real-world applications. CNN architectures vary depending on the problem at hand, and in our proposed model, there are three convolutional layers followed by a max-pooling layer, and a fully connected MLP layer. ReLu activation function is applied to the output of every convolutional layer and fully connected layer.

- Our project is using a pre-trained MobileNet model for image classification. MobileNet is a popular convolutional neural network architecture designed for mobile and embedded vision applications, where computational resources are limited. It achieves this by using depthwise separable convolutions, which reduces the number of parameters and operations needed to process an image, while still achieving high accuracy.

- In our model, we are using a pre-trained MobileNet model that has been trained on the ImageNet dataset, which contains millions of images. Have added a global average pooling layer after the last convolutional layer, followed by a dense layer with 128 neurons and a ReLU activation function. The output of this layer is then passed to the final dense layer with 2 neurons, one for each class, and a softmax activation function to produce a probability distribution over the output classes.

- Then we are fine-tuning the last few layers of the MobileNet model by training them on the dataset, which consists of chest X-ray images with and without pneumonia. Fine-tuning is a common technique used in transfer learning, where the weights of the pre-trained model are used as a starting point, and then further trained on a new dataset to improve the model's accuracy on a specific task.

Following are some common symptoms of bacterial Pneumonia and viral pneumonia.

5.5.1 Bacterial Pneumonia

Bacterial pneumonia is caused by bacteria such as Streptococcus pneumoniae, Legionella pneumophila, and Mycoplasma pneumonia. The symptoms of bacterial pneumonia usually start suddenly and can be severe. They include high fever, chills, cough with yellow, green, or bloody mucus, shortness of breath, chest pain, and rapid heartbeat. Other symptoms may include headache, fatigue, sweating, loss of appetite, and muscle aches. The severity of symptoms can vary depending on the age and overall health of the affected person, as well as the type of bacteria causing the infection. Bacterial pneumonia can be a serious and potentially life-threatening condition, especially for young children, older adults, and people with weakened immune systems.

5.5.2 Viral pneumonia

Viral pneumonia is caused by viruses such as influenza, respiratory syncytial virus (RSV), and SARS-CoV-2 (the virus that causes COVID-19). The symptoms of viral pneumonia can be similar to those of bacterial pneumonia, but they may develop more slowly and can be milder. They include cough, fever, shortness of breath, chest pain, and fatigue. Other symptoms may include headache, muscle aches, and sore throat. In some cases, people with viral pneumonia may not have any symptoms at all. Like bacterial pneumonia, the severity of symptoms can vary depending on the age and overall health of the affected person, as well as the type of virus causing the infection. While viral pneumonia is usually less severe than bacterial pneumonia, it can still be a serious condition, especially for young children, older adults, and people with weakened immune systems.

5.5.3 Convolutional Neural Network (ConvNet) Algorithm

1) A convolutional neural network is a class of deep neural network, most commonly applied to analysing visual imagery.
2) A Convolutional Neural Network is a Machine Learning algorithm that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other.

5.5.4 Input Layer

In the figure below, we have an RGB image which has been separated by its three color planes—Red, Green, and Blue. There are a number of such color spaces in which images exist—Grayscale, RGB, HSV, CMYK, etc.

• The role of the ConvNet is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction.

This is important when we are to design an architecture which is not only good at learning features but also is scalable to massive datase

5.5.5 Convolution Layer--The Kernel

In the below demonstration, the green section resembles our 5x5x1 input image. The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the Kernel/Filter, K, represented in the color yellow. We have selected K as a 3x3x1 matrix{{1,0,1},{0,1,0},{1,0,1}}.

• The Kernel shifts 9 times because of Stride Length = 1, every time performing a matrix multiplication operation between K and the portion P of the image

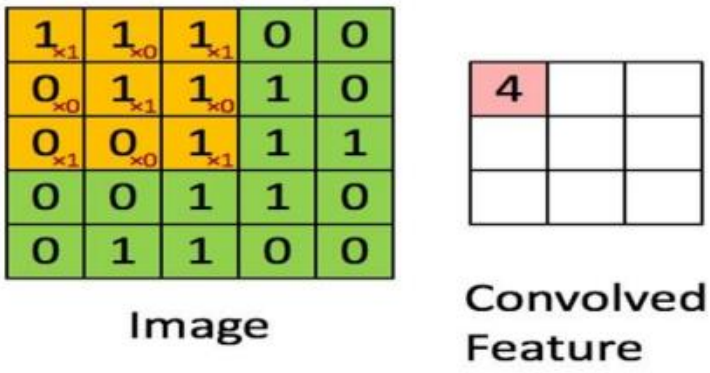


Fig 5.2 convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature

The filter moves to the right with a certain Stride Value till it parses the complete width. Moving on, it hops down to the beginning (left) of the image with the same Stride Value and repeats the process until the entire image is traversed.

5.5.6 Pooling Layer

• The Pooling layer is responsible for reducing the spatial size of the Convolved Feature.

• It is even useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model.

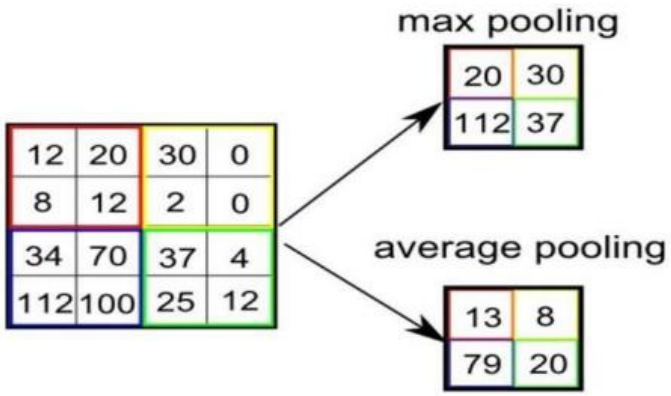


Fig 5.3 Selecting maximum and average pooling matrix

There are two types of Pooling: Max Pooling and Average Pooling.

• Max Pooling returns the maximum value from the portion of the image covered by the Kernel.
• On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the Kernel.

5.5.7 MobileNet

MobileNet is a popular convolutional neural network architecture designed for mobile and embedded vision applications. It was introduced by Google in 2017

and has since been widely used due to its computational efficiency and high accuracy.

The MobileNet architecture is based on a series of depthwise separable convolution layers, which are used to reduce the computational cost while maintaining a high degree of accuracy.

In traditional convolutional neural networks, each layer consists of a convolutional layer followed by a pooling layer. This approach is computationally expensive because the convolutional layer requires a large number of parameters to be learned, and the pooling layer discards a large portion of the information.

In contrast, the MobileNet architecture uses depthwise separable convolutions, which consist of two separate layers: a depthwise convolution layer followed by a pointwise convolution layer. The depthwise convolution layer applies a single filter to each input channel, while the pointwise convolution layer combines the outputs of the depthwise convolution layer using a 1x1 convolution.

This approach significantly reduces the number of parameters and computations required while still producing high-quality results. In addition, MobileNet uses techniques such as batch normalization and ReLU activation to further improve the accuracy of the model.

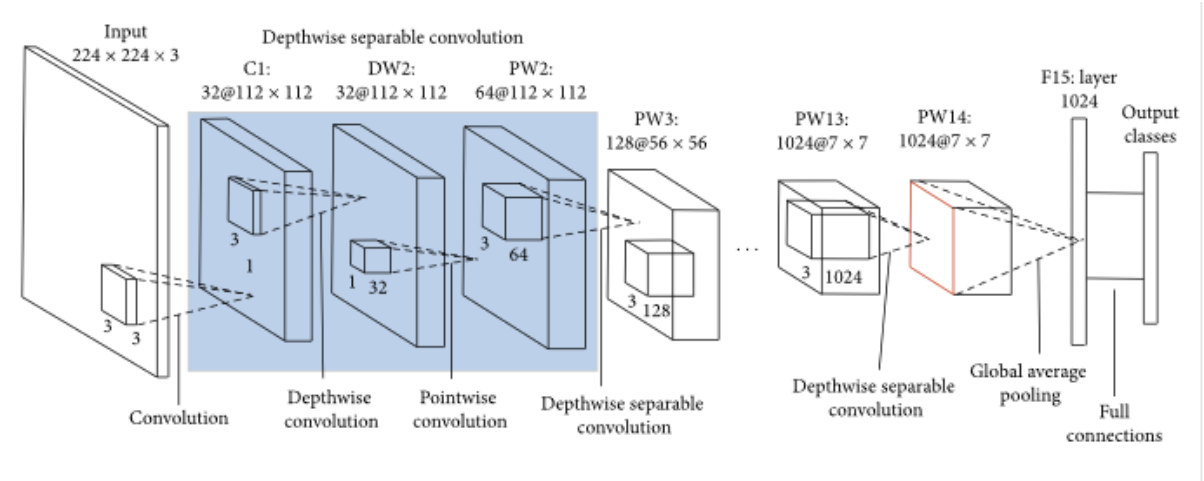


Fig 5.4 : Architecture of MobileNet

5.6 Output

A pneumonia detection and classification model has been developed and deployed using Streamlit, a popular open-source framework for building data visualization tools. With this model, users can easily upload an image of a chest X-ray and receive an output of the probability score of bacterial, viral, and normal pneumonia detected in the image. This model leverages the power of deep learning algorithms to analyze the chest X-ray images and accurately classify the type of pneumonia present in the image. By making this model accessible through Streamlit, healthcare professionals and researchers can easily and quickly diagnose pneumonia cases, which can ultimately help in providing timely treatment to patients.

CHAPTER 6

SOURCE CODES

6.1 Code for Model Building and Predictions

Importing the Necessary Libraries and Packages

```
from numpy.random import seed
seed(101)
import tensorflow as tf
tf.random.set_seed(101)

import pandas as pd
import numpy as np
import keras
from keras import backend as K
from keras.layers.core import Dense, Dropout
from keras.optimizers import Adam
from keras.metrics import categorical_crossentropy
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Model
from keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint
from tensorflow.keras.metrics import categorical_accuracy, top_k_categorical_accuracy

import os
from sklearn.utils import shuffle
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
import itertools
import shutil
import matplotlib.pyplot as plt
%matplotlib inline
```

Check for the count of files in each folder

```
# Train
# normal
print(len(os.listdir('Data/chest_xray/train/NORMAL')))
# pneumonia
print(len(os.listdir('Data/chest_xray/train/PNEUMONIA')))

# Val
# normal
print(len(os.listdir('Data/chest_xray/val/NORMAL')))
# pneumonia
print(len(os.listdir('Data/chest_xray/val/PNEUMONIA')))


# Test
# normal
print(len(os.listdir('Data/chest_xray/test/NORMAL')))
# pneumonia
print(len(os.listdir('Data/chest_xray/test/PNEUMONIA')))
```

Create a Dataframe containing all images

```
# create a list of files in each folder
train_normal_list = os.listdir('Data/chest_xray/train/NORMAL')
train_pneu_list = os.listdir('Data/chest_xray/train/PNEUMONIA')
val_normal_list = os.listdir('Data/chest_xray/val/NORMAL')
val_pneu_list = os.listdir('Data/chest_xray/val/PNEUMONIA')
test_normal_list = os.listdir('Data/chest_xray/test/NORMAL')
test_pneu_list = os.listdir('Data/chest_xray/test/PNEUMONIA')

# initialize the function
```

```
def assign_pneu_type(x):
    x = str(x)
    if 'virus' in x:
        return 'viral'
    if 'bacteria' in x:
        return 'bacterial'

# TRAIN_NORMAL
# create the dataframe
df_train_normal = pd.DataFrame(train_normal_list, columns=['image_id'])
# delete the entry named .DS_Store
df_train_normal = df_train_normal[df_train_normal['image_id'] != '.DS_Store']
# create a new target column
df_train_normal['target'] = 'normal'

# TRAIN_PNEU
# create the dataframe
df_train_pneu = pd.DataFrame(train_pneu_list, columns=['image_id'])
# delete the entry named .DS_Store
df_train_pneu = df_train_pneu[df_train_pneu['image_id'] != '.DS_Store']
# create a target column that's a copy of the image column
df_train_pneu['target'] = df_train_pneu['image_id']
# apply the function to this target column
df_train_pneu['target'] = df_train_pneu['target'].apply(assign_pneu_type)

# VAL_NORMAL
# create the dataframe
df_val_normal = pd.DataFrame(val_normal_list, columns=['image_id'])
# delete the entry named .DS_Store
df_val_normal = df_val_normal[df_val_normal['image_id'] != '.DS_Store']
# create a new target column
df_val_normal['target'] = 'normal'



# VAL_PNEU
```

```
# create the dataframe
df_val_pneu = pd.DataFrame(val_pneu_list, columns=['image_id'])
# delete the entry named .DS_Store
df_val_pneu = df_val_pneu[df_val_pneu['image_id'] != '.DS_Store']
# create a target column that's a copy of the image column
df_val_pneu['target'] = df_val_pneu['image_id']
# apply the function to this target column
df_val_pneu['target'] = df_val_pneu['target'].apply(assign_pneu_type)


# TEST_NORMAL
# create the dataframe
df_test_normal = pd.DataFrame(test_normal_list, columns=['image_id'])
# delete the entry named .DS_Store
df_test_normal = df_test_normal[df_test_normal['image_id'] != '.DS_Store']
# create a new target column
df_test_normal['target'] = 'normal'


# TEST_PNEU
# create the dataframe
df_test_pneu = pd.DataFrame(test_pneu_list, columns=['image_id'])
# delete the entry named .DS_Store
df_test_pneu = df_test_pneu[df_test_pneu['image_id'] != '.DS_Store']
# create a target column that's a copy of the image column
df_test_pneu['target'] = df_test_pneu['image_id']
# apply the function to this target column
df_test_pneu['target'] = df_test_pneu['target'].apply(assign_pneu_type)
```

Concatenate the dataframes

```
df_data = \
pd.concat([df_train_normal,df_train_pneu,df_val_normal,df_val_pneu,df_test_n
ormal,
        df_test_pneu],axis=0).reset_index(drop=True)
```

```
# shuffle
df_data = shuffle(df_data)
# Check the target distribution
df_data['target'].value_counts()

#Print the Dataframe
df_data.head()
```

Create the directory structure

```
# The original dataset directory structure was set up to support a binary classification problem. Because we will be predicting 3 classes we need to change this structure. In these folders we will store the images that will later be fed to the Keras generators.

# Create a new directory
base_dir = 'base_dir'
os.mkdir(base_dir)


#[CREATE FOLDERS INSIDE THE BASE DIRECTORY]

# now we create 3 folders inside 'base_dir':

# train
   # normal
   # bacterial
   # viral

# val
   # normal
   # bacterial
   # viral
```

```
# create a path to 'base_dir' to which we will join the names of the new folders
# train_dir
train_dir = os.path.join(base_dir, 'train_dir')
os.mkdir(train_dir)

# val_dir
val_dir = os.path.join(base_dir, 'val_dir')
os.mkdir(val_dir)


# [CREATE FOLDERS INSIDE THE TRAIN AND VALIDATION FOLDERS]
# Inside each folder we create seperate folders for each class

# create new folders inside train_dir
normal = os.path.join(train_dir, 'normal')
os.mkdir(normal)
bacterial = os.path.join(train_dir, 'bacterial')
os.mkdir(bacterial)
viral = os.path.join(train_dir, 'viral')
os.mkdir(viral)


# create new folders inside val_dir
normal = os.path.join(val_dir, 'normal')
os.mkdir(normal)
bacterial = os.path.join(val_dir, 'bacterial')
os.mkdir(bacterial)
viral = os.path.join(val_dir, 'viral')
os.mkdir(viral)

os.listdir('base_dir/train_dir')
# Create Train and Val Sets

y = df_data['target']
```

```
df_train, df_val = train_test_split(df_data, test_size=0.15, random_state=101, stratify=y)

print(df_train.shape)
print(df_val.shape)


# check df_train class distribution
df_train['target'].value_counts()


# check df_val class distribution
df_val['target'].value_counts()
```

Transfer the Images into the Folder

```
# Set the image_id as the index in df_data
df_data.set_index('image_id', inplace=True)

# Get a list of images in each of the folders
train_normal_list = os.listdir('Data/chest_xray/train/NORMAL')
train_pneu_list = os.listdir('Data/chest_xray/train/PNEUMONIA')
val_normal_list = os.listdir('Data/chest_xray/val/NORMAL')
val_pneu_list = os.listdir('Data/chest_xray/val/PNEUMONIA')
test_normal_list = os.listdir('Data/chest_xray/test/NORMAL')
test_pneu_list = os.listdir('Data/chest_xray/test/PNEUMONIA')

# Get a list of train and val images
train_list = list(df_train['image_id'])
val_list = list(df_val['image_id'])

# Transfer the train images

for image in train_list:
```

```
fname = image
label = df_data.loc[image,'target']

if fname in train_normal_list:
    # source path to image
    src = os.path.join('Data/chest_xray/train/NORMAL', fname)
    # destination path to image
    dst = os.path.join(train_dir, label, fname)
    # copy the image from the source to the destination
    shutil.copyfile(src, dst)

if fname in train_pneu_list:
    # source path to image
    src = os.path.join('Data/chest_xray/train/PNEUMONIA', fname)
    # destination path to image
    dst = os.path.join(train_dir, label, fname)
    # copy the image from the source to the destination
    shutil.copyfile(src, dst)

if fname in val_normal_list:
    # source path to image
    src = os.path.join('Data/chest_xray/val/NORMAL', fname)
    # destination path to image
    dst = os.path.join(train_dir, label, fname)
    # copy the image from the source to the destination
    shutil.copyfile(src, dst)

if fname in val_pneu_list:
    # source path to image
    src = os.path.join('Data/chest_xray/val/PNEUMONIA', fname)
    # destination path to image
    dst = os.path.join(train_dir, label, fname)
    # copy the image from the source to the destination
    shutil.copyfile(src, dst)
```

```
    if fname in test_normal_list:
        # source path to image
        src = os.path.join('Data/chest_xray/test/NORMAL', fname)
        # destination path to image
        dst = os.path.join(train_dir, label, fname)
        # copy the image from the source to the destination
        shutil.copyfile(src, dst)

    if fname in test_pneu_list:
        # source path to image
        src = os.path.join('Data/chest_xray/test/PNEUMONIA', fname)
        # destination path to image
        dst = os.path.join(train_dir, label, fname)
        # copy the image from the source to the destination
        shutil.copyfile(src, dst)




# Transfer the val images

for image in val_list:

    fname = image
    label = df_data.loc[image,'target']

    if fname in train_normal_list:
        # source path to image
        src = os.path.join('Data/chest_xray/train/NORMAL', fname)
        # destination path to image
        dst = os.path.join(val_dir, label, fname)
        # copy the image from the source to the destination
        shutil.copyfile(src, dst)
```

```
if fname in train_pneu_list:
    # source path to image
    src = os.path.join('Data/chest_xray/train/PNEUMONIA', fname)
    # destination path to image
    dst = os.path.join(val_dir, label, fname)
    # copy the image from the source to the destination
    shutil.copyfile(src, dst)

if fname in val_normal_list:
    # source path to image
    src = os.path.join('Data/chest_xray/val/NORMAL', fname)
    # destination path to image
    dst = os.path.join(val_dir, label, fname)
    # copy the image from the source to the destination
    shutil.copyfile(src, dst)

if fname in val_pneu_list:
    # source path to image
    src = os.path.join('Data/chest_xray/val/PNEUMONIA', fname)
    # destination path to image
    dst = os.path.join(val_dir, label, fname)
    # copy the image from the source to the destination
    shutil.copyfile(src, dst)

if fname in test_normal_list:
    # source path to image
    src = os.path.join('Data/chest_xray/test/NORMAL', fname)
    # destination path to image
    dst = os.path.join(val_dir, label, fname)
    # copy the image from the source to the destination
    shutil.copyfile(src, dst)

if fname in test_pneu_list:
    # source path to image
    src = os.path.join('Data/chest_xray/test/PNEUMONIA', fname)
```

```
    # destination path to image
    dst = os.path.join(val_dir, label, fname)
    # copy the image from the source to the destination
    shutil.copyfile(src, dst)
```

Check how many train images we have in each folder

```
print(len(os.listdir('base_dir/train_dir/normal')))
print(len(os.listdir('base_dir/train_dir/bacterial')))
print(len(os.listdir('base_dir/train_dir/viral')))
```

Check how many validation images we have in each folder

```
print(len(os.listdir('base_dir/val_dir/normal')))
print(len(os.listdir('base_dir/val_dir/bacterial')))
print(len(os.listdir('base_dir/val_dir/viral')))
```

Copy the train images into aug_dir

```
class_list = ['normal','bacterial','viral']

for item in class_list:

# We are creating temporary directories here because we delete these directories later
   # create a base dir
   aug_dir = 'aug_dir'
   os.mkdir(aug_dir)
   # create a dir within the base dir to store images of the same class
   img_dir = os.path.join(aug_dir, 'img_dir')
   os.mkdir(img_dir)

   # Choose a class
   img_class = item
```

```
# list all images in that directory
img_list = os.listdir('base_dir/train_dir/' + img_class)

# Copy images from the class train dir to the img_dir e.g. class 'bacterial'
for fname in img_list:
        # source path to image
        src = os.path.join('base_dir/train_dir/' + img_class, fname)
        # destination path to image
        dst = os.path.join(img_dir, fname)
        # copy the image from the source to the destination
        shutil.copyfile(src, dst)


# point to a dir containing the images and not to the images themselves
path = aug_dir
save_path = 'base_dir/train_dir/' + img_class

# Create a data generator
datagen = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.05,
    height_shift_range=0.05,
    zoom_range=0.05,
    horizontal_flip=True,
    fill_mode='nearest')

batch_size = 50

aug_datagen = datagen.flow_from_directory(path,
                                save_to_dir=save_path,
                                save_format='jpg',
                                        target_size=(224,224),
                                        batch_size=batch_size)
```

```
    # Generate the augmented images and add them to the training folders

    ############

    num_aug_images_wanted = 5000 # total number of images we want to have in
each class

    ############

    num_files = len(os.listdir(img_dir))
    num_batches = int(np.ceil((num_aug_images_wanted-num_files)/batch_size))

    # run the generator and create augmented images
    for i in range(0,num_batches):

        imgs, labels = next(aug_datagen)

    # delete temporary directory with the raw image files
    shutil.rmtree('aug_dir')
```

Check how many train images we now have in each folder after Data Augumentation

```
print(len(os.listdir('base_dir/train_dir/normal')))
print(len(os.listdir('base_dir/train_dir/bacterial')))
print(len(os.listdir('base_dir/train_dir/viral')))
```

Check how many Validation images we now have in each folder after Data Augumentation

```
print(len(os.listdir('base_dir/val_dir/normal')))
print(len(os.listdir('base_dir/val_dir/bacterial')))
print(len(os.listdir('base_dir/val_dir/viral')))
```

Visualizing some of the Augumented Images

```
# plots images with labels within jupyter notebook
# source: https://github.com/smileservices/keras_utils/blob/master/utils.py

def plots(ims, figsize=(12,6), rows=5, interp=False, titles=None): # 12,6
    if type(ims[0]) is np.ndarray:
        ims = np.array(ims).astype(np.uint8)
        if (ims.shape[-1] != 3):
            ims = ims.transpose((0,2,3,1))
    f = plt.figure(figsize=figsize)
    cols = len(ims)//rows if len(ims) % 2 == 0 else len(ims)//rows + 1
    for i in range(len(ims)):
        sp = f.add_subplot(rows, cols, i+1)
        sp.axis('Off')
        if titles is not None:
            sp.set_title(titles[i], fontsize=16)
        plt.imshow(ims[i], interpolation=None if interp else 'none')

plots(imgs, titles=None) # titles=labels will display the image labels
```

 # End of Data Preparation

```
###
============================================================
============================ ###
```

 # Start of Model Building

Set Up the Generators

```
train_path = 'base_dir/train_dir'
valid_path = 'base_dir/val_dir'

num_train_samples = len(df_train)
num_val_samples = len(df_val)
```

```
train_batch_size = 10
val_batch_size = 10
image_size = 224

train_steps = np.ceil(num_train_samples / train_batch_size)
val_steps = np.ceil(num_val_samples / val_batch_size)


train_batches = ImageDataGenerator(
    preprocessing_function= \
    keras.applications.mobilenet.preprocess_input).flow_from_directory(
                                        train_path,
                                        target_size=(image_size,image_size),
                                        batch_size=train_batch_size,
                                        class_mode='categorical')
valid_batches = ImageDataGenerator(
    preprocessing_function= \
    keras.applications.mobilenet.preprocess_input).flow_from_directory(
                                        valid_path,
                                        target_size=(image_size,image_size),
                                        batch_size=val_batch_size,
                                        class_mode='categorical')

# Note: shuffle=False causes the test dataset to not be shuffled
test_batches = ImageDataGenerator(
    preprocessing_function= \
    keras.applications.mobilenet.preprocess_input).flow_from_directory(
                                        valid_path,
                                        target_size=(image_size,image_size),
                                        batch_size=val_batch_size,
                                        class_mode='categorical',
                                        shuffle=False)
```

Load MobileNet Model

```
mobile = keras.applications.mobilenet.MobileNet()

mobile.summary()
```

CREATE THE MODEL ARCHITECTURE

```
# Exclude the last 5 layers of the above model.
# This will include all layers up to and including global_average_pooling2d_1
x = mobile.layers[-6].output

# Create a new dense layer for predictions
# 3 corresponds to the number of classes
x = Dropout(0.25)(x)
predictions = Dense(3, activation='softmax')(x)

# inputs=mobile.input selects the input layer, outputs=predictions refers to the
# dense layer we created above.

model = Model(inputs=mobile.input, outputs=predictions)
type(mobile.layers)

# How many layers does MobileNet have?
len(mobile.layers)

model.summary()
```

Modifying the Model to get expected Shape in output for our project

```
from keras.models import Model
from keras.layers import Dropout, Dense, GlobalAveragePooling2D

# Remove the existing last layer
x = model.layers[-2].output

# Add new layers
```

```
x = GlobalAveragePooling2D()(x)
x = Dropout(rate=0.5)(x)
predictions = Dense(3, activation='softmax')(x)

# Define a new model
model = Model(inputs=model.input, outputs=predictions)

model.summary()

# We need to choose how many layers we actually want to be trained.

# Here we are freezing the weights of all layers except the
# last 40 layers in the new model.
# The last 40 layers of the model will be trained.

for layer in model.layers[:-40]:
    layer.trainable = False
```

Train the Model

```
model.compile(Adam(lr=0.001), loss='categorical_crossentropy',
              metrics=[categorical_accuracy])

# Get the labels that are associated with each index
print(valid_batches.class_indices)

# Add weights to try to make the model more sensitive to a specific class

class_weights={
    0: 1.0, # bacterial
    1: 1.0, # normal
    2: 1.0, # viral
}
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
```

```
from keras.layers import GlobalAveragePooling2D
filepath = "model.h5"
checkpoint = ModelCheckpoint(filepath, monitor='val_categorical_accuracy', verbose=1,
                             save_best_only=True, mode='max')

reduce_lr = ReduceLROnPlateau(monitor='val_categorical_accuracy', factor=0.5, patience=2,
                              verbose=1, mode='max', min_lr=0.00001)


callbacks_list = [checkpoint, reduce_lr]

history = model.fit_generator(train_batches, steps_per_epoch=train_steps,
                    validation_data=valid_batches,
                    validation_steps=val_steps,
                    epochs=10, verbose=1,
                   callbacks=callbacks_list)
```

Saving the model

```
model.save('model.h5')
```

Evaluate the model using the val set

```
# get the metric names so we can use evaulate_generator
model.metrics_names

# Here the the last epoch will be used.

val_loss, val_cat_acc = \
model.evaluate_generator(test_batches,
                         steps=val_steps)

print('val_loss:', val_loss)
print('val_cat_acc:', val_cat_acc)
```

```
# Here the best epoch will be used.

model.load_weights('model.h5')

val_loss, val_cat_acc = \
model.evaluate_generator(test_batches,
                         steps=val_steps)

print('val_loss:', val_loss)
print('val_cat_acc:', val_cat_acc)
```

Plot the Training Curves

```
# display the loss and accuracy curves

import matplotlib.pyplot as plt

acc = history.history['categorical_accuracy']
val_acc = history.history['val_categorical_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.figure()

plt.plot(epochs, acc, 'bo', label='Training cat acc')
plt.plot(epochs, val_acc, 'b', label='Validation cat acc')
plt.title('Training and validation cat accuracy')
```

```
plt.legend()
plt.figure()
```

Create a Confusion Matrix

```
# Get the labels of the test images.
# Note that cats and dogs are in seperate folders therefore
# the code below can get the labels depending on the folder the image is in.

test_labels = test_batches.classes

# We need these to plot the confusion matrix.
test_labels

# Print the label associated with each class
test_batches.class_indices

# make a prediction
predictions = model.predict_generator(test_batches, steps=val_steps, verbose=1)

Predictions.shape

# Source: Scikit Learn website
# http://scikit-learn.org/stable/auto_examples/
# model_selection/plot_confusion_matrix.html#sphx-glr-auto-examples-model-
# selection-plot-confusion-matrix-py



def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
```

```
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

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

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()

test_labels.shape



# argmax returns the index of the max value in a row
cm = confusion_matrix(test_labels, predictions.argmax(axis=1))

test_batches.class_indices
```

```
# Define the labels of the class indices. These need to match the
# order shown above.
cm_plot_labels = ['bacterial', 'normal', 'viral']

plot_confusion_matrix(cm, cm_plot_labels, title='Confusion Matrix')

# with above code we can view the confusion matrix
```

Create a Classification Report

```
# Get the filenames, labels and associated predictions

# This outputs the sequence in which the generator processed the test images
test_filenames = test_batches.filenames

# Get the true labels
y_true = test_batches.classes

# Get the predicted labels
y_pred = predictions.argmax(axis=1)

from sklearn.metrics import classification_report

# Generate a classification report


report = classification_report(y_true, y_pred, target_names=cm_plot_labels)

print(report)
```

Let's Make some Predictions

Load the Model

```
from tensorflow.keras.models import load_model

model = load_model('model.h5')

import cv2
import numpy as np
import matplotlib.pyplot as plt

#make prediction for the image at a specific location
img = cv2.imread('Data/chest_xray/test/PNEUMONIA/person1644_virus_2844.jpeg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # convert to RGB format
img = cv2.resize(img, (224, 224)) # resize to match the input size of the model
img = img / 255.0 # normalize the pixel values

# Display the image
plt.imshow(img)
plt.show()

class_names = ['Bacterial', 'Normal', 'Viral'] # List of class names in the order of the model's output
pred = model.predict(np.array([img]))[0] # make a prediction on the image and get the probabilities

for i, class_name in enumerate(class_names):
    class_perc = pred[i] * 100 # get the percentage of prediction for the class
    print(f"{class_name}: {class_perc:.2f}%")

#Thus we can view the predictions.
```

6.2 Code for Deployment :

The deployment code using streamlit

Importing the libraries

```
import streamlit as st
import cv2
import numpy as np
import tensorflow as tf

# Define the model loading and prediction function
def predict_class(img):
    # Load the model
    model =
tf.keras.models.load_model('C:\Users\MD.ARSHAD\_PNEUMONIA
DETECTION AND CLASSIFICATION\model.h5')
    # Preprocess the input image
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # convert to RGB format
    img = cv2.resize(img, (224, 224)) # resize to match the input size of the model
    img = img / 255.0 # normalize the pixel values
    # Make a prediction
    pred = model.predict(np.array([img]))[0]
    # Get the class names
    class_names = ['Bacterial', 'Normal', 'Viral']
    # Create the output dictionary
    output = {}
    for i, class_name in enumerate(class_names):
        output[class_name] = float(pred[i])
    return output

# Define the Streamlit app
def app():
    # Set the app title
    st.title("Chest X-ray Classifier")
```

```
    # Add an image uploader
    uploaded_file = st.file_uploader("Upload an image",
type=["jpg","jpeg","png"])
    if uploaded_file is not None:
        # Read the image file and show it
        image = cv2.imdecode(np.frombuffer(uploaded_file.read(), np.uint8), 1)
        st.image(image, caption='Uploaded Image', use_column_width=True)
        # Make a prediction on the image
        pred = predict_class(image)
        # Show the prediction results
        st.write("Prediction Results:")
        # Sort the predictions in descending order
        sorted_pred = sorted(pred.items(), key=lambda x: x[1], reverse=True)
        for class_name, prob in sorted_pred:
            st.write(f"{class_name}: {prob*100:.2f}%")
            st.progress(prob)
            st.text('\n') # Add some spacing between the progress bars
    else:
        # Load the default image
        default_image = cv2.imread('C:/Users/MD.ARSHAD/.spyder-
py3/Images/image.jpg')
        st.image(default_image, caption='Default Image', use_column_width=True)

# Run the app
if __name__ == '__main__':
    app()
```

CHAPTER 7

RESULTS

7.1 Classification Report :

The classification report is evaluating the performance of a pneumonia classification model. The report shows various metrics for each class in the dataset, as well as overall metrics for the entire dataset.

	precision	recall	f1-score	support
bacterial	0.88	0.84	0.86	417
normal	0.95	0.97	0.96	238
viral	0.71	0.75	0.73	224
accuracy			0.85	879
macro avg	0.85	0.85	0.85	879
weighted avg	0.85	0.85	0.85	879

Fig.7.1 Classification Report

For each class, the report shows the following metrics:

- **Precision:** This is the ratio of true positives to the total number of positive predictions for that class. In other words, it measures how many of the predicted positive cases are actually positive. A high precision indicates that the model is not making many false positive predictions.

- **Recall:** This is the ratio of true positives to the total number of actual positive cases for that class. In other words, it measures how many of the

actual positive cases the model was able to correctly identify. A high recall indicates that the model is not missing many positive cases.

- **F1-score:** This is the harmonic mean of precision and recall. It provides a single number that summarizes both precision and recall, with a value of 1 indicating perfect precision and recall, and a value of 0 indicating that the model is not performing better than random chance.

- **Support:** This is the number of samples in the dataset that belong to that class.

The report also shows two additional metrics for the entire dataset:

- **Accuracy:** This is the ratio of correct predictions to the total number of predictions. In other words, it measures how many of the predictions the model got right, regardless of the class.

- **Macro-averaged F1-score:** This is the average F1-score across all classes, weighted equally. It provides an overall measure of the model's performance that is not influenced by class imbalance.

- Based on the classification report, the model appears to perform well overall, with an accuracy of 0.85. Also Our Training Accuracy is recorded to be 0.95 which is good.

7.2 Output Screenshots :

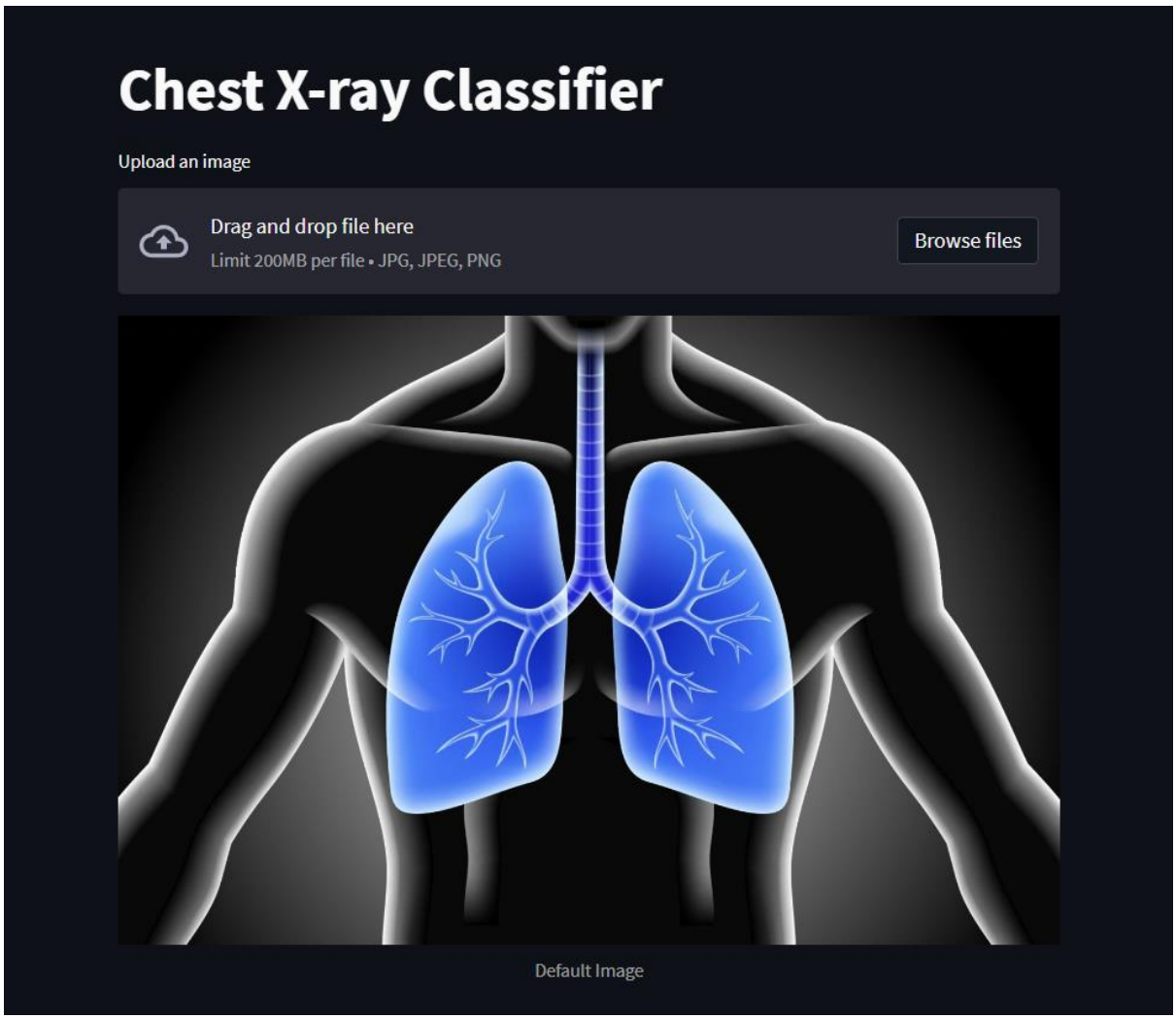


Fig.7.2 Home page

Figure 7.2 depicts the first look of our front end. We have a drag and drop file option with a button called “browse” which can be used to browse the images stored on the system’s hard disk.

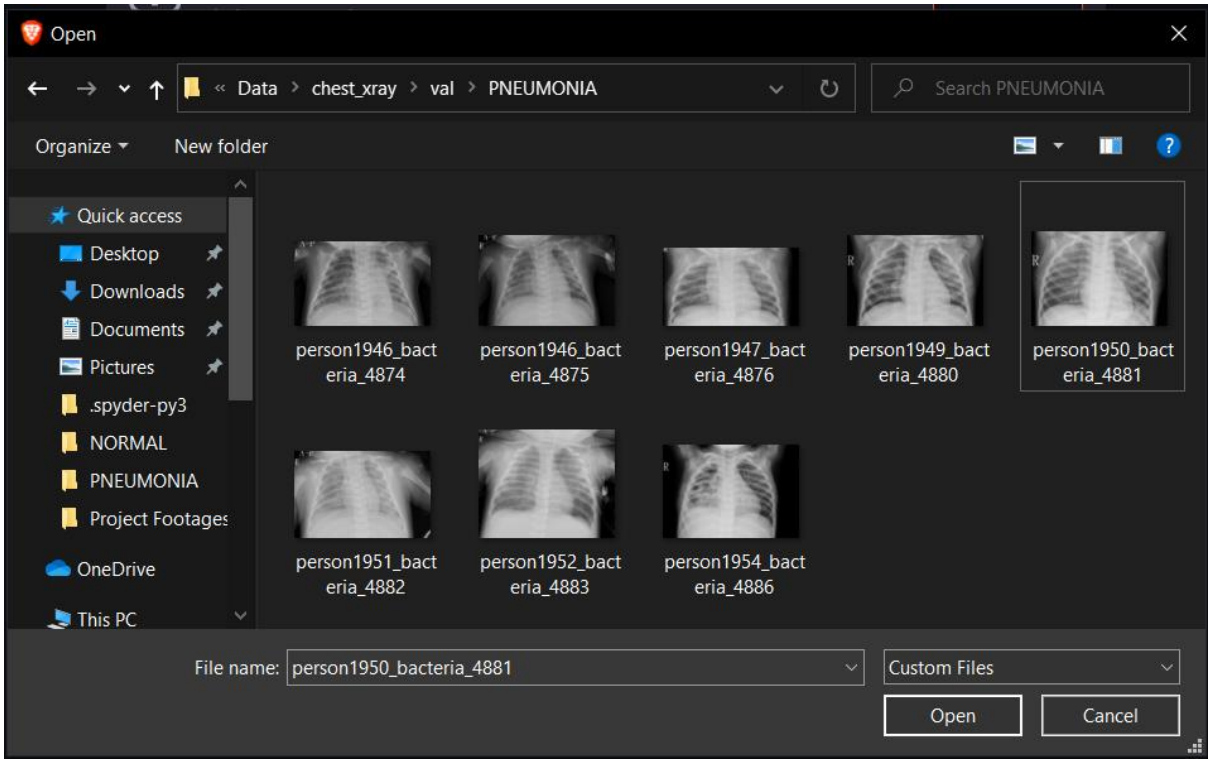


Fig.7.3 Selecting input from dataset

Figure 7.3 shows the popup which appears when the user clicks on the 'browse' button. The popup window will be having number of input images to be selected, this action should be confirmed with a double click or an open button.

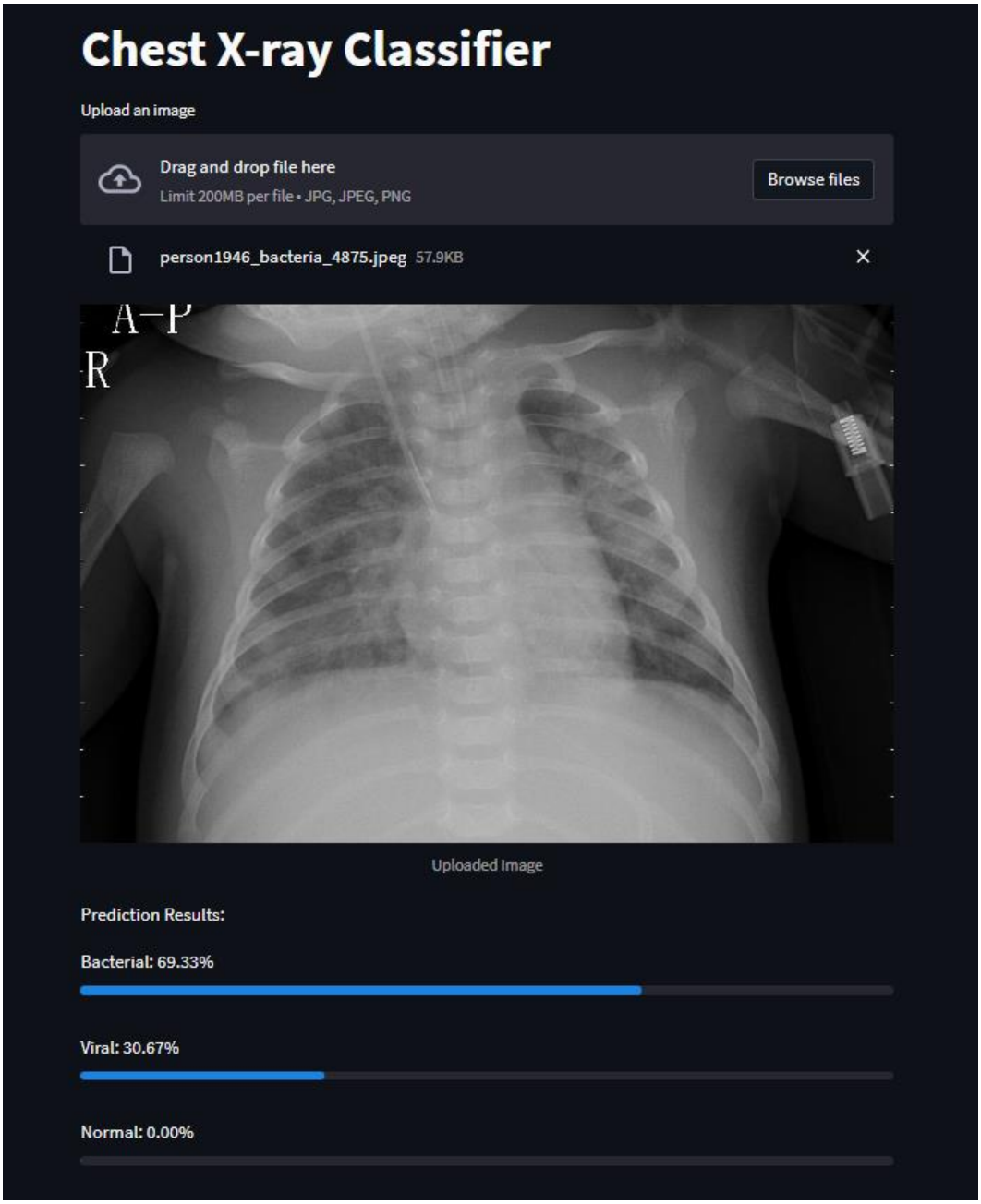


Fig 7.4 output and predictions

Once we selected the respective image, the deployed model will predict whether the person has pneumonia or not and if he has pneumonia it will classify the type of pneumonia i.e. Bacterial Pneumonia or Viral Pneumonia. The highest probability predicted will be displayed first and is followed in a descending order .The horizontal bar below depicts the graphical representation of the percentages.

CHAPTER 8

ETHICAL FRAMEWORK

An ethical framework for the classification of pneumonia into normal, viral, and bacterial categories should consider several key factors. Firstly, the system should prioritize patient safety and well-being, ensuring that the classification is accurate and reliable to enable appropriate treatment. The system should be developed and tested to minimize the potential for misclassification or false positives, which could lead to incorrect treatment decisions and harm patients.

The framework should also consider privacy and confidentiality. Patient data should be handled securely and in accordance with applicable laws and regulations. The system should only collect and use the minimum necessary information for the classification process, and individuals should have the right to access and control their own health data.

The framework should promote fairness and equity. The system should be accessible and affordable to all patients, regardless of their socio-economic status or geographic location. The system should also be designed to minimize bias or discrimination, ensuring that patients from all backgrounds receive accurate and appropriate diagnoses and treatments.

The framework should emphasize transparency and accountability. The system should be transparently designed and tested, with clear documentation of its accuracy and limitations. Any potential biases or errors should be identified and addressed, and patients should have access to a clear process for reporting and addressing any concerns or complaints.

By incorporating these ethical considerations into the development and deployment of the pneumonia classification system, it can be ensured that the system is designed and implemented in a responsible and justifiable manner that prioritizes patient safety and well-being, protects privacy and confidentiality, promotes fairness and equity, and emphasizes transparency and accountability.

CHAPTER 9

CONCLUSION AND FUTURE WORK

9.1 Conclusion

In this project, we have developed a pneumonia classification system using machine learning, computer vision techniques and MobileNet model. The system was deployed using Streamlit, a user-friendly web application framework, and achieved an accuracy of 85%. This indicates that the system can provide an accurate and reliable classification of pneumonia into normal, viral, and bacterial categories.

9.2 Future Work

While the system has demonstrated promising results, there are several areas for future work to improve its performance and functionality. One potential avenue for improvement is to incorporate more sophisticated machine learning algorithms or ensemble methods, which could help to improve the accuracy and reduce false positives. Another possibility is to expand the dataset used for training and testing the system, which could help to improve the robustness and generalizability of the model.

Furthermore, the system could be integrated with electronic health records or other health information systems to enable seamless data sharing and improve patient outcomes. Additionally, the system could be adapted for use in different clinical settings, such as rural or low-resource areas, where access to trained

medical professionals may be limited. Finally, ongoing monitoring and evaluation of the system's performance and ethical implications will be critical to ensure that it remains effective, safe, and equitable over time.

REFERENCES

[1] Rahman, T., Chowdhury, M., Khandakar, A., Islam, K., Mahbub, Z., et al. (2020). Transfer learning with deep convolutional neural network (CNN) for pneumonia detection using chest X-ray. Applied Sciences, vol. 10, no. 9, pp. 3233.

[2] Ibrahim, A., Ozsoz, M., Serte, S., Al-Turjman, F., & Yakoi, P. (2021). Pneumonia classification using deep learning from chest X-ray images during COVID-19. Cognitive Computation, pp. 1-13, PMID: 33425044.

[3] Rothan, H. A., & Byrareddy, S. N. (2020). The epidemiology and pathogenesis of coronavirus disease (COVID-19) outbreak. J. Autoimmun., vol. 109, pp. 102433.

[4] Zhai, P., Ding, Y., Wu, X., Long, J., Zhong, Y., & Li, Y. (2020). The epidemiology, diagnosis and treatment of COVID-19. Int. J. Antimicrob. Agents, vol. 55, no. 5, pp. 105955.

[5] Datta, S., & Roberts, K. (2020). A dataset of chest X-ray reports annotated with spatial role labeling annotations. Data Brief, vol. 32, pp. 106056.

[6] Chen, K-C., Yu, H-R., Chen, W-S., et al. (2020). Diagnosis of common pulmonary diseases in children by X-ray images and deep learning. Sci Rep, vol. 10, no. 1, pp. 17374.

[7] Rajpurkar, P., Irvin, J., Zhu, K., Yang, B., Mehta, H., Duan, T., ... & Lungren, M. P. (2017). Chexnet: Radiologist-level pneumonia detection on chest X-rays with deep learning.

[8] Kermany, D. S., Goldbaum, M., Cai, W., Valentim, C. C., Liang, H., Baxter, S. L., ... & Dong, J. (2018). Identifying medical diagnoses and treatable diseases by image-based deep learning.

[9] Rajpurkar, P., Irvin, J., Zhu, K., Yang, B., Mehta, H., Duan, T., ... & Lungren, M. P. (2018). Automated detection of pneumonia using chest X-ray images.

[10] Wang, W., Zhao, Z., Li, X., Lin, J., & Wong, S. T. (2019). Unsupervised deep feature learning for pneumonia detection in chest X-rays. Pattern Recognition, 86, 627-634.

[11] Li, L., Qin, L., Xu, Z., Yin, Y., Wang, X., Kong, B., ... & Lu, M. (2020). Using artificial intelligence to detect COVID-19 and community-acquired pneumonia based on pulmonary CT: Evaluation of the diagnostic accuracy.

[12] Ahsan, H., Ullah, H., Tariq, R., & Khan, M. A. (2020). Classification of pneumonia from chest X-ray images using deep learning with global and local features.

[13] Chunduri, S. S., Gopichandran, L., & Mathew, R. (2020).Multiclass classification of pneumonia using deep learning with chest X-ray images.

[14] El Asnaoui, K., Mehdaoui, H., El Hassouni, M., & El Yassa, M. (2021). Automatic detection and classification of pneumonia from chest X-ray images using deep convolutional neural networks.

[15] Ganie, M. A., Rather, M. A., Mir, N. A., & Hussain, M. (2021). Detection of pneumonia using chest X-ray images with convolutional neural networks: A review.

[16] Hussain, M., Aboalsamh, H., & Fakhrulddin, S. (2020). Pneumonia detection using deep learning: A review.

[17] Akselrod-Ballin, A., Friedman, E., & Goldberger, J. (2018). Pneumonia detection using deep convolutional neural networks and multi-label learning.

[18] Sinha, A., Ahmed, A., Kumar, P., & Bhattacharyya, P. (2020). A comparative study of deep learning models for pneumonia detection in chest X-ray images.

[19] Kumar, A., Kim, J. H., Lyndon, D., Kim, M. S., & Cai, W. (2018). Automatic detection and classification of pneumonia using X-ray images and deep convolutional neural networks.

[20] Singh, R., Garg, N., Gupta, N., & Gupta, A. (2020). Detection and classification of pneumonia in chest X-ray images using deep convolutional neural networks.