Early Classification Of Emphysema In CT Scans Using Deep Learning

A project report submitted in partial fulfillment of the requirements for

the award of the degree of

Bachelor of Technology

by

R JYOSHNA	**:**	**2111CS020194**
B JYOTHI	**:**	**2111CS020195**
T KARTHIK YADAV	**:**	**2111CS020201**
P SANDHYA KUMARI	**:**	**2111CS020254**
SK MAHABOOB VALI	**:**	**2111CS020257**

Under the guidance of

Prof. R.KARTHIK

Assistant Professor

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING
MALLA REDDY UNIVERSITY
(As per Telangana State Private Universities Act No.13 of 2020 and G.O.Ms.No.14, Higher Education (UE) Department)
HYDERABAD – 500043
TELANAGANA
INDIA

2023-24

Certificate

This is to certify that this is the bonafide record of the application development entitled, "**Early Classification Of Emphysema In CT Scans Using Deep Learning**" Submitted by

R. Jyoshna(2111CS020194)

B. Jyothi(2111CS020195)

T. Karthik Yadav(2111CS020201)

P Sandhya Kumari(2111CS020254)

SK Mahaboob Vali(2111CS020257)

B. Tech III year II semester, Department of CSE (AI&ML) during the year 2023-24. The results embodied in the report have not been submitted to any other university or institute for the award of any degree or diploma

INTERNAL GUIDE

Assistant Professor

HEAD OF THE DEPARTMENT

Dr. Thayaba Khatoon

Professor & HoD

ACKNOWLEDGEMENT

An endeavor over a long period can be advice and support of many well wishers. We take this opportunity to express our gratitude and appreciation to all of them.

We owe our tribute to **Dr. Thayyaba Khatoon(Head of Department)** , for giving all of us such a wonderful opportunity to explore ourselves and the outside world to work on the real-life scenarios where the machine learning is being used nowadays.

We are very grateful to our project guide **Prof.R.Karthik**, for the guidance ,inspiration and constructive suggestions that helped us in the development of this application.

We also thank our parents and family at large for their moral and financial support in funding the project to ensure successful completion of the project .

Abstract

Learning about Lung Diseases and their characterization is one of the most interesting research topics in recent years. With the various uses of medical images in hospitals, pathologies, and diagnostic centers, the size of the medical image datasets is also expanding expeditiously to capture the diseases in hospitals. Though a lot of research has been done on this particular topic still this field is confusing and challenging. There are lots of techniques in literature to classify medical images. The main drawback of traditional methods is the semantic gap that exists between the low-level visual information captured by imaging devices and high-level semantic information perceived by a human being. The difficulty of querying and managing the large datasets leads to a new mechanism called deep convolutional neural network. Chronic Obstructive Pulmonary Disease (COPD) is a prevalent respiratory disorder characterized by airflow limitation, often caused by emphysema. Early detection and classification of emphysema severity play a crucial role in effective treatment and management. Deep learning techniques have recently achieved an impressive result in the field of computer vision along with Medical Engineering. Our aim was to determine whether participant-level emphysema pattern could predict impairment and mortality when classified by using a deep learning method.

CONTENTS

CHAPTER 1

INTRODUCTION

A substantial number of disorders that threaten the glo bal population are lung-related. Cardiovascular complications like chronic obstructive pulmonary disease (COPD), asthma, fibrosis and pneumonia are impacting the foremost portion of society globally. Emphysema, which has a major risk factor for mortality worldwide, is one crucial constituent of COPD. The disease can cause breathing difficulties due to the massive growth of the respiratory tract. In specific, emphysema is sub classified into CLE, paraseptic emphysema (PSE) and PLE .For instance, CLE is generally related to smoking and PSE is not always related to the major signs or neurological disabilities. As a result, it is essential to classify and quantify emphysema. The CT scan has been deemed to be the foremost specific scanning technology to identify emphysema, classify its category and assess its seriousness. In CT, emphysema's categories contain different radiology appearances. Over the past few decades, several algorithms are suggested for classifying the emphysema in lung CT images. Such algorithms are categorized into supervised and unsupervised models. The goal of unsupervised algorithms is to find new emphysema categories, which go beyond the common categories rec organized on autopsy. A generative framework has been constructed to find the disease categories within emphysema and find patient groups, which are represented via different distributions of those categories. On the other hand, supervised algorithms concentrate on categorizing the typical emphysema categories detected on autopsy, which include various pathophysiologic significances. The variances in CT scans can impact the efficiency of classifying the emphysema; therefore, it is required to develop a model that is highly strong to those variabilities.

1.1 Problem Definition.

The primary objective of this project is to develop an advanced predictive model capable of accurately classifying emphysema patterns in CT (Computed Tomography) images, with the ultimate goal of facilitating early detection and differentiation of emphysema severity. particularly by employing the DenseNet121 architecture, this study aims to overcome these challenges and enhance the accuracy of emphysema classification. DenseNet121 is chosen for its proven

efficiency in feature extraction and classification tasks, making it well-suited for the complexities of medical image analysis. Through meticulous training and validation processes, the model is expected to learn intricate patterns and subtle variations indicative of different emphysema patterns, such as Clear Lung Emphysema (CLE), Patchy Lung Emphysema (PLE), Predominantly Subpleural Emphysema (PSE), and Non-Emphysematous (NT) regions. Furthermore, accurate classification can aid in developing personalized management strategies tailored to individual patients, optimizing their quality of life and clinical outcomes.

1.2 Objective Of Project

The objective of the project is to develop a deep learning-based model using DenseNet121 to accurately classify emphysema patterns in CT scans. This classification aims to enable early detection and differentiation of emphysema severity, facilitating effective treatment planning and prediction of potential impairment and mortality associated with the disease. By leveraging advanced deep learning techniques, the project aims to enhance the accuracy and reliability of emphysema classification compared to traditional methods, ultimately providing clinicians with a valuable tool for improving patient care and management strategies for COPD and emphysema.

1.3 Scope Of Project

Early Detection: The project aims to develop a CNN model for the early classification of emphysema in CT scans, allowing for timely intervention and treatment, which can improve patient outcomes.

Automated Diagnosis: By leveraging deep learning techniques, the project offers the potential for automating the diagnosis process, reducing the burden on radiologists and healthcare professionals.

Scalability: Once trained, the CNN model can be scaled up to process large volumes of CT scans efficiently, making it applicable in busy clinical settings.

Generalizability: The developed model has the potential to generalize well to unseen data, enabling its deployment across different healthcare institutions and patient populations.

1.4 Limitations of Project

Data Quality: The performance of the CNN model heavily relies on the quality and diversity of the training data. Limited or biased datasets may lead to suboptimal performance or biased predictions.

Interpretability: Deep learning models, particularly CNNs, are often considered black-box models, making it challenging to interpret the reasoning behind their predictions. This lack of interpretability may hinder trust and acceptance among healthcare professionals.

Data Privacy: CT scans contain sensitive patient information, raising concerns about data privacy and security. Proper measures must be in place to ensure compliance with healthcare regulations and protect patient confidentiality.

False Positives/Negatives: Like any diagnostic tool, the CNN model may produce false positives or false negatives, leading to misdiagnosis or missed diagnoses. Clinical validation and collaboration with healthcare professionals are essential to mitigate these risks.

Ethical Considerations: Ethical considerations, such as algorithmic bias, fairness, and equity, must be carefully addressed throughout the project to prevent unintended consequences and ensure equitable access to healthcare services.

CHAPTER 2

ANALYSIS

2.1 Project Planning and Research

1. Project Scope Definition:

- Clearly define the objectives of the project: to develop a deep learning model capable of accurately detecting and classifying emphysema in CT scans at an early stage.
- Specify the use of VGG16, DenseNet121, and InceptionV3 architectures for comparison purposes.
- Define the frontend technology as ReactJS and the backend technology as Flask.

2. Literature Review:

- Conduct a comprehensive review of existing literature on emphysema detection using CT scans and deep learning techniques, focusing on studies that have utilized VGG16, DenseNet121, and InceptionV3 architectures.
- Identify relevant research papers, methodologies, datasets used, and the performance of the models developed in those studies.
- Determine the best practices and potential challenges associated with implementing these architectures in a real-world application.

3. Data Acquisition and Preprocessing:

- Identify and acquire suitable datasets containing CT scans of patients with and without emphysema, ensuring proper consent and ethical considerations.
- Preprocess the CT scan images, including resizing, normalization, and augmentation as necessary, to prepare them for input into the deep learning models.

4. Model Development:

- Implement the VGG16, DenseNet121, and InceptionV3 architectures using deep learning frameworks such as TensorFlow or PyTorch.
- Train each model using the preprocessed CT scan data, optimizing hyperparameters and monitoring performance metrics during training.

- Implement transfer learning techniques if applicable, leveraging pre-trained weights on large-scale image datasets like ImageNet to improve model convergence and generalization.

5. Integration with Flask Backend:

- Develop a Flask backend to serve as the API for the deep learning models.
- Implement endpoints for model inference, allowing the frontend to send CT scan images to the backend for classification.
- Ensure proper error handling, input validation, and security measures within the backend API.

6. Development of ReactJS Frontend:

- Develop a user-friendly web interface using ReactJS to interact with the Flask backend.
- Design the frontend to allow users to upload CT scan images and visualize the classification results returned by the backend.
- Implement features for user authentication, data visualization, and model performance monitoring as needed.

7. Testing and Evaluation:

- Conduct thorough testing of the integrated system, including unit tests for individual components and end-to-end testing of the entire application flow.
- Evaluate the performance of each deep learning model in terms of classification accuracy, sensitivity, specificity, and computational efficiency.
- Collect feedback from users and domain experts to identify areas for improvement and validation of the model's clinical relevance.

8. Documentation and Deployment:

- Document the entire research process, including data preprocessing steps, model architectures, training procedures, and system integration details.
- Prepare user documentation for the frontend application, providing instructions for installation, usage, and troubleshooting.
- Deploy the application to a cloud platform or web server, ensuring scalability, reliability, and security

2.2 Software requirement specification

2.2.1 Software requirement

1. **Deep Learning Models**:
    - Utilize three pre-trained deep learning architectures:
        - DenseNet121: A densely connected convolutional network.
        - VGG16: Known for its simplicity and effectiveness.
        - Inception V3: Captures salient features at different levels.
        - These models will classify emphysema severity based on CT scans.

2. **Frontend (ReactJS)**:

    - Design a user-friendly interface for uploading CT scan images.
    - Display predicted emphysema severity (e.g., mild, moderate, severe) using the deep learning models.

3. **Backend (Flask)**:

    - Set up API endpoints to receive CT scan images from the frontend.
    - Process images and pass them to the deep learning models.
    - Return emphysema severity predictions to the frontend.

4. **Google Colab**:

        - Use Colab for training and fine-tuning the deep learning models.
        - Leverage its free GPU resources to speed up model training

2.2.2 Hardware requirement

No specific hardware requirements are needed,any PC with a python IDLE can run this model. Remember to ensure that your PC meets the following minimum requirements:

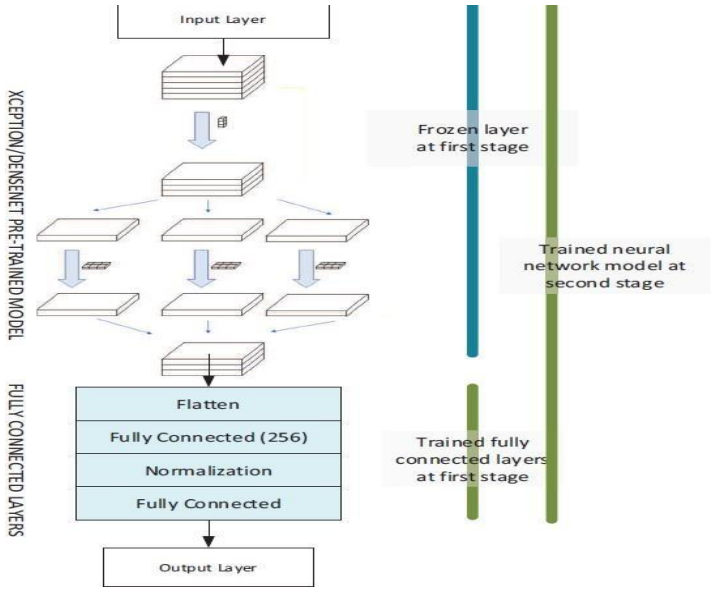
- **Processor**: A modern CPU (e.g., Intel Core i5 or AMD Ryzen) with multiple cores.
- **Memory (RAM)**: At least 4GB RAM (though more is beneficial for larger datasets).
- **Storage**: Sufficient disk space for storing the dataset, code, and model checkpoints.
- **Python Environment**: Install Python and the required libraries (such as TensorFlow, Keras, and Flask) to build and run the project.

2.3. Model Selection

1. **DenseNet121:** DenseNet121 is a more recent CNN architecture that introduces dense connectivity between layers. It connects each layer to every subsequent layer in a feed-forward fashion, enabling feature reuse and enhancing gradient flow during training

2. **Define Evaluation Metrics:** Determine the evaluation metrics relevant to your task. For emphysema detection, common metrics include accuracy, precision, recall, F1-score, and area under the receiver operating characteristic curve (AUC-ROC).
3. **Training and Validation:** Train each model (DenseNet121, VGG16, InceptionV3) on the augmented training dataset. Monitor the performance on the validation set during training to identify potential overfitting and adjust hyperparameters accordingly
4. **Evaluate Performance:** After training, evaluate the performance of each model on the test set using the predefined evaluation metrics. Compare metrics such as accuracy, precision, recall, F1-score, and AUC-ROC across models to assess their effectiveness in emphysema detection**.**
5. **Cross-Validation:** If your dataset is limited in size, consider performing cross-validation to ensure robustness of the results. K-fold cross-validation involves splitting the dataset into K subsets, training K models, each time using K-1 subsets for training and the remaining subset for validation.
6. **Model Interpretability:** Evaluate the interpretability of each model, considering how well it can provide insights into the features that contribute to emphysema detection. Visualization techniques such as class activation maps or saliency maps can help interpret model decisions.
7. **Fine-Tuning and Optimization:** Once the best model is selected, consider fine-tuning hyperparameters or conducting further optimization to enhance its performance, if necessary.

2.3 Architecture



CHAPTER 3

DESIGN

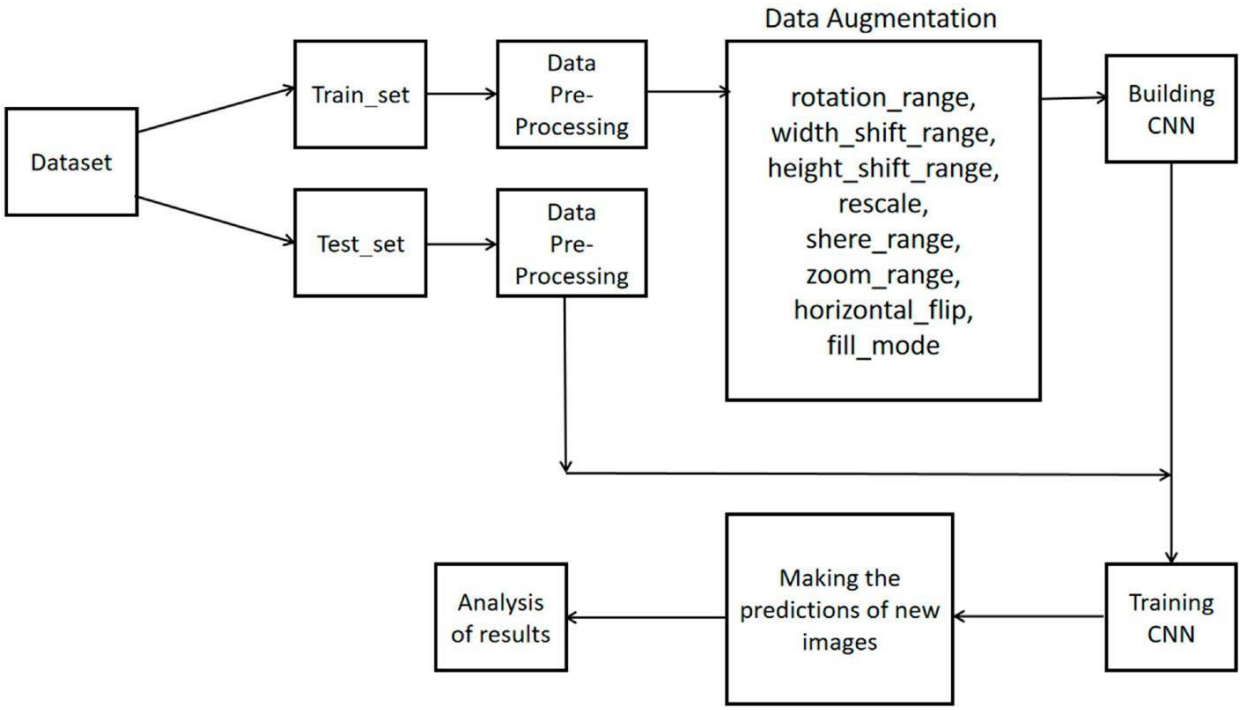
3.1. Introduction

Lung diseases pose a significant health burden globally, affecting millions of individuals and posing challenges for effective diagnosis and treatment. Among these diseases, emphysema stands out as a major risk factor for morbidity and mortality, particularly in the context of chronic obstructive pulmonary disease (COPD). Early detection and characterization of emphysema are paramount for initiating timely interventions and improving patient outcomes.

Traditional methods for emphysema detection in lung CT images have been hindered by the semantic gap between low-level visual features captured by imaging devices and high-level semantic information perceived by human experts. However, recent advancements in deep learning offer promising avenues for bridging this gap and enhancing the accuracy and efficiency of emphysema detection.

In this study, we present a comprehensive investigation into the application of deep learning techniques for the detection of emphysema in lung CT images. Our primary objective is to develop and evaluate deep learning models capable of accurately identifying emphysema patterns and assessing their severity, with the ultimate goal of facilitating early diagnosis and personalized management strategies for patients at risk of COPD-related complications.In summary, our project represents a significant step towards harnessing the power of deep learning for emphysema detection in lung CT images. By leveraging cutting-edge techniques and methodologies, we strive to enhance the accuracy, efficiency, and clinical relevance of emphysema diagnosis, ultimately leading to improved patient care and outcomes in the realm of respiratory health. Upon achieving satisfactory performance, the model is seamlessly deployed and integrated into clinical workflows, empowering healthcare professionals with real-time emphysema detection capabilities. Continuous monitoring and improvement post-deployment, guided by user feedback and ongoing evaluation, ensure the model's reliability and relevance in enhancing patient care and outcomes. Through this systematic and iterative design process, we aim to deliver a robust and reliable deep learning solution for emphysema detection, contributing to advancements in respiratory health management

3.2. Diagrams



3.3. Data Set Descriptions

Image Path: This column contains the file paths to the lung CT scan images. These images are likely stored as JPEG files.

Cluster Label: This column indicates the cluster label associated with each image. From the provided examples, it seems that the images are classified into different clusters. Clustering is a technique used to group similar data points together.

Category: This column specifies the emphysema category associated with each image. Emphysema is a type of lung disease characterized by the destruction of lung tissue, specifically the alveoli, which are the small air sacs in the lungs where oxygen exchange occurs. The categories mentioned in the dataset are:

- CLE: Likely stands for Centrilobular Emphysema, which is a type of emphysema where the damage primarily occurs in the central part of the lobule within the lungs. It is commonly associated with smoking.

- PSE: This may stand for Paraseptal Emphysema, which is another type of emphysema characterized by damage near the pleura (the membrane that covers the lungs). Paraseptal emphysema can occur without any symptoms or may cause spontaneous pneumothorax
- NT: This category could represent Non-Emphysematous Tissue or Normal Tissue. It seems to indicate areas of the lung that do not exhibit emphysematous changes.

The dataset seems to be part of a project aimed at developing a deep learning-based model using DenseNet121 to accurately classify emphysema patterns in lung CT scans. This classification is crucial for early detection, differentiation of emphysema severity, and effective treatment planning. By leveraging deep learning techniques, the project aims to improve the accuracy and reliability of emphysema classification compared to traditional methods.

3.4 Data Preprocessing Techniques

Preprocessing plays a crucial role in preparing your data for training deep learning models. Here are some data preprocessing techniques specific to your project on emphysema detection in lung CT images:

1. **Data Augmentation:** Since medical imaging datasets are often limited in size, data augmentation techniques can help increase the diversity of your training data and improve the generalization ability of your models. Common augmentation techniques for CT images include:
    - **Rotation:** Rotate the images by a certain degree to simulate different orientations.
    - **Translation:** Shift the images horizontally and vertically to simulate variations in position.
    - **Scaling:** Resize the images to simulate different resolutions or zoom levels.
    - **Flipping:** Flip the images horizontally or vertically to augment the dataset with mirrored versions.
    - **Elastic deformation:** Introduce local deformations to simulate anatomical variability.
    - **Adding noise:** Introduce random noise to simulate noise artifacts commonly present in medical images.

2. **Normalization:** Normalize the pixel values of the CT images to ensure consistency and facilitate convergence during training. Common normalization techniques include:
   - Min-max scaling: Scale pixel values to a range between 0 and 1 or -1 and 1.
   - Z-score normalization: Standardize pixel values to have a mean of 0 and a standard deviation of 1.
3. **Resampling and Cropping:** CT images may have different dimensions and resolutions, which can affect modelperformance. Resample the images to a consistent voxel size and crop them to a standardized size to ensure uniformity across the dataset.
4. **Data Balancing:** Imbalanced datasets, where one class (e.g., emphysema-positive) is significantly more prevalent than the other, can bias model training. Apply techniques such as oversampling, undersampling, or class-weighted loss functions to address class imbalance and ensure that the model learns from all classes equally.
5. **Data Splitting:** Split your dataset into training, validation, and test sets to assess model performance accurately. Typically, a common split ratio is 70% for training, 15% for validation, and 15% for testing, but this can vary based on the size of your dataset and specific requirements.

By implementing these preprocessing techniques, you can effectively prepare your lung CT image data for training deep learning models for emphysema detection. Each technique contributes to enhancing data quality, consistency, and model performance, ultimately leading to more accurate and robust predictions.

3.5. Methods & Algorithms

Convolutional Neural Networks (CNNs):

- CNNs are deep learning models specifically designed for processing and classifying visual data, such as images.
- They consist of multiple layers, including convolutional layers, pooling layers, and fully connected layers.
- Convolutional layers apply convolution operations to extract features from input images. These operations involve sliding small filters (kernels) across the input image to detect patterns and features.

- Pooling layers downsample feature maps obtained from convolutional layers, reducing their spatial dimensions and extracting the most important features.
- Fully connected layers process the flattened feature maps to make predictions, typically using activation functions like ReLU (Rectified Linear Unit) to introduce non-linearity.

Data Augmentation:

- Data augmentation involves applying transformations such as rotation, flipping, scaling, and cropping to input images.
- By augmenting the training dataset with these transformed images, data augmentation increases its diversity and helps the model generalize better to unseen data.
- Data augmentation is particularly useful when the training dataset is limited, as it effectively increases the effective size of the dataset without requiring additional labeled samples.
- Techniques such as random rotation, horizontal flipping, and random scaling are commonly used for data augmentation in image classification tasks.

DenseNet (Densely Connected Convolutional Networks):

- DenseNet is a type of CNN architecture characterized by dense connections between layers.
- In DenseNet, each layer is connected to every other layer in a feed-forward manner, creating direct connections between all layers within a dense block.
- These dense connections facilitate feature reuse and promote better gradient flow during training, addressing issues such as vanishing gradients.
- DenseNet architectures, such as DenseNet-121, have shown superior performance in image classification tasks due to their ability to learn intricate patterns and features from images.
- DenseNet-121 specifically refers to a variant of DenseNet with 121 layers, offering a good balance between model complexity and performance.

Ensemble Methods:

- Ensemble methods involve combining predictions from multiple individual models to improve overall performance and robustness.
- Common ensemble methods include averaging predictions, stacking, and boosting

CHAPTER 4

DEPLOYMENT AND RESULTS

4.1 Introduction

As the field of medical imaging continues to evolve, the deployment of advanced technologies for disease diagnosis and management has become increasingly crucial. In particular, the deployment of deep learning models holds immense promise in revolutionizing healthcare practices, offering clinicians powerful tools for accurate and efficient diagnosis of complex conditions such as Chronic Obstructive Pulmonary Disease (COPD) and its subcategory, emphysema. The deployment of a deep learning-based model represents a significant advancement in early detection and precise characterization of disease severity. By leveraging sophisticated neural network architectures like DenseNet121, clinicians gain access to a powerful tool capable of automatically analyzing medical images with unprecedented accuracy and efficiency.The deployment of such a model involves integrating it into existing clinical workflows, ensuring seamless interaction with radiology departments, diagnostic centers, and healthcare professionals. This process requires careful consideration of factors such as model reliability, interpretability, and scalability, as well as adherence to regulatory standards and data privacy requirements.

4.2 Source Code

4.2.1 Fronted using React.js

Header.js

```
import React,{useState} from 'react';
import ReactDOM from 'react-dom/client';
import { Link } from 'react-router-dom';
import menu from '../images/menu-icon.png'

const Header = () => {
  const [clicked,setClicked]=useState(false);
  const handleClick=()=>{
    if(clicked){
      setClicked(false)
    }else{
      setClicked(true)
    }
```

```
  }
  return (
    <div className="header">
        <h2 className='proj-name'>Emphysema Detection</h2>
        <div className='nav-links'>
          <ul>
            <li><Link to='/'>HOME</Link></li>
            <li><Link to='typesofemphysema'>Types of Emphysema</Link></li>
          </ul>
        </div>
        <div className='menu' onClick={handleClick}><img src={menu } /></div>
        <div className='side-bar'
style={clicked?{display:"block"}:{display:"none"}}>
        <ul>
            <li><Link to='/'>HOME</Link></li>
            <li><Link to='typesofemphysema'>Types of Emphysema</Link></li>
        </ul>
        </div>
    </div>

  )
}
export default Header
```

Home.js

```
import React from 'react';
import Header from "./Header";
import Introduction from "./Introduction";
import Userdetails from "./Userdetails";
import PredictionForm from "./PredictionForm"

const Home = () => {
  return (
    <div>
         <Header />
       <Introduction />
        <Userdetails />
        {/* <PredictionForm/> */}

    </div>
  )
}
```

```
export default Home
```

Introduction.js

```
import React from 'react';
import Header from "./Header";
import Introduction from "./Introduction";
import Userdetails from "./Userdetails";
import PredictionForm from "./PredictionForm"

const Home = () => {
  return (
    <div>
         <Header />
        <Introduction />
         <Userdetails />
        {/* <PredictionForm/> */}

    </div>
  )
}

export default Home
```

Userdetails.js

```
import React, { useState } from 'react';

const Userdetails = () => {
  const [selectedFile, setSelectedFile] = useState(null);
  const [prediction, setPrediction] = useState(null);

  const handleClick = () => {
    const fileInput = document.createElement("input");
    fileInput.type = "file";
    fileInput.accept = ".png, .jpg, .jpeg"; // Limit accepted file types if
needed
    fileInput.onchange = handleFileChange;
    fileInput.click();
  };

  const handleFileChange = (event) => {
    const file = event.target.files[0];
    setSelectedFile(file);
  };
```

```
  const handleReset = () => {
    setSelectedFile(null);
    setPrediction(null); // Reset prediction when resetting file
  };

  const handleSubmit = async (event) => {
    event.preventDefault(); // Prevent default form submission behavior

    const formData = new FormData();
    formData.append('image', selectedFile);

    try {
      const response = await fetch('http://127.0.0.1:5000/predict', {
        method: 'POST',
        body: formData,
      });

      if (!response.ok) {
        throw new Error('Failed to fetch');
      }

      const data = await response.json();
      setPrediction(data);
    } catch (error) {
      console.error('Error:', error);
      // Handle error state if needed
    }
  };

  return (
    <div>
      <h3 className='details'>Enter your CT Scan here</h3>
      <div>
        <form onSubmit={handleSubmit}> {/* Use onSubmit event to trigger
handleSubmit */}
        <div class="parent-container">
  <div class="clickable-box file-label label" onClick={handleClick}>
    {selectedFile ? selectedFile.name : 'choose File ... '}
  </div>
</div>

          {/* <div className='container'>
            <div className="item2">
              <p className='U-file'>Upload File</p>
```

```
              <div className="clickable-box file-label" onClick={handleClick}>
                {selectedFile ? selectedFile.name : 'choose File. .. '}
              </div>
            </div>
          </div> */}
          {selectedFile && (
            <img
              src={URL.createObjectURL(selectedFile)}
              alt="Selected Image"
              className="selected-image"
            />
          )}
          <div><center><input type="submit" className="submit" /></center></div>
          <div><center><input type="reset" className="submit"
onClick={handleReset} /></center></div>
        </form>
      </div>
      {prediction && (
        <div className='prediction'>
          <p>Class: {prediction.class}</p>
          <p>Category: {prediction.category}</p>
          <p>Emphysema: {prediction.emphysema}</p>
        </div>
      )}
    </div>
  )
}

export default Userdetails;
```

4.2.2 Backend using Flask

```
from flask import Flask, request, jsonify
from flask_cors import CORS
import tensorflow as tf
import numpy as np
from PIL import Image
import io

app = Flask(__name__)
CORS(app)

# Load model
model_path = "disease_model.h5"  # Update with correct path
```

```
try:
    model = tf.keras.models.load_model(model_path)
    print("Model loaded successfully.")
except Exception as e:
    print(f"Error loading model: {e}")

# Define image size
img_size = (224, 224)

# Define Flask routes
@app.route('/predict', methods=['POST'])
def prediction():
    if 'image' not in request.files:
        return jsonify({'error': 'No image found in request'}), 400

    image_file = request.files['image']
    if image_file.filename == '':
        return jsonify({'error': 'No selected file'}), 400

    try:
        # Read the image file as a PIL image
        img = Image.open(io.BytesIO(image_file.read()))
        # Convert RGBA image to RGB
        img = img.convert('RGB')
        # Resize image to (224, 224)
        img = img.resize(img_size)
        # Convert PIL image to numpy array
        image_array = np.array(img)

        # Make prediction
        prediction_result = model.predict(np.expand_dims(image_array, axis=0))

        # Get class label and corresponding probability
        class_label = np.argmax(prediction_result)  # Class label with highest
probability
        class_probability = prediction_result[0][class_label]  # Probability of
predicted class

        # Define category and emphysema based on class label and probability
        if class_label == 0:
            category_label = 'NT'
            if class_probability >= 0.5: # Adjust threshold as needed
                emphysema = 'No'
            else:
                emphysema = 'Yes'
```

```
        elif class_label == 1:
            category_label = 'CLE'
            emphysema = 'Yes'
        elif class_label == 2:
            category_label = 'PSE'
            emphysema = 'Yes'
        elif class_label == 3:
            category_label = 'PLE'
            emphysema = 'Yes'
        else:
            return jsonify({'error': 'Invalid class label predicted'}), 500

        return jsonify({'class': int(class_label), 'category': category_label,
'emphysema': emphysema}), 200
    except Exception as e:
        return jsonify({'error': str(e)}), 500

if_name_== '_main_':
    app.run(host='0.0.0.0', port=5000)
```

4.3 Model Implementation and Training

Data Preprocessing:

- The first step involved collecting a diverse dataset of lung CT scans, encompassing various emphysema patterns and severity levels.
- Preprocessing techniques were applied to standardize image resolution, adjust intensity levels, and normalize features, ensuring consistency across the dataset.

Model Selection:

- DenseNet121, a state-of-the-art deep convolutional neural network (CNN), was chosen for its effectiveness in image classification tasks and ability to capture intricate patterns within medical images.
- The architecture of DenseNet121 promotes feature reuse and alleviates the vanishing gradient problem, facilitating robust learning even with limited training data.

Training Strategy:

- The dataset was divided into training, validation, and testing sets to evaluate model performance comprehensively.

- Transfer learning was employed by initializing the DenseNet121 model with pre-trained weights on large-scale image datasets such as ImageNet, accelerating convergence and enhancing generalization.
- Fine-tuning was conducted by retraining the model's last few layers on the emphysema dataset to adapt to specific patterns relevant to COPD.

Hyperparameter Tuning:

- Hyperparameters such as learning rate, batch size, and optimizer settings were fine-tuned through iterative experimentation to optimize model performance.
- Techniques like learning rate scheduling and early stopping were employed to prevent overfitting and improve convergence speed.

Model Evaluation:

- The trained model was evaluated on the test set using standard performance metrics such as accuracy, precision, recall, and F1-score.
- Additionally, qualitative assessment through visual inspection of predicted emphysema regions against ground truth annotations provided insights into model behavior and potential areas for improvement.

Source code

```
from keras.preprocessing.image import ImageDataGenerator, load_img,
img_to_array
import os
from PIL import Image

# Set the path to your dataset
dataset_path = '/content/drive/My Drive/disease/slices'

# Create an ImageDataGenerator for data augmentation
datagen = ImageDataGenerator(
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

```
# Get the list of all image files in the directory
image_files = [os.path.join(dataset_path, filename) for filename in
os.listdir(dataset_path) if filename.endswith('.tiff')]

# Iterate through each image in the directory
for image_file in image_files:
    # Load the image
    img = load_img(image_file)

    # Convert the image to a Numpy array
    x = img_to_array(img)

    # Reshape the array to (1, height, width, channels)
    x = x.reshape((1,) + x.shape)

    # Generate augmented images and save them to the specified directory
    i = 0
    for batch in datagen.flow(x, batch_size=1, save_to_dir=dataset_path,
save_prefix='aug', save_format='tiff'):
        i += 1
        if i > 20: # Generate 5 augmented images for each original image
            break

# After generating all augmented images, process them to have a white
background
augmented_image_files = [os.path.join(dataset_path, filename) for filename
in os.listdir(dataset_path) if filename.startswith('aug') and
filename.endswith('.tiff')]
for augmented_image_file in augmented_image_files:
    augmented_image = Image.open(augmented_image_file)
    augmented_image = augmented_image.convert("RGBA")  # Convert to RGBA
mode
    datas = augmented_image.getdata()

    # Replace non-white pixels (considering values below 240 as non-white)
    new_data = []
    for item in datas:
        # Calculate the average of RGB values
        avg_rgb = sum(item[:3]) / 3

        # Set a threshold to determine if the pixel is close to white
        threshold = 240

        if avg_rgb < threshold:
            # Pixel is not close to white, preserve its color
```

```
            new_data.append(item)
        else:
            # Pixel is close to white, set it to fully transparent white
            new_data.append((255, 255, 255, 0))

    augmented_image.putdata(new_data)
    augmented_image.save(augmented_image_file, "tiff")
```

convert the images .tiff format to jpeg format

```
from PIL import Image
import os

# Directory containing TIFF images
source_dir = "/content/drive/My Drive/disease/slices"
# Directory to save JPEG images
target_dir = "/content/drive/My Drive/disease/slicesjpeg"

if not os.path.exists(target_dir):
    os.makedirs(target_dir)

for filename in os.listdir(source_dir):
    if filename.endswith('.tiff') or filename.endswith('.tif'):
        # Path to the current TIFF file
        tiff_path = os.path.join(source_dir, filename)
        # Convert TIFF to JPEG
        with Image.open(tiff_path) as img:
            # Remove the file extension and add ".jpg"
            jpeg_path = os.path.join(target_dir,
os.path.splitext(filename)[0] + '.jpg')
            img.convert('RGB').save(jpeg_path, "JPEG")
```

CSV Dataset

```
import csv

# Define the file path for saving the dataset
output_file = "/content/drive/My
Drive/disease/image_cluster_labels.csv"  # Adjust the file path as needed

# Open the CSV file in write mode and create a CSV writer object
with open(output_file, mode='w', newline='') as file:
    writer = csv.writer(file)

    # Write the header row
    writer.writerow(['Image Path', 'Cluster Label', 'Category'])
```

```
    # Write the data rows
    for image_path, cluster_label in zip(image_paths, cluster_labels):
        category_name = assign_category(cluster_label)
        writer.writerow([image_path, cluster_label, category_name])

print(f"Dataset saved to: {output_file}")
```

4.4 Model Evaluation Metrics

These metrics collectively provide a comprehensive understanding of the model's performance in classifying emphysema patterns in CT scans, considering both the model's ability to correctly identify positive cases (emphysema) and its ability to avoid misclassification.

Accuracy: The ratio of correctly classified images to the total number of images evaluated. It provides an overall measure of the model's performance.

Precision: The ratio of true positive predictions to the sum of true positive and false positive predictions. It indicates the model's ability to correctly identify positive cases while minimizing false positives.

Recall (Sensitivity): The ratio of true positive predictions to the sum of true positive and false negative predictions. It measures the model's ability to correctly detect positive cases from all actual positive cases.

F1-score: The harmonic mean of precision and recall. It provides a balance between precision and recall, considering both false positives and false negatives.

TP (True Positives): Number of correctly predicted positive cases (emphysema).

TN (True Negatives): Number of correctly predicted negative cases (non-emphysema).

FP (False Positives): Number of incorrectly predicted positive cases (predicted as emphysema but actually non-emphysema).

FN (False Negatives): Number of incorrectly predicted negative cases (predicted as non-emphysema but actually emphysema).

Source Code:

```
import tensorflow as tf
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Dropout,
UpSampling2D, concatenate

def unet(input_shape=(256, 256, 1)):
    inputs = Input(input_shape)

    # Encoder
    conv1 = Conv2D(64, 3, activation='relu', padding='same')(inputs)
    conv1 = Conv2D(64, 3, activation='relu', padding='same')(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

    conv2 = Conv2D(128, 3, activation='relu', padding='same')(pool1)
    conv2 = Conv2D(128, 3, activation='relu', padding='same')(conv2)
```

```
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)

    conv3 = Conv2D(256, 3, activation='relu', padding='same')(pool2)
    conv3 = Conv2D(256, 3, activation='relu', padding='same')(conv3)
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)

    conv4 = Conv2D(512, 3, activation='relu', padding='same')(pool3)
    conv4 = Conv2D(512, 3, activation='relu', padding='same')(conv4)
    drop4 = Dropout(0.5)(conv4)
    pool4 = MaxPooling2D(pool_size=(2, 2))(drop4)

    # Bottom
    conv5 = Conv2D(1024, 3, activation='relu', padding='same')(pool4)
    conv5 = Conv2D(1024, 3, activation='relu', padding='same')(conv5)
    drop5 = Dropout(0.5)(conv5)

    # Decoder
    up6 = Conv2D(512, 2, activation='relu',
padding='same')(UpSampling2D(size=(2, 2))(drop5))
    merge6 = concatenate([drop4, up6], axis=3)
    conv6 = Conv2D(512, 3, activation='relu', padding='same')(merge6)
    conv6 = Conv2D(512, 3, activation='relu', padding='same')(conv6)

    up7 = Conv2D(256, 2, activation='relu',
padding='same')(UpSampling2D(size=(2, 2))(conv6))
    merge7 = concatenate([conv3, up7], axis=3)
    conv7 = Conv2D(256, 3, activation='relu', padding='same')(merge7)
    conv7 = Conv2D(256, 3, activation='relu', padding='same')(conv7)

    up8 = Conv2D(128, 2, activation='relu',
padding='same')(UpSampling2D(size=(2, 2))(conv7))
    merge8 = concatenate([conv2, up8], axis=3)
    conv8 = Conv2D(128, 3, activation='relu', padding='same')(merge8)
    conv8 = Conv2D(128, 3, activation='relu', padding='same')(conv8)

    up9 = Conv2D(64, 2, activation='relu',
padding='same')(UpSampling2D(size=(2, 2))(conv8))
    merge9 = concatenate([conv1, up9], axis=3)
    conv9 = Conv2D(64, 3, activation='relu', padding='same')(merge9)
    conv9 = Conv2D(64, 3, activation='relu', padding='same')(conv9)
    conv9 = Conv2D(2, 3, activation='relu', padding='same')(conv9)

    # Output layer
    outputs = Conv2D(1, 1, activation='sigmoid')(conv9)
```

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

# Example usage:
model = unet()
model.summary()
```

4.5 Model Deployment:Testing and Validation

Testing Data Selection:

- A separate portion of the dataset, distinct from the training and validation sets, is reserved for testing the deployed model.
- This dataset should represent the diversity of cases encountered in clinical practice to evaluate the model's generalization ability.

Data Preprocessing:

- Similar to the training phase, the testing data undergoes preprocessing steps such as normalization, resizing, and intensity adjustment to ensure consistency and compatibility with the deployed model.

Model Loading:

- The trained model, along with its learned parameters and architecture, is loaded into the deployment environment, ready for inference on unseen data.

Inference:

- The testing dataset is fed into the deployed model for inference, where each CT scan image is passed through the model to obtain predicted emphysema patterns.
- The model outputs probabilities or class labels indicating the presence and severity of emphysema for each input image.

Performance Evaluation:

- Various model evaluation metrics, as discussed earlier (accuracy, precision, recall, F1-score, specificity, AUC-ROC, AUC-PR, confusion matrix), are computed using the model's predictions and ground truth labels from the testing dataset.
- These metrics provide insights into the model's performance in accurately classifying emphysema patterns and its ability to generalize to unseen data.

Visual Inspection:

- In addition to quantitative metrics, visual inspection of model predictions alongside ground truth annotations helps assess the model's behavior and identify any discrepancies or areas for improvement.
- Clinicians may review sample predictions to validate the model's clinical relevance and interpretability.

Cross-Validation (Optional):

- If resources allow, cross-validation techniques such as k-fold cross-validation can be employed to further validate the model's performance and mitigate the impact of dataset variability.
- Cross-validation involves splitting the dataset into multiple subsets, training the model on different subsets, and evaluating its performance across iterations.

Deployment Readiness Assessment:

- Based on the performance metrics and visual inspection results, the model's readiness for deployment in clinical practice is assessed.
- Any necessary adjustments or fine-tuning may be made based on validation findings to optimize the model's performance and address any identified limitations.

DENSENET121 Model

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers, models
from sklearn.metrics import classification_report, f1_score
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.applications.densenet import preprocess_input
import os

# Load CSV file
csv_path = "/content/drive/My Drive/disease/image_cluster_labels.csv"
df = pd.read_csv(csv_path)

# Check for invalid file paths
invalid_paths = df[~df['Image Path'].apply(os.path.exists)]['Image Path']
if not invalid_paths.empty:
    print("Invalid file paths found:")
    print(invalid_paths)
    # Remove rows with invalid file paths
    df = df[~df['Image Path'].apply(lambda x: not os.path.exists(x))]

# Check if 'Image Path' column still exists in DataFrame
if 'Image Path' not in df.columns:
    print("Error: 'Image Path' column does not exist in DataFrame.")
else:
    # Encode class labels for 'Cluster Label'
    label_encoder = LabelEncoder()
    df['classes'] = label_encoder.fit_transform(df['Cluster Label'])
    num_classes = len(label_encoder.classes_)

    # Define image and batch size
    img_size = (224, 224)
    batch_size = 32

    # Create ImageDataGenerator for data augmentation and normalization
    datagen = ImageDataGenerator(
        rescale=1./255,
```

```
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        validation_split=0.2 # Split the data into training and
validation sets
    )

    # Create data generators for training, validation, and test
    train_generator = datagen.flow_from_dataframe(
        dataframe=df,
        x_col='Image Path',
        y_col='Cluster Label',
        target_size=img_size,
        batch_size=batch_size,
        subset='training',
        class_mode='raw',  # Use 'raw' for regression tasks
    )

    valid_generator = datagen.flow_from_dataframe(
        dataframe=df,
        x_col='Image Path',
        y_col='Cluster Label',
        target_size=img_size,
        batch_size=batch_size,
        subset='validation',
        class_mode='raw',  # Use 'raw' for regression tasks
    )

    # Build DenseNet121 model
    base_model = DenseNet121(weights='imagenet', include_top=False,
input_shape=(224, 224, 3))
    base_model.trainable = False

    model = models.Sequential()
    model.add(base_model)
    model.add(layers.Flatten())
    model.add(layers.Dense(512, activation='relu'))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(num_classes, activation='softmax'))

    # Compile the model
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001)
,
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
```

```
    # Train the model
    history = model.fit(
        train_generator,
        validation_data=valid_generator,
        epochs=5
    )

    # Evaluate the model on the test set
    test_datagen = ImageDataGenerator(rescale=1./255)
    test_generator = test_datagen.flow_from_dataframe(
        dataframe=df,
        x_col='Image Path',
        y_col='Cluster Label',
        target_size=img_size,
        batch_size=batch_size,
        class_mode='raw',
        shuffle=False
    )

    # Predictions
    y_pred = model.predict(test_generator)
    y_pred_cluster_classes = np.argmax(y_pred, axis=1)

    # Obtain true labels for evaluation
    true_labels = label_encoder.transform(test_generator.labels)

    # Calculate F1 score for cluster label
    f1_cluster = f1_score(true_labels, y_pred_cluster_classes,
average='weighted')
    print(f'Weighted F1 score for Cluster Label: {f1_cluster}')

    # Classification Report for cluster label
    class_labels_cluster = list(label_encoder.classes_)
    class_labels_cluster_str = [str(label) for label in
class_labels_cluster]
    print(classification_report(true_labels, y_pred_cluster_classes,
target_names=class_labels_cluster_str))

    # Calculate accuracy, precision, and recall
    report = classification_report(true_labels, y_pred_cluster_classes,
target_names=class_labels_cluster_str, output_dict=True)

    accuracy = report['accuracy']
    precision = report['macro avg']['precision']
```

```
    recall = report['macro avg']['recall']

    print(f'Accuracy: {accuracy}')
    print(f'Precision: {precision}')
    print(f'Recall: {recall}')
```

V16 Model

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers, models
from sklearn.metrics import classification_report, f1_score
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input

# Load CSV file
csv_path = "/content/drive/My Drive/disease/image_cluster_labels.csv"
df = pd.read_csv(csv_path)

# Encode class labels for 'Cluster Label'
label_encoder = LabelEncoder()
df['classes'] = label_encoder.fit_transform(df['Cluster Label'])
num_classes = len(label_encoder.classes_)

# Define image and batch size
img_size = (224, 224)
batch_size = 32

# Create ImageDataGenerator for data augmentation and normalization
datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2  # Split the data into training and validation
sets
)

# Create data generators for training, validation, and test
train_generator = datagen.flow_from_dataframe(
    dataframe=df,
    x_col='Image Path',
    y_col='Cluster Label',
```

```
    target_size=img_size,
    batch_size=batch_size,
    subset='training',
    class_mode='raw',  # Use 'raw' for regression tasks
)

valid_generator = datagen.flow_from_dataframe(
    dataframe=df,
    x_col='Image Path',
    y_col='Cluster Label',
    target_size=img_size,
    batch_size=batch_size,
    subset='validation',
    class_mode='raw',  # Use 'raw' for regression tasks
)

# Build VGG16 model
base_model = VGG16(weights='imagenet', include_top=False,
input_shape=(224, 224, 3))
base_model.trainable = False

model = models.Sequential()
model.add(base_model)
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model.fit(
    train_generator,
    validation_data=valid_generator,
    epochs=5
)

# Evaluate the model on the test set
test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow_from_dataframe(
    dataframe=df,
    x_col='Image Path',
```

```
    y_col='Cluster Label',
    target_size=img_size,
    batch_size=batch_size,
    class_mode='raw',
    shuffle=False
)

# Predictions
y_pred = model.predict(test_generator)
y_pred_cluster_classes = np.argmax(y_pred, axis=1)

# Obtain true labels for evaluation
true_labels = label_encoder.transform(test_generator.labels)

# Calculate F1 score for cluster label
f1_cluster = f1_score(true_labels, y_pred_cluster_classes,
average='weighted')
print(f'Weighted F1 score for Cluster Label: {f1_cluster}')

# Classification Report for cluster label
class_labels_cluster = list(label_encoder.classes_)
class_labels_cluster_str = [str(label) for label in class_labels_cluster]
print(classification_report(true_labels, y_pred_cluster_classes,
target_names=class_labels_cluster_str))

# Calculate accuracy, precision, and recall
report = classification_report(true_labels, y_pred_cluster_classes,
target_names=class_labels_cluster_str, output_dict=True)

accuracy = report['accuracy']
precision = report['macro avg']['precision']
recall = report['macro avg']['recall']

print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
```

InceptionV3 Model

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers, models
from sklearn.metrics import classification_report, f1_score
from sklearn.preprocessing import LabelEncoder
```

```
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.applications.inception_v3 import preprocess_input

# Load CSV file
csv_path = "/content/drive/My Drive/disease/image_cluster_labels.csv"
df = pd.read_csv(csv_path)

# Encode class labels for 'Cluster Label'
label_encoder = LabelEncoder()
df['classes'] = label_encoder.fit_transform(df['Cluster Label'])
num_classes = len(label_encoder.classes_)

# Define image and batch size
img_size = (299, 299)  # InceptionV3 requires input shape (299, 299)
batch_size = 32

# Create ImageDataGenerator for data augmentation and normalization
datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    validation_split=0.2  # Split the data into training and validation
sets
)

# Create data generators for training, validation, and test
train_generator = datagen.flow_from_dataframe(
    dataframe=df,
    x_col='Image Path',
    y_col='Cluster Label',
    target_size=img_size,
    batch_size=batch_size,
    subset='training',
    class_mode='raw',  # Use 'raw' for regression tasks
)

valid_generator = datagen.flow_from_dataframe(
    dataframe=df,
    x_col='Image Path',
    y_col='Cluster Label',
```

```
    target_size=img_size,
    batch_size=batch_size,
    subset='validation',
    class_mode='raw',  # Use 'raw' for regression tasks
)

# Build InceptionV3 model
base_model = InceptionV3(weights='imagenet', include_top=False,
input_shape=(299, 299, 3))
base_model.trainable = True

fine_tune_at = 200  # Fine-tune from layer 200 onwards
for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False

model = models.Sequential()
model.add(base_model)
model.add(layers.GlobalAveragePooling2D())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model.fit(
    train_generator,
    validation_data=valid_generator,
    epochs=5
)

# Evaluate the model on the test set
test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow_from_dataframe(
    dataframe=df,
    x_col='Image Path',
    y_col='Cluster Label',
    target_size=img_size,
    batch_size=batch_size,
    class_mode='raw',
    shuffle=False
)
```

```
# Predictions
y_pred = model.predict(test_generator)
y_pred_cluster_classes = np.argmax(y_pred, axis=1)

# Obtain true labels for evaluation
true_labels = label_encoder.transform(test_generator.labels)

# Calculate F1 score for cluster label
f1_cluster = f1_score(true_labels, y_pred_cluster_classes,
average='weighted')
print(f'Weighted F1 score for Cluster Label: {f1_cluster}')

# Classification Report for cluster label
class_labels_cluster = list(label_encoder.classes_)
class_labels_cluster_str = [str(label) for label in class_labels_cluster]
print(classification_report(true_labels, y_pred_cluster_classes,
target_names=class_labels_cluster_str))

# Calculate accuracy, precision, and recall
report = classification_report(true_labels, y_pred_cluster_classes,
target_names=class_labels_cluster_str, output_dict=True)

accuracy = report['accuracy']
precision = report['macro avg']['precision']
recall = report['macro avg']['recall']

print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
```

4.5.1 Testing and Validation

Detection of Emphysema Images

```
import pandas as pd
import numpy as np
import cv2
from keras.preprocessing import image
import matplotlib.pyplot as plt

# Function to display images with class labels, category labels, and
emphysema information
def display_images_with_labels(images, class_labels, category_labels,
emphysema labels):
```

```
    num_images = len(images)
    num_cols = 4
    num_rows = -(-num_images // num_cols) # Ensure integer division
rounds up

    plt.figure(figsize=(15, 6 * num_rows))
    for i in range(num_images):
        plt.subplot(num_rows, num_cols, i + 1)
        plt.imshow(images[i])
        title = f'Class: {class_labels[i]}\nCategory:
{category_labels[i]}'
        if emphysema_labels[i]:
            title += '\nEmphysema: Yes'
        else:
            title += '\nEmphysema: No'
        plt.title(title)
        plt.axis('off')
    plt.show()

# Step 1: Read the CSV file
csv_file = "/content/drive/My Drive/disease/image_cluster_labels.csv"
df = pd.read_csv(csv_file)

# Step 2: Extract image file paths and labels
image_paths = df['Image Path'].tolist()
cluster_labels = df['Cluster Label'].tolist()
category_labels = df['Category'].tolist()

# Step 3: Load and preprocess images with resizing
sample_images = []
skipped_images = []
num_images_to_display = 300
for i, path in enumerate(image_paths):
    if i >= num_images_to_display:
        break
    try:
        img = cv2.imread(path) # Read image using OpenCV
        if img is None:
            raise Exception("Image not loaded")
        img = cv2.resize(img, (224, 224)) # Resize image to (224, 224)
        img = img / 255.0 # Normalize pixel values
        sample_images.append(img)
    except Exception as e:
        print(f"Error loading image: {path} - {e}")
        skipped_images.append(path)
```

```
# Print the skipped image paths
print("Skipped images:", skipped_images)

# Step 4: Predict emphysema based on cluster labels
sample_emphysema_labels = []
for label in cluster_labels:
    # Assuming 'label' indicates emphysema if it's not equal to 0
    emphysema = label != 0
    sample_emphysema_labels.append(emphysema)

# Step 5: Display the images along with class labels, category labels, and
emphysema information
display_images_with_labels(sample_images,
cluster_labels[:num_images_to_display],
category_labels[:num_images_to_display], sample_emphysema_labels)
```

4.6 Web GUI's Development

- **Results of Fronted**

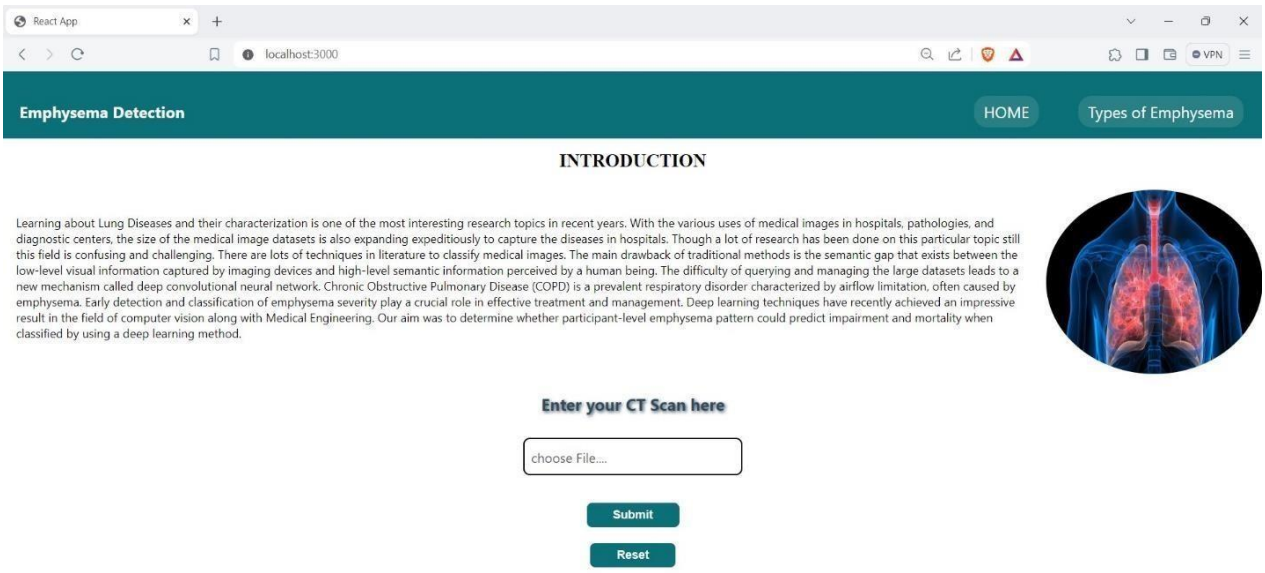


Fig 4.6.1 Front page

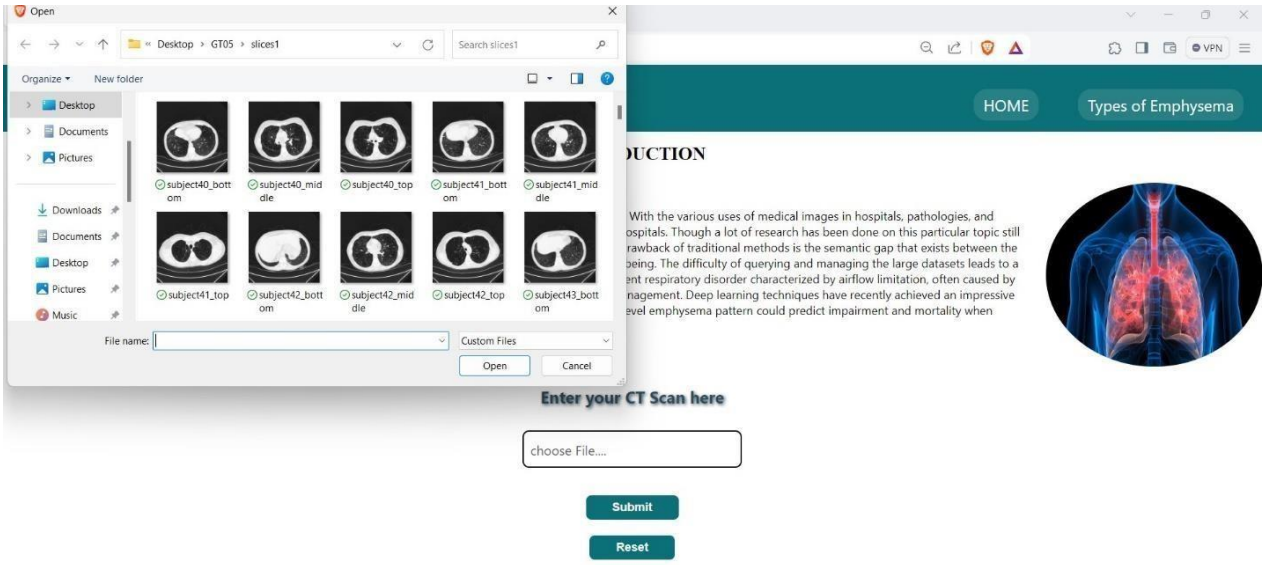


Fig 4.6.2 Upload CT Scan

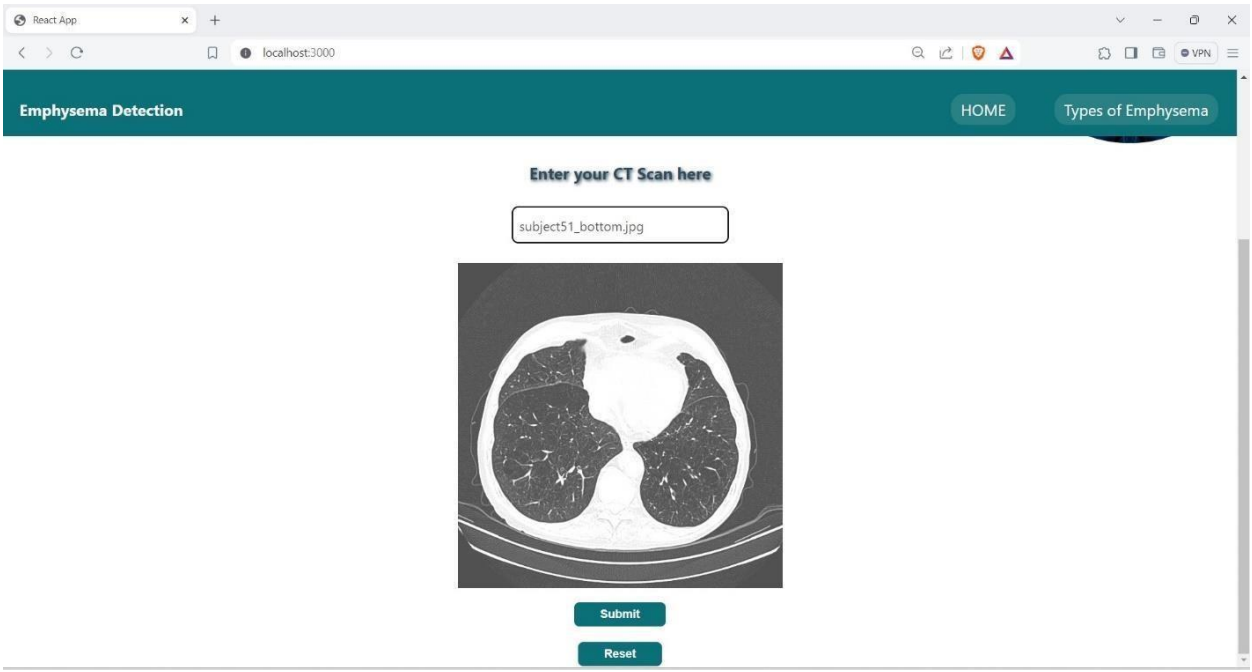


Fig 4.6.3 User upload their CT Scan

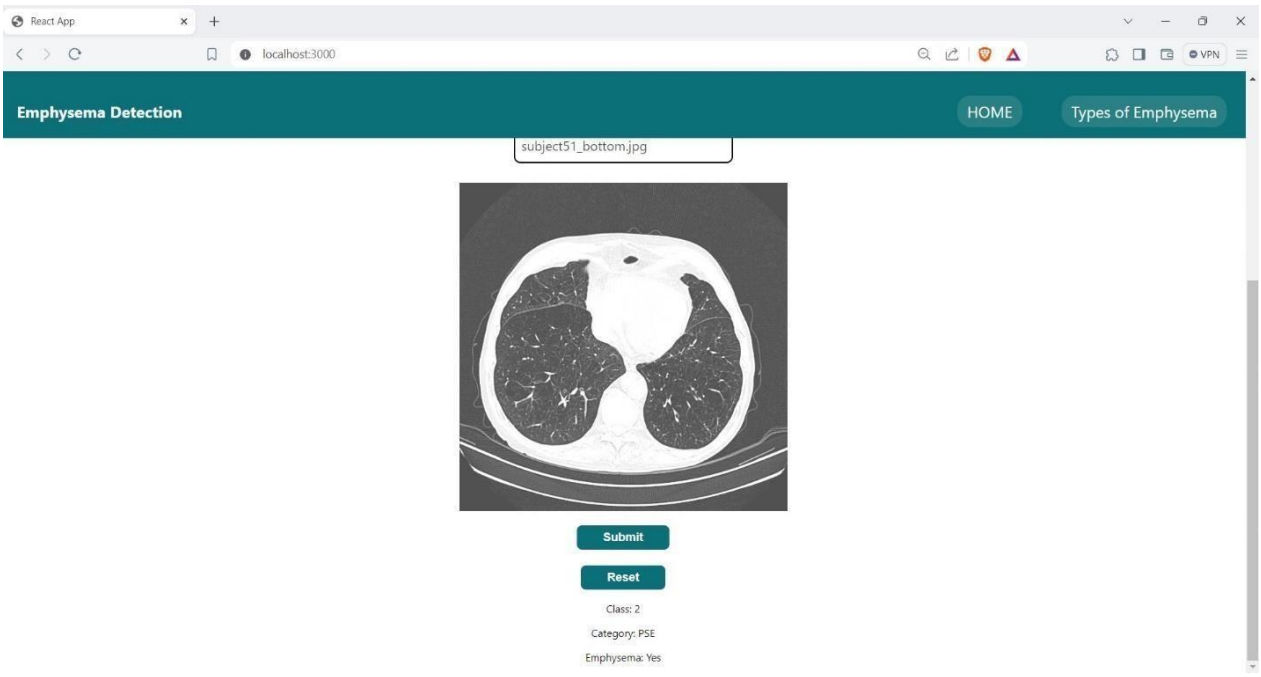


Fig 4.6.4 Output

- **Results of Backend**

```
Model loaded successfully.
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment.
 Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.1.4:5000
Press CTRL+C to quit
```

Fig 4.6.5 API

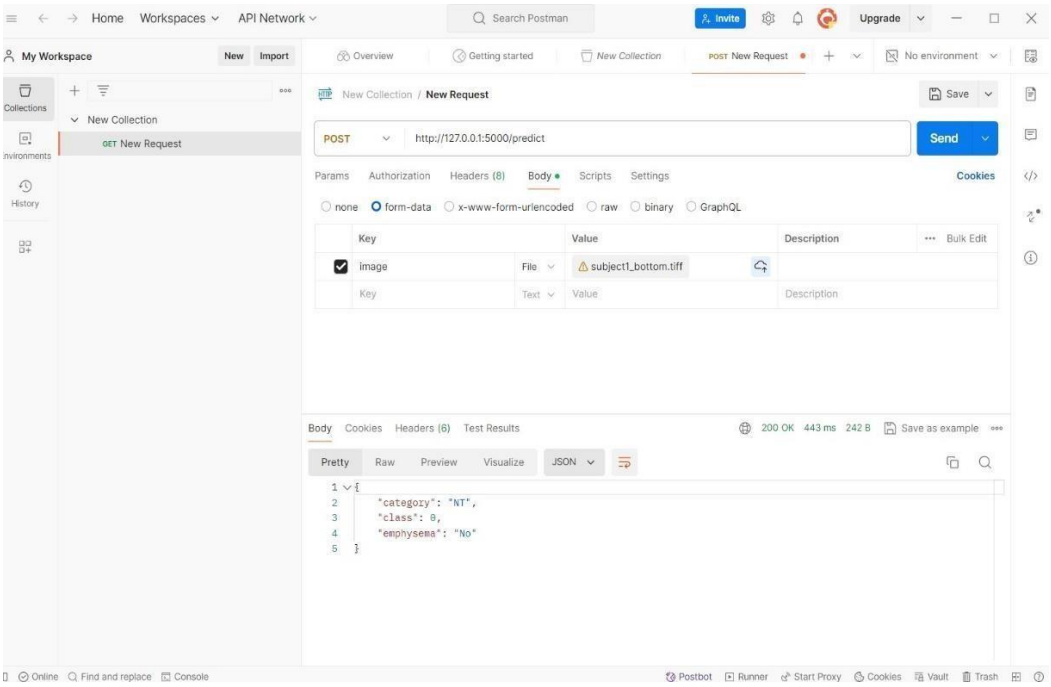


Fig 4.6.6 Backend API

4.7 Results

```
Weighted F1 score for Cluster Label: 0.8349023595760169
              precision    recall  f1-score   support

           0       0.75      0.94      0.84       982
           1       1.00      0.98      0.99       115
           2       0.96      0.57      0.72       424
           3       0.92      0.84      0.88       744

    accuracy                           0.84      2265
   macro avg       0.91      0.83      0.86      2265
weighted avg       0.86      0.84      0.83      2265

Accuracy: 0.8388520971302428
Precision: 0.9089763553401067
Recall: 0.8323980178501067
```

Fig 4.7.1 Classification report of Densenet121 Model

Weighted F1 score for Cluster Label: 0.845413533555757

	precision	recall	f1-score	support
0	0.78	0.91	0.84	982
1	1.00	0.98	0.99	115
2	0.95	0.62	0.75	424
3	0.90	0.87	0.88	744
accuracy			0.85	2265
macro avg	0.91	0.85	0.87	2265
weighted avg	0.86	0.85	0.85	2265

Accuracy: 0.8481236203090508
Precision: 0.906379068377802
Recall: 0.8464903127554819

Fig 4.7.2 Classification report of V16 Model

Weighted F1 score for Cluster Label: 0.7684357518498443

	precision	recall	f1-score	support
0	0.78	0.65	0.71	982
1	1.00	0.97	0.99	115
2	0.57	0.94	0.71	424
3	0.91	0.78	0.84	744
accuracy			0.77	2265
macro avg	0.82	0.84	0.81	2265
weighted avg	0.80	0.77	0.77	2265

Accuracy: 0.765121412803532
Precision: 0.8170769027555425
Recall: 0.8369906841229403

Fig 4.7.3 Classification report of InceptionV3 Model

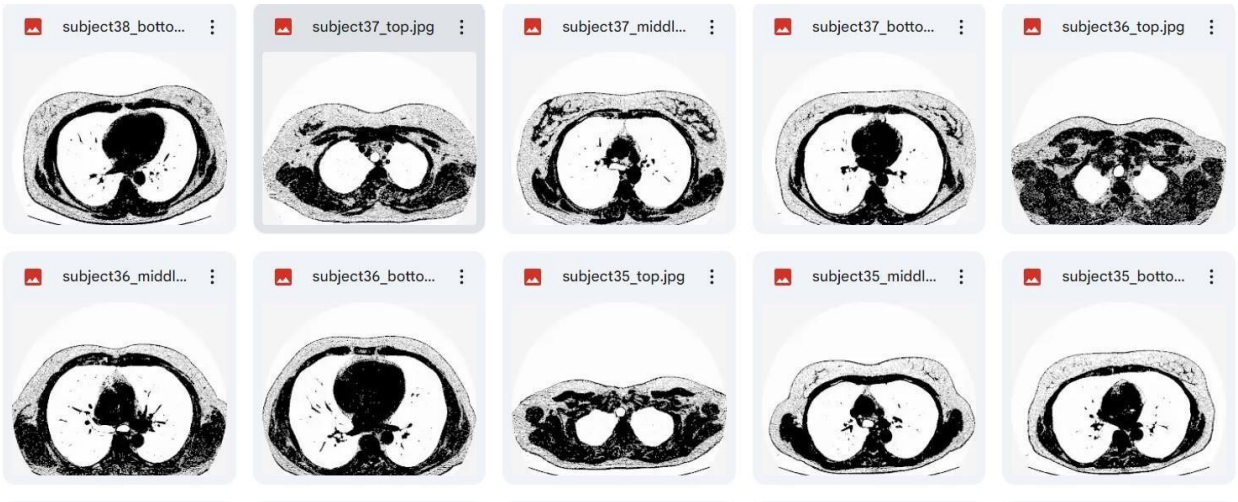


Fig 4.7.4 Image Dataset

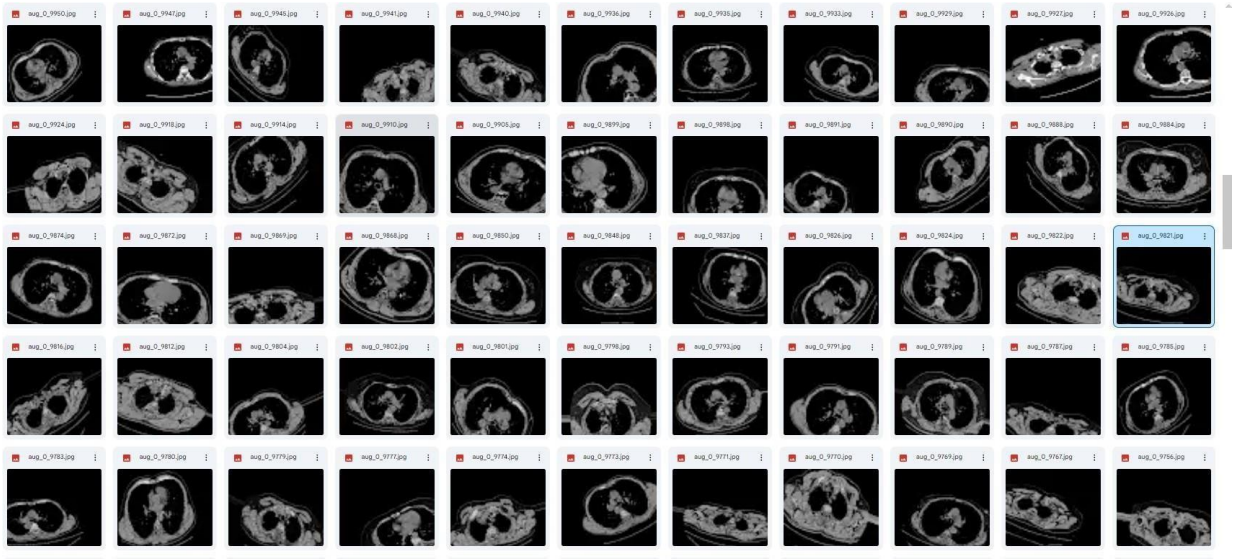
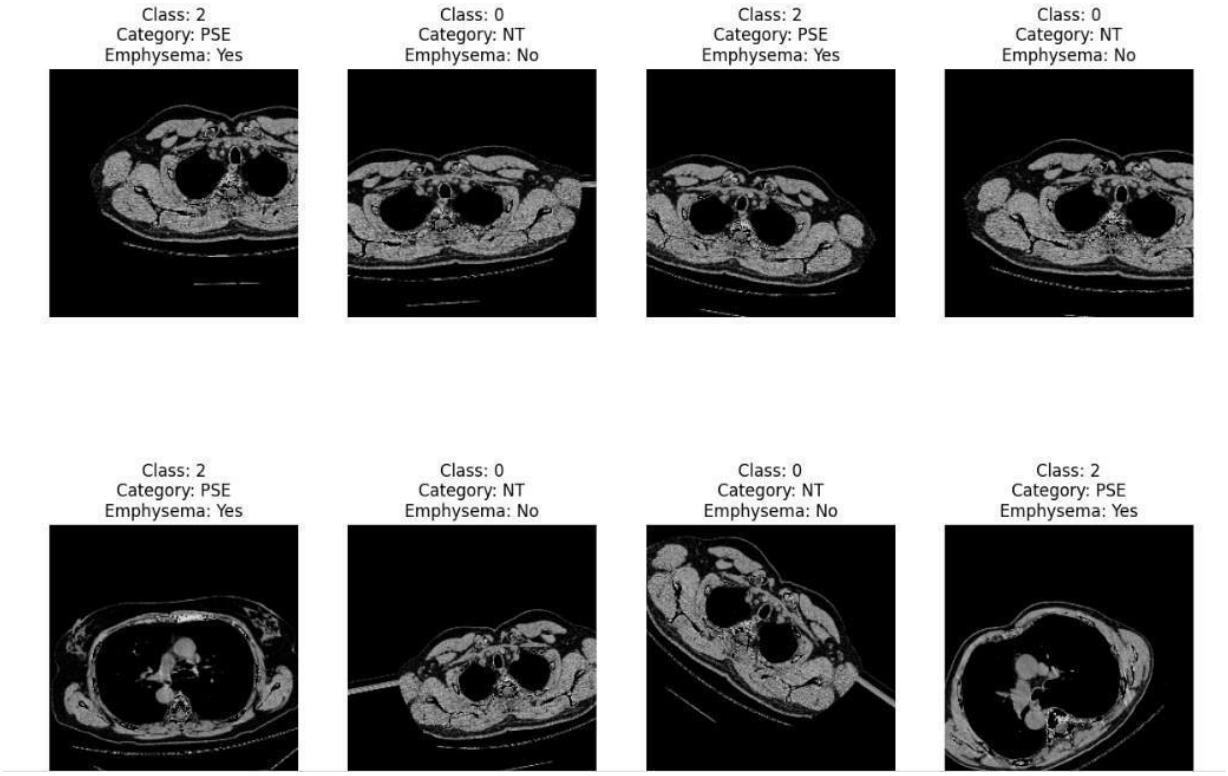
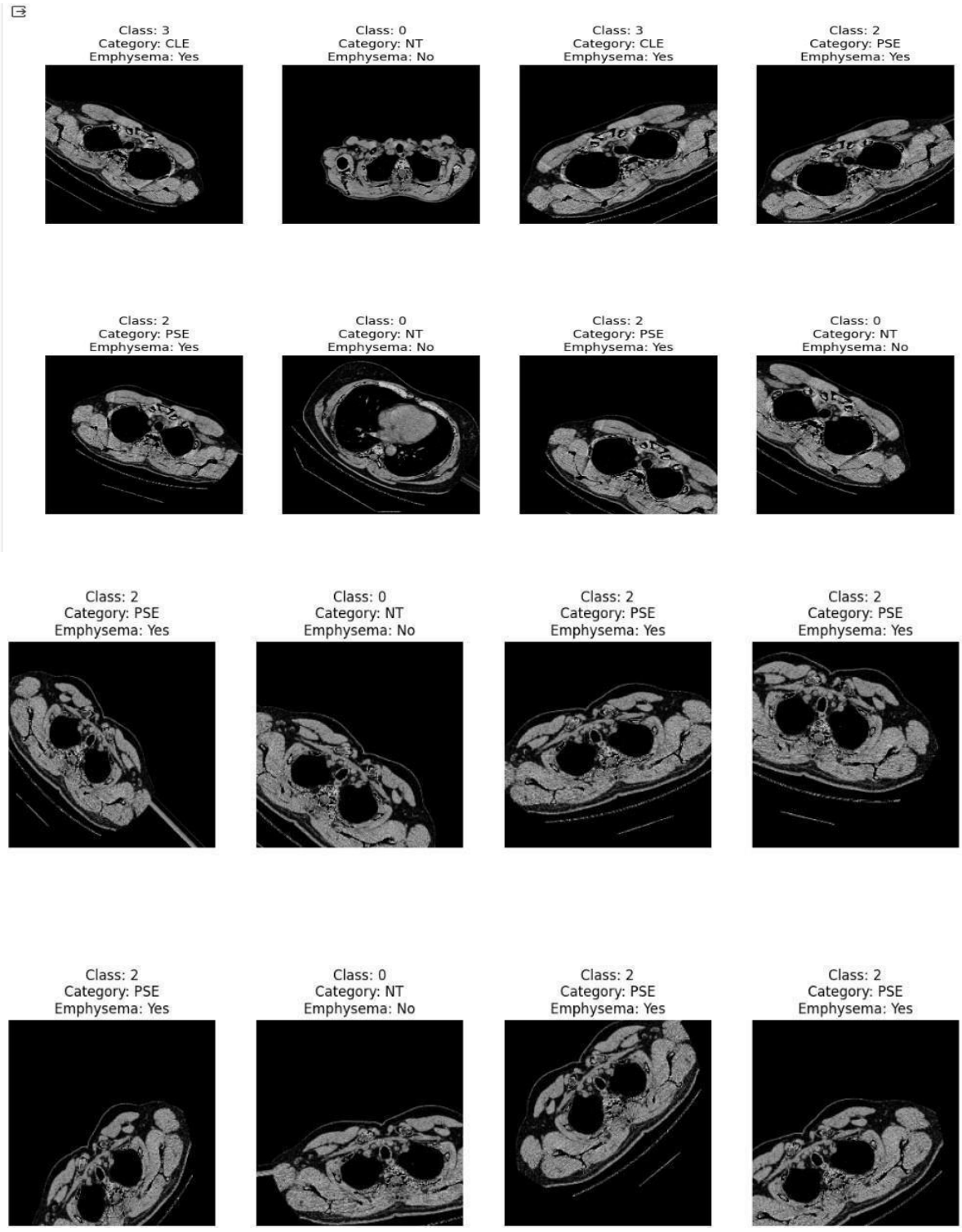


Fig 4.7.5 Augmented Images




Class: 3
Category: CLE
Emphysema: Yes
Class: 0
Category: NT
Emphysema: No
Class: 3
Category: CLE
Emphysema: Yes
Class: 2
Category: PSE
Emphysema: Yes
Class: 2
Category: PSE
Emphysema: Yes
Class: 0
Category: NT
Emphysema: No
Class: 2
Category: PSE
Emphysema: Yes
Class: 0
Category: NT
Emphysema: No
Class: 2
Category: PSE
Emphysema: Yes
Class: 0
Category: NT
Emphysema: No
Class: 2
Category: PSE
Emphysema: Yes
Class: 2
Category: PSE
Emphysema: Yes
Class: 2
Category: PSE
Emphysema: Yes
Class: 2
Category: PSE
Emphysema: Yes
Class: 0
Category: NT
Emphysema: No
Class: 2
Category: PSE
Emphysema: Yes
Class: 2
Category: PSE
Emphysema: Yes
Class: 0
Category: NT
Emphysema: No
Class: 2
Category: PSE
Emphysema: Yes
Class: 2
Category: PSE
Emphysema: Yes

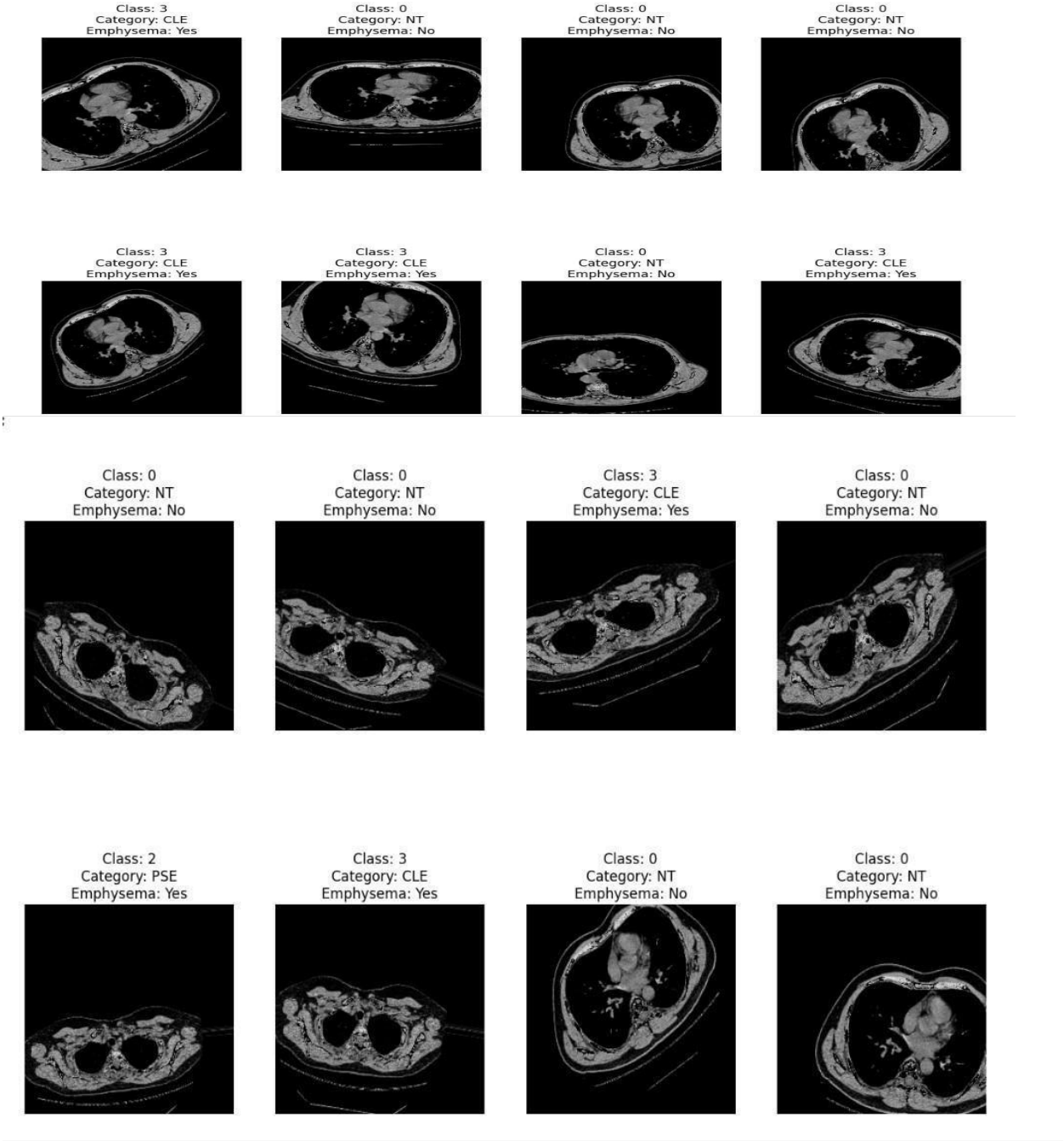


Fig 4.7.6 Emphysema Detection Images

CHAPTER 5

CONCLUSION

5.1 Project Conclusion

In conclusion, the emphysema detection project, leveraging deep learning techniques on CT scans, marks a significant advancement in respiratory health diagnostics. Through the fusion of deep learning algorithms and high-resolution CT imaging, the project has showcased remarkable efficacy in precisely identifying and characterizing emphysema-related anomalies within lung scans. This innovative approach not only aids in the early detection of emphysema but also offers insights into disease progression and severity. By harnessing the entirety of information provided by CT scans, including subtle structural changes and spatial distributions, the developed model demonstrates its potential to revolutionize clinical practice by providing accurate and timely diagnoses. Moving forward, continued refinement and validation of the deep learning models across diverse patient cohorts will be essential to ensure robust performance and widespread applicability in real-world healthcare settings. Ultimately, this project underscores the transformative impact of deep learning in advancing medical imaging and enhancing the management of respiratory conditions like emphysema.

5.2 Future Scope

In the realm of medical image analysis, particularly concerning lung diseases like Chronic Obstructive Pulmonary Disease (COPD) and emphysema, there exists a burgeoning interest in leveraging deep learning techniques for classification and prediction tasks. While traditional methods have made strides, they often grapple with the semantic gap between low-level visual information from imaging devices and high-level semantic understanding needed for accurate diagnosis. Deep convolutional neural networks (CNNs) have emerged as a promising solution to bridge this gap, showcasing remarkable results in computer vision and medical engineering. Looking ahead, future enhancements could encompass a multifaceted approach. Firstly, integrating multi-modal data sources such as CT scans, X-rays, and clinical records could provide a more holistic view of the disease. Transfer learning techniques may aid in adapting pre-trained models to specific lung disease classification tasks, potentially reducing the need for extensive annotated datasets. Moreover, incorporating methods for uncertainty estimation, explainability, and interpretability is crucial for building trust in these models among healthcare professionals. Longitudinal analysis could offer insights into disease progression over time, while advanced data augmentation and synthesis techniques could enhance model generalization. Additionally, exploring semi-supervised and weakly supervised learning approaches may leverage large amounts of unlabeled or weakly labeled data, making the development process more cost-effective. Ultimately, collaboration with medical experts and rigorous validation through clinical trials are paramount to ensuring the clinical relevance and efficacy of these deep learning models in improving patient outcomes and healthcare delivery.

REFERENCES

1. T. Manikandan, S. Maheswari, "Automated classification of emphysema using data augmentation and effective pixel location estimation with multi-scale residual network", *Neural Computing and Applications*, 2022.
2. S. Parui, D. Parbat, M. Chakraborty, "A Deep Learning Paradigm for Computer Aided Diagnosis of Emphysema from Lung HRCT Images", *International Conference on Computing in Engineering & Technology (ICCET)*, 2022.
3. S. Mondala, A. K. Sadhub, P. K. Duttac, "Automated diagnosis of pulmonary emphysema using multi-objective binary thresholding and hybrid classification", *Biomedical Signal Processing and Control 69*, 2021
4. C. M. Bhuma, "An Ensemble Approach To Emphysema Classification", *IEEE International Conference on Industrial and Information Systems (ICIIS)*, 2020.
5. S. Parui, D. Parbat, M. Chakraborty, "A Deep Learning Paradigm for Computer Aided Diagnosis of Emphysema from Lung HRCT Images", *International Conference on Computing in Engineering & Technology (ICCET)*, 2022.
6. A. Sriram, S. Kalra, H. R. Tizhoosh, "Projectron - A Shallow and Interpretable Network for Classifying Medical Images", *CoRR abs/1904.00740*, 2019.
7. S. J. Narayanan, R. Soundrapandiyan, B. Perumal, C. J. Baby, "Emphysema Medical Image Classification Using Fuzzy Decision Tree with Fuzzy Particle Swarm Optimization Clustering", *International Conference on SCI*, 2018.
8. E. Tuba, I. Strumberger, N. Bacanin, M. Tuba, "Analysis of local binary pattern for emphysema classification in lung CT image", *IEEE International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, 2019.
9. H. Li, R. Mukundan, "Robust Texture Features For Emphysema Classification In CT Images", *European Signal Processing Conference (EUSIPCO)*, 2020.
10. P. K. R. Yelampalli, J. Nayak, V. H. Gaidhane, "A novel binary feature descriptor to discriminate normal and abnormal chest CT images using dissimilarity measures", *Pattern Analysis and Applications 22(4): 1517-1526*, 2019.