A black background with white text

Description automatically generated with low confidence

MSc Data Science Project

7PAM2002-0901-2024

Department of Physics, Astronomy and Mathematics

**Data Science FINAL PROJECT REPORT**

**Project Title:**

Lung Disease Prediction Using Machine Learning Algorithms

**Student Name and SRN:**

Balaram Ravella and 21064285

Supervisor: Michael Kuhn

Date Submitted: 04/01/2025

Word Count:

GitHub address:

DECLARATION STATEMENT

This report is submitted in partial fulfilment of the requirement for the degree of Master of Science in Data Science at the University of Hertfordshire.

I have read the guidance to students on academic integrity, misconduct and plagiarism information at [Assessment Offences and Academic Misconduct](https://www.herts.ac.uk/__data/assets/pdf_file/0007/237625/AS14-Apx3-Academic-Misconduct-v17.0.pdf) and understand the University process of dealing with suspected cases of academic misconduct and the possible penalties, which could include failing the project module or course.

I certify that the work submitted is my own and that any material derived or quoted from published or unpublished work of other persons has been duly acknowledged. (Ref. UPR AS/C/6.1, section 7 and UPR AS/C/5, section 3.6).

I have not used chatGPT, or any other generative AI tool, to write the reportor code (other than where declared or referenced).

I did not use human participants or undertake a survey in my MSc Project.

I hereby give permission for the report to be made available on module websites provided the source is acknowledged.

Student SRN number: 21064285

Student Name printed: Balaram Ravella

Student signature: Balaram

UNIVERSITY OF HERTFORDSHIRE

SCHOOL OF PHYSICS, ENGINEERING AND COMPUTER SCIENCE

# Abstract:

This research project explores the application of machine learning algorithms for the prediction of lung diseases, focusing on Tuberculosis (TB) and Pneumonia. With the increasing burden of respiratory diseases globally, early and accurate diagnosis is crucial for effective treatment and management. The objective of this study is to develop a robust machine learning-based model that can assist in the rapid diagnosis of these diseases by analyzing chest X-ray images. To achieve this, two publicly available datasets containing X-ray images of patients diagnosed with TB and Pneumonia were used for training and evaluation.

In this work, Convolutional Neural Networks (CNNs) were employed to automatically extract features from the X-ray images and classify them into distinct categories, i.e., TB, Pneumonia, or normal. Several architectures of CNNs were explored, including VGG16, DenseNet121, and InceptionResNetV2. Various preprocessing techniques, such as image normalization and data augmentation, were applied to improve model generalization and accuracy. Hyperparameter tuning was conducted to identify the best configuration for the CNN models, with performance evaluated using metrics such as accuracy, precision, recall, and F1-score.

The results indicate that while all models demonstrated significant capabilities in lung disease classification, the VGG16 architecture consistently outperformed the others. The multi-disease classifier built using VGG16 achieved a test accuracy of approximately 89%, with training accuracy reaching 98% over 75 epochs. The validation accuracy stabilized around 85%, demonstrating strong generalization to unseen data. VGG16’s relatively simple yet deep structure allowed for effective feature extraction and classification, proving particularly robust against issues like data imbalance and overfitting. Further analysis showed a precision of 89.2%, recall of 89.5%, and an F1 score of 89%, highlighting its effectiveness in minimizing false positives and detecting true cases. The model’s ability to distinguish between different lung conditions with high precision highlights its potential for practical deployment in clinical environments.

This project underscores the efficacy of machine learning in supporting healthcare professionals in diagnosing lung diseases, especially in settings where access to radiologists may be limited. Automating the detection process can reduce the diagnostic burden, enabling faster patient assessment and treatment. The findings from this study contribute to ongoing research in AI-driven medical diagnostics, reinforcing the value of VGG16 as a highly effective model for lung disease prediction and laying the groundwork for future advancements in this field.

Contents

[Abstract: 3](#_Toc186924617)

[Introduction: 5](#_Toc186924618)

[Literature Review: 7](#_Toc186924619)

[Methodology: 10](#_Toc186924620)

[Dataset overview: 10](#_Toc186924621)

[1. Pneumonia Dataset 10](#_Toc186924622)

[2. Tuberculosis (TB) Dataset 10](#_Toc186924623)

[3. Challenges and Ethical Considerations 11](#_Toc186924624)

[Data Preprocessing 11](#_Toc186924625)

[Architectures used: 12](#_Toc186924626)

[Convolutional Neural Network (CNN) Architecture 12](#_Toc186924627)

[VGG16 Architecture 14](#_Toc186924628)

[InceptionResNetV2 Architecture: 15](#_Toc186924629)

[Chexnet-DenseNet121 Architecture: 17](#_Toc186924630)

[Results: 17](#_Toc186924631)

[Difficulties Encountered During the Project 23](#_Toc186924632)

[Conclusion: 23](#_Toc186924633)

[Future work: 24](#_Toc186924634)

[References: 25](#_Toc186924635)

[Appendix: 26](#_Toc186924636)

# Introduction:

Lung infections, particularly pneumonia and tuberculosis (TB), remain significant global health challenges, leading to substantial morbidity and mortality. In 2023, tuberculosis was responsible for approximately 1.25 million deaths worldwide, underscoring its status as a leading infectious killer. (World, 2024)

Pneumonia remains a significant health concern, particularly among vulnerable populations such as the elderly, young children, and individuals with pre-existing health conditions. In the United Kingdom, pneumonia affects approximately 220,000 individuals annually and is responsible for over 25,000 deaths each year. (Chalmers, 2024).

Notably, the UK has the third highest pneumonia death rate in Europe. (www.asthmaandlung.org.uk, 2022)

Globally, pneumonia is the leading infectious cause of death among children under five, accounting for 14% of all deaths in this age group, with over 740,000 fatalities in 2019. (World Health Organization, 2022)

The disease disproportionately affects children in southern Asia and sub-Saharan Africa, where access to timely and adequate healthcare is often limited.

These statistics underscore the critical importance of prompt diagnosis and treatment of pneumonia to prevent severe outcomes and fatalities. Delays in obtaining medical consultations and diagnostic results can lead to disease progression, increasing the risk of complications and death. Implementing rapid diagnostic tools and improving access to healthcare services are essential steps in reducing the global burden of pneumonia.

Traditional diagnostic processes for lung infections often involve obtaining chest X-ray images, which are then analyzed by radiologists or physicians. This sequence can be time-consuming due to factors like scheduling delays, limited availability of specialists, and processing times. Such delays are critical, as both pneumonia and TB can progress rapidly, and timely intervention is crucial to prevent severe complications or death.

To address these challenges, this project focuses on developing machine learning models capable of swiftly and accurately detecting pneumonia and TB from chest X-ray images. By leveraging advanced algorithms, these models can analyze medical images and provide diagnostic results within seconds, significantly reducing the time between imaging and diagnosis. Implementing such models in clinical settings could facilitate immediate preliminary assessments, enabling healthcare providers to prioritize urgent cases and initiate prompt treatment.

The integration of machine learning diagnostics into healthcare systems holds the potential to revolutionize patient care by providing rapid, reliable, and accessible diagnostic services. This approach not only alleviates the burden on healthcare professionals but also ensures that patients receive timely interventions, thereby improving outcomes and reducing mortality rates associated with lung infections. As technology continues to advance, the future of healthcare is likely to see an increased reliance on such models, paving the way for more efficient and effective disease management.

Advantages of Implementing Machine Learning Models for Lung Disease Diagnosis

Implementing machine learning models, such as those developed in this project, offers numerous advantages in diagnosing lung diseases like pneumonia and tuberculosis. These benefits address critical challenges in traditional diagnostic workflows and improve the overall efficiency and effectiveness of healthcare systems. The key advantages include:

1. Rapid Diagnosis

Machine learning models can analyze chest X-ray images and provide diagnostic results within seconds, significantly reducing the time taken compared to traditional methods. This rapid turnaround ensures timely medical interventions, which is crucial in preventing the progression of diseases and reducing mortality rates.

2. Enhanced Accessibility

In regions with limited access to medical specialists, machine learning models can serve as an intermediary diagnostic tool. By providing preliminary assessments, these models empower healthcare facilities in underserved areas to identify severe cases and prioritize them for further examination.

3. Alleviating Healthcare System Burden

The growing demand for radiological services often overwhelms healthcare systems. By automating the initial diagnostic process, machine learning models alleviate the burden on radiologists and physicians, allowing them to focus on complex cases and patient care.

## Research Question:

* What are the most effective machine learning models for lung disease prediction from medical imaging, and how do they compare in terms of performance and reliability?

**AIM:**

The aim of this project is to develop machine learning models using Convolutional Neural Networks (CNNs) for the accurate and rapid diagnosis of lung diseases, specifically pneumonia and tuberculosis, based on chest X-ray images. By leveraging the power of deep learning, the project seeks to address delays in traditional diagnostic processes, providing an efficient and reliable tool for preliminary disease detection. This approach aims to enhance timely medical interventions, reduce the burden on healthcare professionals, and improve patient outcomes, particularly in resource-limited settings.

**Objectives**

* **Addressing Data Imbalance in the Dataset**: To implement techniques such as data augmentation and oversampling to mitigate the issue of class imbalance in the datasets, ensuring that the machine learning models are trained on balanced data for improved accuracy and fairness in predictions.
* **Integrating Multiple Datasets into a Unified Model**: To effectively combine two different datasets—one focused on pneumonia and the other on tuberculosis—into a single predictive model capable of diagnosing both diseases accurately, leveraging shared patterns and features in chest X-ray images.

# Literature Review:

The integration of Convolutional Neural Networks (CNNs), particularly the VGG family, into medical image analysis has significantly advanced the field of automated disease diagnosis. Several studies have demonstrated the effectiveness of VGG architectures, particularly VGG16, in detecting pneumonia from chest X-ray images. Below is an in-depth discussion of four key papers for Pneumonia and two key papers for Tuberculosis that explored these applications.

**Paper 1**by (Zhang et al., 2021) published in ***Applied Sciences***, researchers investigated the application of the VGG16 architecture for pneumonia detection from X-ray images. The study leveraged the pre-trained weights of VGG16 on ImageNet for transfer learning to generalize features effectively for medical imaging tasks. Data augmentation techniques, including flipping, zooming, and,rotating were used to balance the dataset

The architecture was fine-tuned by modifying its fully connected layers to adapt to the binary classification of pneumonia versus healthy lungs. By optimizing hyperparameters such as learning rate and batch size, the model achieved remarkable results, demonstrating high accuracy, precision, and recall. The study concluded that VGG16’s straightforward structure and its ability to capture intricate features made it a robust choice for pneumonia detection. Furthermore, the research emphasized its potential utility in real-world healthcare scenarios, where quick and correct diagnosis is paramount.

**Paper 2** by (Li, 2024) presents a novel approach that integrates attention mechanisms with the ResNet architecture to enhance diagnostic performance. The researchers utilized an attention module that focuses on the most critical features within the X-ray images, allowing the model to prioritize regions indicative of pneumonia. This mechanism improved the interpretability and sensitivity of the model, ensuring that it focused on pathological areas rather than irrelevant features.

To address the common challenge of class imbalance, the authors employed a focal loss function. This helps to put great weights for imbalanced class, effectively balancing the training process and reducing bias. The model was evaluated on a benchmark chest X-ray dataset with an accuracy of 98% and outperforming traditional models, including standard ResNet-50. The study highlights how attention mechanisms, when paired with ResNet architectures, can significantly improve the detection o f pneumonia by emphasizing pathological features while maintaining computational efficiency.

**Paper 3** by (Shafi, Hasan and Das, 2022) published in *IEEE Xplore*, researchers proposed a novel approach that help to mix both the abilities of VGG16 and Xception architectures. The study utilized transfer learning to extract features from both pre-trained models and fused these features to create a comprehensive representation of the input chest X-ray images.

The researchers focused on enhancing image quality through advanced preprocessing techniques and fine-tuned both architectures on a balanced pneumonia dataset. The fusion model outperformed standalone architectures, achieving a classification accuracy of over 95%. This approach demonstrated the potential of combining CNN models to significantly improve diagnostic accuracy. The study emphasized the benefits of feature fusion in medical imaging, providing a pathway for developing more solutions.

**Paper 4** by (Rajpurkar et al., 2017) highlights the use of DenseNet121 to detect pneumonia from chest X-rays, achieving performance that surpasses human radiologists. This model was trained with ChestX-ray14 dataset, containing over 100,000 images labeled with 14 different thoracic pathologies. CheXNet fine-tuned the DenseNet121 model by freezing the initial layers, which preserved low-level feature extraction, and modifying the top layers to focus on binary classification between pneumonia and non-pneumonia cases.

DenseNet121, recognized for its tightly interconnected convolutional layers, encourages the reuse of features by linking each layer to all previous layers in a sequential manner. This design enhances gradient flow and substantially decreases the total number of parameters, resulting in a model that is both computationally efficient and robust. Consequently, CheXNet demonstrated improved sensitivity and specificity, surpassing radiologists in performance, establishing itself as a reliable tool for clinical use.

To mitigate overfitting, they’ve employed data augmentation techniques such as horizontal flipping, rotation, and intensity normalization. These steps increased the model’s robustness by simulating a variety of real-world imaging conditions. Furthermore, the study introduced Class Activation Maps (CAM) to visualize which areas of the image has most of the predictions. This feature added a layer of transparency to the model’s decision-making process, allowing radiologists to verify the highlighted areas against their own expertise.

This project adopts a similar methodology by fine-tuning DenseNet121 for multi-class classification to distinguish between Normal, Bacterial Pneumonia, and Viral Pneumonia cases. In addition to freezing early layers, dropout and global average pooling layers are incorporated to reduce overfitting and enhance generalization. Adam optimization with a low learning rate ensures stable convergence during training, balancing speed and accuracy.

| **Paper Title** | **Authors (Year)** | **Architecture** | **Accuracy/Results** |
| --- | --- | --- | --- |
| Paper 1 | Zhang et al. (2021) | VGG16,  RES – 50,  DenseNet,  MobileNet | 94.35%  92.82%  87.35%  95.47% |
| Paper 2 | Li (2024) | Attention Network with ResNet | 96%(Attention Network) 98% (With focal loss) |
| Paper 3 | Shafi, Hasan, and Das (2022) | Xception + VGG16 (Feature Fusion) | 91.6% |
| Paper 4 | Rajpurkar et al. (2017) | ChexNet DenseNet121 | 96% |

Table 1: Research papers for pneumonia

The use of (CNNs) has significantly advanced tuberculosis (TB) detection from chest X-ray images. This section reviews key papers and methodologies that highlight different approaches to improving diagnostic accuracy using CNN and ResNet architectures.

**Paper 5 by Rahman et al. (2020)**

This paper presented a novel TB detection framework using a deep CNN architecture. The study applied preprocessing techniques, including image segmentation, to isolate lung regions from chest X-rays, reducing irrelevant noise and enhancing model focus. The authors trained their CNN model on publicly available TB chest X-ray datasets and utilized data augmentation to improve model robustness. By segmenting lung areas, they achieved higher accuracy (99.9%) compared to the 97.07% achieved on whole X-ray images. The paper also introduced visualization techniques like Grad-CAM to highlight pathological regions, aiding interpretability and clinical validation.

**Paper 6 by Liu et al. (2017)**

Liu et al. proposed TX-CNN, a custom deep learning model built on ResNet architecture. The study fine-tuned ResNet layers on a large tuberculosis chest X-ray dataset, demonstrating the importance of residual connections in maintaining performance during deep network training. TX-CNN outperformed traditional machine learning methods, achieving over 96% accuracy by leveraging the power of deep residual learning. The authors emphasized the use of data augmentation and dropout layers to prevent overfitting and enhance model generalization across diverse X-ray datasets.

Both studies highlight the critical role of CNN and ResNet architectures in TB detection, showcasing how preprocessing, segmentation, and augmentation techniques contribute to achieving high diagnostic accuracy. These approaches underscore the growing potential of AI-driven tools in supporting radiologists and improving TB diagnosis in resource-limited settings.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Paper Title** | **Authors (Year)** | **Architecture** | |  |  | | --- | --- | |  | **Accuracy** | |
| Paper 5 | Rahman et al. (2020) | CNN Custom Built | 99.9% (segmented), 97.07% (whole) |
| Paper 6 | Liu et al. (2017) | ResNet (TX-CNN) | 96% |

Table:2 Research Papers for Tuberculosis

# Methodology:

## Dataset overview:

This research utilizes two distinct medical image datasets focused on lung disease detection, specifically Pneumonia and Tuberculosis (TB), through chest X-ray analysis. Both datasets are ideal for the objective of this study, which aims to develop robust machine learning models for the rapid and accurate diagnosis of these diseases. The datasets were manually sourced for in-depth analysis using machine learning algorithms to enhance diagnostic processes in healthcare.

### 1. Pneumonia Dataset

The initial dataset is composed of chest X-ray images taken in the anterior-posterior position from pediatric patients aged between one and five years. The images originate from the Guangzhou Women and Children’s Medical Center in Guangzhou, China, as part of routine clinical practices (Kermany, Zhang, & Goldbaum, 2018). This dataset includes 5856 images, which are divided into two key categories: Normal and Pneumonia.

The Pneumonia category is further divided into two subtypes: Bacterial Pneumonia and Viral Pneumonia. The distribution of these categories is provided below.

Normal: 1583

Bacterial Pneumonia: 2780

Viral Pneumonia: 14937

These labelled X-ray images provided in .jpeg format, serve as the foundation for training machine learning models to differentiate between normal lung conditions and the two major types of pneumonia.

### 2. Tuberculosis (TB) Dataset

The second dataset is focused on Tuberculosis (TB) detection and includes chest X-ray images from both TB-positive cases and healthy individuals. This dataset was developed by team from the University of Dhaka, Qatar University, and their Malaysian counterparts, with support from doctors and team at Hamad Medical Corporation in Bangladesh and Qatar (Rahman et al., 2020). The dataset includes:

Normal images: 3500

TB-positive images: 700

These X-rays were collected as part of standard clinical procedures for TB diagnosis. The dataset’s structure makes it well-suited for training models to distinguish between TB and healthy lung images.

### 3. Challenges and Ethical Considerations

Both datasets used in this research have been anonymized to ensure patient privacy. As a result, no ethical approval was required for their use. These datasets are publicly available, and they do not contain any personal identifiable information, ensuring compliance with privacy standards.

While the focus of this research is on leveraging machine learning for accurate disease prediction, it is also important to highlight the need for validating the models to ensure their performance is reliable and unbiased. The emphasis of this study lies in developing models that can generalize well across diverse data and scenarios, thus minimizing the risk of misdiagnosis or biases in real-world clinical applications. By prioritizing model accuracy and fairness, this study aims to demonstrate the potential for AI to support clinicians in diagnosing lung diseases, while also mitigating the risks associated with deploying AI in healthcare settings.

## Data Preprocessing

It is an important step in the machine learning pipeline, particularly when working with image data. It involves preparing raw data for model training by cleaning, transforming, and structuring it in a way that optimizes the performance of the model. In this research, where the attention is on medical image analysis, data preprocessing specifically targets the preparation of X-ray images for training models.

The raw X-ray images collected from different sources often come with discrepancies in size, resolution, and quality. These variations can impact the model’s capacity to learn effective patterns. Therefore, preprocessing is necessary to standardize the data, ensuring that all images are uniform in terms of format and quality.

The first step in the preprocessing pipeline is image resizing. To ensure consistency, all X-ray images are resized to 224x224 pixels. The particular size is chosen because it aligns well with the input requirements of popular Convolutional Neural Networks (CNNs) and retains enough detail for feature extraction. Resizing helps model to receives inputs of uniform dimensions, making the training process more efficient.

Next, pixel normalization is applied. Since pixel values in images can vary significantly, this step ensures that all pixel intensities are scaled to a constant range. Specifically, the pixel values are divided by 255.0, bringing them into the range of 0 to 1. This normalization reduces the impact of variations in pixel intensity, promoting faster convergence during model training and improving overall model stability.

To implement this, each image in the dataset is checked to see if it matches the target size of 224x224 pixels. If the dimensions differ, the image is resized using bilinear interpolation, a method that adjusts the image size while preserving as much of the original quality as possible. This resizing process is combined with the normalization step, ensuring that the final image has pixel values scaled appropriately for input into the model.

**For tuberculosis (TB)** classification, additional preprocessing steps were applied to address class imbalance and ensure uniform dataset sizes. Initially, the dataset contained 3500 images for the higher class, while the lower class had significantly fewer images. To balance the dataset, resampling techniques were employed. The higher class was initially resampled to 3500 images, and attempts were made to up sample the lowest class to 1500 images. However, the model did not perform as expected with this configuration.

To improve performance, the upper class was subsequently reduced to 700 images, matching the lower class, resulting in a balanced dataset with 700 images for each class. This down sampling allowed for more stable training and better generalization. The images were then resized to 224x224 pixels and normalized, following the same pipeline as the other X-ray images. The balanced and pre-processed dataset was used to train the deep learning model effectively, ensuring that the model could learn features from both classes without being biased by class imbalance.

Through these preprocessing steps—resizing, normalization, and class resampling—the dataset is prepared for efficient and effective training of deep learning models. This process minimizes the potential for errors caused by image inconsistencies, enabling the model to focus on learning relevant features from the data.

## Architectures used:

### Convolutional Neural Network (CNN) Architecture

In this project, a Convolutional Neural Network (CNN) was employed to predict lung diseases from chest X-ray images. CNNs are adept at image analysis due to their capacity to learn spatial hierarchies of features, making them suitable for medical image analysis.

**Structure of the CNN:**

1. **Input Layer:** Processes preprocessed chest X-ray images standardized to a fixed resolution, ensuring uniform input dimensions.
2. **Convolutional Layers:** Apply filters to detect features like edges and textures, preserving spatial relationships. ReLU activation functions introduce non-linearity, enhancing feature extraction.
3. **Pooling Layers:** Utilize MaxPooling to downsample feature maps, reducing spatial dimensions and computational complexity while retaining significant information.
4. **Flattening Layer:** Converts multidimensional feature maps into a single vector, bridging convolutional layers and fully connected layers.
5. **Fully Connected Layers:** Process the flattened vector through dense layers to combine learned features. A Softmax activation function in the final layer outputs probabilities for classification into categories such as Normal, Bacterial Pneumonia, Viral Pneumonia, or Tuberculosis.

This architecture enables the model to effectively learn and identify patterns associated with various lung diseases in chest X-ray images.

**The Below setup is for Pneumonia Model which is developed from scratch with CNN**

The Convolutional Neural Network (CNN) developed for this project follows a sequential architecture to classify chest X-ray images into three categories: Normal, Bacterial Pneumonia, and Viral Pneumonia. Additionally, the model is designed to analyze another dataset to determine its effectiveness. The architecture consists of five convolutional layers with progressively larger filter sizes (32, 64, 64, 128, and 256), which extract increasingly complex features from input images. Batch Normalization follows each convolutional layer to enhance training stability, while MaxPooling layers reduce the spatial dimensions of the feature maps, preserving crucial information and minimizing computational load. Dropout layers (0.1 to 0.2) are employed to mitigate overfitting.

The fully connected section of the model includes a dense layer with 128 units, followed by a dropout layer for regularization. A softmax-activated output layer predicts the class probabilities for the three categories. The model is optimized using the Adam algorithm, with categorical cross-entropy as the loss function and accuracy as the evaluation metric. Data augmentation techniques such as rotation, zoom, and horizontal flipping enhance the model’s robustness. Additionally, early stopping and learning rate adjustments improve convergence and prevent overfitting. This architecture effectively balances complexity and generalization, resulting in high classification performance.

A diagram of a network diagram

Description automatically generated

Fig: 1 CNN Architecture (Paperswithcode.com, 2024)

**The Below setup is for Tuberculosis Model which is developed from scratch with CNN**

In this project, I developed a CNN model designed for tuberculosis classification, utilizing a sequential architecture. The model consists of four convolutional layers with progressively increasing filter sizes (32, 64, and 128). Each convolution is followed by MaxPooling layers to downsample spatial dimensions while preserving critical features. A dropout layer (0.5) is applied after flattening to reduce the risk of overfitting.

The final dense layer contains two units with softmax activation for binary classification. The model is compiled using the Adam optimizer, with categorical cross-entropy as the loss function and accuracy as the primary evaluation metric. This configuration ensures efficient and adaptive learning throughout the training process.

Data augmentation is applied using rescaled images for both validation and training, enhancing the model's generalization. The training process is supported by early stopping, monitoring validation accuracy to prevent overfitting, and learning rate reduction to fine-tune performance during plateau phases. The model is trained for 50 epochs and saved for future use, demonstrating a robust and efficient approach to tuberculosis detection.

### VGG16 Architecture

VGG16 is a deep convolutional neural network (CNN) created by the Visual Geometry Group at the University of Oxford. The architecture features 16 layers, comprising 13 convolutional layers and 3 fully connected layers. The convolutional layers utilize small 3x3 filters, enhancing feature extraction while maintaining computational efficiency. VGG16 is renowned for its simplicity and high performance in image recognition tasks.

To optimize feature retention and reduce overfitting, VGG16 employs max-pooling layers that downsample spatial dimensions. The final layers are fully connected and use softmax activation to classify images into predefined categories. Initially trained on the ImageNet dataset, which includes millions of labeled images across various categories, VGG16 is highly effective for transfer learning. Its deep structure and accuracy make it a preferred model for image classification tasks (Simonyan and Zisserman, 2015).

A diagram of a graph

Description automatically generated

Fig 2: VGG16 Architecture (Bangar, 2022)

The diagram above illustrates the VGG16 architecture, featuring 5 CNN layers, max-pool layers, and fully connected layers. It shows the flow of the image through the network, with reductions in spatial dimensions and an increase in feature maps, followed by the classification layer.

**The Below setup is for Pneumonia Model**

Another dataset has been added to the training. The model uses flattening, a dense layer (256 units), and dropout (0.5) to prevent overfitting. Flattening converts feature maps into a one-dimensional vector. The last 20 layers of VGG16 are fine-tuned for four-category classification. Adam optimizer (5e-5) and categorical cross-entropy ensure stable learning with gradual updates.

Data augmentation (rotation, shifting, zooming) boosts robustness. Early stopping halts training when validation accuracy plateaus, improving generalization and saving resources. The model, trained for 75 epochs on rescaled images, is saved for future use. This approach leverages transfer learning and fine-tuning for multi-class chest X-ray classification.

The model uses global average pooling, a dense layer (256 units), and dropout (0.4) to prevent overfitting. The last 10 layers of VGG16 are fine-tuned, balancing adaptation with retained pre-learned features. Adam optimizer (0.00003) and categorical cross-entropy ensure stable learning.

Data augmentation (rotation, shifting, zooming) boosts robustness. Early stopping prevents overfitting by halting on validation loss plateaus. The model, trained for 40 epochs on rescaled images, is saved for future use. This approach leverages transfer learning and fine-tuning for pneumonia classification.

The Below setup is for combinational modal for both Pneumonia and Tuberculosis

### InceptionResNetV2 Architecture:

**InceptionResNetV2** combines the benefits of both the **Inception** and **ResNet** architectures, integrating the efficient feature extraction of Inception modules with the deep residual connections of ResNet. This design has multiple Inception blocks, each one has parallel CNN layers with different filter sizes to capture a wide range of features. the vanishing gradient problem will be relieved with the help of the residual connections, allowing for deeper networks. The final layers of the model are fully connected, utilizing **softmax** activation for multi-class classification. InceptionResNetV2 has shown outstanding results in image distinguish tasks due to its balance between computational efficiency and model depth, making it well-suited for complex datasets like medical images (Szegedy et al., 2015)

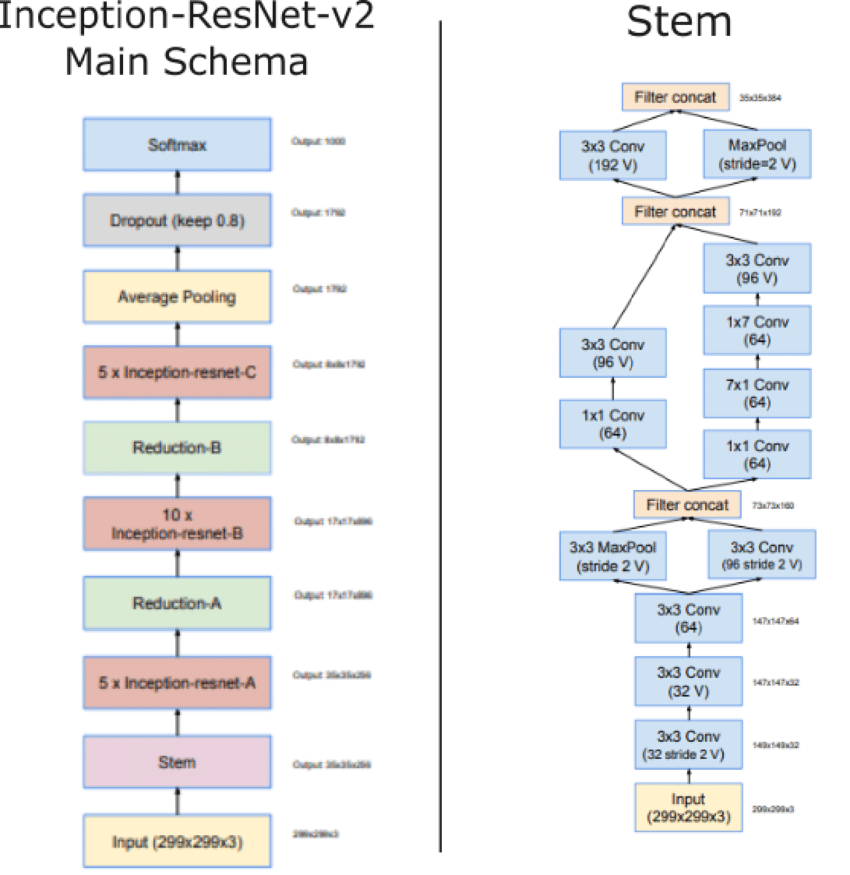


Fig 3: Inception ResNetV2 Architecture and schema (Szegedy et al., 2015)

The image above depicts the architecture of **InceptionResNet-v2**, showcasing the flow from input through various **Inception-ResNet blocks** and the **Stem** section. It illustrates how convolutional layers with different filter sizes and residual connections are used to efficiently extract features for image classification tasks.

In this project, InceptionResNetV2 with pre-trained ImageNet weights is used for binary classification of tuberculosis. The base model's layers are frozen to retain pre-trained features. Three additional layers are integrated: a flattening layer, a dropout layer with a rate of 0.5, and a dense output layer with softmax activation for binary classification.

The model is compiled with the Adam optimizer and categorical cross-entropy as the loss function. Data augmentation is performed through rescaling during both training and validation. Early stopping is applied to track validation accuracy, ceasing training after 10 epochs without improvement. Additionally, learning rate reduction is triggered by plateauing validation loss. The model undergoes 50 epochs of training and is saved for future tuberculosis classification tasks.

### Chexnet-DenseNet121 Architecture:

CheXNet is a CNN based on the DenseNet121 architecture, designed to detect pneumonia disease from X-ray photos. It utilizes densely connected layers to enhance feature propagation and reduce the risk of vanishing gradients. The model leverages pre-trained ImageNet weights, with the final layers fine-tuned for pneumonia classification. ChexNet’s architecture includes global average pooling, dropout, and fully connected layers, making it highly efficient for medical image analysis. It has demonstrated performance comparable to or exceeding that of radiologists in pneumonia detection (Rajpurkar et al., 2017).

For this project, DenseNet121 with pre-trained ImageNet weights was used to classify chest X-ray images into three categories. The majority of the base model’s layers were kept frozen, with the last 20 layers made trainable to fine-tune the model for the classification task. To customize the architecture, three additional layers were incorporated: a global average pooling layer, a dense layer featuring 256 units with ReLU activation, and a dropout layer (rate of 0.7) to minimize overfitting. The final layer is a dense output layer with three units, employing softmax activation to generate classification probabilities.

The model is compiled with the Adam optimizer, using a low learning rate of 1e-5 and categorical cross-entropy as the loss function. The training process involves augmented and rescaled images. Early stopping was implemented to monitor validation accuracy, ceasing training if no improvement occurred for 10 consecutive epochs. The model underwent 50 epochs of training to enhance the classification of pneumonia into normal, bacterial, and viral categories.

# Results:

**Pneumonia:**

**Pneumonia model developed with Scratch:**

A custom Convolutional Neural Network (CNN) was developed for lung disease classification using chest X-ray images to distinguish between bacterial pneumonia, normal , and viral pneumonia cases. The architecture consists of five convolutional layers (32, 64, 64, 128, 256 filters) with Batch Normalization and MaxPooling to minimize overfitting. Dropout layers (0.1–0.2) were applied to enhance generalization. A final fully connected layer of 128 units with softmax activation performed multi-class classification. For optimizer it has used Adam, and cross-entropy loss for categorical, with callbacks like early stopping and learning rate adjustment.

Training involved up to 50 epochs on augmented data (rotation, shifts, zoom, flips). Iterative tuning of layer structures, dropout rates, and data augmentation led to the final model. The training accuracy reached **80%**, while validation accuracy peaked at **78%**. Test results showed **70.19% accuracy** and a test loss of 0.72. Additional metrics included **61.54% overall accuracy**, **75% precision**, **62% recall**, and an **F1 score of 61%**. Further improvements could focus on additional hyperparameter tuning and dataset expansion.

A graph showing a line graph

Description automatically generated

Fig 4 : Accuracy plot for Pneumonia scratch model

**Pneumonia model developed with VGG Architecture.**

The VGG16 model fine-tuned with ImageNet weights demonstrated promising performance on the chest X-ray dataset, achieving a test accuracy of approximately **96.7%**. The training accuracy gradually increased over the epochs, eventually stabilizing near **0.95**, reflecting the model's successful learning during the training process. On the other hand, the validation accuracy also followed a similar upward trend, with fluctuations, and reached around **0.90** by the end of the training. These results suggest that the model was able to generalize well to unseen data, despite the fluctuations observed in the validation accuracy. The model’s performance on the validation set closely aligned with the training results, indicating that overfitting was effectively managed.

Further evaluation of the model using **precision**, **recall**, and **F1 score** metrics showed that the model not only achieved high accuracy but also had balanced performance across the classes. The **precision** of approximately **88.3%** indicates that the model is effective in identifying true positive cases (correctly classifying each class), while the **recall** value of **86.7%** reflects its ability to correctly detect instances from the true classes. The **F1 score**, which balances precision and recall, stood at **87.1%**, suggesting a solid trade-off between the two metrics and reinforcing the reliability of the model in classifying chest X-ray images. Additionally, the use of early stopping during training helped prevent overfitting, as seen in the steady performance on the validation set. These findings confirm the effectiveness of transfer learning with VGG16 for medical image classification tasks.

A graph with blue and orange lines

Description automatically generated

Fig 5: Accuracy plot for Pneumonia VGG model

**Pneumonia with ChexNet- DenseNet121 architecture**

The DenseNet121 model with ImageNet weights showed a solid performance on the chest X-ray classification task, achieving a test accuracy of approximately **84.7%**. The training accuracy increased steadily over the epochs, reaching about **0.91** by the end of the 50 epochs, which indicates that the model successfully learned to identify patterns from the data. The validation accuracy exhibited a similar upward trend, with some fluctuations but ultimately stabilized around **0.83**. This indicates the model's generalization ability, as it performed well on unseen data, with validation accuracy close to the training accuracy. The fluctuations in validation accuracy throughout training suggest the model's robustness and the potential challenges in balancing between underfitting and overfitting.

In addition to the accuracy, the model's **precision** was **88.3%**, indicating that it is effective in identifying the correct class for each image, with a lower false-positive rate. The **recall** value of **86.7%** shows that the model correctly detects most of the instances of the positive classes in the validation set. The **F1 score** of **87.1%** demonstrates a solid trade-off between precision and recall, ensuring balanced performance across all classes. This model's ability to achieve such a high F1 score reflects its well-tuned nature and its effectiveness in the medical image classification task. These results are consistent with the use of a pre-trained DenseNet121 model, fine-tuned on the chest X-ray dataset. The model's successful adaptation to the dataset, aided by the transfer learning approach, underscores the effectiveness of using pre-trained models for medical image analysis.

A graph with blue and orange lines

Description automatically generated

Fig 6: Accuracy plot for Pneumonia ChexNet-DenseNet121 model

**Tuberculosis:**

**Tuberculosis model developed from Scratch CNN:**

The CNN model trained for tuberculosis classification demonstrated strong performance, achieving a test accuracy of approximately **93.3%**. During the training process, the training accuracy increased steadily and reached nearly **1.0** by the end of the 50 epochs. This sharp rise in training accuracy indicates that the model successfully learned to classify the images during training. Similarly, the validation accuracy followed a similar trend, stabilizing around **0.94**. Although there were slight fluctuations in the validation accuracy, the overall pattern indicates that the model generalizes well to unseen data, effectively avoiding overfitting.

Further evaluation using **precision**, **recall**, and **F1 score** metrics showed the model’s excellent classification ability. The **precision** of **98%** suggests that the model is highly effective at correctly identifying true positive cases of tuberculosis, with minimal false positives. The **recall** value of **95%** reflects the model’s ability to correctly detect tuberculosis cases. The **F1 score** of **97%** balances the precision and recall, showing that the model performs well across both metrics. These results underline the reliability of the CNN model in detecting tuberculosis in chest X-ray images, making it a promising tool for medical applications in this domain. The use of callbacks, such as early stopping and learning rate adjustment, helped optimize the training process and ensure robust performance without overfitting.

A graph with blue and orange lines

Description automatically generated

Fig 7: Accuracy plot for Tuberculosis scratch model

**Tuberculosis model developed from InceptionResNetV2**

The InceptionResNetV2 model fine-tuned with pre-trained ImageNet weights achieved an impressive test accuracy of approximately **97.6%** on the tuberculosis dataset. The training accuracy increased rapidly during the first few epochs, eventually reaching **1.0**, indicating that the model effectively learned the features from the training data. Similarly, the validation accuracy followed a comparable upward trend, stabilizing around **0.96**, with some minor fluctuations. These results suggest that the model generalized well to the validation set, avoiding overfitting and effectively recognizing patterns in unseen data.

Further evaluation of the model's performance using **precision**, **recall**, and **F1 score** metrics shows outstanding results. The **precision** of **98.2%** indicates that the model correctly identified almost all positive cases (tuberculosis), with very few false positives. The **recall** of **98.1%** demonstrates that the model accurately identified nearly all tuberculosis cases from the validation set. The **F1 score**, which balances precision and recall, was **98.1%**, indicating that the model performs exceptionally well across both metrics. These results highlight the effectiveness of the InceptionResNetV2 model for the task of tuberculosis detection, confirming its potential for reliable medical image classification. The use of early stopping and learning rate adjustment helped optimize the training process and further improve the model's performance.

**Combination of both the datasets developed with VGG:**

The multi-disease classifier model, built using the **VGG16 architecture** and fine-tuned with both the **pneumonia** and **tuberculosis** datasets, showed promising results, achieving a **test accuracy** of approximately **89%**. The **training accuracy** steadily increased over the course of **75 epochs**, reaching close to **0.98**, indicating that the model learned effectively from both datasets. Similarly, the **validation accuracy** followed an upward trajectory, stabilizing around **0.85**, demonstrating the model’s ability to generalize well to different data. This slight gap between the validation and training accuracy tells that the model balanced both datasets and perform well across the diverse categories without significant overfitting.

In my experimentation with various setups and configurations, I explored the impact of adjusting the model architecture by reducing and increasing the number of layers. I also **fine-tuned** the model by freezing different numbers of layers and varying the learning rate. I observed that when I **froze more layers** in the VGG16 base model, the accuracy **increased**. In contrast, when **fewer layers were frozen**, the accuracy **decreased**. This insight was crucial, as it indicated that the model performed better when most of the base layers were kept frozen, suggesting that the **pre-trained weights** from ImageNet were already highly beneficial for the task and did not need much adjustment.

Further evaluation of the model’s performance using **precision**, **recall**, and **F1 score** metrics showed excellent results. The **precision** of **89.2%** indicates that the model effectively minimized false positives, correctly identifying positive cases of pneumonia, tuberculosis, and other diseases in the dataset. The **recall** of **89.5%** reflects the model’s strong ability to detect true positive cases, ensuring that most instances of pneumonia and tuberculosis were correctly identified. The **F1 score** of **89%** is a balanced measure of recall and precision, showing that the model performs well across both metrics and is well-suited for multi-disease classification tasks. These results underscore the effectiveness of the VGG16 model for classifying both pneumonia and tuberculosis, making it a valuable tool for **medical diagnosis** in chest X-ray images. The use of **early stopping** during training helped to further optimize the model’s performance, avoiding overfitting while still achieving high accuracy across all classes.

A graph with blue and orange lines

Description automatically generated

Fig 8: Accuracy plot for Combination of Both the diseases using VGG

**Best Model for Pneumonia and Tuberculosis:**

The **InceptionResNetV2** model fine-tuned with ImageNet weights demonstrated the best performance for **tuberculosis** detection, achieving a **test accuracy** of **97.6%**. It showed excellent precision (**98.2%**), recall (**98.1%**), and F1 score (**98.1%**), effectively identifying tuberculosis cases with minimal false positives and negatives.

For **pneumonia**, the **VGG16 model**, fine-tuned with ImageNet weights, achieved an **precision of 88.3%**, with **accuracy of 96.5%** and **recall** is **86.7%**. While **InceptionResNetV2** excelled in tuberculosis detection, **VGG16** proved to be the most reliable for pneumonia, making it a strong model for multi-disease classification tasks.

After checking all the parameters, **VGG16** is considered for both the datasets and added Tb class retrained again with the both the datasets.  
  
At the end, all the models which were trained which are loaded and placed in the cell so all the models can be tested back with the actual X-ray image, and the result will look something like in the below figure fig-9

A x-ray of a chest

Description automatically generated A screenshot of a computer screen

Description automatically generated

Fig-9: Example result tested with one of our trained model

# Difficulties Encountered During the Project

Throughout the project, several challenges were encountered that required careful attention and problem-solving to ensure progress. Two of the most prominent issues involved managing the large dataset and addressing limitations in computational resources.

**Computational Resource Limitations:**  
The project relied on a T4 GPU processor for model training. While effective for most tasks, prolonged training sessions and large batches of data sometimes led to GPU exhaustion. This necessitated adjustments in batch sizes, checkpointing strategies, and occasional breaks to allow the GPU to recover. Although this did not halt the project, it introduced delays, requiring patience and efficient scheduling of training sessions to make the most of available resources.

**Working with Multiple Datasets and Balancing Data:**  
A significant aspect of this project was the use of two distinct medical image datasets – the pneumonia and tuberculosis (TB) datasets. Each dataset varied in size and class distribution, with differing numbers of images and imbalanced classes. This discrepancy posed challenges during model training, biased predictions can led due to data imbalance. To address this, I employed oversampling and under sampling techniques to ensure the datasets were balanced. Oversampling was used to increase the representation of minority classes, while under sampling helped reduce the size of dominant classes. This process not only mitigated bias but also ensured that the combined dataset provided fair and accurate model training.

# Conclusion:

The implementation of the **VGG16 model**, enhanced with pre-trained ImageNet weights, has proven to be a highly effective solution for the automated diagnosis of lung diseases like **pneumonia** and **tuberculosis**. The model achieved a **test accuracy** of **86.7%** for pneumonia and **97.6%** for tuberculosis, making it a reliable tool for diagnostic purposes. Its ability to streamline the diagnostic process significantly improves healthcare efficiency, particularly in resource-limited settings where timely access to accurate diagnostics is often a challenge. By integrating VGG16 into clinical workflows, healthcare providers can enhance early detection, leading to better patient outcomes, especially in underserved areas.

Through this project, we observed that the **VGG16 model** performed exceptionally well for both pneumonia and tuberculosis detection. After experimenting with various models, we found that VGG16’s performance was robust across both datasets, which led to the decision to train it on the **combination of both pneumonia and tuberculosis datasets**. This approach enables the model to handle multi-disease classification efficiently. The success of VGG16 in these tasks highlights its potential for widespread use in medical diagnostics, paving the way for AI-driven healthcare tools that can improve access to quality care, promote early intervention, and support more inclusive healthcare systems globally.

# Future work:

This study has laid a strong foundation for lung disease diagnosis using machine learning, focusing on Pneumonia and Tuberculosis. However, several potential directions for future work can enhance the model's scalability, reliability, and clinical impact:

1. **Expanding Disease Coverage**  
   Currently, the model focuses on detecting Pneumonia and Tuberculosis. Future research could expand its capabilities to identify a broader spectrum of respiratory diseases, including Chronic Obstructive Pulmonary Disease (COPD), lung cancer, and interstitial lung conditions. Creating a comprehensive diagnostic model capable of identifying multiple lung conditions would significantly improve its practical utility, enabling its deployment in diverse healthcare settings.
2. **Incorporating Advanced Data Augmentation Techniques**

In this study, standard data augmentation techniques were applied to enhance the dataset. Future research could investigate more advanced approaches, such as Generative Adversarial Networks (GANs), to generate synthetic images. This method can help address class imbalances, expand the dataset, and improve the model’s overall robustness.

1. **Developing Lightweight Models for Deployment**  
   Given the computational intensity of CNN-based models, developing lightweight versions of the model that can run on low-power devices, such as mobile phones or portable diagnostic tools, could greatly expand its accessibility. Techniques like model pruning, quantization, or knowledge distillation could be explored to reduce the model’s size and resource requirements without compromising accuracy.

References:

Bangar, S. (2022). *VGG-Net Architecture Explained*. [online] Medium. Available at: <https://medium.com/@siddheshb008/vgg-net-architecture-explained-71179310050f>.

Chalmers, V. (2024). *The 13 symptoms of pneumonia you must NEVER ignore after Queen Camilla’s diagnosis...* [online] The Sun. Available at: <https://www.thesun.co.uk/health/20387965/urgent-warning-deaths-pneumonia-winter-symptoms/> [Accessed 10 Dec. 2024].  
Kermany, D., Zhang, K., & Goldbaum, M. H. (2018, January 6). Labeled Optical Coherence Tomography (OCT) and Chest X-Ray Images for Classification. <https://doi.org/10.17632/rscbjbr9sj.2>

Li, D. (2024). Attention-enhanced architecture for improved pneumonia detection in chest X-ray images. *BMC Medical Imaging*, 24(1). doi:<https://doi.org/10.1186/s12880-023-01177-1>.

Natashia, C. (2021). Chest X-rays Pneumonia Detection using Convolutional Neural Network. Medium. <https://towardsdatascience.com/chest-x-rays-pneumonia-detection-using-convolutional-neural-network-63d6ec2d1dee>

Paperswithcode.com. (2024). Available at: <https://production-media.paperswithcode.com/method_collections/cnn.jpeg> [Accessed 10 Dec. 2024]

Rajpurkar, P., Irvin, J., Zhu, K., Yang, B., Mehta, H., Duan, T., Ding, D., Bagul, A., Langlotz, C., Shpanskaya, K., Lungren, M.P. and Ng, A.Y. (2017). *CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1711.05225>.

Simonyan, K. and Zisserman, A. (2015). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1409.1556>.

Shafi, A., Hasan, M. and Das, S. (2022). Pneumonia Detection from Chest X-ray Images Using Transfer Learning by Fusing the Features of Pre-trained Xception and VGG16 Networks. doi:<https://doi.org/10.1109/iccit57492.2022.10054672>.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. and Wojna, Z. (2015). *Rethinking the Inception Architecture for Computer Vision*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1512.00567>

Szepesi, P. and Szilágyi, L. (2022). Detection of pneumonia using convolutional neural networks and deep learning. Biocybernetics and Biomedical Engineering. doi:<https://doi.org/10.1016/j.bbe.2022.08.001>.

Varshni, D., Thakral, K., Agarwal, L., Nijhawan, R. and Mittal, A. (2019). Pneumonia Detection Using CNN based Feature Extraction. [online] IEEE Xplore. doi:<https://doi.org/10.1109/ICECCT.2019.8869364>.

World (2024). *Tuberculosis*. [online] Who.int. Available at: <https://www.who.int/news-room/fact-sheets/detail/tuberculosis> [Accessed 10 Dec. 2024].

World Health Organization (2022). *Pneumonia*. [online] Who.int. Available at: <https://www.who.int/news-room/fact-sheets/detail/pneumonia>.

www.asthmaandlung.org.uk. (2022). *Asthma + Lung UK analysis reveals UK has highest number of pneumonia deaths in Europe | Asthma + Lung UK*. [online] Available at: <https://www.asthmaandlung.org.uk/media/press-releases/asthma-lung-uk-analysis-reveals-uk-has-highest-number-pneumonia-deaths-europe>.

Zhang, D., Ren, F., Li, Y., Na, L. and Ma, Y. (2021). Pneumonia Detection from Chest X-ray Images Based on Convolutional Neural Network. *Electronics*, 10(13), p.1512. doi:<https://doi.org/10.3390/electronics10131512>.

# Appendix:

Codelink: <https://colab.research.google.com/drive/1NgNLC2iszSYD-CidqNsT0a2Jw0Dql7Ci?usp=sharing>

# Function to categorize images based on their labels

def categorize\_images(directory):

    virus\_folder = os.path.join(directory, 'Vir')

    bacteria\_folder = os.path.join(directory, 'Bac')

    os.makedirs(virus\_folder, exist\_ok=True)

    os.makedirs(bacteria\_folder, exist\_ok=True)

    for file in os.listdir(directory):

        if file.lower().endswith('.jpeg'):

            img\_path = os.path.join(directory, file)

            try:

                with Image.open(img\_path) as img:

                    if 'bacteria' in file.lower():

                        move(img\_path, os.path.join(bacteria\_folder, file))

                    elif 'virus' in file.lower():

                        move(img\_path, os.path.join(virus\_folder, file))

                    else:

                        print(f"not found")

            except UnidentifiedImageError:

                print(f"diff img format")

train\_directory = '/content/drive/MyDrive/chest\_xray/train/PNEUMONIA'

test\_directory = '/content/drive/MyDrive/chest\_xray/test/PNEUMONIA'

# Categorize train and test images

categorize\_images(train\_directory)

categorize\_images(test\_directory)

train\_bacteria\_dir = '/content/drive/MyDrive/chest\_xray/train/PNEUMONIA/BACTERIA'

train\_normal\_dir = '/content/drive/MyDrive/chest\_xray/train/NORMAL'

train\_virus\_dir = '/content/drive/MyDrive/chest\_xray/train/PNEUMONIA/VIRUS'

test\_bacteria\_dir = '/content/drive/MyDrive/chest\_xray/test/PNEUMONIA/BACTERIA'

test\_normal\_dir = '/content/drive/MyDrive/chest\_xray/test/NORMAL'

test\_virus\_dir = '/content/drive/MyDrive/chest\_xray/test/PNEUMONIA/VIRUS'

# Function to load images from a specified directory

def load\_images(directory):

    loaded\_images = []

    for file in os.listdir(directory):

        if file.endswith('.jpeg'):

            img\_path = os.path.join(directory, file)

            img = Image.open(img\_path)

            loaded\_images.append(img)

    return loaded\_images

# Load training images

train\_bacteria\_images = load\_images(train\_bacteria\_dir)

train\_normal\_images = load\_images(train\_normal\_dir)

train\_virus\_images = load\_images(train\_virus\_dir)

# Load test images

test\_bacteria\_images = load\_images(test\_bacteria\_dir)

test\_normal\_images = load\_images(test\_normal\_dir)

test\_virus\_images = load\_images(test\_virus\_dir)

# Print dataset statistics

print("Training data counts - Bacteria:", len(train\_bacteria\_images),

      "Normal:", len(train\_normal\_images),

      "Virus:", len(train\_virus\_images))

print("Testing data counts - Bacteria:", len(test\_bacteria\_images),

      "Normal:", len(test\_normal\_images),

      "Virus:", len(test\_virus\_images))

train\_bacteria\_resized = '/content/drive/MyDrive/chest\_xray/train/PNEUMONIA/BACTERIA\_normalized'

train\_virus\_resized = '/content/drive/MyDrive/chest\_xray/train/PNEUMONIA/VIRUS\_normalized'

train\_normal\_resized = '/content/drive/MyDrive/chest\_xray/train/NORMAL\_normalized'

test\_bacteria\_resized = '/content/drive/MyDrive/chest\_xray/test/PNEUMONIA/BACTERIA\_normalized'

test\_virus\_resized = '/content/drive/MyDrive/chest\_xray/test/PNEUMONIA/VIRUS\_normalized'

test\_normal\_resized = '/content/drive/MyDrive/chest\_xray/test/NORMAL\_normalized'

# Function to preprocess images: resizing and normalization

def preprocess\_images(directory, target\_size=(224, 224), output\_dir=None):

    if output\_dir is None:

        output\_dir = directory

    else:

        os.makedirs(output\_dir, exist\_ok=True)  # Ensure the output directory exists

    resized\_count = 0

    already\_resized\_count = 0

    for file in os.listdir(directory):

        if file.endswith('.jpeg'):

            input\_path = os.path.join(directory, file)

            output\_path = os.path.join(output\_dir, file)

            with Image.open(input\_path) as img:

                # Resize the image if it doesn't match the target size

                if img.size != target\_size:

                    resized\_img = img.resize(target\_size)

                    resized\_img.save(output\_path)

                    resized\_count += 1

                    normalized\_img = np.array(resized\_img) / 255.0

                    normalized\_img = Image.fromarray((normalized\_img \* 255).astype(np.uint8))

                    normalized\_img.save(output\_path)

                else:

                    already\_resized\_count += 1

                    normalized\_img = np.array(img) / 255.0

                    normalized\_img = Image.fromarray((normalized\_img \* 255).astype(np.uint8))

                    normalized\_img.save(output\_path)

    print("Number of images resized:", resized\_count)

    print("Number of images already resized:", already\_resized\_count)

# Target size for resizing

image\_size = (224, 224)

# Process training images

preprocess\_images(bacteria\_dir, image\_size, train\_bacteria\_resized)

preprocess\_images(virus\_dir, image\_size, train\_virus\_resized)

preprocess\_images(normal\_dir, image\_size, train\_normal\_resized)

# Process testing images

preprocess\_images(test\_bacteria\_dir, image\_size, test\_bacteria\_resized)

preprocess\_images(test\_virus\_dir, image\_size, test\_virus\_resized)

preprocess\_images(test\_normal\_dir, image\_size, test\_normal\_resized)

cnn\_model = Sequential()

cnn\_model.add(Conv2D(filters=32, kernel\_size=(3, 3), strides=1, padding='same', activation='relu', input\_shape=(224, 224, 3)))

cnn\_model.add(BatchNormalization())

cnn\_model.add(MaxPooling2D(pool\_size=(2, 2), strides=2, padding='same'))

cnn\_model.add(Conv2D(filters=64, kernel\_size=(3, 3), strides=1, padding='same', activation='relu'))

cnn\_model.add(Dropout(rate=0.1))

cnn\_model.add(BatchNormalization())

cnn\_model.add(MaxPooling2D(pool\_size=(2, 2), strides=2, padding='same'))

cnn\_model.add(Conv2D(filters=64, kernel\_size=(3, 3), strides=1, padding='same', activation='relu'))

cnn\_model.add(BatchNormalization())

cnn\_model.add(MaxPooling2D(pool\_size=(2, 2), strides=2, padding='same'))

cnn\_model.add(Conv2D(filters=128, kernel\_size=(3, 3), strides=1, padding='same', activation='relu'))

cnn\_model.add(Dropout(rate=0.2))

cnn\_model.add(BatchNormalization())

cnn\_model.add(MaxPooling2D(pool\_size=(2, 2), strides=2, padding='same'))

cnn\_model.add(Conv2D(filters=256, kernel\_size=(3, 3), strides=1, padding='same', activation='relu'))

cnn\_model.add(Dropout(rate=0.2))

cnn\_model.add(BatchNormalization())

cnn\_model.add(MaxPooling2D(pool\_size=(2, 2), strides=2, padding='same'))

cnn\_model.add(Flatten())

cnn\_model.add(Dense(units=128, activation='relu'))

cnn\_model.add(Dropout(rate=0.2))

cnn\_model.add(Dense(units=3, activation='softmax'))

cnn\_model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

cnn\_model.summary()

train\_data\_gen = ImageDataGenerator(

    rescale=1./255,

    rotation\_range=20,

    width\_shift\_range=0.2,

    height\_shift\_range=0.2,

    shear\_range=0.2,

    zoom\_range=0.2,

    horizontal\_flip=True,

    fill\_mode='nearest'

)

train\_data\_generator = train\_data\_gen.flow\_from\_directory(

    '/content/drive/MyDrive/chest\_xray/train',

    target\_size=(224, 224),

    batch\_size=32,

    classes=['NORMAL\_normalized', 'BACTERIA\_normalized', 'VIRUS\_normalized'],

    class\_mode='categorical',

    subset='training'

)

val\_data\_gen = ImageDataGenerator(rescale=1./255, validation\_split=0.2)

validation\_data\_generator = val\_data\_gen.flow\_from\_directory(

    '/content/drive/MyDrive/chest\_xray/train',

    target\_size=(224, 224),

    batch\_size=32,

    classes=['NORMAL\_normalized', 'BACTERIA\_normalized', 'VIRUS\_normalized'],

    class\_mode='categorical',

    subset='validation'

)

lr\_adjuster = ReduceLROnPlateau(monitor='val\_loss', factor=0.5, patience=3, min\_lr=1e-6)

early\_stop = EarlyStopping(

    monitor='val\_loss',

    patience=5,

    verbose=1,

    restore\_best\_weights=True

)

cnn\_model.fit(

    train\_data\_generator,

    epochs=50,

    validation\_data=validation\_data\_generator,

    callbacks=[lr\_adjuster, early\_stop]

)

# Save the trained model

cnn\_model.save('/content/drive/MyDrive/chest\_xray/model\_custom.h5')

test\_data\_generator = ImageDataGenerator(rescale=1./255)

test\_data\_flow = test\_data\_generator.flow\_from\_directory(

    '/content/drive/MyDrive/chest\_xray/test',

    target\_size=(224, 224),

    batch\_size=32,

    classes=['NORMAL\_normalized', 'BACTERIA\_normalized', 'VIRUS\_normalized'],

    class\_mode='categorical',

    shuffle=False

)

loss, accuracy = model.evaluate(test\_data\_flow)

print("Test Loss:", loss)

print("Test Accuracy:", accuracy)

test\_dirs = ['/content/drive/MyDrive/chest\_xray/test/NORMAL\_normalized',

             '/content/drive/MyDrive/chest\_xray/test/BACTERIA\_normalized',

             '/content/drive/MyDrive/chest\_xray/test/VIRUS\_normalized']

X\_test\_images = []

true\_labels = []

for label, directory in enumerate(test\_dirs):

    for file\_name in os.listdir(directory):

        image\_path = os.path.join(directory, file\_name)

        image = load\_img(image\_path, target\_size=(224, 224))

        image\_array = img\_to\_array(image) / 255.0

        X\_test\_images.append(image\_array)

        true\_labels.append(label)

X\_test\_images = np.array(X\_test\_images)

true\_labels = np.array(true\_labels)

model = load\_model('/content/drive/MyDrive/chest\_xray/model\_custom.h5')

predicted\_probs = model.predict(X\_test\_images)

predicted\_classes = np.argmax(predicted\_probs, axis=1)

accuracy = accuracy\_score(true\_labels, predicted\_classes)

precision = precision\_score(true\_labels, predicted\_classes, average='weighted')

recall = recall\_score(true\_labels, predicted\_classes, average='weighted')

f1 = f1\_score(true\_labels, predicted\_classes, average='weighted')

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

print("F1 Score:", f1)

#vgg model with imagenet weights

from tensorflow.keras.applications import VGG16

from tensorflow.keras import layers, models, optimizers

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import EarlyStopping

from tensorflow.keras.preprocessing.image import ImageDataGenerator

vgg\_base = VGG16(weights='imagenet', include\_top=False, input\_shape=(224, 224, 3))

for layer in vgg\_base.layers[:-10]:

    layer.trainable = False

transfer\_model = models.Sequential()

transfer\_model.add(vgg\_base)

transfer\_model.add(layers.GlobalAveragePooling2D())

transfer\_model.add(layers.Dense(256, activation='relu'))

transfer\_model.add(layers.Dropout(0.4))

transfer\_model.add(layers.Dense(3, activation='softmax'))

transfer\_model.compile(optimizer=Adam(learning\_rate=0.00003),

                       loss='categorical\_crossentropy',

                       metrics=['accuracy'])

train\_aug = ImageDataGenerator(

    rescale=1./255,

    rotation\_range=15,

    width\_shift\_range=0.15,

    height\_shift\_range=0.15,

    shear\_range=0.1,

    zoom\_range=0.15,

    horizontal\_flip=True,

    fill\_mode='nearest'

)

train\_data = train\_aug.flow\_from\_directory(

    '/content/drive/MyDrive/chest\_xray/train',

    target\_size=(224, 224),

    batch\_size=32,

    classes=['NORMAL\_normalized', 'BACTERIA\_normalized', 'VIRUS\_normalized'],

    class\_mode='categorical'

)

validation\_aug = ImageDataGenerator(rescale=1./255, validation\_split=0.2)

val\_data = validation\_aug.flow\_from\_directory(

    '/content/drive/MyDrive/chest\_xray/train',

    target\_size=(224, 224),

    batch\_size=32,

    classes=['NORMAL\_normalized', 'BACTERIA\_normalized', 'VIRUS\_normalized'],

    class\_mode='categorical',

    subset='validation'

)

early\_stopping = EarlyStopping(monitor='val\_loss', patience=8, restore\_best\_weights=True)

history = transfer\_model.fit(train\_data,

                             epochs=40,

                             validation\_data=val\_data,

                             callbacks=[early\_stopping])

model.save('/content/drive/MyDrive/chest\_xray/vggmodel.h5')

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Training and Validation Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

test\_data\_generator = ImageDataGenerator(rescale=1.0 / 255)

#test data loader

test\_data\_loader = test\_data\_generator.flow\_from\_directory(

    directory='/content/drive/MyDrive/chest\_xray/test',

    target\_size=(224, 224),

    batch\_size=32,

    classes=['NORMAL\_normalized', 'BACTERIA\_normalized', 'VIRUS\_normalized'],

    class\_mode='categorical',

    shuffle=False

)

loss\_on\_test, accuracy\_on\_test = model.evaluate(test\_data\_loader)

# Print the evaluation results

print("Test Loss:", loss\_on\_test)

print("Test Accuracy:", accuracy\_on\_test)

test\_folders = [

    '/content/drive/MyDrive/chest\_xray/test/NORMAL\_normalized',

    '/content/drive/MyDrive/chest\_xray/test/BACTERIA\_normalized',

    '/content/drive/MyDrive/chest\_xray/test/VIRUS\_normalized'

]

X\_test = []

y\_actual = []

for class\_label, folder in enumerate(test\_folders):

    for file in os.listdir(folder):

        file\_path = os.path.join(folder, file)

        image = load\_img(file\_path, target\_size=(224, 224))

        image\_array = img\_to\_array(image) / 255.0

        X\_test.append(image\_array)

        y\_actual.append(class\_label)

X\_test = np.array(X\_test)

y\_actual = np.array(y\_actual)

predicted\_probs = model.predict(X\_test)

predicted\_classes = np.argmax(predicted\_probs, axis=1)

accuracy = accuracy\_score(y\_actual, predicted\_classes)

precision = precision\_score(y\_actual, predicted\_classes, average='weighted')

recall = recall\_score(y\_actual, predicted\_classes, average='weighted')

f1 = f1\_score(y\_actual, predicted\_classes, average='weighted')

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

print("F1 Score:", f1)

# Load the DenseNet121 base model with pre-trained ImageNet weights

base\_model = DenseNet121(weights='imagenet', include\_top=False, input\_shape=(224, 224, 3))

# Freeze all layers except the last 20 for fine-tuning

for layer in base\_model.layers[:-20]:

    layer.trainable = False

# Add custom layers for the new classification task

features = base\_model.output

features = GlobalAveragePooling2D()(features)

features = Dense(256, activation='relu')(features)

features = Dropout(0.7)(features)

output\_layer = Dense(3, activation='softmax')(features)

# Create the final model

model = Model(inputs=base\_model.input, outputs=output\_layer)

# Compile the model with a low learning rate for fine-tuning

model.compile(optimizer=Adam(learning\_rate=1e-5),

              loss='categorical\_crossentropy',

              metrics=['accuracy'])

# Define data generators for training and validation datasets

train\_data\_generator = ImageDataGenerator(rescale=1.0 / 255)

train\_loader = train\_data\_generator.flow\_from\_directory(

    '/content/drive/MyDrive/chest\_xray/train',

    target\_size=(224, 224),

    batch\_size=32,

    classes=['NORMAL\_normalized', 'BACTERIA\_normalized', 'VIRUS\_normalized'],

    class\_mode='categorical',

    subset='training'

)

validation\_data\_generator = ImageDataGenerator(rescale=1.0 / 255, validation\_split=0.2)

validation\_loader = validation\_data\_generator.flow\_from\_directory(

    '/content/drive/MyDrive/chest\_xray/train',

    target\_size=(224, 224),

    batch\_size=32,

    classes=['NORMAL\_normalized', 'BACTERIA\_normalized', 'VIRUS\_normalized'],

    class\_mode='categorical',

    subset='validation'

)

# Define early stopping callback to monitor validation accuracy

early\_stop = EarlyStopping(monitor='val\_accuracy', patience=10, restore\_best\_weights=True)

# Train the model with the training and validation datasets

history = model.fit(

    train\_loader,

    epochs=50,

    validation\_data=validation\_loader,

    callbacks=[early\_stop],

    workers=4,

    use\_multiprocessing=True,

    max\_queue\_size=10

)

model.save('/content/drive/MyDrive/chest\_xray/chexNet.h5')

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Training and Validation Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

test\_data\_generator = ImageDataGenerator(rescale=1./255)

test\_data\_loader = test\_data\_generator.flow\_from\_directory(

    '/content/drive/MyDrive/chest\_xray/test',

    target\_size=(224, 224),

    batch\_size=32,

    classes=['NORMAL\_normalized', 'BACTERIA\_normalized', 'VIRUS\_normalized'],

    class\_mode='categorical',

    shuffle=False

)

test\_loss, test\_accuracy = model.evaluate(test\_data\_loader)

print("Test Loss:", test\_loss)

print("Test Accuracy:", test\_accuracy)

preds = model.predict(test\_data\_generator)

pred\_classes = np.argmax(preds, axis=1)

classes\_true = test\_data\_generator.classes

class\_labels = list(test\_data\_generator.class\_indices.keys())

precision\_test = precision\_score(classes\_true, pred\_classes, average='weighted')

recall\_test = recall\_score(classes\_true, pred\_classes, average='weighted')

f1\_test = f1\_score(classes\_true, pred\_classes, average='weighted')

print("Precision:", precision\_test)

print("Recall:", recall\_test)

print("F1 Score:", f1\_test)

# directories for Tuberculosis and Normal images

tb\_directory = '/content/drive/MyDrive/TB\_Chest\_Radiography\_Database/Tuberculosis'

normal\_directory = '/content/drive/MyDrive/TB\_Chest\_Radiography\_Database/Normal'

# directory for the resampled dataset

resample\_base\_directory = '/content/drive/MyDrive/tuber2'

os.makedirs(resample\_base\_directory, exist\_ok=True)

# subdirectories for Tuberculosis and resampled Normal images

resampled\_tb\_directory = os.path.join(resample\_base\_directory, 'Tuberculosis')

resampled\_normal\_directory = os.path.join(resample\_base\_directory, 'Normal\_RS')

os.makedirs(resampled\_tb\_directory, exist\_ok=True)

os.makedirs(resampled\_normal\_directory, exist\_ok=True)

# Copyying all Tuberculosis images into the new directory

for file\_name in os.listdir(tb\_directory):

    shutil.copy(os.path.join(tb\_directory, file\_name), resampled\_tb\_directory)

# Randomly sampling 700 Normal images

normal\_images\_list = [os.path.join(normal\_directory, file) for file in os.listdir(normal\_directory)]

resampled\_normal\_images = resample(normal\_images\_list, replace=False, n\_samples=700, random\_state=42)

# Copy the resampled Normal images to the new directory

for file\_path in resampled\_normal\_images:

    shutil.copy(file\_path, resampled\_normal\_directory)

# Directories for storing resampled and normalized datasets

resampled\_tb\_directory = '/content/drive/MyDrive/tuber2/Tuberculosis'

resampled\_normal\_directory = '/content/drive/MyDrive/tuber2/Normal\_RS'

normalized\_normal\_directory = '/content/drive/MyDrive/tuber2/Normal\_Norm'

normalized\_tb\_directory = '/content/drive/MyDrive/tuber2/Tuberculosis\_Norm'

def get\_image\_count(folder\_path):

    valid\_extensions = ['.jpg', '.jpeg', '.png']

    total\_images = 0

    for filename in os.listdir(folder\_path):

        if any(filename.lower().endswith(extension) for extension in valid\_extensions):

            total\_images += 1

    return total\_images

target\_directory = '/content/drive/MyDrive/tuber2/Normal\_RS'

print(f"Total images in the folder: {get\_image\_count(target\_directory)}")

def process\_images(directory, target\_size=(224, 224), output\_directory=None):

    if output\_directory is None:

        output\_directory = directory

    else:

        os.makedirs(output\_directory, exist\_ok=True)

    resized\_count = 0

    already\_resized\_count = 0

    for file in os.listdir(directory):

        if file.endswith('.png'):

            input\_path = os.path.join(directory, file)

            output\_path = os.path.join(output\_directory, file)

            with Image.open(input\_path) as img:

                if img.size != target\_size:

                    adjusted\_image = img.resize(target\_size)

                    adjusted\_image.save(output\_path)

                    resized\_count += 1

                    scaled\_image = np.array(adjusted\_image) / 255.0

                    scaled\_image = Image.fromarray((scaled\_image \* 255).astype(np.uint8))

                    scaled\_image.save(output\_path)

                else:

                    already\_resized\_count += 1

                    scaled\_image = np.array(img) / 255.0

                    scaled\_image = Image.fromarray((scaled\_image \* 255).astype(np.uint8))

                    scaled\_image.save(output\_path)

    print("Number of images resized:", resized\_count)

    print("Number of images already resized:", already\_resized\_count)

target\_size = (224, 224)

process\_images(resampled\_normal\_directory, target\_size, normalized\_normal\_directory)

process\_images(resampled\_tb\_directory, target\_size, normalized\_tb\_directory)

import shutil

from sklearn.model\_selection import train\_test\_split

# Define source directories

normalized\_normal\_dir = '/content/drive/MyDrive/tuber2/Normal\_Norm'

normalized\_tb\_dir = '/content/drive/MyDrive/tuber2/Tuberculosis\_Norm'

# Define target directories for Normal and TB images

normal\_train\_dir = os.path.join(normalized\_normal\_dir, 'train')

normal\_val\_dir = os.path.join(normalized\_normal\_dir, 'val')

normal\_test\_dir = os.path.join(normalized\_normal\_dir, 'test')

tb\_train\_dir = os.path.join(normalized\_tb\_dir, 'train')

tb\_val\_dir = os.path.join(normalized\_tb\_dir, 'val')

tb\_test\_dir = os.path.join(normalized\_tb\_dir, 'test')

# Create the target directories

os.makedirs(normal\_train\_dir, exist\_ok=True)

os.makedirs(normal\_val\_dir, exist\_ok=True)

os.makedirs(normal\_test\_dir, exist\_ok=True)

os.makedirs(tb\_train\_dir, exist\_ok=True)

os.makedirs(tb\_val\_dir, exist\_ok=True)

os.makedirs(tb\_test\_dir, exist\_ok=True)

# Function to split and copy data into train, validation, and test sets

def distribute\_data(source\_directory, train\_directory, val\_directory, test\_directory, test\_split=0.15, val\_split=0.1765):

    all\_files = [os.path.join(source\_directory, file) for file in os.listdir(source\_directory) if os.path.isfile(os.path.join(source\_directory, file))]

    train\_val\_files, test\_files = train\_test\_split(all\_files, test\_size=test\_split, random\_state=42)

    train\_files, val\_files = train\_test\_split(train\_val\_files, test\_size=val\_split, random\_state=42)

    def copy\_to\_target(files\_list, target\_directory):

        for file in files\_list:

            shutil.copy(file, target\_directory)

    copy\_to\_target(train\_files, train\_directory)

    copy\_to\_target(val\_files, val\_directory)

    copy\_to\_target(test\_files, test\_directory)

    print(f"Data split for {source\_directory}: {len(test\_files)} for testing, {len(val\_files)} for validation, {len(train\_files)} for training ")

# Perform the data splitting for both Normal and Tuberculosis datasets

distribute\_data(normalized\_normal\_dir, normal\_train\_dir, normal\_val\_dir, normal\_test\_dir)

distribute\_data(normalized\_tb\_dir, tb\_train\_dir, tb\_val\_dir, tb\_test\_dir)

training\_data\_generator = ImageDataGenerator(rescale=1.0 / 255)

validation\_data\_generator = ImageDataGenerator(rescale=1.0 / 255)

# Setting up the data generators for training and validation datasets

train\_data = training\_data\_generator.flow\_from\_directory(

    directory='/content/drive/MyDrive/tuber2/train',

    target\_size=(224, 224),

    batch\_size=64,

    class\_mode='categorical'

)

validation\_data = validation\_data\_generator.flow\_from\_directory(

    directory='/content/drive/MyDrive/tuber2/val',

    target\_size=(224, 224),

    batch\_size=64,

    class\_mode='categorical'

)

# Building the convolutional neural network (CNN) model

cnn\_model = models.Sequential([

    layers.Conv2D(filters=32, kernel\_size=(3, 3), activation='relu', input\_shape=(224, 224, 3)),

    layers.MaxPooling2D(pool\_size=(2, 2)),

    layers.Conv2D(filters=64, kernel\_size=(3, 3), activation='relu'),

    layers.MaxPooling2D(pool\_size=(2, 2)),

    layers.Conv2D(filters=128, kernel\_size=(3, 3), activation='relu'),

    layers.MaxPooling2D(pool\_size=(2, 2)),

    layers.Conv2D(filters=128, kernel\_size=(3, 3), activation='relu'),

    layers.MaxPooling2D(pool\_size=(2, 2)),

    layers.Flatten(),

    layers.Dropout(rate=0.5),

    layers.Dense(units=2, activation='softmax')

])

# Compiling the CNN model

cnn\_model.compile(

    optimizer='adam',

    loss='categorical\_crossentropy',

    metrics=['accuracy']

)

# Setting up callbacks for learning rate adjustment and early stopping

learning\_rate\_reduction = ReduceLROnPlateau(

    monitor='val\_loss',

    factor=0.1,

    patience=5,

    min\_lr=0.0001

)

early\_stop\_callback = EarlyStopping(

    monitor='val\_accuracy',

    patience=10,

    restore\_best\_weights=True

)

# Training the model

training\_history = cnn\_model.fit(

    train\_data,

    epochs=50,

    validation\_data=validation\_data,

    callbacks=[early\_stop\_callback, learning\_rate\_reduction]

)

# Saving the trained model

cnn\_model.save('/content/drive/MyDrive/tuber2/CNN\_tuber2.h5')

trained\_model = load\_model('/content/drive/MyDrive/tuber2/CNN\_tuber2.h5')

test\_data\_generator = ImageDataGenerator(rescale=1.0 / 255)

test\_data = test\_data\_generator.flow\_from\_directory(

    directory='/content/drive/MyDrive/tuber2/test',

    target\_size=(224, 224),

    batch\_size=32,

    class\_mode='categorical',

    shuffle=False

)

loss\_on\_test, accuracy\_on\_test = trained\_model.evaluate(test\_data)

print("Test Loss:", loss\_on\_test)

print("Test Accuracy:", accuracy\_on\_test)

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Training and Validation Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

predicted\_probabilities = model.predict(test\_data)

predicted\_labels = np.argmax(predicted\_probabilities, axis=1)

true\_labels = test\_data.classes

label\_names = list(test\_data.class\_indices.keys())  # Retrieve class labels directly from the data generator

precision = precision\_score(true\_labels, predicted\_labels, average='weighted')

recall = recall\_score(true\_labels, predicted\_labels, average='weighted')

f1\_score\_value = f1\_score(true\_labels, predicted\_labels, average='weighted')

print("Precision:", precision)

print("Recall:", recall)

print("F1 Score:", f1\_score\_value)

report = classification\_report(true\_classes, predicted\_classes, target\_names=class\_labels)

print(report)

train\_data\_gen = ImageDataGenerator(rescale=1./255)

validation\_data\_gen = ImageDataGenerator(rescale=1./255)

# Training data generator

train\_data = train\_data\_gen.flow\_from\_directory(

    '/content/drive/MyDrive/tuber2/train',

    target\_size=(224, 224),

    batch\_size=64,

    class\_mode='categorical'

)

# Validation data generator

validation\_data = validation\_data\_gen.flow\_from\_directory(

    '/content/drive/MyDrive/tuber2/val',

    target\_size=(224, 224),

    batch\_size=64,

    class\_mode='categorical',

    shuffle=False

)

# Load the pre-trained InceptionResNetV2 model

base\_inception\_resnet = InceptionResNetV2(weights='imagenet', include\_top=False, input\_shape=(224, 224, 3))

# Freeze base model layers to prevent training

for base\_layer in base\_inception\_resnet.layers:

    base\_layer.trainable = False

# Add custom classification layers

flatten\_layer = layers.Flatten()(base\_inception\_resnet.output)

dropout\_layer = layers.Dropout(0.5)(flatten\_layer)

output\_layer = layers.Dense(2, activation='softmax')(dropout\_layer)

model = models.Model(inputs=base\_inception\_resnet.input, outputs=output\_layer)

# Compile the model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Callbacks for training

learning\_rate\_scheduler = ReduceLROnPlateau(monitor='val\_loss', factor=0.1, patience=5, min\_lr=0.0001)

early\_stop = EarlyStopping(monitor='val\_accuracy', patience=10, restore\_best\_weights=True)

# Train the model

training\_history = model.fit(

    train\_data,

    epochs=50,

    validation\_data=validation\_data,

    callbacks=[early\_stop, learning\_rate\_scheduler]

)

# Save the trained model

model.save('/content/drive/MyDrive/tuber2/InceptionResNetV2\_tuber2.h5')

cnn\_model = load\_model('/content/drive/MyDrive/tuber2/InceptionResNetV2\_tuber2.h5')

# Prepare the test data generator

test\_data\_gen = ImageDataGenerator(rescale=1./255)

test\_data = test\_data\_gen.flow\_from\_directory(

    '/content/drive/MyDrive/tuber2/test',

    target\_size=(224, 224),

    batch\_size=32,

    class\_mode='categorical',

    shuffle=False

)

# Evaluate the model on the test dataset

test\_loss, test\_accuracy = cnn\_model.evaluate(test\_data)

# Generate predictions and compute the confusion matrix

predicted\_probabilities = cnn\_model.predict(test\_data)

predicted\_labels = np.argmax(predicted\_probabilities, axis=1)

actual\_labels = test\_data.classes

conf\_matrix = confusion\_matrix(actual\_labels, predicted\_labels)

# Visualize the confusion matrix

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.xlabel('Predicted Labels')

plt.ylabel('True Labels')

plt.title('Confusion Matrix')

plt.show()

# Print test performance metrics

print("Test Loss:", test\_loss)

print("Test Accuracy:", test\_accuracy)

# Plot training and validation accuracy over epochs

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Training and Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

# Generate predictions for the test dataset

predicted\_probabilities = model.predict(test\_generator)

predicted\_labels = np.argmax(predicted\_probabilities, axis=1)

# Get the true labels and class names

actual\_labels = test\_generator.classes

class\_names = list(test\_generator.class\_indices.keys())

# Calculate precision, recall, and F1 score

precision = precision\_score(actual\_labels, predicted\_labels, average='weighted')

recall = recall\_score(actual\_labels, predicted\_labels, average='weighted')

f1\_score\_value = f1\_score(actual\_labels, predicted\_labels, average='weighted')

# Display the evaluation metrics

print("Precision:", precision)

print("Recall:", recall)

print("F1 Score:", f1\_score\_value)

# Load the pre-trained VGG16 model

vgg\_base = VGG16(weights='imagenet', include\_top=False, input\_shape=(224, 224, 3))

# Freeze all layers except the last 20 for fine-tuning

for layer in vgg\_base.layers[:-20]:

    layer.trainable = False

# Build the complete model

model = models.Sequential([

    vgg\_base,

    layers.Flatten(),

    layers.Dense(256, activation='relu'),

    layers.Dropout(0.5),

    layers.Dense(4, activation='softmax')

])

# Compile the model with a low learning rate

model.compile(optimizer=Adam(learning\_rate=5e-5),

              loss='categorical\_crossentropy',

              metrics=['accuracy'])

# Define the training data generator with augmentation

train\_augmenter = ImageDataGenerator(

    rescale=1.0 / 255,

    rotation\_range=20,

    width\_shift\_range=0.2,

    height\_shift\_range=0.2,

    shear\_range=0.2,

    zoom\_range=0.2,

    horizontal\_flip=True,

    fill\_mode='nearest'

)

# Prepare the training data loader

train\_loader = train\_augmenter.flow\_from\_directory(

    '/content/drive/MyDrive/chest\_xray/train',

    target\_size=(224, 224),

    batch\_size=32,

    classes=['NORMAL\_normalized', 'BACTERIA\_normalized', 'VIRUS\_normalized', 'tuberculosis\_Norm'],

    class\_mode='categorical',

    subset='training'

)

# Define the validation data generator

validation\_augmenter = ImageDataGenerator(rescale=1.0 / 255, validation\_split=0.2)

# Prepare the validation data loader

validation\_loader = validation\_augmenter.flow\_from\_directory(

    '/content/drive/MyDrive/chest\_xray/train',

    target\_size=(224, 224),

    batch\_size=32,

    classes=['NORMAL\_normalized', 'BACTERIA\_normalized', 'VIRUS\_normalized', 'tuberculosis\_Norm'],

    class\_mode='categorical',

    subset='validation'

)

# Early stopping callback

early\_stop = EarlyStopping(monitor='val\_accuracy', patience=10, restore\_best\_weights=True)

# Train the model

history = model.fit(

    train\_loader,

    epochs=75,

    validation\_data=validation\_loader,

    callbacks=[early\_stop],

    workers=4,

    use\_multiprocessing=True,

    max\_queue\_size=10

)

model.save('/content/drive/MyDrive/chest\_xray/Allclass.h5')

# Define the test data generator with rescaling

test\_data\_generator = ImageDataGenerator(rescale=1.0 / 255)

# Load the test data using the generator

test\_data\_loader = test\_data\_generator.flow\_from\_directory(

    '/content/drive/MyDrive/chest\_xray/test',

    target\_size=(224, 224),

    batch\_size=32,

    classes=['NORMAL\_normalized', 'BACTERIA\_normalized', 'VIRUS\_normalized', 'tuberculosis\_Norm'],

    class\_mode='categorical'

)

# Evaluate the model on the test dataset

loss, accuracy = model.evaluate(test\_data\_loader)

# Print the evaluation results

print("Test Loss:", loss)

print("Test Accuracy:", accuracy)

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Training and Validation Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

# Generate predictions for the test dataset

predictions = model.predict(test\_data\_loader)

predicted\_labels = np.argmax(predictions, axis=1)

# Extract the true labels from the test dataset

actual\_labels = test\_data\_loader.classes

# Calculate evaluation metrics

precision = precision\_score(actual\_labels, predicted\_labels, average='macro')

recall = recall\_score(actual\_labels, predicted\_labels, average='macro')

f1\_score\_value = f1\_score(actual\_labels, predicted\_labels, average='macro')

# Print the evaluation metrics

print(f"Test Precision: {precision}")

print(f"Test Recall: {recall}")

print(f"Test F1 Score: {f1\_score\_value}")

base\_path = '/content/drive/MyDrive/chest\_xray/'

model\_name ='model\_custom.h5'

#model\_name ='vggmodel.h5'

#model\_name ='chexNet.h5'

#model\_name ='CNN\_From\_Scratch\_TB\_Model.h5'

#model\_name ='inception\_tb.h5'

#model\_name ='Allclass.h5'

model = load\_model(base\_path + model\_name, compile=False)

def preprocess\_image(img\_path):

    img = image.load\_img(img\_path, target\_size=(224, 224))

    img\_array = image.img\_to\_array(img)

    img\_array = np.expand\_dims(img\_array, axis=0)

    img\_array = img\_array / 255.0

    return img\_array

img\_path = '/content/drive/MyDrive/chest\_xray/test/BACTERIA\_normalized/person78\_bacteria\_378.jpeg'

img\_preprocessed = preprocess\_image(img\_path)

prediction = model.predict(img\_preprocessed)

predicted\_class = np.argmax(prediction, axis=1)

class\_labels = {

    0: "Normal - No signs of pneumonia",

    1: "Bacterial Pneumonia Detected",

    2: "Viral Pneumonia Detected"

}

predicted\_label = class\_labels[predicted\_class[0]]

print(f"Prediction: {predicted\_label}")

predicted\_label = class\_labels[predicted\_class[0]]

plt.imshow(image.load\_img(img\_path))

plt.title(f"Predicted: {predicted\_label}")

plt.show()

probabilities = prediction[0]

for i, class\_name in class\_labels.items():

    print(f"{class\_name}: {probabilities[i]\*100:.2f}%")

predicted\_label = class\_labels[predicted\_class[0]]

print(f"\nFinal Prediction: {predicted\_label}")

def preprocess\_image(img\_path):

    img = image.load\_img(img\_path, target\_size=(224, 224))

    img\_array = image.img\_to\_array(img)

    img\_array = np.expand\_dims(img\_array, axis=0)

    img\_array = img\_array / 255.0

    return img\_array

img\_path = '/content/drive/MyDrive/TB\_Chest\_Radiography\_Database/Tuberculosis/Tuberculosis-1.png'

img\_preprocessed = preprocess\_image(img\_path)

prediction = model.predict(img\_preprocessed)

predicted\_class = np.argmax(prediction, axis=1)

class\_labels = {

    0: "Normal - No signs of Tuberculosis",

    1: "Tuberculosis Detected",

}

predicted\_label = class\_labels[predicted\_class[0]]

print(f"Prediction: {predicted\_label}")

predicted\_label = class\_labels[predicted\_class[0]]

plt.imshow(image.load\_img(img\_path))

plt.title(f"Predicted: {predicted\_label}")

plt.show()

probabilities = prediction[0]

for i, class\_name in class\_labels.items():

    print(f"{class\_name}: {probabilities[i]\*100:.2f}%")

predicted\_label = class\_labels[predicted\_class[0]]

print(f"\nFinal Prediction: {predicted\_label}")