

# **Decision Support System to Diagnose COVID-19 through Chest X-Ray Analysis**

**A.W.S.D Perera**

**2022**



# **Decision Support System to Diagnose COVID-19 through Chest X-Ray Analysis**

**A dissertation submitted for the Degree of Master of  
Business Analytics**

**A.W.S.D Perera**

**University of Colombo School of Computing**

**2022**



## DECLARATION

The thesis is my original work and has not been submitted previously for a degree at this or any other university/institute.

To the best of my knowledge, it does not contain any material published or written by another person, except as acknowledged in the text.

Student Name: A.W.S.D Perera

Registration Number: 2019/BA/026

Index Number: 19880261



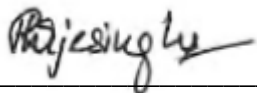
Signature:

Date: 20.11.2022

This is to certify that this thesis is based on the work of Mr. Shehan Perera under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Certified by:

Supervisor Name: C.R. Wijesinghe



Signature:

Date: 20.11.2022

I would like to dedicate this thesis to my Mother Deepthi Perera  
and to my Late Father Calistus Perera

## **ACKNOWLEDGEMENTS**

I want to express my heartfelt gratitude to my supervisor, Mrs Rupika Wijesinghe, for her invaluable guidance and support throughout this project. Furthermore, I extend my gratitude to my advisor Dr Nival Kolambage, for his advice and support throughout, especially in areas related to medicine.

Also, I would like to thank my project coordinator Dr Thilina Halloluwa for his effort in coordinating this program and all the lecturers at the University of Colombo School of Computing for enriching me with the knowledge and skills required to complete this project.

## ABSTRACT

The emergence of COVID-19 in early December 2019 has caused enormous damage to health and global well-being. It is still spreading worldwide, with some countries/ areas under lockdown. Although vaccinations for emergency use are available, no proven and tested vaccination is available in the market today to make a person entirely immune to COVID-19. Hence, continuous testing and monitoring are vital to hinder the spread of COVID-19. Today, PCR with reverse transcription (RT-PCR) is the choice test for diagnosing COVID-19. As an alternative, Rapid antigen test kits are also used. However, hospitals, especially in rural areas in countries like Sri Lanka, are deprived of these test kits. Also, except for a few countries, most people now tend to refrain from testing since they are more accustomed to COVID-19, and their fear of it has gradually decreased. Therefore, it is imperative to design an automated decision support system through different means, which can assist in providing fast decision-making with a low diagnosis error.

Chest X-ray images and Deep Learning algorithms have recently become a worthy choice for COVID-19 screening. This thesis proposes a model-based decision support system to diagnose COVID-19. It is a multi-class classification system that can classify an X-Ray DICOM (Digital Imaging and Communications in Medicine) object into one of the four COVID-19 Pneumonia classes: ‘Negative for Pneumonia’, ‘Typical Appearance’, ‘Indeterminate Appearance’ and ‘Atypical Appearance’. Furthermore, this decision support system can interpret using a heat map why a specific classification or a decision has been made.

This thesis further discusses the three main modules, or the building blocks used to develop this decision support system: Data Pre-Processing Module, Model Training Module and Model Inference Module. The Data Pre-Processing Module describes the pre-processing steps that must be applied to a DICOM object. The Model Training Module focuses on developing the best-performing model. Here the effectiveness of five pre-trained Convolutional Neural Network (CNN) models, namely Densenet121, VGG-16, ResNet-50, Inception-V3 and CheXNet, have been evaluated. The evaluation is done through comparative analysis considering several important factors such as batch size, learning rate, number of epochs, types of loss functions and optimizers. A publicly available DICOM chest X-ray dataset from Kaggle is used to validate the models, and CheXNet obtains the best performance. Finally, the Model Inference Module focuses on the Web Application developed to make inferences from Chest X-Ray images.

# TABLE OF CONTENTS

<b>LIST OF FIGURES</b> .....	8
<b>LIST OF TABLES</b> .....	10
<b>1. CHAPTER 1 INTRODUCTION</b> .....	11
<b>1.1 The Project Overview</b> .....	11
<b>1.2 Motivation</b> .....	11
<b>1.3 Objectives</b> .....	12
<b>1.4 Background of the Study</b> .....	13
<b>1.5 Scope of the Study</b> .....	14
<b>1.6 The Novelty of the Study</b> .....	14
<b>1.7 The credibility of the Dataset</b> .....	14
<b>1.8 Structure of the Dissertation</b> .....	14
<b>2. CHAPTER 2 BACKGROUND AND RELATED WORK</b> .....	15
<b>2.1 Introduction</b> .....	15
<b>2.2 Related Work</b> .....	15
<b>2.2.1 Diagnostic Models using Transfer Learning</b> .....	15
<b>2.2.2 Diagnostic Models using Conventional Convolutional Neural Networks</b> .....	17
<b>2.3 Summary of related work</b> .....	17
<b>2.4 Conclusion</b> .....	18
<b>3. CHAPTER 3 METHODOLOGY</b> .....	19
<b>3.1 Overall Design of the System</b> .....	19
<b>3.2 Data Pre-Processing Module</b> .....	20
<b>3.2.1 DICOM Metadata Analysis and Pre-Processing: Monochrome Conversion</b> .....	20
<b>3.2.2 DICOM Metadata Analysis and Pre-Processing: Image Re-Sizing</b> .....	21
<b>3.2.3 DICOM Metadata Analysis and Pre-Processing: CLAHE Conversion</b> .....	21
<b>3.2.4 OUTPUT Stage 1: Non-Standardized Image</b> .....	22
<b>3.2.5 OUTPUT Stage 2: Standardized Image</b> .....	22
<b>3.3 Model Training Module</b> .....	23
<b>3.3.1 Metadata Retrieval and Exploratory Data Analysis</b> .....	24
<b>3.3.2 Application of the Data Pre-Processing Module</b> .....	27
<b>3.3.3 Model Training</b> .....	27
<b>3.3.3.1 Train-Test-Validation Split</b> .....	27

3.3.3.2	Defining Image Generators with Data Augmentation .....	27
3.3.3.3	Model Training Architectures and Techniques to Improve Performance.....	29
3.3.4	Model Interpretation through Visualization .....	36
3.3.5	Perform Error Analysis .....	38
3.3.5.1	Establishing a Baseline.....	38
3.3.5.2	Evaluation Approach .....	38
3.4	Model Inference Module.....	43
4.	CHAPTER 4 DISCUSSION AND RESULTS .....	45
4.1	Custom CNN Model Performance.....	45
4.2	'densenet_model' Performance.....	45
4.3	'vgg16_final_model' performance .....	46
4.4	'resnet50_x_final_model' Performance .....	47
4.5	'InceptionV3_x_final_model' Performance.....	47
4.6	'chexnet_model' Performance .....	48
4.7	False Positive Vs False Negative Tradeoff in Medicine .....	50
4.8	Best Model Selection .....	52
4.9	Summary of Model Performances .....	55
5.	CHAPTER 5 CONCLUSION AND FUTURE WORK .....	57
5.1	Conclusion.....	57
5.2	Future Work.....	58
6.	LIST OF REFERENCES .....	59



## LIST OF FIGURES

Figure 1.1 Number of cases reported from June 2020 to March 2022 in Sri-Lanka.....	11
Figure 3.1 Overall Design of the System .....	19
Figure 3.2 Data Pre-Processing Module .....	20
Figure 3.3 Monochrome Conversion from MONOCHROME1 TO MONOCHROME2.....	21
Figure 3.4 Image Re-Sizing to 320 x 320 Pixels.....	21
Figure 3.5 CLAHE Applied to Enhance Visibility .....	22
Figure 3.7 Pre-processed Output at Stage 1 .....	23
Figure 3.6 Pre-Processed Output at Stage 2.....	23
Figure 3.8 Model Training Module.....	23
Figure 3.9 Distribution of Classes.....	24
Figure 3.10 Gender-wise ‘Negative for Pneumonia’ and ‘Typical Appearance’ Class Label Distributions .....	25
Figure 3.11 Photometric Interpretation of the Dataset .....	26
Figure 3.12 Image Resolution Distribution of Images .....	26
Figure 3.13 Data Augmentation applied on the Training Set.....	28
Figure 3.14 Train Data Generator .....	28
Figure 3.15 Custom CNN Architecture.....	29
Figure 3.16 Custom CNN Model Python Implementation.....	30
Figure 3.17 densenet_model Python Implementation.....	31
Figure 3.18 vgg16_final_model Python Implementation.....	32
Figure 3.19 resnet50_x_final_model Python Implementation .....	32
Figure 3.20 InceptionV3_x_final_model Python Implementation.....	33
Figure 3.21 chexnet_model Python Implementation .....	34
Figure 3.22 Weighted Cross-Entropy Loss Function.....	35
Figure 3.23 Saving PNG with different Color Maps.....	36
Figure 3.24 Image Interpretation with GRADCAM .....	37
Figure 3.25 GRADCAM Heatmap Python Implementation .....	37
Figure 3.26 'get_performance' Function Output.....	39
Figure 3.27 'get_performance' Python Implementation .....	39
Figure 3.28 'plot_confusion_matrix' Function .....	40
Figure 3.29 'plot_confusion_matrix' Python Implementation.....	41
Figure 3.30 'get_roc_curve' Function .....	42
Figure 3.31 'get_roc_curve' Python Implementation.....	42
Figure 3.32 Model Inference Module .....	43
Figure 3.33 Gradio Application Development .....	44
Figure 3.34 Huggingface Application.....	44
Figure 4.1 Custom Model Performance on the Training Set .....	45
Figure 4.2 Training and Validation Set Accuracy - densenet_model .....	46
Figure 4.3 'get_performance' report - densenet_model on the Training Set .....	46
Figure 4.4 Test Set Performance of ‘vgg16_final_model’ trained with 'weighted_crossentropy' loss function...	47
Figure 4.5 Test Set Performance of ‘resnet50_x_final_model' .....	47
Figure 4.6 Validation Accuracy Reduction with Time – Inception_x_final_model .....	48
Figure 4.7 Test Set Performance of 'InceptionV3_x_final_model' .....	48
Figure 4.8 Validation Accuracies of Non-Flattened Vs Flattened Architectures.....	49
Figure 4.9 Image Re-scaling in Image Generator .....	49
Figure 4.10 Test Set Performance of 'chexnet_model' trained on re-scaled data.....	50

Figure 4.11 Confusion Matrix with more False Positives.....	51
Figure 4.12 Confusion Matrix with more False Negatives .....	52
Figure 4.13 Performance metrics of the best model.....	53
Figure 4.14 ROC Curves of the Best Model .....	54
Figure 4.15 Confusion Matrix of the Final Model .....	54

## LIST OF TABLES

Table 1.1 COVID-19 Pneumonia Classification.....	13
Table 2.1 Comparison between different related works.....	17
Table 4.1 Model Performance Comparison on Test Set .....	56

# 1. CHAPTER 1 INTRODUCTION

## 1.1 The Project Overview

The pandemic associated with coronavirus is now considered one of the deadliest epidemics creating havoc on the health and financial systems of the world. The numbers of COVID-19 cases exponentially increase with no specific treatment. Although vaccines are now available for emergency use, more is needed to guarantee protection against COVID-19 fully.

Far deadlier than the flu, COVID-19 causes significant morbidity and mortality. The Figure 1.1 shows that the fatality rate in Sri Lanka and the daily number of reported cases of COVID-19 show an increasing trend from October 2020 until August 2021 and then a decreasing trend. However, recorded numbers only partially justify the current situation since people are generally reluctant to test.

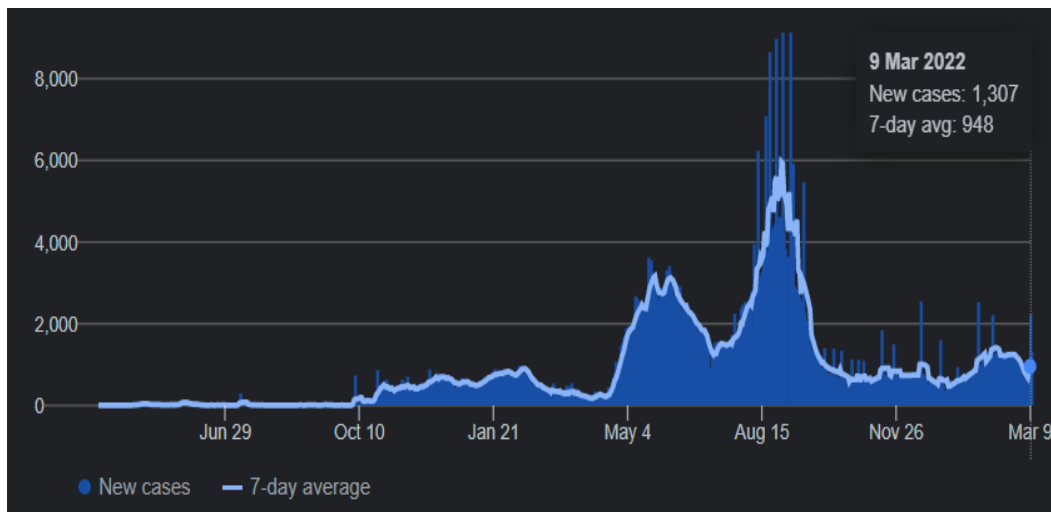


Figure 1.1 Number of cases reported from June 2020 to March 2022 in Sri-Lanka

In Sri Lanka, many people die in their own houses, and the post-mortem reports suggest that many people die of COVID-19. This would have been a different story had the patients been identified earlier and proper medical treatment was provided before the condition worsened. Therefore, continuous testing and monitoring are vital to hinder the spreading of COVID-19 and a need for a cheap, feasible, and, more importantly, fast diagnosis mechanism arises.

## 1.2 Motivation

Today, PCR with reverse transcription (RT-PCR) is the choice test for diagnosing COVID-19. However, it can take a few hours and sometimes days before the molecular test results arrive. By the time we

identify some patients to be COVID-19 positive, it might be too late for them. As an alternative solution, rapid antigen test kits to detect COVID-19 are available on the market today. The rapid antigen test yields the test result in minutes. However, they tend to have a high false-positive rate (Gans *et al.*, 2022).

Furthermore, these conventional diagnosis processes cause an increased risk for medical staff due to direct exposure. Moreover, these tests are costly, and the limited availability of diagnostic test kits has become a problem, especially in rural areas in Sri Lanka. Despite the drawbacks, these two methods are the widely used techniques to diagnose COVID-19.

Also, except for a few countries, most people now tend to refrain from testing since they are more accustomed to COVID-19, and their fear of it has gradually decreased. Therefore, designing an automated decision support system through separate means is essential, which can assist in providing fast decisions and significantly reduce diagnosis errors. As a solution, imaging (chest radiograph (CXR), Chest CT Scans) can be used to achieve greater diagnostic certainty (Gans *et al.*, 2022). These techniques are relatively safe, faster, and easily accessible. Therefore X-ray imaging has been used extensively for COVID-19 screening as it necessitates lower cost and less imaging time, and X-ray scanners are widely available even in rural areas (Nayak *et al.*, 2021).

However, it is a cumbersome and time-consuming task for radiologists to inspect X-ray images visually. Moreover, it may lead to erroneous diagnoses due to a lack of prior knowledge about the virus-infected regions. Thus, there is a robust requirement for the design of automated methods to enable a quick and accurate COVID-19 diagnosis.

There have been many attempts to develop models to diagnose COVID-19 using imaging. Nevertheless, their readiness to deploy clinically is a question, mainly due to methodological flaws and underlying biases in data (AIX-COVNET *et al.*, 2021). Furthermore, most research in this area does not contain a visualization component for interpretability.

### **1.3 Objectives**

It is very convenient to get a chest radiograph in Sri Lanka, irrespective of the area a person lives. However, it is a complex task to diagnose COVID-19 using chest radiographs (CXR) visually. Like other cases of Pneumonia, inflammation and fluid in the lungs occur as a result of pulmonary infection with COVID-19. This appears very similar to other bacterial and viral types of Pneumonia on chest

radiographs (CXR) (Simpson *et al.*, 2020), making it difficult to diagnose visually. The COVID-19 Pneumonia imaging classification (Simpson *et al.*, 2020) used in medicine is depicted in Table 1.1 below.

Table 1.1 COVID-19 Pneumonia Classification

	<b>Class Label</b>	<b>Rationale</b>
1	‘Typical Appearance’	Commonly reported imaging features of greater specificity for COVID-19 Pneumonia.
2	‘Indeterminate Appearance’	Nonspecific imaging features of COVID-19 Pneumonia.
3	‘Atypical Appearance’	Uncommonly or not reported features of COVID-19 Pneumonia.
4	‘Negative for Pneumonia’	No features of Pneumonia

This project aims to build a model-based decision support system based on a multi-class classification model using chest radiographs, which predicts the probabilities for each of the four classes mentioned above. This model will have enhanced interpretability with a heat map overlaying the image, highlighting why a specific decision has been made.

If successful, this will be particularly useful to people who need to be more privileged to find COVID-19 test kits at their convenience. Furthermore, many people obtain Chest X-Rays for different requirements. This decision support system can be used to gain insight into COVID-19 Pneumonia upon their consent. In any case, this would assist doctors in seeing the severity of the disease and making decisions regarding treatment, such as hospitalization, admission into an intensive care unit (ICU), or supportive therapies like mechanical ventilation. As a result, more patients will quickly receive the best care for their condition, which could mitigate the COVID-19 spread and fatality rate in Sri Lanka. Furthermore, this decision support system provides an opportunity to make inferences about the whole population and on what level COVID-19 is spread in society.

## **1.4 Background of the Study**

Since the pandemic began in early 2020, researchers have put forward numerous machine-learning models for diagnosing COVID-19 using Chest X-Rays (CXR). A comprehensive study of the background and related work is presented in Chapter 2.

## **1.5 Scope of the Study**

The output of this project is a model-based decision support system based on a multi-class classification model using chest radiographs, which predicts the probabilities associated with each of the four classes: ‘Typical Appearance’, ‘Indeterminate Appearance’, ‘Atypical Appearance’, and ‘Negative for Pneumonia’. This model will have an enhanced interpretability function through visualization using a heat map highlighting why a particular decision has been made. The front end of the decision support system is deployed as a web application.

## **1.6 The Novelty of the Study**

This study aims at building a decision support system to assist in diagnosing COVID-19, using labelled training images verified by an experienced set of radiologists, compared to many unverified public datasets used in many of the studies, as stated in detail in Chapter 2.2. This decision support system attempts to classify an X-Ray image into the four standard COVID-19 pneumonia classes, whereas all the other researchers studied used different class labels. Furthermore, the model performance will be driven mainly by PPV (Positive Predicted Value) and NPV (Negative Predicted Value). Moreover, an explainability component will be added to the final output, highlighting why a particular classification has been made. This further enhances the trustworthiness of the model.

## **1.7 The credibility of the Dataset**

A publicly available labelled input dataset is used for this project. The Society for Imaging Informatics in Medicine (SIIM) has partnered with the Foundation for the Promotion of Health and Biomedical Research of Valencia Region (FISABIO), Medical Imaging Databank of the Valencia Region (BIMCV), and the Radiological Society of North America (RSNA) for a Kaggle competition. They have put forward a dataset of 6,334 chest radiographs labelled by a panel of experienced radiologists for the presence of opacities and overall appearance. The images in the Dataset are in DICOM (Digital Imaging and Communications in Medicine) format, the international standard for transmitting, storing, retrieving, processing, and displaying medical imaging information. Hence, this data set can be considered a credible data set.

## **1.8 Structure of the Dissertation**

The dissertation's structure includes the chapters Background and Related Work, Methodology, Discussion and Results, and References.

## **2. CHAPTER 2 BACKGROUND AND RELATED WORK**

### **2.1 Introduction**

Since the pandemic began in early 2020, researchers have developed numerous machine-learning models for diagnosing COVID-19 using Chest X-Rays (CXR). Most experiments have used public datasets in which a responsible governing body has not verified the chest X-rays. Furthermore, critical performance metrics such as PPV and NPV have not been calculated in any of the studies analyzed so far. PPV indicates the probability that a person has the disease given that the model predicts positive, and NPV suggests the probability that a person does not have the disease given that the model predicts negative. These two are critical metrics in this use case.

Moreover, most of the literature focuses only on performance, whereas explainability is essential in the majority of Machine Learning/ Deep Learning applications. It is critical to highlight the area in the image which contributed to the decision-making. The following section describes past work that has been done related to diagnosing COVID-19 using Chest X-Rays.

### **2.2 Related Work**

This section describes related work from 2020 to 2022 in chronological order. Six papers published in 2020, two published in 2021, and one published in 2022 have been considered for this literature review. The literature set is further segregated into two subdomains: literature that uses transfer learning approaches and conventional Convolutional Neural Networks (CNNs).

#### **2.2.1 Diagnostic Models using Transfer Learning**

In 2020, Tartaglione et al. (Tartaglione *et al.*, 2020) developed a ResNet-50-based CNN to classify if a person is COVID-19 Positive and COVID-19 negative. However, the number of images used for training and testing is less than 500 images, and the underlying biases of the data are unknown. Furthermore, model evaluation needs to be more precise in the paper. Ghoshal and Tucker (Ghoshal and Tucker, 2020) have also used a deep learning approach based on the ResNet-50 model to classify a CXR of a person into four categories; COVID-19, non-COVID-19 viral Pneumonia, non-COVID-19 bacterial Pneumonia, and Normal. The Dataset contained more than 5000 images, but it is highly imbalanced, with less than 60 images with COVID-19. Furthermore, this Dataset had CXRs of pediatric patients



making it biased. The paper had an unclear validation procedure, and the model evaluation was also not precise.

Rahaman et al. (Rahaman *et al.*, 2020) used a transfer learning approach using the VGG-19 model to classify CXR s into normal, COVID-19, and Non-COVID-10 Pneumonia. The model used less than 1000 images with a high-class imbalance and performed internal holdout validation. The model yielded an Accuracy= 0.89, Precision = 0.90, Recall = 0.89 and F1-Score = 0.90. However, the data used for this analysis is insufficient, and significant metrics such as PPV and NPV have not been evaluated.

Tsiknakis et al. (*Interpretable artificial intelligence framework for COVID-19 screening on chest X-rays*, no date) used a model based on the Inception model, another transfer learning approach to classify a CXR into four categories as COVID-19, non-COVID-19 viral Pneumonia, non-COVID-19 bacterial Pneumonia, and normal. The model used about 700 images of CXRs with five-fold internal cross-validation. The model yielded an AUC =1.0, Accuracy= 1.0, Sensitivity = 0.99 and Specificity = 1.0. However, there is a question about the quality of the public datasets used in this analysis. Furthermore, PPV and NPV metrics have not been calculated.

Luz et al. (Luz *et al.*, 2021) also utilized a transfer learning approach using the EfficientNet. This model classifies a CXR into three categories; COVID-19, Non-COVID-19 Pneumonia, and Normal. The authors used more than 13000 images and performed internal holdout validation. The model yielded an accuracy = 0.94, Sensitivity = 0.97 and PPV = 1.00. However, the test set size of 231 images is insufficient to justify the model's performance. Moreover, NPV or the Negative Predicted Value and Specificity have not been calculated.

In 2021, Zhang et al. (Zhang *et al.*, 2021) used Transfer Learning using the Dense Net-121 to classify if a patient has COVID-19 Pneumonia or non-COVID-19 Pneumonia. Five thousand eight hundred six chest radiographs with COVID-19 Pneumonia and 5300 chest radiographs with non-COVID-19 Pneumonia were included and split into training, validation, and test data sets. This model used internal holdout validation yielding AUC = 0.92, Sensitivity = 0.88 and Specificity = 0.79. However, PPV and NPV, which are more important metrics than Sensitivity and Specificity, have not been calculated in this study.

Jain et al.(Jain *et al.*, 2021) attempted a classification problem with three classes; COVID-19 affected and Healthy and Pneumonia. They have used deep learning-based CNN models with transfer learning,

compared Inception V3, Xception, and ResNeXt models and examined their accuracy. The Xception model yielded the highest accuracy of 0.98. However, comparing only the accuracy in medical applications is not suitable due to the high class imbalance in reality.

### 2.2.2 Diagnostic Models using Conventional Convolutional Neural Networks

In 2022, Moura et al.(de Moura, Novo and Ortega, 2022) proposed deep learning approaches for the classification of chest X-ray images under the analysis of 3 different categories: Covid-19, Pneumonia, and healthy cases. A densely convoluted neural network architecture is used in this experiment. The authors adopt four approaches in classifying the CXR images; Healthy vs Pneumonia, tested with Covid-19, Healthy vs Pneumonia/Covid-19 Healthy/Pneumonia vs Covid-19, and Healthy vs Pneumonia vs Covid-19. The fourth approach yields excellent performance with F1-score = 0.99, but the Dataset used is only 621 X-ray radiographs. A comprehensive summary of the literature is presented in Table 2.1.

### 2.3 Summary of related work

Table 2.1 Comparison between different related works

Reference	Published Year	Predictors	Target Variables	Development Sample Size	Test Sample Size	Model Performance	Identified Issues
Tartaglione et al.	2020	Deep Learning (CNN)	Covid Positive, Covid Negative	231 images, including 126 COVID-19 images	135 images with 90 COVID-19 images	Not Clear	Performance metrics are not evident in the paper
Ghoshal and Tucker	2020	Deep Learning (Transfer Learning)	COVID-19, non-COVID-19 viral pneumonia, non-COVID-19 bacterial pneumonia and Normal	4,752 images including 54 COVID-19 images	1,189 images, 14 COVID-19 images	Not Clear	Performance metrics are not evident in the paper
Rahaman et al.	2020	Deep Learning (Transfer Learning)	Normal, COVID-19, and Non-COVID-10 Pneumonia	720 images, 220 COVID-19	140 images, 40 COVID-19	Accuracy = 0.89 Precision = 0.90 Recall = 0.89 F1 score = 0.90	Data is not sufficient. Furthermore, NPV and PPV are not calculated
Tsiknakis et al.	2020	Deep Learning (Transfer Learning)	COVID-19, non-COVID-19 viral pneumonia, non-COVID-19 bacterial	458 (CV) images, 98 COVID-19	114 (CV) images, 24 COVID-19	AUC = 1.00 Accuracy = 1.00 Sensitivity = 0.99	Data is not sufficient. Furthermore, NPV and PPV are not calculated

			pneumonia, and Normal			Specificity = 1.00	
Luz et al.	2020	Deep Learning (Transfer Learning)	COVID-19, Non-COVID-19 Pneumonia, and Normal	13,569 images, 152 COVID-19	231 images, 31 COVID-19	Accuracy = 0.94 Sensitivity = 0.97 PPV = 1.00	The test set size is not sufficient. Also, Specificity and NPV are not calculated.
Zhang et al.	2021	Deep Learning (Transfer Learning)	COVID-19 pneumonia , Non-COVID-19 pneumonia	5236 images, including 2582 COVID-19 images	5,869 images with 3,223 COVID-19	AUC = 0.92 Sensitivity = 0.88 Specificity = 0.79	PPV and NPV not calculated
Jain et al.	2021	Deep Learning (Transfer Learning)	COVID-19 affected, Healthy and Pneumonia.	5467 images	965 images	Accuracy = 0.97	Accuracy is not a good measure for medical applications with a high class imbalance.
De Moura et al.	2022	Deep Learning (CNN)	Covid-19, Pneumonia, and Healthy	621 X-ray radiographs ( 207 each for Healthy, Pneumonia, and COVID-19	Not Clear	Precision >= 0.98 Recall >= 0.98 F1 score >= 0.99 for all the three classes	Data size is not sufficient

In a nutshell, ResNet and DenseNet architectures showed better performance than the others, with accuracies ranging from 0.88 to 0.99 (AIX-COVNET *et al.*, 2021). However, direct comparisons across papers cannot be made because of different input datasets and class variables.

## 2.4 Conclusion

Since 2020, many papers have been put forward to diagnose COVID-19 through Chest X-Rays. Nevertheless, their readiness to deploy clinically is a question, mainly due to methodological flaws and underlying biases in data (AIX-COVNET *et al.*, 2021). These include using unverified public datasets with inherent biases, failing to compute essential performance metrics such as PPV and NPV, and failing to include an explainability component in the final models to enhance the trustworthiness of the model. The following section presents a comprehensive solution that addresses the research gaps mentioned above.

### 3. CHAPTER 3 METHODOLOGY

#### 3.1 Overall Design of the System

The Figure 3.1 illustrates the overall design and the workflow of the project. The final architecture consists of three modules:

1. Data Pre-Processing Module
2. Model Training Module
3. Model Inference Module

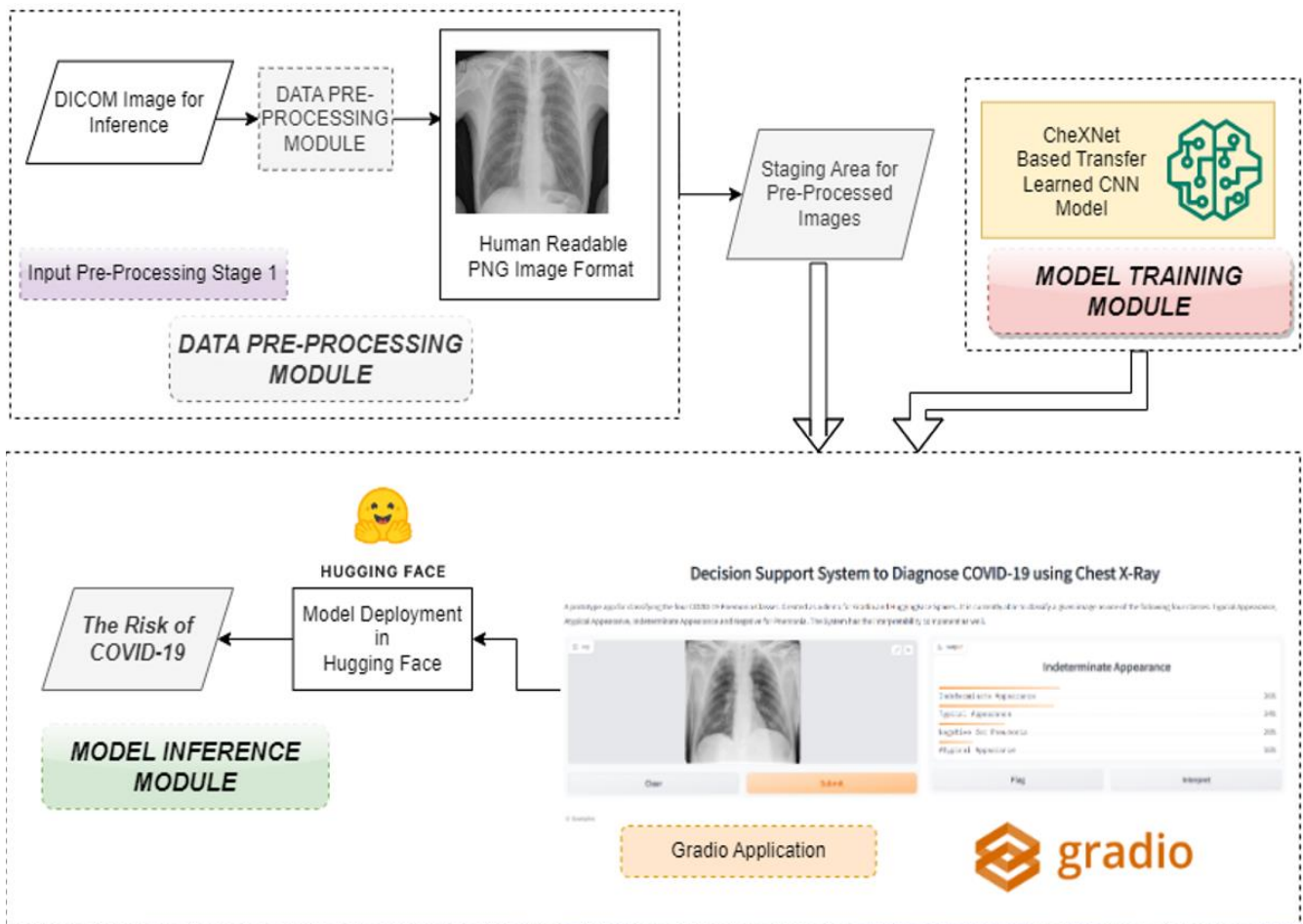


Figure 3.1 Overall Design of the System

The following sections 3.2, 3.3 and 3.4 elaborate on the three modules in detail.

## 3.2 Data Pre-Processing Module

**INPUT:** DICOM Image

**OUTPUT:** PNG Image or Standardized PNG Image

The Data Pre-Processing Module is depicted in Figure 3.2. It takes a DICOM Image as Input, applies three pre-processing techniques and is capable of giving the PNG Image output in two formats: Non-Standardized and Standardized. These are referred to as Stage 1 and Stage 2.

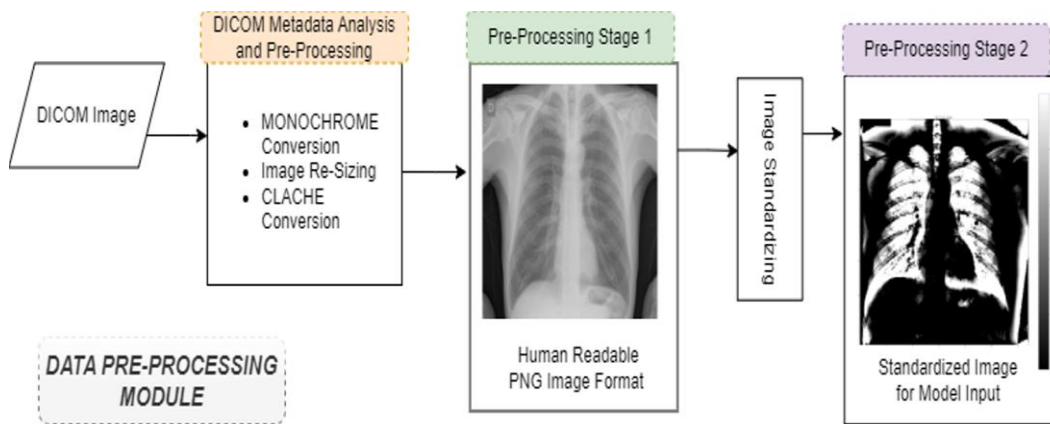


Figure 3.2 Data Pre-Processing Module

The three mandatory pre-processing techniques applied to a DICOM Image are:

1. Monochrome Conversion
2. Image Re-Sizing
3. CLAHE Conversion

### 3.2.1 DICOM Metadata Analysis and Pre-Processing: Monochrome Conversion

The metadata 'Photometric Interpretation' of a DICOM image is observed to decide on Monochrome Conversion. There are two 'Photometric Interpretation' types: MONOCHROME1 and MONOCHROME2. MONOCHROME1 indicates that the greyscale ranges from bright to dark with ascending pixel values, whereas MONOCHROME2 ranges from dark to bright with ascending pixel values.

If the photometric interpretation of a DICOM Image is MONOCHROME1, it is converted to MONOCHROME2. If it is MONOCHROME1, then no Monochrome Conversion is done. This ensures

that all the images follow the same photometric interpretation of MONOCHROME2. The Figure 3.3 shows the same image in MONOCHROME1 and MONOCHROME2.

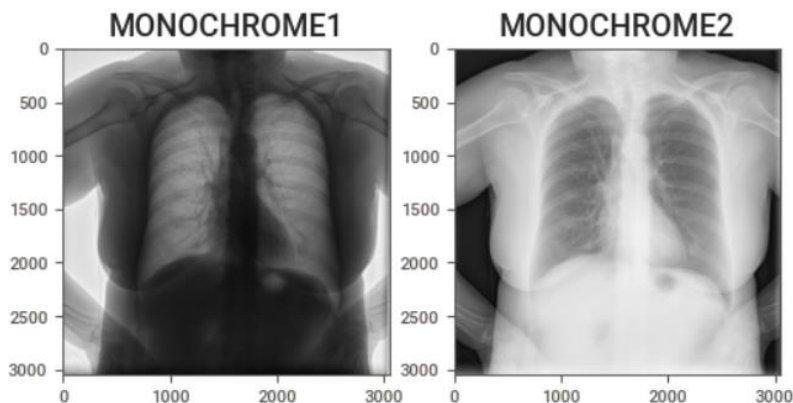


Figure 3.3 Monochrome Conversion from MONOCHROME1 TO MONOCHROME2

### 3.2.2 DICOM Metadata Analysis and Pre-Processing: Image Re-Sizing

All images are re-sized to 320 x 320 pixels, irrespective of their original size. This ensures that all the input images follow the same size and that the input pixel size is convenient for model training and inference. The Figure 3.4 depicts an image of size 3000 x 3000 re-sized to 320 x 320.

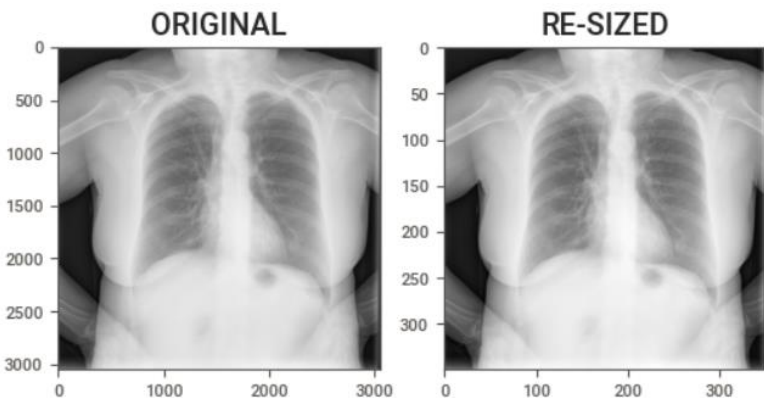


Figure 3.4 Image Re-Sizing to 320 x 320 Pixels

### 3.2.3 DICOM Metadata Analysis and Pre-Processing: CLAHE Conversion

Contrast Limited Adaptive Histogram Equalization (CLAHE) (Reza, 2004) is used to improve the visibility level of a foggy image by amplifying the contrast. CLAHE operates on small regions in the image, called tiles, rather than the entire image. The neighboring tiles are combined using bilinear interpolation to remove the artificial boundaries. The Figure 3.5 illustrates how the original image is enhanced with CLAHE.

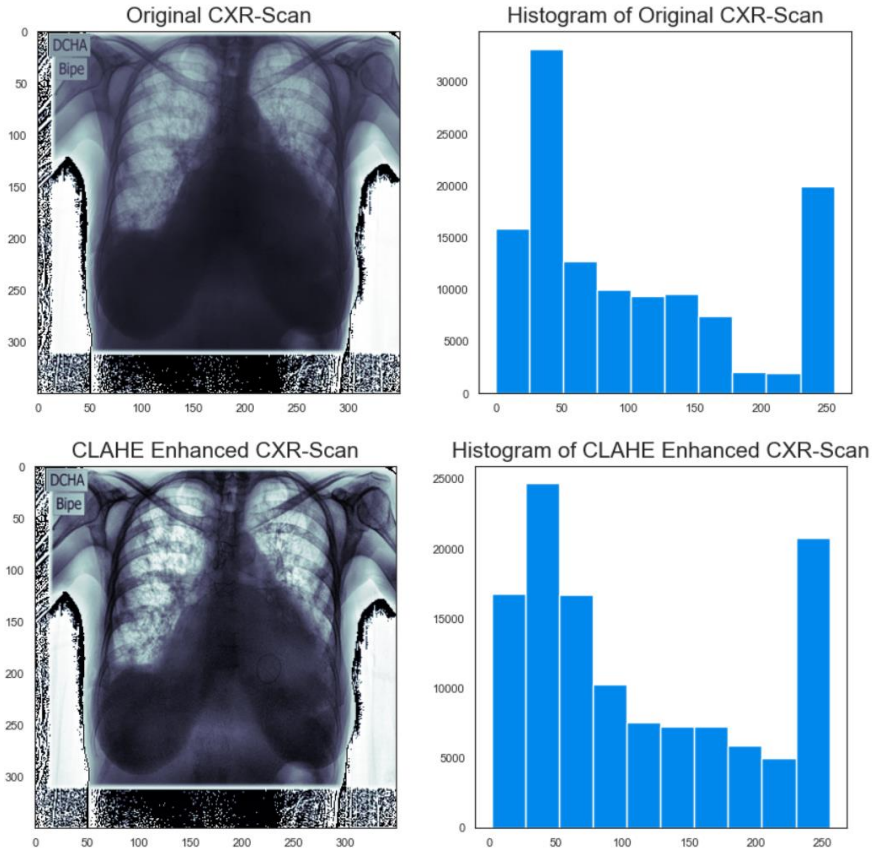


Figure 3.5 CLAHE Applied to Enhance Visibility

### 3.2.4 OUTPUT Stage 1: Non-Standardized Image

This is the first option of the two PNG image output options in the Image Processing Module. This is the stage where an image is easily identifiable for a Human. In the Model Inference Module, Pre-processing is initially done only up to Stage 1.

### 3.2.5 OUTPUT Stage 2: Standardized Image

This is the second option of the two PNG image output options in the Image Processing Module. This is the stage where an image is standardized over the output of Stage 1. A comparison of Stage 1 and Stage 2 is illustrated in Figures 3.6 and 3.7. Image standardizing makes convergence faster while training a model



Figure 3.7 Pre-processed Output at Stage 1

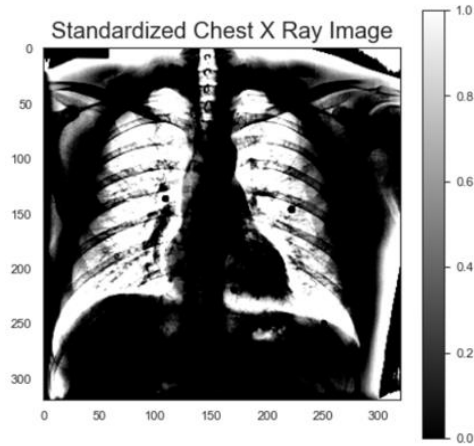


Figure 3.6 Pre-Processed Output at Stage 2

In the Model Training Module depicted in Figure 3.8, Pre-processing is done up to Stage 2 to get Standardized Images. However, in the Model Inference Module, Pre-processing is initially done only up to Stage 1.

### 3.3 Model Training Module

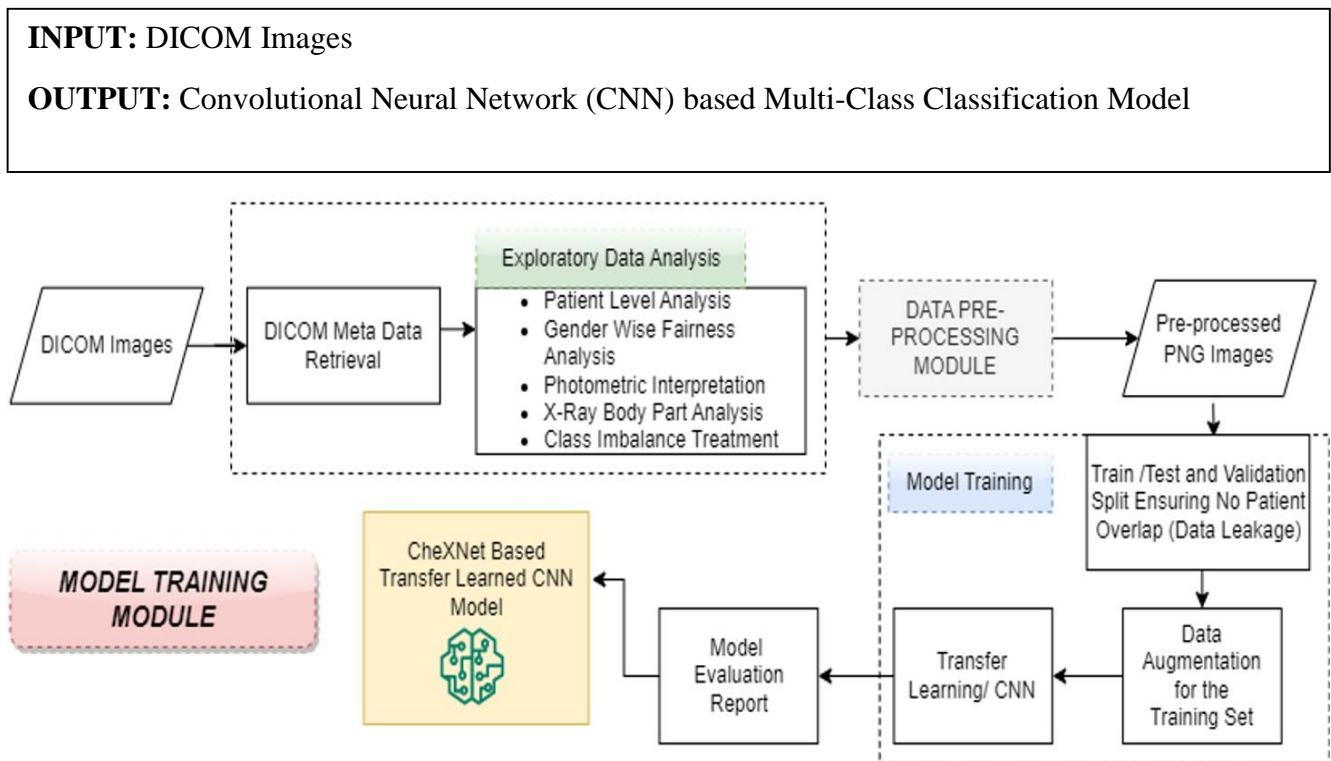


Figure 3.8 Model Training Module



### 3.3.1 Metadata Retrieval and Exploratory Data Analysis

An Exploratory Data Analysis (EDA) was performed to identify distributions of data and underlying biases in data and to analyze the data availability for each class variable.

There are 6334 images in this Dataset, where 6054 images can be used for training and validation, while the remaining 280 images can only be used for model evaluation in Kaggle. All the images are in DICOM (Digital Imaging and Communications in Medicine) format. A single DICOM file contains a header (which stores metadata about the patient and the image) and the image data, which can contain information in three dimensions.

The python libraries ‘pydicom’ and ‘os’ were used to traverse the image folders and read the DICOM files. Of the 6054 images, 232 returned errors, leaving only 5822 images for training and validation. During this process, the metadata collected from every image and analyzed are ‘Patient id’, ‘Gender’, ‘Photometric Interpretation’, ‘Pixel width’, ‘Pixel height’, and ‘Body part’. In addition, the image data and the class variables were also analyzed.

#### 3.3.1.1 Class Variable Distribution

As stated in Chapter 1 and shown in Figure 3.9, this Dataset has four target variables. They are ‘Typical Appearance’, ‘Indeterminate Appearance’, ‘Atypical Appearance’, and ‘Negative for Pneumonia’. The following figure illustrates the distribution of class variables in the Dataset of 5822 images.

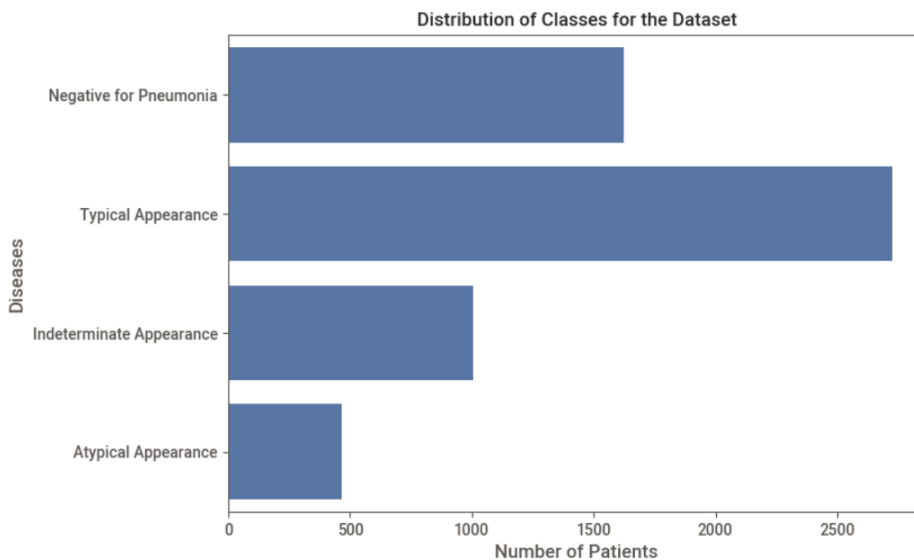


Figure 3.9 Distribution of Classes

The Dataset is imbalanced. The class variable ‘Atypical Appearance’ has the lowest prevalence, indicating the highest imbalance.

### 3.3.1.2 Metadata: Patient ID

Out of the total of 5822 patient IDs, there are only 3198 (55%) distinct values. This indicates that there are only 3198 unique patients, and multiple X-rays of the same patient are included in the Dataset. This is essential when splitting the input data for training and validation sets to avoid data leakage. For patients with multiple X-rays, it should be made sure that they do not show up in both the training and validation sets in order to avoid data leakage.

### 3.3.1.3 Metadata: Gender

Fifty-five percent (55%) of the complete X-Rays belong to Males, and the remaining 45% belong to Females. Gender-wise class variable distribution was also analyzed to see if there was any gender preference for the class labels. Gender fairness assessment is vital since female X-Rays may contain breast shadows, and unbalanced proportions in male and female X-Rays may lead to incorrect model learning during the training process.

Gender-wise distributions for the class labels ‘Negative for Pneumonia’, ‘Indeterminate Appearance’, and ‘Atypical Appearance’ showed approximately similar proportions. However, there is a significantly higher proportion of ‘Male’ X-Rays with ‘Typical Appearance’ as shown in Figure 3.10 below.

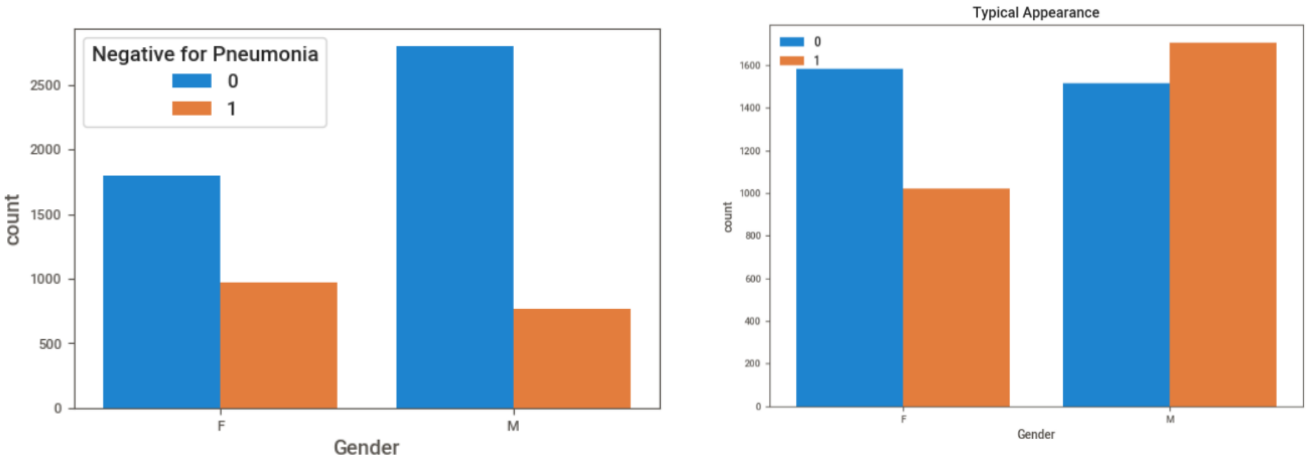


Figure 3.10 Gender-wise ‘Negative for Pneumonia’ and ‘Typical Appearance’ Class Label Distributions

### 3.3.1.4 Metadata: Photometric Interpretation

The photometric interpretation of the X-Ray images was either ‘MONOCHROME1’ or ‘MONOCHROME2’. As shown in Figure 3.11, 73% of the Dataset are MONOCHROME2 images.

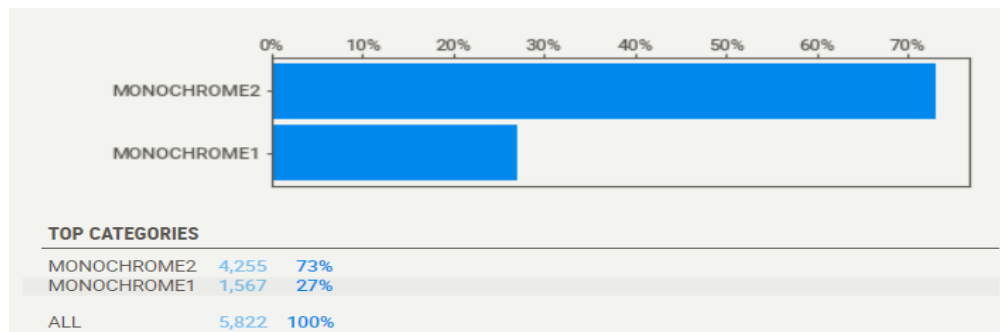


Figure 3.11 Photometric Interpretation of the Dataset

MONOCHROME1 indicates that the greyscale ranges from bright to dark with ascending pixel values, whereas MONOCHROME2 ranges from dark to bright with ascending pixel values. This indicates that the pixel values in MONOCHROME1 images are reversed, and this is addressed in the Image Pre-Processing Module.

### 3.3.1.5 Image Resolution (Pixel width and Pixel height)

The pixel width and the pixel height of images are not constant. Most images have pixel width and height between 2000 to 3500 pixels as depicted in Figure 3.12. However, the images need to be scaled down to a constant pixel width and height. This constant should be a low value, such as 320 x 320 pixels, since having higher resolutions would severely increase the processing power required to train this deep-learning model. Image re-sizing is handled from the Image Pre-Processing Module.

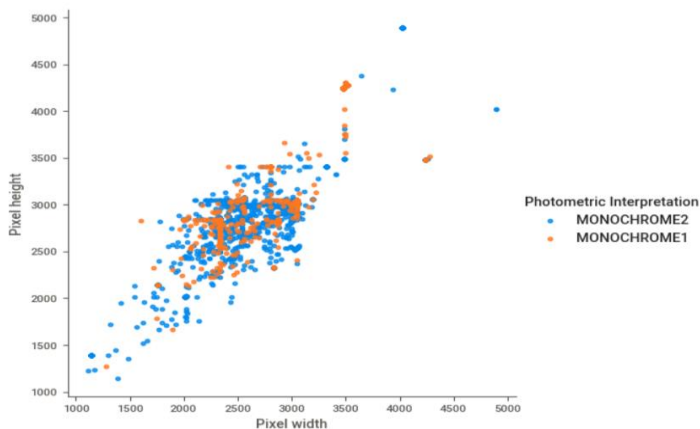


Figure 3.12 Image Resolution Distribution of Images

### 3.3.2 Application of the Data Pre-Processing Module

The DICOM images are processed using the Data Pre-Processing Module by explicitly using Monochrome Conversion, Image Re-Sizing, CLAHE Conversion and Image Standardizing.

### 3.3.3 Model Training

#### 3.3.3.1 Train-Test-Validation Split

In medical use cases, it is vital to avoid data leakage by ensuring the train, validation and test sets contain only images of unique patients to each set that don't overlap. In other words, this avoids training on a patient's data point and predicting on the same patient's data point.

The entire set of 6334 images belonging to 3261 patients was divided using the following mechanism to achieve this.

**Test Set** → A random selection of 14% of unique patient data was allocated to the Test Set. Precisely, 882 images belonging to 451 patients were allocated to the Test Set.

**Validation Set** → A random selection of 14% of unique patient data after removing the Test Set patients, was allocated to the Validation Set. 894 images belonging to 464 patients were allocated to the Validation Set.

**Training Set** → The remaining 72% of unique patient data was allocated to the Training Set. Precisely, 4558 images belonging to 2346 patients were allocated to the Training Set.

#### 3.3.3.2 Defining Image Generators with Data Augmentation

Image generators generate batches of tensor image data with real-time data augmentation. However, data augmentation is only applied to the Training set on a real-time basis when training the model. It is not applied to the validation or test set during inference.

The Image generator 'datagen\_train' was created as below. It standardizes an image by setting the mean to zero and the standard deviation to one, executes small random rotations to the image, performs minute random horizontal (left or right) shifts and vertical (up or down) shifts to the image, varies the brightness level of the image randomly within a range and zooms in or out by a small degree. During data

augmentation, it fills the points outside the input boundaries by the nearest pixel values. The Python implementation of the Data Augmentation is given in the Figure 3.13.

```
datagen_train = ImageDataGenerator(  
    samplewise_center=True, #Set each sample mean to 0.  
    samplewise_std_normalization=True, # Divide each input by its standard deviation  
    rotation_range=6,  
    width_shift_range=0.2,  
    height_shift_range=0.2,  
    brightness_range=[0.2,1.0],  
    #shear_range=0.2,  
    zoom_range=0.02,  
    #horizontal_flip=True,  
    fill_mode='nearest')
```

Figure 3.13 Data Augmentation applied on the Training Set.

The training generator is defined using the 'datagen\_train' image generator and the python code for the training generator is given in Figure 3.14 below.

```
# Flow from dataframe with specified batch size and target image size  
train_generator_aug = datagen_train.flow_from_dataframe(  
    dataframe=train_df,  
    directory=image_dir,  
    x_col=x_col,  
    y_col=y_cols,  
    classes = ['Negative for Pneumonia', 'Typical Appearance',  
              'Indeterminate Appearance', 'Atypical Appearance'],  
    class_mode="categorical",  
    batch_size=batch_size,  
    shuffle=shuffle,  
    seed=seed,  
    target_size=(320,320) # width and height of output image  
)
```

Figure 3.14 Train Data Generator

The training data generator normalizes each image per batch, meaning that it uses batch statistics. This should not be done with the test and validation data since, in real life, incoming images are processed one at a time, not batch at a time. Knowing the average per batch of test data would effectively give the model an advantage. The model should not have any information about the test data. Therefore, it is mandatory to normalize incoming test and validation data using the statistics computed from the training set. Hence, separate generators are required for validation and test sets.

Ideally, to calculate the statistics from the training set, the sample mean and standard deviation should be computed using the entire training set. However, since this is considerably large, that would be very time-consuming. In the interest of time, a random sample of the training dataset was taken to calculate the sample mean and sample standard deviation. Another image generator was fit to this sample data and was used to define the validation and test data generators.

### 3.3.3.3 Model Training Architectures and Techniques to Improve Performance

#### 3.3.3.3.1 Convolutional Neural Network (CNN) Architectures

Six types of CNN-based models were trained to solve this medical problem. The first model consists of a custom CNN architecture, while the other five are based on Transfer Learning from Densenet 121, VGG 16, Resnet 50, InceptionV3 and ChexNet.

##### 3.3.3.3.1.1 Custom Convolutional Neural Network (CNN) Architecture

A Custom CNN Architecture was developed as shown in Figure 3.15 below. The model consists of six Convolutional blocks with 16 filters each of size 3x3 with a stride 1, Six Batch Normalization layers, Six Max Pooling layers of filter size 2x2 with a stride 1, a Flattening layer, a Dropout layer with a dropout rate of 0.4 and two Dense layers. The final dense layer has four neurons with a Softmax activation function.

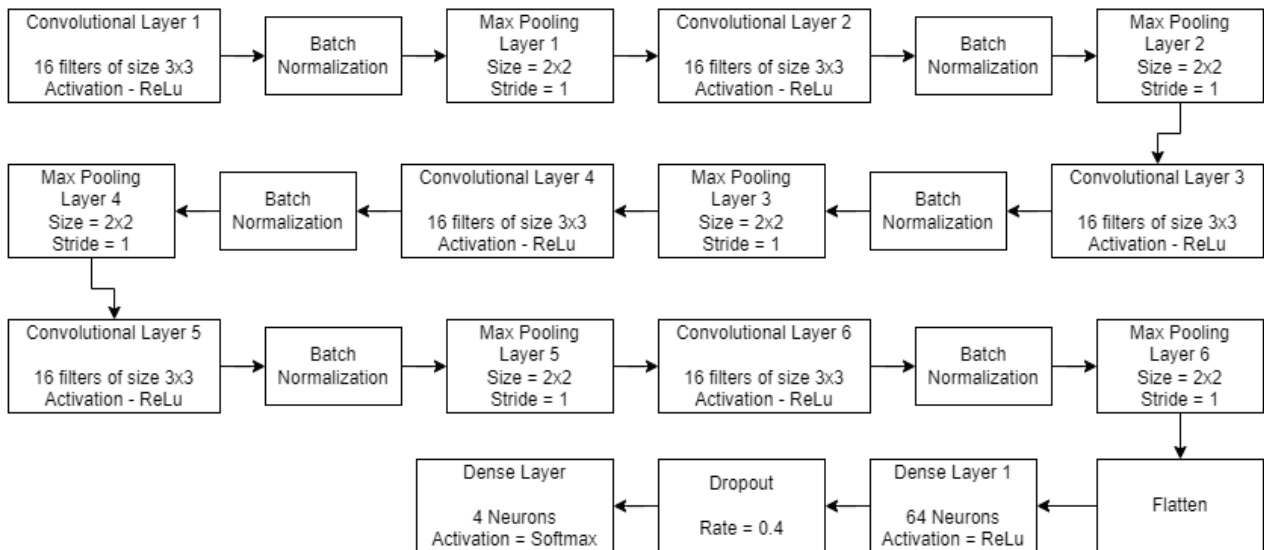


Figure 3.15 Custom CNN Architecture

A Convolutional layer is the primary building block of a CNN. It contains a set of filters (or kernels), the parameters of which are to be learned throughout the training. Convolution layers are used to extract the features from input training samples. Each convolution layer has a set of filters that helps in feature extraction by creating feature maps.

Pooling layers are used to reduce the dimensions of the feature maps. Thus, it reduces the number of parameters to learn and the amount of computation performed in the network.

Batch normalization layers allow every layer of the network to do learning more independently. It is used to normalize the output of the previous layers. Also, it can be used as regularization to avoid over-fitting the model. In practical coding, we add Batch Normalization after the activation function of the output layer or before the activation function of the input layer.

The Flatten layer converts the output of the final convolutional layer into a single one-dimensional vector.

A Dense layer is a layer that is deeply connected with its preceding layer, which means the neurons of the layer are connected to every neuron of its preceding layer.

Dropouts are a regularization technique that is used to prevent over-fitting in the model. Dropouts are added to randomly switch off some percentage of neurons of the network. Dropouts are usually advised not to use after the convolution layers; they are mainly used after the dense layers of the network. This is done to enhance the learning of the model.

The Figure 3.16 shows the Python implementation of the custom CNN.

```
model = Sequential()
model.add(Conv2D(16, (3, 3), activation='relu', padding="same", input_shape=(img_size,img_size,3)))
model.add(BatchNormalization())
model.add(Conv2D(16, (3, 3), padding="same", activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3), activation='relu', padding="same"))
model.add(BatchNormalization())
model.add(Conv2D(32, (3, 3), padding="same", activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu', padding="same" ))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3), padding="same", activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(64, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(4 , activation='softmax'))
```

Figure 3.16 Custom CNN Model Python Implementation

### 3.3.3.3.1.2 Densenet121

DenseNet (Dense Convolutional Network) (Huang *et al.*, 2018) is a convolutional neural network where each layer is connected to all other layers that are deeper in the network. This is done to enable maximum information flow between the layers of the network. To preserve the feed-forward nature, each layer obtains inputs from all the previous layers and passes on its feature maps to all the layers which will come after it. DenseNet consists of two critical blocks other than the basic convolutional and pooling layers: the Dense Blocks and the Transition layers. In DenseNet, the classifier uses features of all complexity levels. It tends to give more smooth decision boundaries. It also explains why DenseNet performs well when training data is insufficient.

The Densenet121 model loaded with the 'imagenet' weights and the final layer removed was used as a base model to build a Densenet121-based CNN model through transfer learning. All the layers in the base model were frozen from training and learning new weights. An Average Pooling layer was added on top of the base model, and finally, a Dense layer with four neurons and a Softmax activation function. The final model was referred to as the 'densenet\_model' and its Python implementation is given in Figure 3.17 below.

```
# Create the base pre-trained model
base_model = DenseNet121(include_top=False, weights = 'imagenet', input_shape=(320,320,3))

for layers in base_model.layers:
    layers.trainable=False

# Add a global spatial average pooling layer
x_pool = GlobalAveragePooling2D()(x)

# Create prediction layer for classification of our images
x = base_model.output
x = Flatten()(x)
prediction_layer = Dense(len(labels), activation='softmax')(x)
densenet_model = Model(inputs=base_model.input, outputs=prediction_layer)
```

Figure 3.17 densenet\_model Python Implementation

### 3.3.3.3.1.3 VGG16

VGG-16 (Simonyan and Zisserman, 2015) is a convolutional neural network that is 16 layers deep. In VGG16, there are thirteen convolutional layers, five Max Pooling layers, and three Dense layers, which sum up to 21 layers. However, it has only sixteen weight layers, that is learnable parameter layers.



The VGG16 model loaded with the ‘imagenet’ weights and the final layer removed was used as a base model to build a VGG16-based CNN model through transfer learning. All the layers in the base model were frozen from training and learning new weights. On top of the base model, the convolutional output was flattened and a dense layer with 128 neurons with an activation function ReLu was added. Finally, another dense layer with four output neurons and a softmax activation function was added to create the ‘vgg16\_final\_model’. The Python implementation is as shown in Figure 3.18.

```
from keras.applications.vgg16 import VGG16
vgg16_model = VGG16(pooling='avg', weights='imagenet', include_top=False, input_shape=(320,320,3))
for layers in vgg16_model.layers:
    layers.trainable=False
last_output = vgg16_model.layers[-1].output
vgg_x = Flatten()(last_output)
vgg_x = Dense(128, activation = 'relu')(vgg_x)
vgg_x = Dense(4, activation = 'softmax')(vgg_x)
vgg16_final_model = Model(vgg16_model.input, vgg_x)
```

Figure 3.18 vgg16\_final\_model Python Implementation

### 3.3.3.3.1.4 Resnet50

ResNet-50 (He *et al.*, 2015), or Residual Network, is a convolutional neural network that is 50 layers deep. ResNet first introduced the concept of skip connection, which is it adds the original input to the output of the convolution block.

The ResNet-50 model loaded with the ‘imagenet’ weights and the final layer removed was used as a base model to build a ResNet-50-based CNN model through transfer learning. All the layers in the base model were not frozen from training and learning new weights, as opposed to the other models. On top of the base model, a flattening layer of the convolutional output, a dense layer with 256 neurons and an activation function ReLu, and finally, another dense layer with four output neurons and a softmax activation function was added to create the ‘resnet50\_x\_final\_model’ as illustrated in Figure 3.19.

```
from keras.applications.resnet import ResNet50
ResNet50_model = ResNet50(weights='imagenet', include_top=False, input_shape=(320,320,3), classes=4)

for layers in ResNet50_model.layers:
    layers.trainable=True

opt = SGD(lr=0.01, momentum=0.7)
resnet50_x = Flatten()(ResNet50_model.output)
resnet50_x = Dense(256, activation='relu')(resnet50_x)
resnet50_x = Dense(4, activation='softmax')(resnet50_x)
resnet50_x_final_model = Model(inputs=ResNet50_model.input, outputs=resnet50_x)
```

Figure 3.19 resnet50\_x\_final\_model Python Implementation

### 3.3.3.3.1.5 InceptionV3

InceptionV3 (Szegedy *et al.*, 2015) is an image recognition model shown to attain greater than 78.1% accuracy on the ImageNet Dataset. It consists of many convolution and max pooling layers and, finally, a set of fully connected networks.

The InceptionV3 model loaded with the 'imagenet' weights and the final layer removed was used as a base model to build an InceptionV3-based CNN model through transfer learning. All the layers in the base model until the 250<sup>th</sup> layer were frozen from training and learning new weights. From the 250<sup>th</sup> layer, all the layers were not frozen from training. On top of the base model, a flattening layer of the convolutional output, a dense layer with 256 neurons and an activation function ReLu was added, followed by a dropout layer with a dropout rate of 50%. Finally, another dense layer with four output neurons and a softmax activation function was added to create the 'InceptionV3\_x\_final\_model.' The Python implementation is given in Figure 3.20.

```
from keras.applications.inception_v3 import InceptionV3
InceptionV3_model = InceptionV3(input_shape=(320,320,3),weights='imagenet', include_top=False)
for layer in InceptionV3_model.layers[:249]:
    layer.trainable = False
for layer in InceptionV3_model.layers[249:]:
    layer.trainable = True
InceptionV3_last_output = InceptionV3_model.output
InceptionV3_flattened_output = Flatten()(InceptionV3_last_output)

InceptionV3_x = Dense(256, activation='relu')(InceptionV3_flattened_output)
InceptionV3_x = Dropout(0.5)(InceptionV3_x)

InceptionV3_x = Dense(4, activation='softmax')(InceptionV3_x)

InceptionV3_x_final_model = Model(inputs=InceptionV3_model.input,outputs=InceptionV3_x)
```

Figure 3.20 InceptionV3\_x\_final\_model Python Implementation

### 3.3.3.3.1.6 CheXNet

CheXNet (Rajpurkar *et al.*, 2017) is a 121-layer convolutional neural network model proposed by some researchers at Stanford University to diagnose Pneumonia. The model is trained on ChestX-ray14 Dataset and diagnose all the 14 pathologies of the Dataset. It should be noted that CheXNet has been built on the DenseNet121 model.

The Densenet121 model loaded with no weights, and the final layer removed was used as a base model. A dense layer with 14 output neurons and with a sigmoid activation function was added on top of the base so that the model architecture is similar to the CheXNet architecture. The derived model was loaded

with the CheXNet trained weights (*CheXNet-Keras/weights.py at master · brucechou1983/CheXNet-Keras*, no date). All the layers in the CheXNet weights loaded model were frozen from training and learning new weights, and then the final two layers were removed. An Average Pooling layer was added on top of the fourth layer from the end, and finally, a Dense layer with four neurons and a Softmax activation function. The final model was referred to as the ‘chexnet\_model’ and its Python implementation is given below in Figure 3.21.

```
## Instantiate cheXnet model with pretrained weights. Pop last layers, add average pooling
from keras.models import Model
from tensorflow.keras.applications import densenet
chexnet_weights_path = "brucechou1983_CheXNet_Keras_0.3.0_weights.h5"
IMG_SIZE = 320

base = densenet.DenseNet121(weights=None,
                            include_top=False,
                            input_shape=(IMG_SIZE, IMG_SIZE, 3)
                            )
predictions = tf.keras.layers.Dense(14, activation='sigmoid', name='predictions')(base.output)
base = tf.keras.Model(inputs=base.input, outputs=predictions)
base.load_weights(chexnet_weights_path)
print("CheXNet loaded")
base.trainable=False # freeze most layers
base.training=False

base.layers.pop()
base.layers.pop()

new_model = GlobalAveragePooling2D()(base.layers[-4].output)
new_model = Dense(4, activation='softmax')(new_model)
chexnet_model = keras.Model(base.input, new_model)
```

Figure 3.21 chexnet\_model Python Implementation

### 3.3.3.3.2 Class Imbalance Treatment

It can be considered ideal for training a model on an evenly balanced dataset so that the positive and negative training cases would contribute equally to the loss. Generally, the categorical cross-entropy loss function is used as the loss function in multi-class classification problems to measure how well a model fits the data.

However, if the categorical cross-entropy loss function is used with a highly unbalanced dataset, the algorithm will be incentivized to prioritize the majority class since it contributes more to the loss.

A weighted cross-entropy loss function was defined, which is the categorical cross-entropy loss function weighted by class, increasing or decreasing the relative penalty of a probabilistic false negative for an individual class. The Python implementation of the weighted cross-entropy loss function is shown in Figure 3.22.

```

def get_weighted_loss(pos_weights, neg_weights, epsilon=1e-7):
    """
    Return weighted loss function given negative weights and positive weights.

    Args:
        pos_weights (np.array): array of positive weights for each class, size (num_classes)
        neg_weights (np.array): array of negative weights for each class, size (num_classes)

    Returns:
        weighted_loss (function): weighted loss function
    """
    def weighted_loss(y_true, y_pred):
        """
        Return weighted loss value.

        Args:
            y_true (Tensor): Tensor of true labels, size is (num_examples, num_classes)
            y_pred (Tensor): Tensor of predicted labels, size is (num_examples, num_classes)

        Returns:
            loss (Tensor): overall scalar loss summed across all classes
        """
        # initialize loss to zero
        loss = 0.0

        for i in range(len(pos_weights)):
            # for each class, add average weighted loss for that class
            loss_pos = -1 * K.mean(pos_weights[i] * y_true[:, i] * K.log(y_pred[:, i] + epsilon))
            loss_neg = -1 * K.mean(neg_weights[i] * (1 - y_true[:, i]) * K.log(1 - y_pred[:, i] + epsilon))
            loss += loss_pos + loss_neg

        return loss

    return weighted_loss

```

Figure 3.22 Weighted Cross-Entropy Loss Function

### 3.3.3.3 Using different Optimizers

Optimizers are algorithms or methods used to change the attributes/ parameters of a convolutional neural network so that the loss computed by the loss function is minimized. Their parameters include weights of neurons in different layers, bias values and learning rates. The following two optimizers were tested with the six convolutional neural network architectures stated in this thesis.

#### 3.3.3.3.1 Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent or SGD (Amari, 1993) is an optimization algorithm used to find the model parameters that correspond to the best fit between predicted and actual outputs. Compared to Gradient Descent, Stochastic Gradient Descent is much faster and more suitable for large-scale datasets. It is called ‘stochastic’ since the gradient it's not computed for the entire Dataset but only for one random point on each iteration. Stochastic gradient descent maintains a single learning rate for all weight updates, and the learning rate does not change during training.

#### 3.3.3.3.2 Adam Optimizer

Adam (Kingma and Ba, 2017) optimization algorithm combines the advantages of two other extensions of stochastic gradient descent; Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp). Adam changes the learning rate during training.

### 3.3.3.3.4 Using Data Augmentation on the Training Set

Data augmentation is a popular strategy for artificially increasing the amount of data needed to train robust ML/Deep Learning models. It increases the number of examples in the training set while also introducing more variety in what the model sees and learns from. Chapter 3.3.3.2 states in detail how data augmentation was applied to the training set using Keras Image Data Generators.

### 3.3.3.3.5 Application of different Pre-processing Techniques on the Data

Chapter 3.2 explains the different pre-processing techniques used in detail. In a nutshell, these include Monochrome conversion, Image Re-Sizing, CLAHE Conversion, Image Standardization and using different colour maps when converting to PNG format. Multiple colour maps applied to a Chest X-Ray is depicted in Figure 3.23.



Figure 3.23 Saving PNG with different Color Maps

A combination of the above-mentioned pre-processing techniques was applied before training multiple models before concluding with the final set of pre-processing steps in the Data Pre-Processing Module, as explained in chapter 3.2.

## 3.3.4 Model Interpretation through Visualization

The objective of this project is to build a multi-class classification model using chest radiographs, which predicts the probabilities for each of the four classes, as described in Chapter 1.3. It is vital to add an interpretability component to the final output, highlighting why a particular classification or decision has been made. This further enhances the trustworthiness of the decision support system.

One of the most common approaches aimed at increasing the interpretability of models for computer vision tasks is to use Class Activation Maps (CAM). Class activation maps are helpful for understanding where the model is "looking" when classifying an image. Gradient-weighted Class Activation Mapping (Grad-CAM) (Selvaraju *et al.*, 2020) is a type of CAM that uses the gradients of any class variable flowing into the final convolutional layer to produce a coarse localization map highlighting the critical regions in the image for predicting the class variable. GRADCAM applied to a Chest X-Ray image is depicted in Figure 3.24. This is a useful debugging tool for medical experts to validate that the model is indeed using the correct areas of the image to predict the class.



Figure 3.24 Image Interpretation with GRADCAM

The Python implementation of GRADCAM Heatmap is given in Figure 3.25 below.

```
def make_gradcam_heatmap(img_array, model, last_conv_layer_name, pred_index=None):
    import numpy as np
    # First, we create a model that maps the input image to the activations
    # of the last conv layer as well as the output predictions
    grad_model = tf.keras.models.Model(
        [model.inputs], [model.get_layer(last_conv_layer_name).output, model.output]
    )

    # Then, we compute the gradient of the top predicted class for our input image
    # with respect to the activations of the last conv layer
    with tf.GradientTape() as tape:
        last_conv_layer_output, preds = grad_model(img_array)
        if pred_index is None:
            pred_index = tf.argmax(preds[0])
        class_channel = preds[:, pred_index]

    # This is the gradient of the output neuron (top predicted or chosen)
    # with regard to the output feature map of the last conv layer
    grads = tape.gradient(class_channel, last_conv_layer_output)

    # This is a vector where each entry is the mean intensity of the gradient
    # over a specific feature map channel
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

    # We multiply each channel in the feature map array
    # by "how important this channel is" with regard to the top predicted class
    # then sum all the channels to obtain the heatmap class activation
    last_conv_layer_output = last_conv_layer_output[0]
    heatmap = last_conv_layer_output @ pooled_grads[..., tf.newaxis]
    heatmap = tf.squeeze(heatmap)

    # For visualization purpose, we will also normalize the heatmap between 0 & 1
    heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)
    return preds, heatmap.numpy()
```

Figure 3.25 GRADCAM Heatmap Python Implementation

### 3.3.5 Perform Error Analysis

#### 3.3.5.1 Establishing a Baseline

Since the pandemic began in early 2020, researchers have developed numerous machine-learning models for diagnosing COVID-19 using Chest X-Rays (CXR). Most experiments have used public datasets in which a responsible governing body has not verified the chest X-rays. Furthermore, critical performance metrics such as PPV and NPV have not been calculated in any of the studies analyzed so far. Moreover, the target classes used in almost all the studies analyzed do not follow the standard COVID-19 pneumonia classification. Hence, it is not convenient to set a baseline target using literature. It is also important to note that no Human-Level Performance (HLP) figure is available for this task to set up as a baseline. Therefore, setting up an initial baseline for this use case is not convenient.

#### 3.3.5.2 Evaluation Approach

The evaluation approach is experiment-based. A credible publicly available dataset elaborated in chapter 1.7, has been used to train several deep learning CNNs (Convolutional Neural Networks). The different deep learning architectures used in the model training process are explained in chapter 3.3.3.3.1.

The model performance evaluation is mainly driven by PPV (Positive Predicted Value) and NPV (Negative Predicted Value). The intuition behind choosing the metrics as PPV and NPV is that PPV indicates the probability that a patient is positive given that the model predicts positive. NPV indicates the probability that a patient is negative given that the model predicts negative. Hence, optimizing these two metrics leads to a trustworthy model. In addition to these two metrics, the confusion matrix and the ROC curves are evaluated to make an informed decision on model usability and applicability.

To evaluate the models, three user-defined functions were developed in Python to evaluate all the critical metrics, including PPV and NPV, to visualize the confusion matrix and the ROC curves. The three functions are namely: `get_performance`, `plot_confusion_matrix` and `get_roc_curve`.

##### 3.3.5.2.1 `get_performance` Function

The '`get_performance`' function indicates all the critical metrics, including PPV and NPV. These critical metrics are True Positives (TP), True Negatives (TN), False Positives (FP), False Negatives (FN), Accuracy, Prevalence, Sensitivity, Specificity, PPV, NPV, Area Under the Curve (AUC) and the F1-Score. Separate user-defined functions were developed for the metrics mentioned above and collated in

the 'get\_performance' function. These metrics are generated for each of the four class variables. The output of the function 'get\_performance' is as depicted in Figure 3.26.

Class	TP	TN	FP	FN	Accuracy	Prevalence	Sensitivity	Specificity	PPV	NPV	AUC	F1
Negative for Pneumonia	471	1914	1055	745	0.569892	0.290562	0.387336	0.644662	0.308650	0.719819	0.515999	0.343545
Typical Appearance	1058	1051	1225	851	0.503943	0.456153	0.554217	0.461775	0.463425	0.552576	0.507996	0.504771
Indeterminate Appearance	39	3232	228	686	0.781601	0.173238	0.053793	0.934104	0.146067	0.824911	0.493949	0.078629
Atypical Appearance	14	3755	95	321	0.900597	0.080048	0.041791	0.975325	0.128440	0.921246	0.508558	0.063063

Figure 3.26 'get\_performance' Function Output

The Python implementation of the 'get\_performance' function is provided in Figure 3.27 below.

```
def get_performance(predictions):
    class_var = []
    TP = []
    TN = []
    FP = []
    FN = []
    Accuracy = []
    Prevalence = []
    Sensitivity = []
    Specificity = []
    PPV = []
    NPV = []
    AUC = []
    F1 = []
    threshold = 0.5

    for var in ['Negative for Pneumonia', 'Typical Appearance', 'Indeterminate Appearance', 'Atypical Appearance']:
        y = predictions[var]
        pred = predictions[var + ' Pred']

        class_var.append(var)
        TP.append(true_positives(y, pred, th = threshold))
        TN.append(true_negatives(y, pred, th = threshold))
        FN.append(false_negatives(y, pred, th = threshold))
        FP.append(false_positives(y, pred, th = threshold))
        Accuracy.append(get_accuracy(y, pred, th = threshold))
        Prevalence.append(get_prevalence(y))
        Sensitivity.append(get_sensitivity(y, pred, th = threshold))
        Specificity.append(get_specificity(y, pred, th = threshold))
        try:
            PPV.append(get_ppv(y, pred, th = threshold))
        except:
            PPV.append(np.nan)

        try:
            NPV.append(get_npv(y, pred, th = threshold))
        except:
            NPV.append(np.nan)
        AUC.append(roc_auc_score(y, pred))
        F1.append(f1_score(y, pred))

    eval_matrix = pd.DataFrame({'Class':class_var, 'TP':TP, 'TN':TN, 'FP':FP, 'FN':FN, 'Accuracy':Accuracy
                               , 'Prevalence':Prevalence, 'Sensitivity':Sensitivity, 'Specificity':Specificity, 'PPV':PPV, 'NPV':NPV, 'AUC':AUC, 'F1':F1})

    return eval_matrix
```

Figure 3.27 'get\_performance' Python Implementation



### 3.3.5.2.2 plot\_confusion\_matrix Function

The 'plot\_confusion\_matrix' function was developed to visualize the confusion matrix. The function output is similar to Figure 3.28.

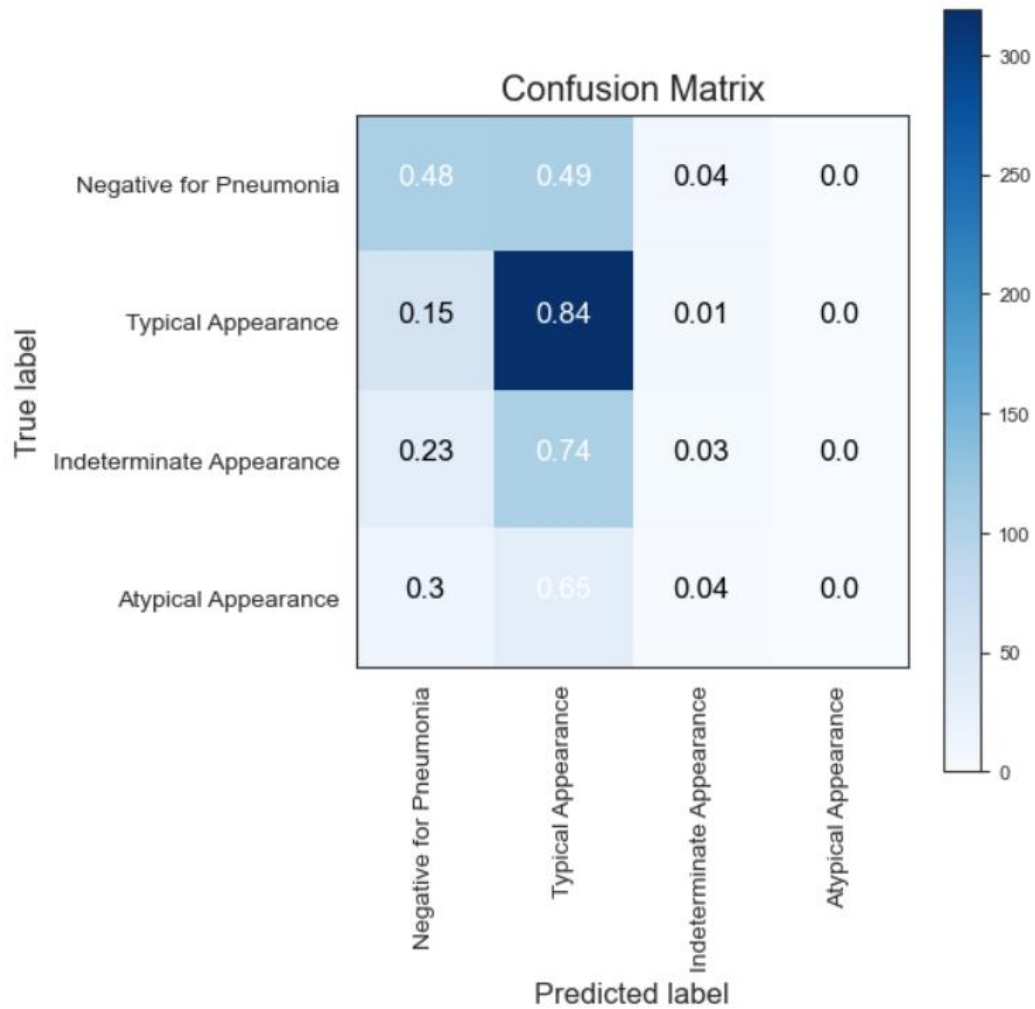


Figure 3.28 'plot\_confusion\_matrix' Function

The confusion matrix is critical to deciding the best-performing model, specifically by considering the False Positive and False Negative tradeoff. However, in medical applications, it is a common practice to optimize a model to reduce the number of False Negatives.

The following Figure 3.29 depicts the Python implementation of the 'plot\_confusion\_matrix' function.

```

def plot_confusion_matrix(cm, classes, normalize=True, title='Confusion matrix', cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.figure(figsize=(6,6))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90, fontsize = 10)
    plt.yticks(tick_marks, classes, fontsize = 10)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        cm = np.around(cm, decimals=2)
        cm[np.isnan(cm)] = 0.0
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')
    thresh = cm.max() / 2.

    for i in range (cm.shape[0]):
        for j in range (cm.shape[1]):
            plt.text(j, i, cm[i, j],
                    horizontalalignment="center",
                    color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

```

Figure 3.29 'plot\_confusion\_matrix' Python Implementation

### 3.3.5.2.3 get\_roc\_curve Function

The 'get\_roc\_curve' function plots the ROC curve for each of the four class variables. The ROC curve or the Receiver Operating Characteristic curve shows the tradeoff between Sensitivity and Specificity. This assists in visually seeing the goodness of the fitted models for each class variable. The output of the function 'get\_roc\_curve' is similar to Figure 3.30 and its Python implementation is provided in Figure 3.31 below.

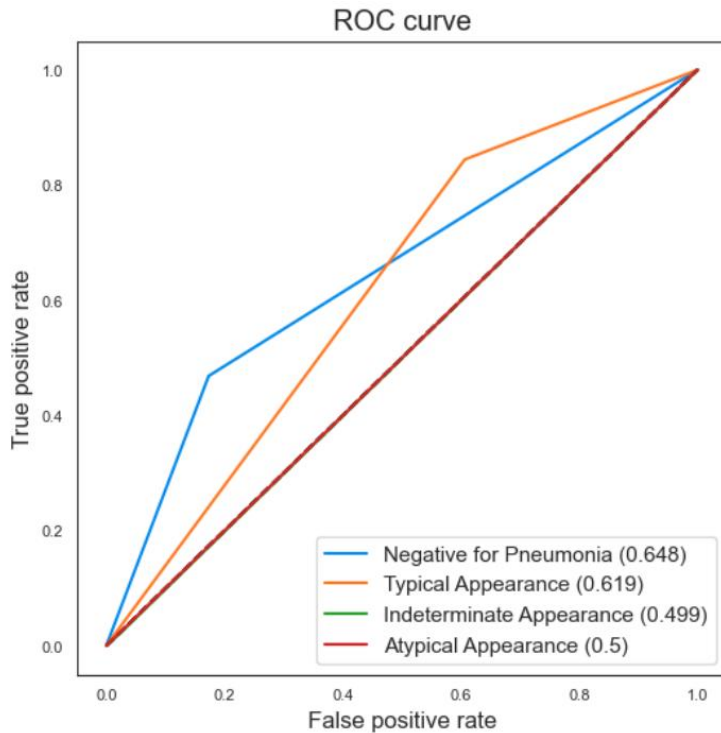


Figure 3.30 'get\_roc\_curve' Function

Classifiers that give curves closer to the top-left corner indicate better performance. As a baseline, a random classifier is expected to give points lying along the diagonal.

```
def get_roc_curve(labels, predicted_matrix):
    auc_roc_vals = []
    for i, var in enumerate(labels):
        try:
            gt = predicted_matrix[var]
            pred = predicted_matrix[var + ' Pred']
            auc_roc = roc_auc_score(gt, pred)
            auc_roc_vals.append(auc_roc)
            fpr_rf, tpr_rf, _ = roc_curve(gt, pred)
            plt.figure(1, figsize=(6, 6))
            plt.plot([0, 1], [0, 1], 'k--')
            plt.plot(fpr_rf, tpr_rf,
                    label=labels[i] + " (" + str(round(auc_roc, 3)) + ")")
            plt.xlabel('False positive rate')
            plt.ylabel('True positive rate')
            plt.title('ROC curve')
            plt.legend(loc='best')
        except:
            print(
                f"Error in generating ROC curve for {labels[i]}. "
                f"Dataset lacks enough examples."
            )
    plt.show()
    return auc_roc_vals
```

Figure 3.31 'get\_roc\_curve' Python Implementation

### 3.4 Model Inference Module

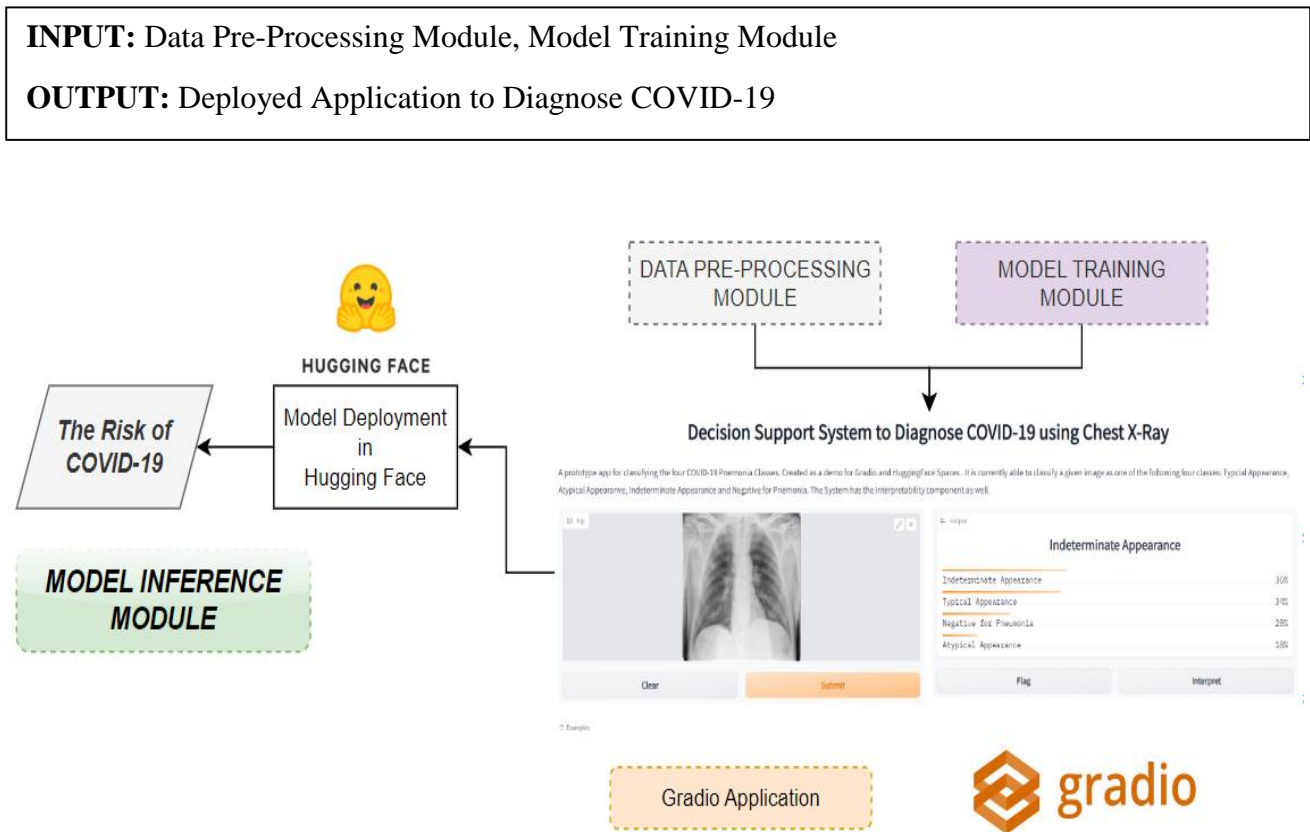


Figure 3.32 Model Inference Module

The Figure 3.32 above demonstrates the Model Inference Module. It utilizes the Data Pre-Processing Module and the Model Training Modules as input to provide inference for a Chest X-Ray through a deployed application. The application not only can classify into a COVID-19 Pneumonia class but can also highlight using a heat map to point out why the particular COVID-19 Pneumonia class was predicted.

A DICOM image to be diagnosed is processed using the Data Pre-processing module and is staged in a staging area as a PNG human-identifiable image. The PNG image is then provided as input to the deployed application. The application was developed using Python, explicitly using the 'gradio' library as shown in Figure 3.33.

```

### Application Code

import gradio as gr

title = "Decision Support System to Diagnose COVID-19 using Chest X-Ray"
description = """
A prototype app for classifying the four COVID-19 Pneumonia Classes, created as a Demo using Gradio and HuggingFace Spaces
. It is currently able to classify a given image as one of the following four classes: Typical Appearance, Atypical Appearance,
Indeterminate Appearance and Negative for Pneumonia. The System has an additional interpretability component to assist the user why a particular decision has been made
"""

examples = ['0a3afeef9c01.png', '65761e66de9f.png', '0bf205469ffa.png', '0c06f6f96a5a.png', '0c1ba97ad7c8.png', '0c2d323a04bf.png', '0c6b440ba98e.png', '0c7b15362352.png']
interpretation='default'
enable_queue=True

gr.Interface(fn=classify_image,inputs=gr.inputs.Image(shape=(320, 320)),outputs=gr.outputs.Label(num_top_classes=4),
title=title,description=description, examples=examples,
interpretation=interpretation,enable_queue=enable_queue).launch()

```

Figure 3.33 Gradio Application Development

The Gradio application developed was deployed in Huggingface with a public URL. The deployed Huggingface application can be accessed via the link below. [https://huggingface.co/spaces/shehan16/decision\\_support\\_system\\_covid](https://huggingface.co/spaces/shehan16/decision_support_system_covid) . A snapshot of the web application is given in Figure 3.34.

## Decision Support System to Diagnose COVID-19 using Chest X-Ray

A prototype app for classifying the four COVID-19 Pneumonia Classes, created as a Demo using Gradio and HuggingFace Spaces . It is currently able to classify a given image as one of the following four classes: Typical Appearance, Atypical Appearance, Indeterminate Appearance and Negative for Pneumonia. The System has an additional interpretability component to assist the user why a particular decision has been made

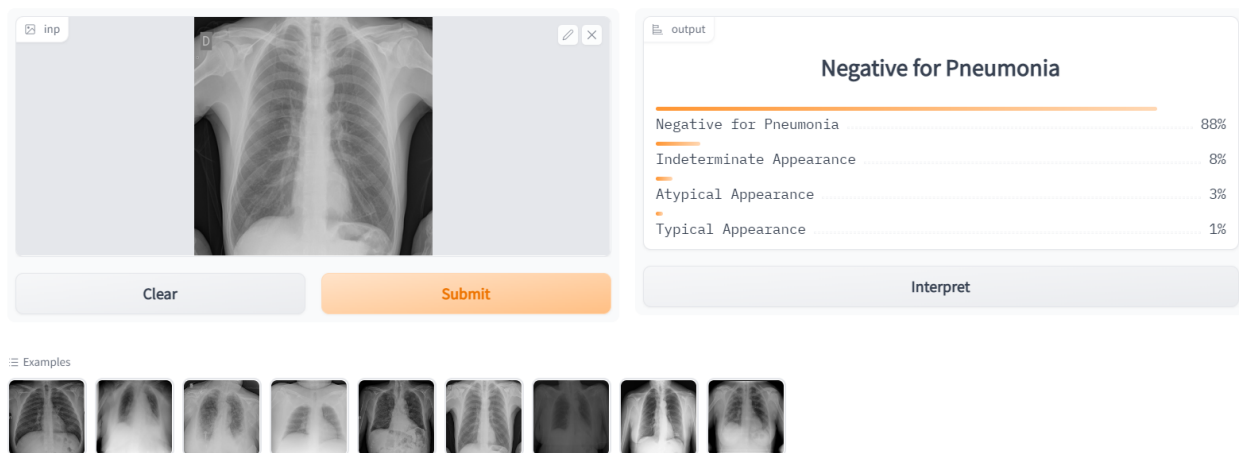


Figure 3.34 Huggingface Application

## 4. CHAPTER 4 DISCUSSION AND RESULTS

As stated in chapter 3.3.3.3, multiple models were trained with six CNN architectures, different optimizers, different pre-processing techniques, and different loss functions. The techniques and processes which yielded the best-performing model were used to develop the Data Pre-Processing Module (Chapter 3.2) and the Model Training Module (Chapter 3.3)

A detailed evaluation of model performances of the best model performance from each of the six CNN architectures is provided below.

### 4.1 Custom CNN Model Performance

The Custom CNN model was trained on the training image set without data augmentation over 32 epochs with a batch size of 32. The custom model was trained to optimize the validation set accuracy. During the training process, Adam was used as the optimization algorithm, while the Weighted cross-entropy loss was used as the loss function. However, the model could not even overfit the training set, which implied that the model was not learning. This is evident in the model performance shown in Figure 4.1.

Class	TP	TN	FP	FN	Accuracy	Prevalence	Sensitivity	Specificity	PPV	NPV	AUC	F1
Negative for Pneumonia	3	3058	4	1179	0.721254	0.278511	0.002538	0.998694	0.428571	0.721737	0.500616	0.005046
Typical Appearance	1950	7	2280	7	0.461122	0.461122	0.996423	0.003061	0.460993	0.500000	0.499742	0.630354
Indeterminate Appearance	0	3465	5	774	0.816447	0.182375	0.000000	0.998559	0.000000	0.817410	0.499280	0.000000
Atypical Appearance	1	3912	1	330	0.922008	0.077992	0.003021	0.999744	0.500000	0.922207	0.501383	0.006006

Figure 4.1 Custom Model Performance on the Training Set

### 4.2 ‘densenet\_model’ Performance

The ‘densenet\_model’ was trained on the training image set without data augmentation over eight epochs to optimise the validation set accuracy. The Adam optimization algorithm was used, and weighted cross-entropy loss was used as the loss function. The validation accuracy reached 0.54 as the training ended. The Loss and the Accuracy change of the Test and Validation sets over the training epochs is depicted in Figure 4.2 below.

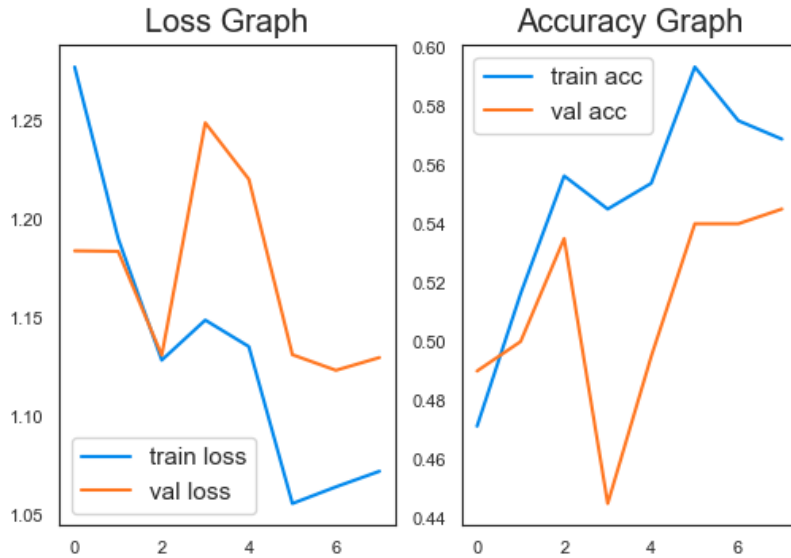


Figure 4.2 Training and Validation Set Accuracy - densenet\_model

However, the ‘densenet\_model’ was not successful as a suitable model for the classifier since it predicted either Positive only or Negative only for each of the four classes as shown in Figure 4.3.

Class	TP	TN	FP	FN	Accuracy	Prevalence	Sensitivity	Specificity	PPV	NPV	AUC	F1
Negative for Pneumonia	1184	0	3082	0	0.277543	0.277543	1.0	0.0	0.277543	NaN	0.5	0.434495
Typical Appearance	0	2300	0	1966	0.539147	0.460853	0.0	1.0	NaN	0.539147	0.5	0.000000
Indeterminate Appearance	0	3501	0	765	0.820675	0.179325	0.0	1.0	NaN	0.820675	0.5	0.000000
Atypical Appearance	0	3915	0	351	0.917722	0.082278	0.0	1.0	NaN	0.917722	0.5	0.000000

Figure 4.3 'get\_performance' report - densenet\_model on the Training Set

### 4.3 'vgg16\_final\_model' performance

The ‘vgg16\_final\_model’ was trained on the training image set without data augmentation over 12 epochs so that the validation set accuracy is optimized. The Adam optimization algorithm was used as the optimizer. The weighted cross-entropy loss and the categorical cross-entropy loss functions were used as the loss function to compare performance.

The model trained with the weighted cross entropy loss function yielded better results. The Figure 4.4 depicts the model performance when it was trained with the weighted cross entropy loss function.

Class	TP	TN	FP	FN	Accuracy	Prevalence	Sensitivity	Specificity	PPV	NPV	AUC	F1
Negative for Pneumonia	178	448	104	53	0.799489	0.295019	0.770563	0.811594	0.631206	0.894212	0.791078	0.693957
Typical Appearance	276	277	145	85	0.706258	0.461047	0.764543	0.656398	0.655582	0.765193	0.710471	0.705882
Indeterminate Appearance	2	641	15	125	0.821201	0.162197	0.015748	0.977134	0.117647	0.836815	0.496441	0.027778
Atypical Appearance	12	668	51	52	0.868455	0.081737	0.187500	0.929068	0.190476	0.927778	0.558284	0.188976

Figure 4.4 Test Set Performance of 'vgg16\_final\_model' trained with 'weighted\_crossentropy' loss function

The evaluation results show that the 'vgg16\_final\_model' had been able to learn from the data, compared to the 'densenet\_model'. Nevertheless, the PPV for 'Indeterminate Appearance' and 'Atypical Appearance' are 11% and 19%, respectively, which is not a well-desired outcome.

#### 4.4 'resnet50\_x\_final\_model' Performance

The 'resnet50\_x\_final\_model' was trained on the training image set without data augmentation over ten epochs so that the validation set accuracy is optimized. The Stochastic Gradient Descent algorithm was used as the optimizer with a learning rate of 0.01. However, like the 'densenet\_model', the 'resnet50\_x\_final\_model' was unsuccessful during the training process, with the model always predicting Negative for the 'Negative for Pneumonia' class and always predicts Positive for the other three classes. The Figure 4.5 is clear evidence for the above statement.

Class	TP	TN	FP	FN	Accuracy	Prevalence	Sensitivity	Specificity	PPV	NPV	AUC	F1
Negative for Pneumonia	0	552	0	231	0.704981	0.295019	0.0	1.0	NaN	0.704981	0.5	0.000000
Typical Appearance	0	422	0	361	0.538953	0.461047	0.0	1.0	NaN	0.538953	0.5	0.000000
Indeterminate Appearance	0	656	0	127	0.837803	0.162197	0.0	1.0	NaN	0.837803	0.5	0.000000
Atypical Appearance	64	0	719	0	0.081737	0.081737	1.0	0.0	0.081737	NaN	0.5	0.151122

Figure 4.5 Test Set Performance of 'resnet50\_x\_final\_model'

#### 4.5 'InceptionV3\_x\_final\_model' Performance

The 'InceptionV3\_x\_final\_model' was trained on the training image set without data augmentation over 15 epochs so that the validation set accuracy is optimized. However, the validation accuracy worsened as the number of epochs increased, as shown in Figure 4.6.



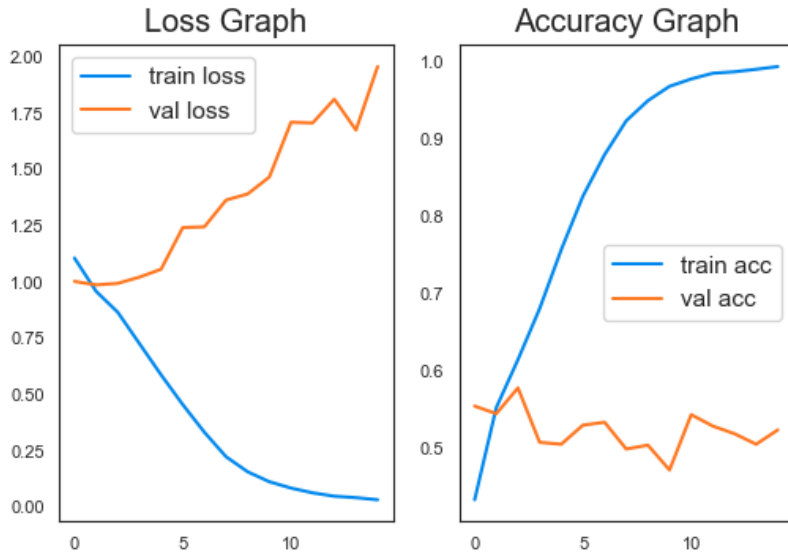


Figure 4.6 Validation Accuracy Reduction with Time – Inception\_x\_final\_model

The model was trained with a Stochastic Gradient Descent algorithm as the optimizer, with a learning rate of 0.001. The loss function used was the weighted cross-entropy function. The following figure depicts the Test Set Performance of the ‘InceptionV3\_x\_final\_model’. The ‘get\_performance’ output for the ‘InceptionV3\_x\_final\_model’ is shown below in Figure 4.7.

Class	TP	TN	FP	FN	Accuracy	Prevalence	Sensitivity	Specificity	PPV	NPV	AUC	F1
Negative for Pneumonia	78	512	82	169	0.701546	0.293698	0.315789	0.861953	0.487500	0.751836	0.588871	0.383292
Typical Appearance	288	186	267	100	0.563615	0.461356	0.742268	0.410596	0.518919	0.650350	0.576432	0.610817
Indeterminate Appearance	19	637	68	117	0.780024	0.161712	0.139706	0.903546	0.218391	0.844828	0.521626	0.170404
Atypical Appearance	5	737	34	65	0.882283	0.083234	0.071429	0.955901	0.128205	0.918953	0.513665	0.091743

Figure 4.7 Test Set Performance of 'InceptionV3\_x\_final\_model'

This model performed worse than the 'vgg16\_final\_model', with AUC scores for all four classes falling below 0.5.

#### 4.6 ‘chexnet\_model’ Performance

A more significant number of experiments were performed to find the best performing ‘chexnet\_model’ since it was pretty evident for the 'chexnet\_model' to perform better on the Chest X-Ray dataset. The main reason for this assumption was that the original CheXNet model had already been trained on Chest X-Rays, and it was ideal for transfer learning.

The following experiments were conducted in search of the best-performing 'chexnet\_model'.

## 1. Adam Optimizer Vs Stochastic Gradient Descent Optimizer (SGD)

Both the chexnet models were trained with the weighted cross entropy loss function. The classifier trained with the adam optimizer returned a higher validation accuracy of 0.6091 compared to 0.6049 with the SGD optimizer. Both the classifiers were trained over eight epochs.

## 2. Testing with and without a flattened final layer before the dense layers

The classifier with the final convolutional layer flattened trained with an adam optimizer returned a validation accuracy of 0.6166 as shown in Figure 4.8, which is slightly better than 0.6091 of the non-flattened architecture trained with the adam optimizer. The weighted cross entropy loss function was used in both cases.

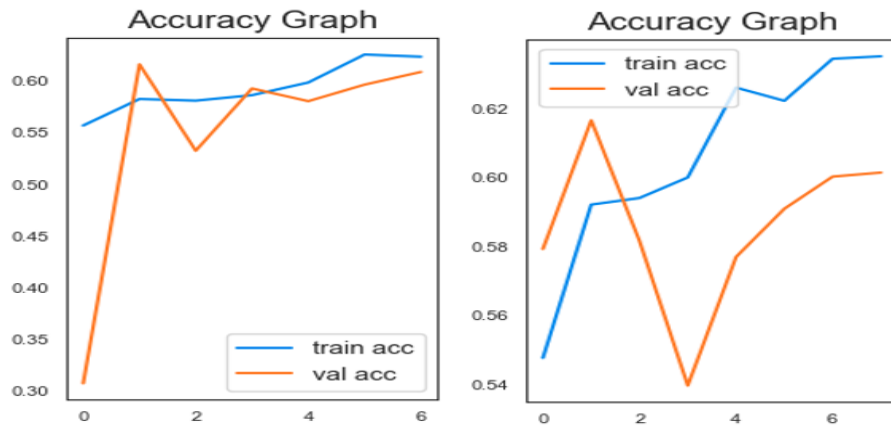


Figure 4.8 Validation Accuracies of Non-Flattened Vs Flattened Architectures

## 3. Re-scaling images Vs Standardizing images before model training

Separate chexnet models were trained with Image Re-Scaling and with Image standardization. The Figure 4.9 illustrates the Python implementation for rescaling.

```
datagen = ImageDataGenerator(  
    rescale=1./255)
```

Figure 4.9 Image Re-scaling in Image Generator

The model trained with standardized images obtained better performance in terms of PPV values. This is why Image standardization was added as a component of the Image Pre-Processing Module.

The following performance report in Figure 4.10 summarizes the lack of performance in the model trained with re-scaled data.

Class	TP	TN	FP	FN	Accuracy	Prevalence	Sensitivity	Specificity	PPV	NPV	AUC	F1
Negative for Pneumonia	217	66	486	14	0.361430	0.295019	0.939394	0.119565	0.308677	0.825000	0.529480	0.464668
Typical Appearance	9	420	2	352	0.547893	0.461047	0.024931	0.995261	0.818182	0.544041	0.510096	0.048387
Indeterminate Appearance	13	600	56	114	0.782886	0.162197	0.102362	0.914634	0.188406	0.840336	0.508498	0.132653
Atypical Appearance	0	719	0	64	0.918263	0.081737	0.000000	1.000000	NaN	0.918263	0.500000	0.000000

Figure 4.10 Test Set Performance of 'chexnet\_model' trained on re-scaled data

It is to be noted that the best-performing model was trained on standardized images, and this is discussed in detail in Chapter 4.8.

#### 4. Training models on PNGs saved in different Color Maps

The Data Pre-Processing Module takes a DICOM object and converts it to a PNG Image with color map 'grey'. This was decided after the color map 'grey' images contributed to the highest performances in the trained models. The other color maps tested were 'bone' and the default color map. The different color maps used are depicted in Figure 3.23 in Chapter 3.3.3.3.5.

#### 5. Testing with data augmentation on the training set

The model was trained for 15 epochs, and the same image would be presented differently in each epoch with width, height shifts, brightness and zoom level adjustments. As expected, the test set performance was better with data augmentation compared to the other model performances. The best-performing model was obtained with data augmentation, which is discussed in Chapter 4.8.

#### 4.7 False Positive Vs False Negative Tradeoff in Medicine

After multiple experiments and analysis, it was evident that the most suitable model must strike a balance between False Positives and False Negatives.

The following figure depicts the confusion matrix output of a 'chexnet\_model' that was trained on color map 'grey' images with no CLAHE conversion. An example of a confusion matrix having high False Positives is depicted in Figure 4.11.

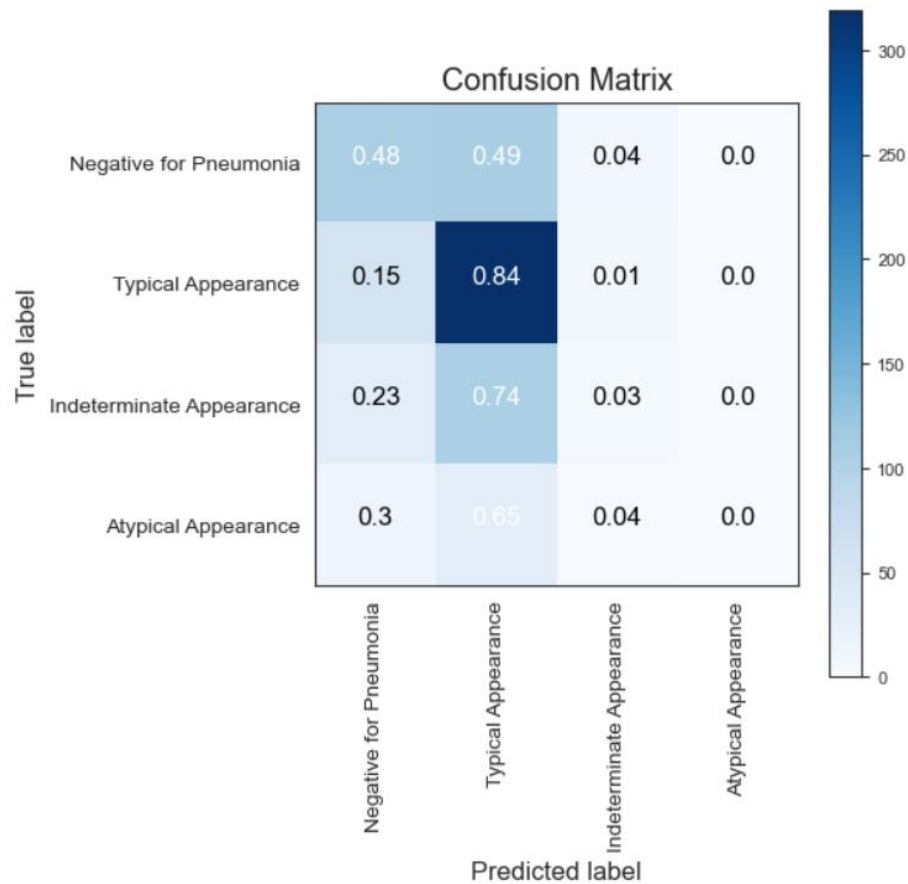


Figure 4.11 Confusion Matrix with more False Positives

It should be noted that ‘Typical Appearance’, ‘Atypical Appearance’ and ‘Indeterminate Appearance’ are all classes of COVID-19 Pneumonia, whereas ‘Negative for Pneumonia’ is the equivalent of not having COVID-19 Pneumonia. Having said the above statement, 84% of ‘Typical Appearance’ have been correctly classified, and only 15% of the cases have been misclassified as ‘Negative for Pneumonia’. Similarly, a more significant majority of ‘Indeterminate Appearance’ and ‘Atypical Appearance’ have also been classified as COVID-19 Pneumonia, although they have been tagged under a different COVID-19 Pneumonia class. This proves that this model detects COVID-19 well. However, ~52% of ‘Negative for Pneumonia’ X-Rays have been misclassified as COVID-19 Pneumonia, which is a significant False Positive rate.

On the other hand, the following figure depicts the confusion matrix output of a 'chexnet\_model' that was trained on color map ‘bone’ images with CLAHE conversion. An example of a confusion matrix having high False Negatives is depicted in Figure 4.12.

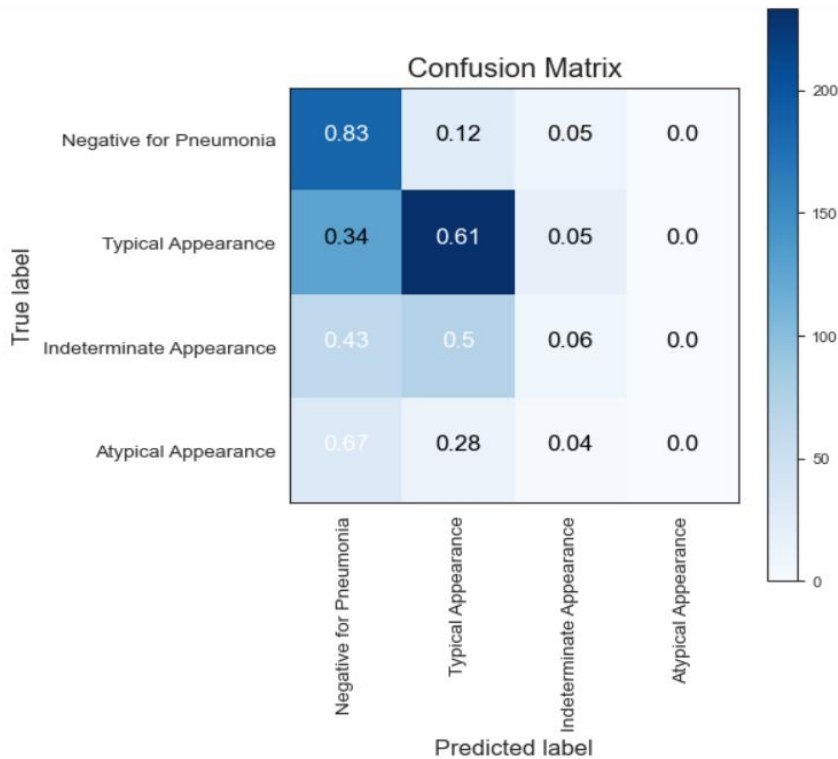


Figure 4.12 Confusion Matrix with more False Negatives

It can be clearly seen that more than 83% of the cases have been predicted correctly, which proves the model has a low False Positive rate of 17% compared to 52% in the previous model. However, this model has an increased False Negative rate with significant portions from the three COVID-19 Pneumonia classes that have been predicted as ‘Negative for Pneumonia’.

Generally, in medical applications, there is more harm in not identifying a positive case rather than falsely identifying a negative case as positive. Hence, the final model that has been selected has more tendency to have fewer False Negatives. However, the number of False Positives should also be not too large to avoid unnecessary complications.

## 4.8 Best Model Selection

After a series of experiments, the best model with the highest performance has the following properties. It possesses a CheXNet-based architecture, as explained in detail in chapter 3.3.3.3.1.6. The model follows the same pre-processing steps as stated clearly in the Data Pre-Processing Module in chapter 3.2. This model used the weighted cross entropy loss function, Adam optimizer as the optimization algorithm and Data Augmentation during the model training phase. The model pre-processing steps and the model training steps that were involved in the model building were used as the foundation for

developing the Data Pre-Processing Module (Chapter 3.2) and the Model Training Module (Chapter 3.3).

The Evaluation Report of the best model is explained below. A detailed explanation of the Evaluation Report is provided in chapter 3.3.5.2.

### 1. ‘get\_performance’ function

The Figure 4.13 below summarizes the performance metrics relevant to the best model.

Class	TP	TN	FP	FN	Accuracy	Prevalence	Sensitivity	Specificity	PPV	NPV	AUC	F1
Negative for Pneumonia	192	472	99	48	0.818742	0.295931	0.800000	0.826620	0.659794	0.907692	0.813310	0.723164
Typical Appearance	308	307	116	80	0.758323	0.478422	0.793814	0.725768	0.726415	0.793282	0.759791	0.758621
Indeterminate Appearance	13	627	67	104	0.789149	0.144266	0.111111	0.903458	0.162500	0.857729	0.507285	0.131980
Atypical Appearance	5	734	11	61	0.911221	0.081381	0.075758	0.985235	0.312500	0.923270	0.530496	0.121951

Figure 4.13 Performance metrics of the best model

The final model can identify ‘Negative for Pneumonia’ and ‘Typical Appearance’ better than the other two classes because of the class imbalance in the dataset. For a chest X-Ray, if the final model predicts the class ‘Negative for Pneumonia’, there is a 66% chance that the X-Ray is actually ‘Negative for Pneumonia’. The same is true for ‘Typical Appearance’ with a chance of 73%. However, this model needs to better identify the two classes ‘Indeterminate Appearance’ and ‘Atypical Appearance’. The PPV scores are 16% and 31% respectively, indicating a lesser chance of believing the model when it predicts one of the two underperforming classes.

### 2. get\_roc\_curve Function

The ROC curves also support the same ideology that the final model does a satisfactory job in identifying the two classes, ‘Negative for Pneumonia’ and ‘Typical Appearance’, but fails to do well with the other two classes. This is a graphical representation of the AUC scores mentioned in the ‘get\_performance’ report. The ROC curves indicate the model does the best in identifying ‘Negative for Pneumonia’ with an AUC of 0.81 and does equally well in identifying ‘Typical Appearance’ with an AUC of 0.76. However, the ROC curves for the other two classes almost fall upon the diagonal line indicating that the classifier is similar to a random classifier when predicting the said classes. The ROC curve of the best model is given in Figure 4.14.

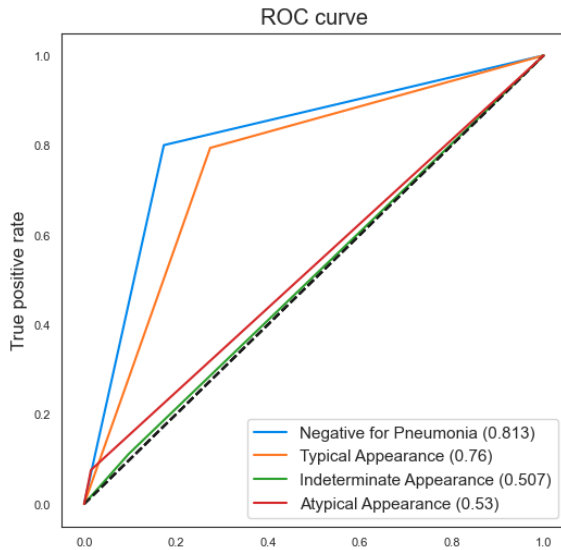


Figure 4.14 ROC Curves of the Best Model

### 3. 'plot\_confusion\_matrix' function

It should be noted that the objective of this project is not to diagnose COVID-19 Pneumonia directly but to act as a decision support system to guide patients and medical staff. Hence, an indication of whether the subject is suffering from COVID-19 or not is sufficient to decide on getting further medical advice and checkups. This is a vital idea since all three classes, 'Typical Appearance', 'Indeterminate Appearance' and the 'Atypical Appearance' lead to COVID-19 Pneumonia. The validity of the final model from this aspect is addressed from the 'plot\_confusion\_matrix' function given in Figure 4.15.

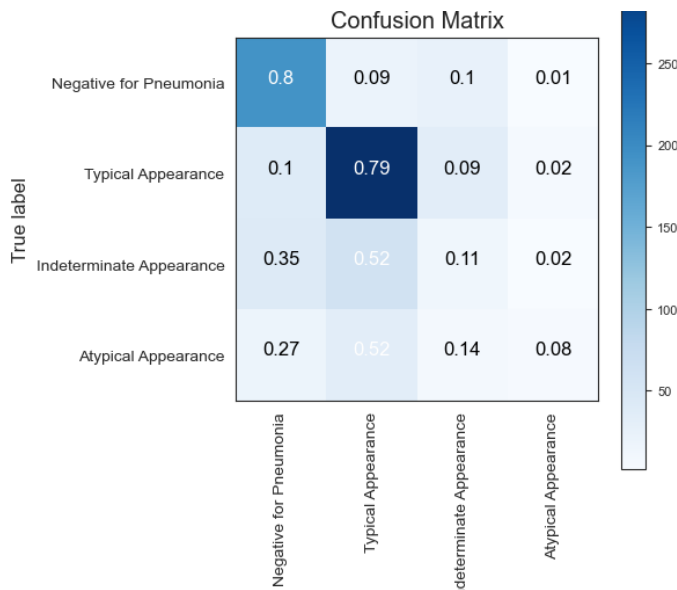


Figure 4.15 Confusion Matrix of the Final Model

The final model has a low false negative rate while still maintaining a low false positive rate. Approximately, ~80% of the ‘Negative for Pneumonia’ cases have been classified correctly, whereas around 20% of the cases have been classified into one of the three COVID-10 Pneumonia classes. This indicates that the False Positive Rate of the model for the class ‘Negative for Pneumonia’ is at 20%.

On the other hand, 79% of the ‘Typical Appearance’ cases have been correctly classified, and 11% of the ‘Typical Appearance’ cases have been incorrectly classified into the ‘Indeterminate Appearance’ and ‘Atypical Appearance’ classes. However, it should be noted these all these three classes are COVID-19 Pneumonia classes. Hence, the purpose of this decision support system is served. However, around 10% of the ‘Typical Appearance’ cases have been misclassified as ‘Negative for Pneumonia’. This is considered a False Negative and is the worst-case scenario. The False Negative Rate for the class ‘Typical Appearance’ is approximately 10%.

Furthermore, the cases with the class labels ‘Atypical Appearance’ and ‘Indeterminate Appearance’ are subclasses of COVID-19 Pneumonia with a low Prevalence, contributing only up to 22% of the cases in the data set. The best model itself is performing poorly in the two classes, ‘Atypical Appearance’ and ‘Indeterminate Appearance’, with 8% and 11% correct classifications, respectively. It should be noted that the predicted class label for most misclassifications is ‘Typical Appearance’, which itself is a COVID-19 Pneumonia class. Having said that, the effective False Positive Rate is comparatively lower at 27% and 35%, respectively, where the model predicted both these classes as ‘Negative for Pneumonia’.

#### **4.9 Summary of Model Performances**

A comprehensive summary of model performances on the Test Set is provided in Table 4.1 below. Four critical metrics, namely Sensitivity, Specificity, PPV and NPV, which are critical in medical applications, have been considered for the comparison. The chexnet\_model was selected as the best-performing model, and this is clearly seen when comparing the said critical metrics calculated separately for all four classes.



Table 4.1 Model Performance Comparison on Test Set

Metric	Model	Class			
		‘Negative for Pneumonia’	‘Typical Appearance’	‘Indeterminate Appearance’	‘Atypical Appearance’
Sensitivity	Custom CNN Model	-	1.00	-	-
	densenet_model	1.00	-	-	-
	vgg16_final_model	0.77	0.76	0.02	0.19
	resnet50_x_final_model	-	-	-	1.00
	InceptionV3_x_final_model	0.32	0.74	0.14	0.07
	<b>chexnet_model</b>	0.80	0.79	0.11	0.08
Specificity	Custom CNN Model	1.00	-	1.00	1.00
	densenet_model	1.00	1.00	1.00	-
	vgg16_final_model	0.81	0.66	0.98	0.93
	resnet50_x_final_model	1.00	1.00	1.00	-
	InceptionV3_x_final_model	0.86	0.41	0.90	0.96
	<b>chexnet_model</b>	0.83	0.73	0.90	0.99
PPV	Custom CNN Model	-	0.48	-	-
	densenet_model	0.30	-	-	-
	vgg16_final_model	0.63	0.66	0.12	0.19
	resnet50_x_final_model	-	-	-	0.08
	InceptionV3_x_final_model	0.49	0.52	0.22	0.13
	<b>chexnet_model</b>	0.66	0.73	0.16	0.31
NPV	Custom CNN Model	0.70	-	0.86	0.92
	densenet_model	-	0.54	0.84	0.92
	vgg16_final_model	0.89	0.77	0.84	0.93
	resnet50_x_final_model	0.70	0.54	0.84	-
	InceptionV3_x_final_model	0.75	0.65	0.84	0.92
	<b>chexnet_model</b>	0.91	0.79	0.86	0.92

## 5. CHAPTER 5 CONCLUSION AND FUTURE WORK

### 5.1 Conclusion

As stated in Chapter 1.3, the objective of this project is to develop a decision support system to diagnose COVID-19, which is a multi-class classification system that can classify an X-Ray DICOM Object to one of the four COVID-19 Pneumonia classes: 'Negative for Pneumonia', 'Typical Appearance', 'Indeterminate Appearance' and 'Atypical Appearance'. The classes 'Typical Appearance', 'Indeterminate Appearance' and 'Atypical Appearance' indicate the presence of COVID-19, while the class 'Negative for Pneumonia' indicates the absence of COVID-19. The decision support system is also incorporated with the capability to highlight using a heat map why a specific classification or a decision has been made.

Successful implementation of this decision support system will be advantageous to people who are not privileged enough to find COVID-19 test kits at their convenience. Furthermore, many people obtain Chest X-Rays for different requirements, and this decision support system can be used to get an insight into COVID-19 Pneumonia upon their consent. This provides an immense contribution to society, on the one hand, to the people getting a cost-free diagnosis. This will aid radiologists quickly and confidently diagnosing thousands of COVID-19 patients in Sri Lanka and, if required, channel them for further testing through accepted means like PCR and advise on further medical assistance. On the other hand, this is beneficial for society as a whole to identify the level of spread of COVID-19 in society.

The development of this decision support system is based on three main modules; Data Pre-Processing, Model Training and Model Inference Modules. The Data Pre-Processing Module describes the pre-processing steps that need to be applied to a DICOM object after metadata analysis. The Model Training Module focuses on developing the best-performing model. It should be noted that even the best model chosen performs poorly on two of the COVID-19 Pneumonia classes, 'Atypical Appearance' and 'Indeterminate Appearance' classes. Nevertheless, the best model is still usable since a significant percentage of misclassifications of the above-mentioned classes are misclassified as 'Typical Appearance', which is again an indication of COVID-19 Pneumonia. Also, the occurrences of the minority classes are low due to low prevalence.

This chosen best model, or the 'chexnet\_model', is incorporated inside the decision support system to perform the classifications. The most feasible solution to deploy this decision support system technically

is to deploy it on a web server. The prototype decision support system that was developed as a part of this project can be accessed using the link below.

[https://huggingface.co/spaces/shehan16/decision\\_support\\_system\\_covid](https://huggingface.co/spaces/shehan16/decision_support_system_covid)

A DICOM image to be diagnosed is processed using the Data Pre-processing module and is staged in a staging area as a PNG human-identifiable image. The PNG image is then provided as input to the deployed application to get a classification result which can then be used as a support mechanism to diagnose COVID-19.

## **5.2 Future Work**

The primary concern of the deployed decision support system is that the underlying multi-class classification model is performing poorly on the ‘Atypical Appearance’ and ‘Indeterminate Appearance’ classes. Hence, the model needs to be re-trained by acquiring more DICOM Images for the said two classes. Based on initial discussions with the Kalubowila hospital, DICOM Objects of X-Rays can be obtained from their database after a proposal of work is presented to the hospital as well as to the Ethical governing body. Support from radiologists will also be required to label the DICOM Objects. Acquiring Sri-Lankan X-Rays will help to validate the model in a local context.

Furthermore, the current application is deployed in a Huggingface space which is entirely free. However, the number of inferences per day is limited in Huggingface. More cost-effective, especially cloud-related services can be exploited to deploy the application.

## 6. LIST OF REFERENCES

- AIX-COVNET *et al.* (2021) ‘Common pitfalls and recommendations for using machine learning to detect and prognosticate for COVID-19 using chest radiographs and CT scans’, *Nature Machine Intelligence*, 3(3), pp. 199–217. Available at: <https://doi.org/10.1038/s42256-021-00307-0>.
- Amari, S. (1993) ‘Backpropagation and stochastic gradient descent method’, *Neurocomputing*, 5(4), pp. 185–196. Available at: [https://doi.org/10.1016/0925-2312\(93\)90006-O](https://doi.org/10.1016/0925-2312(93)90006-O).
- CheXNet-Keras/weights.py at master · brucechou1983/CheXNet-Keras* (no date) *GitHub*. Available at: <https://github.com/brucechou1983/CheXNet-Keras> (Accessed: 13 November 2022).
- Gans, J.S. *et al.* (2022) ‘False-Positive Results in Rapid Antigen Tests for SARS-CoV-2’, *JAMA*, 327(5), pp. 485–486. Available at: <https://doi.org/10.1001/jama.2021.24355>.
- Ghoshal, B. and Tucker, A. (2020) ‘Estimating Uncertainty and Interpretability in Deep Learning for Coronavirus (COVID-19) Detection’, *arXiv:2003.10769 [cs, eess, stat]* [Preprint]. Available at: <http://arxiv.org/abs/2003.10769> (Accessed: 10 March 2022).
- He, K. *et al.* (2015) ‘Deep Residual Learning for Image Recognition’. arXiv. Available at: <https://doi.org/10.48550/arXiv.1512.03385>.
- Huang, G. *et al.* (2018) ‘Densely Connected Convolutional Networks’. arXiv. Available at: <http://arxiv.org/abs/1608.06993> (Accessed: 12 November 2022).
- Interpretable artificial intelligence framework for COVID-19 screening on chest X-rays* (no date). Available at: <https://www.spandidos-publications.com/10.3892/etm.2020.8797> (Accessed: 10 March 2022).
- Jain, R. *et al.* (2021) ‘Deep learning based detection and analysis of COVID-19 on chest X-ray images’, *Applied Intelligence*, 51(3), pp. 1690–1700.
- Kingma, D.P. and Ba, J. (2017) ‘Adam: A Method for Stochastic Optimization’. arXiv. Available at: <https://doi.org/10.48550/arXiv.1412.6980>.
- Luz, E. *et al.* (2021) ‘Towards an effective and efficient deep learning model for COVID-19 patterns detection in X-ray images’, *Research on Biomedical Engineering* [Preprint]. Available at: <https://doi.org/10.1007/s42600-021-00151-6>.
- de Moura, J., Novo, J. and Ortega, M. (2022) ‘Fully automatic deep convolutional approaches for the analysis of COVID-19 using chest X-ray images’, *Applied Soft Computing*, 115, p. 108190.
- Nayak, S.R. *et al.* (2021) ‘Application of deep learning techniques for detection of COVID-19 cases using chest X-ray images: A comprehensive study’, *Biomedical Signal Processing and Control*, 64, p. 102365.
- Rahaman, M.M. *et al.* (2020) ‘Identification of COVID-19 samples from chest X-Ray images using deep learning: A comparison of transfer learning approaches’, *Journal of X-Ray Science and Technology*, 28(5), pp. 821–839. Available at: <https://doi.org/10.3233/XST-200715>.
- Rajpurkar, P. *et al.* (2017) ‘CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning’. arXiv. Available at: <https://doi.org/10.48550/arXiv.1711.05225>.

Reza, A.M. (2004) ‘Realization of the Contrast Limited Adaptive Histogram Equalization (CLAHE) for Real-Time Image Enhancement’, *Journal of VLSI signal processing systems for signal, image and video technology*, 38(1), pp. 35–44. Available at: <https://doi.org/10.1023/B:VLSI.0000028532.53893.82>.

Selvaraju, R.R. *et al.* (2020) ‘Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization’, *International Journal of Computer Vision*, 128(2), pp. 336–359. Available at: <https://doi.org/10.1007/s11263-019-01228-7>.

Simonyan, K. and Zisserman, A. (2015) ‘Very Deep Convolutional Networks for Large-Scale Image Recognition’. arXiv. Available at: <https://doi.org/10.48550/arXiv.1409.1556>.

Simpson, S. *et al.* (2020) ‘Radiological Society of North America Expert Consensus Document on Reporting Chest CT Findings Related to COVID-19: Endorsed by the Society of Thoracic Radiology, the American College of Radiology, and RSNA’, *Radiology: Cardiothoracic Imaging*, 2(2), p. e200152. Available at: <https://doi.org/10.1148/ryct.2020200152>.

Szegedy, C. *et al.* (2015) ‘Rethinking the Inception Architecture for Computer Vision’. arXiv. Available at: <https://doi.org/10.48550/arXiv.1512.00567>.

Tartaglione, E. *et al.* (2020) ‘Unveiling COVID-19 from CHEST X-Ray with Deep Learning: A Hurdles Race with Small Data’, *International Journal of Environmental Research and Public Health*, 17(18), p. 6933. Available at: <https://doi.org/10.3390/ijerph17186933>.

Zhang, R. *et al.* (2021) ‘Diagnosis of Coronavirus Disease 2019 Pneumonia by Using Chest Radiography: Value of Artificial Intelligence’, *Radiology*, 298(2), pp. E88–E97. Available at: <https://doi.org/10.1148/radiol.2020202944>.