

5/25/2022

PROJECT TITLE:

Detection and highlighting of affected areas of COVID-19 and Pneumonia in Chest X-Rays

GUIDE NAME: Dr. K. Lalitha Devi

TEAM MEMBERS:

Name	Roll Number
Avantika Sivakumar	2019103007
Shruthi Gopalakrishnan	2019103061
Sweta Chakravarthi	2019103070

PROBLEM STATEMENT

As we know, COVID-19 has affected so many people around the world and early detection of covid patients is very important in preventing the spread of the virus. Adding on to this we also know that pneumonia is a severe lung disease. Both these diseases are deadly and cause severe injuries to our health. Hence it is important to devise a model that can classify based on chest x-rays and highlight the region affected. The project aims to tackle this issue.

OVERALL OBJECTIVE

Deep learning and convolutional neural networks (CNN) have been playing a major role in the classification of medical images for detecting diseases like Coronary Artery Disease, Malaria, Alzheimer's disease, different dental diseases, and Parkinson's disease. CNN also helps in detecting COVID-19 and Pneumonia from chest X-ray images.

This project is designed to help the medical field in fast detection of the COVID-19 and non-COVID-19 viral pneumonia diseases. It aims to create a model with multiple layer that is trained with a volatile dataset including images of both normal and affected chest x-rays. The model is also validated using a dataset contained images from both categories, normal and infected. The image is passed through the several layers of the CNN, the presence of COVID-19 or/and pneumonia is indicated.

In first phase a classification model is developed to classify COVID-19, pneumonia or normal and in second phase a model is built for segmentation task on given image. The segmentation model will activate only when the classification model classifies an X-ray image as COVID-19 or pneumonia.

Taking a real-time COVID-19 test is expensive and time consuming. CNN can automatically detect the presence of COVID-19 virus in patients. This can save both time and money which will eventually save lives. Moreover, this can add an extra layer of validation as the prevailing tests are not always reliable.

INTRODUCTION

Chest X-rays produce images of your heart, lungs, blood vessels, airways, and the bones of your chest and spine. Chest X-rays are a common type of exam. A chest X-ray is often among the first procedures you'll have if your doctor suspects heart or lung disease. A chest X-ray can also be used to check how you are responding to treatment. Chest X-rays can detect cancer, infection or air collecting in the space around a lung, which can cause the lung to collapse. They can also show chronic lung conditions, such as COVID-19, Pneumonia, or cystic fibrosis, as well as complications related to these conditions.

COVID-19 is a disease caused by a virus named SARS-CoV-2 and was discovered in December 2019 in Wuhan, China. It is very contagious and has quickly spread around the world. COVID-19 most often causes respiratory symptoms that can feel much like a cold, a flu, or pneumonia. COVID-19 may attack more than your lungs and respiratory system. Other parts of your body may also be affected by the disease. COVID-19 is detected by the popularly known RT-PCR test. An RT-PCR test is a laboratory test that combines reverse transcription of RNA into DNA for the detection of the virus.

Similarly, Pneumonia is an infection that inflames the air sacs in one or both lungs. The air sacs may fill with fluid or pus (purulent material), causing cough with phlegm or pus, fever, chills, and difficulty breathing. A variety of organisms, including bacteria, viruses and fungi, can cause pneumonia. Pneumonia can range in seriousness from mild to life-threatening. It is most serious for infants and young children, people older than age 65, and people with health problems or weakened immune systems.

A chest x-ray is the typical imaging test used to diagnose COVID-19 and pneumonia. This testing can show the presence of the infection in the lungs. Statistical results obtained demonstrates that pretrained CNN models employed along with supervised classifier algorithms can be very beneficial in analyzing chest X-ray images, specifically to detect COVID-19 and Pneumonia.

Medical tests nowadays are not the first choice since money is quantized in today's medical field and hence taking those tests are really expensive. There are changes of the tests showing a false positive or a false negative which makes the English method of screening to be unreliable. Whereas researchers have successfully applied CNNs for many medical image understanding applications and proved to show better and more accurate detection results.

The CNN model we have chosen for implementation is ResNet50. It is a convolutional neural network that is 50 layers deep. A model is made with multiple layer that is trained with a volatile dataset including images of both normal and affected chest x-rays. The model is also validated using a dataset contained images from both categories, normal and infected. The image is passed through the several layers of the CNN, the presence of COVID-19 or/and pneumonia is indicated. In first phase a classification model is developed to classify COVID-19, pneumonia or normal and in second phase a model is built for segmentation task on given image. The segmentation model will activate only when the classification model classifies an X-ray image as COVID-19 or pneumonia.

As a prerequisite to classification the images are pre-processed to remove noise, resize the images and normalize the pixel values. Following this the dataset is split into training and validation dataset. A pretrained CNN model is imported from keras which was trained using the training data. Layers were added to the imported model for classification, which was done by a fully connected output layer.

After classification, thresholding and binary masking is done on the affected chest X-rays. Otsu's thresholding has been used. Otsu's method, named after its inventor Nobuyuki Otsu, is one of many binarization algorithms. Otsu's thresholding method involves iterating through all the possible threshold values and calculating a measure of spread for the pixel levels each side of the threshold, i.e. the pixels that either fall in foreground or background. The aim is to find the threshold value where the sum of foreground and background spreads is at its minimum. When a mask is applied to a thresholded image of the same size, all pixels which are zero in the mask are set to zero in the output image. All others remain unchanged.

The masked image is then segmented using Grad-CAM algorithm which highlights the affected region in the chest X-Rays. Gradient-weighted Class Activation Mapping (Grad-CAM), uses the gradients of any target concept flowing into the final convolutional layer to produce a coarse localization map highlighting the important regions in the image for predicting the concept.

When chest X-Ray images are given as input, the model segments the presence of COVID-19 or pneumonia in the chest x-rays. A heatmap is generated as the output which highlights the affected region which helps practitioners for convenient diagnostics.

SUMMARY OF RELATED WORKS

The aim of the project was to build a classification model that would accurately categorise chest X-Rays as affected or not. We decided to make use of various image processing and computer vision techniques to analyse and process our images. Deep learning algorithms were also utilised for the successful classification of the skin lesions.

One of the most important implementations of covid/pneumonia classification with CNNs was achieved by Iosif Mporas(2020). The authors present an evaluation of several pre-trained deep convolutional neural network (CNN) models on the detection of positive cases the new COVID-19 virus from chest X-ray images. Different experimental setups are tested using two X-ray datasets either separately to each other or by mixing them.

In 2021 by K. Lee et al. analysed chest X-ray and CT images from nine COVID-19 infected patients by two radiologists to assess the correspondence of abnormal findings on X-rays with those on CT images. M. Mishra et.al. proposed a transfer learning based pneumonia and COVID-19 detection process from chest-X-ray image of open source dataset and achieved 98.2% accuracy in COVID-19 classification. Narin et al. evaluated different CNN for the diagnosis of COVID-19 and achieved an accuracy of 98% using a pre-trained ResNet50 model for single class classification.

In another study by T. Ozturk et al. [11], proposed a deep learning model called DarkCovidNet for the automatic diagnosis of COVID-19 based on 125 chest X-ray images to diagnosis i) binary classification (COVID-19 vs no-findings) and ii) multiclass classification (COVID-19 vs no-findings vs pneumonia) and report accuracy 98.08% on binary and 87.02% on multiclass classification task.

The emergence of deep learning has made it possible to study thousands of patient samples, and the use of deep learning technology can achieve accurate and reliable predictions. In 2020, Kallianos et al. published a review explaining the importance of artificial intelligence in chest X-ray image classification and analysis. Wang et al. solved this problem and prepared a new database ChestX-ray8, containing 108,948 front view chest x-ray images, containing 32,717 unique patients. Each chest X-ray image has multiple tags. They used deep convolutional neural networks to verify the results of using these data and achieved good

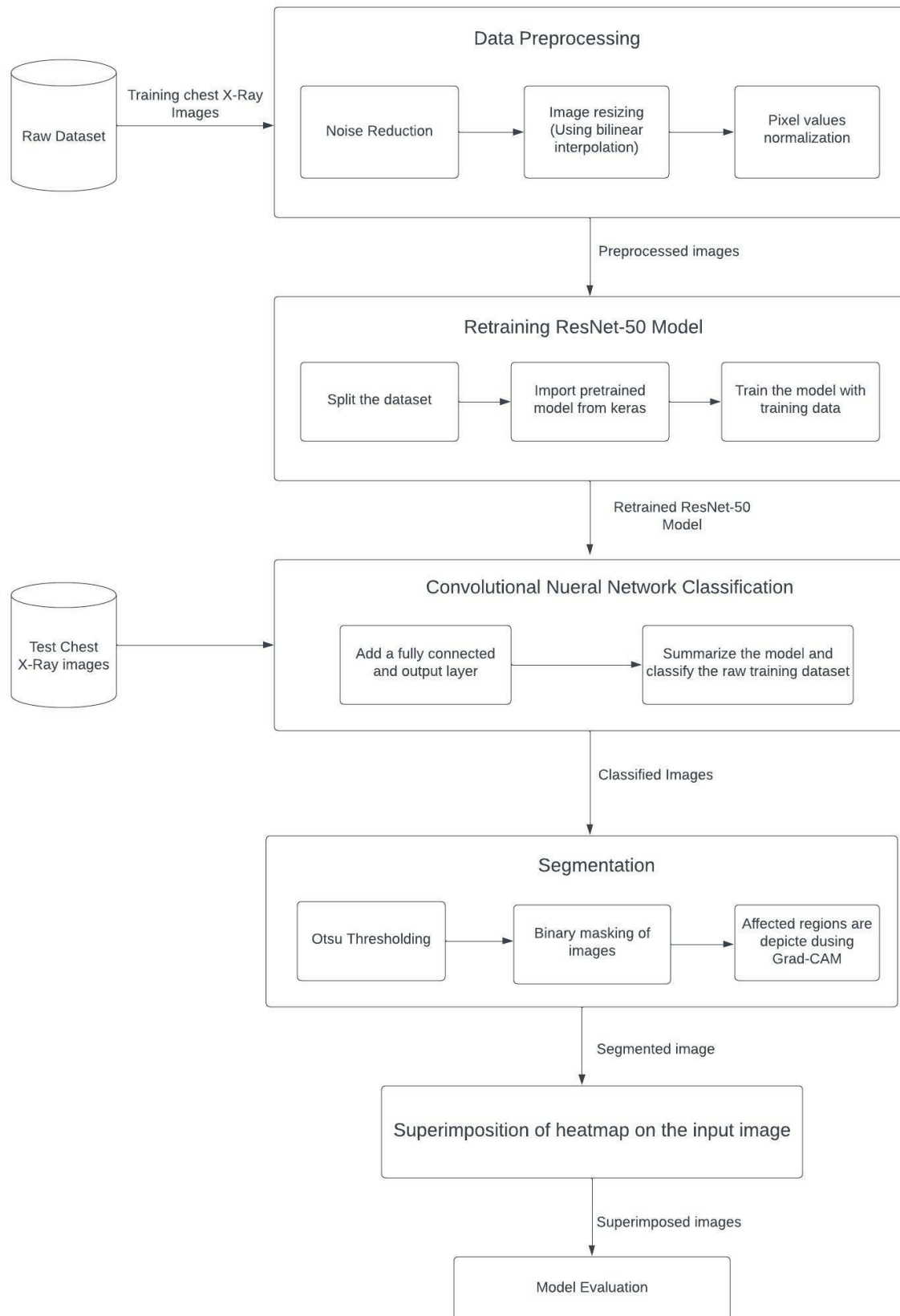
results. Rajpurkar et al. developed a 121-layer deep convolutional neural network and tried it on the ChestX-ray14 dataset.

Irvin et al. [9] pointed out that largescale labeled data sets are the key to the success of prediction and classification tasks. They provided a huge data set including 224,316 chest radiographs of 65,240 patients. They named this dataset CheXpert. They use convolutional neural networks to assign labels based on the probability assigned by the model. The target detection in medical imaging is mainly to solve the problem of target positioning, such as pathological positioning, nuclear detection, and abnormal point detection. This article is to solve the problem of pneumonia positioning in chest X-rays. It mainly proposes a scheme for ensemble of different models RetinaNet and Mask R-CNN for pneumonia detection. Among them, single model RetinaNet and Mask RCNN are models trained by different backbone networks ensemble.

Recent findings indicate the presence of COVID-19 in patients with irregular findings on chest X-rays. There are many reports on this topic that include machine learning strategies for the identification of COVID-19 using chest X-rays. Other current studies have used non-public datasets and complex artificial intelligence (AI) systems. In 2021, Ahmed Hamza Osman ,Hani Moetque Aljahdali, Sultan Menwer Altarrazi, Ali Ahmed suggested a new COVID-19 identification technique based on the locality-weighted learning and self-organization map (LWL-SOM) strategy for detecting and capturing COVID-19 cases. They first grouped images from chest X-ray datasets based on their similar features in different clusters using the SOM strategy in order to discriminate between the COVID-19 and non-COVID-19 cases. Then, they built an intelligent learning model based on the LWL algorithm to diagnose and detect COVID-19 cases. The proposed SOM-LWL model improved the correlation coefficient performance results between the Covid19, no-finding, and pneumonia cases; pneumonia and no-finding cases; Covid19 and pneumonia cases; and Covid19 and no-finding cases from 0.9613 to 0.9788, 0.6113 to 1 0.8783 to 0.9999, and 0.8894 to 1, respectively. The proposed LWL-SOM had better results for discriminating COVID-19 and non-COVID-19 patients than the current machine learning-based solutions using AI evaluation measures.

Method	Performance	Advantages	Disadvantages
Thin-slice chest CT	A full score for COVID-19 in 155 of the 167 patients (92.8%)	The low sensitivity of RT-PCR screening tools (60–70%) allows symptoms to be detected by analysing radiographic images of patients. Thin-slice chest CT is simple to administer, swift, and highly sensitive to early COVID-19 pneumonia, offering useful evidence for further diagnosis while helping to avoid and monitor COVID-19.	<ul style="list-style-type: none"> The method used the CT tool which it is a sensitive diagnostic tool for COVID-19 pneumonia diagnosis. CT results are often found long after symptoms occur, and patients typically undergo CT analysis within the first 0 to 2 days
<ul style="list-style-type: none"> Crazy-paving pattern and GGO. Quantitative analysis using SPSS. 	A cumulative CT score of 0 (no involvement) to 25 (maximum involvement) was calculated as the amount of lung inference.	Determine improvements from original diagnostic up to patient recuperation with COVID-19-related Chest CT findings.	In research on lung X-rays, the most severe lung illness was found 10 days after symptoms were shown in patients who survived COVID-19 pneumonia
RT-PCR	60–70% sensitivity	The low sensitivity of RT-PCR (60–70%) allows symptoms to be detected by analysing radiographic images of patients. but on initial negative RT-PCR	CT results are can be found in initial negative RT-PCR only due to abnormalities on chest CT scan images.
Thin-slice chest CT	A full dant for COVID-19 in 155 of the 167 patients (92.8%)	The CT system used for COVID-19 results involves multifocal floor-to-ground (GGO's) peripherally scattered with patchy consolidations and tastes in the back and under lobe. In early identification, observation and disease assessment, chest CT played a crucial role.	It is uncertain that if chest x-rays are regular, the criterion for undertaking CT tests of probable lung changes may be smaller. Further experiments are required to increase the selection of CT patients, to identify the effectiveness of CT in COVID-19 pneumonia and to investigate the use of artificial intelligence in chest X-rays in suspicious cases. The COVID-19 can be detected only using CT data only rather than other types of dataset.
Viral pneumonia CT diagnosis Method	In chest CT there was a low diagnosis incidence of COVID-19 missing (3.9%, 2/51)	The method can able to determine and evaluate the mis-diagnosis error of radiologists for COVID-19	The method still limited for recognising distinguish viruses and distinctive between them. During the research time the number of patients was reduced by the lack of laboratory test kits.
Chest CT Interaction Results and Coronavirus Clinical Conditions	In emergency patients, the prevalence of diffuse lesions was higher than in the non-emergency population (78.6% vs 24.1%).	The study discusses medical and technical viewpoints to promote the outbreak of COVID-19 by virologists, policymakers and IA researchers.	The paper has taken initial steps in compiling and highlighting existing state-of-the-art, but does not discriminate between working cases in wild and in laboratory circumstances.
COVIDX-Net, VGG19 and (DenseNet)	f1-scores of VGG19 is 0.89% and DenseNet is 0.91%	The technique allows radiologists to detect COVID-19 instantly in X-ray images	X-ray scans cannot differentiate between the soft tissue and the medium dose to minimize exposure to the patients
Convolutional Neural Network (CNN) and CLSTM-based deep learning models	91% accuracy for the ConvLSTM DLMS and the CNN and	The method Improved the learning capacities of the Convolutional Neural Network (CNN) and CLSTM-based deep learning models (DADLMs) by introduced a two machine learning models to in order to enhance the prediction accuracy of COVID-19 identification.	The method has been investigated under two machine learning techniques SVM and k-NN only.
Deep learning model	92.4% classification accuracy	The model classified and define groups as regular, non-COVID, and COVID-19 based on a deep learning model with accuracy of 92.4%	The method is need to be improved in term of defining and classification accuracy.
CNN	98.3% accuracy and a precision of 96.72%	This model identifies Coronavirus patients with very little time and energy, and is CNN models very accurate. In their work, the CNN models in COVID-19 are also studied in a comparative analysis.	The study preserving the dataset images with transformed to 224×224 pixels due to the image quality. Actually, the converting process is considered as an extra step before using CNN model.
ResNet50 plus SVM	The accuracy of SVM scored, 95.38%, 91.41%, 95.52%, 90.76% for FPR, MCC, F1-score, and Kappa respectively for COVID-19 detection	The method identified the features extracted from various CNN models using X-ray images and employed a support vector machine (SVM). Their analysis notes the highest results of the ResNet50 model with the SVM classifier.	The method was used the SVM only rather than other machine learning techniques.
FrMEMs approach	The classification accuracy scores with 96.09% and 98.09% for the COVID-19 datasets.	The model used the FrMEMs approach to take advantage of chest X-ray images features. The computing process was accelerated using a parallel multi-core computational architecture.	The limitation of the method is that the time of the CPU is considered as the third rank.

SYSTEM ARCHITECTURE



DETAILS OF MODULE DESIGN

1. Classification:

The raw dataset is split into training and testing dataset. The dataset is preprocessed to fit the chosen pretrained CNN model. The X-Ray images are classified using ResNet-50 CNN model. The images are classified either as affected by covid-19 or pneumonia or normal.

Input: Chest X-ray dataset is passed through ResNet-50

Output: The images are classified into different classes

2. Thresholding and Binary masking:

The chest X-Rays that are classified as affected go through Otsu Thresholding to change the pixels of an image and change it from grayscale into a binary image. The images are then binary masked to change specific bits in the original value to the desired setting(s) or to create a specific output value

Input: Classified images(COVID-19/Pneumonia) go through Otsu Thresholding and binary masking

Output: Binary masked images are generated

3. Segmentation and Superimposition:

The regions of the image that have a positive contribution to the predefined class of COVID-19 and pneumonia are depicted using Grad-CAM. The predicted heatmap is superimposed on the input image to give the final predicted segmented image. The segmentation model will activate if the ensemble classification model detects an X-ray image as COVID-19 or pneumonia so that radiology expert can easily analyse and diagnose this problem.

Input: Binary masked images of the chest X-Rays

Output: The final superimposed images are generated with affected region highlighted

IMPLEMENTATION

1. Importing all required Libraries and unzipping the dataset folder

```
import matplotlib.pyplot as plt
import numpy as np
import PIL
import tensorflow as tf
from tensorflow.keras import layers
from keras.layers.core import Dense, Flatten
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam

import pathlib
!unzip -u "/content/drive/MyDrive/COVID-19.zip" -d "/content/Dataset/"
!unzip -u "/content/drive/MyDrive/Pneumonia.zip" -d "/content/Dataset/"
!unzip -u "/content/drive/MyDrive/Normal.zip" -d "/content/Dataset/"

data_dir=pathlib.Path("/content/Dataset")
print(data_dir)
```

Output:

```
Streaming output truncated to the last 5000 lines.
inflating: /content/Dataset/Normal/Normal-6116.png
inflating: /content/Dataset/Normal/NORMAL-6116579-0001.jpeg
inflating: /content/Dataset/Normal/Normal-6117.png
inflating: /content/Dataset/Normal/Normal-6118.png
inflating: /content/Dataset/Normal/Normal-6119.png
inflating: /content/Dataset/Normal/Normal-612.png
inflating: /content/Dataset/Normal/Normal-6120.png
inflating: /content/Dataset/Normal/Normal-6121.png
inflating: /content/Dataset/Normal/Normal-6122.png
inflating: /content/Dataset/Normal/Normal-6123.png
inflating: /content/Dataset/Normal/NORMAL-6123465-0001.jpeg
inflating: /content/Dataset/Normal/Normal-6124.png
inflating: /content/Dataset/Normal/Normal-6125.png
inflating: /content/Dataset/Normal/Normal-6126.png
inflating: /content/Dataset/Normal/NORMAL-6126748-0001.jpeg
inflating: /content/Dataset/Normal/Normal-6127.png
```

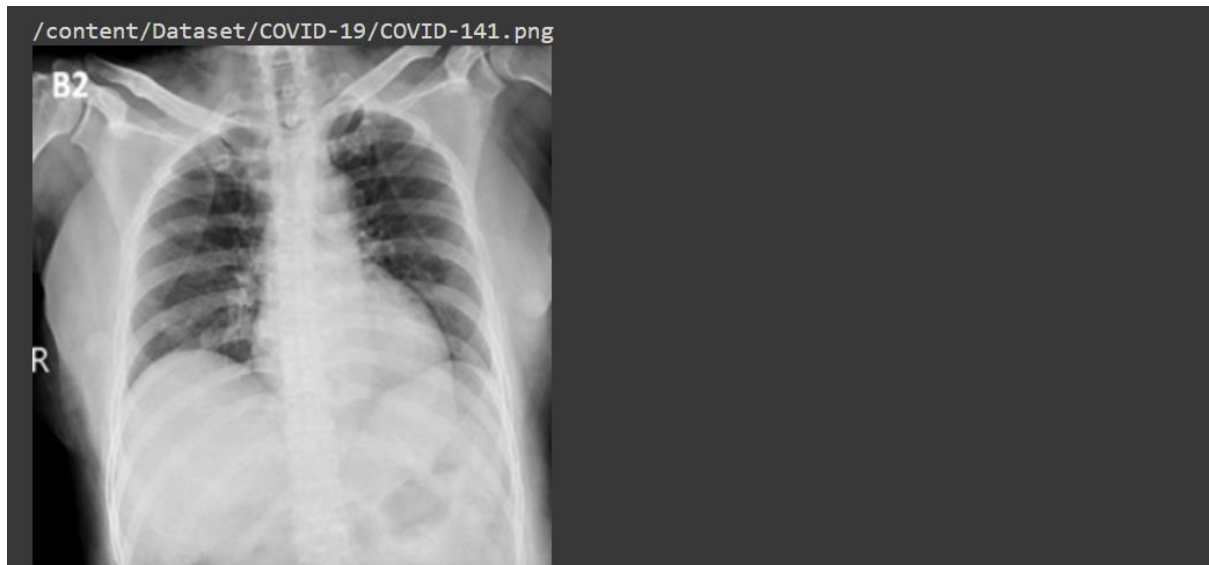
2. Importing the dataset

```
data_dir=pathlib.Path("/content/Dataset")
print(data_dir)
```

3. Printing a random image from the dataset

```
covid=list(data_dir.glob('COVID-19/*'))
print(covid[0])
PIL.Image.open(str(covid[0]))
```

Output:



4. Pre-process the data and split the dataset into training and testing

Training set:

```
img_height,img_width=224,224
batch_size=32
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)
```

Output:

```
Found 35524 files belonging to 3 classes.
Using 28420 files for training.
```

Validation set:

```
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)
```

Output:

```
Found 35524 files belonging to 3 classes.
Using 7104 files for validation.
```

5. Import pretrained model

```
from tensorflow.keras.applications import ResNet50
# load model
model = ResNet50()
# summarize the model
model.summary()
```

Output:

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50\_weights\_tf\_dim\_ordering
102973440/102967424 [=====] - 0s 0us/step
102981632/102967424 [=====] - 0s 0us/step
Model: "resnet50"

```

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 224, 224, 3)	0	input_1[0][0]
conv1_pad (ZeroPadding2D)	(None, 230, 230, 3)	0	conv1_pad[0][0]
conv1_conv (Conv2D)	(None, 112, 112, 64)	9472	conv1_conv[0][0]
conv1_bn (BatchNormalization)	(None, 112, 112, 64)	256	conv1_bn[0][0]
conv1_relu (Activation)	(None, 112, 112, 64)	0	conv1_relu[0][0]
pool1_pad (ZeroPadding2D)	(None, 114, 114, 64)	0	pool1_pad[0][0]
pool1_pool (MaxPooling2D)	(None, 56, 56, 64)	0	pool1_pool[0][0]

conv2_block1_1_conv (Conv2D)	(None, 56, 56, 64)	4160	['pool1_pool[0][0]']
conv2_block1_1_bn (BatchNormalization)	(None, 56, 56, 64)	256	['conv2_block1_1_conv[0][0]']
conv2_block1_1_relu (Activation)	(None, 56, 56, 64)	0	['conv2_block1_1_bn[0][0]']
conv2_block1_2_conv (Conv2D)	(None, 56, 56, 64)	36928	['conv2_block1_1_relu[0][0]']
conv2_block1_2_bn (BatchNormalization)	(None, 56, 56, 64)	256	['conv2_block1_2_conv[0][0]']
conv2_block1_2_relu (Activation)	(None, 56, 56, 64)	0	['conv2_block1_2_bn[0][0]']
conv2_block1_0_conv (Conv2D)	(None, 56, 56, 256)	16640	['pool1_pool[0][0]']
conv2_block1_3_conv (Conv2D)	(None, 56, 56, 256)	16640	['conv2_block1_2_relu[0][0]']
conv2_block1_0_bn (BatchNormalization)	(None, 56, 56, 256)	1024	['conv2_block1_0_conv[0][0]']
conv2_block1_3_bn (BatchNormalization)	(None, 56, 56, 256)	1024	['conv2_block1_3_conv[0][0]']
conv2_block1_add (Add)	(None, 56, 56, 256)	0	['conv2_block1_0_bn[0][0]',

conv2_block1_out (Activation)	(None, 56, 56, 256)	0	['conv2_block1_add[0][0]']
conv2_block2_1_conv (Conv2D)	(None, 56, 56, 64)	16448	['conv2_block1_out[0][0]']
conv2_block2_1_bn (BatchNormalization)	(None, 56, 56, 64)	256	['conv2_block2_1_conv[0][0]']
conv2_block2_1_relu (Activation)	(None, 56, 56, 64)	0	['conv2_block2_1_bn[0][0]']
conv2_block2_2_conv (Conv2D)	(None, 56, 56, 64)	36928	['conv2_block2_1_relu[0][0]']
conv2_block2_2_bn (BatchNormalization)	(None, 56, 56, 64)	256	['conv2_block2_2_conv[0][0]']
conv2_block2_2_relu (Activation)	(None, 56, 56, 64)	0	['conv2_block2_2_bn[0][0]']
conv2_block2_3_conv (Conv2D)	(None, 56, 56, 256)	16640	['conv2_block2_2_relu[0][0]']
conv2_block2_3_bn (BatchNormalization)	(None, 56, 56, 256)	1024	['conv2_block2_3_conv[0][0]']
conv2_block2_add (Add)	(None, 56, 56, 256)	0	['conv2_block1_out[0][0]', 'conv2_block2_3_bn[0][0]']
conv2_block2_out (Activation)	(None, 56, 56, 256)	0	['conv2_block2_add[0][0]']

conv2_block3_1_conv (Conv2D)	(None, 56, 56, 64)	16448	['conv2_block2_out[0][0]']
conv2_block3_1_bn (BatchNormalization)	(None, 56, 56, 64)	256	['conv2_block3_1_conv[0][0]']
conv2_block3_1_relu (Activation)	(None, 56, 56, 64)	0	['conv2_block3_1_bn[0][0]']
conv2_block3_2_conv (Conv2D)	(None, 56, 56, 64)	36928	['conv2_block3_1_relu[0][0]']
conv2_block3_2_bn (BatchNormalization)	(None, 56, 56, 64)	256	['conv2_block3_2_conv[0][0]']
conv2_block3_2_relu (Activation)	(None, 56, 56, 64)	0	['conv2_block3_2_bn[0][0]']

conv2_block3_3_bn (BatchNormalization)	(None, 56, 56, 256)	1024	['conv2_block3_3_conv[0][0]']
conv2_block3_add (Add)	(None, 56, 56, 256)	0	['conv2_block2_out[0][0]', 'conv2_block3_3_bn[0][0]']
conv2_block3_out (Activation)	(None, 56, 56, 256)	0	['conv2_block3_add[0][0]']
conv3_block1_1_conv (Conv2D)	(None, 28, 28, 128)	32896	['conv2_block3_out[0][0]']
conv3_block1_1_bn (BatchNormalization)	(None, 28, 28, 128)	512	['conv3_block1_1_conv[0][0]']
conv3_block1_1_relu (Activation)	(None, 28, 28, 128)	0	['conv3_block1_1_bn[0][0]']
conv3_block1_2_conv (Conv2D)	(None, 28, 28, 128)	147584	['conv3_block1_1_relu[0][0]']
conv3_block1_2_bn (BatchNormalization)	(None, 28, 28, 128)	512	['conv3_block1_2_conv[0][0]']
conv3_block1_2_relu (Activation)	(None, 28, 28, 128)	0	['conv3_block1_2_bn[0][0]']
conv3_block1_0_conv (Conv2D)	(None, 28, 28, 512)	131584	['conv2_block3_out[0][0]']
conv3_block1_3_conv (Conv2D)	(None, 28, 28, 512)	66048	['conv3_block1_2_relu[0][0]']

conv3_block1_3_bn (BatchNormalization)	(None, 28, 28, 512)	2048	['conv3_block1_3_conv[0][0]']
conv3_block1_add (Add)	(None, 28, 28, 512)	0	['conv3_block1_0_bn[0][0]', 'conv3_block1_3_bn[0][0]']
conv3_block1_out (Activation)	(None, 28, 28, 512)	0	['conv3_block1_add[0][0]']
conv3_block2_1_conv (Conv2D)	(None, 28, 28, 128)	65664	['conv3_block1_out[0][0]']
conv3_block2_1_bn (BatchNormalization)	(None, 28, 28, 128)	512	['conv3_block2_1_conv[0][0]']
conv3_block2_1_relu (Activation)	(None, 28, 28, 128)	0	['conv3_block2_1_bn[0][0]']
conv3_block2_2_conv (Conv2D)	(None, 28, 28, 128)	147584	['conv3_block2_1_relu[0][0]']
conv3_block2_2_bn (BatchNormalization)	(None, 28, 28, 128)	512	['conv3_block2_2_conv[0][0]']
conv3_block2_2_relu (Activation)	(None, 28, 28, 128)	0	['conv3_block2_2_bn[0][0]']
conv3_block2_3_conv (Conv2D)	(None, 28, 28, 512)	66048	['conv3_block2_2_relu[0][0]']

conv3_block2_add (Add)	(None, 28, 28, 512)	0	['conv3_block1_out[0][0]', 'conv3_block2_3_bn[0][0]']
conv3_block2_out (Activation)	(None, 28, 28, 512)	0	['conv3_block2_add[0][0]']
conv3_block3_1_conv (Conv2D)	(None, 28, 28, 128)	65664	['conv3_block2_out[0][0]']
conv3_block3_1_bn (BatchNormalization)	(None, 28, 28, 128)	512	['conv3_block3_1_conv[0][0]']
conv3_block3_1_relu (Activation)	(None, 28, 28, 128)	0	['conv3_block3_1_bn[0][0]']
conv3_block3_2_conv (Conv2D)	(None, 28, 28, 128)	147584	['conv3_block3_1_relu[0][0]']
conv3_block3_2_bn (BatchNormalization)	(None, 28, 28, 128)	512	['conv3_block3_2_conv[0][0]']

conv3_block3_out (Activation)	(None, 28, 28, 512)	0	['conv3_block3_add[0][0]']
conv3_block4_1_conv (Conv2D)	(None, 28, 28, 128)	65664	['conv3_block3_out[0][0]']
conv3_block4_1_bn (BatchNormalization)	(None, 28, 28, 128)	512	['conv3_block4_1_conv[0][0]']
conv3_block4_1_relu (Activation)	(None, 28, 28, 128)	0	['conv3_block4_1_bn[0][0]']
conv3_block4_2_conv (Conv2D)	(None, 28, 28, 128)	147584	['conv3_block4_1_relu[0][0]']
conv3_block4_2_bn (BatchNormalization)	(None, 28, 28, 128)	512	['conv3_block4_2_conv[0][0]']
conv3_block4_2_relu (Activation)	(None, 28, 28, 128)	0	['conv3_block4_2_bn[0][0]']
conv3_block4_3_conv (Conv2D)	(None, 28, 28, 512)	66048	['conv3_block4_2_relu[0][0]']
conv3_block4_3_bn (BatchNormalization)	(None, 28, 28, 512)	2048	['conv3_block4_3_conv[0][0]']
conv3_block4_add (Add)	(None, 28, 28, 512)	0	['conv3_block3_out[0][0]', 'conv3_block4_3_bn[0][0]']
conv3_block4_out (Activation)	(None, 28, 28, 512)	0	['conv3_block4_add[0][0]']
conv4_block1_1_bn (BatchNormalization)	(None, 14, 14, 256)	1024	['conv4_block1_1_conv[0][0]']
conv4_block1_1_relu (Activation)	(None, 14, 14, 256)	0	['conv4_block1_1_bn[0][0]']
conv4_block1_2_conv (Conv2D)	(None, 14, 14, 256)	590080	['conv4_block1_1_relu[0][0]']
conv4_block1_2_bn (BatchNormalization)	(None, 14, 14, 256)	1024	['conv4_block1_2_conv[0][0]']
conv4_block1_2_relu (Activation)	(None, 14, 14, 256)	0	['conv4_block1_2_bn[0][0]']
conv4_block1_0_conv (Conv2D)	(None, 14, 14, 1024)	525312	['conv3_block4_out[0][0]']
conv4_block1_3_conv (Conv2D)	(None, 14, 14, 1024)	263168	['conv4_block1_2_relu[0][0]']
conv4_block1_0_bn (BatchNormalization)	(None, 14, 14, 1024)	4096	['conv4_block1_0_conv[0][0]']
conv4_block1_3_bn (BatchNormalization)	(None, 14, 14, 1024)	4096	['conv4_block1_3_conv[0][0]']
conv4_block2_1_bn (BatchNormalization)	(None, 14, 14, 256)	1024	['conv4_block2_1_conv[0][0]']
conv4_block2_1_relu (Activation)	(None, 14, 14, 256)	0	['conv4_block2_1_bn[0][0]']
conv4_block2_2_conv (Conv2D)	(None, 14, 14, 256)	590080	['conv4_block2_1_relu[0][0]']
conv4_block2_2_bn (BatchNormalization)	(None, 14, 14, 256)	1024	['conv4_block2_2_conv[0][0]']
conv4_block2_2_relu (Activation)	(None, 14, 14, 256)	0	['conv4_block2_2_bn[0][0]']
conv4_block2_3_conv (Conv2D)	(None, 14, 14, 1024)	263168	['conv4_block2_2_relu[0][0]']

conv4_block2_3_bn (BatchNormalization)	(None, 14, 14, 1024)	4096	['conv4_block2_3_conv[0][0]']
conv4_block2_add (Add)	(None, 14, 14, 1024)	0	['conv4_block1_out[0][0]', 'conv4_block2_3_bn[0][0]']
conv4_block2_out (Activation)	(None, 14, 14, 1024)	0	['conv4_block2_add[0][0]']
conv4_block3_1_conv (Conv2D)	(None, 14, 14, 256)	262400	['conv4_block2_out[0][0]']
conv4_block3_1_bn (BatchNormalization)	(None, 14, 14, 256)	1024	['conv4_block3_1_conv[0][0]']
conv4_block3_1_relu (Activation)	(None, 14, 14, 256)	0	['conv4_block3_1_bn[0][0]']
conv4_block3_2_conv (Conv2D)	(None, 14, 14, 256)	590080	['conv4_block3_1_relu[0][0]']
conv4_block3_2_bn (BatchNormalization)	(None, 14, 14, 256)	1024	['conv4_block3_2_conv[0][0]']
conv4_block3_2_relu (Activation)	(None, 14, 14, 256)	0	['conv4_block3_2_bn[0][0]']
conv4_block3_3_conv (Conv2D)	(None, 14, 14, 1024)	263168	['conv4_block3_2_relu[0][0]']
conv4_block3_3_bn (BatchNormalization)	(None, 14, 14, 1024)	4096	['conv4_block3_3_conv[0][0]']
conv4_block3_add (Add)	(None, 14, 14, 1024)	0	['conv4_block2_out[0][0]', 'conv4_block3_3_bn[0][0]']
conv4_block3_out (Activation)	(None, 14, 14, 1024)	0	['conv4_block3_add[0][0]']
conv4_block4_1_conv (Conv2D)	(None, 14, 14, 256)	262400	['conv4_block3_out[0][0]']
conv4_block4_1_bn (BatchNormalization)	(None, 14, 14, 256)	1024	['conv4_block4_1_conv[0][0]']
conv4_block4_1_relu (Activation)	(None, 14, 14, 256)	0	['conv4_block4_1_bn[0][0]']
conv4_block4_2_conv (Conv2D)	(None, 14, 14, 256)	590080	['conv4_block4_1_relu[0][0]']
conv4_block4_2_bn (BatchNormalization)	(None, 14, 14, 256)	1024	['conv4_block4_2_conv[0][0]']
conv4_block4_2_relu (Activation)	(None, 14, 14, 256)	0	['conv4_block4_2_bn[0][0]']
conv4_block4_3_conv (Conv2D)	(None, 14, 14, 1024)	263168	['conv4_block4_2_relu[0][0]']
conv5_block3_add (Add)	(None, 7, 7, 2048)	0	['conv5_block2_out[0][0]', 'conv5_block3_3_bn[0][0]']
conv5_block3_out (Activation)	(None, 7, 7, 2048)	0	['conv5_block3_add[0][0]']
avg_pool (GlobalAveragePooling2D)	(None, 2048)	0	['conv5_block3_out[0][0]']
predictions (Dense)	(None, 1000)	2049000	['avg_pool[0][0]']

=====

Total params: 25,636,712
Trainable params: 25,583,592
Non-trainable params: 53,120

6. Training the model over 10 epochs

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

/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/adam.py:105: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead
  super(Adam, self).__init__(name, **kwargs)

model.compile(optimizer='adam',loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])

history=model.fit(train_ds,validation_data=val_ds,epochs=10)
```

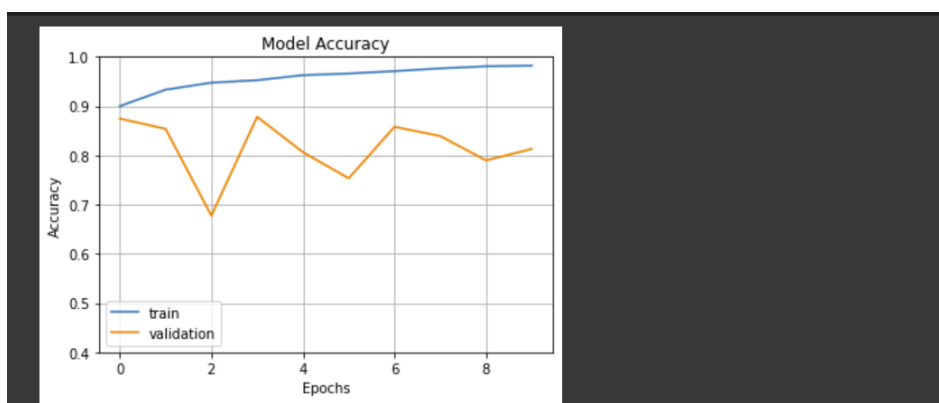
Output:

```
Epoch 1/10
/usr/local/lib/python3.7/dist-packages/tensorflow/python/util/dispatch.py:1082: UserWarning: "`sparse_categorical_crossentropy` received
  return dispatch_target(*args, **kwargs)
889/889 [=====] - 358s 380ms/step - loss: 0.2935 - accuracy: 0.9000 - val_loss: 0.3311 - val_accuracy: 0.8747
Epoch 2/10
889/889 [=====] - 343s 384ms/step - loss: 0.1843 - accuracy: 0.9333 - val_loss: 0.4569 - val_accuracy: 0.8537
Epoch 3/10
889/889 [=====] - 343s 384ms/step - loss: 0.1460 - accuracy: 0.9475 - val_loss: 1.2347 - val_accuracy: 0.6775
Epoch 4/10
889/889 [=====] - 341s 382ms/step - loss: 0.1330 - accuracy: 0.9526 - val_loss: 0.3396 - val_accuracy: 0.8782
Epoch 5/10
889/889 [=====] - 341s 382ms/step - loss: 0.1026 - accuracy: 0.9628 - val_loss: 0.6248 - val_accuracy: 0.8063
Epoch 6/10
889/889 [=====] - 343s 385ms/step - loss: 0.0930 - accuracy: 0.9662 - val_loss: 1.0011 - val_accuracy: 0.7534
Epoch 7/10
889/889 [=====] - 341s 382ms/step - loss: 0.0810 - accuracy: 0.9710 - val_loss: 0.4492 - val_accuracy: 0.8580
Epoch 8/10
889/889 [=====] - 341s 383ms/step - loss: 0.0668 - accuracy: 0.9766 - val_loss: 0.7756 - val_accuracy: 0.8391
Epoch 9/10
889/889 [=====] - 341s 382ms/step - loss: 0.0559 - accuracy: 0.9808 - val_loss: 0.9712 - val_accuracy: 0.7897
Epoch 10/10
889/889 [=====] - 341s 382ms/step - loss: 0.0509 - accuracy: 0.9823 - val_loss: 0.8204 - val_accuracy: 0.8131
```

7. Model Evaluation:

```
fig1 = plt.gcf()
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.axis(ymin=0.4,ymax=1)
plt.grid()
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['train', 'validation'])
plt.show()
```

Output:



8. Model Inference

```
import cv2
image=cv2.imread('/content/Dataset/COVID-19/063.jpeg')
image_resized= cv2.resize(image, (img_height,img_width))
image=np.expand_dims(image_resized,axis=0)

pred=model.predict(image)

output_class=class_names[np.argmax(pred)]
print("The predicted class is", output_class)
```

Output:

```
The predicted class is COVID-19
```

9. Threshold – masking the affected chest X-ray

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
img = cv2.imread('/content/Dataset/COVID-19/054.jpeg')

# cv2.cvtColor is applied over the
# image input with applied parameters
# to convert the image in grayscale
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# applying Otsu thresholding
# as an extra flag in binary
# thresholding
ret, thresh1 = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

# the window showing output image
# with the cor
plt.imshow(img, cmap='gray')
plt.imshow(thresh1, cmap='gray')

t = 0.7
binary_mask = thresh1 < t
```

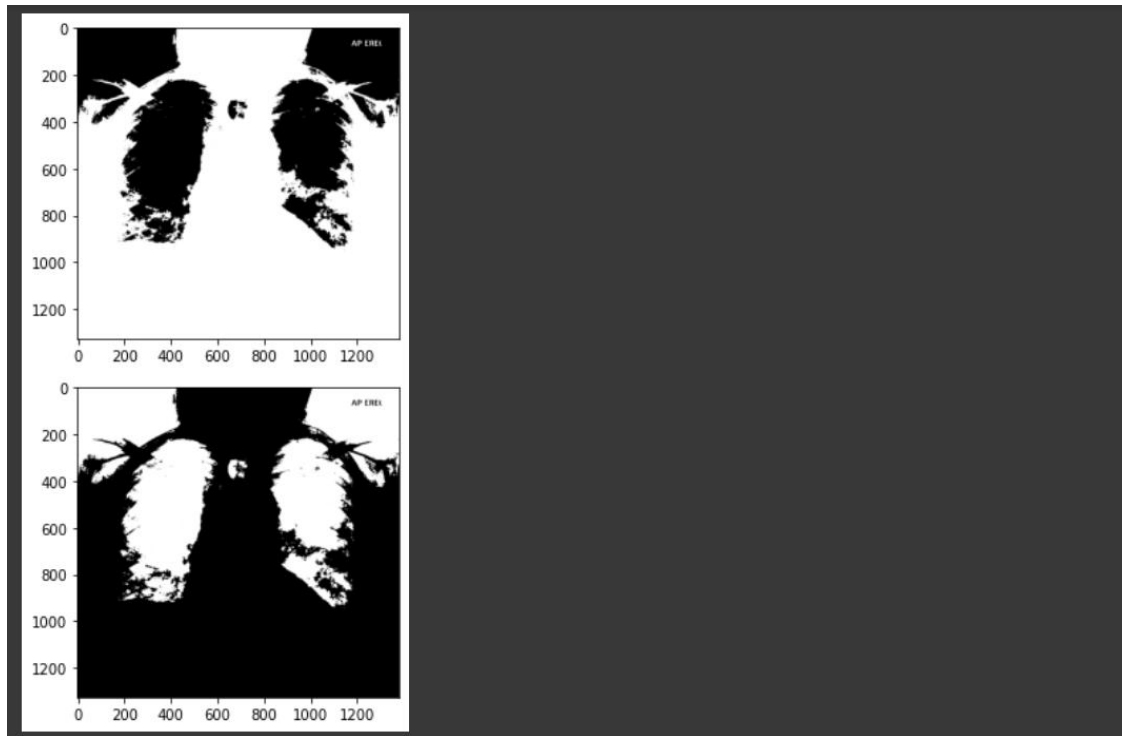
```
fig, ax = plt.subplots()
plt.imshow(binary_mask, cmap='gray')
plt.show()

filename = 'threshold-masked.jpg'

# Using cv2.imwrite() method
# Saving the image
cv2.imwrite(filename, thresh1)

# De-allocate any associated memory usage
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

Output:



10. Pre-process the threshold-masked image to fit the GradCAM model

```
orig = cv2.imread('/content/threshold-masked.jpg')
resized = cv2.resize(orig, (224, 224))
# load the input image from disk (in Keras/TensorFlow format) and
# preprocess it
image = load_img('/content/threshold-masked.jpg', target_size=(224, 224))
image = img_to_array(image)
image = np.expand_dims(image, axis=0)
image = imagenet_utils.preprocess_input(image)
```

11. GradCAM Algorithm:

```
from tensorflow.keras.models import Model
import tensorflow as tf
import numpy as np
import cv2

class GradCAM:
    def __init__(self, model, classIdx, layerName=None):
        # store the model, the class index used to measure the class
        # activation map, and the layer to be used when visualizing
        # the class activation map
        self.model = model
        self.classIdx = classIdx
        self.layerName = layerName
        # if the layer name is None, attempt to automatically find
        # the target output layer
        if self.layerName is None:
            self.layerName = self.find_target_layer()
```

```

def find_target_layer(self):
    # attempt to find the final convolutional layer in the network
    # by looping over the layers of the network in reverse order
    for layer in reversed(self.model.layers):
        # check to see if the layer has a 4D output
        if len(layer.output_shape) == 4:
            return layer.name
    # otherwise, we could not find a 4D layer so the GradCAM
    # algorithm cannot be applied
    raise ValueError("Could not find 4D layer. Cannot apply GradCAM.")

def compute_heatmap(self, image, eps=1e-8):
    # construct our gradient model by supplying (1) the inputs
    # to our pre-trained model, (2) the output of the (presumably)
    # final 4D layer in the network, and (3) the output of the
    # softmax activations from the model
    gradModel = Model(
        inputs=[self.model.inputs],
        outputs=[self.model.get_layer(self.layerName).output, self.model.output])

```

```

    with tf.GradientTape() as tape:
        # cast the image tensor to a float-32 data type, pass the
        # image through the gradient model, and grab the loss
        # associated with the specific class index
        inputs = tf.cast(image, tf.float32)
        (convOutputs, predictions) = gradModel(inputs)

        loss = predictions[:, tf.argmax(predictions[0])]

        # use automatic differentiation to compute the gradients
        grads = tape.gradient(loss, convOutputs)

        # compute the guided gradients
        castConvOutputs = tf.cast(convOutputs > 0, "float32")
        castGrads = tf.cast(grads > 0, "float32")
        guidedGrads = castConvOutputs * castGrads * grads
        # the convolution and guided gradients have a batch dimension
        # (which we don't need) so let's grab the volume itself and
        # discard the batch

```

```

convOutputs = convOutputs[0]
guidedGrads = guidedGrads[0]

    # compute the average of the gradient values, and using them
    # as weights, compute the ponderation of the filters with
    # respect to the weights
    weights = tf.reduce_mean(guidedGrads, axis=(0, 1))
    cam = tf.reduce_sum(tf.multiply(weights, convOutputs), axis=-1)

    # grab the spatial dimensions of the input image and resize
    # the output class activation map to match the input image
    # dimensions
    (w, h) = (image.shape[2], image.shape[1])
    heatmap = cv2.resize(cam.numpy(), (w, h))

    # normalize the heatmap such that all values lie in the range
    # [0, 1], scale the resulting values to the range [0, 255],
    # and then convert to an unsigned 8-bit integer
    numer = heatmap - np.min(heatmap)
    denom = (heatmap.max() - heatmap.min()) + eps
    heatmap = numer / denom
    heatmap = (heatmap * 255).astype("uint8")
    # return the resulting heatmap to the calling function
    return heatmap

```

```
def overlay_heatmap(self, heatmap, image, alpha=0.5,
                    colormap=cv2.COLORMAP_VIRIDIS):
    # apply the supplied color map to the heatmap and then
    # overlay the heatmap on the input image
    heatmap = cv2.applyColorMap(heatmap, colormap)
    output = cv2.addWeighted(image, alpha, heatmap, 1 - alpha, 0)
    # return a 2-tuple of the color mapped heatmap and the output,
    # overlaid image
    return (heatmap, output)
```

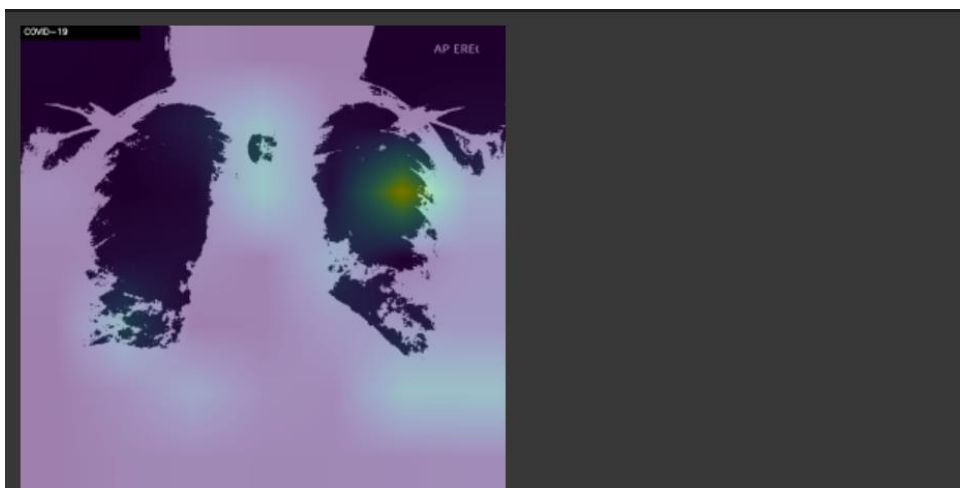
Fitting the model:

```
cam = GradCAM(model, 'conv5_block3_out')
heatmap = cam.compute_heatmap(image)
# resize the resulting heatmap to the original input image dimensions
# and then overlay heatmap on top of the image
heatmap = cv2.resize(heatmap, (orig.shape[1], orig.shape[0]))
(heatmap, output) = cam.overlay_heatmap(heatmap, orig, alpha=0.5)
```

```
from google.colab.patches import cv2_imshow
cv2.rectangle(output, (0, 0), (340, 40), (0, 0, 0), -1)
cv2.putText(output, label, (10, 25), cv2.FONT_HERSHEY_SIMPLEX,
            0.8, (255, 255, 255), 2)
# display the original image and resulting heatmap and output image
# to our screen
#output = np.vstack([orig, heatmap, output])
output = imutils.resize(output, height=300)
cv2_imshow(output)
cv2.waitKey(0)

filename = 'heatmap.jpg'
cv2.imwrite(filename, output)
```

Output:



METRICS FOR EVALUATION

Here, we compare two CNN models: VGG and ResNet50 and evaluate the two models in a comparison fashion

VGG Model:

1. Import pretrained model

```
# example of loading the vgg16 model
from keras.applications.vgg16 import VGG16
# load model
model = VGG16()
# summarize the model
model.summary()
```

Output:

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/weights\_tf\_dim\_ordering\_tf\_kernels.h5
553467904/553467096 [=====] - 4s 0us/step
553476096/553467096 [=====] - 5s 0us/step
Model: "vgg16"
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080

block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000

```
=====
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
```

2. Training the model over 10 epochs

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

/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/adam.py:105: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)

model.compile(optimizer='adam',loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])

history=model.fit(train_ds,validation_data=val_ds,epochs=10)
```

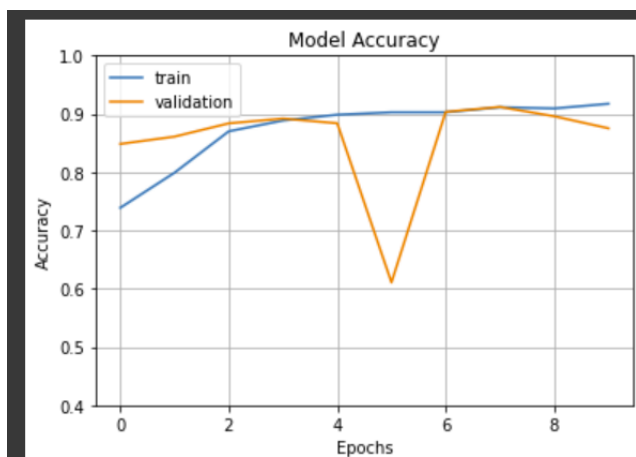
Output:

```
Epoch 1/10
/usr/local/lib/python3.7/dist-packages/tensorflow/python/util/dispatch.py:1082: UserWarning: ``sparse_categorical_crossentropy`` received ``
  return dispatch_target(*args, **kwargs)
889/889 [=====] - 477s 515ms/step - loss: 3.7312 - accuracy: 0.7387 - val_loss: 0.3787 - val_accuracy: 0.8484
Epoch 2/10
889/889 [=====] - 460s 516ms/step - loss: 0.5359 - accuracy: 0.7990 - val_loss: 0.3601 - val_accuracy: 0.8609
Epoch 3/10
889/889 [=====] - 460s 516ms/step - loss: 0.3315 - accuracy: 0.8701 - val_loss: 0.3173 - val_accuracy: 0.8837
Epoch 4/10
889/889 [=====] - 461s 517ms/step - loss: 0.2881 - accuracy: 0.8886 - val_loss: 0.2923 - val_accuracy: 0.8919
Epoch 5/10
889/889 [=====] - 460s 516ms/step - loss: 0.2643 - accuracy: 0.8985 - val_loss: 0.3222 - val_accuracy: 0.8839
Epoch 6/10
889/889 [=====] - 460s 516ms/step - loss: 0.2595 - accuracy: 0.9027 - val_loss: 1.0060 - val_accuracy: 0.6106
Epoch 7/10
889/889 [=====] - 461s 517ms/step - loss: 0.2550 - accuracy: 0.9027 - val_loss: 0.2614 - val_accuracy: 0.9034
Epoch 8/10
889/889 [=====] - 459s 514ms/step - loss: 0.2338 - accuracy: 0.9114 - val_loss: 0.2436 - val_accuracy: 0.9119
Epoch 9/10
889/889 [=====] - 457s 513ms/step - loss: 0.2418 - accuracy: 0.9093 - val_loss: 0.3033 - val_accuracy: 0.8958
Epoch 10/10
889/889 [=====] - 457s 512ms/step - loss: 0.2203 - accuracy: 0.9173 - val_loss: 0.3398 - val_accuracy: 0.8753
```

3. Model Evaluation:

```
fig1 = plt.gcf()
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.axis(ymin=0.4,ymax=1)
plt.grid()
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['train', 'validation'])
plt.show()
```

Output:



4. Model Inference

```
import cv2
image=cv2.imread('/content/Dataset/COVID-19/230.jpg')
image_resized= cv2.resize(image, (img_height,img_width))
image=np.expand_dims(image_resized,axis=0)

pred=model.predict(image)

output_class=class_names[np.argmax(pred)]
print("The predicted class is", output_class)
```

Output:

```
The predicted class is COVID-19
```

ResNet50 Model:

1. Import pretrained model

```
from tensorflow.keras.applications import ResNet50
# load model
model = ResNet50()
# summarize the model
model.summary()
```

Output:

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50\_weights\_tf\_dim\_ordering
102973440/102967424 [=====] - 0s 0us/step
102981632/102967424 [=====] - 0s 0us/step
Model: "resnet50"
```

Layer (type)	Output Shape	Param #	Connected to
=====			
input_1 (InputLayer)	(None, 224, 224, 3)	0	[]
conv1_pad (ZeroPadding2D)	(None, 230, 230, 3)	0	['input_1[0][0]']
conv1_conv (Conv2D)	(None, 112, 112, 64)	9472	['conv1_pad[0][0]']
conv1_bn (BatchNormalization)	(None, 112, 112, 64)	256	['conv1_conv[0][0]']
conv1_relu (Activation)	(None, 112, 112, 64)	0	['conv1_bn[0][0]']
pool1_pad (ZeroPadding2D)	(None, 114, 114, 64)	0	['conv1_relu[0][0]']
pool1_pool (MaxPooling2D)	(None, 56, 56, 64)	0	['pool1_pad[0][0]']

conv4_block2_3_bn (BatchNormalization)	(None, 14, 14, 1024)	4096	['conv4_block2_3_conv[0][0]']
conv4_block2_add (Add)	(None, 14, 14, 1024)	0	['conv4_block1_out[0][0]', 'conv4_block2_3_bn[0][0]']
conv4_block2_out (Activation)	(None, 14, 14, 1024)	0	['conv4_block2_add[0][0]']
conv4_block3_1_conv (Conv2D)	(None, 14, 14, 256)	262400	['conv4_block2_out[0][0]']
conv4_block3_1_bn (BatchNormalization)	(None, 14, 14, 256)	1024	['conv4_block3_1_conv[0][0]']
conv4_block3_1_relu (Activation)	(None, 14, 14, 256)	0	['conv4_block3_1_bn[0][0]']
conv4_block3_2_conv (Conv2D)	(None, 14, 14, 256)	590080	['conv4_block3_1_relu[0][0]']
conv4_block3_2_bn (BatchNormalization)	(None, 14, 14, 256)	1024	['conv4_block3_2_conv[0][0]']
conv4_block3_2_relu (Activation)	(None, 14, 14, 256)	0	['conv4_block3_2_bn[0][0]']
conv4_block3_3_conv (Conv2D)	(None, 14, 14, 1024)	263168	['conv4_block3_2_relu[0][0]']
conv4_block3_3_bn (BatchNormalization)	(None, 14, 14, 1024)	4096	['conv4_block3_3_conv[0][0]']
conv4_block3_add (Add)	(None, 14, 14, 1024)	0	['conv4_block2_out[0][0]', 'conv4_block3_3_bn[0][0]']
conv4_block3_out (Activation)	(None, 14, 14, 1024)	0	['conv4_block3_add[0][0]']
conv4_block4_1_conv (Conv2D)	(None, 14, 14, 256)	262400	['conv4_block3_out[0][0]']
conv4_block4_1_bn (BatchNormalization)	(None, 14, 14, 256)	1024	['conv4_block4_1_conv[0][0]']
conv4_block4_1_relu (Activation)	(None, 14, 14, 256)	0	['conv4_block4_1_bn[0][0]']
conv4_block4_2_conv (Conv2D)	(None, 14, 14, 256)	590080	['conv4_block4_1_relu[0][0]']
conv4_block4_2_bn (BatchNormalization)	(None, 14, 14, 256)	1024	['conv4_block4_2_conv[0][0]']
conv4_block4_2_relu (Activation)	(None, 14, 14, 256)	0	['conv4_block4_2_bn[0][0]']
conv4_block4_3_conv (Conv2D)	(None, 14, 14, 1024)	263168	['conv4_block4_2_relu[0][0]']
conv5_block3_add (Add)	(None, 7, 7, 2048)	0	['conv5_block2_out[0][0]', 'conv5_block3_3_bn[0][0]']
conv5_block3_out (Activation)	(None, 7, 7, 2048)	0	['conv5_block3_add[0][0]']
avg_pool (GlobalAveragePooling2D)	(None, 2048)	0	['conv5_block3_out[0][0]']
predictions (Dense)	(None, 1000)	2049000	['avg_pool[0][0]']

=====

Total params: 25,636,712
Trainable params: 25,583,592
Non-trainable params: 53,120

2. Training the model over 10 epochs

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

/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/adam.py:105: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead
  super(Adam, self).__init__(name, **kwargs)

model.compile(optimizer='adam',loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])

history=model.fit(train_ds,validation_data=val_ds,epochs=10)
```

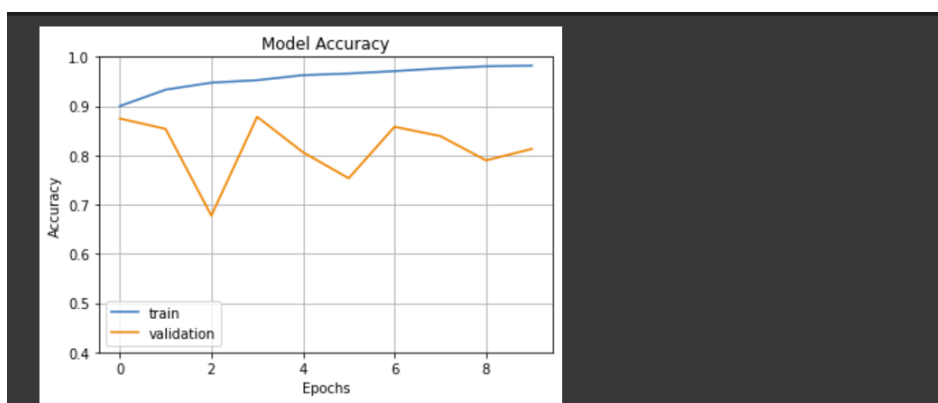
Output:

```
Epoch 1/10
/usr/local/lib/python3.7/dist-packages/tensorflow/python/util/dispatch.py:1082: UserWarning: "`sparse_categorical_crossentropy` received
  return dispatch_target(*args, **kwargs)
889/889 [=====] - 358s 380ms/step - loss: 0.2935 - accuracy: 0.9000 - val_loss: 0.3311 - val_accuracy: 0.8747
Epoch 2/10
889/889 [=====] - 343s 384ms/step - loss: 0.1843 - accuracy: 0.9333 - val_loss: 0.4569 - val_accuracy: 0.8537
Epoch 3/10
889/889 [=====] - 343s 384ms/step - loss: 0.1460 - accuracy: 0.9475 - val_loss: 1.2347 - val_accuracy: 0.6775
Epoch 4/10
889/889 [=====] - 341s 382ms/step - loss: 0.1330 - accuracy: 0.9526 - val_loss: 0.3396 - val_accuracy: 0.8782
Epoch 5/10
889/889 [=====] - 341s 382ms/step - loss: 0.1026 - accuracy: 0.9628 - val_loss: 0.6248 - val_accuracy: 0.8063
Epoch 6/10
889/889 [=====] - 343s 385ms/step - loss: 0.0930 - accuracy: 0.9662 - val_loss: 1.0011 - val_accuracy: 0.7534
Epoch 7/10
889/889 [=====] - 341s 382ms/step - loss: 0.0810 - accuracy: 0.9710 - val_loss: 0.4492 - val_accuracy: 0.8580
Epoch 8/10
889/889 [=====] - 341s 383ms/step - loss: 0.0668 - accuracy: 0.9766 - val_loss: 0.7756 - val_accuracy: 0.8391
Epoch 9/10
889/889 [=====] - 341s 382ms/step - loss: 0.0559 - accuracy: 0.9808 - val_loss: 0.9712 - val_accuracy: 0.7897
Epoch 10/10
889/889 [=====] - 341s 382ms/step - loss: 0.0509 - accuracy: 0.9823 - val_loss: 0.8204 - val_accuracy: 0.8131
```

3. Model Evaluation:

```
fig1 = plt.gcf()
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.axis(ymin=0.4,ymax=1)
plt.grid()
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['train', 'validation'])
plt.show()
```

Output:



4. Model Inference

```
import cv2
image=cv2.imread('/content/Dataset/COVID-19/063.jpeg')
image_resized= cv2.resize(image, (img_height,img_width))
image=np.expand_dims(image_resized,axis=0)

pred=model.predict(image)

output_class=class_names[np.argmax(pred)]
print("The predicted class is", output_class)
```

Output:

```
The predicted class is COVID-19
```

From the above model accuracy graphs plotted for each model, we infer that the accuracy of ResNet50 CNN model is better and loss function of it is lower. Hence ResNet50 has been chosen as the CNN model for our project

Other evaluation Metrics:

```
from keras.preprocessing.image import ImageDataGenerator
import numpy
test_generator = ImageDataGenerator()
test_data_generator = test_generator.flow_from_directory(
    '/content/Dataset', # Put your path here
    target_size=(img_width, img_height),
    batch_size=32,
    shuffle=False)
test_steps_per_epoch = numpy.math.ceil(test_data_generator.samples / test_data_generator.batch_size)

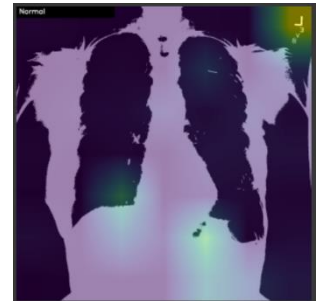
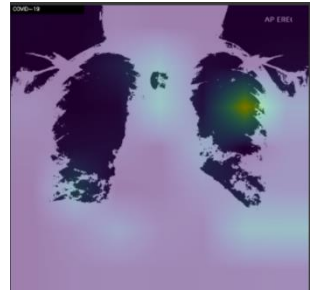
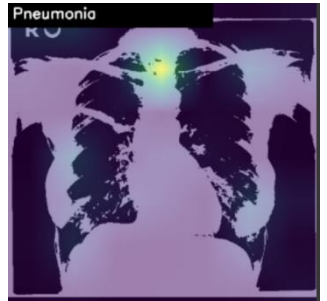
predictions = model.predict_generator(test_data_generator, steps=test_steps_per_epoch)
# Get most likely class
predicted_classes = numpy.argmax(predictions, axis=1)

true_classes = test_data_generator.classes
class_labels = list(test_data_generator.class_indices.keys())

import sklearn.metrics as metrics
report = metrics.classification_report(true_classes, predicted_classes, target_names=class_labels)
print(report)
```

	precision	recall	f1-score	support
COVID-19	0.95	0.39	0.55	6463
Normal	0.74	0.98	0.84	17988
Pneumonia	0.95	0.78	0.86	11073
accuracy			0.81	35524
macro avg	0.88	0.72	0.75	35524
weighted avg	0.84	0.81	0.79	35524

ALL POSSIBLE TEST CASES

Image	Classification result	Final Output
Normal Chest X-Ray	<pre>img = cv2.imread('/content/Dataset/Normal/00000621_011.png') image_resized= cv2.resize(img, (img_height,img_width)) img=np.expand_dims(image_resized,axis=0) preds = model.predict(img) label=class_names[np.argmax(preds)] print("The predicted class is", label) The predicted class is Normal</pre>	
Covid-19 affected Chest X-Ray	<pre>img = cv2.imread('/content/Dataset/COVID-19/063.jpeg') image_resized= cv2.resize(img, (img_height,img_width)) img=np.expand_dims(image_resized,axis=0) preds = model.predict(img) label=class_names[np.argmax(preds)] print("The predicted class is", label) The predicted class is COVID-19</pre>	
Pneumonia affected Chest X-Ray	<pre>img = cv2.imread('/content/Dataset/Pneumonia/212.png') image_resized= cv2.resize(img, (img_height,img_width)) img=np.expand_dims(image_resized,axis=0) preds = model.predict(img) label=class_names[np.argmax(preds)] print("The predicted class is", label) The predicted class is Pneumonia</pre>	

VARIOUS INPUT/OUTPUT

Covid-19

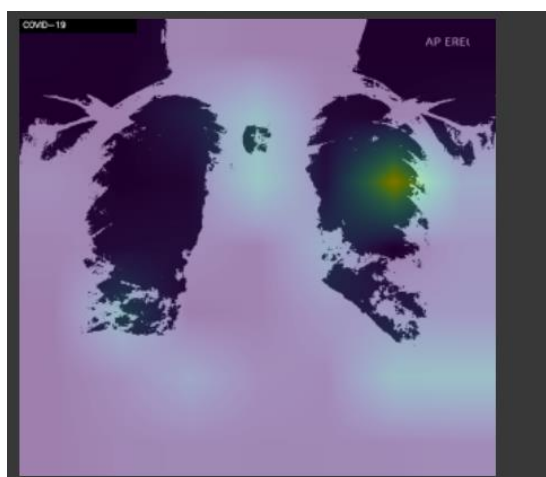
```
img = cv2.imread('/content/Dataset/COVID-19/063.jpeg')
image_resized= cv2.resize(img, (img_height,img_width))
img=np.expand_dims(image_resized,axis=0)
preds = model.predict(img)
label=class_names[np.argmax(preds)]
print("The predicted class is", label)
```

The predicted class is COVID-19

Binary Masked image:



Final Output:

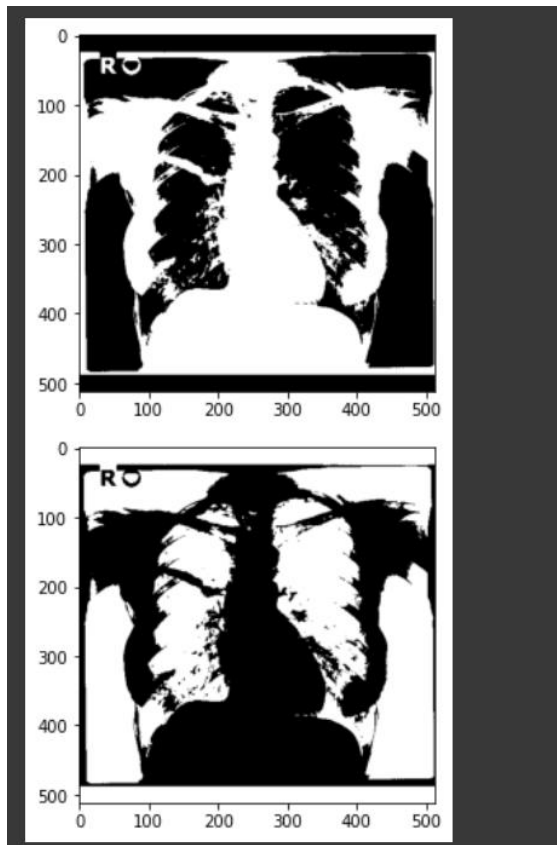


Pneumonia:

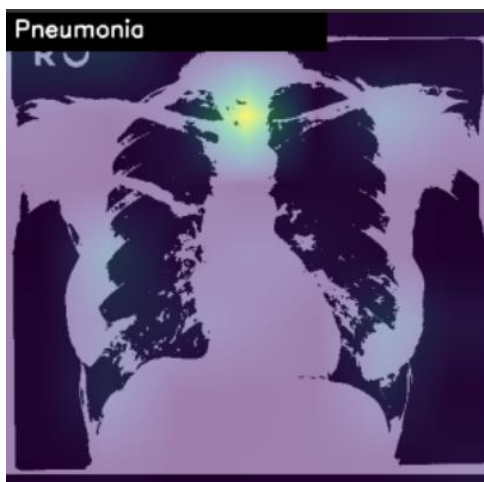
```
img = cv2.imread('/content/Dataset/Pneumonia/212.png')
image_resized= cv2.resize(img, (img_height,img_width))
img=np.expand_dims(image_resized,axis=0)
preds = model.predict(img)
label=class_names[np.argmax(preds)]
print("The predicted class is", label)
```

The predicted class is Pneumonia

Binary Masked image:



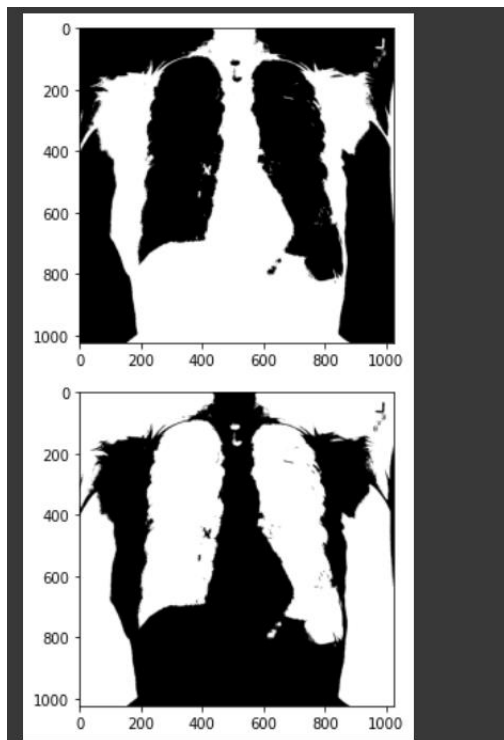
Final Output:



Normal:

```
img = cv2.imread('/content/Dataset/Normal/00000621_011.png')  
  
image_resized= cv2.resize(img, (img_height,img_width))  
  
img=np.expand_dims(image_resized,axis=0)  
preds = model.predict(img)  
label=class_names[np.argmax(preds)]  
print("The predicted class is", label)  
  
The predicted class is Normal
```

Binary Masked image



Final Output:



REFERENCES

1. Bassi, P. R., & Attux, R. (2022). A deep convolutional neural network for COVID-19 detection using chest X-rays. *Research on Biomedical Engineering*, 38(1), 139-148.
2. Hasan, M. J., Alom, M. S., & Ali, M. S. (2021, February). Deep learning based detection and segmentation of COVID-19 & pneumonia on chest X-ray image. In *2021 International Conference on Information and Communication Technology for Sustainable Development (ICICT4SD)* (pp. 210-214). IEEE.