COMPUTER VISION

Pneumonia Detection Challenge - Capstone Project

An Interim Report

Submitted by:

DineshKumarr AN

Gayatri M

Mohit Shrivastava

Somnath Gadekar

Vishal Jani

Vivek Dehulia

Mentor: Aniket Chhabra

In fulfilment of the requirements for the award of

Post Graduate Program in Artificial Intelligence & Machine Learning

From:

Great Learning & **TEXAS McCombs** The University of Texas at Austin

November 24

TABLE OF CONTENTS

Learning for Life

1. Introduction

Pneumonia is a disease in one or the two lungs. Microscopic organisms, infections, and growths cause it. The contamination causes aggravation in the air sacs in your lungs, which are called alveoli.

Pneumonia represents more than 15% of all passings of youngsters under 5 years of age universally. In 2015, 920,000 kids younger than 5 passed on from the sickness.

While normal, precisely diagnosing pneumonia is a difficult task. It requires a survey of a chest radiograph (CXR) by exceptionally prepared trained professionals and affirmation through clinical history, important bodily functions and lab tests.

Pneumonia generally appears as an area or area of expanded obscurity on CXR.

However, the diagnosis of pneumonia on CXR is complicated as a result of various different circumstances in the lungs like liquid over-burden (pneumonic oedema), dying, volume misfortune (atelectasis or breakdown), cellular breakdown in the lungs, or post-radiation or surgical changes.

Outside of the lungs, fluid in the pleural space (pleural emanation) additionally shows up as increased opacity on CXR. When accessible, examination of CXRs of the patient taken at various times focuses and connection with clinical side effects and history are useful in making the diagnosis.

CXRs are the most generally performed symptomatic imaging study. Various factors, such as the patient's situating and profundity of motivation, can change the presence of the CXR, complicating interpretation further.

2. Summary of Problem Statement

Problem Statement:

Pneumonia detection using chest radiographs (X-rays) is a critical task in medical imaging, as early detection and diagnosis significantly improve patient outcomes. The task in this project involves designing a deep learning (DL) based algorithm to automatically detect pneumonia in medical images by identifying lung opacities on chest radiographs. The goal is to identify regions of the lungs that exhibit signs of pneumonia, which typically appear as white patches or areas of opacification in the image.

Problem Statement, Data and Finding

PROBLEM STATEMENT:

- The goal is to build an algorithm to detect a visual signal for pneumonia in medical images. Specifically, our algorithm needs to automatically locate lung opacities on chest radiographs which will help us detect Inflammation of the lungs in Pneumonia detection.
- In the dataset, some features are labelled as "Not Normal No Lung Opacity." This extra third class indicates that while pneumonia was determined not to be present, there was nonetheless some type of abnormality on the image and oftentimes this finding may mimic the appearance of true pneumonia.

Original link to the dataset: https://www.kaggle.com/c/rsna-pneumonia-detection-challenge/data.

Acknowledgements: https://www.kaggle.com/c/rsna-pneumonia-detection-challenge/overview/acknowledgements.

Dataset Description:

1. File descriptions:

stage_2_train_labels.csv

In **train labels,** the dataset is given the patient ID and the window (x min, y min, width, and height) containing evidence of pneumonia.

stage_2_train_images.zip

The directory containing 26000 training set raw image (DICOM) files
stage_2_test_images.zip

The directory containing 3000 testing set raw image (DICOM) files

stage_2_sample_submission.csv

A sample submission file in the correct format. Contains the patients' ID for the test set. Note that the sample submission contains one box per image, but there is no limit to the number of bounding boxes that can be assigned to a given image.

stage_2_detailed_class_info.csv

This file provides detailed information about the type of positive or negative class for each image.

2. Data fields:

1. **Patient ID** - A patient ID. Each patient ID corresponds to a unique image.
2. **x** - the upper-left x coordinate of the bounding box.
3. **y** - the upper-left y coordinate of the bounding box.
4. **width** - the width of the bounding box.
5. **height** - the height of the bounding box.
6. **Target** - the binary Target, indicating whether this sample has evidence of pneumonia.

The dataset for this challenge consists of medical images stored in the **DICOM format** (.dcm), which includes both pixel data and metadata. The dataset consists of annotated chest radiographs, divided into categories:

- **Not Normal No Lung Opacity**: Indicates abnormalities mimicking pneumonia.
- **Normal**: Images without abnormalities.
- **Lung Opacity (Pneumonia)**: Confirmed pneumonia cases.

- The dataset is divided into **training** and **testing** sets.
- Each image has an associated **annotation file** that includes bounding boxes marking areas of interest, specifically lung opacities.

Key features of the dataset:

- Images are stored as DICOM files.
- The images are labelled into three categories: **"Normal" "Lung Opacity," and "No Lung Opacity / Not Normal."**
- The bounding boxes indicate where opacities, which could be signs of pneumonia, are located within the radiographs.

Initial Findings

The data includes labelled abnormalities, creating a multi-class classification problem. Annotations guide the bounding box localization task, making it both a classification and object detection challenge.

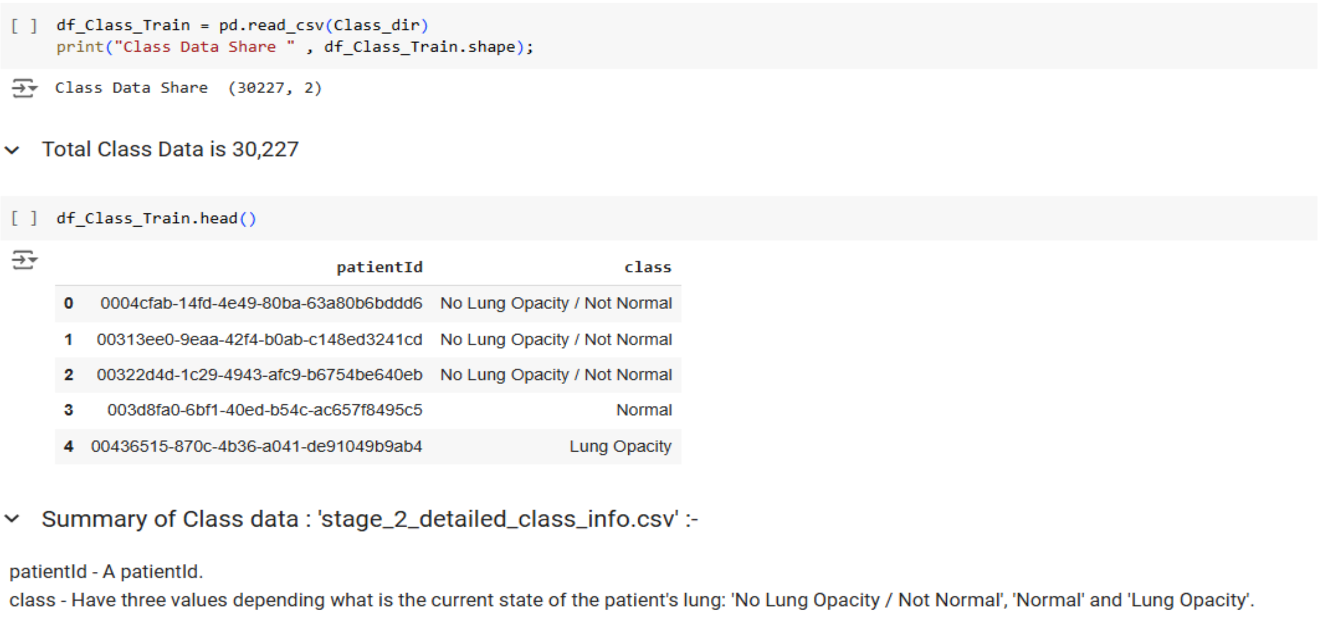
EDA – Exploratory Data Analysis

Preprocessing Techniques

- Imported essential libraries (e.g., Pandas, NumPy, Seaborn, Matplotlib, and Pydicom for handling DICOM images).
- Mapped training/testing images to respective annotations and classes.

Exploratory Data Analysis (EDA)

DATASET INFORMATION:



```
[ ] df_Class_Train = pd.read_csv(Class_dir)
    print("Class Data Share " , df_Class_Train.shape);
```

```
Class Data Share  (30227, 2)
```

Total Class Data is 30,227

```
[ ] df_Class_Train.head()
```

	patientId	class
0	0004cfab-14fd-4e49-80ba-63a80b6bddd6	No Lung Opacity / Not Normal
1	00313ee0-9eaa-42f4-b0ab-c148ed3241cd	No Lung Opacity / Not Normal
2	00322d4d-1c29-4943-afc9-b6754be640eb	No Lung Opacity / Not Normal
3	003d8fa0-6bf1-40ed-b54c-ac657f8495c5	Normal
4	00436515-870c-4b36-a041-de91049b9ab4	Lung Opacity

Summary of Class data : 'stage_2_detailed_class_info.csv' :-

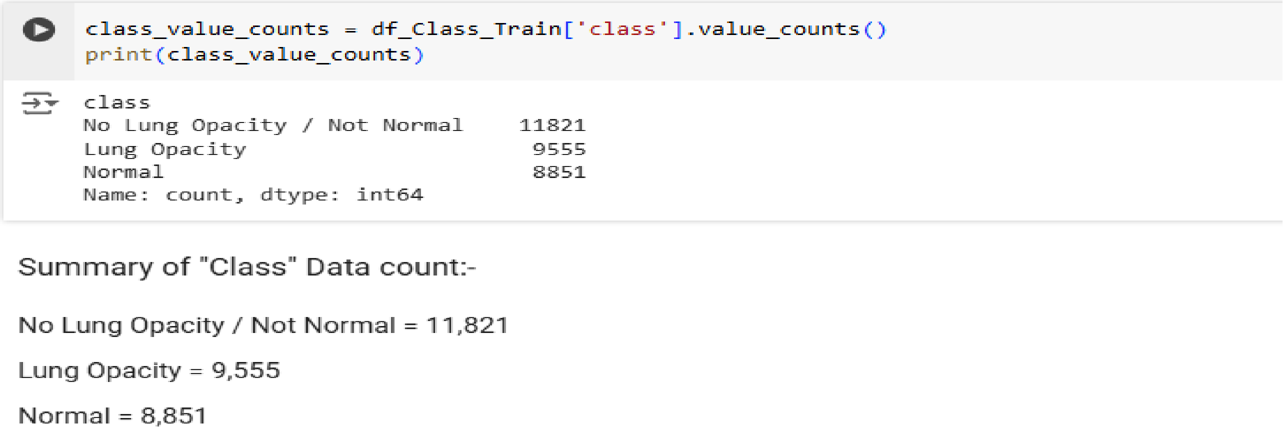
patientId - A patientId.
class - Have three values depending what is the current state of the patient's lung: 'No Lung Opacity / Not Normal', 'Normal' and 'Lung Opacity'.

The total class data is 30,227.

patientId - A patientId.

Class- Having three values depending upon what is the current state of the patient’s lung.

: 'No Lung Opacity / Not Normal', 'Normal' and 'Lung Opacity'.



```
class_value_counts = df_Class_Train['class'].value_counts()
print(class_value_counts)
```

```
class
No Lung Opacity / Not Normal    11821
Lung Opacity                     9555
Normal                           8851
Name: count, dtype: int64
```

Summary of "Class" Data count:-

No Lung Opacity / Not Normal = 11,821

Lung Opacity = 9,555

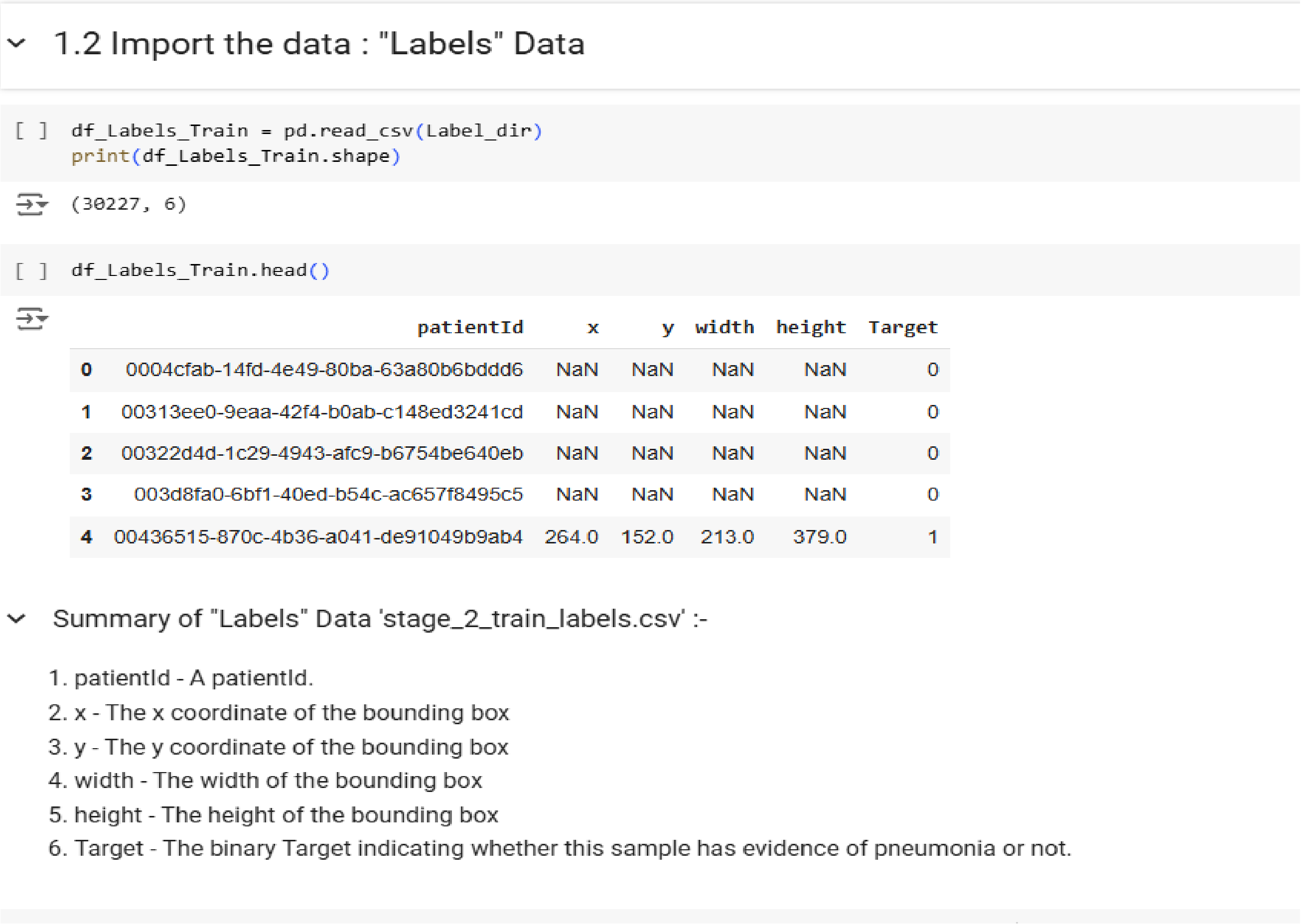
Normal = 8,851

There are the following findings in class Data counts-

No Lung Opacity / Not Normal = 11,821

Lung Opacity = 9,555

Normal = 8,851



1.2 Import the data : "Labels" Data

```
[ ] df_Labels_Train = pd.read_csv(Label_dir)
    print(df_Labels_Train.shape)
```

```
(30227, 6)
```

```
[ ] df_Labels_Train.head()
```

	patientId	x	y	width	height	Target
0	0004cfab-14fd-4e49-80ba-63a80b6bddd6	NaN	NaN	NaN	NaN	0
1	00313ee0-9eaa-42f4-b0ab-c148ed3241cd	NaN	NaN	NaN	NaN	0
2	00322d4d-1c29-4943-afc9-b6754be640eb	NaN	NaN	NaN	NaN	0
3	003d8fa0-6bf1-40ed-b54c-ac657f8495c5	NaN	NaN	NaN	NaN	0
4	00436515-870c-4b36-a041-de91049b9ab4	264.0	152.0	213.0	379.0	1

Summary of "Labels" Data 'stage_2_train_labels.csv' :-

1. patientId - A patientId.
2. x - The x coordinate of the bounding box
3. y - The y coordinate of the bounding box
4. width - The width of the bounding box
5. height - The height of the bounding box
6. Target - The binary Target indicating whether this sample has evidence of pneumonia or not.

There are the following findings in Labels Data-

patientId - A patientId.

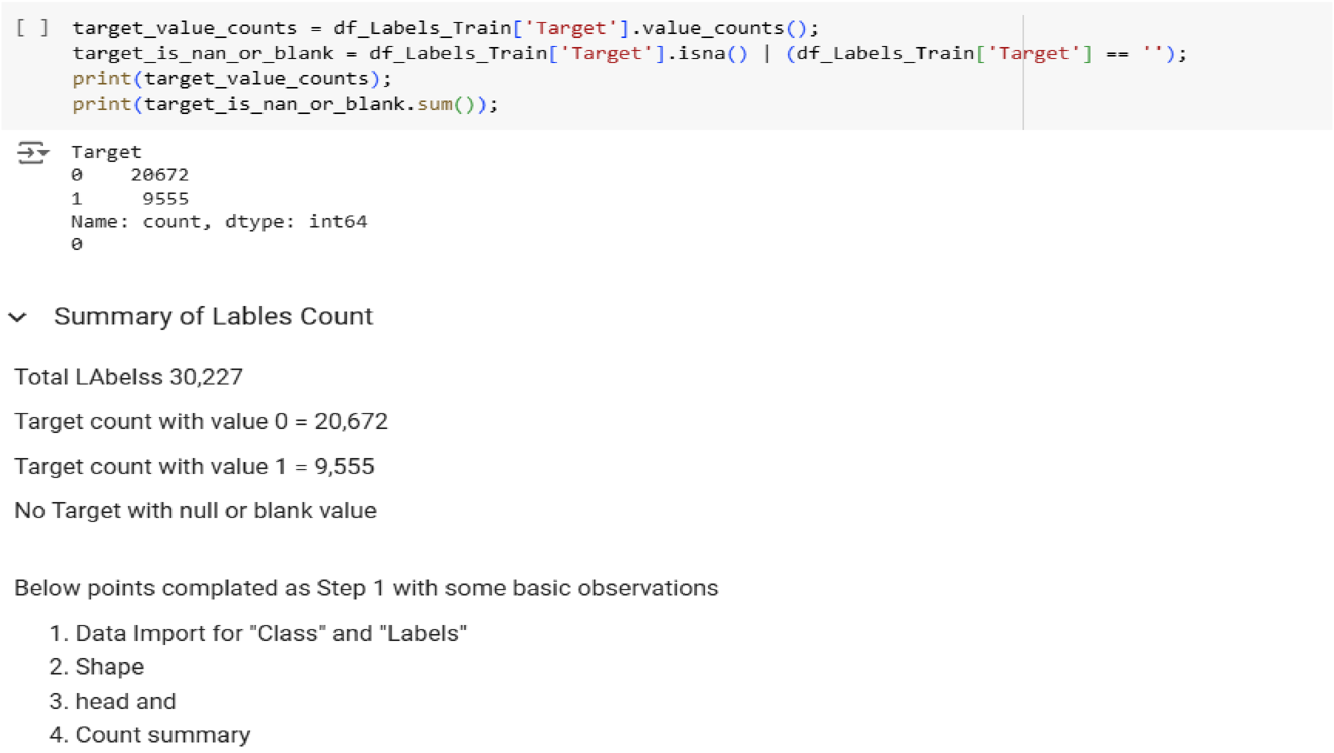
x - The x coordinate of the bounding box

y - The y coordinate of the bounding box

width - The width of the bounding box

height - The height of the bounding box

Target - The binary Target indicates whether this sample has evidence of pneumonia or not.



```
[ ] target_value_counts = df_Labels_Train['Target'].value_counts();
    target_is_nan_or_blank = df_Labels_Train['Target'].isna() | (df_Labels_Train['Target'] == '');
    print(target_value_counts);
    print(target_is_nan_or_blank.sum());
```

```
Target
0    20672
1     9555
Name: count, dtype: int64
0
```

Summary of Lables Count

Total LAbelss 30,227

Target count with value 0 = 20,672

Target count with value 1 = 9,555

No Target with null or blank value

Below points complated as Step 1 with some basic observations

1. Data Import for "Class" and "Labels"
2. Shape
3. head and
4. Count summary

There are the following findings in Labels count-

Total Labels are 30,227

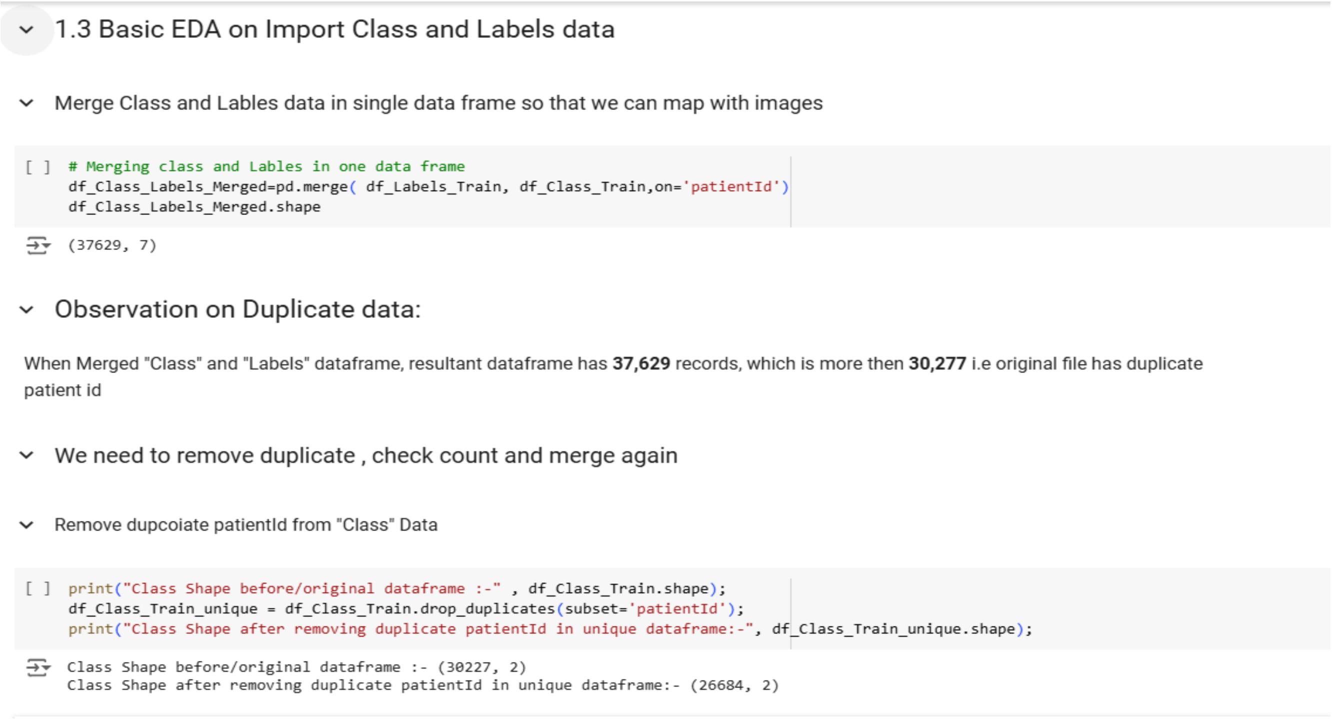
Target count with value 0 = 20,672

Target count with value 1 = 9,555

No Target with a null or blank value

Below are points completed as Step 1 with some basic observations

1. Data Import for "Class" and "Labels"
2. Shape
3. head and
4. Count summary



1.3 Basic EDA on Import Class and Labels data

Merge Class and Lables data in single data frame so that we can map with images

```
[ ] # Merging class and Lables in one data frame
    df_Class_Labels_Merged=pd.merge( df_Labels_Train, df_Class_Train,on='patientId')
    df_Class_Labels_Merged.shape
```

```
(37629, 7)
```

Observation on Duplicate data:

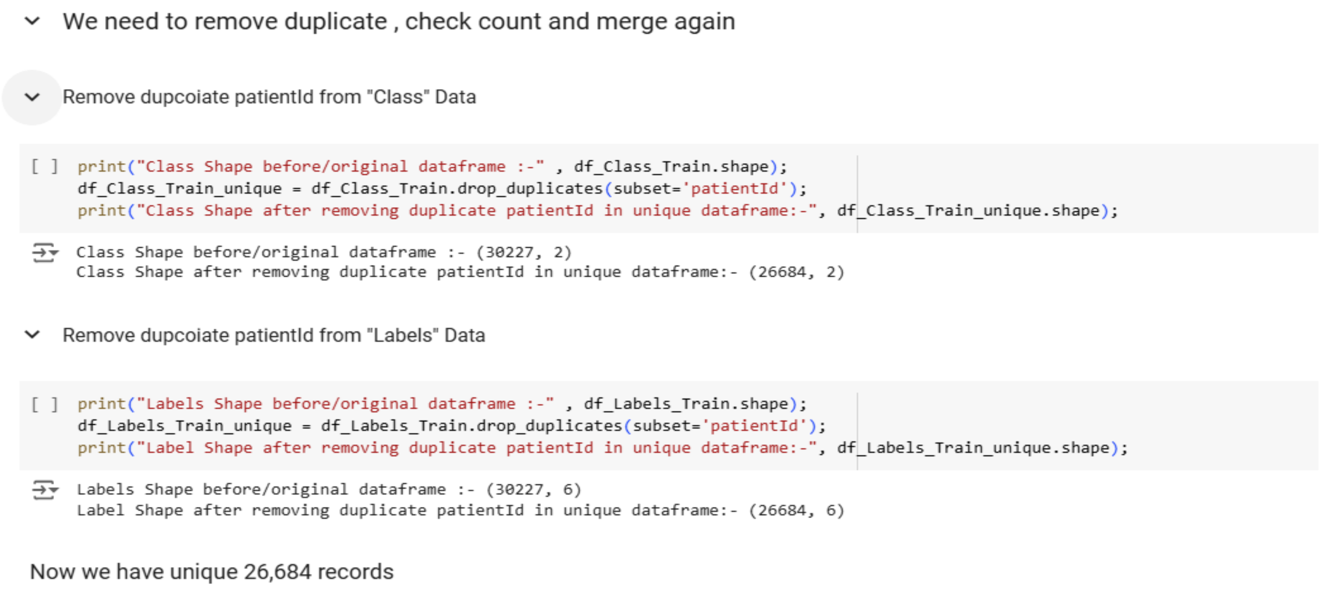
When Merged "Class" and "Labels" dataframe, resultant dataframe has **37,629** records, which is more then **30,277** i.e original file has duplicate patient id

We need to remove duplicate , check count and merge again

Remove dupcoiate patientId from "Class" Data

```
[ ] print("Class Shape before/original dataframe :-" , df_Class_Train.shape);
    df_Class_Train_unique = df_Class_Train.drop_duplicates(subset='patientId');
    print("Class Shape after removing duplicate patientId in unique dataframe:-", df_Class_Train_unique.shape);
```

```
Class Shape before/original dataframe :- (30227, 2)
Class Shape after removing duplicate patientId in unique dataframe:- (26684, 2)
```



We need to remove duplicate , check count and merge again

Remove dupcoiate patientId from "Class" Data

```
[ ] print("Class Shape before/original dataframe :-" , df_Class_Train.shape);
    df_Class_Train_unique = df_Class_Train.drop_duplicates(subset='patientId');
    print("Class Shape after removing duplicate patientId in unique dataframe:-", df_Class_Train_unique.shape);
```

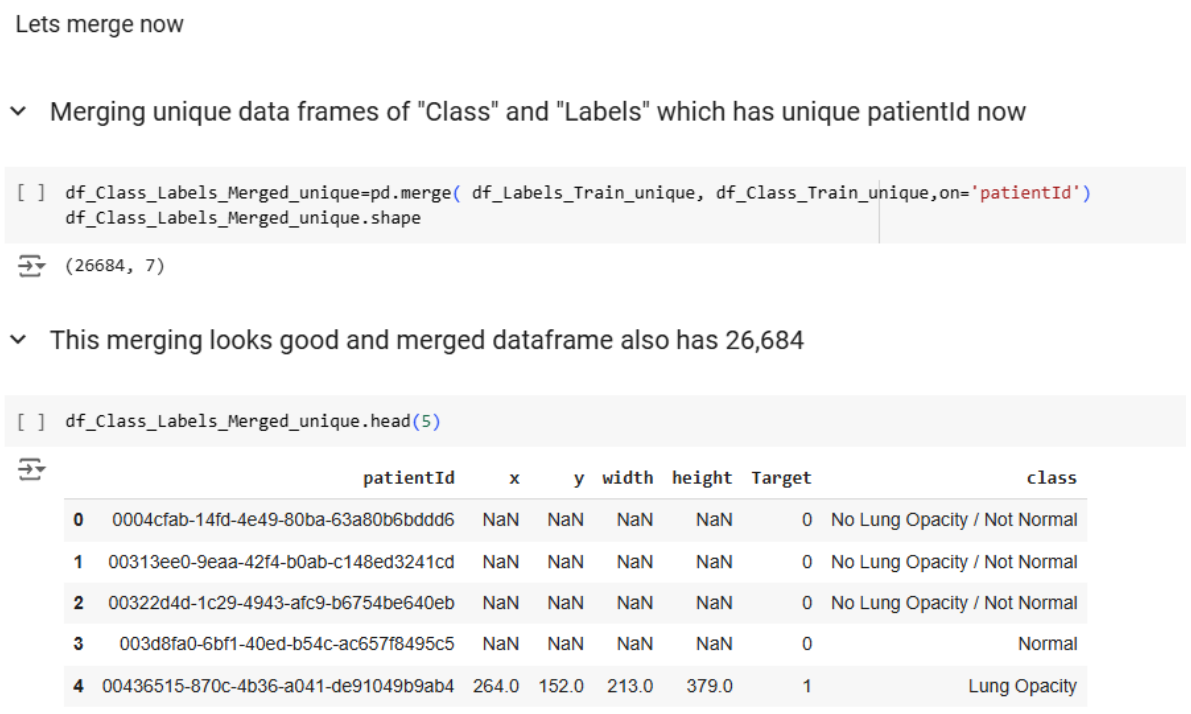
```
Class Shape before/original dataframe :- (30227, 2)
Class Shape after removing duplicate patientId in unique dataframe:- (26684, 2)
```

Remove dupcoiate patientId from "Labels" Data

```
[ ] print("Labels Shape before/original dataframe :-" , df_Labels_Train.shape);
    df_Labels_Train_unique = df_Labels_Train.drop_duplicates(subset='patientId');
    print("Label Shape after removing duplicate patientId in unique dataframe:-", df_Labels_Train_unique.shape);
```

```
Labels Shape before/original dataframe :- (30227, 6)
Label Shape after removing duplicate patientId in unique dataframe:- (26684, 6)
```

Now we have unique 26,684 records



Lets merge now

Merging unique data frames of "Class" and "Labels" which has unique patientId now

```
[ ] df_Class_Labels_Merged_unique=pd.merge( df_Labels_Train_unique, df_Class_Train_unique,on='patientId')
    df_Class_Labels_Merged_unique.shape
```

```
(26684, 7)
```

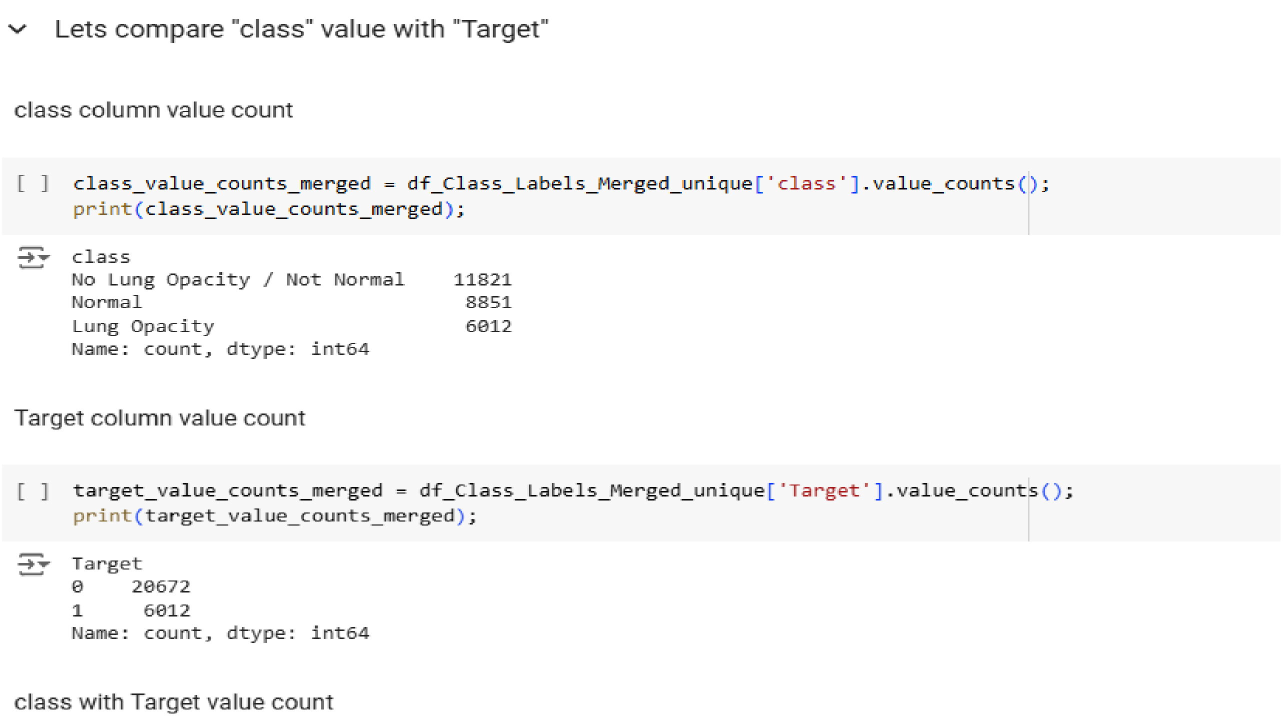
This merging looks good and merged dataframe also has 26,684

```
[ ] df_Class_Labels_Merged_unique.head(5)
```

	patientId	x	y	width	height	Target	class
0	0004cfab-14fd-4e49-80ba-63a80b6bddd6	NaN	NaN	NaN	NaN	0	No Lung Opacity / Not Normal
1	00313ee0-9eaa-42f4-b0ab-c148ed3241cd	NaN	NaN	NaN	NaN	0	No Lung Opacity / Not Normal
2	00322d4d-1c29-4943-afc9-b6754be640eb	NaN	NaN	NaN	NaN	0	No Lung Opacity / Not Normal
3	003d8fa0-6bf1-40ed-b54c-ac657f8495c5	NaN	NaN	NaN	NaN	0	Normal
4	00436515-870c-4b36-a041-de91049b9ab4	264.0	152.0	213.0	379.0	1	Lung Opacity

- After merging the class and Labels data frame,
- The resultant dataframe is 37,629 records which is more than 30,277 records.
- i.e. original file has duplicate records.
- Unique records are only 26,684.

This merging looks good and the merged dataframe also has 26,684.



Lets compare "class" value with "Target"

class column value count

```
[ ]  class_value_counts_merged = df_Class_Labels_Merged_unique['class'].value_counts();
     print(class_value_counts_merged);
```

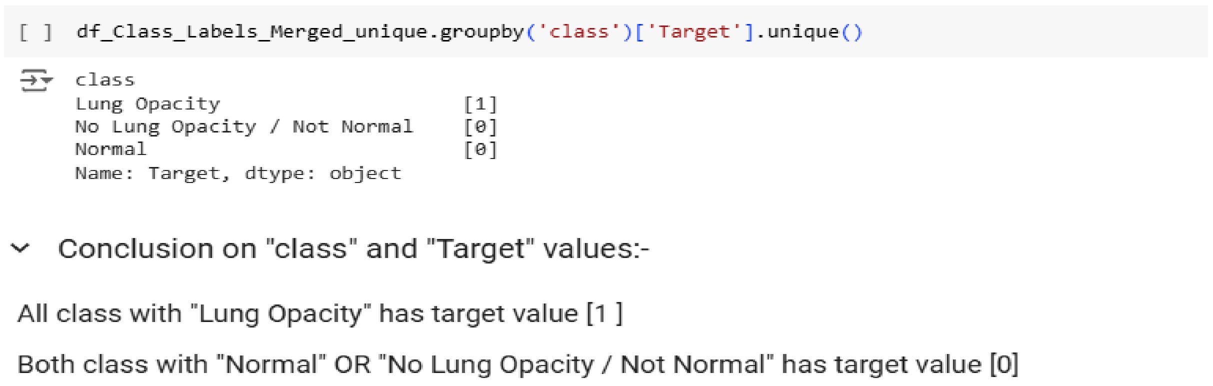
```
class
No Lung Opacity / Not Normal    11821
Normal                           8851
Lung Opacity                     6012
Name: count, dtype: int64
```

Target column value count

```
[ ]  target_value_counts_merged = df_Class_Labels_Merged_unique['Target'].value_counts();
     print(target_value_counts_merged);
```

```
Target
0    20672
1     6012
Name: count, dtype: int64
```

class with Target value count



```
[ ]  df_Class_Labels_Merged_unique.groupby('class')['Target'].unique()
```

```
class
Lung Opacity                    [1]
No Lung Opacity / Not Normal    [0]
Normal                          [0]
Name: Target, dtype: object
```

Conclusion on "class" and "Target" values:-

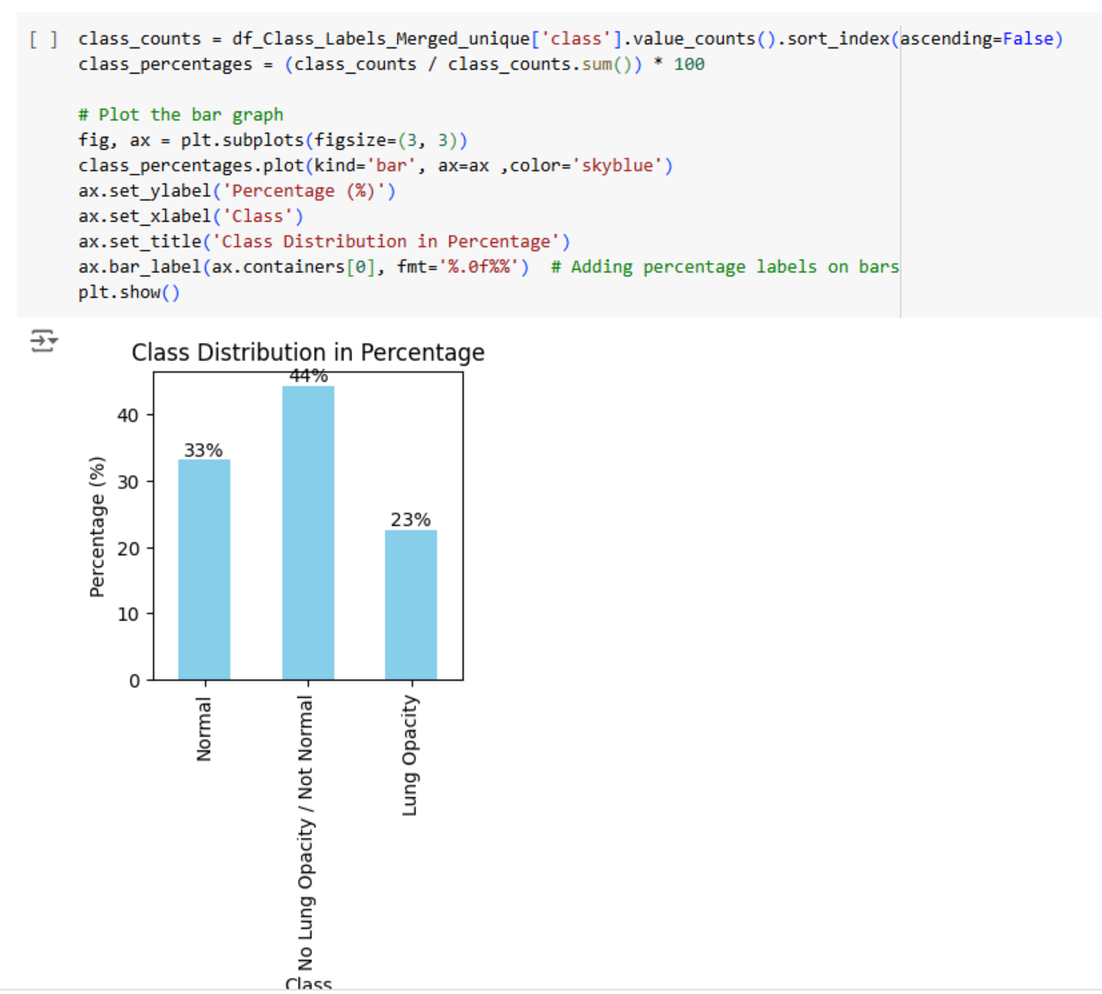
All class with "Lung Opacity" has target value [1]

Both class with "Normal" OR "No Lung Opacity / Not Normal" has target value [0]

Conclusion on "class" and "Target" values:-

- All class with "Lung Opacity" has a target value [1].
- Both class with "Normal" OR "No Lung Opacity / Not Normal" has target value [0].

Class Distribution in Percentage.



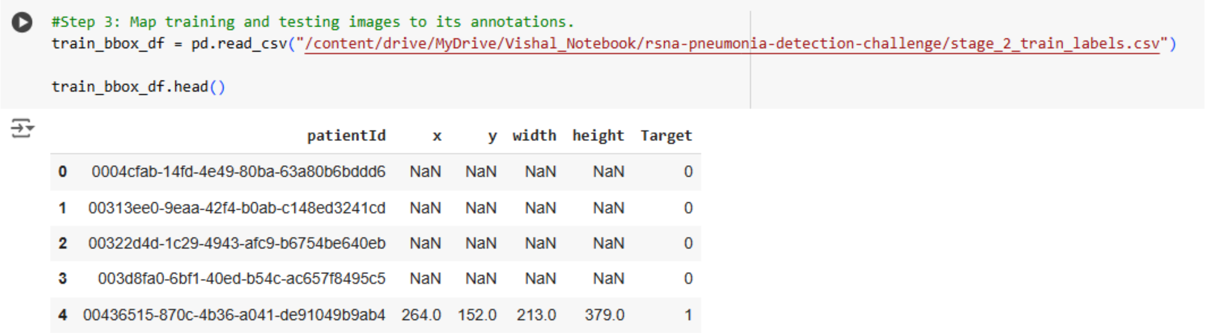
```
[ ]  class_counts = df_Class_Labels_Merged_unique['class'].value_counts().sort_index(ascending=False)
     class_percentages = (class_counts / class_counts.sum()) * 100

     # Plot the bar graph
     fig, ax = plt.subplots(figsize=(3, 3))
     class_percentages.plot(kind='bar', ax=ax ,color='skyblue')
     ax.set_ylabel('Percentage (%)')
     ax.set_xlabel('Class')
     ax.set_title('Class Distribution in Percentage')
     ax.bar_label(ax.containers[0], fmt='%.0f%%')  # Adding percentage labels on bars
     plt.show()
```

Class Distribution in Percentage
Percentage (%)
0
10
20
30
40
33%
44%
23%
Normal
No Lung Opacity / Not Normal
Lung Opacity
Class

Conclusion: All Pneumonia cases are of class "Lung Opacity" and target value 1

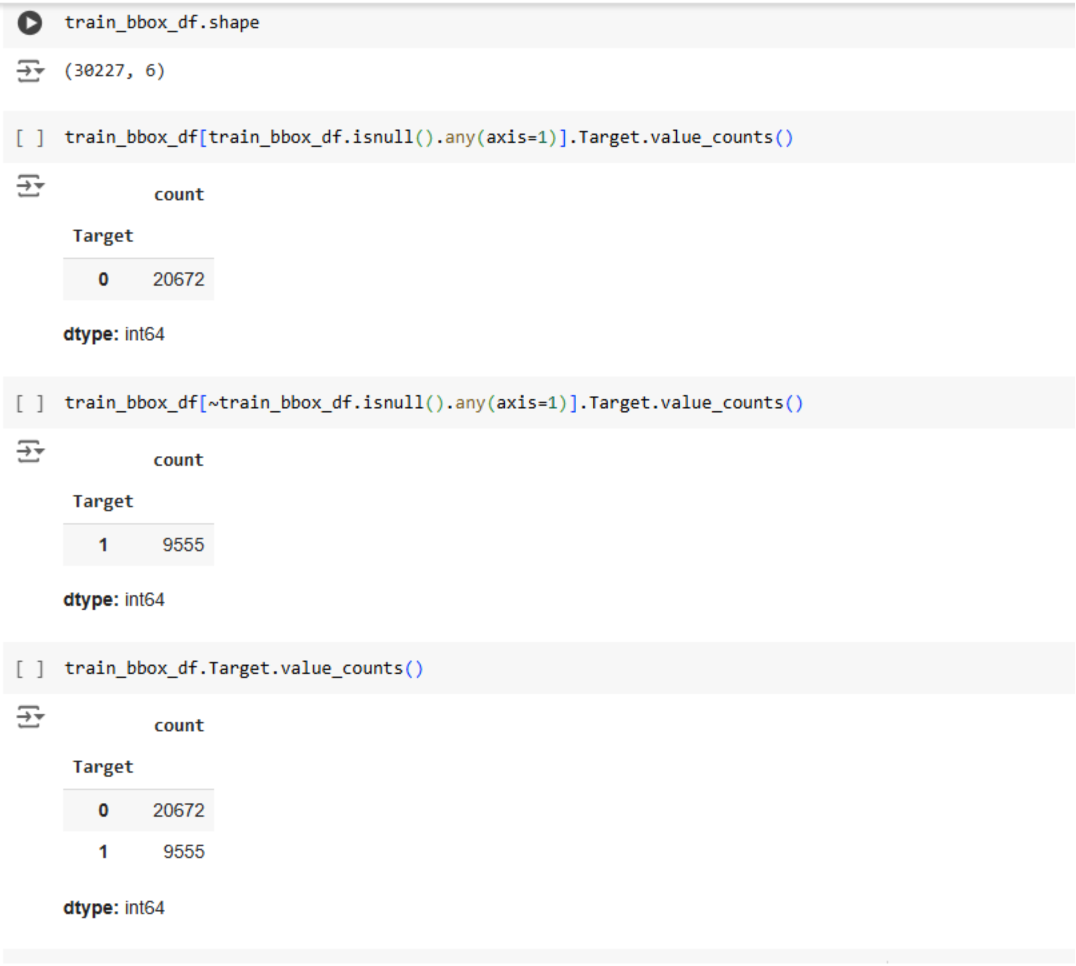
1. 23 % data has Lung Opacity
2. 33+ 44= 77% data has no Lung Opacity its either Normal or No Lung Opacity

Map training and testing images to their annotations



```
#Step 3: Map training and testing images to its annotations.
train_bbox_df = pd.read_csv("/content/drive/MyDrive/Vishal_Notebook/rsna-pneumonia-detection-challenge/stage_2_train_labels.csv")

train_bbox_df.head()
```

```
                                  patientId      x      y  width  height  Target
0  0004cfab-14fd-4e49-80ba-63a80b6bddd6    NaN    NaN    NaN     NaN       0
1  00313ee0-9eaa-42f4-b0ab-c148ed3241cd    NaN    NaN    NaN     NaN       0
2  00322d4d-1c29-4943-afc9-b6754be640eb    NaN    NaN    NaN     NaN       0
3  003d8fa0-6bf1-40ed-b54c-ac657f8495c5    NaN    NaN    NaN     NaN       0
4  00436515-870c-4b36-a041-de91049b9ab4  264.0  152.0  213.0   379.0       1
```



```
train_bbox_df.shape
```

```
(30227, 6)
```

```
train_bbox_df[train_bbox_df.isnull().any(axis=1)].Target.value_counts()
```

```
         count
Target
     0   20672

dtype: int64
```

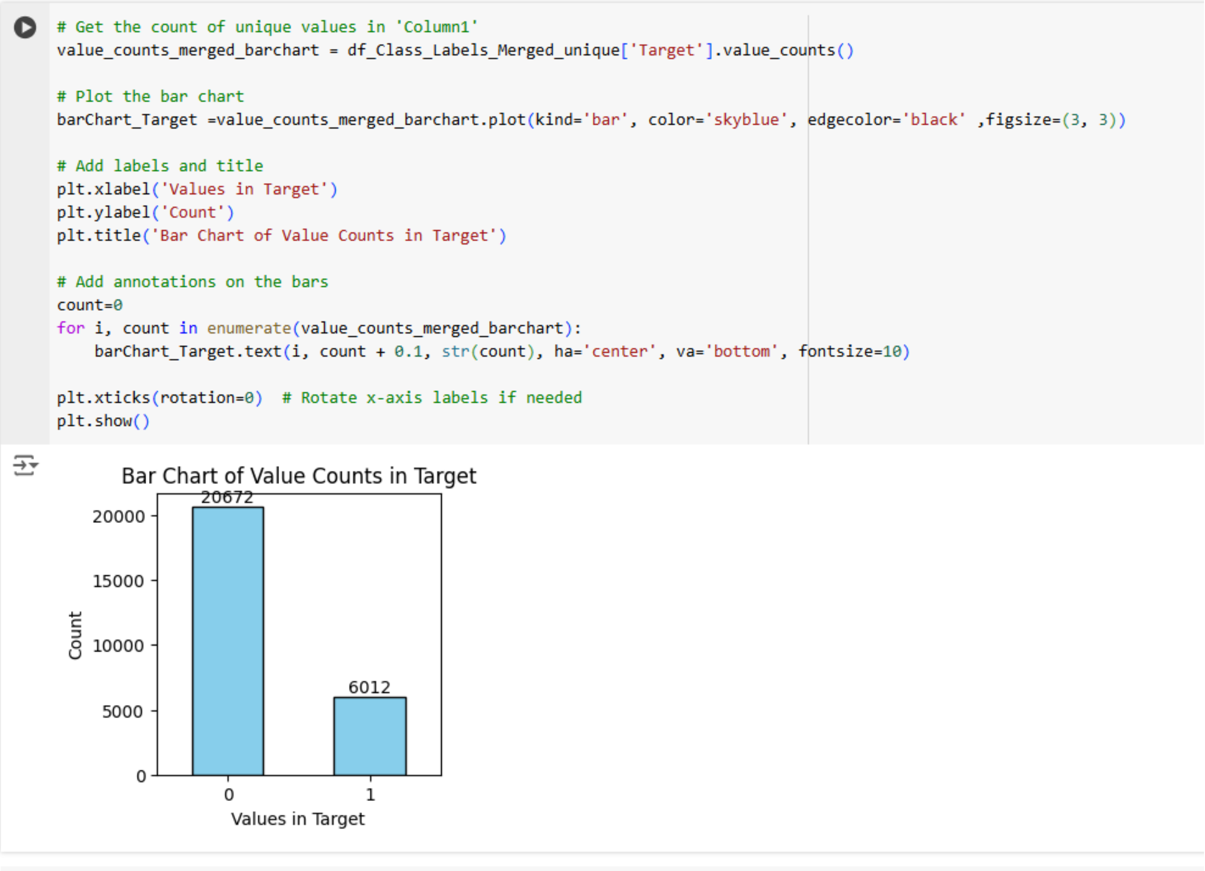
```
train_bbox_df[~train_bbox_df.isnull().any(axis=1)].Target.value_counts()
```

```
         count
Target
     1    9555

dtype: int64
```

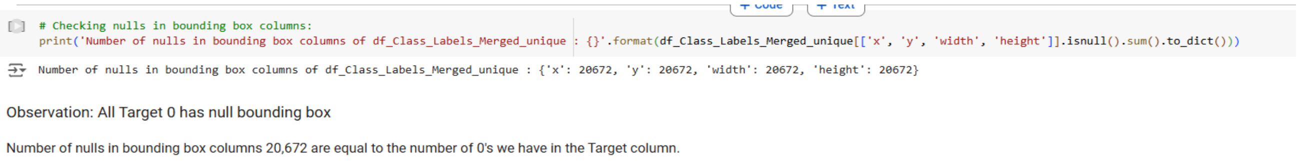
```
train_bbox_df.Target.value_counts()
```

```
         count
Target
     0   20672
     1    9555

dtype: int64
```

Value count in Target.



```
# Get the count of unique values in 'Column1'
value_counts_merged_barchart = df_Class_Labels_Merged_unique['Target'].value_counts()

# Plot the bar chart
barChart_Target =value_counts_merged_barchart.plot(kind='bar', color='skyblue', edgecolor='black' ,figsize=(3, 3))

# Add labels and title
plt.xlabel('Values in Target')
plt.ylabel('Count')
plt.title('Bar Chart of Value Counts in Target')

# Add annotations on the bars
count=0
for i, count in enumerate(value_counts_merged_barchart):
    barChart_Target.text(i, count + 0.1, str(count), ha='center', va='bottom', fontsize=10)

plt.xticks(rotation=0)  # Rotate x-axis labels if needed
plt.show()
```



```
# Checking nulls in bounding box columns:
print('Number of nulls in bounding box columns of df_Class_Labels_Merged_unique : {}'.format(df_Class_Labels_Merged_unique[['x', 'y', 'width', 'height']].isnull().sum().to_dict()))
```

```
Number of nulls in bounding box columns of df_Class_Labels_Merged_unique : {'x': 20672, 'y': 20672, 'width': 20672, 'height': 20672}
```

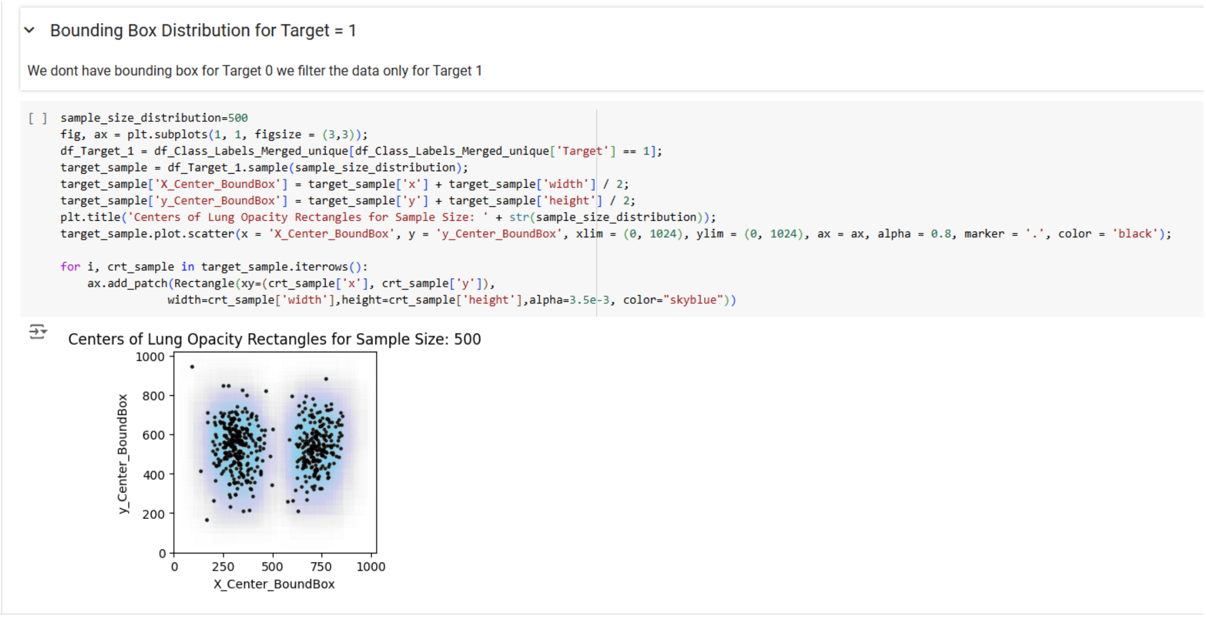
Observation: All Target 0 has null bounding box

Number of nulls in bounding box columns 20,672 are equal to the number of 0's we have in the Target column.

Observation: All Target 0 has a null bounding box

- The number of nulls in bounding box columns 20,672 is equal to the number of 0s we have in the Target column.

Bounding Box Distribution-



Bounding Box Distribution for Target = 1

We dont have bounding box for Target 0 we filter the data only for Target 1

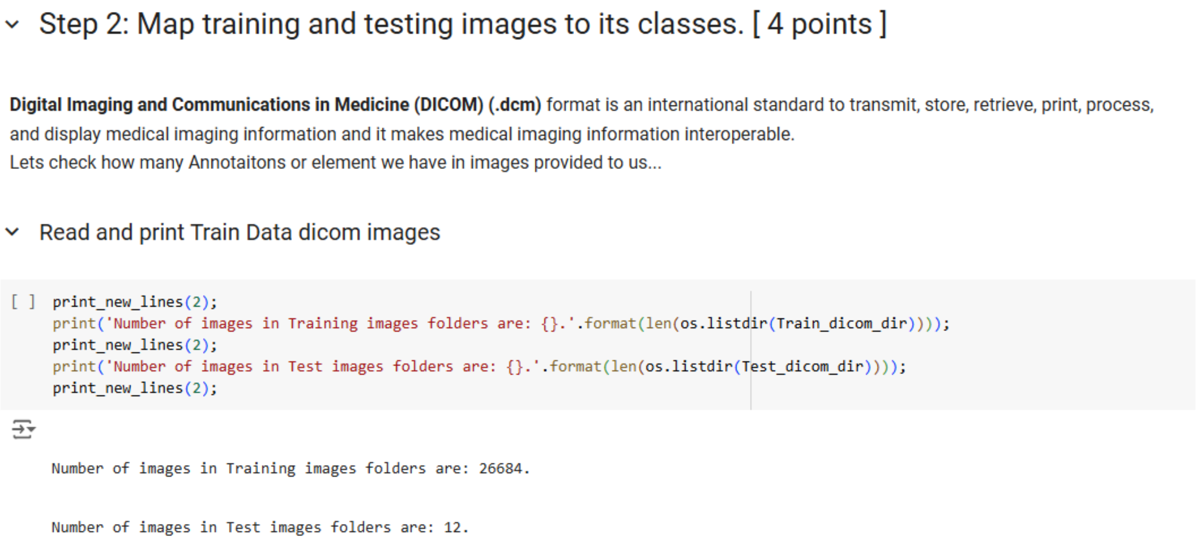
```
[ ] sample_size_distribution=500
    fig, ax = plt.subplots(1, 1, figsize = (3,3));
    df_Target_1 = df_Class_Labels_Merged_unique[df_Class_Labels_Merged_unique['Target'] == 1];
    target_sample = df_Target_1.sample(sample_size_distribution);
    target_sample['X_Center_BoundBox'] = target_sample['x'] + target_sample['width'] / 2;
    target_sample['y_Center_BoundBox'] = target_sample['y'] + target_sample['height'] / 2;
    plt.title('Centers of Lung Opacity Rectangles for Sample Size: ' + str(sample_size_distribution));
    target_sample.plot.scatter(x = 'X_Center_BoundBox', y = 'y_Center_BoundBox', xlim = (0, 1024), ylim = (0, 1024), ax = ax, alpha = 0.8, marker = '.', color = 'black');

    for i, crt_sample in target_sample.iterrows():
        ax.add_patch(Rectangle(xy=(crt_sample['x'], crt_sample['y']),
                    width=crt_sample['width'],height=crt_sample['height'],alpha=3.5e-3, color="skyblue"))
```

Centers of Lung Opacity Rectangles for Sample Size: 500

Conclusion:

Centres for the bounding box are spread out evenly across the lungs. Though a large portion of the bounding box has its centres at the centres of the lung, some centres of the box are also located at the edges of the lung.

Read and print Train Data dicom images



Step 2: Map training and testing images to its classes. [4 points]

Digital Imaging and Communications in Medicine (DICOM) (.dcm) format is an international standard to transmit, store, retrieve, print, process, and display medical imaging information and it makes medical imaging information interoperable.
Lets check how many Annotaitons or element we have in images provided to us...

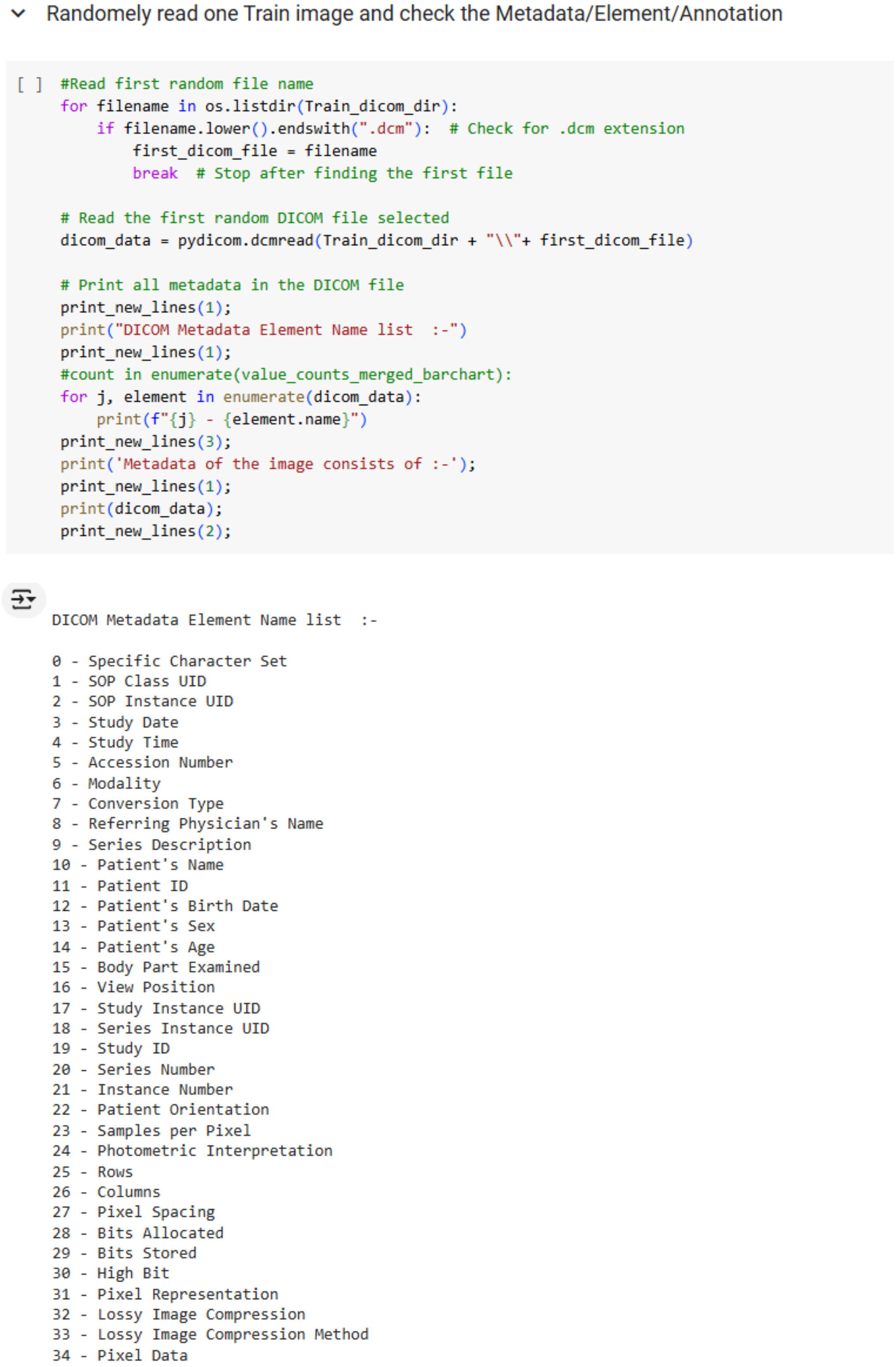
Read and print Train Data dicom images

```
[ ] print_new_lines(2);
    print('Number of images in Training images folders are: {}.'.format(len(os.listdir(Train_dicom_dir))));
    print_new_lines(2);
    print('Number of images in Test images folders are: {}.'.format(len(os.listdir(Test_dicom_dir))));
    print_new_lines(2);
```

```
Number of images in Training images folders are: 26684.


Number of images in Test images folders are: 12.
```

- The training images folder has **26,684** images
- unique patient IDs are also **26,684**

- Each of the unique patient IDs present in CSV files **corresponds** to an image present in the folder.



Randomely read one Train image and check the Metadata/Element/Annotation

```
[ ] #Read first random file name
    for filename in os.listdir(Train_dicom_dir):
        if filename.lower().endswith(".dcm"):  # Check for .dcm extension
            first_dicom_file = filename
            break  # Stop after finding the first file

    # Read the first random DICOM file selected
    dicom_data = pydicom.dcmread(Train_dicom_dir + "\\"+ first_dicom_file)

    # Print all metadata in the DICOM file
    print_new_lines(1);
    print("DICOM Metadata Element Name list  :-")
    print_new_lines(1);
    #count in enumerate(value_counts_merged_barchart):
    for j, element in enumerate(dicom_data):
        print(f"{j} - {element.name}")
    print_new_lines(3);
    print('Metadata of the image consists of :-');
    print_new_lines(1);
    print(dicom_data);
    print_new_lines(2);
```

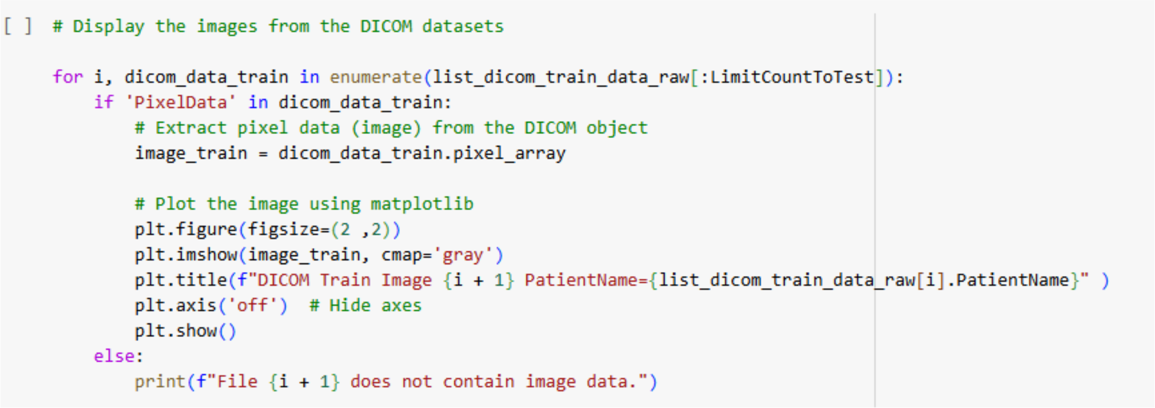
```
DICOM Metadata Element Name list  :-

0 - Specific Character Set
1 - SOP Class UID
2 - SOP Instance UID
3 - Study Date
4 - Study Time
5 - Accession Number
6 - Modality
7 - Conversion Type
8 - Referring Physician's Name
9 - Series Description
10 - Patient's Name
11 - Patient ID
12 - Patient's Birth Date
13 - Patient's Sex
14 - Patient's Age
15 - Body Part Examined
16 - View Position
17 - Study Instance UID
18 - Series Instance UID
19 - Study ID
20 - Series Number
21 - Instance Number
22 - Patient Orientation
23 - Samples per Pixel
24 - Photometric Interpretation
25 - Rows
26 - Columns
27 - Pixel Spacing
28 - Bits Allocated
29 - Bits Stored
30 - High Bit
31 - Pixel Representation
32 - Lossy Image Compression
33 - Lossy Image Compression Method
34 - Pixel Data
```

Observations –

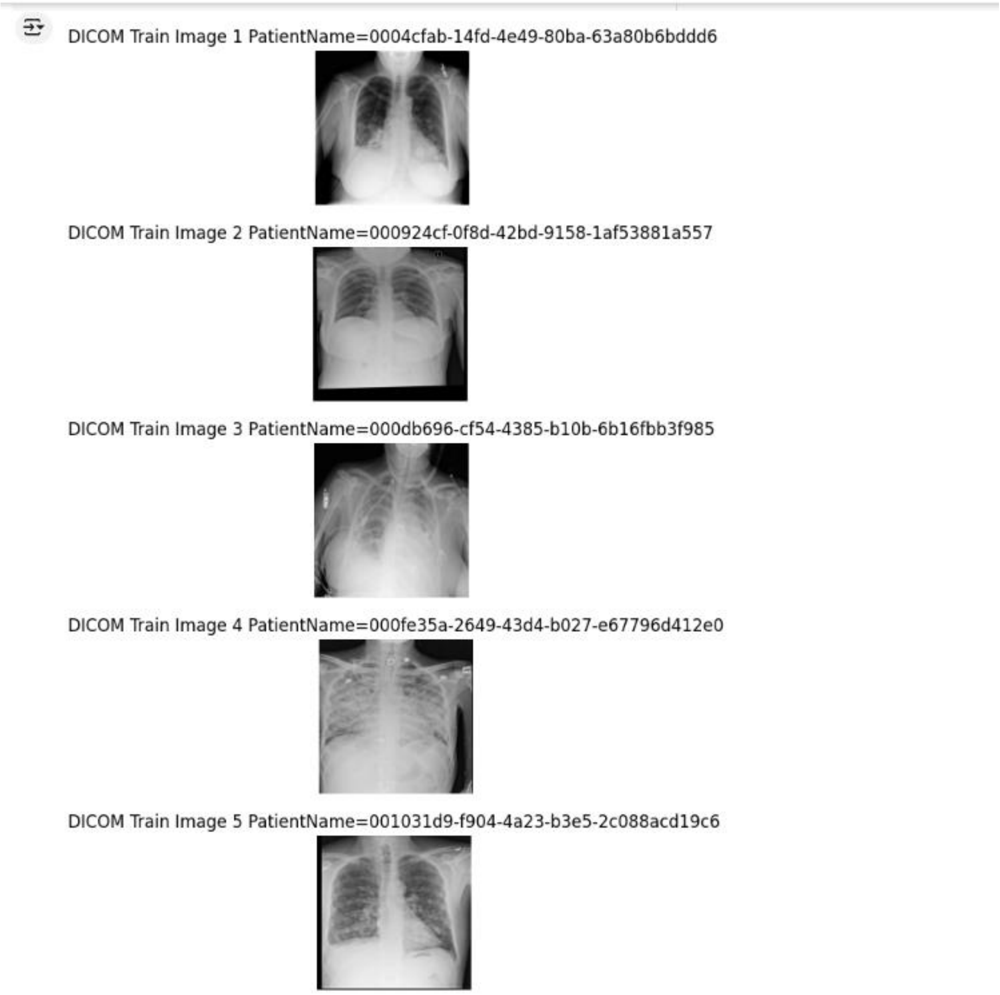
- The file has 35 Elements (0 to 34) including Pixel Data, which is the image

- From the above sample we can see that the dicom file contains some of the information that can be used for further analysis such as sex, age, body part examined, view position and modality.
- The size of this image in Pixes data Rows and Columns elements is 1024 x 1024 (rows x columns).

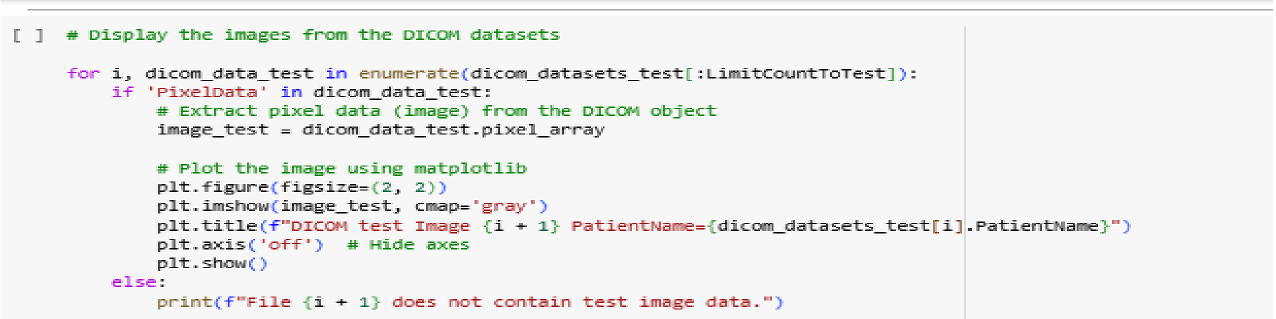
Images From Dicom Dataset(Train and Test)-



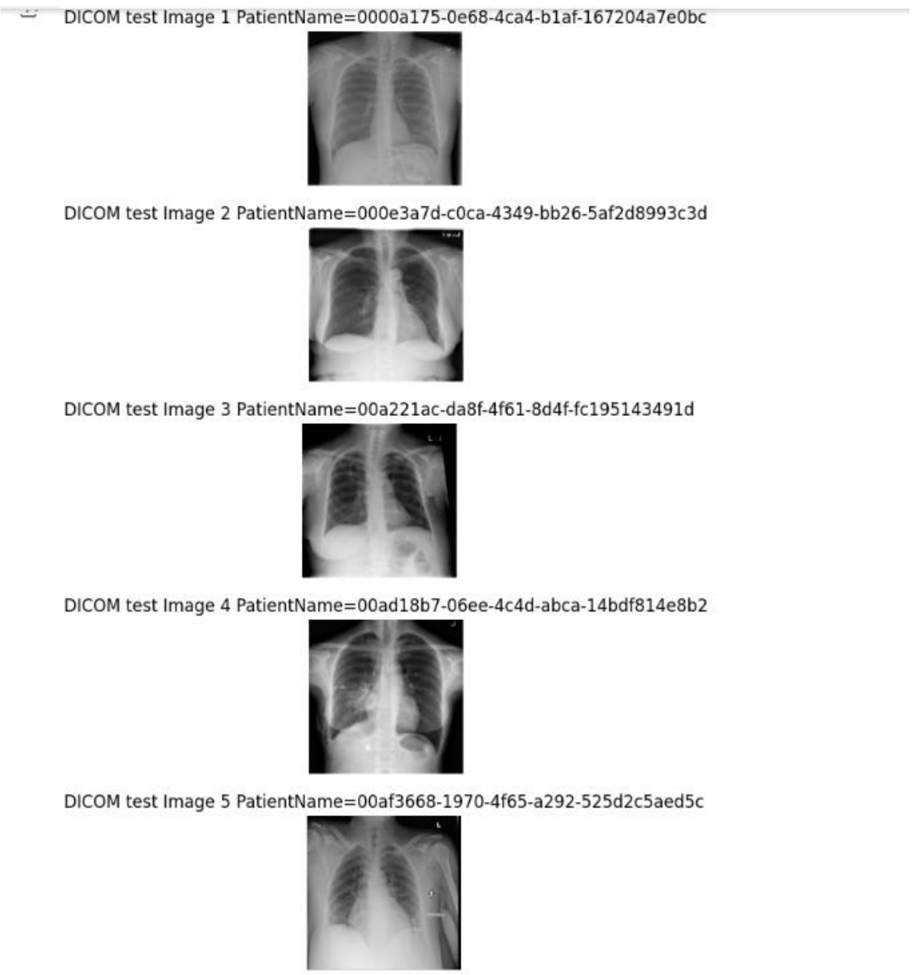
```
[ ]  # Display the images from the DICOM datasets

     for i, dicom_data_train in enumerate(list_dicom_train_data_raw[:LimitCountToTest]):
         if 'PixelData' in dicom_data_train:
             # Extract pixel data (image) from the DICOM object
             image_train = dicom_data_train.pixel_array

             # Plot the image using matplotlib
             plt.figure(figsize=(2 ,2))
             plt.imshow(image_train, cmap='gray')
             plt.title(f"DICOM Train Image {i + 1} PatientName={list_dicom_train_data_raw[i].PatientName}" )
             plt.axis('off')  # Hide axes
             plt.show()
         else:
             print(f"File {i + 1} does not contain image data.")
```



```
DICOM Train Image 1 PatientName=0004cfab-14fd-4e49-80ba-63a80b6bddd6

DICOM Train Image 2 PatientName=000924cf-0f8d-42bd-9158-1af53881a557

DICOM Train Image 3 PatientName=000db696-cf54-4385-b10b-6b16fbb3f985

DICOM Train Image 4 PatientName=000fe35a-2649-43d4-b027-e67796d412e0

DICOM Train Image 5 PatientName=001031d9-f904-4a23-b3e5-2c088acd19c6
```

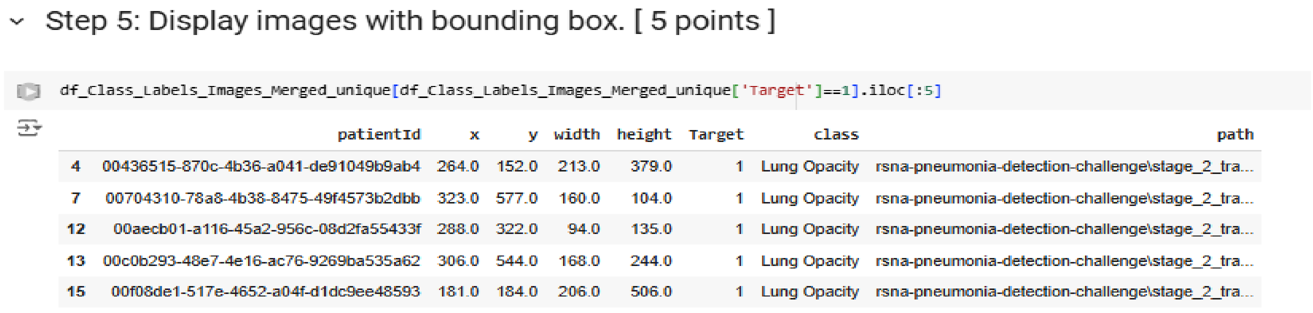


```
[ ]  # Display the images from the DICOM datasets

     for i, dicom_data_test in enumerate(dicom_datasets_test[:LimitCountToTest]):
         if 'PixelData' in dicom_data_test:
             # Extract pixel data (image) from the DICOM object
             image_test = dicom_data_test.pixel_array

             # Plot the image using matplotlib
             plt.figure(figsize=(2, 2))
             plt.imshow(image_test, cmap='gray')
             plt.title(f"DICOM test Image {i + 1} PatientName={dicom_datasets_test[i].PatientName}")
             plt.axis('off')  # Hide axes
             plt.show()
         else:
             print(f"File {i + 1} does not contain test image data.")
```



```
DICOM test Image 1 PatientName=0000a175-0e68-4ca4-b1af-167204a7e0bc
DICOM test Image 2 PatientName=000e3a7d-c0ca-4349-bb26-5af2d8993c3d
DICOM test Image 3 PatientName=00a221ac-da8f-4f61-8d4f-fc195143491d
DICOM test Image 4 PatientName=00ad18b7-06ee-4c4d-abca-14bdf814e8b2
DICOM test Image 5 PatientName=00af3668-1970-4f65-a292-525d2c5aed5c
```

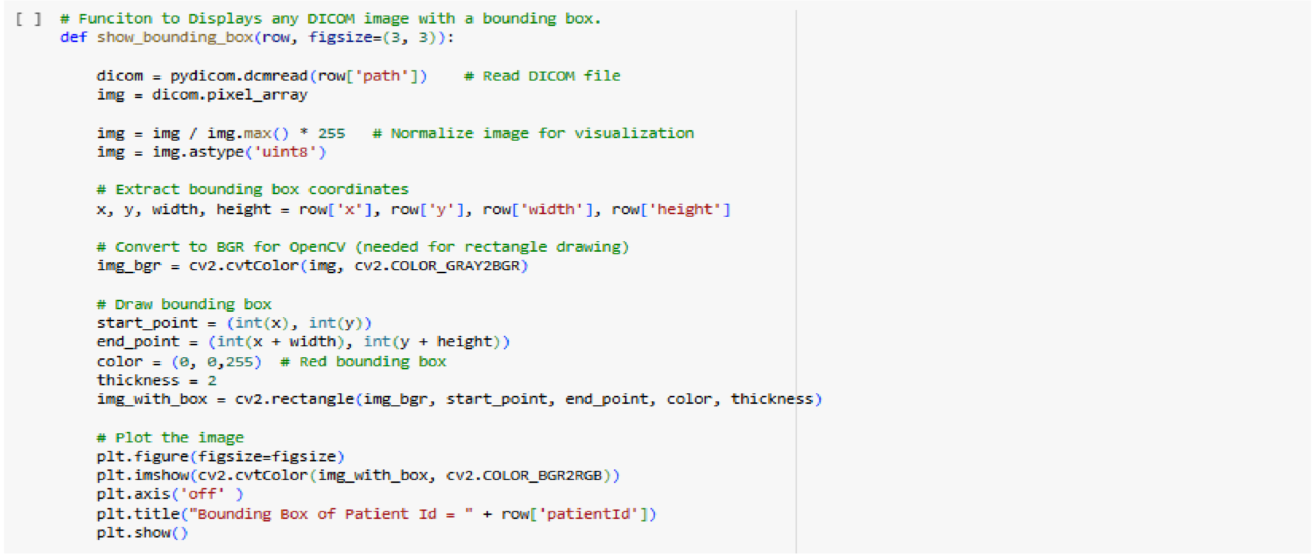
Display images with Bounding Box-



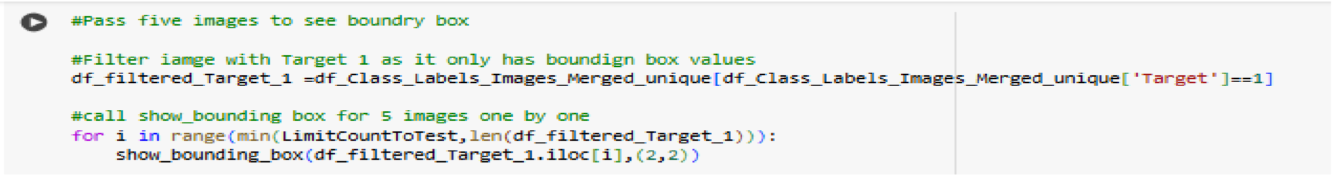
Step 5: Display images with bounding box. [5 points]

```
df_Class_Labels_Images_Merged_unique[df_Class_Labels_Images_Merged_unique['Target']==1].iloc[:5]
```

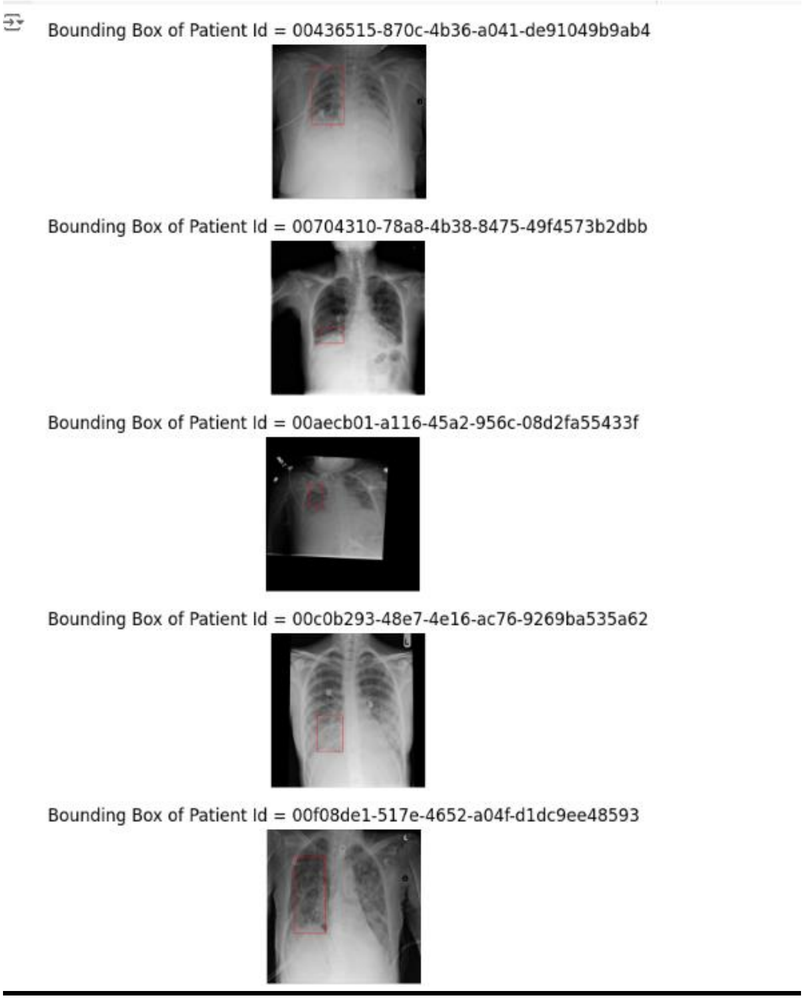
	patientId	x	y	width	height	Target	class	path
4	00436515-870c-4b36-a041-de91049b9ab4	264.0	152.0	213.0	379.0	1	Lung Opacity	rsna-pneumonia-detection-challenge\stage_2_tra...
7	00704310-78a8-4b38-8475-49f4573b2dbb	323.0	577.0	160.0	104.0	1	Lung Opacity	rsna-pneumonia-detection-challenge\stage_2_tra...
12	00aecb01-a116-45a2-956c-08d2fa55433f	288.0	322.0	94.0	135.0	1	Lung Opacity	rsna-pneumonia-detection-challenge\stage_2_tra...
13	00c0b293-48e7-4e16-ac76-9269ba535a62	306.0	544.0	168.0	244.0	1	Lung Opacity	rsna-pneumonia-detection-challenge\stage_2_tra...
15	00f08de1-517e-4652-a04f-d1dc9ee48593	181.0	184.0	206.0	506.0	1	Lung Opacity	rsna-pneumonia-detection-challenge\stage_2_tra...



```
# Funciton to Displays any DICOM image with a bounding box.
def show_bounding_box(row, figsize=(3, 3)):

    dicom = pydicom.dcmread(row['path'])    # Read DICOM file
    img = dicom.pixel_array

    img = img / img.max() * 255   # Normalize image for visualization
    img = img.astype('uint8')

    # Extract bounding box coordinates
    x, y, width, height = row['x'], row['y'], row['width'], row['height']

    # Convert to BGR for OpenCV (needed for rectangle drawing)
    img_bgr = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)

    # Draw bounding box
    start_point = (int(x), int(y))
    end_point = (int(x + width), int(y + height))
    color = (0, 0,255)  # Red bounding box
    thickness = 2
    img_with_box = cv2.rectangle(img_bgr, start_point, end_point, color, thickness)

    # Plot the image
    plt.figure(figsize=figsize)
    plt.imshow(cv2.cvtColor(img_with_box, cv2.COLOR_BGR2RGB))
    plt.axis('off' )
    plt.title("Bounding Box of Patient Id = " + row['patientId'])
    plt.show()
```

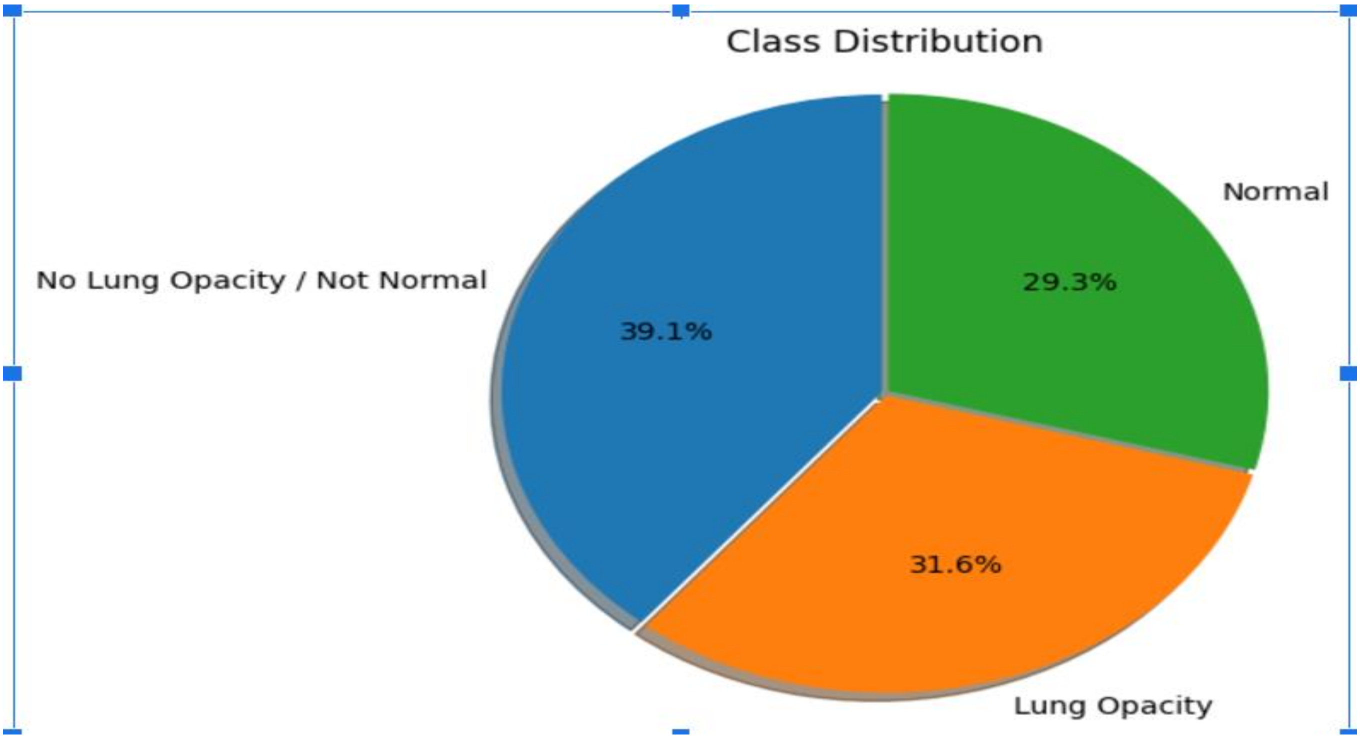


```
#Pass five images to see boundry box

#Filter iamge with Target 1 as it only has boundign box values
df_filtered_Target_1 =df_Class_Labels_Images_Merged_unique[df_Class_Labels_Images_Merged_unique['Target']==1]

#call show_bounding box for 5 images one by one
for i in range(min(LimitCountToTest,len(df_filtered_Target_1))):
    show_bounding_box(df_filtered_Target_1.iloc[i],(2,2))
```



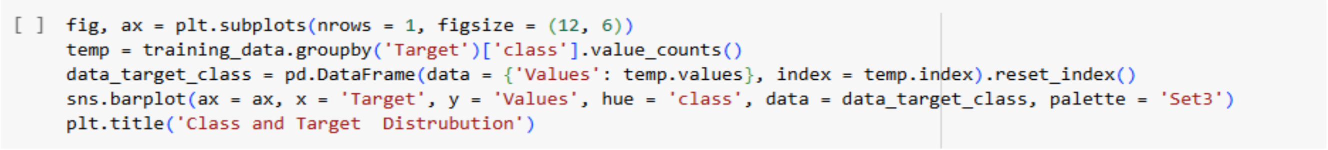
```
Bounding Box of Patient Id = 00436515-870c-4b36-a041-de91049b9ab4
Bounding Box of Patient Id = 00704310-78a8-4b38-8475-49f4573b2dbb
Bounding Box of Patient Id = 00aecb01-a116-45a2-956c-08d2fa55433f
Bounding Box of Patient Id = 00c0b293-48e7-4e16-ac76-9269ba535a62
Bounding Box of Patient Id = 00f08de1-517e-4652-a04f-d1dc9ee48593
```

Visualized class distributions to identify dataset imbalance.



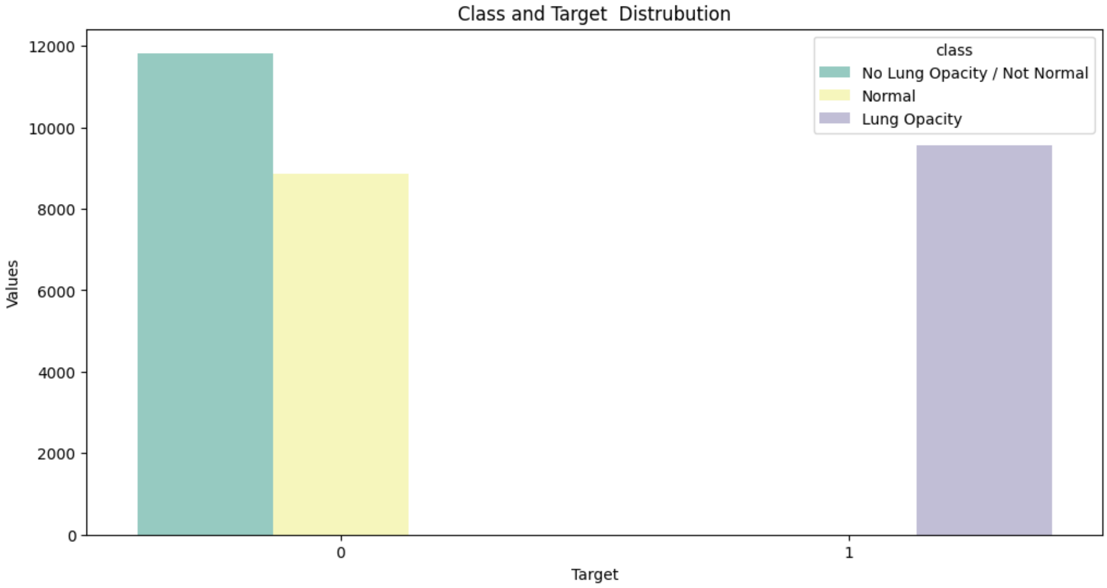
- Displayed example images from different classes with bounding boxes.
- Analyzed bounding box dimensions and pixel intensity distributions.
- Each patient ID corresponds to a unique image class as per below depending what is the current state of the patient's lung,
- 'No Lung Opacity / Not Normal'
- 'Normal'
- 'Lung Opacity'.

Class and Target Distribution-



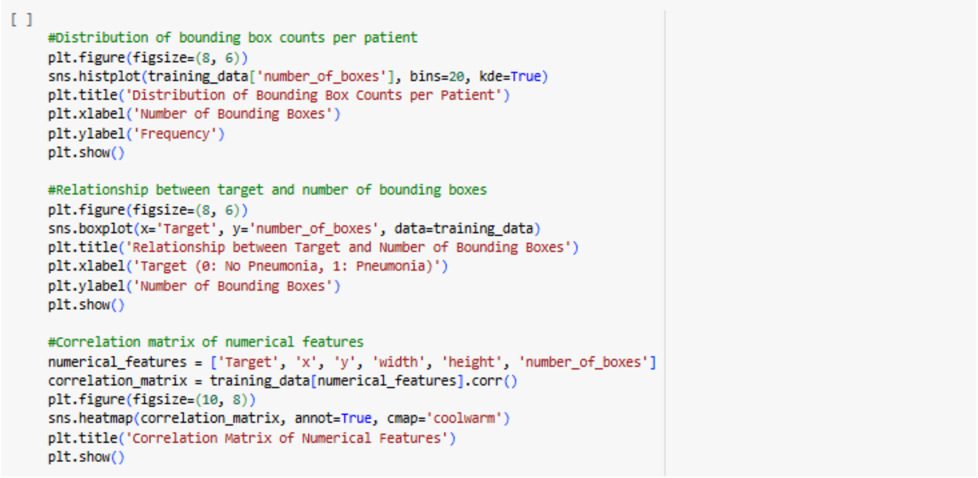
```
[ ] fig, ax = plt.subplots(nrows = 1, figsize = (12, 6))
    temp = training_data.groupby('Target')['class'].value_counts()
    data_target_class = pd.DataFrame(data = {'Values': temp.values}, index = temp.index).reset_index()
    sns.barplot(ax = ax, x = 'Target', y = 'Values', hue = 'class', data = data_target_class, palette = 'Set3')
    plt.title('Class and Target  Distrubution')
```

```
Text(0.5, 1.0, 'Class and Target  Distrubution')
```

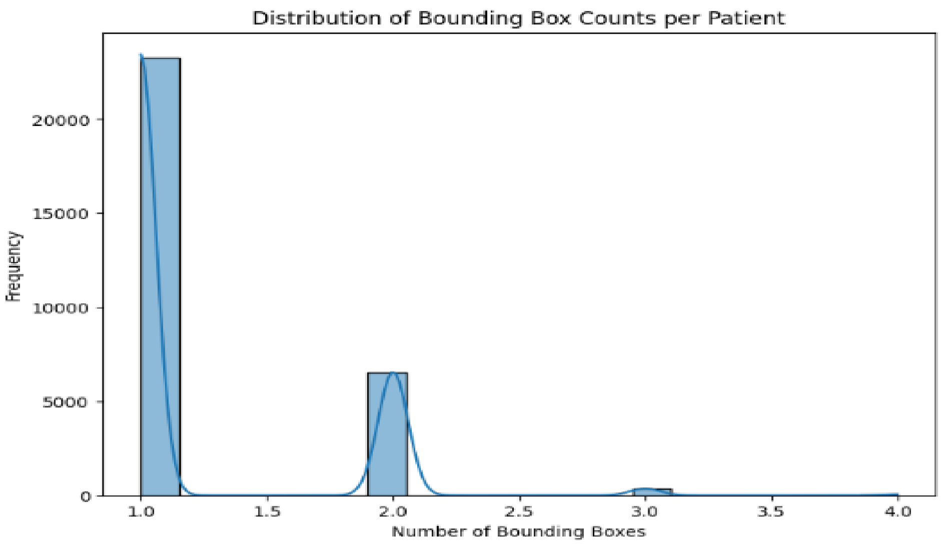


We can observe that class and Target Distribution where No lung opacity or Not Normal distribution is higher than both normal and lung opacity

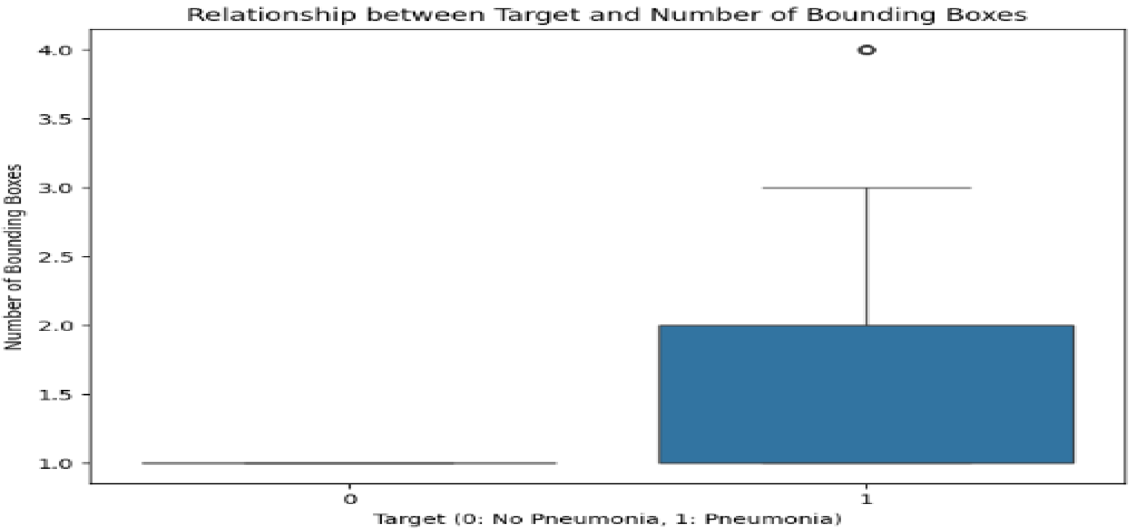
Distribution of bounding box count as per patient ID



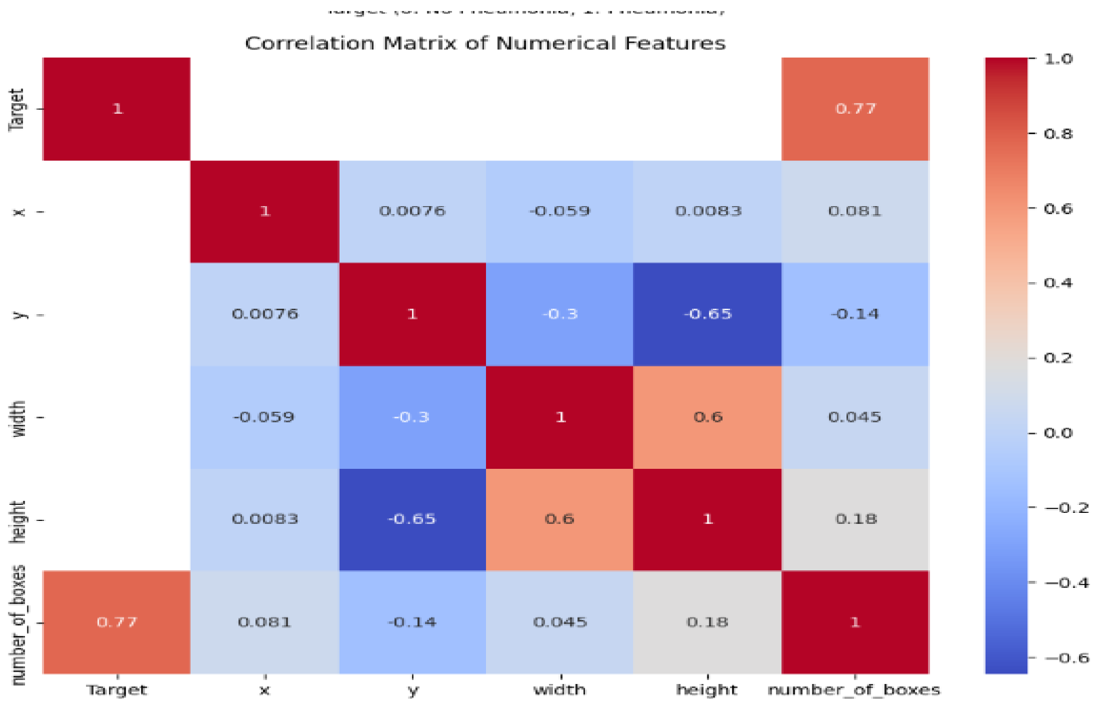
```
[ ]
    #Distribution of bounding box counts per patient
    plt.figure(figsize=(8, 6))
    sns.histplot(training_data['number_of_boxes'], bins=20, kde=True)
    plt.title('Distribution of Bounding Box Counts per Patient')
    plt.xlabel('Number of Bounding Boxes')
    plt.ylabel('Frequency')
    plt.show()

    #Relationship between target and number of bounding boxes
    plt.figure(figsize=(8, 6))
    sns.boxplot(x='Target', y='number_of_boxes', data=training_data)
    plt.title('Relationship between Target and Number of Bounding Boxes')
    plt.xlabel('Target (0: No Pneumonia, 1: Pneumonia)')
    plt.ylabel('Number of Bounding Boxes')
    plt.show()

    #Correlation matrix of numerical features
    numerical_features = ['Target', 'x', 'y', 'width', 'height', 'number_of_boxes']
    correlation_matrix = training_data[numerical_features].corr()
    plt.figure(figsize=(10, 8))
    sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
    plt.title('Correlation Matrix of Numerical Features')
    plt.show()
```



- Most patients have exactly 1 bounding box, indicating single detections in many cases.
- Fewer patients have multiple bounding boxes, with counts rapidly declining as the number of bounding boxes increases.



- Bounding boxes for Target 0: Patients without pneumonia consistently have a single bounding box, indicating simple, singular detections.
- Bounding boxes for Target 1: Pneumonia cases often involve multiple bounding boxes, which may correspond to multiple regions of interest in an image (e.g., affected areas or overlapping detections).



Correlation insights:

- The Target has a high positive correlation (0.77) with number_of_boxes, suggesting the number of boxes strongly influences the target variable.
- Weak correlations are observed between other features and the Target.
- width and height are positively correlated (0.6), indicating a relationship between the dimensions.
- The feature y has a notable negative correlation with height (-0.65), possibly reflecting a positional relationship.

Summary of the above classification-

1. Class Imbalance:

The pie chart shows a significant class imbalance, with a majority of patients being negative for pneumonia. This imbalance needs to be addressed during model training (e.g., using class weights or resampling techniques).

2. Bounding Box Analysis:

- The code shows that most patients have a single bounding box.
- A small percentage have multiple bounding boxes, indicating multiple pneumonia instances in a single image.

- The distribution plot of bounding boxes shows the frequency of patients with different numbers of boxes.
- The box plot comparing the target variable and the number of bounding boxes helps visualize the relationship between the presence of pneumonia and the number of bounding boxes detected.

3. Image Metadata Analysis: The code extracts various metadata fields (Modality, PatientAge, PatientSex, etc.) from DICOM files.

The analysis can be improved using this metadata:

- Patient Sex: Analyzing the relationship between patient sex and the presence or absence of pneumonia.
- Patient Age: Investigating the distribution of patient ages, identifying if certain age groups have a higher prevalence of pneumonia.
- The distributions of ages of all patients and patients with pneumonia are shown in separate histograms.
- Age bins were created, and counts of patients within those age bins were displayed.
- A bar chart displays the percentage of patients within each age group.
- Body Part Examined: Checking if all images are of the chest area.
- View Position: Analyzing the distribution of 'PA' and 'AP' views, potentially assessing if certain view positions might bias the model.
- A pie chart shows the distribution of view positions for patients with pneumonia.
- Pixel Spacing and Image Size: Analyzing if there are inconsistencies in image dimensions or pixel spacing that could affect model performance.
- The code prints the row and column size of the images.
- Correlation Matrix: The correlation heatmap reveals relationships between numerical features, including 'Target', bounding box coordinates, and the number of bounding boxes.

4. Data Visualization:

- The code visualizes the sample images with bounding boxes overlaid.
- The visualizations are crucial for understanding how the model is performing.

Summary

1. Class imbalance is present in the dataset.
2. The training dataset(both of the CSV files and training image folder) contains information on 26684 patients (unique)
3. Most of the recorded patients don't have pneumonia(target-0)
4. some of the patients have more than one bounding box. The maximum is 4.

5. The classes "No Lung Opacity/Not Normal" and "Normal" are associated with the target = 0 whereas "Lung Opacity" belong to Target = 1.
6. The images are present in dicom format, from which information like patientAge, PatientSex, ViewPosition etc are obtained
7. There are two ways from which images were obtained AP and PA. The age ranges from 1- 92.

We have performed Exploratory Data Analysis (EDA) on the dataset in the above steps which helped us identify the following items:

---

1. The shape of the dataset was checked.
2. Observations in class data and label data are checked.
3. The number of duplicate records in the dataset was checked.
4. The number of unique records in the dataset was checked.
5. The head of the dataset was printed.
6. The number of null values in the bounding boxes was checked.
7. The distribution of the 'Target' and 'class' columns was visualized.
8. The number of patient IDs per bounding box in the dataset was checked.
9. Bounding Box Images checked/Displayed
10. After Merging, the unique value of the image path, classes, Target and Bounding box value is checked.
11. Images in the Training image folder and Test image folder are checked.
12. The number of classes associated with each patient ID was checked.
13. Train_labels, class_info & train_images dataframes were merged.
14. Columns in the training images data frame were checked.
15. The shape of the train_class data frame after the merge was checked.
16. Class and Target Distribution are checked.
17. Imbalanced class data is checked.
18. Images from the Dicom Dataset (Train and Test) checked.
19. Bounding Box Distribution checked.
20. We found that the centre for the bounding box is spread out evenly across the lungs. Though a large portion of the bounding box has its centres at the centre of the lungs, some centres of the box are also located at the edges of the lungs.

5. Deciding Models and Model Building

Model Selection:

The key algorithm implemented is a **VGG16-like Convolutional Neural Network (CNN)** architecture.

- **Why VGG16-like CNN?**
  The VGG16 architecture is a proven approach for image classification tasks due to its ability to capture hierarchical features through sequential convolutional and pooling layers. This architecture provides a good balance of performance and computational complexity.
- **Dataset**: The input data consists of chest X-ray images stored in DICOM format. Images are classified into three categories:
  1. **No Lung Opacity/Not Normal**
  2. **Normal**
  3. **Lung Opacity**

1. Architecture Design

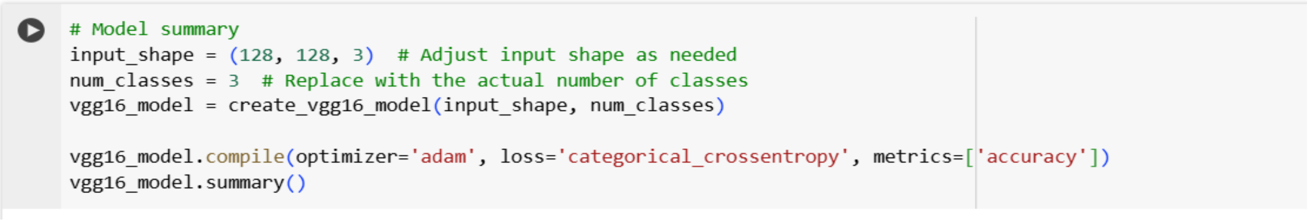
The model is based on a custom **VGG16-like CNN**, which incorporates several layers:

- Convolutional layers for feature extraction, capturing spatial hierarchies in the images (like edges, textures, and complex patterns).
- Max-pooling layers to downsample the feature maps, reducing dimensionality and enabling the model to focus on the most important features.
- Flattening layer to convert 2D feature maps into a 1D vector for further processing by dense layers.
- Dense layers for making predictions based on the features learned by the convolutional layers.
- Softmax output layer for multi-class classification, ensuring that the model can predict one of the three possible classes ("No Lung Opacity", "Normal", and "Lung Opacity").

The VGG16-inspired architecture was chosen due to its effectiveness in image classification tasks, particularly in medical imaging, where attention to fine-grained details is crucial.

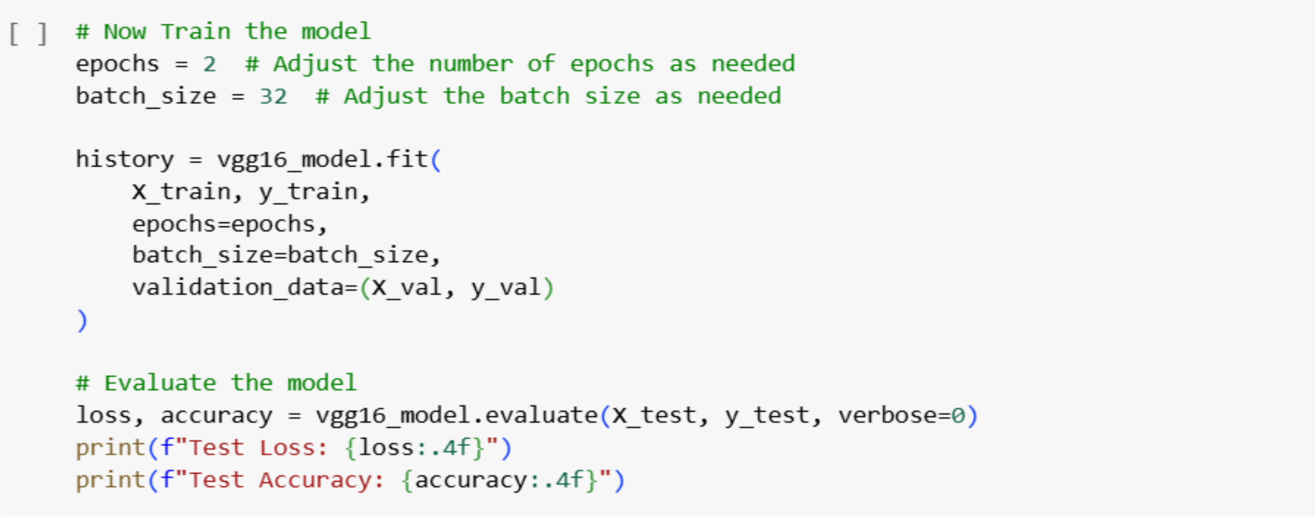
2. Model Compilation and Training

- The model is compiled using the Adam optimizer, which is an adaptive learning rate method that is often preferred for complex neural networks. This helps with efficient convergence and reduces the need for manual tuning of the learning rate.
- Categorical cross-entropy is used as the loss function, appropriate for multi-class classification tasks like this one, where each image belongs to one of several classes.
- Accuracy is used as the evaluation metric to monitor the model's performance during training.



```
# Model summary
input_shape = (128, 128, 3)  # Adjust input shape as needed
num_classes = 3  # Replace with the actual number of classes
vgg16_model = create_vgg16_model(input_shape, num_classes)

vgg16_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
vgg16_model.summary()
```

The model was trained for 30 epochs, with a batch size of 32. This setup strikes a balance between training speed and memory requirements, with sufficient epochs to allow the model to converge while not overfitting excessively.



```
[ ] # Now Train the model
    epochs = 2  # Adjust the number of epochs as needed
    batch_size = 32  # Adjust the batch size as needed

    history = vgg16_model.fit(
        X_train, y_train,
        epochs=epochs,
        batch_size=batch_size,
        validation_data=(X_val, y_val)
    )

    # Evaluate the model
    loss, accuracy = vgg16_model.evaluate(X_test, y_test, verbose=0)
    print(f"Test Loss: {loss:.4f}")
    print(f"Test Accuracy: {accuracy:.4f}")
```

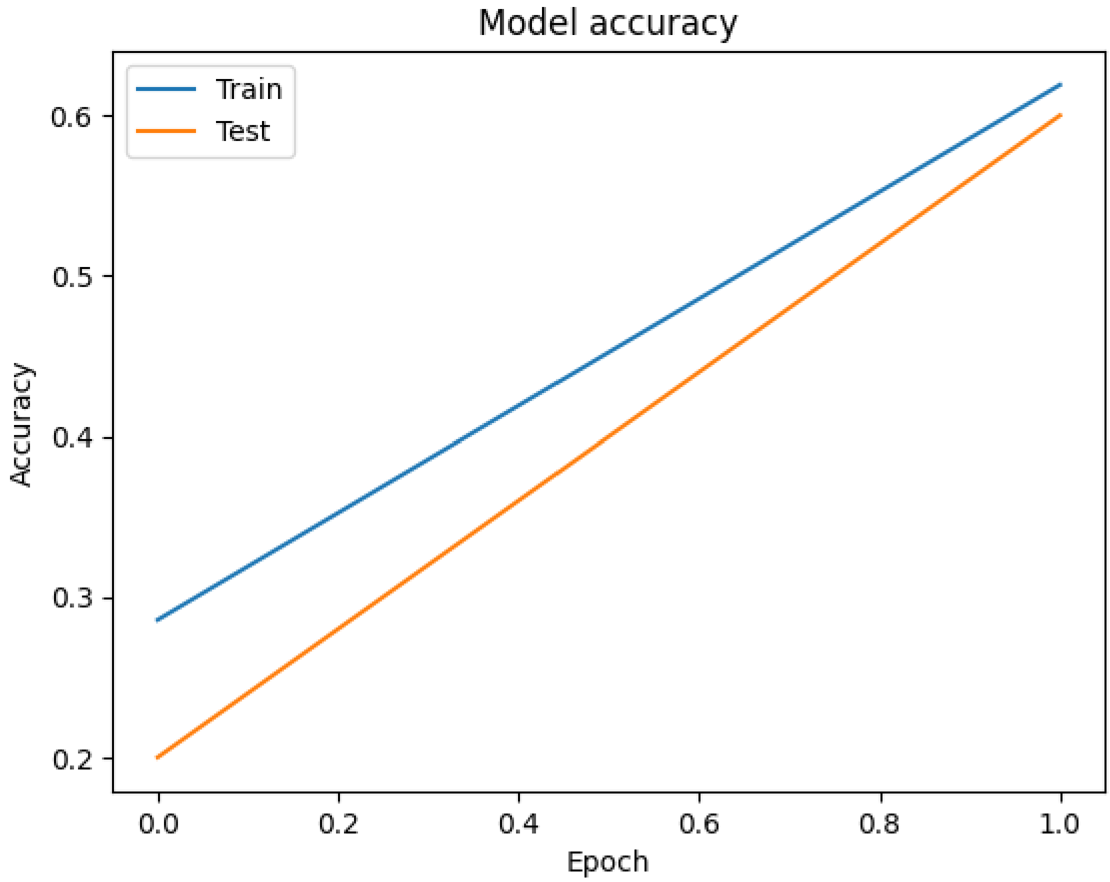
3. Evaluation and Results

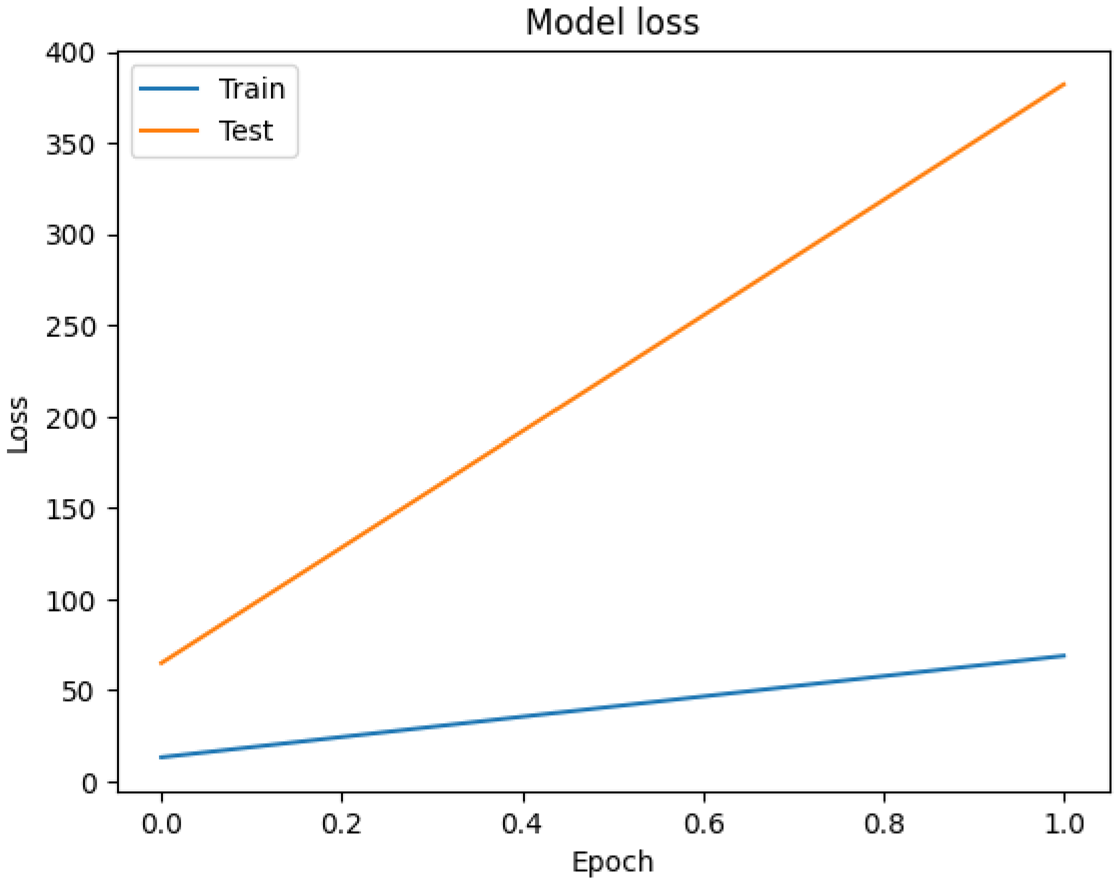
- After training, the model was evaluated on a separate test set, providing insights into how well the model generalizes to unseen data.

- Test loss and accuracy were the primary metrics used for evaluation. The results indicate that the model performs reasonably well, but may show signs of overfitting (i.e., when the validation loss diverges from training loss), a common challenge when training deep models.

4. Visualizations

- The training and validation accuracy/loss curves are key tools for understanding the model's learning process.





Insights and Implications

- Overall Model Performance: The combination of low accuracy and high loss values across training, validation, and test datasets indicates significant underperformance of the model.
- Generalization Issues: The drastic drop in validation accuracy compared to training accuracy suggests that the model may be overfitting to the training data rather than learning meaningful patterns that generalize to new data.

MobileNet:

MobileNet was selected after VGG16 to reduce computational complexity and memory requirements while maintaining reasonable accuracy. The performance metrics are:

- Training Accuracy: 66.67%
- Test Accuracy: 66.67%
- Demonstrates a balanced performance with a good generalization capability.

ResNet:

While MobileNet serves as an excellent starting point due to its efficiency and lower resource requirements, transitioning to ResNet allows for exploring higher accuracy potentials in more demanding tasks. The performance metrics are:

- Training Accuracy: 100.00%
- Test Accuracy: 0.00%
- Indicates severe overfitting, where the model has memorized the training data but fails to generalize to unseen data.

Model Performance Summary

Metric	VGG16 Value	MobileNet Value	ResNet Value
Training Accuracy	0.6190	0.6667	1.0000
Training Loss	68.7941	0.9733	0.0000
Test Loss	855.0109	0.9733	162.8343
Test Accuracy	0.2500	0.6667	0.0000

Summary of Results:

- Training Performance: ResNet shows perfect training accuracy and loss, indicating it has perfectly fit the training data.
- Test Performance: MobileNet outperforms both VGG16 and ResNet in terms of test accuracy, while ResNet shows a concerning test accuracy of 0.0000.

How to Improve Model Performance?

The performance of the VGG16-like CNN model can be enhanced through several strategies aimed at improving generalization and reducing errors. One key area is handling class imbalance, which is crucial for ensuring that the model performs well across all classes. This can be addressed by oversampling the minority classes, such as using the Synthetic Minority Oversampling Technique (SMOTE), or by undersampling the majority classes. Alternatively, implementing class weights during training can penalize errors in underrepresented classes, ensuring a balanced learning process.

Data augmentation is another effective method for improving the model. By applying random transformations to the training images, such as rotations, scaling, or flipping, the model is exposed to diverse variations of the input data, helping it generalize better and reducing the likelihood of overfitting.

Hyperparameter tuning can further optimize the model’s performance. This involves experimenting with various learning rates, batch sizes, and network parameters like the number of layers or filters. Automated tools like Optuna or manual grid searches can be used to systematically find the best combination of hyperparameters for the task.

Additionally, feature engineering could be leveraged by extracting more information from the DICOM metadata, such as patient age or the position in which the X-ray was taken. Incorporating such domain-specific features can add valuable context to the model’s predictions and improve accuracy.

Regularization techniques can be employed to mitigate overfitting. Adding dropout layers to the network architecture randomly disables a subset of neurons during training, reducing co-adaptation and enhancing the model's robustness. Similarly, L1 or L2 regularization can constrain the weight magnitudes by adding a penalty term to the loss function.

Error analysis plays a significant role in model improvement. Misclassified samples should be thoroughly examined to understand the reasons behind the errors, such as noisy data, ambiguous cases, or visually similar classes. Correcting labelling errors or refining the dataset based on these insights can improve the model's overall performance.

Finally, exploring advanced architectures or leveraging pre-trained models like ResNet, EfficientNet, or Inception through transfer learning can provide a substantial boost. Pre-trained models benefit from large-scale training on diverse datasets, which often leads to better feature extraction and classification accuracy compared to custom architectures.

These combined strategies can significantly enhance the robustness, accuracy, and real-world applicability of the VGG16-like CNN model for pneumonia detection.

Conclusion:

The performance comparison of three prominent deep learning models—VGG16, MobileNet, and ResNet—reveals significant differences in their training and test metrics. Further fine-tuning of the model is required to enhance accuracy and reduce loss. With strategic improvements such as advanced data augmentation, handling class imbalance, and leveraging transfer learning, the model can achieve higher accuracy and robustness. The addition of metadata and further hyperparameter optimization will enhance generalization and performance.