AIML INTERIM REPORT
Group-2 CV 1

CAPSTONE PROJECT

TABLE OF CONTENTS

ABOUT THIS DOCUMENT

Title: Interim Report

Date: 03/12/2023

Project Name: PNEUMONIA DETECTION CHALLENGE

Authors/Contributors: **Sephalika Nayak, Aabshar Pasha, Darshan Sethiya & Manvendra Wagadre**

Purpose: To design a DL based algorithm for detecting pneumonia.

Acknowledgments: Special Thanks to Mr Jayant for mentoring us during this Capstone project.

PROBLEM STATEMENT

- **DOMAIN**: Health Care
- **CONTEXT**:

Computer vision can be used in health care for identifying diseases. In Pneumonia detection we need to detect Inflammation of the lungs. In this challenge, you're required to build an algorithm to detect a visual signal for pneumonia in medical images. Specifically, your algorithm needs to automatically locate lung opacities on chest radiographs.

- **DATA DESCRIPTION**:

In the dataset, some of the features are labeled "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. Dicom original images: - Medical images are stored in a special format called DICOM files (*.dcm). They contain a combination of header metadata as well as underlying raw image arrays for pixel data.

Dataset has been attached along with this project. Please use the same for this capstone project

Original link to the dataset: https://www.kaggle.com/c/rsna-pneumonia-detection-challenge/data [for your reference only]. You can refer to the det ails of the dataset in the above link

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

- **PROJECT OBJECTIVE**: Design a DL based algorithm for detecting pneumonia.
- **PROJECT TASK**: [Score: 100 points]

Milestone 1: [Score: 40 points]

Input: Context and Dataset

Process:

- Import the data. [3 points]
- Map training and testing images to its classes. [4 points]
- Map training and testing images to its annotations. [4 points]
- Preprocessing and Visualisation of different classes [4 Points]
- Display images with bounding box. [5 points]
- Design, train and test basic CNN models for classification. [10 points]
- Interim report [10 points]

PROJECT

MILESTONE 1

We had downloaded data to our local machine and unzipped to a location in local folder.

Local path:

C:\Users\manve\Downloads\rsna-pneumonia-detection-challenge

STEP 1: IMPORT THE DATA.

We had used Python with the pandas library to read CSV files and define file paths.

Step 1: Import the data.

```
In [7]: classInfo = pd.read_csv(r'C:\Users\manve\Downloads\rsna-pneumonia-detection-challenge\stage_2_detailed_class_info.csv')
        trainlabels = pd.read_csv(r'C:\Users\manve\Downloads\rsna-pneumonia-detection-challenge\stage_2_train_labels.csv')
        trainImagesPath = Path(r'C:\Users\manve\Downloads\rsna-pneumonia-detection-challenge\stage_2_train_images')
        testImagesPath = Path(r'C:\Users\manve\Downloads\rsna-pneumonia-detection-challenge\stage_2_test_images')
        sampleSubPath = Path(r'C:\Users\manve\Downloads\rsna-pneumonia-detection-challenge\stage_2_sample_submission.csv')
```

EXPLORATORY DATA ANALYSIS:

There are 26684 unique patient info available. Total numbers of records are 30227, but unique patient IDs are 26684. We observe some duplicated records for patient Id.

```
In [9]: classInfo.head()
```

Out[9]:

	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

```
In [10]: classInfo.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30227 entries, 0 to 30226
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   patientId  30227 non-null  object
 1   class      30227 non-null  object
dtypes: object(2)
memory usage: 472.4+ KB
```

There are two features 1. Patient ID 2. Class.

```
[13] classInfo.shape

(30227, 2)

[14] classInfo.patientId.nunique()

26684
```

we observed 3543 duplicated records.

```
classInfo[classInfo.duplicated()].shape

(3543, 2)
```

In [18]:
```
def missing_check(df):
    total = df.isnull().sum().sort_values(ascending=False)  # total number of null values
    percent = (df.isnull().sum() / df.isnull().count()).sort_values(
        ascending=False)  # percentage of values that are null
    missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])  # putting the above two together
    return missing_data  # return the dataframe
```

In [19]:
```
missing_check(classInfo)
```

Out[19]:

	Total	Percent
patientId	0	0.0
class	0	0.0

There are no missing values.

Reading the Trainlabels dataset:

```
trainlabels = pd.read_csv(r'C:\Users\manve\Downloads\rsna-pneumonia-detection-challenge\stage_2_train_labels.csv')
trainlabels.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

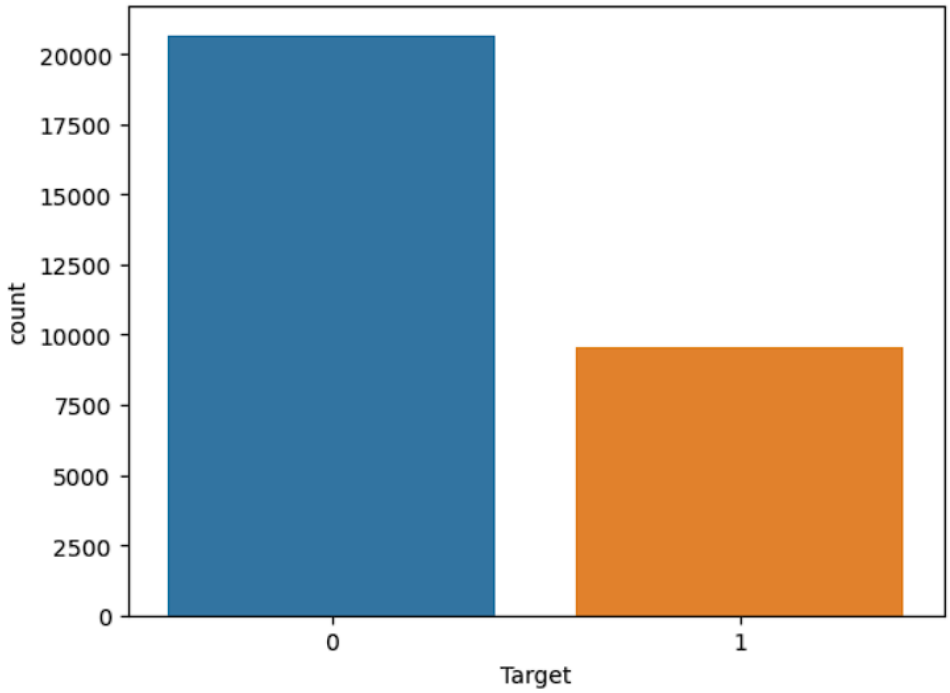
```
trainlabels.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30227 entries, 0 to 30226
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   patientId  30227 non-null  object
 1   x          9555 non-null   float64
 2   y          9555 non-null   float64
 3   width      9555 non-null   float64
 4   height     9555 non-null   float64
 5   Target     30227 non-null  int64
dtypes: float64(4), int64(1), object(1)
memory usage: 1.4+ MB
```

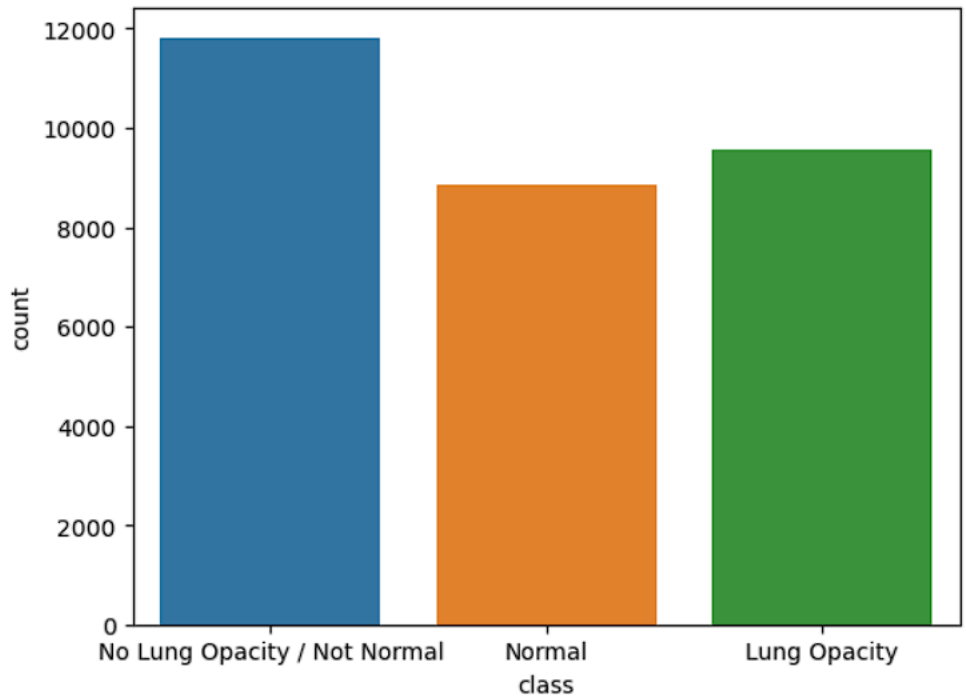
We observed there are X, Y values are missing for few records. This can be due to the fact that for a normal patient these values could be not applicable.

We noticed 75% of the data represents target value 1. We can see the same in Count plot. It's an imbalanced data set.

```
In [30]: sns.countplot(x='Target',data=trainlabels);
```




```
In [13]: sns.countplot(x='class',data=classInfo);
```



We can see three different classes in the above count plot which are

1. Normal
2. No Lung Opacity / Not Normal
3. Lung Opacity

In [14]:
```
def get_feature_distribution(data, feature):
    # Get the count for each label
    label_counts = data[feature].value_counts()

    # Get total number of samples
    total_samples = len(data)

    # Count the number of items in each class
    print("Feature: {}".format(feature))
    for i in range(len(label_counts)):
        label = label_counts.index[i]
        count = label_counts.values[i]
        percent = int((count / total_samples) * 10000) / 100
        print("{:<30s}:   {} or {}%".format(label, count, percent))

get_feature_distribution(classInfo, 'class')
```

```
Feature: class
No Lung Opacity / Not Normal  :   11821 or 39.1%
Lung Opacity                  :   9555 or 31.61%
Normal                        :   8851 or 29.28%
```

Its proved that only for normal patients dimensions are not available.

```
trainlabels.describe()
```

	x	y	width	height	Target
count	9555.000000	9555.000000	9555.000000	9555.000000	30227.000000
mean	394.047724	366.839560	218.471376	329.269702	0.316108
std	204.574172	148.940488	59.289475	157.750755	0.464963
min	2.000000	2.000000	40.000000	45.000000	0.000000
25%	207.000000	249.000000	177.000000	203.000000	0.000000
50%	324.000000	365.000000	217.000000	298.000000	0.000000
75%	594.000000	478.500000	259.000000	438.000000	1.000000
max	835.000000	881.000000	528.000000	942.000000	1.000000

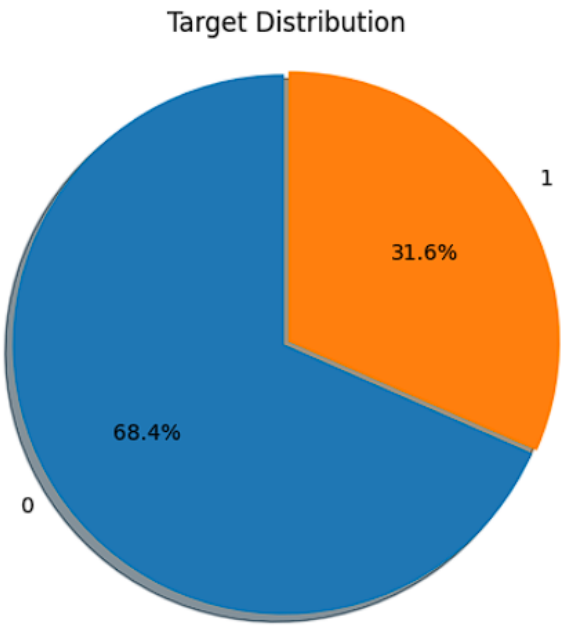
Trainlabels contains 30227 records same as meta data.

We had concatenated classInfo and Trainlabels. Before concatenating them, we removed duplicate records from classInfo.

We observed X & Y values are missing for few records. This can be due to the fact that for a normal patient these values could be not applicable.

There are 31.6% of patients with pneumonia and the remaining 68.4% are no pneumonia.

```
label_count=trainlabels['Target'].value_counts()
explode = (0.01,0.01)

fig1, ax1 = plt.subplots(figsize=(5,5))
ax1.pie(label_count.values, explode=explode, labels=label_count.index, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')
plt.title('Target Distribution')
plt.show()
```



STEP 2: MAP TRAINING AND TESTING IMAGES TO ITS CLASSES.

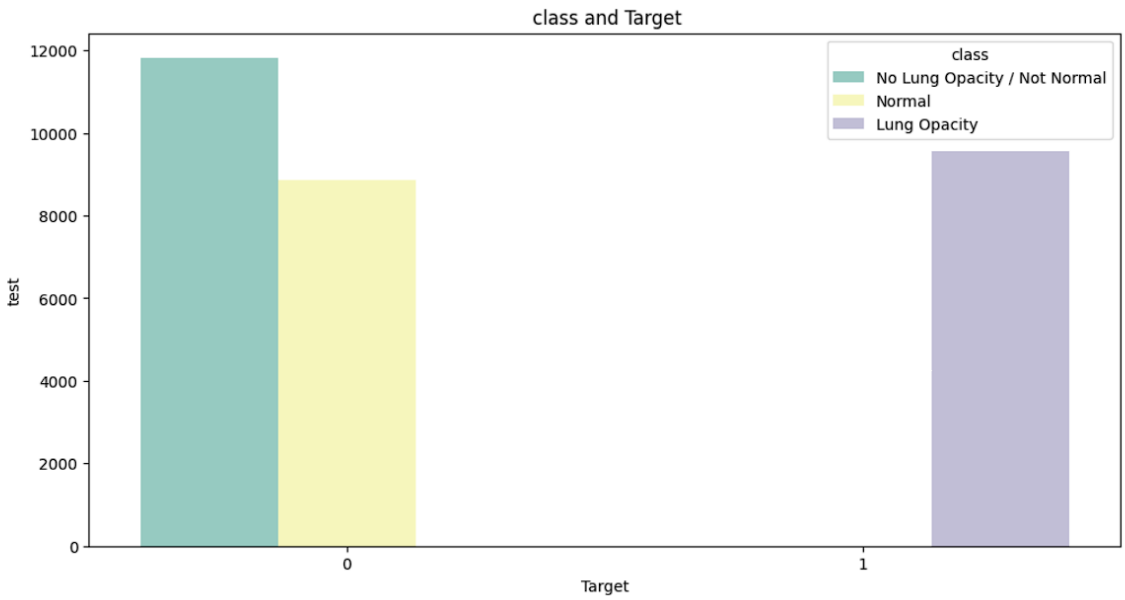
We had done inner merge between two DataFrames, trainlabels and classInfo, based on the common column 'patientId'.

```
# inner merge between two DataFrames, trainlabels and classInfo, based on the common column 'patientId'.
traindf = trainlabels.merge(classInfo, left_on='patientId', right_on='patientId', how='inner')
```

```
traindf.head()
```

	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

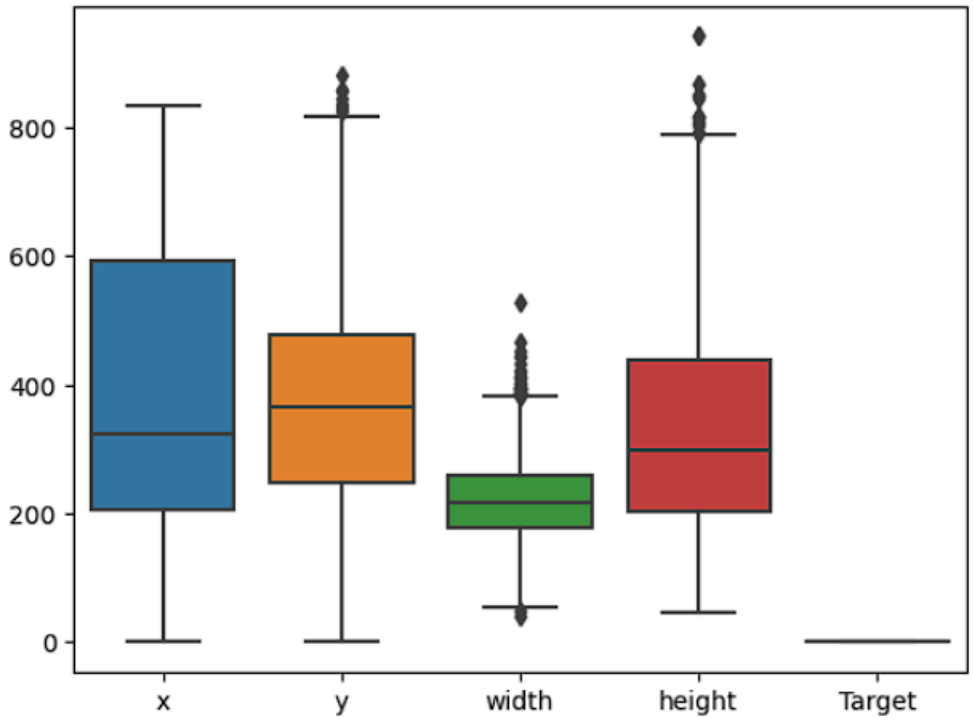
We observed class with Normal and No Lung Opacity / Not Normal has been classified into single target value that is 'O'. So, we can say that the prediction which we have to do is like patient has Lung Opacity or not. Because Normal and not normal patients are combines in same Target.



Prediction: Binary classification i.e., Patient has Lung Opacity or not?

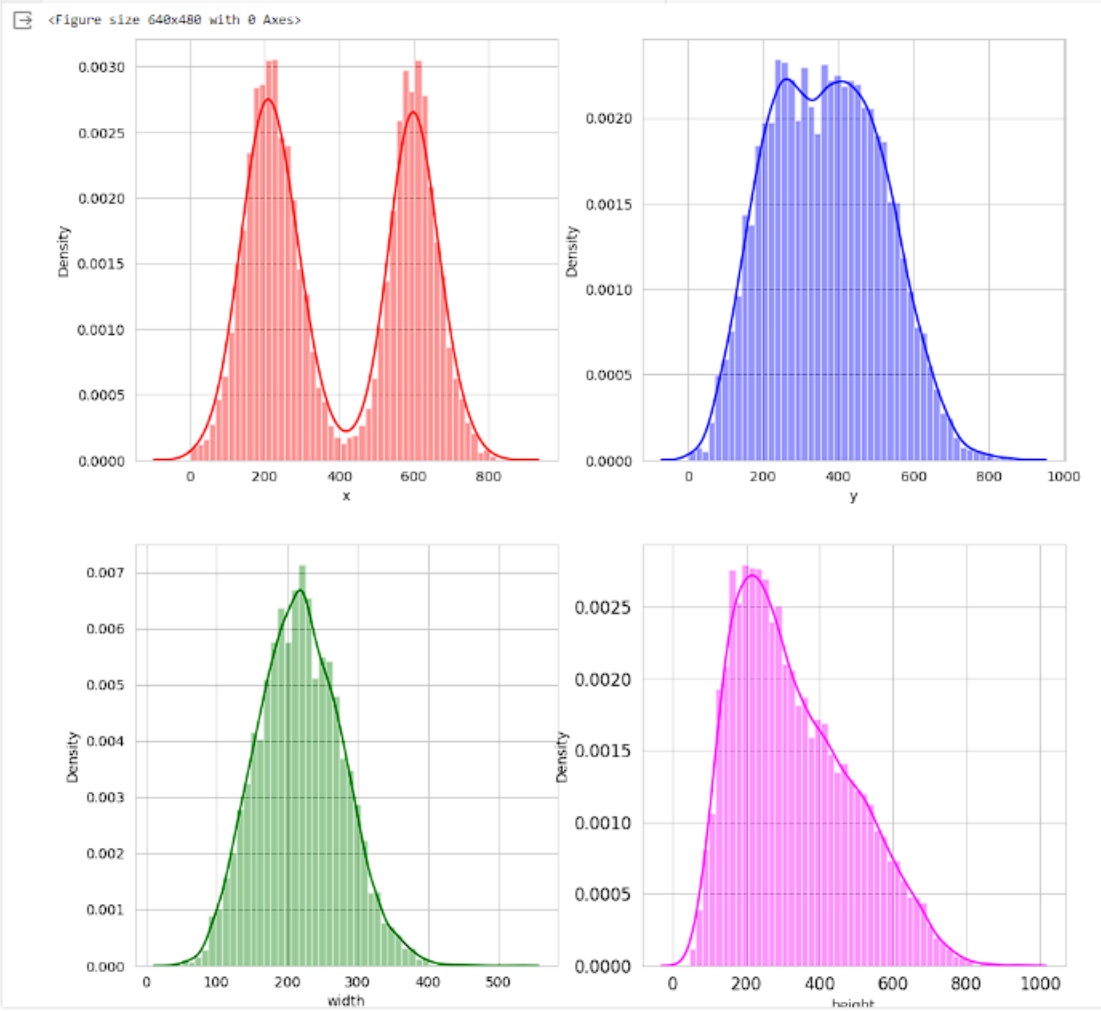
```
[54] sns.boxplot(data=traindf)
```

```
<Axes: >
```



We can see the distribution plot with respect to X, Y, height and width.

```
target1 = traindf[traindf['Target']==1]
sns.set_style('whitegrid')
plt.figure()
fig, ax = plt.subplots(2,2,figsize=(12,12))
sns.distplot(target1['x'],kde=True,bins=50, color="red", ax=ax[0,0]);
sns.distplot(target1['y'],kde=True,bins=50, color="blue", ax=ax[0,1]);
sns.distplot(target1['width'],kde=True,bins=50, color="green", ax=ax[1,0]);
sns.distplot(target1['height'],kde=True,bins=50, color="magenta", ax=ax[1,1]);
locs, labels = plt.xticks()
plt.tick_params(axis='both', which='major', labelsize=12)
plt.show()
```




```
trainImagesPath = Path(r'C:\Users\manve\Downloads\rsna-pneumonia-detection-challenge\stage_2_train_images')
testImagesPath = Path(r'C:\Users\manve\Downloads\rsna-pneumonia-detection-challenge\stage_2_test_images')
```

```
import os;
image_train_path = os.listdir(trainImagesPath)
image_test_path = os.listdir(testImagesPath)

print("Number of images in train set:", len(image_train_path),"\nNumber of images in test set:", len(image_test_path))
```

```
Number of images in train set: 26684
Number of images in test set: 3000
```

Number of images in train set: 26684

Number of images in test set: 3000

We observed Train images length matched with unique patient id's in traindf.

STEP 3: MAP TRAINING AND TESTING IMAGES TO ITS ANNOTATIONS.

We had read DICOM file using the pydicom library based on a specific patient ID from traindf DataFrame.

```
samplePatientID = list(traindf[:3].T.to_dict().values())[0]['patientId']
dcm_path = trainImagesPath/samplePatientID
dcm_path = dcm_path.with_suffix(".dcm")
dcm = pydicom.read_file(dcm_path)
dcm
```

```
Dataset.file_meta -------------------------------
(0002, 0000) File Meta Information Group Length  UL: 202
(0002, 0001) File Meta Information Version       OB: b'\x00\x01'
(0002, 0002) Media Storage SOP Class UID         UI: Secondary Capture Image Storage
(0002, 0003) Media Storage SOP Instance UID      UI: 1.2.276.0.7230010.3.1.4.8323329.28530.1517874485.775526
(0002, 0010) Transfer Syntax UID                 UI: JPEG Baseline (Process 1)
(0002, 0012) Implementation Class UID            UI: 1.2.276.0.7230010.3.0.3.6.0
(0002, 0013) Implementation Version Name         SH: 'OFFIS_DCMTK_360'
-------------------------------------------------
(0008, 0005) Specific Character Set              CS: 'ISO_IR 100'
(0008, 0016) SOP Class UID                       UI: Secondary Capture Image Storage
(0008, 0018) SOP Instance UID                    UI: 1.2.276.0.7230010.3.1.4.8323329.28530.1517874485.775526
(0008, 0020) Study Date                          DA: '19010101'
(0008, 0030) Study Time                          TM: '000000.00'
(0008, 0050) Accession Number                    SH: ''
(0008, 0060) Modality                            CS: 'CR'
(0008, 0064) Conversion Type                     CS: 'WSD'
(0008, 0090) Referring Physician's Name          PN: ''
(0008, 103e) Series Description                  LO: 'view: PA'
(0010, 0010) Patient's Name                      PN: '0004cfab-14fd-4e49-80ba-63a80b6bddd6'
(0010, 0020) Patient ID                          LO: '0004cfab-14fd-4e49-80ba-63a80b6bddd6'
(0010, 0030) Patient's Birth Date                DA: ''
(0010, 0040) Patient's Sex                       CS: 'F'
(0010, 1010) Patient's Age                       AS: '51'
(0018, 0015) Body Part Examined                  CS: 'CHEST'
(0018, 5101) View Position                       CS: 'PA'
(0020, 000d) Study Instance UID                  UI: 1.2.276.0.7230010.3.1.2.8323329.28530.1517874485.775525
(0020, 000e) Series Instance UID                 UI: 1.2.276.0.7230010.3.1.3.8323329.28530.1517874485.775524
(0020, 0010) Study ID                            SH: ''
(0020, 0011) Series Number                       IS: '1'
(0020, 0013) Instance Number                     IS: '1'
(0020, 0020) Patient Orientation                 CS: ''
(0028, 0002) Samples per Pixel                   US: 1
(0028, 0004) Photometric Interpretation          CS: 'MONOCHROME2'
(0028, 0010) Rows                                US: 1024
(0028, 0011) Columns                             US: 1024
(0028, 0030) Pixel Spacing                       DS: [0.14300000000000002, 0.14300000000000002]
(0028, 0100) Bits Allocated                      US: 8
(0028, 0101) Bits Stored                         US: 8
(0028, 0102) High Bit                            US: 7
(0028, 0103) Pixel Representation                US: 0
(0028, 2110) Lossy Image Compression             CS: '01'
(0028, 2114) Lossy Image Compression Method      CS: 'ISO_10918_1'
(7fe0, 0010) Pixel Data                          OB: Array of 142006 elements
```

STEP 4: PREPROCESSING AND VISUALISATION OF DIFFERENT CLASSES

We can observe that we do have available some useful information in the DICOM metadata with predictive value, for example:

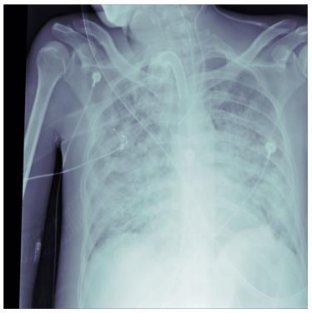
- Patient sex
- Patient age
- Modality
- Body part examined
- View position
- Rows & Columns
- Pixel Spacing

```
def show_dicom_images(data):
    img_data = list(data.T.to_dict().values())
    f, ax = plt.subplots(3,3, figsize=(16,18))
    for i,data_row in enumerate(img_data):
        dcm_path = trainImagesPath/data_row['patientId']
        dcm_path = dcm_path.with_suffix(".dcm")
        data_row_img_data = pydicom.read_file(dcm_path)
        modality = data_row_img_data.Modality
        age = data_row_img_data.PatientAge
        sex = data_row_img_data.PatientSex
        ax[i//3, i%3].imshow(data_row_img_data.pixel_array, cmap=plt.cm.bone)
        ax[i//3, i%3].axis('off')
        ax[i//3, i%3].set_title('ID: {}\nModality: {} Age: {} Sex: {} Target: {}\nClass: {}\nWindow: {}:{}:{}:{}'.format(
                data_row['patientId'],
                modality, age, sex, data_row['Target'], data_row['class'],
                data_row['x'],data_row['y'],data_row['width'],data_row['height']))
    plt.show()
```

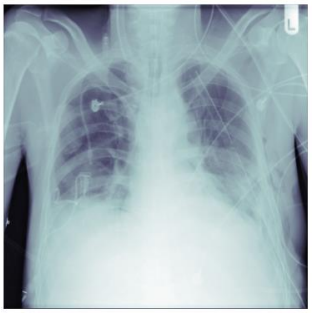
```
show_dicom_images(traindf[traindf['Target']==1].sample(9))
```

To visualize the images with overlay boxes superimposed, our initial step involves parsing the entire dataset where the target is equal to 1.

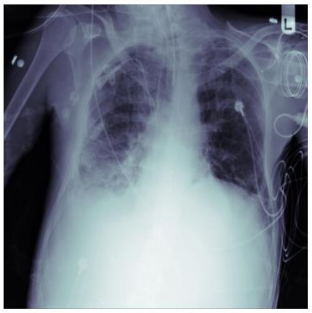
ID: 93870d09-4d81-4d16-b703-791700c8eb57
Modality: CR Age: 40 Sex: M Target: 1
Class: Lung Opacity
Window: 662.0:286.0:328.0:553.0



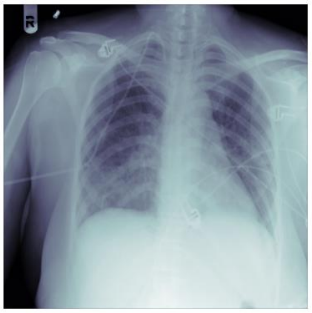
ID: b3a287c5-9fc6-42fb-93c5-310515024948
Modality: CR Age: 52 Sex: M Target: 1
Class: Lung Opacity
Window: 182.0:526.0:210.0:171.0



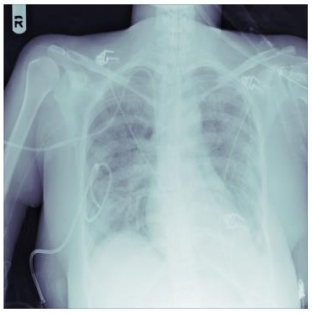
ID: aafda091-8953-40c4-802e-853a206e50d2
Modality: CR Age: 33 Sex: M Target: 1
Class: Lung Opacity
Window: 278.0:431.0:193.0:205.0



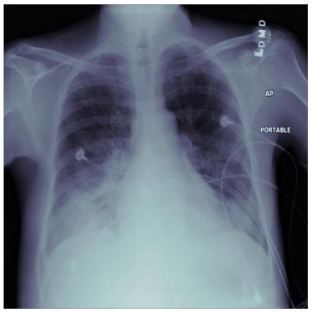
ID: e21bd38d-504c-4256-9168-2f1ccc10ad94
Modality: CR Age: 43 Sex: F Target: 1
Class: Lung Opacity
Window: 279.0:547.0:130.0:146.0



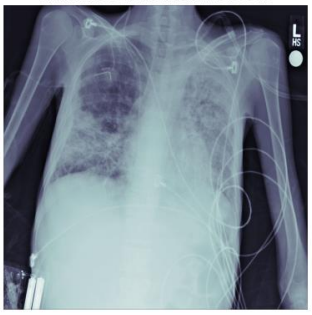
ID: 074f91b5-e915-4b6f-bdf0-af1ee55ebd9b
Modality: CR Age: 40 Sex: F Target: 1
Class: Lung Opacity
Window: 224.0:188.0:329.0:675.0



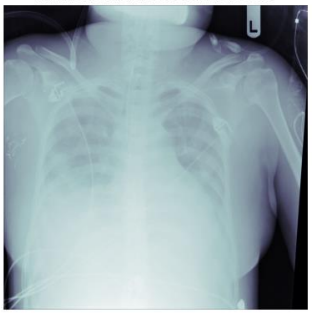
ID: 96f6e753-c609-440a-9c94-fa65d47b38d7
Modality: CR Age: 47 Sex: M Target: 1
Class: Lung Opacity
Window: 182.0:521.0:222.0:315.0



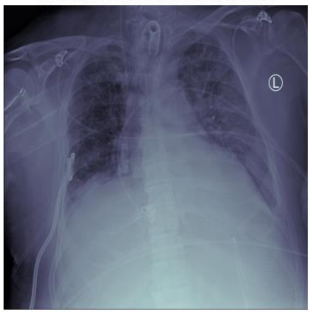
ID: 8fe8e95c-46a1-4ee2-a2d6-2174c6ec50cf
Modality: CR Age: 22 Sex: F Target: 1
Class: Lung Opacity
Window: 590.0:191.0:177.0:306.0



ID: ba07114f-788c-4779-98cd-1190bfb6bdc2
Modality: CR Age: 54 Sex: F Target: 1
Class: Lung Opacity
Window: 143.0:239.0:229.0:367.0



ID: 52510771-8dac-4123-9d82-6a8df9ce0a50
Modality: CR Age: 64 Sex: M Target: 1
Class: Lung Opacity
Window: 590.0:143.0:266.0:356.0



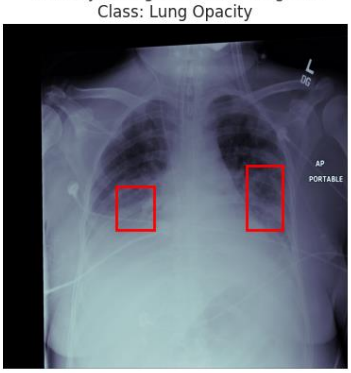
STEP 5: DISPLAY IMAGES WITH BOUNDING BOX.

We can see the bounding boxes for the patient with Pneumonia in the below images output.

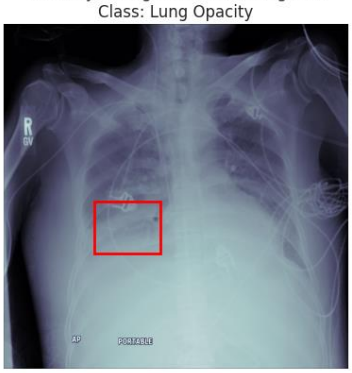
```
def show_dicom_images_with_boxes(data):
    img_data = list(data.T.to_dict().values())
    f, ax = plt.subplots(3,3, figsize=(16,18))
    for i,data_row in enumerate(img_data):
        dcm_path = trainImagesPath/data_row['patientId']
        dcm_path = dcm_path.with_suffix(".dcm")
        data_row_img_data = pydicom.read_file(dcm_path)
        modality = data_row_img_data.Modality
        age = data_row_img_data.PatientAge
        sex = data_row_img_data.PatientSex
        ax[i//3, i%3].imshow(data_row_img_data.pixel_array, cmap=plt.cm.bone)
        ax[i//3, i%3].axis('off')
        ax[i//3, i%3].set_title('ID: {}\nModality: {} Age: {} Sex: {} Target: {}\nClass: {}'.format(
                data_row['patientId'],modality, age, sex, data_row['Target'], data_row['class']))
        rows = traindf[traindf['patientId']==data_row['patientId']]
        box_data = list(rows.T.to_dict().values())
        for j, row in enumerate(box_data):
            ax[i//3, i%3].add_patch(Rectangle(xy=(row['x'], row['y']),
                        width=row['width'],height=row['height'],
                       linewidth=2, edgecolor='r', facecolor='none'))
    plt.show()
```

```
show_dicom_images_with_boxes(traindf[traindf['Target']==1].sample(9))
```

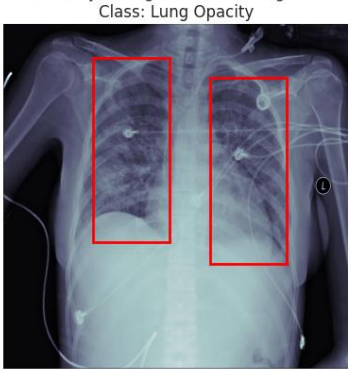
ID: 0ecd37a6-c1bb-40cf-8e48-1020140ae173
Modality: CR Age: 43 Sex: F Target: 1
Class: Lung Opacity



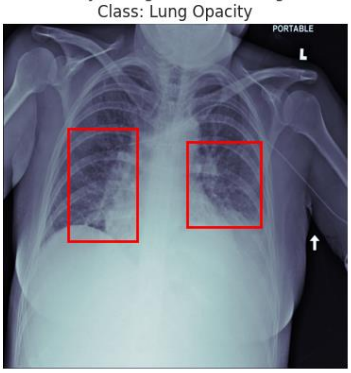
ID: d91b519f-9216-4ff9-9e2a-3356bfe4e391
Modality: CR Age: 59 Sex: M Target: 1
Class: Lung Opacity



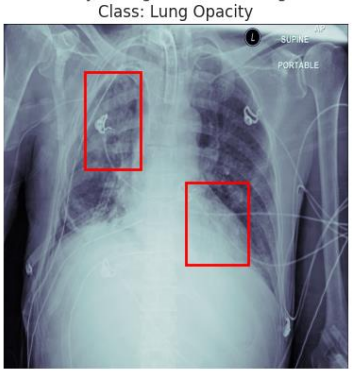
ID: 93351a14-30a7-494c-a02b-7f2d72c724c8
Modality: CR Age: 21 Sex: F Target: 1
Class: Lung Opacity



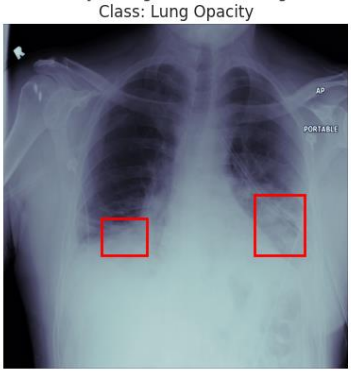
ID: beaee8dc-2b30-41c5-b646-b6f602b965a7
Modality: CR Age: 41 Sex: F Target: 1
Class: Lung Opacity



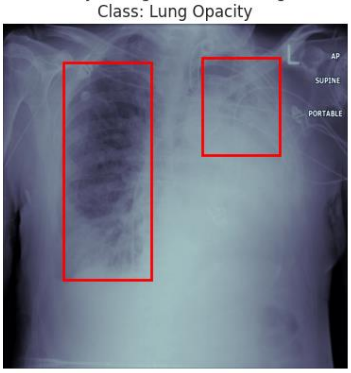
ID: 3bc44999-08aa-4429-ab87-a2958f22b2f8
Modality: CR Age: 51 Sex: M Target: 1
Class: Lung Opacity



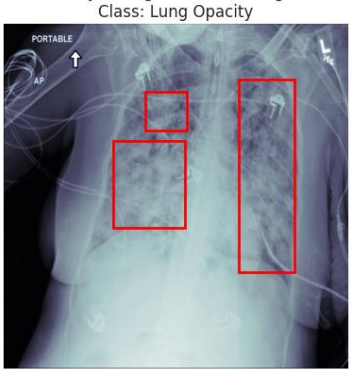
ID: dbc1e726-39b3-4df1-988e-ed7119a4eaa0
Modality: CR Age: 49 Sex: M Target: 1
Class: Lung Opacity



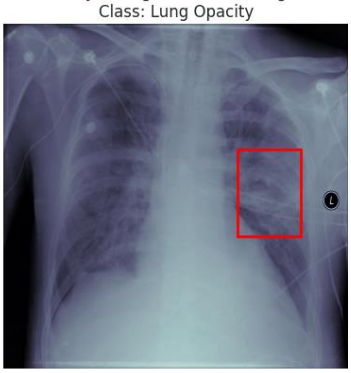
ID: bbd92872-acee-4b27-98b7-07faa4e90919
Modality: CR Age: 67 Sex: M Target: 1
Class: Lung Opacity



ID: f4362a52-b9f5-4d96-a6ba-04284a233eb0
Modality: CR Age: 58 Sex: F Target: 1
Class: Lung Opacity

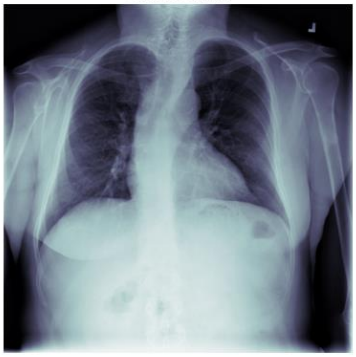


ID: 7a477b7b-f2d2-40ac-a20d-661887491d6b
Modality: CR Age: 39 Sex: M Target: 1
Class: Lung Opacity

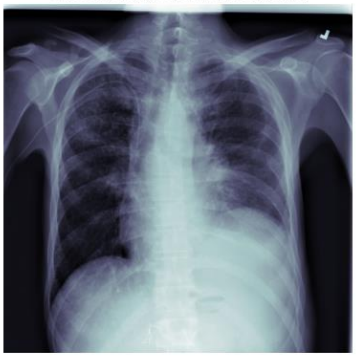


```
show_dicom_images_with_boxes(traindf[traindf['Target']==0].sample(9))
```

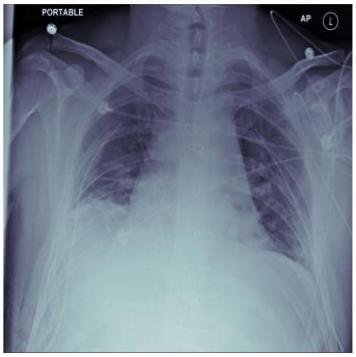
ID: 88a4cd49-f63d-4894-982a-48e7b60bd353
Modality: CR Age: 59 Sex: F Target: 0
Class: Normal



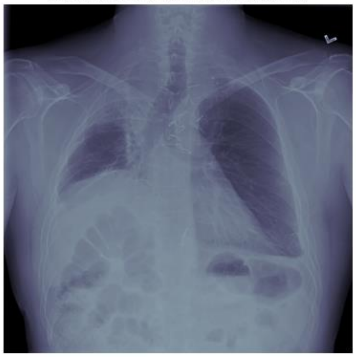
ID: cf6ea318-8cf0-4383-9e13-500d0f01c248
Modality: CR Age: 35 Sex: M Target: 0
Class: No Lung Opacity / Not Normal



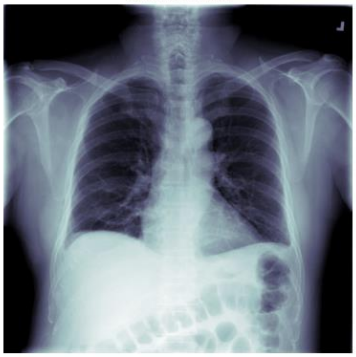
ID: 2d98d4dc-bc2e-4c50-9ca6-5fbffa75d006
Modality: CR Age: 41 Sex: M Target: 0
Class: No Lung Opacity / Not Normal



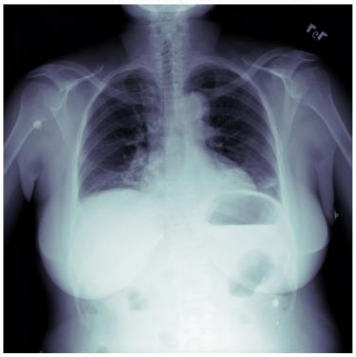
ID: 74425fb9-d1d5-4813-97d8-6b25a55d62a6
Modality: CR Age: 42 Sex: M Target: 0
Class: No Lung Opacity / Not Normal



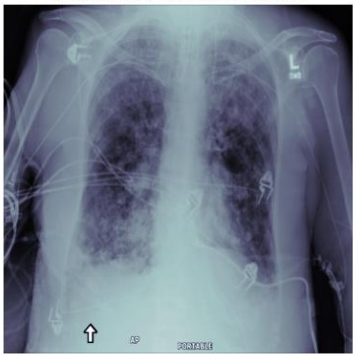
ID: ded4e621-a3ea-4210-8349-06d991c8c056
Modality: CR Age: 57 Sex: M Target: 0
Class: Normal



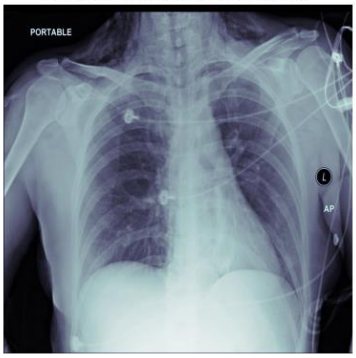
ID: 79deaced-e8d3-4db6-998c-f05798818416
Modality: CR Age: 56 Sex: F Target: 0
Class: No Lung Opacity / Not Normal



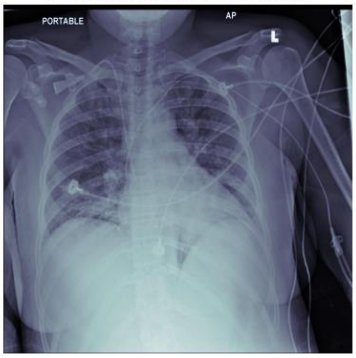
ID: 95e94513-0036-4cda-80da-e5f8d5fb2738
Modality: CR Age: 81 Sex: F Target: 0
Class: No Lung Opacity / Not Normal



ID: 900c9d52-e475-425e-9188-cfd15028c26f
Modality: CR Age: 38 Sex: F Target: 0
Class: No Lung Opacity / Not Normal



ID: 9e9fe9c5-cdff-4b89-85ca-734f96d7fe48
Modality: CR Age: 38 Sex: F Target: 0
Class: No Lung Opacity / Not Normal



We can see the head of traindf data below.

```
traindf.head()
```

	patientId	x	y	width	height	Target	class	Modality	PatientAge	PatientSex	BodyPartExamined	ViewPosition	ConversionType	Rows	Colur
0	0004cfab-14fd-4e49-80ba-63a80b6bddd6	NaN	NaN	NaN	NaN	0	No Lung Opacity / Not Normal	CR	51	F	CHEST	PA	WSD	1024	1
1	00313ee0-9eaa-42f4-b0ab-c148ed3241cd	NaN	NaN	NaN	NaN	0	No Lung Opacity / Not Normal	CR	48	F	CHEST	PA	WSD	1024	1
2	00322d4d-1c29-4943-afc9-b6754be640eb	NaN	NaN	NaN	NaN	0	No Lung Opacity / Not Normal	CR	19	M	CHEST	AP	WSD	1024	1
3	003d8fa0-6bf1-40ed-b54c-ac657f8495c5	NaN	NaN	NaN	NaN	0	Normal	CR	28	M	CHEST	PA	WSD	1024	1
4	00436515-870c-4b36-a041-de91049b9ab4	264.0	152.0	213.0	379.0	1	Lung Opacity	CR	32	F	CHEST	AP	WSD	1024	1

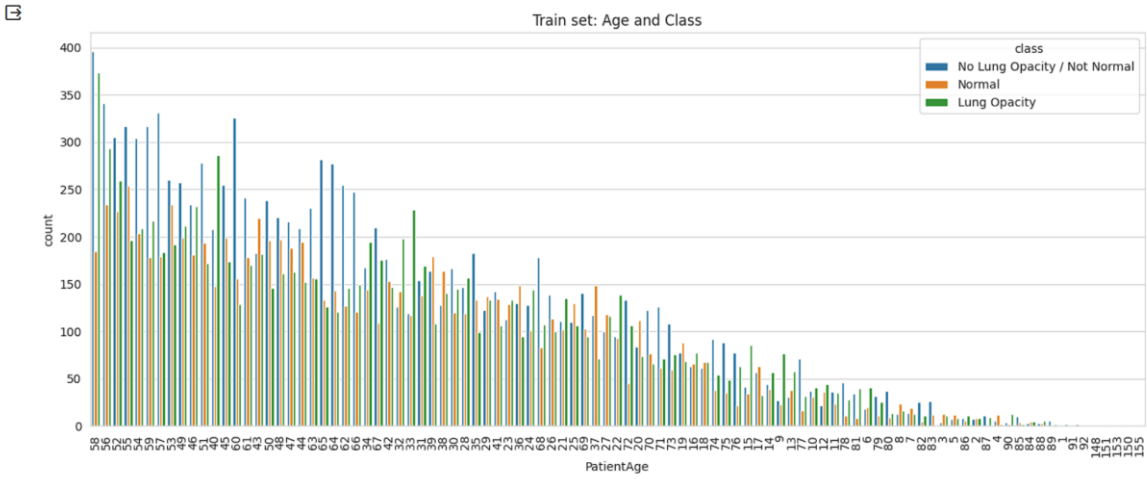
We can see the head of testdf data below.

```
testdf.head()
```

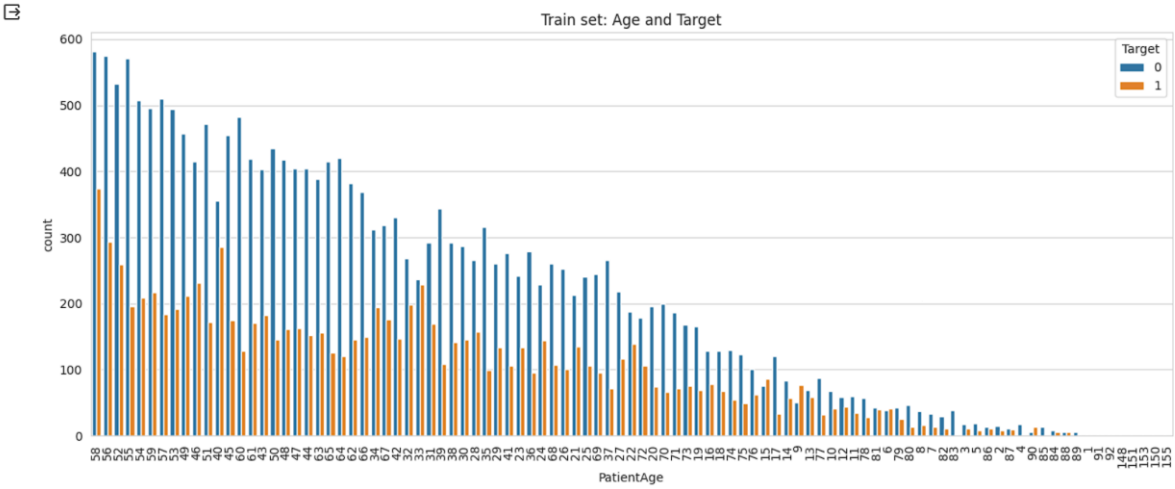
	patientId	Modality	PatientAge	PatientSex	BodyPartExamined	ViewPosition	ConversionType	Rows	Columns	PixelSpacing
0	0000a175-0e68-4ca4-b1af-167204a7e0bc	CR	46	F	CHEST	PA	WSD	1024	1024	0.194
1	0005d3cc-3c3f-40b9-93c3-46231c3eb813	CR	22	F	CHEST	PA	WSD	1024	1024	0.143
2	000686d7-f4fc-448d-97a0-44fa9c5d3aa6	CR	64	M	CHEST	PA	WSD	1024	1024	0.143
3	000e3a7d-c0ca-4349-bb26-5af2d8993c3d	CR	75	F	CHEST	PA	WSD	1024	1024	0.143
4	00100a24-854d-423d-a092-edcf6179e061	CR	66	F	CHEST	AP	WSD	1024	1024	0.139

Below is count plot using Seaborn to visualize the distribution of classes ('Normal' and 'Lung Opacity') for different age groups in the training dataset.

```
fig, (ax) = plt.subplots(nrows=1,figsize=(16,6))
sns.countplot(ax=ax, x = 'PatientAge',hue='class',data=traindf, order = traindf['PatientAge'].value_counts().index)
plt.title("Train set: Age and Class")
plt.xticks(rotation=90)
plt.show()
```

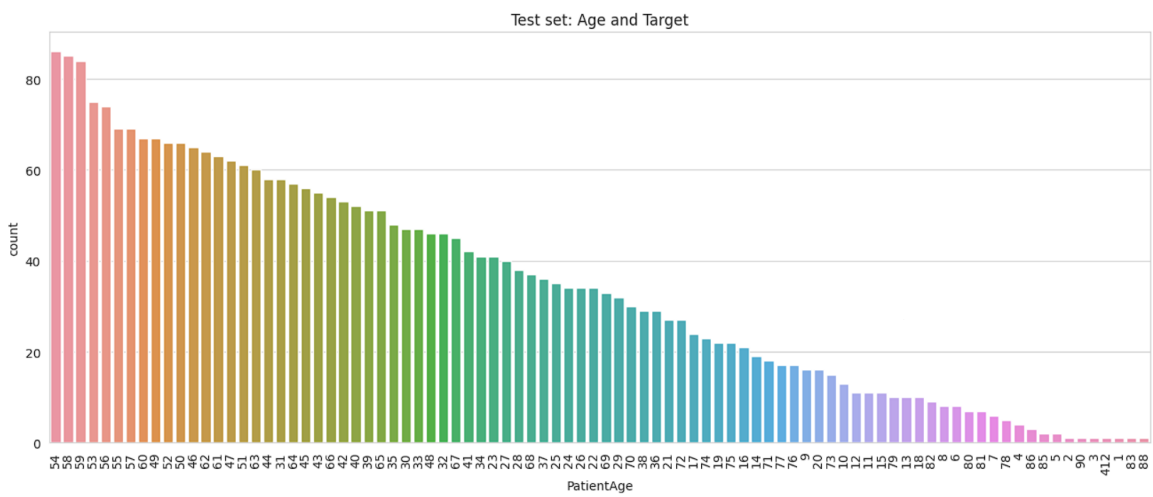



```
fig, (ax) = plt.subplots(nrows=1,figsize=(16,6))
sns.countplot(ax=ax, x = 'PatientAge',hue='Target',data=traindf, order = traindf['PatientAge'].value_counts().index)
plt.title("Train set: Age and Target")
plt.xticks(rotation=90)
plt.show()
```

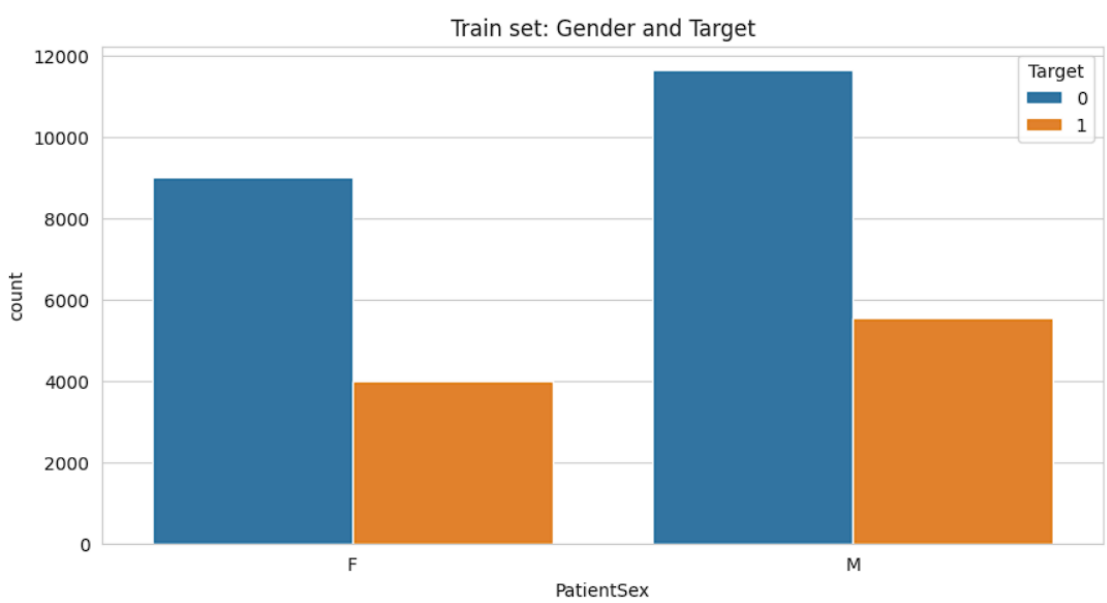


The majority of the data is recorded for individuals aged between 40 to 50. However, an outlier is present with an age of 151. There are limited data points for age groups between 1 to 5 and 80 to 90.

```
fig, (ax) = plt.subplots(nrows=1,figsize=(16,6))
sns.countplot(ax=ax, x = 'PatientAge',data=testdf, order = testdf['PatientAge'].value_counts().index)
plt.title("Test set: Age and Target")
plt.xticks(rotation=90)
plt.show()
```



In test set also similar kind of behavior observed in data among different age groups. Outlier with Age 412



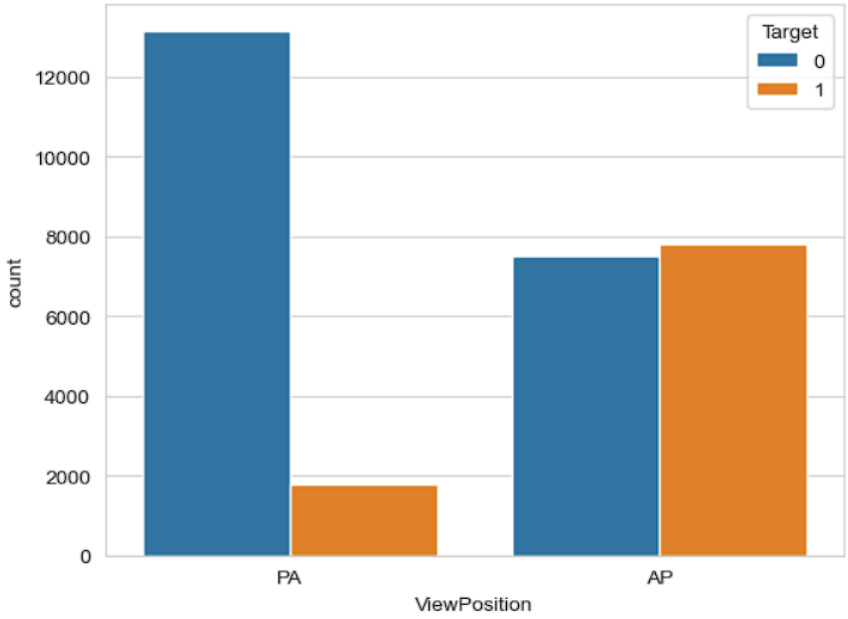
We can see view position below.

```
traindf['ViewPosition'].value_counts()
```

```
AP    15297
PA    14930
Name: ViewPosition, dtype: int64
```

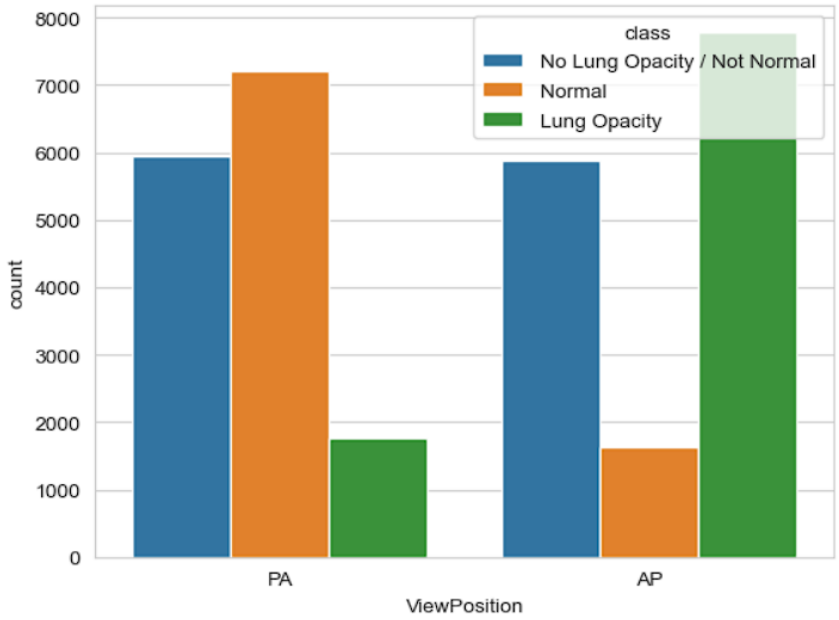
```
sns.countplot(x='ViewPosition',hue='Target',data=traindf)
```

```
<AxesSubplot:xlabel='ViewPosition', ylabel='count'>
```




```
sns.countplot(x='ViewPosition',hue='class',data=traindf)
```

```
<AxesSubplot:xlabel='ViewPosition', ylabel='count'>
```



We can see the pre processing of traindf data.

```
from sklearn import preprocessing

label_encoder = preprocessing.LabelEncoder()

traindf['ViewPosition']= label_encoder.fit_transform(traindf['ViewPosition'])
print(label_encoder.classes_)
traindf['ViewPosition'].unique()
```

```
['AP' 'PA']
```

```
[105] traindf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30227 entries, 0 to 30226
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   patientId     30227 non-null  object
 1   x             9555 non-null   float64
 2   y             9555 non-null   float64
 3   width         9555 non-null   float64
 4   height        9555 non-null   float64
 5   Target        30227 non-null  int64
 6   class         30227 non-null  object
 7   PatientAge    30227 non-null  object
 8   PatientSex    30227 non-null  object
 9   ViewPosition  30227 non-null  int64
 10  PixelSpacing  30227 non-null  object
dtypes: float64(4), int64(2), object(5)
memory usage: 2.8+ MB
```

We can see the correlation of traindf.

```
traindf.corr()
```

	x	y	width	height	Target	ViewPosition
x	1.000000	0.007604	-0.058665	0.008256	NaN	-0.022248
y	0.007604	1.000000	-0.299897	-0.645369	NaN	0.124721
width	-0.058665	-0.299897	1.000000	0.597461	NaN	-0.087427
height	0.008256	-0.645369	0.597461	1.000000	NaN	-0.275490
Target	NaN	NaN	NaN	NaN	1.000000	-0.420189
ViewPosition	-0.022248	0.124721	-0.087427	-0.275490	-0.420189	1.000000

STEP 6: DESIGN, TRAIN AND TEST BASIC CNN MODELS FOR CLASSIFICATION.

We have merged the class info at trainlabels and class info based on patient id.

```
pneumonia = pd.merge(classInfo, trainlabels, on='patientId', how='inner')
```

```
print(pneumonia.shape)
```

```
(37629, 7)
```

```
pneumonia.head()
```

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

```
len(pneumonia['patientId'])
```

```
37629
```

Because the patientId column is duplicated after merging detail_class and bbox_info, we have dropped paitientId from detail_class.

```
pneumonia = pd.concat([classInfo.drop(columns = 'patientId'), trainlabels], axis = 1)
```

```
#Here we go
pneumonia.shape
```

```
(30227, 7)
```

```
pneumonia['patientId'].nunique()
```

```
26684
```

Now we can see the unique records in the above dataset.

We can see the bound inbox parameter in the bbox and dropped x, y, height and width. Below is the final data in the dataset.

```
# Add all Bound In Box parameters in 'bbox'
pneumonia['bbox'] = pneumonia[['x', 'y', 'height', 'width']].apply(lambda x: '-'.join(str(i) for i in x), axis=1)
```

```
pneumonia = pneumonia.drop(columns = ['x', 'y', 'height', 'width'])
```

```
pneumonia.head()
```

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

Below is the meta data of random image from train set and find the final dicom_data.

```
#look at the meta data of random image from train set
random_patient_id = pneumonia['patientId'].sample().values[0]
dicom_data = pydicom.read_file(image_path)


print(dicom_data)
```

```
Dataset.file_meta ------------------------------
(0002, 0000) File Meta Information Group Length  UL: 200
(0002, 0001) File Meta Information Version       OB: b'\x00\x01'
(0002, 0002) Media Storage SOP Class UID         UI: Secondary Capture Image Storage
(0002, 0003) Media Storage SOP Instance UID      UI: 1.2.276.0.7230010.3.1.4.8323329.9955.1517874345.895762
(0002, 0010) Transfer Syntax UID                 UI: JPEG Baseline (Process 1)
(0002, 0012) Implementation Class UID            UI: 1.2.276.0.7230010.3.0.3.6.0
(0002, 0013) Implementation Version Name         SH: 'OFFIS_DCMTK_360'
-------------------------------------------------
(0008, 0005) Specific Character Set              CS: 'ISO_IR 100'
(0008, 0016) SOP Class UID                       UI: Secondary Capture Image Storage
(0008, 0018) SOP Instance UID                    UI: 1.2.276.0.7230010.3.1.4.8323329.9955.1517874345.895762
(0008, 0020) Study Date                          DA: '19010101'
(0008, 0030) Study Time                          TM: '000000.00'
(0008, 0050) Accession Number                    SH: ''
(0008, 0060) Modality                            CS: 'CR'
(0008, 0064) Conversion Type                     CS: 'WSD'
(0008, 0090) Referring Physician's Name          PN: ''
(0008, 103e) Series Description                  LO: 'view: AP'
(0010, 0010) Patient's Name                      PN: 'c1f7889a-9ea9-4acb-b64c-b737c929599a'
(0010, 0020) Patient ID                          LO: 'c1f7889a-9ea9-4acb-b64c-b737c929599a'
(0010, 0030) Patient's Birth Date                DA: ''
(0010, 0040) Patient's Sex                       CS: 'F'
(0010, 1010) Patient's Age                       AS: '72'
(0018, 0015) Body Part Examined                  CS: 'CHEST'
(0018, 5101) View Position                       CS: 'AP'
(0020, 000d) Study Instance UID                  UI: 1.2.276.0.7230010.3.1.2.8323329.9955.1517874345.895761
(0020, 000e) Series Instance UID                 UI: 1.2.276.0.7230010.3.1.3.8323329.9955.1517874345.895760
(0020, 0010) Study ID                            SH: ''
(0020, 0011) Series Number                       IS: '1'
(0020, 0013) Instance Number                     IS: '1'
(0020, 0020) Patient Orientation                 CS: ''
(0028, 0002) Samples per Pixel                   US: 1
(0028, 0004) Photometric Interpretation          CS: 'MONOCHROME2'
(0028, 0010) Rows                                US: 1024
(0028, 0011) Columns                             US: 1024
(0028, 0030) Pixel Spacing                       DS: [0.139, 0.139]
(0028, 0100) Bits Allocated                      US: 8
(0028, 0101) Bits Stored                         US: 8
(0028, 0102) High Bit                            US: 7
(0028, 0103) Pixel Representation                US: 0
(0028, 2110) Lossy Image Compression             CS: '01'
(0028, 2114) Lossy Image Compression Method      CS: 'ISO_10918_1'
(7fe0, 0010) Pixel Data                          OB: Array of 132632 elements
```

```
def read_and_resize_images(pneumonia):
    resized_images = []
    boxes = []
    for i in range(len(pneumonia)):
        patient_id = pneumonia['patientId'][i]
        image_path = pneumonia['image_path'][i]
        target = pneumonia['Target'][i]
        dicom_data = pydicom.read_file(image_path)
        img = dicom_data.pixel_array

        #Resize image to 224x224
        img = cv2.resize(img, (224, 224))
        img = cv2.cvtColor(img, cv2.COLOR_GRAY2RGB)
        resized_images.append(img)
        boxes.append(np.array(target, dtype=np.float32))
    return np.array(resized_images), np.array(boxes)
```

SPLIT THE DATA TO TRAIN AND TEST

We had split the data into train and test as below.

```
X, y = read_and_resize_images(pneumonia[:1000])
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

```
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(800, 224, 224, 3) (800,)
(200, 224, 224, 3) (200,)
```

We had used Python function save_img_from_dcm converts the DICOM file to a JPEG image. It uses the pydicom library to read the DICOM file and cv2 (OpenCV) to save the corresponding JPEG image.

```
In [133]: def save_img_from_dcm(dcm_dir, img_dir, patient_id):
              img_fp = os.path.join(img_dir, "{}.jpg".format(patient_id))
              if os.path.exists(img_fp):
                  return
              dcm_fp = os.path.join(dcm_dir, "{}.dcm".format(patient_id))
              img_1ch = pydicom.read_file(dcm_fp).pixel_array
              img_3ch = np.stack([img_1ch]*3, -1)

              img_fp = os.path.join(img_dir, "{}.jpg".format(patient_id))
              cv2.imwrite(img_fp, img_3ch)
```

```
def get_image(dcm_file):
    ADJUSTED_IMAGE_SIZE = 128
    dcm_data = dcm.read_file(dcm_file)
    img = dcm_data.pixel_array
    img = np.stack((img,) * 3, -1)

    img = np.array(img).astype(np.uint8)
    res = cv2.resize(img,(ADJUSTED_IMAGE_SIZE,ADJUSTED_IMAGE_SIZE), interpolation = cv2.INTER_LINEAR)
    return res
```

```
def read_train(rowData):
    imageList = []
    for index, row in tqdm(rowData.iterrows()):
        patientId = row.patientId
        dcm_file = r'C:\Users\manve\Downloads\rsna-pneumonia-detection-challenge\stage_2_train_images/'+'{}.dcm'.format(patientId
        imageList.append(get_image(dcm_file))

    return np.array(imageList)
```

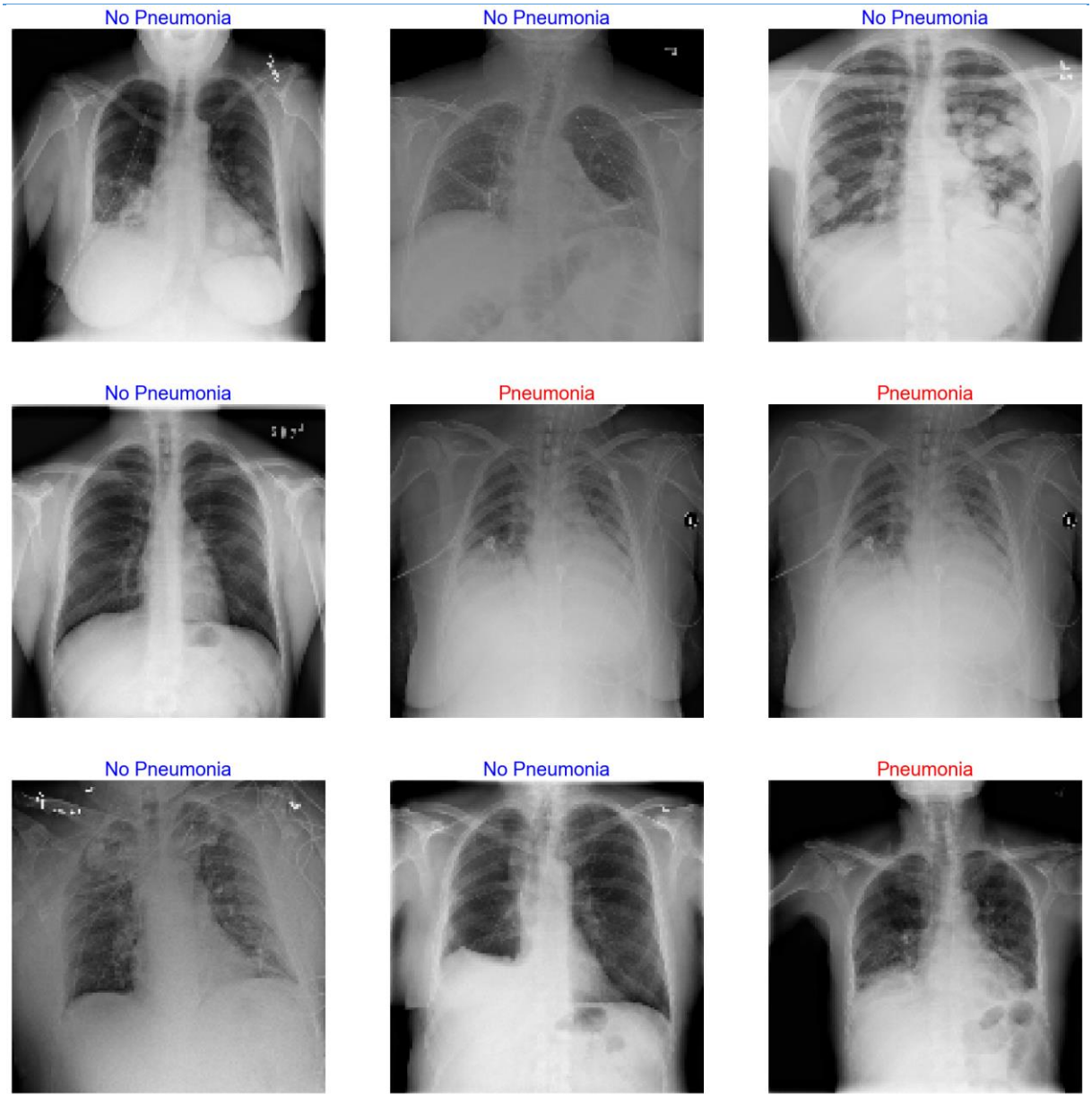
Reading the test data.

```
def read_test(path):
    imageList = []
    for file_name in tqdm(os.listdir(path)):
        dcm_file = dcm_file = os.sep.join([path, file_name])
        imageList.append(get_image(dcm_file))
    return np.array(imageList)

test_images_path = r'C:\Users\manve\Downloads\rsna-pneumonia-detection-challenge\stage_2_test_images'
test_images = read_test(test_images_path)

train_images = read_train(trainlabels)
print(train_images.shape)
```

```
100% 3000/3000 [00:29<00:00, 99.15it/s]

30227/? [07:24<00:00, 55.99it/s]

(30227, 128, 128, 3)
```

We can see differentiate between the Pneumonia and no Pneumonia in the below images.

```
plt.figure(figsize=(25,25)) #
for i, image in enumerate(train_images[:9]):

    plt.subplot(3,3,i+1)
    plt.imshow(image)
    if trainlabels.loc[i]["Target"]:
        plt.title("Pneumonia", color="red", fontsize=25)
    else:
        plt.title("No Pneumonia", color="blue", fontsize=25)
    plt.axis('off')
plt.show()
```




```
y = pd.get_dummies(trainlabels["Target"]).values
random_state = 42

X_train, X_test, y_train, y_test = train_test_split(train_images, y, test_size=0.2, random_state=random_state)
```

BUILDING CNN MODEL

Below code defines three custom metrics for evaluating the performance of a classification model: recall, precision, and F1-score.

```
def recall_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall

def precision_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision

def f1_m(y_true, y_pred):
    precision = precision_m(y_true, y_pred)
    recall = recall_m(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))
```

We had used below code to defines and trains a CNN for image classification using TensorFlow and Keras.

```
ADJUSTED_IMAGE_SIZE = 128
input_shape = (ADJUSTED_IMAGE_SIZE, ADJUSTED_IMAGE_SIZE, 3)
num_classes = y_train.shape[1]

model = Sequential()
model.add(RandomRotation(factor=0.15))
model.add(Rescaling(1./255))
model.add(Conv2D(32, (3, 3), input_shape=input_shape)) # (3, 3) - conv kernel

model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(32, (3, 3)))

model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

model.add(Conv2D(64, (3, 3)))

model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.4))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005),
              metrics=['accuracy'] # ,f1_m
             )
# model.summary()

history = model.fit(X_train,
                    y_train,
                    epochs = 50,
                    validation_data = (X_test,y_test),
                    batch_size = 16,
                      callbacks=[
                          EarlyStopping(monitor = "val_loss", patience = 10, restore_best_weights = True),
                          ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2, mode='min')
                      ]
                     )

fcl_loss, fcl_accuracy = model.evaluate(X_test, y_test, verbose=1) # , fcl_f1
print('Test loss:', fcl_loss)
print('Test accuracy:', fcl_accuracy)
# print('Test F1:', fcl_f1)
```

```
df = pd.DataFrame({"pred": np.argmax(model.predict(X_test), axis=1), "true": np.argmax(y_test, axis=1)})
print(classification_report(df["true"], df["pred"]))
```

```
Epoch 1/50
1512/1512 [==============================] - 155s 102ms/step - loss: 0.
5459 - accuracy: 0.7181 - val_loss: 0.5144 - val_accuracy: 0.7585 - lr:
 5.0000e-04
Epoch 2/50
1512/1512 [==============================] - 152s 101ms/step - loss: 0.
5102 - accuracy: 0.7547 - val_loss: 0.4912 - val_accuracy: 0.7693 - lr:
 5.0000e-04
Epoch 3/50
```

```
1512/1512 [==============================] - 161s 106ms/step - loss: 0.
4946 - accuracy: 0.7675 - val_loss: 0.4911 - val_accuracy: 0.7663 - lr:
 5.0000e-04
Epoch 4/50
1512/1512 [==============================] - 164s 108ms/step - loss: 0.
4879 - accuracy: 0.7723 - val_loss: 0.4823 - val_accuracy: 0.7736 - lr:
 5.0000e-04
Epoch 5/50
1512/1512 [==============================] - 157s 104ms/step - loss: 0.
4839 - accuracy: 0.7766 - val_loss: 0.4860 - val_accuracy: 0.7670 - lr:
 5.0000e-04
Epoch 6/50
1512/1512 [==============================] - 159s 105ms/step - loss: 0.
4823 - accuracy: 0.7736 - val_loss: 0.4715 - val_accuracy: 0.7762 - lr:
 5.0000e-04
Epoch 7/50
1512/1512 [==============================] - 158s 105ms/step - loss: 0.
4753 - accuracy: 0.7793 - val_loss: 0.4649 - val_accuracy: 0.7810 - lr:
 5.0000e-04
Epoch 8/50
1512/1512 [==============================] - 160s 106ms/step - loss: 0.
4752 - accuracy: 0.7799 - val_loss: 0.4647 - val_accuracy: 0.7777 - lr:
 5.0000e-04
Epoch 9/50
1512/1512 [==============================] - 162s 107ms/step - loss: 0.
4671 - accuracy: 0.7835 - val_loss: 0.4623 - val_accuracy: 0.7795 - lr:
 5.0000e-04
Epoch 10/50
1512/1512 [==============================] - 160s 106ms/step - loss: 0.
4665 - accuracy: 0.7841 - val_loss: 0.4570 - val_accuracy: 0.7808 - lr:
 5.0000e-04
Epoch 11/50
1512/1512 [==============================] - 156s 103ms/step - loss: 0.
4664 - accuracy: 0.7847 - val_loss: 0.4548 - val_accuracy: 0.7843 - lr:
 5.0000e-04
Epoch 12/50
1512/1512 [==============================] - 151s 100ms/step - loss: 0.
4654 - accuracy: 0.7837 - val_loss: 0.4530 - val_accuracy: 0.7878 - lr:
 5.0000e-04
Epoch 13/50
1512/1512 [==============================] - 154s 102ms/step - loss: 0.
4642 - accuracy: 0.7837 - val_loss: 0.4600 - val_accuracy: 0.7825 - lr:
 5.0000e-04
Epoch 14/50
1512/1512 [==============================] - 152s 101ms/step - loss: 0.
4606 - accuracy: 0.7840 - val_loss: 0.4530 - val_accuracy: 0.7883 - lr:
 5.0000e-04
Epoch 15/50
1512/1512 [==============================] - 151s 100ms/step - loss: 0.
4525 - accuracy: 0.7910 - val_loss: 0.4490 - val_accuracy: 0.7880 - lr:
 1.0000e-04
Epoch 16/50
1512/1512 [==============================] - 151s 100ms/step - loss: 0.
4515 - accuracy: 0.7922 - val_loss: 0.4490 - val_accuracy: 0.7870 - lr:
 1.0000e-04
Epoch 17/50
```

```
1512/1512 [==============================] - 161s 107ms/step - loss: 0.
4495 - accuracy: 0.7923 - val_loss: 0.4478 - val_accuracy: 0.7873 - lr:
 1.0000e-04
Epoch 18/50
1512/1512 [==============================] - 158s 105ms/step - loss: 0.
4508 - accuracy: 0.7928 - val_loss: 0.4466 - val_accuracy: 0.7883 - lr:
 1.0000e-04
Epoch 19/50
1512/1512 [==============================] - 157s 104ms/step - loss: 0.
4490 - accuracy: 0.7934 - val_loss: 0.4438 - val_accuracy: 0.7899 - lr:
 1.0000e-04
Epoch 20/50
1512/1512 [==============================] - 157s 104ms/step - loss: 0.
4482 - accuracy: 0.7917 - val_loss: 0.4460 - val_accuracy: 0.7890 - lr:
 1.0000e-04
Epoch 21/50
1512/1512 [==============================] - 157s 104ms/step - loss: 0.
4472 - accuracy: 0.7934 - val_loss: 0.4526 - val_accuracy: 0.7890 - lr:
 1.0000e-04
Epoch 22/50
1512/1512 [==============================] - 160s 106ms/step - loss: 0.
4464 - accuracy: 0.7965 - val_loss: 0.4490 - val_accuracy: 0.7906 - lr:
 2.0000e-05
Epoch 23/50
1512/1512 [==============================] - 158s 104ms/step - loss: 0.
4462 - accuracy: 0.7942 - val_loss: 0.4491 - val_accuracy: 0.7906 - lr:
 2.0000e-05
Epoch 24/50
1512/1512 [==============================] - 159s 105ms/step - loss: 0.
4480 - accuracy: 0.7948 - val_loss: 0.4467 - val_accuracy: 0.7904 - lr:
 4.0000e-06
Epoch 25/50
1512/1512 [==============================] - 155s 102ms/step - loss: 0.
4470 - accuracy: 0.7940 - val_loss: 0.4461 - val_accuracy: 0.7899 - lr:
 4.0000e-06
Epoch 26/50
1512/1512 [==============================] - 161s 107ms/step - loss: 0.
4486 - accuracy: 0.7950 - val_loss: 0.4460 - val_accuracy: 0.7899 - lr:
 8.0000e-07
Epoch 27/50
1512/1512 [==============================] - 154s 102ms/step - loss: 0.
4475 - accuracy: 0.7945 - val_loss: 0.4455 - val_accuracy: 0.7903 - lr:
 8.0000e-07
Epoch 28/50
1512/1512 [==============================] - 156s 103ms/step - loss: 0.
4499 - accuracy: 0.7954 - val_loss: 0.4455 - val_accuracy: 0.7904 - lr:
 1.6000e-07
Epoch 29/50
1512/1512 [==============================] - 159s 105ms/step - loss: 0.
4465 - accuracy: 0.7944 - val_loss: 0.4454 - val_accuracy: 0.7904 - lr:
 1.6000e-07
189/189 [==============================] - 7s 35ms/step - loss: 0.4438
- accuracy: 0.7899
```

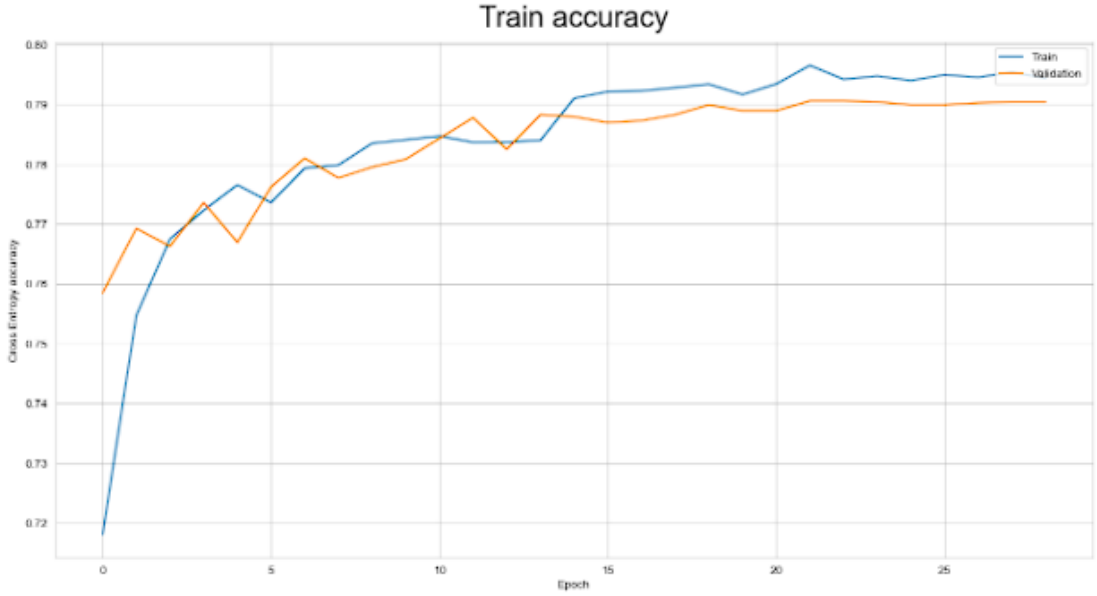
```
Test loss: 0.4438042938709259
Test accuracy: 0.7899437546730042
              precision    recall  f1-score   support

           0       0.82      0.89      0.85      4135
           1       0.71      0.56      0.63      1911

    accuracy                           0.79      6046
   macro avg       0.76      0.73      0.74      6046
weighted avg       0.78      0.79      0.78      6046
```

WE CAN SEE THE CNN MODEL ACCURACY IS 79%.


```
Test loss: 0.4438042938709259
Test accuracy: 0.7899437546730042
              precision    recall  f1-score   support

           0       0.82      0.89      0.85      4135
           1       0.71      0.56      0.63      1911

    accuracy                           0.79      6046
   macro avg       0.76      0.73      0.74      6046
weighted avg       0.78      0.79      0.78      6046
```

The below graph is a visual representation of how the model's accuracy and loss change during training and validation over different epochs. The training curves depict the model's performance on the training set, while the validation curves show how well the model generalizes to new, unseen data.

```
plt.figure(figsize=(16, 8))

plt.plot(history.history['accuracy'], label='Train')
plt.plot(history.history['val_accuracy'], label='Validation')
plt.ylabel('Cross Entropy accuracy')
plt.xlabel('Epoch')
plt.title('Train accuracy', pad=13, fontsize=25)
plt.legend(loc='upper right')
plt.grid(000.1)

plt.show()


plt.figure(figsize=(16, 8))

plt.plot(history.history['loss'], label='Train')
plt.plot(history.history['val_loss'], label='Validation')
plt.ylabel('Cross Entropy Loss')
plt.xlabel('Epoch')
plt.title('Train Loss', pad=13, fontsize=25)
plt.legend(loc='upper right')
plt.grid(000.1)

plt.show()
```



CONVOLUTIONAL NEURAL NETWORK (CNN)

A Convolutional Neural Network (CNN) is a specialized type of artificial neural network designed for processing and analyzing visual data, such as images.

CNNs have proven to be highly effective in various computer vision tasks, including image classification, object detection, and image segmentation. The key innovation of CNNs lies in their ability to automatically and adaptively learn hierarchical representations directly from pixel values.

This is achieved through the use of convolutional layers, pooling layers, and fully connected layers. Convolutional layers apply filters to small, overlapping regions of the input data, enabling the network to capture local patterns and features.

Pooling layers down sample the spatial dimensions, reducing the computational load and focusing on the most critical information.

These layers are typically followed by one or more fully connected layers that combine high-level features for final decision-making.

CNNs excel in feature extraction and pattern recognition, making them the go-to architecture for image-related tasks in fields like computer vision, medical imaging, and autonomous vehicles.

The ability to automatically learn hierarchical representations, coupled with parameter sharing and translation invariance properties, makes CNNs powerful and efficient for visual information processing.

CONCLUSION

We have used the basic CNN model to find the prediction of our Pneumonia detection dataset and achieved the CNN model accuracy of 79%.

Thank You